Improving Recommender Systems with Adaptive Conversational Strategies

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

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

ABSTRACT

Conversational recommender systems (CRSs) assist online users in their information-seeking and decision making tasks by supporting an interactive process. Although these processes could be rather diverse, CRSs typically follow a fixed strategy, e.g., based on critiquing or on iterative query reformulation. In a previous paper, we proposed a novel recommendation model that allows conversational systems to autonomously improve a fixed strategy and eventually learn a better one using reinforcement learning techniques. This strategy is optimal for the given model of the interaction and it is adapted to the users' behaviors. In this paper we validate our approach in an online CRS by means of a user study involving several hundreds of testers. We show that the optimal strategy is different from the fixed one, and supports more effective and efficient interaction sessions.

Categories and Subject Descriptors

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

General Terms

Design, Algorithms, Experimentation

Keywords

Conversational Recommender Systems, Markov Decision Process, Reinforcement Learning, Adaptivity, User Study

1. INTRODUCTION

Recommender systems are intelligent applications which assist users in their information-seeking tasks, by suggesting those items (products, services, information) that best suit their needs and preferences [17]. They have been exploited for recommending travel products, books, CDs, financial services, and in many other applications [1, 6, 8]. During the interaction, recommender systems acquire the users' preferences, and use them to build some type of a user model. Then, the system predicts the set of products that best "matches" the user model, and recommends them to the user. Users specify their preferences by providing during the interaction various types of information or feedbacks, which are used to update the user model, and hence inform future recommendations.

Many recommender systems support a simple human/computer interaction: when the user enters some search query, it is assumed that all her preferences have been specified such that the system can output its recommendations [1]. These type of systems require all the preferences to be specified upfront, at the beginning of the interaction. This is a stringent requirement, because users might not have a clear idea of their preferences at that point. Moreover, in these simple approaches, if the result does not satisfy the user, then she can only start a new request by modifying, if possible, her user model. Such a behavior is not interactive, and the system does not provide an effective assistance in the information search process.

In order to address these limitations, *conversational recommender systems* [24, 3, 16, 4] have been proposed. These systems mimic the more natural human-to-human approach where preferences are elicited over the length of the interaction, rather than upfront. These systems support conversations where the user and the system interact with each other over a sequence of interaction stages. At each stage the user makes a request, such as querying the product catalogue, and in response the system selects and executes one action in order to assist the user in acquiring her goal. For instance, when the user queries the catalogue the system could execute the query and show the retrieved products, or it could ask the user to input some more preferences. The particular action that the system executes at each stage is dictated by its *recommendation strategy*. We define a "strategy" as the abstract plan of action that the system employs during the interaction session. For instance, imagine that a conversational travel recommender system is required to suggest a few hotels that could suit the users' preferences. To this end, the system may employ the following two strategies (amongst many others): *Strategy 1*, which dictates querying the user for her preferences, and acquiring enough information from her, in order to select and recommend a candidate set of hotels, or; *Strategy 2,* which dictates initially making some suggestions, and then acquiring user preferences as ratings, in order to personalize the future recommendations.

A major limitation of conversational systems is that their strategy is typically hard-coded in advance; at each stage, the system always executes a fixed, pre-determined action, notwithstanding the fact that other actions could also be available for execution. In order to understand this limitation, suppose that, in the above scenario, Strategy 2 is adopted by the system. This would not be suitable for users who are unwilling to provide ratings, and therefore might lead to failure situations (e.g., the user quits the session). In this situation, the system should be able to dynamically decide by itself, when to stop collecting ratings and to start taking some alternative action, for instance, querying the user for her preferences (as in Strategy 1). In this case, we say that the system should be able to improve Strategy 2 in order to switch to Strategy 1.

In fact, some conversational systems could support quite complex interactions with the users [13, 4]. For instance, in [4], the authors employ a complex strategy in order to allow users to configure E-commerce products through a dialogue. However, even in this case, the response of the system to each possible user action/request is hard-coded in advance and the system cannot learn from its failures or successes.

In a previous paper we have tackled these requirements by proposing a new type of conversational recommender system that interacting with users is able to autonomously improve an initial default strategy in order to eventually learn and employ a better one [10]. This new strategy is optimal for a particular Markov Decision Process, and is learnt through Reinforcement Learning (RL) techniques [20]. In RL the system basically executes different system actions through trial-and-error, and then observes the response of the users to these actions, in order to maximize a numerical cumulative reward function that models how much benefit the user is supposed to get from each session. In the context of RL we use the term *policy* rather than "strategy", where a policy simply specifies how the strategy is implemented in terms of the system actions.

In [10] we validated our approach in off-line experiments through simulations applied to a portion of the interaction flow of the NutKing travel recommender system. Moreover, in [11] we performed some more simulations in order to determine the information, i.e., the state variables, that the system needs to acquire in order to learn the optimal policy. We showed that the relevancy of a state variable depends on the user behavior as well as on the numerical reward function. In another short paper, we provided an initial brief account of the application of our approach to an online context [12], within the Travel Planner (TP) tool that we prototyped for the Austrian tourism web portal (Austria.info), in the context of the $_{et}$Packaging project funded by Austrian Network for E-Tourism (ANET).

This paper extends [12] providing a full description of the experimental evaluation strategy and its results. In particular, we discuss the changes that we made to TP to adopt the proposed technology. We illustrate the interaction points where the new system is designed to be adaptive, we illustrate the evaluation strategy and the outcome of a user study involving several hundreds of real users. Our results show that the optimal policy, which is adapted to the served users, deviates from the initial default policy. Moreover, the optimal policy is able to assist the users in acquiring their goals more *effectively*, i.e., it increases the task completion rate, and more *efficiently*, i.e., in a lesser number of interaction stages and in a lesser amount of time. Consequently, users spend lesser effort during the interaction: composing a smaller number of queries and viewing fewer result pages.

In this paper, we will initially give an example illustrating the the application of our recommendation approach to the mentioned Travel Planning tool. We will illustrate two system decision points, which we have incorporated within the interaction flow, and we will show how the system can learn a more sophisticated action selection policy in these situations. Then, we will describe the policy learning problem and the proposed procedure for learning the optimal policy. Further on, we will present our proposed evaluation methodology and its results: the characteristics of the learnt optimal policy and the comparison of several system performance variables. Then, we will discuss some related work and finally present our conclusions and future work.

2. DECISION POINTS EXAMPLE

In this section, we will illustrate, by referring to the TP tool, the proposed adaptive recommendation approach. Our method is based on the introduction of one or more decision points for the system, within the default interaction flow of the TP. We label these points as the *system decision points* (SDP). In each SDP a set of system actions is available for execution, and the system's job is to learn to choose the best action from amongst this set depending on the value of some observed state variables (more on this later on).

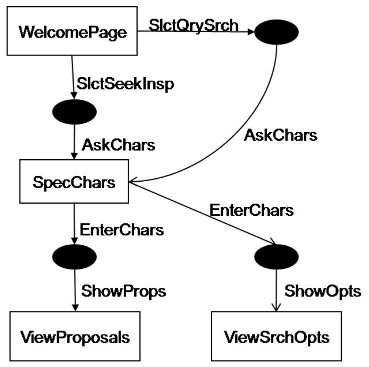
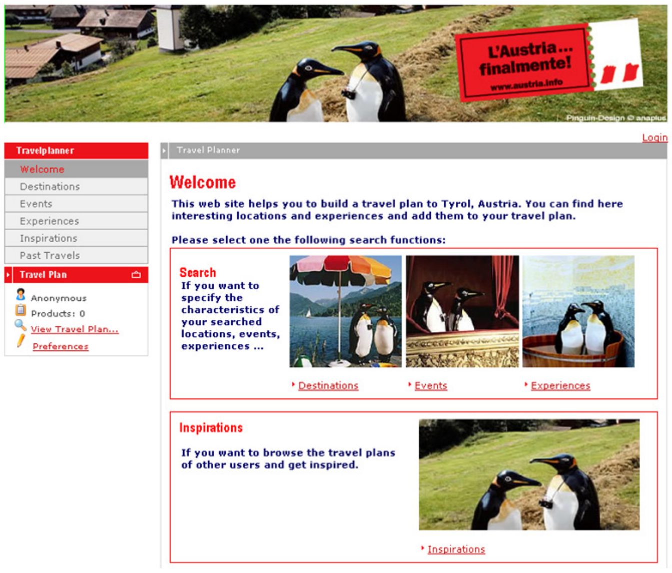


Figure 1: Default navigation flow from the Welcome Page



Welcome

This web site helps you to build a travel plan to Tyrol, Austria. You can find here interesting locations and experiences and add them to your travel plan.

Please select one the following search functions:

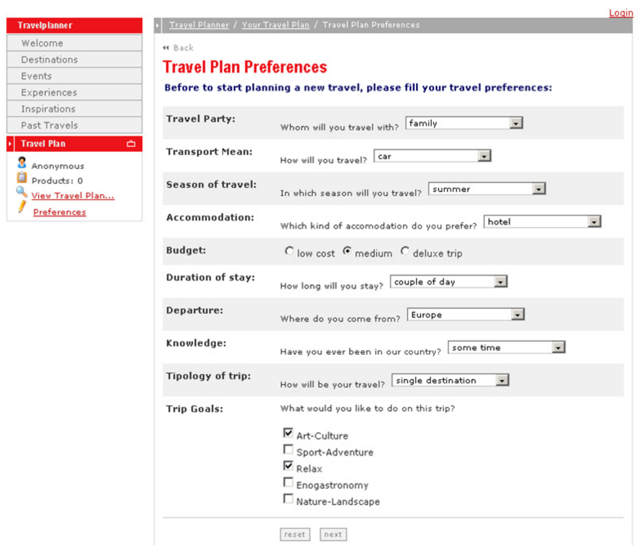
Search
If you want to specify the characteristics of your searched locations, events, experiences ...

Inspirations
If you want to browse the travel plans of other users and get inspired.

Figure 2: Welcome Page

The basic model of the interaction management assumes that at each interaction stage, the user is presented with a particular *view page*, which represents any web page, and makes one *user request*, e.g., submit a product query. This leads the interaction into a *system state*, which depicts the point where the system must select the next (system) action, and lead the user to a next view page.

Figure 1 depicts the initial part of the fixed interaction flow of TP as defined by the Trip@dvice recommendation technology. In Trip@dvice the user can select travel components from catalogues (experiences, destinations, or travel events) and bundle them in a plan [14]. In Figure 1, view pages are depicted as white boxes, e.g., the *WelcomePage* representing the view shown in Figure 2, and from this page an exit arrow labelled as *SlctQrySrch* indicates

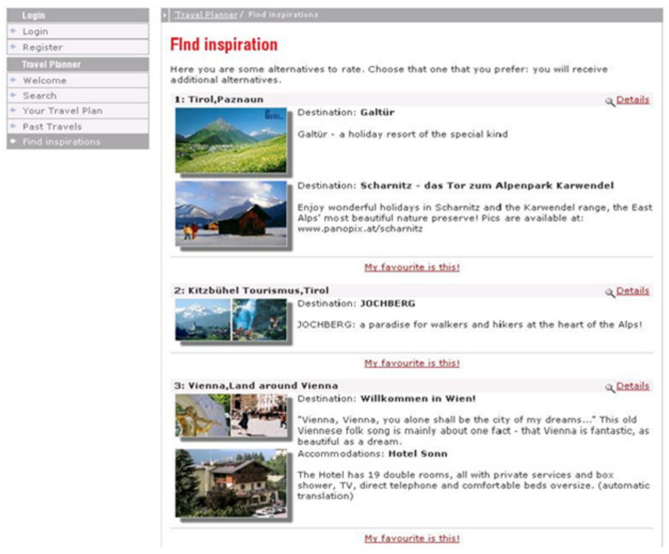


Travel Plan Preferences

Before to start planning a new travel, please fill your travel preferences:

Travel Party:	Whom will you travel with?	family
Transport Mean:	How will you travel?	car
Season of travel:	In which season will you travel?	summer
Accommodation:	Which kind of accomodation do you prefer?	hotel
Budget:	low cost [✓ medium] deluxe trip	
Duration of stay:	How long will you stay?	couple of day
Departure:	Where do you come from?	Europe
Knowledge:	Have you ever been in our country?	some time
Tipology of trip:	How will be your travel?	single destination
Trip Goals:	What would you like to do on this trip?	☑ Art-Culture ☐ Sport-Adventure ☑ Relax ☐ Enogastronomy ☐ Nature-Landscape

Figure 3: View page SpecChars



FInd inspiration

Here you are some alternatives to rate. Choose that one that you prefer: you will receive additional alternatives.

1: Tirol,Paznaun

Destination: **Galtür**

Galtür - a holiday resort of the special kind

Destination: **Scharnitz - das Tor zum Alpenpark Karwendel**

Enjoy wonderful holidays in Scharnitz and the Karwendel range, the East Alps' most beautiful nature preserve! Pics are available at: www.panopix.at/scharnitz

My favourite is this!

2: Kitzbühel Tourismus,Tirol

Destination: **JOCHBERG**

JOCHBERG: a paradise for walkers and hikers at the heart of the Alps!

My favourite is this!

3: Vienna,Land around Vienna

Destination: **Willkommen in Wien!**

"Vienna, Vienna, you alone shall be the city of my dreams..." This old Viennese folk song is mainly about one fact - that Vienna is fantastic, as beautiful as a dream.

Accommodations: **Hotel Sonn**

The Hotel has 19 double rooms, all with private services and box shower, TV, direct telephone and comfortable beds oversize. (automatic translation)

My favourite is this!

Figure 4: View page ViewProposals

the user request to initiate a query search, that brings to a system state (black ellipse), in which the system can just perform the action *AskChars*, i.e., ask for travel characteristics in the view shown in Figure 3.

In this default navigation flow of TP, from the *WelcomePage*, the users can make two requests:

- *SlctSeekInsp*: the user requests the system to compute some complete travel plan proposals, also called "Inspirations". At this point the default strategy of Trip@dvice dictates asking the user for several travel characteristics (system action *AskChars*), bringing the interaction to the view state *SpecChars* (Figure 3). After the user has specified the travel characteristics (user request *EnterChars*), the system shows in the view state *ViewProposals* some travel plan proposals (inspirations) and then after a user has tentatively selected one it will keep showing some similar proposals until the user finally choose one (see Figure 4). This flow implements a preference-based recommendation approach [13].

- *SlctQrySrch*: the user requests to formulate a query (not shown here for lack of space) to one of the available product catalogues ("Destinations", "Events", or "Experiences") in order to retrieve some item that are ranked with a social filtering approach described in [14]. The first time the user makes one of these search requests the default strategy of Trip@dvice, as for the request *SlctSeekInsp*, dictates asking the user for travel characteristics (*AskChars*). After the user has specified the travel characteristics (the user requests *EnterChars*), the system shows in the view state *ViewSrchOpts* the relevant query form, depending on the user's previously-selected query function, i.e., search for destinations, experiences, or events. This flow implements a search-based recommendation approach.

In fact, one may wonder if asking for travel characteristics (system action *AskChars*) is always beneficial. On one hand, the system needs this information. In fact, in the strategy initiated by *SlctSeekInsp*, the travel characteristics are exploited to shown relevant "Inspirations", and in the strategy initiated by *SlctQrySrch*, these are used for computing the score of the products, which depicts

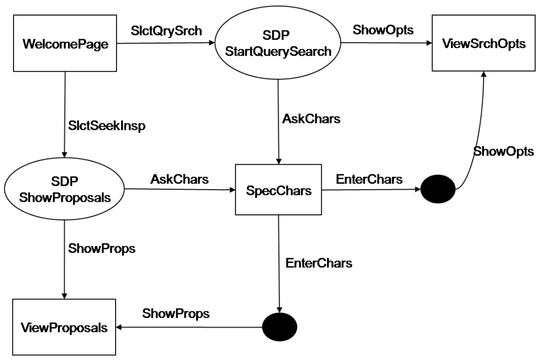

$$\begin{tikzcd}
\text{WelcomePage} \arrow[r, "\text{SlctQrySrch}"] \arrow[d, "\text{SlctSeekInsp}"] & \text{SDP StartQuerySearch} \arrow[r, "\text{ShowOpts}"] \arrow[d, "\text{AskChars}"] & \text{ViewSrchOpts} \\
\text{SDP ShowProposals} \arrow[r, "\text{AskChars}"] \arrow[d, "\text{ShowProps}"] & \text{SpecChars} \arrow[r, "\text{EnterChars}"] \arrow[d, "\text{EnterChars}"] & \bullet \arrow[u, bend right, "\text{ShowOpts}"'] \\
\text{ViewProposals} & \bullet \arrow[l, "\text{ShowProps}"'] &
\end{tikzcd}$$

Figure 5: Adaptive Navigation Flow from the Welcome Page

how much a recommendation is predicted to be suitable for the user, and for ranking the query results accordingly [14]. On the other hand, the user may not wish to invest the additional effort of specifying the travel characteristics, and might prefer to obtain immediately some travel inspirations or query results, even if they are less relevant to her.

In order to learn a better system behavior, we have incorporated two SDPs, *StartQuerySearch* and *ShowProposals*, within this portion of the default interaction flow, as shown in Figure 5[1]. Here, if the user request is *SlctQrySrch* (i.e., start a query search), the flow enters into the SDP *StartQuerySearch*, where the system should decide whether to ask the user for travel characteristics (system action *AskChars*) as in the default flow, or to deviate from the default flow, skip that step and show immediately the query search form (system action *ShowOpts*). Similarly, if the user requests *SlctSeekInsp* (travel "Inspirations"), then the flow enters into the SDP *ShowProposals*, where the system should decide whether to ask for travel characteristics (*AskChars*) as in the default flow, or show directly the proposals (*ShowProposals*) without asking the travel characteristics.

[1]We note that in the final prototype, as it will be explained later, three SDPs have been included in the full interaction flow. Here we describe just these two to illustrate the core idea.

3. ADAPTIVE RECOMMENDER MODEL

In our proposed model the recommender system is comprised of two entities: the *Information System* (IS) and the *Recommendation Agent* (RA). IS is the non-adaptive entity which is accessed by the user in order to obtain some information, e.g., when she requests a recommendation. RA is the adaptive entity whose role is to observe the user, the IS and the on-going interaction, in order to assist the user in obtaining the desired information at the appropriate stage. Specifically, when the user makes some request, e.g., submit a product query, the job of Agent is to autonomously decide what to do after this request, e.g., decide to show the requested information, or decide to ask for some additional information etc. The Agent's ultimate goal is to select those system actions that, in the long run, i.e., by the end of the interaction session, are more likely to bring the user quickly to her goal, whatever this might be. As the interaction proceeds, the Agent should optimize it's action selection policy, i.e., it should learn the best recommendation policy. In this context, as the system must decide to take an action at each stage, we model the recommendation process as a sequential decision problem, and we solve it by exploiting the Markov Decision Process (MDP) framework [20]. Then, we use this framework along with techniques from Reinforcement Learning in order to solve the policy-learning problem.

3.1 The Markov Model of the Recommendation Agent

In this section, we will model the sequential system action selection of the Recommendation Agent (RA) as a Markov Decision Process (MDP). The MDP model of the Agent includes the following four entities:

1. **A set of states** S, which represents the different situations that the Agent can observe during the interaction, and depicts information that is required by the Agent in order to learn the optimal policy. We model S through a set of state variables, which we label as the *state model*, e.g., the request of the user, the number of times the user has submitted a particular request, the number of times the system has taken a particular action etc.

2. **A set of possible system actions** A, which the Agent can perform in a given state $s \in S$, and that will produce a transition into a next state $s' \in S$. In fact, the selection of the particular action depends on the Agent's current policy. A policy is a function $\pi : S \to A$ that indicates for each state $s \in S$, the action $\pi(s) = a$ taken by the Agent in that state.

3. **A transition function** $T(s, a, s')$ which gives the probability of making a transition from state s to state s' when the Agent performs the action a.

4. **A reward function** $R(s, a)$ which assigns a numerical reward to the Agent for each action a taken in state s. As we are interested in systems which aid the user in her decision process, the reward here reflects the (positive or negative) effect of the action a. For instance, if some action is suitable for the interaction and leads the user to acquire her main goal (or a sub-goal), then the Agent could be rewarded with a positive value, e.g., +5. On the other hand, if the user didn't acquire her goal, then the Agent could be punished through a negative reward, e.g., -1. In our context the goal of the user is to select some products and add them to the current travel plan.

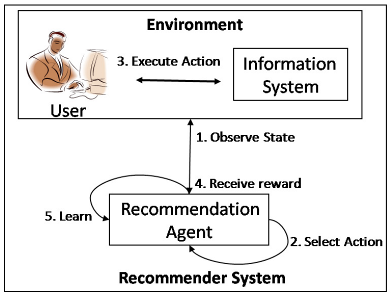


Figure 6: Interaction Model for learning the Optimal Policy

A policy, i.e., a function $\pi : S \to A$, is said to be *optimal* if there is no other policy that can give to the agent a larger expected cumulative reward. The expected cumulative reward is the expected value of the sum of the rewards that the agent gets during a session (see [10, 20] for more details). So we stress that the optimality property of a policy strictly depends on the full MDP model, i.e., the state model, the actions, the transition probabilities and the reward.

3.2 The Agent-Environment Interaction

Having described the MDP model, we will now illustrate how the Agent learns the optimal policy through RL (See Figure 6). Specifically, the Agent executes some trial-and-error sessions with the user, who is part of its environment, along with the Information System. At each stage, and after the user has made some request, the Agent observes of the current state (step 1 in Figure 6). In the current state, multiple system actions could be available to the Agent for execution. If this is the case the system ha entered a decision point (SDP) (refer to Figure 5). In the SDP states the Agent tries out (selects) one among the available actions during the session (Step 2). For each action execution (Step 3), and it's corresponding user response (next user request), the Agent receives a numerical reward from the Environment (Step 4), informing it whether its previous action was suitable for the user or not. As users continue to interact with the system, the Agent exploits the received rewards in order to learn to avoid unsuitable actions and to take the suitable ones (Step 5). The RL procedure guarantees that this process eventually leads the Agent to learn the optimal action in each SDP state, i.e., the optimal policy.

3.3 Optimal Policy Learning

In the experiments reported later, to learn the optimal policy we used the Policy Iteration algorithm [20]. Policy Iteration requires a model of the environment, i.e., the set of state transition probabilities. This means that a certain number of sessions are recorded and transition probabilities are computed based on this sample. Given such a model and an initial policy π, the Agent, using this algorithm, iterates through the following two steps at each run:

1. **Policy Evaluation**, where the Agent determines the expected cumulative reward for each possible state $s \in S$, while the system employs some policy;

2. **Policy Improvement**, where the Agent uses these reward values in order to improve the policy π.

Basically, for each state s, the algorithm determines whether the current action $a = \pi(s)$, can be improved, i.e., whether some other action a', which the Agent can take in s, is better than a, i.e., whether a' can accumulate more expected cumulative reward than

State Variable	Description
UserRequest	Label ranging over all the possible user requests.
CurrentResultSize	The number of products retrieved by the current product search query.
CharacteristicsSpecified	Whether the user, up to the current stage, has specified at least five travel characteristics (or not).
TravelPlanStatus	Whether the user, up to the current stage, has added some product to her travel plan (or not).
ResultPagesViewed	The total number of result pages that the user has currently viewed.
UserGoal	The goal of the user in her session. In our application this is fixed to "travel planning".
UserExperience	The user level of experience about tourism in Austria.
UserTightResponse	The response of the user to the tightening suggestions.
UserRelaxResponse	The response of the user to the relaxation suggestions.
UserAutoRelaxResponse	The response of the user to the auto-relaxation offer.
The position of the most recent product that the user has added to her travel plan	"Position" refers to the rank of the selected product in the result list.
Score of the most recent product that the user has added to her travel plan	The product "Score" ranges between 1 and 100 and is the recommender system's estimation of the goodness of the recommendation.

Table 1: State Model for the TP

a. If this condition holds, then π is improved to take action a' in s, i.e., $\pi(s) = a'$. This procedure is repeated for all the states in S. If any improvements occur in this phase, then a new run starts, i.e., the new policy is evaluated (first step) and then improved (second step). This process continues until no improvement is possible. At that point, the current policy is the optimal one for the given model.

We observed that we decided not to adopt other RL algorithms (as Q-learning), which are capable to learn the optimal policy online, i.e., while interacting with the users, for a major reason: in a real scenario the recommender system owner always want to keep control on the system behavior. An on-line learning procedure would have modified, without control, the current policy while interacting with the users. Moreover, in our validation we wanted to compare a non-optimal policy with an optimal one, and this would have been more difficult with an evolving version of the optimal variant, considering also the limited number of experiences (sessions) that we could use for learning.

4. MDP MODEL FOR THE TP TOOL

In this section, we will detail the MDP model for the online evaluation of the TP tool, i.e., the state model, the set of system actions, and the reward function.

4.1 State Model

Table 1 shows the 12 state variables that are included in the state model. We included the state variables that we conjectured can convey to the Agent useful information in order to learn a good policy.

The most important state variable is *UserRequest*, as it gives information about the user's response to the system actions during the session. In order to learn the optimal action, it is clear that the Agent must know what the user is actually requesting. In all, we have identified 44 different user requests. Moreover, the state variable *CurrentResultSize* depicts the size of the result set that is obtained when a product query of the user is executed. This should be relevant in two different decision situations, i.e., when some products are retrieved, i.e., *CurrentResultSize > 0*, and when the query fails, i.e., *CurrentResultSize = 0* (described in Section 4.2). We note that in case that more than 20 products are retrieved, the initial default policy of TP suggests to the user a set of product attributes to further constrain, or *tighten*, the query, in order to reduce this large result set [14, 10]. In addition, if the user query fails, TP indicates the conditions (on attributes) that if removed (*relaxed*) from the current query, would allow the system to retrieve some products. Moreover, TP also provides the *auto-relax* functionality, which asks the user's consent for automatically selecting the conditions to relax in her failing query. In our evaluation, the system should decide when to offer tightening, relaxation, or auto-relaxation to the users (refer to Section 4.2). To this end, we believe that the system should have knowledge about the previous responses of the user to these offers, given in the same session, which are depicted by the state variables *UserTightResponse*, *UserRelaxResponse*, and *UserAutoRelaxResponse*.

The state variable *CharacteristicsSpecified* is a boolean variable that depicts whether, at some stage, the user has specified at least five travel characteristics (or not), that would allow the system to compute the numerical score for products and offer relevant "Inspirations". This information could be helpful at the SDPs *ShowProposals* and *StartQuerySearch* (see Figure 5), in order to decide whether to ask the user for the characteristics. Moreover, in our scenario, the main goal of the users is to add one or more products to the travel plan, hence the Agent should have information about whether the user's travel plan is currently empty (or not), which is depicted by the state variable *TravelPlanStatus*.

The state variable *ResultPagesViewed* models the number of result pages (showing the products) that the user has viewed, up to the current stage. This information is important because the user's likelihood to interact further with an E-commerce system is related to the number of viewed result pages, up to some stage [2]. So, in our system, users could be willing to provide their travel characteristics when they have viewed a small number of result pages (and not later).

The state variable *UserExperience* models the level of previous knowledge or travel experience of the user related to tourism in Austria. This information could be important because it can significantly influence the behavior of users during the information search session [15]. For instance, at the SDPs *StartQuerySearch* and *ShowProposals*, it could be optimal not to query the experienced users for their travel characteristics. Finally, we included two state variables related to the position and the score of the most recent product that the user has added to the travel plan. The product's "position" refers to its rank in the result list, and the product's score is an estimation of its goodness for the user (as mentioned before). In fact, E-commerce users typically tend to select products that have a higher score, or are located at higher positions [2]. In our system, this information could be useful in deciding whether to push the user to add the top-scored product (with highest rank) to the travel plan (refer to Section 4.2 for details).

4.2 System Actions

Our system comprises 30 actions. We will specify the actions that are available in the SDP states, as only these can characterize

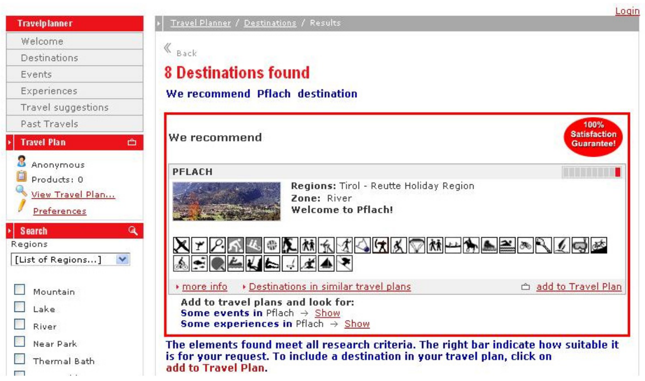
[2]http://www.ecommerce-guide.com/news/news/article.php/2203851

Decision Situation	System Action Set
Decision Situation A: The user enters the system and submits a request to initiate the query search for products, at the SDP *StartQuerySearch*	**a1**: Show the product search options to the user. **a2**: Query the user for travel characteristics before showing the product search options.
Decision Situation B: The user enters the system and requests the travel suggestions (complete travel plans), computed by the system at the SDP *ShowProposals*	**b1**: Show the product proposals to the user. **b2**: Query for travel characteristics before showing the product proposals.
Decision Situation C: After Situation A, the user submits a product query at the SDP *ExecuteQuery*, and one or more products have been retrieved, i.e., $CurrentResultSize > 0$	**c1**: Suggest product attributes for tightening the current query. **c2**: Show the result page with products' scores to the user. **c3**: Show the result page, without scores, and requests the user to specify the travel characteristics. **c4**: Show the result page (with scores), push the user to select the top ranked product, and suggest searching other product types related to the top ranked product. **c5**: Show the result page (with scores) and suggest searching other product types related to the top ranked product.
Decision Situation D: After Situation A, the user submits a product query at the SDP *ExecuteQuery*, and the query fails, i.e., $CurrentResultSize = 0$	**d1**: Suggest product attributes for relaxing the current query. **d2**: Ask the user's consent to automatically relax her product query.

Table 2: Decision Situations and the corresponding System Actions sets

the behavior of the optimal policy. For the sake of clarity, we have grouped the SDP states under four generic decision situations of the system. In Table 2, we describe these situations, and the system actions for each situation. Here the word "product" refers to a destination, a travel event, or a travel experience. Specifically, Decision Situations A and B are related to the system functionality at the SDPs *StartQuerySearch* and *ShowProposals* respectively, which we have described previously in Section 2. Situations C and D are related to the SDP *ExecuteQuery*, i.e., the situation where the user has requested the execution of a query.

Specifically, Situation C occurs when the query retrieves some results, i.e. the state variable *CurrentResultSize* > 0. Then, the system can either suggest tightening attributes (action c1), or can show one of four different types of result pages. The first type of page is the simplest one; it just shows the ranked (and scored) list of products (action c2). That is the page shown in the default interaction flow of TP. The second type shows the list where the scores of the products are not shown, because the user has not provided yet the travel characteristics, hence the system is alerting the user to specify the travel characteristics (action c3). The third type shows the scored list, and a message pushing the user to add the top-ranked destination to her plan, and suggests searching related products, i.e., search for events or experiences in the top-ranked destination (action c4). Figure 7 shows an example of this page type. The fourth type shows the scored list, and suggests (as in c4) searching for other product types in the top-ranked destination (action c5). From each result page, the user can add any of the displayed products to her travel plan. Conversely, situation D occurs when the query fails, i.e., *CurrentResultSize* $= 0$. In this case the system can either suggest the attributes that can be alternatively removed for relaxing the query and obtaining some results (action



8 Destinations found

We recommend Pflach destination

We recommend

PFLACH

Regions: Tirol - Reutte Holiday Region
Zone: River
Welcome to Pflach!

Add to travel plans and look for:
Some events in Pflach → Show
Some experiences in Pflach → Show

The elements found meet all research criteria. The right bar indicate how suitable it is for your request. To include a destination in your travel plan, click on add to Travel Plan.

Figure 7: Result Page shown after Action c4

d1), or can ask the user's consent to automatically relax the failing query (action d2).

The motivation for considering these diverse actions is that in principle we do not know what is the best system response to the user requests. For instance, in Situations A and B, some users might not be willing to specify the travel characteristics at the beginning of the interaction. For these users, the Agent could decide to show the search options (action a1) or the travel proposals (action b1), and for willing users, the Agent could decide to ask for characteristics before showing the options (action a2) or the proposals (action b2). Similarly, in Situation C, if some user is not willing to accept tightening suggestions (action c1), the Agent could decide to show any of the result pages to her (actions c2, c3, c4 or c5). Specifically, the Agent could decide to take action c3, if it needs more information about the travel's characteristics in order to score the retrieved products (see [14] for details). Moreover, in case the user has not added any product to her plan, the Agent could decide that it is the right time to push the user to make a decision (c4) or it could suggest related search functionality to provide diverse, possibly better motivating, information (c4 and c5).

4.3 Reward Function

Finally, we specify our reward function, which is based on the interpretation of the goals that users want to achieve with our system. Basically, we assign positive rewards if the user acquires her goal, with the magnitude of the reward depicting the importance of acquiring this goal. Specifically, we assign a large positive reward (+5), in the stage where the user adds some product to her travel plan, that is the main user's goal in our system. Moreover, we assign a small positive reward (+1) in the stage where the system shows a result page to the user, i.e., any of the view states that shows one or more products. This is the secondary goal of the users, since it dictates an intermediary (through necessary) step in order to achieve the main goal. Finally, we assign no reward (0) in all other situations.

5. EVALUATION METHODOLOGY

To validate our recommendation approach in an online context we built two variants of the system, the *Default* and the *Adaptive*, and we tested some hypotheses about the proposed methodology by comparing a set of performance variables across the two variants. Specifically, in a first phase, we used the *Default* (non-adaptive) variant that employed a set of default (fixed) policies. Then, in a second phase, we used the *Adaptive* variant that employed the optimal policy, which was learnt using the data collected in the first phase.

The default policies used by the *Default* variant were selected randomly during the interaction, so that the system could try the available system actions in each SDP state. In this way, different action executions, generate transitions into different possible states. Such a behavior is necessary in order to learn the set of transition probabilities (environment model), which is required by the Policy Iteration algorithm. Notwithstanding the random method of policy selection, *Default* produced diverse but "sensible" policies for the user, i.e., coherent with the current interaction flow. It is worth noting that the exploration strategy here implemented can influence the numerical estimation of the transition probability and consequently the optimal policy. But this is an unavoidable factor that one should take into account when acquiring the model interacting with real users and not trough simulations (that is going to introduce other limitations). So the model acquired with some finite number of interactions will always be an approximation of the full model that can be acquired with infinite resources (interactions).

During both the first and second phases, we logged the *sequential data* and the *performance data*. The *sequential data* collects the state and the system action (amongst other related data), at each stage of every session, and those collected in the first phase were used to learn the environment model and then the optimal policy. The *performance data* are a set of 26 performance variables, which were logged at the end of each session, such as the number of items added to the plan, or the number of travel characteristics expressed by the user.

The main goal of the evaluation was to prove that, compared to the default (non-adaptive) policy, the optimal policy supports better interactions for the users. In order to qualify this goal, we conjectured that the following hypotheses could be validated:

- **Hypothesis 1**: Compared to the Default variant, users are able to acquire their goals more *efficiently* with the Adaptive variant, i.e., in a lesser number of interaction stages and in a smaller amount of time.

- **Hypothesis 2:** Compared to the Default variant, the Adaptive variant supports more *effective* interaction sessions for the users, i.e., the task completion rate of the users increases during the usage of the Adaptive variant.

- **Hypothesis 3**: Compared to the Default variant, the Adaptive variant is able to increase the *expected response rate* of the users, i.e., the rate at which users provided the responses that were expected by the interaction designers (e.g., the users do relax the query when relaxation was suggested).

In order to test the two variants we selected a set of participants according to the guidelines mentioned in [15][3]. Specifically, 469 users were selected for testing the *Default*, and 307 of them tested also the *Adaptive* variant, hence we adopeted a within subject approach. We note that the two sessions (first the *Default* then the *Adaptive* variants) were separated by a two months period, to minimize the transfer of knowledge from session to session. We observe, that we were not able to control this aspect, since the evaluation involved another research group, and we adapted the evaluation to these constraints.

Our participants were divided into two groups, i.e., *Group A*, which executed *Task 1*, and *Group B*, which executed *Task 2*. *Task 1* was simple, and required an interaction flow that was never encountering an SDP. We decided to use the log data coming from this task to have some additional information on the transition model, but this task was designed in order to perform another test that is not described here for lack of space. *Task 2* was more complicated; it required a navigation path through the three SDPs *StartQuery-Search*, *ShowProposals*, and *ExecuteQuery*. It also informed the participants of the possible system behavior at these SDPs, e.g., that the system can request for travel characteristics, suggest or request for query changes (i.e., the tightening, relaxation and the auto-relaxation functionality). Hence, we note that we learned the optimal policy, which was used by the *Adaptive* variant, by using the sequential data collected from the users interactions along two types of tasks. One can conjecture that a better policy can be learned if the system would support a single task, but this is nonrealistic for an operational recommender system.

[3]These users were selected in collaboration with Vienna University of Economics and Business Administration, under the supervision of Prof. Josef Mazanec.

It is important to note that before carrying out the experimental evaluation, we performed a Heuristic evaluation of the Graphical User Interface (GUI) of the system, in order to judge its compliance with the standard heuristics of web usability [15]. This evaluation was carried out by usability experts as well as by students, who detected some shortcomings in our GUI. We catered for all of these limitations, and modified our GUI before the experimental evaluation. We note that this system (with the default policy) has been iteratively improved over several years and it is now used in many commercial products. Hence, in general the users testing the system had no major usability problems.

6. ANALYSIS OF THE OPTIMAL POLICY

The state model comprises $41,287,680$ possible states, but when the users evaluated our system, the number of states which actually occurred (logged) were 1709, out of which the system had to learn the optimal action for 739 SDP states. This clearly points out another limitation of this test. A longer testing of the *Default* (non-adaptive) variant could have produced log data for more states and ultimately a different optimal policy. So we stress here that the results here described apply to these particular settings.

As mentioned previously, we have grouped the SDP states under four decision situations. In this section, we shall present our analysis of the optimal behavior under each of these situations. The aim of this analysis is to illustrate succinctly the optimal policy, and in particular, the situations (states described by state variables), their corresponding actions dictated by the optimal policy, and how they differ from the default policy.

Let us recall from Section 2 that in Decision Situations A and B, the system must cope with the user's requests *SlctQrySrch* and *SlctSeekInsp*, respectively. For both situations, we found that when the user makes the first product search request at the start of her session, then it is optimal to query for travel characteristics. So, in this situation, the optimal behavior is exactly similar to the default behavior (see Figure 1). However, we also found that it is optimal to query the user for travel characteristics even on subsequent product search requests, when the user has not provided enough characteristics to score the products. This behavior is different from the default one, and is quite meaningful because if the user provides the travel characteristics, the system will better assist the user by showing the scored product list.

In Situation C, the system should decide what to do when the user's query has retrieved some products. Our analysis reveals that if the user has not added any product to her travel plan, then it is optimal to show the result page with an explicit message that pushes her to add the top-ranked destination to the plan, and offers her to make other product searches related to this destination. It turns out that when the system pushes the user to add a product, there is a greater chance that she will actually add this product to her plan,

Performance Variable	Default	Adaptive
Number of products added to the cart	2.8	2.4
Number of elapsed interaction stages	10.5	8.1*
Interaction Session Time (minutes)	29.03	15.8*
Number of result pages viewed by the user	3.3	2.7
Number of user requests for destination search	1.8	1.3*
Number of destination queries executed	1.8	1.5*

Table 3: Comparison of Performance Variables, $*$ indicates significant difference (t-test, p=0.05)

i.e., acquire her main goal. In addition, the optimal policy largely dictates suggesting tightening under-constrained queries later on in the interaction, when the users' main goal has already been acquired. This behavior implies a general unwillingness of our users to accept tightening suggestions during their sessions. In fact, in the above situation, tightening is largely suggested to assist inexperienced users who don't have much knowledge about the travel domain.

Finally, in Situation D, the system should learn to cope with a failing query. In this situation, the default behavior was to always suggest relaxation to the user. However, the optimal behavior suggests relaxation only in case it has not been offered before, or if users have accepted it in the past. In case users have always rejected relaxation in the past, the optimal policy intelligently stops suggesting relaxation, and instead, offers the auto-relax option to the user.

In conclusion, our analysis of Decision Situations A, B, C, and D has revealed that the optimal policy is different from the default policy, and dictates a more intelligent behavior that is adapted to the users. We stress that, the default policy was conjectured to be a good one in the previous development stage of the TP tool. This shows that the optimal policy can help the interaction designers to revise the flow management decisions they have made.

7. COMPARISON OF THE PERFORMANCE VARIABLES

In this section, we will present the impact of the optimal policy on the 26 selected performance variables, and we shall determine whether there was a significant differences in their values, in the two variants: *Default* and *Adaptive*. The significance is determined through a two-tailed, unequal variance t-test, where a p-value of less than 0.05 is considered statistically significant, while a p-value of less than 0.1 is indicative of a statistical trend. We found significant differences in the values of six variables, shown in Table 3 (we don't list the other variables due to space constraints).

From Table 3, we see that on the average, users added at least two products to their carts in both variants, i.e., 2.8 in the *Default* and 2.4 in the *Adaptive*. First, we note that the task required that at least one product were added, so the user were not pushed to add many products. Anyway, one reason for the smaller number in the *Adaptive* variant might be that users spent lesser time interacting with the *Adaptive* as compared to the *Default* (shown later on). We also observe that the average number of elapsed interaction stages was significantly reduced from 10.5 in the *Default* to 8.1 in the *Adaptive*, and the time elapsed during each session significantly reduced from 29 minutes in the *Default* to 15 minutes in the *Adaptive*. This implies that the optimal policy assisted our users in acquiring their goals more efficiently (quickly) during their sessions. Along with this, we see that users acquired their goals using the *Adaptive* variant with a smaller number of result page views (2.7 for *Adaptive* vs. 3.3 for *Default*), lesser number of query executions (1.5 vs. 1.8), and fewer query search requests (1.3 vs. 1.8). These observations again imply that compared to the Default variant, the Adaptive one supported more efficient sessions for the users, in which users had to spend lesser (cognitive) effort in acquiring their goals.

We also calculated the task completion rate of the users. Specifically, we determined the percentage of users who were able to complete their task during the usage of the *Default* and *Adaptive* variants, i.e., users who added at least one item to their travel plan. In our evaluation, 469 users interacted with *Default*, of which 241 users completed their tasks, i.e., a completion rate of 51%. Moreover, 307 users interacted with *Adaptive*, of which 169 completed their tasks, i.e., a completion rate of 55%. This implies that the optimal behavior lead to an additional 10% of the users to complete their tasks, compared to the behavior of the Default variant. We note that this is only due to subtle differences in the way the two systems conducted the interaction (process) as the large majority of the views and all the displayed content (recommendation items) were exactly the same.

We also analyzed expected response rate, i.e., the percentage of times the user provided the expected response for the various offers and suggestions made by the system during the interaction [4].

The TP prototype supports 14 types of offers, of which the expected response rate increased for 10 and decreased for the other 4 offers. In Table 4, we show 6 offers (we don't show all the offers due to lack of space), which dictate pushing the user to add a product or to make related product searches, querying her for travel characteristics, and offering relaxation constraints to her. For these offers, the expected user responses are to add the pushed product or to make the related product searches, specify the travel characteristics, and accept some relaxation constraint, respectively.

In Table 4, the column *System Offer* lists the particular offer of the system, columns *Default* and *Adaptive* show the number of times an offer was made in the *Default* and *Adaptive* variants respectively. Columns *ERR-Def* and *ERR-Ada* show the percentage of times the user provided the expected response in the *Default* and *Adaptive* variants respectively. We may observe that, during the usage of *Adaptive*, users added more products to the travel plan and searched for a larger number of other products when they were pushed to do so by the system. Moreover, they accepted more relaxation constraints while searching for destinations and travel events. Conversely, during the usage of *Adaptive*, users specified their travel characteristics a fewer number of times when they requested to search for destinations and travel events through the query function. We also compared the *overall* expected response rate for all the 14 offers, across the *Default* and *Adaptive* variants, i.e., the ratio of the total number of times a system offer was accepted over the total number of time an offer was made. The overall response rate during the usage of *Default* and *Adaptive* was 30% and 23% respectively. This result implies that, overall, users didn't provide the expected responses to the system's offers during their interaction with Adaptive.

In order to explain these results, we must observe that the optimal policy is aimed at gathering the largest expected cumulative reward, and the system is rewarded when users acquire their goals. Moreover, the expected responses are those that designers thought to be useful for a positive interaction output, i.e., acquiring the user goals. So, in Table 4, a large response rate for some offer implies that a large proportion of the users followed the designers' expectation. However, even if some offers were not followed with the

[4]In order to better estimate this ratio given only a small number of observations, we applied Laplace correction to the above fraction [5]

System Offer	Default	Adaptive	ERR-Def	ERR-Ada
Push the user to add the top-scored product to the plan	104	24	2%	19%
Push the user to make other searches related to the top-scored product	109	32	4%	6%
Offer relaxation while the user searches for destinations	40	26	29%	43%
Offer relaxation while the user searches for travel experiences	15	15	29%	47%
Ask travel characteristics when user searches for destinations	166	203	70%	23%
Ask travel characteristics when user searches for travel events	15	5	29%	14%

Table 4: The Expected Response Rates of the Users for Six System Offers

expected response it could be still optimal to make these offers, in some situations, in order to acquire the ultimate interaction goal (add a product to the plan). So, as we mentioned above, the optimal policy is helping the designers to understand that their conjectures about the expected responses might be wrong. In fact, users may prefer to behave differently, and may still be able to acquire their goals during the interaction. We must observe that we estimated the response rates for a relatively small user population, and in order to have a better confidence in these results, we need to evaluate our system with a larger number of users.

In our evaluation, we also logged the users' subjective ratings regarding different aspects of the quality of the interaction, e.g., how much easy it was to use the system, or whether the system assisted the user in finding her desired information, etc. A summary of our results (not shown here due to lack of space) is that compared to the Default variant, users perceived that the Adaptive one supports a more user-friendly behavior, that it has a shorter response time, it allows them to understand information more easily, and it reduces the level of ambiguity in interpreting the product descriptions. The users also perceived that the Adaptive variant shows a larger number of desired products during the interaction, and retrieves a larger number of these products during their first search attempt.

We will now refer to our research hypotheses, and discuss them according to the aforementioned results.

- **Hypothesis 1**: Compared to the Default variant, users are able to acquire their goals more efficiently with the Adaptive variant, i.e., in a lesser number of interaction stages and in a smaller amount of time.

  We have proved this hypothesis because, as shown in Table 3, users acquired their goals with Adaptive in a smaller number of stages and lesser time. Consequently, users spent lesser (cognitive) effort in acquiring their goals, by composing fewer queries, viewing lesser result pages, and making fewer product search requests.

- **Hypothesis 2**: Compared to the Default variant, the Adaptive variant supports more effective interaction sessions for the users, i.e., the task completion rate of the users increases during the usage of the Adaptive variant.

  We have proved this hypothesis because, as mentioned in Section 7, the task completion rate of the users increased during the usage of Adaptive.

- **Hypothesis 3**: Compared to the Default variant, the Adaptive variant is able to increase the expected response rate of the users, i.e., the rate at which users provided the responses that were expected by the interaction designers.

  We have not proved this hypothesis because the overall expected response rate decreased during the usage of the *Adaptive* variant (as shown in Table 4).

8. RELATED WORK

Reinforcement Learning (RL) 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 [18], the state model comprises the set of products previously bought by the user, and the system action is aimed at computing the next set of recommendations. The ultimate goal of this recommender is still to maximize an cumulative reward, i.e., but this is quite different from ours, they want to maximize the total number of items bought by the user in several recommendation sessions. Quite similarly, in [22] and [21], 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 [23] where the state model 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 [7], where the state model 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. In [25], the authors determine the presentation order of future recommendations by exploiting the current user feedback as a reward.

In all these works, the goal is similar to that of a traditional recommender system i.e., to learn what products to recommend next, or which page to show next to the user. On the contrary, we addressed a different goal, i.e., to decide which conversation action to choose from a diverse set of possible moves, at each stage of the session, in order to actively assist the users in acquiring their search goals, whatever these might be. We note that the product selection and ranking methodology used in the two system variants that we evaluated was exactly the same. In other words, the adaptive version was not providing different product recommendations, and the improvement in efficiency and effectiveness is only due to differences in the recommendation process.

Our approach is more similar RL applications to Spoken Dialogue Systems (SDS), i.e., intelligent applications that support real time, goal-oriented dialogues between humans and computers [19, 9]. A SDS must process the user's utterance and then choose in real time what information to communicate to the user and how to communicate it. However the states and actions models in these systems are completely different from our case and their main goal is still improving the quality of the communication, i.e., understanding the user utterances.

9. CONCLUSIONS AND FUTURE WORK

In a previous paper, we have proposed a novel methodology for conversational recommender systems that exploits Reinforcement Learning techniques in order to autonomously learn an optimal (user-adaptive) strategy for assisting online users in acquiring their goals. In this paper, we have applied our approach within an online travel recommender system; a prototype for the Austrian Tourism portal (Austria.info). We successfully learnt the optimal policy and showed that it dictates intelligent system actions for the users. We successfully validated its performance against a set of default non-adaptive policies. The main contribution of this paper is that with our approach, better conversational systems can be built (than traditional ones), which can assist the users in acquiring their goals more effectively and efficiently, and with a reduced amount of effort. Moreover, our approach can provide important insights into the dynamic of the human-computer interaction by suggesting to the interaction designer useful changes in the conversational policy. According to our knowledge, even if several sophisticated conversational strategies have been proposed, our work is the first attempt to learn an optimal conversational strategy in a recommender system, i.e., to adapt the strategy taking into account an objective measurement of the recommender system performance.

Our approach brings some interesting issues that should be addressed in future research. Firstly, it is difficult to predict the goal of the user in advance. In our evaluation, we fixed that to "Travel Planning", but in a real-life scenario, users might be equally interested in just window-shopping, or selecting a product quickly, rather than actively involve themselves in a detailed interaction with the system. In addition, we did not evaluate our system with different types of users, e.g., those with different types of age groups, or work backgrounds. In fact, we have shown in previous simulations, that different state variables are relevant for different types of users [11], in order to learn an optimal policy for each user category. In the future we are interested in generalizing our recommendation approach and applying it to other types of domains and information system. In this context, we plan to extend the work presented in [11] in order to determine a more general state model which could be used for learning the optimal policy for more diverse user tasks.

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] E. Agichtein, E. Brill, S. Dumais, and R. Ragno. Learning user interaction models for predicting web search result preferences. In *SIGIR '06: Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 3–10, New York, NY, USA, 2006. ACM.

[3] D. Aha and L. Breslow. Refining conversational case libraries. In *Proc. ICCBR-97*, pages 267–278. Springer, 1997.

[4] L. Ardissono, G. Petrone, and M. Segnan. A conversational approach to the interaction with web services. *Computational Intelligence*, 20(4):693–709, 2004.

[5] D. Billsus and M. Pazzani. Learning probabilistic user models. In *UM97 Workshop on Machine Learning for User Modeling*, 1997.

[6] R. Burke. Hybrid web recommender systems. In *The Adaptive Web*, pages 377–408. Springer Berlin / Heidelberg, 2007.

[7] N. Golovin and E. Rahm. Reinforcement learning architecture for web recommendations. In *Proc. ITCC'04*, pages 398–402, 2004.

[8] A. Goy, L. Ardissono, and G. Petrone. Personalization in e-commerce applications. In *The Adaptive Web*, pages 485–520. Springer Berlin / Heidelberg, 2007.

[9] J. Henderson, O. Lemon, and K. Georgila. Hybrid reinforcement/supervised learning of dialogue policies from fixed data sets. *Computational Linguistics*, 34(4):487–511, 2008.

[10] T. Mahmood and F. Ricci. Learning and adaptivity in interactive recommender systems. In *Proc. ICEC'07*, pages 75–84, 2007.

[11] T. Mahmood and F. Ricci. Adapting the interaction state model in conversational recommender systems. In *Proc. ICEC '08*, pages 1–10. ACM, 2008.

[12] T. Mahmood, F. Ricci, A. Venturini, and W. Höpken. Adaptive recommender systems for travel planning. In W. H. Peter OConnor and U. Gretzel, editors, *Proc. ENTER 2008*, pages 1–11. Springer, 2008.

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

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

[15] J. Nielsen. *Usability Engineering*. Academic Press, 1994.

[16] J. Reilly, K. McCarthy, L. McGinty, and B. Smyth. Incremental critiquing. *Knowledge-Based Systems*, 18(4-5):143–151, 2005.

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

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

[19] S. P. Singh, D. J. Litman, M. J. Kearns, and M. A. Walker. Optimizing dialogue management with reinforcement learning: Experiments with the NJFun system. *J. Artif. Intell. Res. (JAIR)*, 16:105–133, 2002.

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

[21] N. Taghipour and A. Kardan. A hybrid web recommender system based on q-learning. In *Proc. 2008 ACM Symposium on Applied Computing*, pages 1164–1168, 2008.

[22] N. Taghipour, A. Kardan, and S. S. Ghidary. Usage-based web recommendations: a reinforcement learning approach. In *Proc. 2007 ACM Conference on Recommender Systems*, pages 113–120, 2007.

[23] 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*, 2003.

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

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