Learning and Adaptivity in Interactive Recommender Systems

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

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

ABSTRACT

Recommender systems are intelligent E-commerce applications that assist users in a decision-making process by offering personalized product recommendations during an interaction session. Quite recently, conversational approaches have been introduced in order to support more interactive recommendation sessions. Notwithstanding the increased interactivity offered by these approaches, the system employs an interaction strategy that is specified apriori (at design time) and followed quite rigidly during the interaction. In this paper, we present a new type of recommender system which is capable of *learning autonomously* an adaptive interaction strategy for assisting the users in acquiring their interaction goals. We view the recommendation process as a sequential decision problem and we model it as a Markov Decision Process (MDP). We learn a model of the user behavior, and use it to acquire the adaptive strategy using Reinforcement Learning (RL) techniques. In this context, the system learns the optimal strategy by observing the consequences of its actions on the users and also on the final outcome of the recommendation session. We apply our approach within an existing travel recommender system which uses a rigid, non-adaptive support strategy for advising a user in refining a query to a travel product catalogue. The initial results demonstrate the value of our approach and show that our system is able to *improve* the non-adaptive strategy in order to learn an optimal (adaptive) recommendation strategy.

Categories and Subject Descriptors

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

General Terms

Design, Algorithms, Verification, Experimentation

Keywords

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

1. INTRODUCTION

Nowadays, the rapid development of the Internet has led to the availability of a tremendous amount of online information. In particular, E-commerce web sites have started offering a large quantity of diverse products and services to their customers. Hence, it becomes difficult for the inexperienced user to choose an item from amongst this potentially-overwhelming set of available options. Recommender systems [13] are intelligent E-commerce applications that are aimed at addressing this problem of *information overload*, by suggesting those products to the user that best suit her needs and preferences, in a given situation and context. They have been exploited for recommending travel products, books, CDs, financial services, and in many other applications [2, 7, 16]. Many recommender systems are designed to support a simple type of human-computer interaction where two phases can be identified: 1) user model construction and 2) recommendation generation. The user model is typically acquired by either exploiting data from a collection of previous user-system interactions, or by information provided by the user during the recommendation session (the user feedback). Then, the recommendation generation reduces to the identification of the subset of products that "match" the user model. For instance, in collaborative filtering-based systems, the user model is comprised of ratings provided by the user for a set of products, and the recommendations are computed by first identifying a set of similar users according to the user profiles, and then, by recommending products that have been rated highly by these similar users [2].

This behavior deviates from a more *natural* (human-to-human) type of interaction, where the user and the recommender (advisor) would interact by exchanging requests and replies, until the user accepts some recommendation. To this end, *conversational recommender systems* [20, 3, 12, 5] have been proposed. In these systems, there is no clear separation between the user model construction and the recommendation generation stages. In fact, in conversational systems a dialogue is supported, in which the user and the system interact with each other over a sequence of *interaction stages*. In this context, we define an *interaction session* as the complete set of interaction stages that constitutes a particular dialogue between the user and the system. Then, at each stage of the interaction session, the user takes some action, i.e., calls some system functionality, in order to acquire her goal, e.g., formulate and execute a query to a product catalogue. In response, the system can execute one action from among a set of alternative moves, in order to assist the user in acquiring her goal. For instance, the system could decide to execute the query and show the retrieved products, or could ask the user to input some product characteristics etc. This exchange of actions continues until the user's goal is fulfilled. Furthermore, we say that the particular action which the system executes at each stage depends on its current *strategy*. We define a *strategy* as the abstract plan of action which the system employs during the interaction session. For instance, imagine that a conversational travel recommender system has the task of suggesting a few hotels that would suit the user preferences. To this end, the system may employ the following two strategies (amongst many others):

1. **Strategy 1:** start querying the user about her preferences, and acquire enough information from her, in order to select a candidate set of hotels, or

2. **Strategy 2:** initially propose some candidates and acquire user preferences as critiques (a critique is a special type of user feedback [12]) in order to personalize the future recommendations.

A major limitation of conversational recommender systems is that they exploit a strategy which is *rigid*, i.e., one that is hard-coded in advance and remains fixed during the interaction session. In order to understand this limitation, suppose that, in the hotel recommendation scenario, the system rigidly follows Strategy 2. We argue, however, that such a strategy is not suitable for users who are, for instance, completely unwilling to provide critiques during the session, or for those users who are willing to give only partial feedback, i.e., they do not *always* provide critiques when the system asks them to do so. For such users, employing Strategy 2 might lead to failure situations, e.g., the users might suddenly quit the session if they are not satisfied with the system's responses. In this situation, we believe that the system should be able to *decide by itself* during the session, when to stop collecting critiques and to start querying the user for her preferences, i.e., abandon Strategy 2 and adopt Strategy 1. In fact, current critique-based recommenders can decide in an adaptive way the products to recommend (e.g. more similar to the query or more diverse [12]), but as they follow a fixed strategy, they cannot decide autonomously, for instance, when to stop collecting critiques and to start offering the user the possibility to sort products. In order to address this limitation, we believe that the system should have the capability of *improving* its current strategy during the session, by abandoning it and adopting a different one, until it is able to identify (and adopt)the *best* strategy for generating and offering useful recommendations to its users.

We would like to tackle these requirements by proposing a new type of recommender system that, rather than simply recommending products by following a rigid interaction design, offers a range of information and decision support functions and, during the interaction session, is *able to improve an initial strategy* and adopt a better one, i.e., a strategy that provides more adaptive support to the user for acquiring her goal. In traditional conversational systems, the particular action that the system takes at each stage of the session is hard-coded in advance, according to the fixed strategy. In our proposed approach, the system is able to *decide autonomously* which action to execute, and hence, is able to adopt different strategies. For instance, suppose that at some stage of the session, the user has taken some action, e.g., request the system to display the top-ten products bought by the customers. Then, the job of the system is to decide which action to execute in response, e.g., it can decide to ask the user to provide some product characteristics, or can decide to show the requested products etc. The system's goal is to select the action that will bring the dialogue in the best possible next state, i.e., in a state that is more closer to her goal.

In the rest of this paper we shall first define more precisely our adaptive recommendation model. Then, we shall model the recommendation process as a sequential decision problem, and in order to solve it, we shall introduce the Markov Decision Process (MDP) framework and the Reinforcement Learning paradigm [19]. We define the concept of a recommendation *policy* (which represents the interaction strategy) and we shall explain in which sense a policy can be optimal. Then, we shall apply our model within a previously-implemented travel recommender system, in order to analyze and optimize a particular type of interaction that occurs quite commonly in recommender systems, i.e., the incremental request of product features to further constrain a query to a product catalogue. The current system uses an initial (rigid) non-adaptive policy in order to decide whether to actively ask the user for additional product features (to retrieve a small set of candidate products) or to let the user to autonomously take this decision and manually change the query. We show that the system, by observing the user responses to its actions, can improve the rigid policy in order to learn an optimal one. The policy dictates asking the user for features, only when the query's result set size is greater than a certain numeric threshold. We show that different policies are learned when the agent estimates in a different way 1) the cost of each interaction, i.e., the dissatisfaction of the user to not have reached the goal, which, in this case, is the selection of the best product, and 2) the Reinforcement Learning-based discount rate parameter γ, that models the probability of the user being willing (or unwilling) to continue the interaction. Finally, we shall point out some related approaches, stress the limitations of our work and list some future extensions.

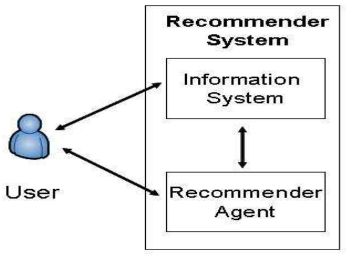


Figure 1: Adaptive Recommender Model

2. ADAPTIVE RECOMMENDER MODEL

In our proposed model, shown in Figure 1, the recommender system is comprised of two entities, namely the Information System (IS) and the Recommendation Agent (RA). Basically, IS is the non-adaptive entity which is accessed by the user in order to obtain some information. Its function is entirely controlled by the user and serves requests like displaying a query form page, displaying the query results, showing the most popular products etc. On the other hand, the Recommendation Agent (RA) is the adaptive entity whose role is to observe the user, the IS and the on-going interaction in order to support the user in his decision task, by helping him to obtain the right information at the right time. For instance, in a travel agency scenario, a traveller browses a catalogue (IS) to get suggestions/recommendations, and the travel agent (RA) helps the user to find suitable products by asking questions or pointing to some catalogue pages or products. The distinction between the Information System and the Recommender Agent is transparent to the user but it helps to understand the proposed wider role of the recommender component. In fact, this definition of recommender system contrasts with the classical idea of recommenders as information filtering tools, i.e., applications that can shade irrelevant products and hence alleviate the information overload problem.

To further explain this recommendation process, we assume that the user might execute a number of actions during her interaction with the system. We label these user actions as the *information functions*, since these are requests for various kinds of information offered by the IS. For instance, the user may request the list of top-N travel destinations, or the list of hotels liked by the people she knows. It is important to stress that the user interacts with the system to achieve some goal, e.g., to buy some product, and at each stage of the interaction session, she decides to call one among the available information function in order to attain her goal. The job of the Recommendation Agent is to *decide* what to do after this user request. In fact, this decision is not uniquely identified; the agent can, for instance, decide to show the requested information, or decide to ask for some additional information, or to recommend modifying the current request etc. We say that the agent's ultimate goal is to take those system actions that, in the long run, i.e., at the end of the interaction session, are more likely to bring the user to her goal, whatever this might be. We name these decisions as the *system actions*.

As the interaction proceeds, the agent should learn to improve or optimize the system actions that it executes, and hence, must learn the best, or an *optimal* recommendation strategy, i.e., one that is most helpful to the user in acquiring her goal. 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. Then, we use this framework along with techniques from Reinforcement Learning in order to solve the strategy-learning problem.

Our approach can be positioned within the iterative personalization process presented in [1]. The process comprises three major phases: 1) understand customers by collecting information about them, 2) deliver personalized offerings based on this information, and 3) measure the personalization impact by determining the user satisfaction with these offerings. Our approach operates within the second phase, where the goal is to learn the best strategy to deliver personalized offerings to the users, i.e., decide when to push some recommendation, acquire some user characteristics, pop-up some information, show most popular products etc.

2.1 The Markov Model of the Recommendation Agent

In this section, we shall present a general model of the Sequential Recommendation Problem as a Markov Decision Process (MDP). Later on, we shall illustrate an example by completely specifying all the elements of the model (States, Actions, Transitions, Reward). The MDP model of the recommender agent includes:

1. **A set of states** S, which represents the different 'situations' that the recommendation agent can observe as it interacts with the user. Basically, a state $s \in S$ must define what is important for the agent to know in order to take a good action. For a given situation, the complete set of states is called the *state space*. We model the state space through a set of variables, i.e., each unique combination of the values of these variables represents a unique state. These variables could be related to: 1) the state of the user (U), e.g., the number of times the user has modified her query, the action taken by the user in the current web page that she is viewing, etc., 2) the state of the agent (A), e.g., the number of times the system has taken a particular action, the system action taken at the previous stage etc., and 3) the state of the interaction session (I), e.g., the number of stages elapsed, the time elapsed since the start of the interaction session etc. In order to formulate the state representation for a given scenario, we can select the relevant variables from one or more of either U, A or I. So, for instance, S might comprise only the variable "number of stages elapsed".

2. **A set of possible system actions** A which the agent can perform in a given state s ($s \in S$) and that will produce a transition into a next state s', where $s' \in S$. In fact, the selection of the particular action depends on the *policy* of the agent, where a "policy" simply specifies how the agent's strategy is implemented during the interaction in terms of the system's actions. Hence, we use the term *optimal policy* to refer to the optimal strategy, i.e. the (improved) final strategy learnt by the agent. We formally define the policy as a function $\pi : S \rightarrow A$ that indicates for each state $s \epsilon S$, the action $\pi(s) = a$ taken by the agent in that state. In general, we assume that the environment, with which the agent interacts, is non-deterministic, i.e., after executing an action, the agent can transit into many alternative states. For instance it may "recommend a product" and this could be either selected or discarded by the user.

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. This function completely describes the non-deterministic nature of the agent's environment.

4. **A reward function** $R(s, a)$ which assigns a scalar value, also known as the *immediate reward*, to the agent for each action a taken in state s. As we are basically interested in systems that aid the user in his decision process, the immediate reward should reflect the user's acceptability of the action a. So, for instance, if the agent takes an action that is satisfactory for the user, then the agent should be rewarded with a positive immediate reward, e.g., +1. On the other hand, if the action is unsatisfactory, the agent should be punished through a negative reward, e.g., -0.01. However, the agent cannot know the reward function exactly, because the reward is assigned to it through the environment. Therefore, we shall make as few assumptions as possible about the reward by assuming that a transition into any terminal (goal) state (for instance a state in which the user buys something) will yield a positive reward to the agent, while a transition into non-terminal states will yield a small negative one. The rationale is to continue to lightly punish the agent for its actions until the goal state is reached.

The transition probabilities and the reward function completely specify the behavior, or the *model*, of the agent's environment. Some techniques for solving MDP-based problems, e.g., Dynamic Programming (DP), require previous knowledge of the environment model in order to learn the optimal policy. This model is generally learned by exploiting extensive off-line experience, with a simulation of the given task [19]. We shall describe this model-learning process later on in the paper.

2.2 The Environment-Recommender Agent Interaction

In this section, we shall describe how the MDP model is exploited along with techniques from Reinforcement Learning (RL), in order to solve the policy learning problem. Basically, Reinforcement Learning is all about learning by directly interacting with an environment, and from the consequences of actions rather than from explicit teaching [19]. In this context, we represent our recommender system by a decision-making agent that is currently

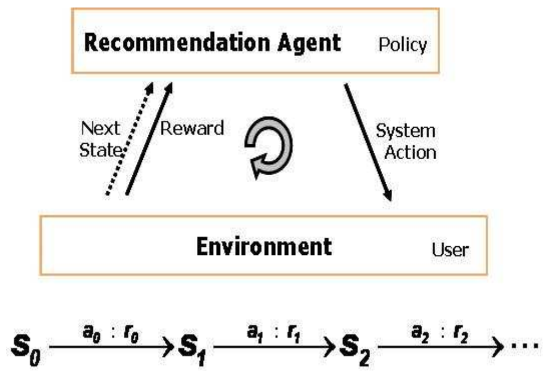

Recommendation Agent
Policy
Next State
Reward
System Action
Environment
User
$S_0 \xrightarrow{a_0 : r_0} S_1 \xrightarrow{a_1 : r_1} S_2 \xrightarrow{a_2 : r_2} \cdots$

Figure 2: Interaction Model

following some policy. The agent interacts with the user, who is part of it's environment, in a series of interaction stages (Figure 2). At each stage, the agent observes different aspects of the environment's state, and takes what it believes to be the best system action in that state, according to its policy; so, during the first stage, the agent receives a representation of the current state S_0 and decides to take action a_0. In response, the user then takes some action which the environment exploits in order to compute the next state S_1, and also the reward r_0 to the agent for action a_0. The agent exploits r_0 in order to learn to take better actions in the future. Then, the agent, after observing S_1, takes action a_1 (according to the policy) and receives a reward r_1, along with the new state S_2. Reinforcement Learning techniques guarantee that, as the agent-environment interaction session proceeds, the agent would eventually learn to take the best actions for all possible environment states, and would hence adopt the optimal policy.

In fact, the optimal policy actions are those which maximize the expected cumulative reward that the agent receives in the long run, i.e., at the end of the interaction session. Reinforcement Learning techniques guarantee that when the expected reward is maximized for each state of the environment, then the agent would converge to the optimal policy. We now present some mathematical notations for the cumulative reward and the optimal policy. In our recommendation scenario, although the interaction session will constitute a finite number of stages, we are not sure about their total count. In such a situation, the cumulative reward obtained by the agent during the session, R_T, is normally computed with an *infinite-horizon discounted model* [19]:

$$R_T = E(\sum_{t=0}^{\infty} \gamma^t R_t)$$

where R_t is the reward obtained at the t-th stage and γ is the discount rate parameter, $0 < \gamma \leq 1$. The discount rate determines the present value of the future rewards: a reward received after some stages (into the future) is likely to be of less worth, i.e., discounted, than if it is received immediately, i.e., during the current stage. So, a large value of γ implies that the agent takes the future rewards into account very strongly while a smaller value implies that the agent aims to maximize just the immediate reward for the current stage.

Now, suppose that the agent is currently following some policy π. Then, we define the *value* of a state s under π, denoted by $V^{\pi}(s)$, as the expected cumulative reward which the agent receives when it starts in s and follows π thereafter. For MDPs, $V^{\pi}(s)$ satisfies the following recursive equation:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s, \pi(s), s') V^{\pi}(s') \qquad (1)$$

where $s' \epsilon S$ is the next state reached, when the agent takes action $\pi(s)$ in s. In fact, Equation 1 is a system of $|S|$ simultaneous linear equations in $|S|$ unknowns, i.e., the value of each state in the set S. The solution of these equations yields the *value function* of the policy π, denoted by V^{π}, which specifies the value for every state $s \epsilon S$.

The optimal behavior of the agent is given by a policy $\pi^* : S \rightarrow A$ such that, for each initial state s, if the agent behaves accordingly to this policy, then R_T is a maximum. In this case, the expected reward R_T for each state s is called the *optimal value* for s, denoted by $V^*(s)$:

$$V^*(s) = \max_{\pi} E(\sum_{t=0}^{\infty} \gamma^t R_t)$$

This optimal value is unique and is the solution of the equations:

$$V^*(s) = \max_{a}(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s')), \forall s \in S$$

Given the optimal value, the optimal policy is given by:

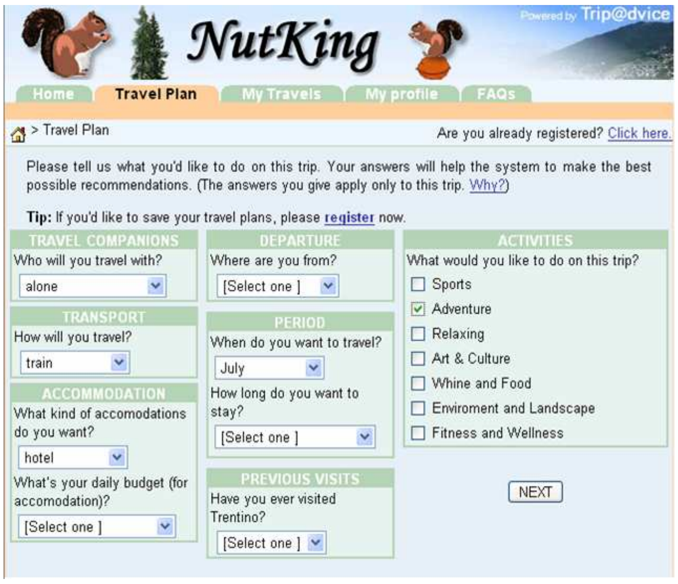
$$\pi^*(s) = arg \max_{a}(R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V^*(s'))$$

2.3 Adaptive Recommendation Algorithm

In this section, we shall describe the Policy Iteration algorithm [19] which we shall exploit later on in the paper in order to evaluate our proposed approach. Policy Iteration requires an environment model in order to compute the optimal policy. Given such a model and an initial arbitrary policy π, the agent (using this algorithm) iterates through the following two steps at each run: 1) **Policy Evaluation**, where the agent computes the value function V^{π} for the policy π by repeatedly solving the system of equations given in 1) **Policy Improvement**, where the agent uses the computed value function, in order to improve the policy π. Basically, for some state s, the algorithm determines whether the current action $\pi(s)$ can be improved, i.e., whether some other action, say a, which the agent can take in s, is better than $\pi(s)$. In fact, a would be better if the agent could accumulate more reward in s by executing a, rather than $\pi(s)$. If the 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 in the current policy is possible. Then, this current policy would be the optimal policy.

3. CASE STUDY

In this section, we will consider the application of the proposed recommendation methodology to the query tightening process supported by NutKing, a conversational recommender system [9]. NutKing combines Interactive Query Management technology with collaborative ranking in order to recommend travel products that are promoted by the regional tourism organization of Trentino (Italy). It assists the user in building a personalized *travel plan*, i.e., a collection of diverse travel products and services, e.g., accommodation, sports activities, cultural activities etc. that the user selects during her conversation with the recommender.



> Travel Plan

Are you already registered? Click here.

Please tell us what you'd like to do on this trip. Your answers will help the system to make the best possible recommendations. (The answers you give apply only to this trip. Why?)

Tip: If you'd like to save your travel plans, please register now.

TRAVEL COMPANIONS

Who will you travel with? alone

TRANSPORT

How will you travel? train

ACCOMMODATION

What kind of accomodations do you want? hotel

What's your daily budget (for accomodation)? [Select one]

DEPARTURE

Where are you from? [Select one]

PERIOD

When do you want to travel? July

How long do you want to stay? [Select one]

PREVIOUS VISITS

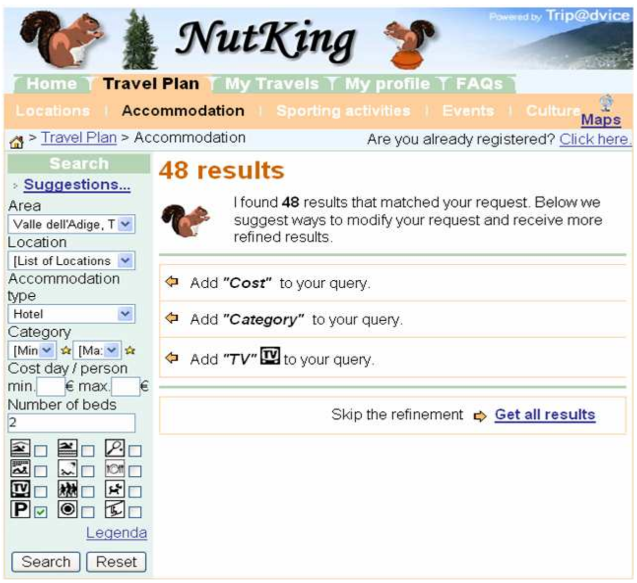
Have you ever visited Trentino? [Select one]

ACTIVITIES

What would you like to do on this trip?

- ☐ Sports
- ☑ Adventure
- ☐ Relaxing
- ☐ Art & Culture
- ☐ Whine and Food
- ☐ Enviroment and Landscape
- ☐ Fitness and Wellness

Figure 3: General Travel Preferences



> Travel Plan > Accommodation

Are you already registered? Click here.

Search

Suggestions...

Area: Valle dell'Adige, T

Location: [List of Locations

Accommodation type: Hotel

Category: [Min ☆ [Ma: ☆

Cost day / person: min. € max. €

Number of beds: 2

48 results

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

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

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

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

Skip the refinement Get all results

Figure 4: Query Tightening Suggestions

NutKing supports a human-computer interaction that mimics a typical counselling session in a real travel agency. In order to construct a plan, the user initially specifies the general preferences for her proposed trip. For example, in Figure 3, the user has specified that she would be travelling alone, she will travel by train and would like to book a hotel, she wants to travel in July and that she is interested in adventure-related activities. These preferences are used both to recommend products and also to set default constraints for the ensuing user product searches. In the next step, the system allows the user to search and add products to her plan by posing queries to the system. A query is a conjunction of constraints on the product features, wherein the user provides values for these constraints. For example, Figure 4 shows a page where the user has formulated the following query in the column on the left: q ≡ (Area=Valle dell'Adige) ∧ (Accommodation Type=Hotel) ∧ (Number of Beds=2) ∧ (Parking=true), i.e., she is searching for a hotel in the area of Valle dell'Adige, with a double-bed room, and that she would prefer the hotel that provides parking for cars. The query is defined on the left part of the user interface, and on the right part the result of the query execution is shown. If the query retrieves either too many (as it is shown in Figure 4) or no products, the system suggests useful query changes by exploiting the Interactive Query Management technology. These changes save the gist of the original user request and help the user in solving the interaction problem. This process goes on iteratively till the user selects a reasonable number of products. In this case study, we are concerned with the situation where too many products are retrieved; hence the system exploits a feature selection method in order to suggest to the user how to *tighten* the current query, i.e., reduce the size of its retrieval set, or its result size. To this end, it suggests three features from amongst those that are as yet unconstrained in the current query [9]. We refer to this process as the Query Tightening Process (QTP).

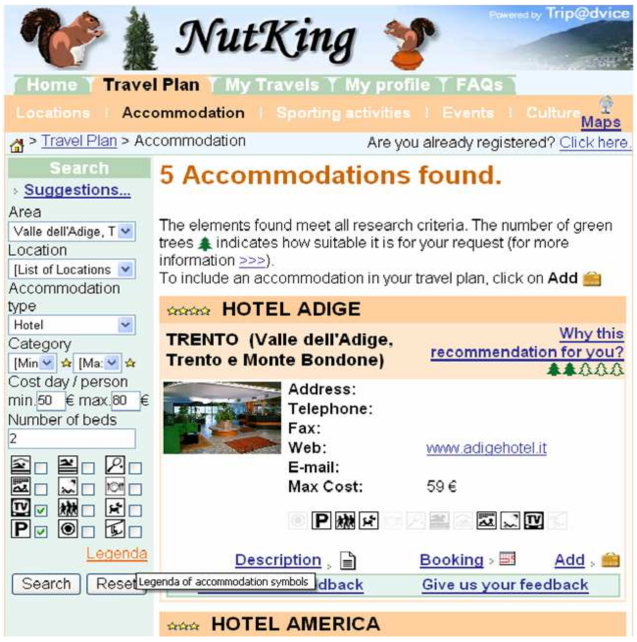
If the user accepts one of these suggestions, she is supposed to provide a preferred value for a feature, hence generating an additional constraint in the current query that could reduce its result size. For example, when the user executes the query in Figure 4, a large result size of forty eight (48) products is produced, hence the system suggests the following features to the user: "cost/day of the hotel", "hotel category" e.g. 3-star, 4-star etc, and "TV" i.e. whether the user would like to have a TV in his room. If the user accepts the features "cost" and "TV" and constrains them in the query, then a small result size of 5 products is produced which the system displays to the user. (See Figure 5).

As mentioned in Section 1, the rigid policy of NutKing prescribes always to suggest features for tightening when the result size is larger than some threshold value. Otherwise, it executes the query and shows the list of retrieved products. We have previously evaluated this policy in an online evaluation with real users (with a threshold set to 10), and the observed user acceptance rate for tightening suggestions was only 28% [14]. This low acceptance rate proves that the rigid non-adaptive policy is not optimal and clearly indicates that a rigid strategy could be improved. Hence the motivation for an adaptive recommendation model is to learn a better (adaptive) tightening policy, i.e., one which can really save the user's searching time, and can maximize the probability that she will ultimately find the desired product.

3.1 The MDP model of QTP

We will now describe the MDP model for QTP. Before defining the states, actions, and rewards of this model we must describe the Information System and the user actions. In this case, the Information System comprises five web-pages $P = \{S, QF, RT, T, G\}$ and a set of six information functions $F = \{go, execq, modq, acct, rejt, add\}$, or user actions, that we shall soon describe. At each stage of the user-system interaction session, the user requests a function $f \in F$ in some page $p \in P$. In response, the recommendation agent will take some action, causing a transition into another page $p' \epsilon P$. This process continues until the interaction terminates. Figure 6 illustrates the user actions in each page for the QTP model.

In the start page (S), which is shown to the user each time he logs on to the system the user can only decide to continue by selecting the go function. This always causes a transition to the page Query Form (QF) in which the user formulates, modifies and executes his queries. If the user executes a query ($execq$) in QF, the system can transit to either the tightening page (T) or the result set page (R).


NutKing

Powered by Trip@dvice

Home | Travel Plan | My Travels | My profile | FAQs

Locations | Accommodation | Sporting activities | Events | Culture | Maps

> Travel Plan > Accommodation   Are you already registered? Click here.

Search

Suggestions...

Area: Valle dell'Adige, T

Location: [List of Locations

Accommodation type: Hotel

Category: [Min ☆ [Ma: ☆

Cost day / person: min 50 € max 80 €

Number of beds: 2

Legenda

Search | Reset | Legenda of accommodation symbols

5 Accommodations found.

The elements found meet all research criteria. The number of green trees indicates how suitable it is for your request (for more information >>>).
To include an accommodation in your travel plan, click on **Add**

HOTEL ADIGE

TRENTO (Valle dell'Adige, Trento e Monte Bondone)   Why this recommendation for you?

Address:	
Telephone:	
Fax:	
Web:	www.adigehotel.it
E-mail:	
Max Cost:	59 €

Description | Booking | Add

dback | Give us your feedback

HOTEL AMERICA

Figure 5: Tightening Accepted

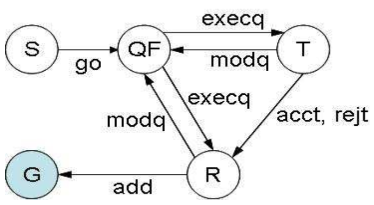


Figure 6: User Interaction Graph

The outcome depends on the system action that will be performed when the user will request to execute a query. We note that the MDP describes the actions of the system and not the actions of the user. The actions of the user and the page where the action is performed are considered as part of the state definition (see below for the state representation of QTP). In the tightening page (T) the user can either accept one of the three proposed features for tightening ($acct$) or reject the suggestion ($rejt$). Both of these user actions will lead to the result page (R), but with different results. In fact, if the user rejected tightening, then the original query is executed, and if the user accepted the tightening and provided a feature value then this user modified query is executed by the system. Finally, in the result set page (R) the user can either add a product to the cart (add) or go back to the query form (QF) page, while requesting to modify the current query ($modq$).

3.1.1 State Space Representation:

In QTP the state model is described by the following variables:

1. The **Page-UserAction variable** pua, which lists the possible combinations of pages $p \in P = \{S, QF, RT, T, G\}$ and user actions $ua \in UA = \{go, execq, modq, acct, rejt, add\}$. Thus, from Figure 6, the set of these combinations is $PUA = \{$*S-go, QF-execq, T-acct, T-rejt, T-modq, R-modq, R-add, G*$\}$.

2. The **result set size** c **of the current query** which can take on of the 3 qualitative values $small, medium, large$. A small value is that of a result size smaller than some threshold c_1. A medium value is between c_1 and another threshold c_2, while a large value is above c_2. In the experiments we set $c_1 = 20$ and $c_2 = 40$. Furthermore, we assume that $c = large$ in the initial state with $pua = S - go$ because we assume that the initial query is the empty one.

3. The **estimated result set size** ec **after tightening** is the (qualitative) system estimation of the result set size after the best feature is used for tightening by the user. (We will describe this variable in detail later on). Similarly to the variable c, we assume that $ec \in$ *small, medium, large* in the initial state.

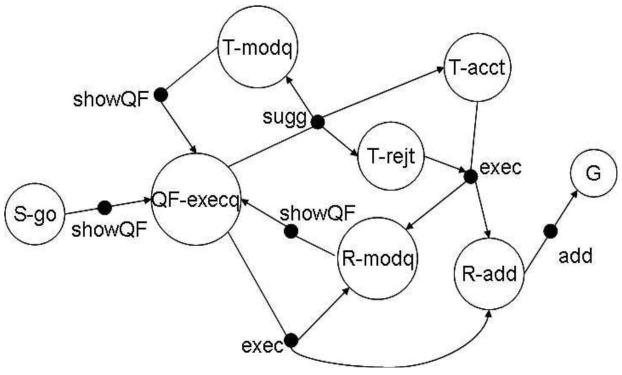


Figure 7: State Transition Diagram

As $|PUA| = 8$ and both c and ec can take three values, the number of possible states is $8*3*3 = 63$. However, as ec can never be greater than c and there is only one (initial) state with $pua = s - go$ which we consider separately, there are 6 (c, ec) possible combinations and the total number of states is $7*6+1 = 43$. These combinations are: $(s, s), (m, s), (m, m), (l, s), (l, m), (l, l)$, where $s = small, m = medium$ and $l = large$.

3.1.2 System actions and state transition diagram

In QTP there are four possible system actions: show the query form page ($showQF$); suggest tightening features ($sugg$); execute the current query ($exec$); add a product to cart (add). Figure 7 depicts how these actions (black nodes) cause transitions between the states (white nodes). For simplicity, we illustrate below only the pua variable in the state representation.

In state *S-go*, the agent executes $showQF$ that causes a transition into state *QF-execq*. This state models a situation where the query form is displayed, the user has formulated a query and has requested its execution. From here, according to the values of c and ec, the system decides to either execute the query ($exec$) going either to states *R-modq* or *R-add*, or to suggest tightening ($sugg$). In the first case, the outcome depends on the user, i.e., whether she decides to modify the query *R-modq* or to add an item to the cart *R-add*. If the system suggests features for tightening ($sugg$) then the transition could be to state: *T-acct* if the user accept tightening; *T-rejt* if the user reject tightening, or *T-modq* if the user decides to modify the query. In the states *T-acct* and *T-rejt* the system can only execute either the tightened query or the original query, respectively. In states *T-modq* and *R-modq* the only possible action for the agent is to show the query form $showQF$ to allow the user to modify the query. Finally, in state *R-add*, the agent can only proceed to G (add). We observe from Figure 7 that the MDP system has a stochastic behavior only for a small subset (4) of the possible state-actions combinations. In fact, only the state-actions: (*QF-*

execq, $sugg$), (*QF-execq*, $exec$), (*T-acct*, $exec$), and (*T-rejt*, $exec$), the outcome of the system action depends on the user choice.

3.1.3 Reward

The reward for a system action is defined in such a way that: for each transition into a non-terminal state, the agent is penalized with a negative reward $cost$, unless it transits to the goal state, where the agent gets a +1 reward. In the experiments below we shall see how different $cost$ values will impact on the optimal strategy. We recall that a negative cost for each non terminal transition models the fact that each interaction has a "cognitive" cost, which we assume to be constant for the sake of simplicity.

4. EVALUATION

In this section we shall perform some experiments aimed at showing that the recommendation agent is able to improve an initial recommendation policy as it interacts with the user, and that it can finally adopt an optimal one. To this end we will conduct some simulations. In these simulations, after having defined a user behavior model, we shall learn the transition probabilities. The user behavior model describes how the user will react to the system actions, hence for instance, it describes when the user will accept or reject a tightening suggestion. After having learned the transition probability model $T(s, a, s')$ we shall apply the Policy Iteration algorithm [19] to improve the initial policy, i.e., one currently used by the Nutking Recommender, and to learn the optimal policy.

The evaluation procedure basically simulates user-system sessions or *trials*, where each trial simulates a user incrementally modifying a query to finally select/add a product and is composed of some interactions, with the initial state being $(pua = S - go, c = large, ec = large)$. We run this procedure for a set of randomly selected products (from the catalogue). In each interaction, the user takes some action which is followed by the agent's response. The interactions continue until the goal state is reached ($pua = G$). In this simulation we perform a leave-one-in selection, i.e., for each trial we select 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 simulate a user who is searching for this item. In the simulation, the values used by the simulated user to tighten a query when a suggested feature is accepted or when a query is modified 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.

4.1 User behavior model

The user behavior model must tell how the simulated user will behave during the simulated interaction in the following three cases:

- (Case 1) When the user is in state *QF-execq*, how will she modify the current query.
- (Case 2) When the system suggests tightening ($sugg$), whether the user will decide to modify the query (*T-modq*), or to accept the tightening (*T-acct*), or to reject the tightening (*T-rejt*).
- (Case 3) When the system execute a query (exec), whether the user will add the test item to the cart (*R-add*) or modify the query (*R-modq*).

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 interactions with NutKing [14]). Then we use this sorting to choose the first feature to use in the initial query and to incrementally select the next feature to constrain when the user is back to state *QF-execq*. For instance, suppose that the first 4 sorted features are (f_1, f_5, f_4, f_3), then the initial query is $q = (f_1 = v_1)$ (where v_1 is the value of the first feature in the test item). Then, if the user decides to modify q, the next constraint that will be added is $(f_5 = v_5)$ (where v_5 is the value of f_5 in the test item) and the query will become $q = (f_1 = v_1)AND(f_5 = v_5)$.

When system suggests a tightening (Case 2), we may have three outcomes:

1. The user accepts tightening (*T-acct*)if one of the three suggested features has a non NULL value in the current test item. If this holds, the first of these features having a non NULL value, in the preference order of the user, is used to further constrain the query, using the value specified in the test item (as above in Case 1).
2. If the user doesn't accept tightening, and the result size is smaller than some threshold (which we set to 40 in the experiments) then the user rejects it and executes the original query (*R-rejt*).
3. In the remaining cases (i.e., for large result sets and when the tightening suggestion does not indicate a feature for whom the user is supposed to have a preferred value) the user is supposed to modify autonomously the query *T-modq*, as described for Case 1.

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

4.2 Model learning and optimal policy

Before applying the Policy Iteration algorithm, we used the user behavior model to learn the transition probabilities. The process of learning the transition model is a supervised learning task where the input is a state-action pair and the output is the resulting state. We keep track of how often each action outcome occurs and made a maximum likelihood estimate of the probability $T(s, a, s')$ using the frequency of reaching s' when a is executed in s. To this end, we ran the Passive ADP algorithm [15] for 6000 trials. Passive ADP captures the experience generated with a fixed policy to update the transition probabilities. A fixed policy implies that the agent doesn't explore it's environment and its actions in each state remain fixed. However, the success of an optimal policy depends heavily on exploration while gaining experience [19]. In our (simple) model, the agent must really choose one action among two alternatives ($sugg$ or $exec$) only in the six states with $pua =$ *QF-execq*. As there are two possible actions, the system can adopt a total of $2^6 = 64$ possible policies. We generated exploratory behavior by alternating the fixed policy amongst these 64, by randomly selecting an action for each of the 6 states after every 150 trials.

At this step, as we mentioned above, we applied the Policy Iteration algorithm to learn the optimal policy. In the experiments we analyzed the policy improvement process, and the resulting optimal policy for two different configurations: 1) different negative rewards ($cost$) for all the transitions into the non-terminal states (interaction cost); and 2) different values of the discount rate γ, which represents the percentage of the expected future reward that the agent considers while maximizing its cumulative reward. In fact, in the context of our system, γ is the probability that, given the agent has taken an action in some state, the user is willing to

	Optimal Actions for states with *pua*=*QF-execq*					
$cost$	(s,s)	(m,s)	(m,m)	(l,s)	(l,m)	(l,l)
$InitPol.$	exec	exec	exec	sugg	sugg	sugg
-0.01	exec	exec	exec	exec	exec	exec
-0.02	exec	exec	exec	exec	exec	sugg
-0.04	exec	exec	sugg	exec	sugg	sugg
-0.08	exec	sugg	sugg	sugg	sugg	sugg
-0.12	sugg	sugg	sugg	sugg	sugg	sugg

Table 1: Optimal Policies for Varying Interaction Costs ($s = small$, $m = medium$, $l = large$)

continue her interaction with the system. So, a high value of γ implies that the user is interested in continuing the interaction till the end, and hence, it is reasonable for the agent to consider a larger amount of the future reward.

5. RESULTS AND DISCUSSION

We shall now present our results and their analyses for these two configurations. We allowed the policy evaluation phase of the Policy Iteration algorithm to run for 100 trials and the algorithm itself to run for 10 policy improvement iterations. Here, we shall show the policy behavior only for the states with *pua* = *QF-execq*, i.e., when the agent must select whether to suggest tightening or to execute the query. Thus, we will list the prescribed actions (under the optimal policy) for only 6 states, i.e., for all the possible combinations of the remaining state variables c and ec. We recall that in the initial policy (implemented in NutKing), the agent suggests tightening only when $c = large$; otherwise it executes the query.

5.1 Different Interaction Costs

In our experiments we considered five possible values for $cost$ (the reward obtained for transitions to a non terminal state): -0.01, -0.02, -0.04, -0.08, -0.12. Moreover, in these experiments $\gamma = 0.85$. Table 1 shows the actions prescribed by the optimal policy for the above mentioned 6 states, for different $cost$ values, and the initial policy (Init Pol. row), that is not optimal. Here the 6 combinations of variables c and ec are shown in abbreviated form. Hence, for instance the column (l, s) shows the action prescribed for different $cost$ values when $c = large$ and $ec = small$.

These results illustrate that for each value of $cost$, the agent is able to improve an initial policy and to finally adopt an optimal one, and that the policy depends on the value of the interaction cost. We observe that when the cost is small (-0.01), the agent has learned that executing the query for all the combinations of c and ec is the optimal behavior. On the other extreme, when the cost is large (-0.12), the optimal policy dictates to suggest tightening for all the c and ec values. All the other optimal policies vary between these two extremes. In order to understand these results let us consider Figure 8, which shows a portion of the QTP state transition diagram, along with the probabilities of some transitions (specifically, for the case where c=$large$, ec=$large$). For this discussion we show only the state variable PUA, and assume that *R-add* is the goal state. Furthermore, let us consider a situation where the user has formulated a query q that has a result set that doesn't contain the current test product in the top three positions; hence, *R-add* cannot be reached. Let us further assume that if an additional feature is constrained in q, this will make the result set smaller and would bring the test product in a top-three position, and therefore *R-add* could be reached.

When the interaction $cost$ is low, (-0.01), the system reaches *R-add* by always executing q in state *QF-execq* (action $exec$). In this case, the probability that the user modifies q is very high (0.99), as compared to the probability that he adds a product to the cart (0.01) to reach *R-add*. Therefore, we need to consider what happens when the user modifies q (in state *R-modq*). In this case the system will reach *R-add* in a minimum of three transitions (the path highlighted in black): first moving to *R-modq*, then again to *QF-execq* (since the test product is not among the first three positions), then finally, after a manual modification of the query to *R-add*, since we are assuming that with one additional feature constrained in it the query will return a result set where the test product is in the top three positions. Conversely, when the interaction $cost$ is large (-0.12), the system tries to reach *R-add* by always suggesting tightening (action $sugg$). In fact this state can be reached in two steps (the path highlighted in gray) if the user accept the tightening, and that occurs with probability 0.32. Thus, although the chances of the user accepting the tightening are small (with probability 0.32), if the user does accept tightening, then one expensive interaction step can be saved. Conversely, if the user modifies the query, which actually occurs with a higher probability(0.68), the situation is equivalent to the *R-modq* scenario discussed previously, i.e., a larger number of interactions are required to reach *R-add*.

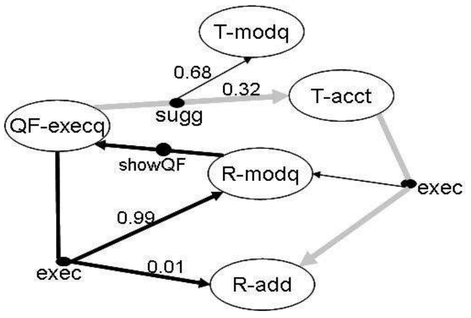


Figure 8: Different Transition Paths to $R - Add$

Thus, we see that when the interaction cost is large, the system always takes actions that have a certain probability to reduce the interaction length, i.e., speed up the process of reaching the goal, even if this probability is small. By doing so, we say that the agent takes a *risk* because the outcome of its action is actually dependent on the user's response, e.g., in Figure 8, the user might not accept tightening and he can modify q or execute q, and even if she does accept tightening, the desired product might not be in the top three positions, forcing the user to modify q one more time. Both these failure situations would lead to the execution of extra costly interactions, which is what the agent wants to avoid. Thus, there is a risk because there is no guarantee that *R-add* would **definitely** be reached. On the other hand, when the interaction cost is low, the agent doesn't take any risk; it always takes those actions that **guarantee** that, in the long run, the goal state would be finally reached. As shown in Figure 8, with this behavior the system always executes the query repeatedly modified by the user in *R-modq*, until, at the end, the test product appears in the top three positions. The behavior of the optimal policies obtained for the remaining values of $cost$ varies between the two behaviors described above. As the interaction cost increases, the system prefers to suggest tightening for more and more states.

It is worth noting that if one correlates the interaction cost with some specific user behavior, then the previous results show how different optimal policies can be offered to different types of users. For instance, the interaction cost can be related to the experience

	Optimal Actions for states with *pua* = *QF-execq*					
γ	(s,s)	(m,s)	(m,m)	(l,s)	(l,m)	(l,l)
$InitPol.$	exec	exec	exec	sugg	sugg	sugg
0.9	exec	exec	exec	exec	sugg	sugg
0.7	exec	exec	sugg	sugg	sugg	sugg
0.5	exec	sugg	sugg	sugg	sugg	sugg
0.3	exec	sugg	sugg	sugg	sugg	sugg
0.1	sugg	sugg	sugg	sugg	sugg	sugg

Table 2: Optimal Policies for Varying γ ($s = small, m = medium, l = large$)

of the users in travel recommendation domain; where a larger cost is assigned to inexperienced users. In this setting our results show that the system is able to learn that inexperienced users need more substantial system guidance (tightening) in order to quickly acquire their goals, whereas experienced users do not need much system guidance, and can eventually acquire their goals more autonomously.

Similarly, if the interaction cost is correlated with the users' navigation strategy, i.e., a small cost is assigned to window-shopping users who are not interested in buying then the adaptive system will allow them to browse by themselves without providing any extensive support. On the other hand, more goal driven buyers (who want to buy something quickly) could be assigned to a larger interaction cost model, and the system will provide plenty of useful suggestions along with speeding-up the process of reaching the goal.

5.2 Different Discount Rates

In a second set of experiments, we considered five possible values for γ: 0.9, 0.7, 0.5, 0.3, 0.1; Table 2 shows the optimal policy actions obtained for a fixed $cost = -0.04$. Here we see that when γ is large (0.9) the system prefers to execute the query for most of the six combinations of c and ec. Previously, we interpreted a high value of γ as depicting a user who is willing to continue the interaction till the end, and as a consequence, a large percentage of future rewards can be considered. Hence, the result agrees with our interpretation, and the system tends to execute the query in state *QF-execq* because this guarantees that the willing user would eventually fulfil his goal in the long run (as discussed in the previous section).

On the other hand, when γ is small (0.1), the system prefers to suggest tightening for all of the six combinations. This result also agrees with our interpretation that the system must use a low value of γ for an unwilling user, i.e., one who is less likely to be engaged in a long conversation. Thus, the system risks by giving a lot of tightening suggestions as this **can** speed-up the process of reaching the goal (as discussed previously). The optimal policy behaviors for the remaining values of γ varies between these two extremes; as γ decreases, the system prefers to suggest tightening for more and more states.

5.3 Additional Example

Our analysis have shown that the optimal policy adopted by the system depends on the selected values of *cost* and γ, and these values, in turn, should be influenced by different user behaviors and characteristics, e.g., browsing strategy, experience etc. We conclude our discussion by illustrating in a simpler MDP the emergency of different optimal interaction policy. In the MDP shown in Figure 9 a simpler (abstract) model of the Query Tightening Process is shown. Here S and G are the start (*QF-execq*) and goal (*R-add*) states respectively, and 1, 2, and 3 are some intermediate states. The system can repeatedly execute a with probability 1.0 (in states S, 1 and 2 respectively) in order to reach G, i.e., a plays the role of the query execution without tightening suggestion ($exec$). But, the system can also execute b, having a probability of 0.5 to bypass an intermediate state, but also having a 0.5 probability to get stuck in state 3. Action b is similar to $sugg$. We assume that the system receives a reward of $-r$ for each transition not leading into G, and a reward of $+1$ for the transition into G. We now introduce the Q^* notation [19]. Basically, $Q^*(s,a)$ is the value of the state s when the system takes action a in s and follows the optimal policy π thereafter. For all states $s \in S$, it is calculated as:

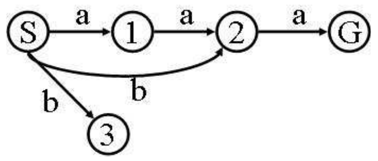


Figure 9: A Simpler Example

$$Q^*(s,a) = \max_a (R(s,a) + \gamma \sum_{s' \in S} T(s,a,s') Q^*(s', \pi(s')))$$

i.e., the action with the highest Q^* value is the best action for that state. Referring to Figure 9 and the definition of $Q^*(s,a)$, the following can be obtained:

$$Q^*(s,a) = \gamma^2 - r(1+\gamma)$$

and

$$Q^*(s,b) = -r + \gamma/2$$

The system will risk taking action b, when $Q^*(s,b) > Q^*(s,a)$. Solving for r and γ, we get that action b is preferred to a iff $r > \gamma - 1/2$.

This inequality shows that if γ is large (for instance 0.85) then the action a can be preferred if r is low (less than 0.35). Conversely if $\gamma < 0.5$, then b is always the best action. The interpretation is that if the cost of the interaction is low and the probability to keep the interaction alive is high, then it is better not to risk and execute a, whereas if the cost of the interaction is larger or if the probability to keep the interaction alive is low, it is better to risk with b.

6. RELATED WORK AND CONCLUSIONS

The MDP model has been successfully applied for adaptive bilateral negotiation in a dynamic E-commerce environment [10, 11]. In these applications, one or more software agents interact on behalf of a retailer, in order to negotiate online with a specific consumer (buyer) for the price of a certain product. The aim, then, is to find a negotiation strategy that adapts to this environment and allows the agent to reach their goals in a timely manner. Furthermore, [4] use a finite state machine (FSM), which is a generalized model of computation for MDP-based problems, in order to describe the sequence of speech acts that are admissible in a standard appointment scheduling dialogue and to check the on-going dialogue whether it follows these expectations. In this context, [18] also use an FSM in order to detect the relevant interaction strategy (from a pre-determined set) that is being pursued by a user during a dialogue, in a given situation. This strategy is then used to further guide the users during the interaction. Although these FSM applications are interesting, their behavior is quite limited, as they concentrate on just mediating the dialogue between users rather than *learning* an adaptive strategy during the dialogue.

The research work in the domain of MDP-based recommender systems is still in its infancy, with only limited results, and with a quite different model of the overall recommendation process. In [17], the authors model the states as the set of products previously bought by a user and the system actions correspond to the next possible product recommendations. Hence, in this case, the goal is similar to that of a classical recommender system i.e., to learn what products to recommend next, rather than to display a more intelligent behavior by deciding what action to choose in a more diverse set of possible moves. Adaptive recommender systems have also exploited Reinforcement Learning to display intelligent behavior, by identifying the best recommendation algorithm among some competitive alternatives in order to personalize the recommendations for some user [8], and also by exploiting the current user feedback as a reward for determining the presentation order of future recommendations [21]. Our approach is strongly inspired by [6] where the objective is to design an intelligent system that actively monitors a user attempting a task and offers assistance in the form of task guidance. In this application the process describes a hand-washing task and the system is supposed to provide cognitive assistance to people with Alzheimer's disease.

In conclusion, in this paper, we have provided a new application of MDP techniques to recommender systems and we have shown that more adaptive recommendation processes can be supported. The experimental study shows promising results in a limited situation. Our next goal is to evaluate our methodology with real users. To this end, we have applied our adaptive 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). Here, our methodology would allow the system to learn a variety of interactive decisions, e.g., whether the system should ask the user to provide the travel characteristics at the beginning of the interaction (or later on), in which situations the system should push the user to add a product to her cart, when is it better to suggest query tightening, or to execute the query, etc. The goal is to validate the improved system performance, while it employs the optimal policy, as compared to the performance of a system with a rigid policy.

7. REFERENCES

[1] G. Adomavicius and A. Tuzhilin. Personalization technologies: a process-oriented perspective. *Commun. ACM*, 48(10):83–90, 2005.

[2] 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.

[3] D. Aha and L. Breslow. Refining conversational case libraries. In *Case-Based Reasoning Research and Development, Proceedings of the 2nd International Conference on Case-Based Reasoning (ICCBR-97)*, pages 267–278. Springer, 1997.

[4] J. Alexandersson, E. Maier, and N. Reithinger. A robust and efficient three-layered dialogue component for a speech-to-speech translation system. In *Proceedings of the 33rd Annual Meeting of the ACL*, Boston, MA., 1995.

[5] L. Ardissono, A. Goy, G. Petrone, A. Felfernig, G. Friedrich, D. Jannach, M. Zanker, and R. Schaefer. A framework for the development of personalized, distributed web-based configuration systems. *AI Magazine*, pages 93–110, 2003.

[6] J. Boger, P. Poupart, J. Hoey, C. Boutilier, G. Fernie, , and A. Mihailidis. A decision-theoretic approach to task assistance for persons with dementia. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Endinburgh, Scotland, 2005.

[7] R. Burke. Hybrid recommender systems: Survey and experiments. *User Modeling and User-Adapted Interaction*, 12(4):331–370, 2002.

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

[9] N. Mirzadeh, F. Ricci, and M. Bansal. Feature selection methods for conversational recommender systems. In *Proceedings of the IEEE International Conference on e-Technology, e-Commerce and e-Services*, Hong Kong, 29 March - 1 April 2005. IEEE Press.

[10] V. Narayanan and N. R. Jennings. An adaptive bilateral negotiation model for e-commerce settings. In *7th International IEEE Conference on E-Commerce Technology*, pages 34–39, Munich, Germany, 2005.

[11] C. Raju, Y. Narahari, and K. Ravikumar. Reinforcement learning applications in dynamic pricing of retail markets. *cec*, 0:339, 2003.

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

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

[14] F. Ricci, A. Venturini, D. Cavada, N. Mirzadeh, D. Blaas, and M. Nones. Product recommendation with interactive query management and twofold similarity. In A. Aamodt, D. Bridge, and K. Ashley, editors, *ICCBR 2003, the 5th International Conference on Case-Based Reasoning*, pages 479–493, Trondheim, Norway, June 23-26 2003.

[15] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

[16] J. B. Schafer, J. A. Konstan, and J. Riedl. E-commerce recommendation applications. *Data Mining and Knowledge Discovery*, 5(1/2):115–153, 2001.

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

[18] A. Stein and U. Thiel. A conversational model of multimodal interaction in information systems. In *Proceedings of the 11th National Conference on Artificial Intelligence (AAAI '93), Washington DC, USA*, pages 283–288. AAAI Press/ MIT Press, 1993.

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

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

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