# Learning Adaptive Recommendation Strategies for Online Travel Planning

Tariq Mahmood[a],
Francesco Ricci[b], and
Adriano Venturini[c]

[a]Department of Information and Communication Technology
University of Trento, Italy
mahmood@dit.unitn.it

[b]Faculty of Computer Science
Free University of Bozen-Bolzano, Italy
fricci@unibz.it

[c]ECTRL Solutions
Trento, Italy
venturini@ectrlsolutions.com

## Abstract

Conversational recommender systems support human-computer interaction strategies in order to assist online tourists in the important activity of dynamic packaging, i.e., in building personalized travel plans and in booking their holidays. In a previous paper, we presented a novel recommendation methodology based on Reinforcement Learning, which allows conversational systems to *autonomously* improve a rigid (non-adaptive) strategy in order to learn an optimal (adaptive) one. We applied our approach within an online travel recommender system, which is supported by the Austrian Tourism portal (Austria.info). In this paper, we present the results of this online evaluation. We show that the optimal strategy adapts its actions to the served users, and deviates from a rigid default strategy. More importantly, we show that the optimal strategy is able to assist online tourists in acquiring their goals more *efficiently* than the rigid strategy, and is able to increase the willingness of the users in accepting several of the system's offers.

**Keywords:** Conversational Recommender Systems, Reinforcement Learning, Markov Decision Process, Travel Planning, Dynamic Packaging, Information Presentation and Delivery

## 1 Introduction

Conversational recommender systems are interactive E-commerce applications that assist users in their information-seeking tasks by offering personalized product recommendations during an interaction session [Bridge et al., 2006]. In particular, conversational travel recommender systems assist online tourists in the important activity of *dynamic packaging*, which is a cheap and flexible way of composing complex travel products [Fesenmaier et al., 2006]. They acquire the user preferences, either explicitly (by querying), or implicitly (by mining the user activity logs), in

order to suggest interesting travel products, hence allowing the users to easily construct their travel plans and book their preferred products. More recently, several travel recommenders have been proposed, and are now operational in major tourism portals [Venturini and Ricci, 2006], travel.yahoo.com [Aug. 15, 2008].

At each stage of the session, conversational systems execute one from amongst a set of available actions. The selected action is specified by the system's *recommendation strategy*. For instance, a conversational recommender for itinerary planning could employ the following strategy: "*explicitly acquire all the user preferences before suggesting any points of interest*". Conversational systems typically employ a *rigid* strategy, i.e., one which is determined at design time and hard-coded into the system. In fact, it is infeasible for system designers to evaluate *all* available strategies for a given task, and they select the strategy based on their experience and knowledge, which doesn't guarantee that this strategy will be optimal for all users. In a previous paper [Mahmood et al., 2007a], we have tackled these requirements by proposing a new type of recommender system, which exploits Reinforcement Learning (RL) techniques in order to *autonomously* improve a rigid strategy and *learn* the *optimal* one, given the particular model of the interaction represented by the system. In the context of RL, we use the term "policy" rather than "strategy", where a policy specifies how the strategy is implemented in terms of the system's actions. In [Mahmood and Ricci, 2007a], we validated our approach through off-line simulations within the NutKing travel recommender system, improving NutKing's rigid policy with an optimal one. Moreover, in [Mahmood and Ricci, 2008], we showed that the system learns different optimal policies for different types of user behaviours, and for different representations of the interaction process (state variables).

All these experiments were carried out offline, with simulated interactions, where the system had to learn only *one* decision. Hence, in [Mahmood et al., 2008], we proposed the application of our approach online, specifically, within the Travel Planner tool (TP) that we built for the Austrian tourism web portal (Austria.info). We showed how the rigid (default) user navigation flow of the TP could be made adaptive, by allowing the system to autonomously decide which action to perform in three particular situations of the interaction (*system decision points* or SDPs). In order to learn and validate the optimal policy, we presented our proposed system architecture and a summary of our proposed evaluation methodology.

In this paper, we present the results of this online evaluation. Specifically, Section 2 elaborates our proposed RL-based model, which we exploited in order to learn the optimal policy. Section 3 presents the specific details of our evaluation methodology, e.g., the evaluation phases, user tasks, the evaluation design etc. Section 4 presents an analysis of the learnt optimal policy. Section 5 illustrates our results, wherein we compare the values of several performance variables across the evaluation phases, in order to validate our recommendation approach. Finally, Section 6 discusses our conclusions and the future work.

## 2 Learning the Optimal Policy

In this section, we shall detail our model in order to learn the optimal policy. Our model exploits Markov Decision Process (MDP) [Sutton and Barto, 1998], modeling the human-computer interaction and the action selection decisions of the system in response to user actions. Our MDP consists of: 1) **a set of states**, observed by the system, and representing all the possible situations (modelled as variables of a state representation) which could occur as users continue to interact with the system; 2) **a set of system actions**, from which the system selects one for execution each time it is in a system decision point (SDP), hence causing a state change; 3) **a reward function**, which assigns a numerical reward to the system for taking a particular action in a given state; and 4) **a transition function**, which defines the probability of a transition to a new state, depending on a particular system action and user response. The set of all *SDP states* depicts all possible situations which could occur at the SDPs of the TP's adaptive flow (in our case there are three SDPs). The optimal policy is defined by learning the optimal action for each SDP state. The reward models the acceptability of system actions by the user. In fact, we assigned a positive reward for favourable actions (e.g., those which lead the user to acquire her goal) and a negative reward for non-favourable actions. As the interaction continues, the system optimises the reward it receives in each state, hence learning the optimal policy that is relative to the specific MDP model.

In order to validate our model, we shall investigate the following hypotheses:

**Hypothesis 1:** *The optimal policy supports more efficient product search sessions (i.e., those requiring lesser user effort) than the current policy (CP) defined by the system designer.*

**Hypothesis 2:** *Users are more inclined to follow the system suggestions generated by the optimal policy than those generated by the CP.*

**Hypothesis 3:** *The optimal policy is able to offer the products that will be preferred by the user at higher positions (than the CP) in the displayed recommendation list.*

We will test these hypotheses in Section 5. In Table 1, we present our (interaction) state representation for our policy-learning task, which comprises 12 variables.

**Table 1.** State Variables and their Descriptions

| State Variable | Description |
|---|---|
| **UserAction** | label ranging on all possible user actions |
| **CurrentResultSize (CRS)** | the number of products retrieved by a query |
| **CharacteristicsSpecified** | whether the user, up to the current stage, has specified her travel characteristics (or not), |
| **CartStatus** | whether the user, up to the current stage, has added some product to her cart (or not), |
| **ResultPagesViewed (RPV)** | the number of result pages viewed by the user up to some stage |
| **UserGoal** | the goal of the user during her session. In our application this is always "travel planning" |
| **UserExperience** | the user experience on 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-relax* offer |
| **Position of the most Recent product which the user has Added to her travel Plan (PRAP)** | "Position" refers to the product's location in the ranked list of displayed products on a given result page |
| **Score of the most Recent product which the user has Added to her travel Plan (SRAP)** | The product "Score" is a value that lies between 1 and 100 (for details, see [Venturini and Ricci, 2006]) and it is the recommender system's estimation of the goodness of the recommendation |

The tightening functionality suggests product features for reducing the result set size of a current query, in case this query has retrieved a large number of items [Mahmood and Ricci, 2007a]. In addition, if the user query fails, i.e., retrieves no products, the relaxation functionality suggests features which, if removed from the current query, would allow the system to retrieve some products. Moreover, the *auto-relax* requests the consent of the user for *automatically* relaxing her failing query. Our set of system actions comprises 31 actions. The three SDPs of TP's adaptive flow are *Execute Query*, *Start Query Search*, and *Show Proposals* (for details, see [Mahmood et al., 2008]). For the sake of clarity, we have identified four decision situations of the system that could occur at these SDPs. In, Table 2 we enlist these situations and the system action set for each situation. Here, the name of each system action is shown in parentheses, and the word "product" refers to a destination, an event, or an experience.

**Table 2.** The Decision Situations, their Explanation, and the set of System Actions available under each Situation.

| Decision Situation | Description | 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 *Start Query Search* | **1)** show the initial product suggestions to the user (*ShowProductSuggestionsView*), **2)** request the user to specify her travel characteristics (*ShowTravelContextView*) and then show the initial product suggestions |
| **Decision Situation B** | The user enters the system and requests the travel suggestions (complete travel plans), computed by the system at the SDP *Show Proposals* | **1)** show the product proposals computed by the system **(***ShowSeekInspirationsView*), **2)** request the user to specify her travel characteristics (*ShowTravelContextView*) and then show the product proposals. |
| **Decision Situation C** | After Situation A, the user submits a product query at the SDP *Execute Query*, and one or more products have been retrieved, i.e., *CurrentResultSize* > 0 | **1)** suggest features for tightening the current query (*ShowProductTightenView*), **2)** show the simplest type of result page to the user (*ShowProductRecView*), **3)** show a result page in which the system requests the user to specify the travel characteristics (*ShowProductAskCharsView*), **4)** show a result page which pushes the user to add the top ranked destination to her plan and also offers her to make related searches on this destination (*ShowAddRelSearchView*), and **5)** show a result page which suggests the user to make related product searches on the top ranked destination (*ShowRelSearchView*) |
| **Decision Situation D** | After Situation A, the user submits a product query at the SDP *Execute Query*, and the query fails, i.e., *CurrentResultSize = 0* | **1)** suggest a set of features for relaxing the current failing product search query (*ShowProductRelaxView*), **2)** acquire the consent of the user for an automatic relaxation of her product query (*ShowProductAutoRelaxView*) |

The execution of each system action leads to a particular View State (web page) being shown to the user. For instance, Fig. 1 shows the View State shown after the execution of *ShowAddRelSearchView*, where the system pushes the user to select the top recommended destination, Pflach, and offers her to search for events and experiences related to Pflach. Due to space constraints, we cannot describe all the View States (for details, see [Mahmood et al., 2007b]).

Finally, we specify our reward function that is centred on the goals of our system. We assign a large positive reward **(+5)** when the user adds some product to her travel plan, i.e., **the main goal** of the users in our system. Moreover, we assign a small positive reward **(+1)** when 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.

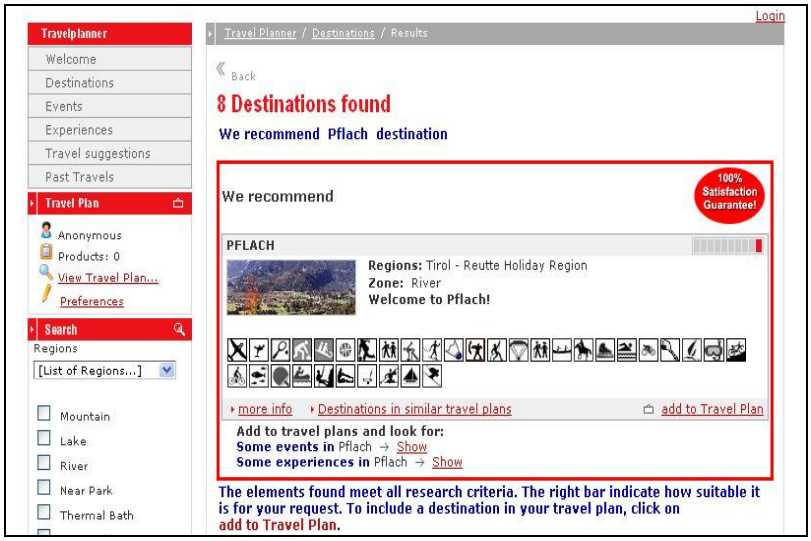


**Fig. 1.** The View State shown after the execution of *ShowAddRelSearchView*

## 3 Evaluation Methodology

We implemented two variants of our system, *VariantTrain* and *VariantTest,* in two evaluation phases: 1) **the training phase**, in which *VariantTrain* employed a set of default policies, which were selected with a uniform distribution from a set of meaningful policies. Hence, *VariantTrain tries out* as many actions as possible, in all the SDP states. Then, the ensuing user responses, modelled within the state representation, were exploited in order to learn the optimal policy that was used in 2) **the testing phase**, by *VariantTest*. During each phase, we logged the following data: 1) **the sequential data**, i.e., the state and the system action (amongst other related data), at each stage of each interaction session. The sequential data logged during the training phase was used to learn the optimal policy and 2) **the performance data**, i.e., the set of our performance variables, at the end of each session. We note that, as we stated earlier in Section 2, *the aim of the evaluation is to prove that the optimal policy, as compared to a non-optimal one selected by the system designer, supports a more effective interaction for the users*. To this end, we compared the performance

data across the two phases (with *VariantTrain* as our baseline system) in order to determine any *improvement* in this data from the training phase to the testing phase.

We selected a set of participants for carrying out the testing phases (330 for the first phase, and 214 for the second), according to the guidelines mentioned in [Nielsen, 1993]. Each participant was required to evaluate one task among two. *Task 1* was simple and dictated an interaction path for the users in which the system will never encounter a SDP. This task was introduced to perform another test that is not described here for lack of space. However the logged data from *Task 1* were used, along with those for *Task 2*, for learning the optimal policy. *Task 2* was more complicated; it dictated an interaction path through the SDP *Start Query Search* and the SDP *Execute Query*. It also informed the participants of the possible system behavior at these SDPs, i.e., that the system could request for preferences, suggest or request for query changes (i.e., the tightening, relaxation and the auto-relaxation functionality), push to add some destination to the plan, and make related searches on a destination. We employed a *within-subjects* evaluation design, in which each group (one for *Task 1* and one for *Task 2*) participated in both phases and evaluated both *VariantTrain* and *VariantTest*. This caters for the large individual variation in the skill of online users [Nielsen, 1993]. A major drawback of this design is the *transfer-of-skill* effect, i.e., participants who have evaluated *VariantTrain* might be more experienced when they evaluate *VariantTest*. An ideal solution is to allow two groups to evaluate the two variants in different order. Such a solution was not possible in our case, as the training and the testing phases did not occur concurrently. We ensured, however, that the transfer-of-skill takes place across the same task, and we observe that the two phases were separated by more than two months to limit the transfer-of-skill.

Before carrying out the experimental evaluation, we performed a Heuristic evaluation of the Graphical User Interface (GUI) of our system, in order to judge its compliance with the standard heuristics of web usability [Nielsen, 1993]. This evaluation was carried out by usability experts as well as by students, who detected several shortcomings in our GUI. We catered for all of these limitations, and modified our GUI before the experimental evaluation.

## 4 Analysing the Optimal Policy

In this section, we shall present an analysis of the learnt optimal policy. Initially, we present the current default policy of the TP tool, which we improved in order to learn the optimal policy, by specifying our selected action for each decision situation (See Table 3).

**Table 3.** Current Rigid Policy of the TP tool

| Decision Situation | Rigid Policy Action |
|---|---|
| Decision Situation A | *ShowProductSuggestionsView* |
| Decision Situation B | *ShowSeekInspirationsView* |
| Decision Situation C | *ShowProductRecView* |
| Decision Situation D | *ShowProductRelaxView* |

This policy doesn't ask the user to specify travel characteristics, or offers her tightening suggestions, auto-relaxation, or related product searches, or pushes her to add a destination to the plan. We will now analyse the optimal policy computed by the system after having observed the user interactions in the training phase. The optimal policy specifies the system action for 739 SDP states, i.e., for all the SDP states which were logged during the training phase. In order to facilitate the comparison of the optimal policy with the rigid policy, we grouped the SDP states under the four decision situations illustrated in Table 2. Our objective is to determine whether the optimal policy dictates different (better) system actions, than those dictated by the rigid policy (Table 3**Fehler! Verweisquelle konnte nicht gefunden werden.**). In fact, our analysis shows that, for each situation, the optimal policy specifies different actions for different groups of SDP states, whereas the manually designed rigid policy doesn't.

In order to illustrate that these differences bring an improvement over the rigid policy, we shall present in the next section a comparison of the performance variables. In addition, here, for each situation, we considered the frequency of all possible actions learnt by the optimal policy in that situation. This is important, as actions learnt for a larger number of states might imply that they are more beneficial for the users (in acquiring their goals), as compared to the others. In addition, we were also interested in understanding the exact situations under which the optimal policy specifies some action. To this end, we selected and analyzed the values of a subset of state variables, for each type of action. We now present our analysis for the different decision situations.

**Decision Situation A** and **Decision Situation B** (the system must cope with the initial request of the user, either for a query search (Situation A), or for a request to show some travel suggestions (Situation B))**:** In Situation A, the optimal policy specifies *ShowProductSuggestionsView* (show the initial product suggestions) for 66% of the states, and *ShowTravelContextView* (request the user for her characteristics at the beginning of her query search session) for the remaining 34% of the states, i.e., it is optimal to show the initial product suggestions, (almost) twice as much as requesting the user for her travel characteristics. In Situation B, the policy specifies both *ShowSeekInspirationsView* (show the travel suggestions to the user) and *ShowTravelContextView* for 50% of the states, i.e., both actions are specified with the same frequency. For both of these situations, our analysis reveals that, it is optimal to execute *ShowTravelContextView* when she has seen none, or only a small number of result pages, and preferably, has not, as yet, started query-searching for products. This

further implies that, *it is not feasible to request users for their characteristics, once they have started searching for the preferred products*.

**Decision Situation C** (the system must cope with a non-failing query): The optimal action with the maximum frequency is *ShowAddRelSearchView* (push the user to add the top-ranked destination to her plan, and suggest her to make related searches on this destination), which occurs for 33% of the states. Consider a user who is querying our system for destinations, and has accepted any of the system's suggestions/offers (tightening, relaxation, auto-relaxation) for modifying her query, or has executed a manually-constrained query, such that a product subset has been retrieved. *Our analysis reveals that, in this situation, it is optimal to execute ShowAddRelSearchView, i.e., push the user to add the top ranked destination, and to make searches related to this destination.* This behavior makes sense because, when the system pushes the user to add a product, there is greater chance that she will actually add this product to her plan, and acquire her main goal. In doing so, the system also fulfils our secondary goal (i.e., show a result page). Furthermore, the optimal policy dictates *ShowProductTightenView* (suggest tightening features) for 30% of the SDP states (second largest occurrence frequency). Our analysis reveals that the *optimal policy dictates tightening, largely for users who are willing to accept it.* In case the users are unwilling, the optimal policy suggests tightening only for users whose goals have already been acquired (so the policy can afford the risk of suggesting tightening), and who might require the system's assistance in continuing their interaction. Moreover, the optimal policy dictates *ShowProductRecView* (show the simplest type of result page) for 24% of the states (third largest occurrence frequency), Overall, we found that *the system largely suggests tightening, and shows the simplest result page, later on in the interaction, when their main and secondary goals have already been acquired and they have viewed a large number of result pages.* Moving on to other actions, only 7% of the states, dictate *ShowProductAskCharsView* (request the user for her characteristics during her query search session), which indicates that *users were largely unwilling to specify their characteristics during their query search sessions*.

**Decision Situation D (**the system must cope with a failing query): The optimal policy specifies *ShowProductAutoRelaxView* (offer auto-relaxation) for 53% of these states, and *ShowProductRelaxView* (suggest relaxation features) for the remaining 47% states, i.e., these actions are specified with an almost similar frequency. Our analysis reveals that *it's optimal to request auto-relaxation and suggest relaxation, earlier on in the interaction, when the main goal of the users has not been acquired*.

## 5 Experimental Results

In this section, we shall present our results for our experimental evaluation. Specifically, we selected a set of 25 performance variables, and determined whether there was a significant improvement in their values, from the training phase to the testing phase. 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 five variables, shown in Table 4**Fehler! Verweisquelle konnte nicht gefunden werden.** (we don't list the other variables due to space constraints). Here, the column "Performance Variable" lists the variables, MTrain and MTest represent their mean values in the Training and Testing phase respectively, Diff represents the mean difference between MTrain and MTest, and P-value represents the significant values.

**Table 4.** Performance Variables

| Performance Variable | MTrain | MTest | Diff | P-value |
|---|---|---|---|---|
| Number of elapsed interaction stages | 10.5 | 8.1 | -2.4 | 0.0005 |
| Number of destination queries executed | 1.8 | 1.5 | -0.3 | 0.014 |
| Number of requests for destination search | 1.8 | 1.3 | -0.5 | 0.00001 |
| Number of result pages viewed by the user | 3.3 | 2.7 | -0.6 | 0.095 |
| Number of products added to the cart | 2.8 | 2.4 | -0.4 | 0.079 |

On the average, users added at least two items to their carts in both the phases, i.e., their main and secondary goals were acquired. Hence, our results imply that *the user goals were acquired in the testing phase with a smaller number of result page views (2.7 for MTest vs. 3.3 for MTrain), query executions (1.5 vs. 1.8) and search requests (1.3 vs. 1.8), i.e., our optimal policy assisted the users in acquiring their goals more efficiently (quickly) in the testing phase, as compared to the training phase.* We must note that the users could also have acquired their goals quickly because they were more experienced with our system in the testing phase. However, the search behavior of E-commerce users is strongly influenced by the actions of the system [Katz, 2001], and the quite large distance in time between the training and test phases (two months), implies that the impact of the previously acquired experience should have been marginal. In fact, our optimal policy *did* influence the users in many ways, as we have shown above. In addition, we analyzed the acceptance rates of the users for the various system offers/suggestions. The aim was to determine whether the optimal policy was able to positively influence the willingness of the users, in accepting the system's requests, i.e., whether there was some improvement in the acceptance rates from the training to the testing phase. Amongst 12 types of requests, we found that for five of them the acceptance rate improved and decreased for two (precise data are not shown here for lack of space). Hence the optimal behavior did influence the willingness of the users in replying to the system's requests. We will now mention our proposed hypotheses, and test them according to the aforementioned results.

- **Hypothesis 1:** *The optimal policy supports more efficient product search sessions (i.e., those requiring lesser user effort) than the current policy (CP) defined by the system designer.* We have validated this hypothesis because, in the testing phase, i.e., with the optimal policy, users added products to their carts with lesser effort (see Table 4).

- **Hypothesis 2:** *Users are more inclined to follow the system suggestions generated by the optimal policy than those generated by the CP.* We have

validated this hypothesis, because the users' acceptance rate of the system's offers, produced by the optimal policy, increased.

- **Hypothesis 3:** *The optimal policy is able to offer the products that will be preferred by the user at higher positions (than the CP) in the displayed recommendation list*. This hypothesis is not validated by our experimentation evaluation, as there is no significant difference between MTest and MTrain (3.0 vs. 3.1). This result may not be unexpected, as the optimal policy is not directly optimising the position of the preferred products.

## 6 Conclusions and Future Work

In a previous paper, we have proposed a novel methodology for conversational recommender systems, which 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 of 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 non-adaptive default policies, hence showing that it is able to 1) assist the users in acquiring their goals more efficiently, and 2) increase the willingness of the users in accepting several of the system's requests/offers, 3) the proposed method can be used to improve a policy defined by a system designer. Our work is the first attempt in the domain of Travel and Tourism applications, and particularly in the domain of dynamic packaging systems, to learn a strategy of information presentation/delivery for assisting online tourists in planning their vacations and booking their holidays. This approach can be adopted by any conversational system to improve the currently selected policy. As our future work, we are interested in generalizing our recommendation approach to other tourism portals. In this context, we plan to extend our work in [Mahmood and Ricci, 2008] in order to initially determine a generic state representation and a generic reward model which could be used for learning the optimal policy for different portals.

## References


Bridge, D., Goker, M., McGinty, L., and Smyth, B. (2006). Case-based recommender systems. The Knowledge Engineering review, 20(3), 315– 320.

Fesenmaier, D. R.,Werthner, H., and Woeber, K. (2006). Destination Recommendation Systems: Behavioural Foundations and Applications. CABI Publishing.

Katz, M.A. (2001). Searching and Browsing on E-commerce Sites: Frequency, Efficiency and Rationale. Doctoral disseration, Rice University, Houston, TX.

Mahmood, T. and Ricci, F. (2007a). Learning and adaptivity in interactive recommender systems. In Proceedings of the ICEC'07 Conference, Minneapolis, USA, 75-84.

Mahmood, T., Cavada, D., Ricci, F., and Venturini, A. (2007b). Search and recommendation functionality. Technical Report D5.1, eTourism Competence Center Austria, Technikerstr. 21a, ICT-Technologiepark, 6020 Innsbruck.

Mahmood, T. and Ricci, F. (2008). Adapting the Interaction State Model in Conversational Recommender Systsems . In Proceedings of the ICEC'08 Conference, Innsbruck, Austria, 1-10.

Mahmood, T., Ricci, F., Venturini, A., and Höpken, W. (2008). Adaptive recommender systems for travel planning. In O'Connor, P., Höpken, W., and Gretzel, U. (eds), *Information and Communication Technologies in Tourism 2008, proceedings of ENTER 2008 International Conference*, Innsbruck, 2008. Springer, 1-11.

Nielsen, J. (1993). *Usability engineer*ing. San Francisco: Morgan Kaufmann Publisher.

Sutton, R. S. and Barto, A. G. (1998). Reinforcement Learning: An Introduction. MIT Press.

Venturini, A. and Ricci, F. (2006). Applying trip@dvice recommendation technology to www.visiteurope.com. In Proceedings of the 17th European Conference on Artificial Intelligence, Riva del Garda, Italy, Aug 28th - Sept 1st, 607-611.

Zanker, M, Fuchs, M., Höpken, W., Tuta, M., and Müller, N. Evaluating Recommender Systems in Tourism — A Case Study from Austria. In *Information and Communication Technologies in Tourism 2008, proceedings of ENTER 2008 International Conference*, Innsbruck, 2008. Springer, 24-34.