Towards Learning User-Adaptive State Models in a Conversational Recommender System

Tariq Mahmood
University of Trento
Trento, Italy
tariq@itc.it

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

Abstract

Typical conversational recommender systems support interactive strategies that are hard-coded in advance and followed *rigidly* during a recommendation session. In fact, Reinforcement Learning techniques can be used in order to *autonomously learn an optimal (user-adaptive) strategy*, basically by exploiting some information encoded as features of a state representation. In this regard, it is important to determine the set of *relevant* state features for a given recommendation task. In this paper, we address the issue of feature relevancy, and determine the relevancy of adding four different features to a baseline representation. We show that adding a feature might not always be beneficial, and that the relevancy could be influenced by the user behavior. The results motivate the application of our approach online, in order to acquire the right mixture of online user behavior for addressing the relevancy problem.

1 Background and Motivation

Recommender systems [Resnick and Varian, 1997] are online applications that assist users in their information-seeking tasks by offering personalized product recommendations during an interaction session. In order to support more interactive sessions, *conversational recommender systems* [Ricci *et al.*, 2003; Thompson *et al.*, 2004; Shimazu, 2001] have been recently proposed. These systems adopt the conversation style of real-life dialogues in order to guide users through complex product spaces. Mainly, at each dialogue stage, the system selects one from amongst a set of available system actions, e.g., recommend some product, ask the user for some information etc. The particular action selection is specified by the system's *recommendation strategy*. For instance, suppose that a conversational recommender is queried for hotels that would suit the user preferences. Then, it could employ the following two strategies (among many others): 1) initially query the user in detail for her preferences, in order to extract a small product subset, or 2) recommend a set of products, and exploit the user feedback to refine future recommendations. Conversational systems typically employ a *rigid* strategy, i.e., one which is hard-coded inside the system and followed rigidly during a session [Ricci *et al.*, 2003; Thompson *et al.*, 2004; Shimazu, 2001]. A limitation of this approach is that there could be numerous rigid strategies for a given recommendation task. Furthermore, as system designers rely on intuition and previous experience in order to select a strategy, it is is possible that they shall have to evaluate several strategies before they discover a (almost) "good" one. This is an infeasible process in terms of budget, time, task force etc. These limitations would be tamed if conversational systems could be capable of determining *by themselves* the *best* (optimal) strategy for assisting the users.

In a previous paper [Ricci and Mahmood, 2007], we have tackled these requirements by proposing a new type of recommender system which is able to *autonomously improve an initial strategy in order to learn and adopt an optimal one*. We validate our approach by applying it within the NutKing travel recommender system [Ricci and Mahmood, 2007], and by improving NutKing's (current) rigid strategy in order to learn an optimal one. We learn the strategy through the Reinforcement Learning (RL) paradigm [Sutton and Barto, 1998] in the context of the Markov Decision Process (MDP) framework (we avoid a lengthy explanation due to space constraints). Briefly, our system, at each stage of the session, observes the current *state* (or situation) of the user and the session activity. In one or more states (labeled as *System Decision Points* (SDPs)), multiple actions could be available for execution. The system executes these different actions as the interaction proceeds, and in return, receives a *numerical reward* informing it of the action's acceptability for the user. As the interaction continues, the system learns to *maximize the reward it receives in each possible state*, i.e., it learns to avoid the unacceptable actions and to take the acceptable ones, until it learns the optimal action in each state, i.e., the *optimal policy* (strategy). Also, in the *non-SDP* states, only a single action is available which we consider as the optimal action. The optimal behavior is hence characterized by information in the observed state, which is represented as a combination of feature values. Considering that online user behavior is generally detailed and complex, the selection of the *relevant* state features for a given recommendation task becomes quite crucial.

In this paper, we address several issues related to the feature relevancy problem. We introduce two relevancy criteria, and use them to determine the relevancy for four different feature sets (or *state representations*). We show that adding a feature might not always be relevant for a given task. We also investigate relevancy across different (simulated) user behaviors models, and show that relevancy is influenced by the user behavior. These results motivate the application of our approach in an online context, where we will be able to acquire behavior for a real user population, and to learn an optimal policy for this population.

2 Evaluation Setup

We will now present the evaluation setup, i.e., our proposed sets of state representations and user behavior models, in order to address the feature relevancy problem. For our evaluation, we will use the scenario presented in [Ricci and Mahmood, 2007]. Specifically, we simulate a user (details provided later on) who formulates a logical query to a product catalogue of NutKing, by constraining zero or more product features. If the query retrieves a large result set, i.e., greater than a certain threshold (set to 50 in this evaluation), the system's rigid policy is to suggest a set of features to the user for *tightening* (or reducing the result size) of the current query (Action: *Suggest*). Otherwise, the system executes the query and shows all the results to the user (Action: *Execute*). The system's action set is shown in Table 1. Although NutKing supports other actions as well, we show only these two in Table 1 because, in order to learn an optimal policy, the system must decide to choose only between these two actions, i.e., these are the actions available in the SDP states. In [Ricci and Mahmood, 2007], we have improved NutKing's rigid policy in order to learn an optimal one. Furthermore, our measure of assigning numerical rewards to the system is shown in Table 2: in our scenario, the user's main goal is to add a product to her cart. Hence, we assign a large positive reward (+1) in this situation. Also, until some product is selected, the system is *punished* with a small reward at each stage, in order to convey that the user's goal has (as yet) not been achieved.

Action	Description
Suggest	Suggest Tightening Features
Execute	Execute Query and Show Results

Table 1: System Action Set

$Value$	$Situation$
$+1$	user adds a product to her cart
-0.05	an interaction session stage elapses

Table 2: Reward Function

2.1 State Representations

Given a state representation, the system's job is to learn the optimal policy, i.e., in each SDP, 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 value discretizations) is shown in Table 3, and the set of representations is shown in Table 4. We now present our rationale for selecting these state features. The feature *PUA* depicts the combinations of the web pages (of NutKing) and the possible user actions in these pages [Ricci and Mahmood, 2007] (here, *S-go* represents the situation where the user has logged on to the system). We will describe the remaining combinations later on. Also, *CRS* depicts the result set size when the user executes a query. We select *PUA* and *CRS* in *Baseline* as they provide the minimum information required by the system in order to decide between *Suggest* and *Execute*. *The evaluation task is to add the remaining features, one-by-one, to Baseline, and to determine (using one or more relevancy criteria) whether adding this feature was really beneficial for the system or not*. In this context, we say that the task is to *determine the relevancy of each representation in our representation set R*, where *R*={*Rep1, Rep2, Rep3, Rep4*}.

The feature *ERS* is the system's estimation (based on summary information about data distribution in the catalogue) of the result set size if the best available feature for tightening (the features are ranked according to a feature selection method) is used for tightening by the user. In fact, in [Ricci and Mahmood, 2007] we used representation *Rep1*, but we did not investigate the relevancy of 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 [Ricci *et al.*, 2003]. 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. Besides this, we believe that the number of session stages which have elapsed up to some 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.

2.2 Evaluation Procedure

In order to address the feature relevancy problem, we conducted some off-line simulations, in which we initially defined a simulated user behavior model. This model specifies how the user responds to (all the possible) system actions during a session. Then, we simulated a sequence of user-system sessions or *trials*. Each trial is composed of some stages, and simulates the user as incrementally modifying a query to finally select/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 are specified for product features, i.e., $t = (v_1, ..., v_n)$. We call *t* as the *test item*, and we simulated a user 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 features in *t* have a specified value, i.e., some of these v_i may be *Null*.

In our evaluation, we determined the relevancy for the set

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*)	{*small (0 - 20) , medium (20 - 50) large (50 - 100) , verylarge (100 - INF)*}
Expected Result Size (*ERS*)	{*small (0 - 20) , medium (20 - 50) large (50 - 100) , verylarge (100 - INF)*}
Freq Tight Sugg (*FTSugg*)	{*small (0 - 2) , medium (2 - 4) , large (4 - INF)*}
Number Int Stages (*NStages*)	{*small (0 - 3), medium (3 - 6) , large (6 - INF)*}
User Tighten Resp (*UserTResp*)	{*accept, mixed, reject*}

Table 3: State Features (*INF*=Infinity)

Rep	State Feature Set
Baseline	{*PUA, CRS*}
Rep1	{*PUA, CRS, ERS*}
Rep2	{*PUA, CRS, FTSugg*}
Rep3	{*PUA, CRS, NStages*}
Rep4	{*PUA, CRS, UserTResp*}

Table 4: Different State Representations (*Rep*=Representation)

R with a set of five different user behavior models. Specifically, for *each* user model, we determined the relevancy for each representation in *R*. 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.

2.3 User Models

In this section, we will describe the different user models used in our evaluation. These models differ in how the simulated user responds to the action *Suggest*. Specifically, in our scenario, each user model specifies how the simulated user behaves during the session in the following three cases:

- (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 (*Execute*), whether the user will add the test item to the cart (*PUA=R-add*) or modify the query (*PUA=T-modq*).
- (Case 3) When the system suggests tightening (*Suggest*), 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. (Case 1): At the beginning of each session, we sort the features of the test product according to their frequency of usage (as observed in real-user interactions with NutKing [Ricci *et al.*, 2003]). Then we use this sorting to choose the first feature to constrain in the initial query, and also to incrementally select the next feature to constrain when the user system again observes the state with *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 [Ricci and Mahmood, 2007], 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. Let us define the set *UM* as {*UM=GUM, WillUM, UnWillUM, ModwillUM, AllUM*}. We now detail *UM* as follows:

1. ***GUM***: This model simulates a generic response behavior to *Suggest*: the user accepts tightening if any one of the suggested features has a *non-Null* value in the test item, and this feature is also the next preferred feature of the user (according to the feature usage order of the test item). If the user doesn't accept tightening and the result size is smaller than 50 (*CRS=small,medium*), then the user rejects it and executes the original query. In the remaining cases (i.e., when *CRS=large,very large* and when acceptance cannot be simulated, the user autonomously modifies her query (as in Case 1) (*T-modq*).
2. ***WillUM***: The user accepts tightening if any one of the suggested features has a *non-Null* value in the test item, *even if it is not her next preferred feature*. 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 test item feature was 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* [Ricci *et al.*, 2003]. 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=small,medium*, the user rejects tightening and executes the query. Otherwise, if *CRS=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. We note that *ModwillUM* also simulates a population behavior, but *AllUM* adds more diversity. It is important to determine the relevancy of *R* under a diverse population because, if we apply our approach in an online setting, the optimal policy will be learnt for a user population showing different behaviors) rather than for users exhibiting just a single type of behavior (e.g., always rejecting tightening).

3 Relevancy Criteria

In this section, we will describe our proposed criteria for determining the relevancy of a given representation. In order to apply these criteria, we will initially learn the optimal policy (OP) for all possible "State Representation - User Model" combinations. As we have a total of five representations (Table 4) and five user models (the set *UM*), we learn a total of $5 * 5 = 25$ Optimal Policies (OPs). For all the OPs, we are interested only in the SDP states, i.e., those which have *PUA=QF-execq*, and for all the state combinations listed in the rest of the paper, we assume that *PUA=QF-execq*.

Our first relevancy criterion, *OPEval*, is based on an *evaluation* of the optimal policies, i.e., on determining the

UM	Different State Representations				
	Baseline	*Rep1*	*Rep2*	*Rep3*	*Rep4*
GUM	0.521	**0.5369**	**0.5623**	**0.5213**	**0.5491**
Will	0.5696	**0.5749**	0.5628	0.5576	0.5436
Mod	0.5326	**0.5469**	**0.5625**	**0.5633**	**0.5526**
Unwill	0.5636	0.5629	0.5491	0.5636	0.5539
All	0.5459	0.5399	**0.5626**	0.5437	0.5418

Table 5: Average cumulative rewards under different "User Model - State Representation" combinations (*UM*=user model, *Will = WillUM*, *Mod = ModwillUM*, *Unwill = UnwillUM*, *All = AllUM*)

total reward which the system can accumulate while employing a particular OP. The rationale is that if an OP for a representation $r \in R$ (called OP_r) allows the system to accumulate more reward than the reward for the OP of *Baseline* representation (OP_{base}), then r is a more relevant representation than *Baseline*. Specifically, we select a set of 300 items from the product catalogue. For each item, we simulate an interaction session and calculate the total reward which the system accumulates in that session. At the end, we compute the average cumulative reward (*AvgCumRwd*) for all the 300 sessions. When we evaluate an optimal policy OP of a representation rep and learnt under a user model $um \in UM$, we mean that during the session, the system takes actions according to OP with the user responses being generated through um. For *each user model in* UM, we evaluate OP_{Base} and the four OPs obtained for each representation in R. Then, a representation $r \in R$ is relevant if the *AvgCumRwd* obtained after evaluating OP_r is greater than the *AvgCumRwd* obtained after evaluating OP_{base}. Otherwise, we say that r is irrelevant. In RL, policy evaluation is a robust metric which is commonly used in order to determine the suitability of an OP. Hence, $OPEval$ is a robust metric for determining relevancy.

We propose another relevancy criterion (*OPComp*) which is based on a comparison of the optimal policies. Here, *for each user model in UM*, we compare OP_{base} with each of the four OPs obtained under the set *R*. Then, we say that a representation $r \in R$ is *relevant under some state s* in OP_{base} if OP_r is able to learn a *different* system action than the optimal action for *s* in OP_{base}. Otherwise, we say that *r* is *irrelevant under s*. For instance, suppose that for some user model $um \in UM$, we compare OP_{base} with the optimal policy for *Rep1* (OP_{rep1}), specifically for the state (S_{base}) in OP_{base} where *CRS=small*. Let us define $R1CRS_{small}$ as the set of all states in OP_{rep1} where *CRS=small*. Also, suppose that the optimal action for S_{base} is *Execute*, and that the optimal action for one or more states in $R1CRS_{small}$ is different than *Execute*, i.e., *Suggest*. Such a behavior implies that adding the feature *ERS* is allowing the system to learn different optimal actions and hence change its optimal behavior. Hence, we imply that *R1* is relevant under S_{base}. Also, we say that *R1* is relevant if it is relevant under one or more states in OP_{base}, and that it is irrelevant if it is not relevant under any state in OP_{base}.

4 Results

In this section, we present our results in order to determine the relevancy of the set R under our proposed relevancy criteria, *OPEval* and *OPComp*.

4.1 *OPEval*

For *OPEval*, we first evaluated the 25 optimal policies learnt for the five representations in Table 3, under each user model $um \in UM$. Table 5 shows the *AvgCumRwd* values obtained for each policy evaluation. For each user model, we compare the reward obtained for *Baseline* with the reward obtained for each representation $r \in R$. All the rewards for the set *R* which are greater than their corresponding *Baseline* reward are shown in bold. The results show that all representations in *R* are relevant under *ModwillUM*, showing that for a particular class of real users, adding *ERS*, *FTSugg*, *NStages* and *UserTResp* separately to *Baseline* leads the system to accumulate more reward. Similar results are also obtained under the model *GUM*. However, none of the representations in *R* are relevant under *UnwillUM*, i.e., for unwilling users it is best for the system to always execute the query (see OP_{base} in Table 9) without considering any further information. Also, only *Rep1* is relevant under *WillUM*, i.e., for willing users, it is best to add only *ERS* to *Baseline*. These results prove that a representation which is relevant for a given user group might not be relevant for some other group, i.e., *the relevancy is influenced by the user behavior*. In this context, it is best to determine relevancy under a behavior for a user population, i.e., under *AllUM*. In this case, only *Rep2* is relevant in *R*, i.e., it is best to add only *FTSugg* to *Baseline*. Let us analyze the OP for *Rep2* in Table 10, which allows the system to acquire more reward. This policy shows that our user population is willing to follow tightening suggestions only when a large number of products are retrieved, and then also, only for the first few times that it is suggested. If the system suggests tightening just more than 3 times, the population ignores it even if a lot of products are retrieved.

4.2 *OPComp*

Let us now determine relevancy under *OPComp*. Tables 6-10 show the optimal policies for the representations in Table 3, under the user model *GUM*, *WillUM*, *ModwillUM*, *UnwillUM* and *AllUM* respectively. In each table, the different state combinations are shown in an abbreviated form in brackets, for instance, the state (s, m) of *Rep1* is the state $\{$*PUA=QF-execq, CRS=small, ERS=medium*$\}$. For each table, all the optimal policy actions of a representation $r \in R$, which are different from their *Baseline* counterparts are shown in bold. In order to facilitate understanding, we have shown the different actions under each value of *CRS* (*small, medium, large, verylarge*) in separate columns. Let us briefly analyze the behavior of the learnt policies.

An interesting result is that the OP_{Base} learnt for models *WillUM*, *ModwillUM* and *AllUM* is the rigid policy (RP) of NutKing which we have improved in [Ricci and Mahmood, 2007], i.e., execute the query for small result sizes (*CRS*=$\{$*small, medium*$\}$) and suggest tightening for larger ones (*CRS*=$\{$*large, verylarge*$\}$). This shows that, under *Baseline*, RP is optimal for these classes of users. It would be also interesting to analyze how adding a feature to *Baseline* affects RP. The results show that adding features doesn't significantly change RP for states with *CRS*=$\{$*small, medium, verylarge*$\}$. In fact, the total number of *different* optimal actions learnt (i.e., the action *Suggest* for *CRS*=$\{$*small, medium*$\}$ and *Execute* for *CRS=verylarge*) for these states are only 29 out of a total of 102 states, i.e., a difference of only $29/102 = 28.4\%$. On the contrary, a significant change is observed for states

Rep	Optimal Policies for states with *PUA=QF-execq*											
Baseline OP_{base}	s exec			m exec			l exec			vl sugg			
Rep1	(s,s) exec			(m,s) exec	(m,m) exec		(l,s) **sugg**	(l,m) exec	(l,l) exec	(vl,s) sugg	(vl,m) **exec**	(vl,l) **exec**	(vl,vl) sugg
Rep2	(s,s) exec	(s,m) exec	(s,l) exec	(m,s) exec	(m,m) exec	(m,l) exec	(l,s) **sugg**	(l,m) **sugg**	(l,l) exec	(vl,s) sugg	(vl,m) sugg	(vl,l) **exec**	
Rep3	(s,s) exec	(s,m) exec	(s,l) exec	(m,s) **sugg**	(m,m) exec	(m,l) exec	(l,s) exec	(l,m) exec	(l,l) exec	(vl,s) **exec**	(vl,m) sugg	(vl,l) sugg	
Rep4	(s,ac) exec	(s,mx) exec	(s,rj) exec	(m,ac) **sugg**	(m,mx) exec	(m,rj) exec	(l,ac) exec	(l,mx) exec	(l,rj) exec	(vl,ac) sugg	(vl,mx) **exec**	(vl,rj) sugg	

Table 6: Optimal Policies learnt under *GUM* (*s=small, m=medium, l=large, vl=very large, ac=accept, mx=mixed, rj=reject*)

Rep	Optimal Policies for states with *PUA=QF-execq*											
Baseline OP_{base}	s exec			m exec			l sugg			vl sugg			
Rep1	(s,s) exec			(m,s) **sugg**	(m,m) exec		(l,s) sugg	(l,m) sugg	(l,l) **exec**	(vl,s) sugg	(vl,m) sugg	(vl,l) sugg	(vl,vl) sugg
Rep2	(s,s) exec	(s,m) exec	(s,l) exec	(m,s) exec	(m,m) exec	(m,l) exec	(l,s) sugg	(l,m) **exec**	(l,l) **exec**	(vl,s) sugg	(vl,m) sugg	(vl,l) **exec**	
Rep3	(s,s) exec	(s,m) exec	(s,l) exec	(m,s) exec	(m,m) exec	(m,l) exec	(l,s) **exec**	(l,m) sugg	(l,l) **exec**	(vl,s) sugg	(vl,m) sugg	(vl,l) sugg	
Rep4	(s,ac) **sugg**	(s,mx) exec	(s,rj) exec	(m,ac) **sugg**	(m,mx) exec	(m,rj) exec	(l,ac) sugg	(l,mx) **exec**	(l,rj) **exec**	(vl,ac) sugg	(vl,mx) **exec**	(vl,rj) **exec**	

Table 7: Optimal Policies learnt under *WillUM* (*s=small, m=medium, l=large, vl=very large, ac=accept, mx=mixed, rj=reject*)

Rep	Optimal Policies for states with *PUA=QF-execq*											
Baseline OP_{base}	s exec			m exec			l sugg			vl sugg			
Rep1	(s,s) exec			(m,s) exec	(m,m) exec		(l,s) sugg	(l,m) **exec**	(l,l) **exec**	(vl,s) sugg	(vl,m) sugg	(vl,l) **exec**	(vl,vl) sugg
Rep2	(s,s) exec	(s,m) exec	(s,l) exec	(m,s) exec	(m,m) exec	(m,l) exec	(l,s) sugg	(l,m) **exec**	(l,l) **exec**	(vl,s) sugg	(vl,m) sugg	(vl,l) **exec**	
Rep3	(s,s) exec	(s,m) exec	(s,l) exec	(m,s) exec	(m,m) exec	(m,l) exec	(l,s) **exec**	(l,m) sugg	(l,l) **exec**	(vl,s) sugg	(vl,m) **exec**	(vl,l) **exec**	
Rep4	(s,ac) **sugg**	(s,mx) exec	(s,rj) exec	(m,ac) **sugg**	(m,mx) exec	(m,rj) exec	(l,ac) sugg	(l,mx) **exec**	(l,rj) **exec**	(vl,ac) sugg	(vl,mx) **exec**	(vl,rj) **exec**	

Table 8: Optimal Policies learnt under *ModwillUM* (*s=small, m=medium, l=large, vl=very large, ac=accept, mx=mixed, rj=reject*)

Rep	Optimal Policies for states with *PUA=QF-execq*											
Baseline OP_{base}	s exec			m exec			l exec			vl exec			
Rep1	(s,s) exec			(m,s) exec	(m,m) exec		(l,s) exec	(l,m) exec	(l,l) exec	(vl,s) **sugg**	(vl,m) exec	(vl,l) exec	(vl,vl) exec
Rep2	(s,s) exec	(s,m) exec	(s,l) exec	(m,s) exec	(m,m) exec	(m,l) exec	(l,s) **sugg**	(l,m) **sugg**	(l,l) **sugg**	(vl,s) **sugg**	(vl,m) **sugg**	(vl,l) exec	
Rep3	(s,s) exec	(s,m) exec	(s,l) exec	(m,s) exec	(m,m) exec	(m,l) exec	(l,s) exec	(l,m) exec	(l,l) exec	(vl,s) exec	(vl,m) exec	(vl,l) exec	
Rep4	(s,ac) exec	(s,mx) exec	(s,rj) exec	(m,ac) **sugg**	(m,mx) exec	(m,rj) exec	(l,ac) **sugg**	(l,mx) **sugg**	(l,rj) exec	(vl,ac) **sugg**	(vl,mx) **sugg**	(vl,rj) exec	

Table 9: Optimal Policies learnt under *UnwillUM* (*s=small, m=medium, l=large, vl=very large, ac=accept, mx=mixed, rj=reject*)

Rep	Optimal Policies for states with *PUA=QF-execq*												
Baseline OP_{base}	s exec			m exec			l sugg			vl sugg			
Rep1	(s,s) exec			(m,s) exec	(m,m) exec		(l,s) sugg	(l,m) sugg	(l,l) **exec**	(vl,s) sugg	(vl,m) **exec**	(vl,l) **exec**	(vl,vl) sugg
Rep2	(s,s) exec	(s,m) exec	(s,l) exec	(m,s) exec	(m,m) exec	(m,l) exec	(l,s) sugg	(l,m) sugg	(l,l) **exec**	(vl,s) sugg	(vl,m) sugg	(vl,l) **exec**	
Rep3	(s,s) exec	(s,m) exec	(s,l) exec	(m,s) exec	(m,m) exec	(m,l) exec	(l,s) **exec**	(l,m) sugg	(l,l) **exec**	(vl,s) sugg	(vl,m) sugg	(vl,l) sugg	
Rep4	(s,ac) exec	(s,mx) exec	(s,rj) exec	(m,ac) **sugg**	(m,mx) exec	(m,rj) exec	(l,ac) sugg	(l,mx) **exec**	(l,rj) **exec**	(vl,ac) sugg	(vl,mx) sugg	(vl,rj) **exec**	

Table 10: Optimal Policies learnt under *AllUM* (*s=small, m=medium, l=large, vl=very large, ac=accept, mx=mixed, rj=reject*)

with *CRS=large*, which yield a difference of 58.3%. Furthermore, the results for *GUM* and *UnwillUM* show a similar behavior for states with *CRS*=$\{$*small, medium*$\}$ (the difference being only 9.1%). Also, most differences are obtained for states with *CRS=verylarge* followed by states with *CRS=large*, with the respective percentages being 38.4% and 33.3% respectively (which are less than 58.3%. Generally speaking, these results imply that, if more features are added to *Baseline*, 1) it is best to execute the query for smaller result sizes, 2) the user population is not too willing to accept tightening even for large result sizes, and 3) it is best to suggest tightening only for very large result sizes.

We now determine the relevancy of our representation set *R* under each user model $um \in UM$. The results show that each representation in R is relevant under *GUM*, *WillUM*, and *ModwillUM*. However, *Rep1*, *Rep2* and *Rep4* are relevant under *UnwillUM* but *Rep3* is irrelevant. Under *AllUM*, each representation in *R* is relevant, i.e., for our user population, it is better to add all our proposed features to *Baseline*. These results again prove that the relevancy is influenced by the user behavior. We also note that the results for *GUM* and *ModwillUM* are similar to those for *OPEval*, but the results for the other models are different. Considering that *OPEval* is a more robust criterion for relevancy, this result shows that even if adding a new feature to some baseline representation allows the system to learn different optimal actions, it does not guarantee that the new OP is indeed a suitable policy.

5 Related Work and Conclusions

In this paper, we have addressed the problem of determining a relevant state representation, and we have shown that adding a new feature is not always beneficial for the system, and that the relevancy is influenced by the user behavior. Also, we have justified considering real-user behavior in order to learn the optimal strategy, which would allow us to determine relevancy of a larger representation set R, and also to depict the users' willingness level through a much larger system action set (rather than only on tightening/executing the query). To this end, we have applied our recommendation methodology within an online travel recommender system and are now running experiments with real users in the context of the etPackaging project funded by Austrian Network for E-Tourism (ANET). We have proposed a set of 10 state features for this system (along with a more detailed system action set), and we intend to develop a technique for performing a sequential feature selection on this set, in order to determine the relevant features. To the best of our knowledge, our work is the first attempt in addressing the relevancy problem in the domain of recommender systems. The relevancy problem has also been addressed in the domain of dialogue systems: [Tetreault and Litman, 2006] exploit the *OPComp* criteria to prove the relevancy of five state representations under a corpus of real-user sessions. However, their results need further validation because we have shown that simply learning different actions doesn't guarantee that the new policy is optimal for the users. Also, [Frampton and Lemon, 2006] adopt a similar criteria to *OPEval* in order to prove the relevancy of adding two dialogue features to a baseline representation.

References

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

[Resnick and Varian, 1997] Paul Resnick and Hal R. Varian. Recommender systems. *Communications of the ACM*, 40(3):56–58, 1997.

[Ricci and Mahmood, 2007] Francesco Ricci and Tariq Mahmood. Learning and adaptivity in interactive recommender systems. In *Proceedings of the ICEC'07 Conference*, August 2007.

[Ricci *et al.*, 2003] Francesco Ricci, Adriano Venturini, Dario Cavada, Nader Mirzadeh, Dennis Blaas, and Marisa Nones. Product recommendation with interactive query management and twofold similarity. In *ICCBR 2003, the 5th International Conference on Case-Based Reasoning*, 2003.

[Shimazu, 2001] Hideo 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.

[Sutton and Barto, 1998] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* MIT Press, 1998.

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

[Thompson *et al.*, 2004] Cynthia A. Thompson, Mehmet H. Goker, and Pat Langley. A personalized system for conversational recommendations. *Artificial Intelligence Research*, 21:393–428, 2004.