Usage-Based Web Recommendations: A Reinforcement Learning Approach

Nima Taghipour
Amirkabir University of Technology
Department of Computer Engineering
424, Hafez, Tehran, Iran
0098 (21) 22286275
n-taghipour@aut.ac.ir

Ahmad Kardan
Amirkabir University of Technology
Department of Computer Engineering
424, Hafez, Tehran, Iran
0098 (21) 64542729
aakardan@aut.ac.ir

Saeed Shiry Ghidary
Amirkabir University of Technology
Department of Computer Engineering
424, Hafez, Tehran, Iran
0098 (21) 64542737
shiry@aut.ac.ir

ABSTRACT

Information overload is no longer news; the explosive growth of the Internet has made this issue increasingly serious for Web users. Users are very often overwhelmed by the huge amount of information and are faced with a big challenge to find the most relevant information in the right time. Recommender systems aim at pruning this information space and directing users toward the items that best meet their needs and interests. Web Recommendation has been an active application area in Web Mining and Machine Learning research. In this paper we propose a novel machine learning perspective toward the problem, based on reinforcement learning. Unlike other recommender systems, our system does not use the static patterns discovered from web usage data, instead it learns to make recommendations as the actions it performs in each situation. We model the problem as Q-Learning while employing concepts and techniques commonly applied in the web usage mining domain. We propose that the reinforcement learning paradigm provides an appropriate model for the recommendation problem, as well as a framework in which the system constantly interacts with the user and learns from her behavior. Our experimental evaluations support our claims and demonstrate how this approach can improve the quality of web recommendations.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *Information Filtering.*

I.2.6 [**Artificial Intelligence**]: Learning.

H.2.8 [**Database Management**]: Applications – *Data mining.*

General Terms

Algorithms, Performance, Design, Experimentation

RecSys'07, October 19–20, 2007, Minneapolis, Minnesota, USA.
Copyright 2007 ACM 978-1-59593-730-8/07/0010...$5.00.

Keywords

Recommender systems, Personalization, Machine Learning, Reinforcement Learning, Web Usage Mining

1. INTRODUCTION

The amount of information available on-line is increasing rapidly with the explosive growth of the World Wide Web and the advent of e-Commerce. Although this surely provides users with more options, at the same time makes it more difficult to find the "right" or "interesting" information from this great pool of information, the problem commonly known as information overload. To address these problems, recommender systems have been introduced [14]. They can be defined as the personalized information technology used to predict a user evaluation of a particular item [3] or more generally as any system that guides users toward interesting or useful objects in a large space of possible options [1].

Recommender systems have been used in various applications ranging from predicting the products a customer is likely to buy [16], movies, music or news that might interest the user [8,22] and web pages that the user is likely to seek[2,4,7,11], which is also the focus of this paper. Web page recommendation is considered a user modeling or web personalization task. One research area that has recently contributed greatly to this problem is web mining. Most of the systems developed in this field are based on web usage mining [17] which is the process of applying data mining techniques to the discovery of usage patterns form web data. These systems are mainly concerned with analyzing web usage logs, discovering patterns from this data and making recommendations based on the extracted knowledge [4,11,15,21]. One important characteristic of these systems is that unlike traditional recommender systems, which mainly base their decisions on user ratings on different items or other explicit feedbacks provided by the user [3,6] these techniques discover user preferences from their implicit feedbacks, namely the web pages they have visited. More recently, systems that take advantage of a combination of content, usage and even structure information of the web have been introduced [9,12,13] and shown superior results in the web page recommendation problem.

We propose a different machine learning perspective toward the problem, which we believe is suitable to the nature of web page recommendation problem and has some intrinsic advantages over previous methods. Our system makes recommendations primarily based on web usage logs. We model the recommendation process as a Reinforcement Learning problem (RL) [20] or more specifically a Q-Learning problem. For this purpose we devise state and action definitions and rewarding policies, considering common concepts and techniques used in the web usage mining domain. Then we train the system using web usage logs available as the training set. During the training, the system learns to make recommendations; this is somehow different from the previous methods in which the purpose was to find explicit and static patterns or rules from the data. We'll explain this matter further in the coming sections. The choice of reinforcement learning was due to several reasons: It seems appropriate for the nature of web page recommendation problem as is discussed in section 3 and as evaluation results show; Due to the characteristics of this type of learning and the fact that we are not making decisions explicitly from the patterns discovered from the data, it provides us with a system which is constantly in the learning process; Does not need periodic updates; can easily adapt itself to changes in website structure and content and new trends in user behavior.

The organization of the paper is as follows: in section 2 we overview the related work done in recommender systems, focusing more on recent systems and on the application of reinforcement learning in these systems. We introduce our solution including modeling the problem as a Q-Learning one and the training procedure in section 3. We evaluate the proposed system in section 4. The conclusion of the paper comes in section 5 along with some recommendations for future work.

2. RELATED WORK

Recommender systems have been developed using various approaches and can be categorized in various ways [1]. Collaborative techniques [6] are the most successful and the most widely used techniques employed in these systems [3,8,21]. Recently, Web mining and especially web usage mining techniques have been used widely in web recommender systems [2,4,11,21,12]. Common approach in these systems is to extract navigational patterns from usage data by data mining techniques such as association rules and clustering, and making recommendations based on them. These approaches differ fundamentally from our method in which no static pattern is extracted from data.

RL has been previously used for recommendations in several applications. WebWatcher [7], exploits Q-Learning to guide users to their desired pages. Pages correspond to states and hyperlinks to actions, rewards are computed based on the similarity of the page content and user profile keywords. In most other systems reinforcement learning is used to reflect user feedback and update current state of recommendations. A general framework is presented in [5], which consists of a database of recommendations generated by various models and a learning module that updates the weight of each recommendation by user feedback. In [18] a travel recommendation agent is introduced which considers various attributes for trips and customers, computes each trip's value with a linear function and updates function coefficients after receiving each user feedback. RL is used for information filtering in [22] which maintains a profile for each user containing keywords of interests and updates each word's weight according to the implicit and explicit feedbacks received from the user. In [16] the recommendation problem is modeled as an MDP. The system's states correspond to user's previous purchases, rewards are based on the profit achieved by selling the items and the recommendations are made using the theory of MDP and their novel state-transition function. To the best of our knowledge our method differs from previous work, as none of them used reinforcement learning to train a system in making web site recommendations merely from web usage data.

3. WEB PAGE RECOMMENDATIONS WITH REINFORCEMENT LEARNING

3.1 Problem Definition

The specific problem which our system is supposed to solve, can be summarized as follows: the system has, as input data, the log file of users' past visits to the website, these log files are assumed to be in any standard log format, containing records each with a user ID, the sequence of pages the user visited during a session and typically the time of each page request. A user enters our website and begins requesting web pages. Considering the pages this user has requested so far the system has to predict in what other pages the user is probably interested and recommend them to her. Table 1 illustrates a sample scenario. Predictions are considered successful if the user chooses to visit those pages in the remaining of that session, e.g. page *c* recommended in the first step in table 1. Obviously the goal of the system would be to make the most successful recommendations.

Table 1: A sample user session and system recommendations

Visited Page	**a**	**b**	**c**	**d**	**e**	**f**
Navigation Trail	a	ab	abc	abcd	abcde	abcdef
System Prediction	{c,g}	{d,m}	{e,d}	{s,r}	{f,b}	{h}

3.2 Recommendations as a Q-Learning Problem

Reinforcement learning [20] is primarily known in machine learning research as a framework in which agents learn to choose the optimal action in each situation or state they are in. The agent is supposed to be in a specific state *s*, in each step it performs some action and transits to another state. After each transition the agent receives a reward *R(s)*. The goal of the agent is to learn which actions to perform in each state to receive the greatest accumulative reward, in its path to the goal state. The set of actions chosen in each state is called the agent's policy. One variation of this method is Q-Learning in which the agent does not compute explicit values for each state and instead computes a value function *Q(s,a)* which indicates value of performing action a in state s. Formally the value of *Q(s,a)* is the discounted sum of future rewards that will be obtained by doing action *a* in s and subsequently choosing optimal actions. In order to solve the problem with Q-Learning we need to make appropriate definitions for our states and actions, consider a reward function suiting the problem and devise a procedure to train the system using web logs available to us.

3.2.1 Using the Analogy of a Game

In order to better represent our approach toward the problem we try to use the notion of a game. In a typical scenario a web user visits pages sequentially from a web site, let's say the sequence a user *u* requested is composed of pages *a, b, c* and *d*. Each page the user requests can be considered a step or move in our game. After each step the user takes, it will be the system's turn to make a move. The system's purpose is to predict user's next move(s) with the knowledge of his previous moves. Whenever the user makes a move (requests a page), if the system has previously predicted the move, it will receive positive points and otherwise it will receive none or negative points. For example predicting a visit of page *d* after viewing pages *a* and *b* by the user in the above example yields in positive points for the system. The ultimate goal of the system would be to gather as much points as possible during a game or actually during a user visit from the web site.

Some important issues can be inferred from this simple analogy: first of all, we can see the problem certainly has a stochastic nature and like most games, the next state cannot be computed deterministically from our current state and the action the system performs due to the fact that the user can choose from a great number of moves. This must be considered in our learning algorithm and our update rules for *Q* values; the second issue is what the system actions should be, as they are what we ultimately expect the system to perform. Actions will be prediction or recommendation of web pages by the system in each state. Regarding the information each state must contain, by considering our definition of actions, we can deduct that each state should at least show the history of pages visited by the user so far. This way we'll have the least information needed to make the recommendations. This analogy also determines the basics of rewarding policy. In its simplest form it shall consider that an action should be rewarded positively if it recommends a page that will be visited in one of the consequent states, of course not necessarily the immediate next state. One last issue which is worth noting about the analogy is that it cannot be categorized as a typical 2-player game in which opponents try to defeat each other, as in this game clearly the user has no intention to mislead the system and prevent the system from gathering points. It might be more suitable to consider the problem as a competition for different recommender systems to gather more points, than a 2-player game. Because of this intrinsic difference, we cannot use self-play, a typical technique used in training RL systems [20], to train our system and we need the actual web usage data for training.

3.2.2 Modeling States and Actions

Considering the above observations we begin the definitions. We tend to keep our states as simple as possible, at least in order to keep their number manageable. Regarding the states, we can see keeping only the user trail can be insufficient. With that definition it won't be possible to reflect the effect of an action a performed in state S_i, in any consequent state S_{i+n} where $n>1$. This means the system would only learn actions that predict the immediate next page which is not the purpose of our system. Another issue we should take into account is the number of possible states: if we allow the states to contain any given sequence of page visits clearly we'll be potentially faced by an infinite number of states. What we chose to do was to limit the page visit sequences to a constant number. For this purpose we adopted the notion of N-Grams which is commonly applied in similar personalization systems based on web usage mining [11,12,19]. In this model we put a sliding window of size w on user's page visits, resulting in states containing only the last w pages requested by the user. The assumption behind this model is that knowing only the last w page visits of the user, gives us enough information to predict his future page requests. The same problem rises when considering the recommended pages' sequence in the states, for which we take the same approach of considering w' last recommendations.

Regarding the actions, we chose simplicity. Our action consists of a single page recommendation in each state. Considering multiple page recommendations might have shown us the effect of the combination of recommended pages on the user, in the expense of making our state space and rewarding policy much more complicated. The corresponding states and actions of the user session of Table 1 are presented in Figure 1 (straight arrows represent the actions performed in each state).

3.2.3 Choosing a Reward Function

The basis of reinforcement learning lies in the rewards the agent receives, and how it updates state and action values. As with most stochastic environments, we should reward the actions performed in each state with respect to the consequent state resulted both from the agent's action and other factor's in the environment on which we might not have control. These consequent states are sometimes called the after-states [20]. Here this factor is the page the user actually chooses to visit. We certainly do not have a predetermined function R(s,a) or even a state transition function $\delta(s,a)$ which gives us the next state according to current state s and performed action a.

It can be inferred that the rewards are dependent on the after state and more specifically on the intersection of previously recommended pages in each state and current page sequence of the state. If we consider each state s consists of two sequences V, R indicating the sequence of visited and previously recommended pages respectively:

$$V_s = < p^v_{s,1}, p^v_{s,2}, \dots, p^v_{s,w} >$$
$$R_s = < p^R_{s,1}, p^R_{s,2}, \dots, p^R_{s,n} > \qquad (1)$$

Where $p^v_{s,i}$ indicates the ith visited page in the state and $p^R_{s,i}$ indicates the i*th* recommended page in the state *s*. Reward for each action would be a function of $V_{s'}$ and $R_{s'}$ where S' is our next state. One tricky issue worth considering is that though tempting, we should not base on rewards on $|V_{s'} \cap R_{s'}|$ since it will cause extra credit for a single correct move. Considering the

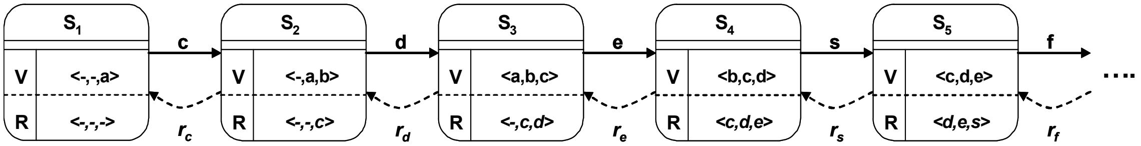

Figure 1. States and actions in the recommendation problem

above example a recommendation of page b in the first state shall be rewarded only in the transition to the second state where user goes to page b, while it will also be present in our recommendation list in the third state. To avoid this, we simply consider only the occurrence of the last page visited in the recommended pages list in state s' to reward the action performed in the previous sate *s*. To complete our rewarding procedure we take into account common metrics used in web page recommender systems. One issue is considering when the page was predicted by the system and when the user actually visited the page. According to the goal of the system this might influence our rewarding. If we consider shortening user navigation as a sign of successful guidance of user to his required information, as is the most common case in recommender systems [11,9] we should consider a greater reward for pages predicted sooner in the user's navigation path and vice versa. Another factor commonly considered in theses systems [22,11,17] is the time the user spends on a page, assuming the more time the user spends on a page the more interested he probably has been in that page. Taking this into account we should reward a successful page recommendation in accordance with the time the user spends on the page. The rewarding can be summarized as follows:

- *Assume* $\delta(s,a) = s'$
- $P_R = V_{s',w} \cap R_{s'}$
- If $p \neq \emptyset$
  For the page *p* in P_R
  $r(s,a) \mathrel{+}= reward(Dist(Rs',p),Time(p_w^v))$

Where *r(s,a)* is the reward of performing action a in state s. *Dist(R$_i$,p)* is the distance of page p from the end of the recommended pages list and *Time(p_w^v)* indicates the time user has spent on the last page of the state. Here Reward is the function combining these values to calculate *r(s,a)*. We chose a simple linear combination of these values as follows:

$$reward\ (dist\ ,Time\ ) = \alpha \times dist\ + \beta \times Time \tag{1}$$

Where $\alpha + \beta = 1$ and both α and β include a normalizing factor according to the maximum values dist and time can take.

Having put all the pieces of the model together, we can see why reinforcement learning might be a good candidate for the recommendation problem: it does not rely on any previous assumptions regarding the probability distribution of visiting a page after having visited a sequence of pages, which makes it general enough for diverse usage patterns as this distribution can take different shapes for different sequences. the nature of the problem matches perfectly with the notion of delayed reward or what is commonly known as temporal difference. The value of performing an action/recommendation might not be revealed to us in the immediate next state and sequence of actions might have led to a successful recommendation for which we must credit rewards. What the system learns is directly what it should perform, though it is possible to extract rules from the learned policy model, its decisions are not based on explicitly extracted rules or patterns from the data. One issue commonly faced in systems based on patterns extracted from training data is the need to periodically update these patterns in order to make sure they still reflect the trends residing in user behavior or the changes of the site structure or content. With reinforcement learning the system is intrinsically learning even when performing in real world, as the recommendations are the actions the system performs, and it is commonplace for the learning procedure to take place during the interaction of system with its environment.

3.3 Training the System

We chose Q-Learning as our learning algorithm. This method is primarily concerned with estimating an evaluation of performing specific actions in each state, known as Q-values. In this setting we are not concerned with evaluating each state in the sense of the accumulative rewards reachable from this state, which with respect to our system's goal can be useful only if we can estimate the probability of visiting the following states by performing each action. On the other hand Q-Learning provides us with a structure that can be used directly in the recommendation problem, as recommendations in fact are the actions and the value of each recommendation/action shows an estimation of how successful that prediction can be. Another decision is the update rule for Q values. Because of the non-deterministic nature of this problem we use the following update rule [20]:

$$\begin{aligned} Q_n(s,a) &= (1-\alpha_n)Q_{n-1}(s,a) + \\ &\quad \alpha_n[r(s,a) + \gamma \max_{a'} Q_{n-1}(\delta(s,a),a')] \end{aligned} \tag{3}$$

With

$$\alpha_n = \frac{1}{1 + visits_n(s,a)} \tag{4}$$

This rule takes into account the fact that doing the same action can yield different rewards each time it is performed in the same state. The decreasing value of α_n causes these values to gradually converge and decreases the impact of changing reward values as the training continues.

What remains about the training phase is how we actually train the system using web usage logs available. As mentioned before these logs consist of previous user sessions in the web site. Comparing to the analogy of the game they can be considered as a set of opponent's previous games and the moves he tends to make. We are actually provided with a set of actual episodes occurred in the environment, of course with the difference that no recommendations were actually made during these episodes. The training process can be summarized as the following:

```
• initial values of $Q(s,a)$ for each pair $s,a$ are set to zero
• Repeat until convergence
    o A random episode is chosen from the set of training
      episodes.
    o $s$ is set to the first step/state of the episode.
    o For each step of the episode do
        ▪ Chose an action $a$ of this state using the ε-
          greedy  policy.
        ▪ Perform action a observe the next state  and
          compute $r(s,a)$ as described before.
        ▪ Update value of $Q(s,a)$ with the above
          equation.
        ▪ $s \leftarrow s'$ .
```

The Choice of ε-greedy action selection is quite important for this specific problem as the exploration especially in the beginning phases of training, is vital. The Q values will converge if each episode, or more precisely each state-action pair is visited infinitely. In our implementation of the problem convergence was reached after a few thousand (between 3000 and 5000) visits of each episode. This definition of the learning algorithm completely follows a *TD(0)* off-policy learning procedure, as we take an estimation of future reward accessible from each state after performing each action by considering the maximum Q value in the next state.

The last modification we experimented was changing our reward function. We noticed as we put a sliding window on our sequence of previously recommended pages, practically we had limited the effect of each action to w' next states as can be seen in Figure 2. After training the system using this definition, the system was mostly successful in recommending pages visited around w' steps ahead. Although this might be quite acceptable while choosing an appropriate value for w', it tends to limit system's prediction ability as large numbers of w' make our state space enormous. To overcome this problem we devised a rather simple modification in our reward function: what we needed was to reward recommendation of a page if it is likely to be visited an unknown number of states ahead. Fortunately our definition of states and actions gives us just the information we need and ironically this information is stored in Q values of each state. The basic idea is that when an action/recommendation is appropriate in state S_i, indicating the recommended page is likely to occur in the following states, it should also be considered appropriate in state S_{i-1} and the actions in that state that frequently lead to S_i. Following this recursive procedure we can propagate the value of performing a specific action beyond the limits imposed by w'. This change is easily reflected in our learning system by considering value of $Q(s',a)$ in computation of $r(s,a)$ with a coefficient like γ. It should be taken into account that the effect of this modification in our reward function must certainly be limited as in its most extreme case where we only take this next Q value into account we're practically encouraging recommendation of pages that tend to occur mostly in the end of user sessions.

4. EXPERIMENTAL EVALUATION

We evaluated system performance in the different settings described above. We used simulated log files generated by a web traffic simulator [10] to tune our rewarding functions. The log files were simulated for a website containing 700 web pages. We pruned user sessions with a length smaller than 5 and were provided with 16000 user sessions with average length of eight. As our evaluation data set we used the web logs of the Depaul University website, made available by the author of [12]. This dataset contains 13745 sessions and 687 pages. 70% of the data set was used as the training set and the remaining was used to test the system. For our evaluation we presented each user session to the system, and recorded the recommendations it made after seeing each page the user had visited. The system was allowed to make r recommendations in each step with $r<10$ and $r < \sqrt{l}$ where l is the number of outgoing links of the last page visited by the user. This limitation on number of recommendations is adopted from [9].

4.1 Evaluation Metrics

To evaluate the recommendations we use the metrics presented in [9] because of the similarity of the settings in both systems and the fact that we believe these metrics can reveal the true performance of the system more clearly than simpler metrics. Recommendation Accuracy and Coverage are two metrics quite similar to the precision and recall metrics commonly used in information retrieval literature.

Recommendation accuracy measures the ratio of correct recommendations among all recommendations, where correct recommendations are the ones that appear in the remaining of the user session. If we have S sessions in our test log, for each visit session s after considering each page p the system generates a set of recommendations *R(p)*. To compute the accuracy, *R(p)* is compared with the rest of the session *T(p)* as follows:

$$Accuracy = \frac{\sum_s \frac{\left|\bigcup_p (T(p)\cap R(p))\right|}{\left|\bigcup_p R(p)\right|}}{S} \qquad (5)$$

Recommendation coverage on the other hand shows the ratio of the pages in the user session that the system is able to predict before the user visits them:

$$Coverage = \frac{\sum_s \frac{\left|\bigcup_p (T(p)\cap R(p))\right|}{\left|\bigcup_p T(p)\right|}}{S} \qquad (6)$$

As is the case with precision and recall, these metrics can be useful indicators of the system performance only when used in accordance to each other and lose their credibility when used individually. As an example, consider a system that recommends all the pages in each step, this system will gain 100% coverage, of course in the price of very low accuracy.

Another metric used for evaluation is called the shortcut gain which measures how many page-visits users can save if they follow the recommendations. If we call the shortened session S', the shortcut gain for each session is measured as follows:

$$ShortcutGain = \frac{\frac{|s|-|s'|}{|s|}}{S} \qquad (7)$$

4.2 Experimental Results

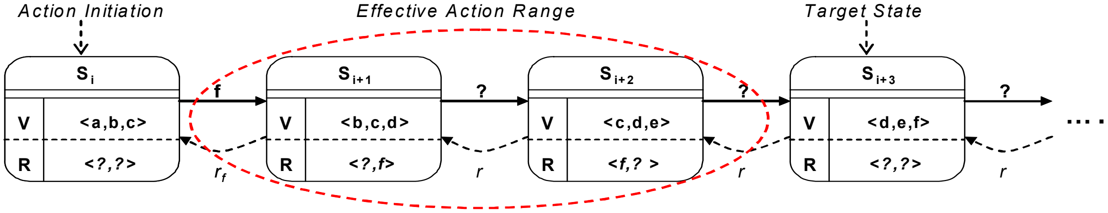

Figure 2. An example of limited action effectiveness due to the size of the recommendation window

In the first set of experiments we tested the effect of different decisions regarding state definition, rewarding function, and the learning algorithm on the system behavior. Afterwards we compared the system performance to the other common techniques used in recommendation systems.

4.2.1 Sensitivity to Active Window Size on User Navigation Trail

In our state definition, we used the notion of N-Grams by putting a sliding window on user navigation paths. The implication of using a sliding window of size w is that we base the prediction of user future visits on his w past visits. The choice of this sliding window size can affect the system in several ways. A large sliding window seems to provide the system a longer memory while on the other hand causing a larger state space with sequences that occur less frequently in the usage logs. We trained our system with different window sizes on user trail and evaluated its performance as seen in Figure 3. In these experiments we used a fixed window size of 3 on recommendation history.

As our experiments show the best results are achieved when using a window of size 3. It can be inferred form this diagram that a window of size 1 which considers only the user's last page visit does not hold enough information in memory to make the recommendation, the accuracy of recommendations improve with increasing the window size and the best results are achieved with a window size of 3. Using a window size larger than 3 results in weaker performance, it seems to be due to the fact that, as mentioned above, in these models, states contain sequences of page visits that occur less frequently in web usage logs, causing the system to make decisions based on weaker evidence. In our evaluation of the short cut gain there was a slight difference when using different window sizes.

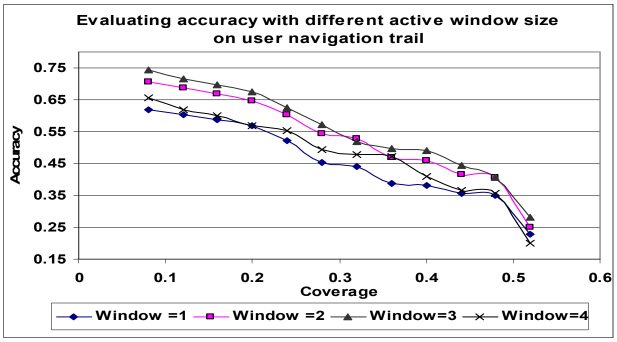

Figure 3: System performance with various user active windows size

4.2.2 Sensitivity to Active Window Size on Recommendations

In the next step we performed similar experiments, this time using a constant sliding window of size 3 on user trail and changing size of active window on recommendations history. As this window size was increased, rather interesting results was achieved as shown in Figure 4.

In evaluating system accuracy, we observed improvement up to a window of size 3, after that increasing the window size caused no improvement while resulting in larger number of states. This increase in the number of states is more intense than when the window size on user trail was increased. This is manly due to the fact that the system is exploring and makes any combination of recommendations to learn the good ones. The model consisting of this great number of states is in no way efficient, as in our experiments on the test data only 25% of these states were actually visited. In the sense of shortcut gain the system achieved, it was observed that shortcut gain increased almost constantly with increase in window size, which seems a natural consequence as described in section 3.

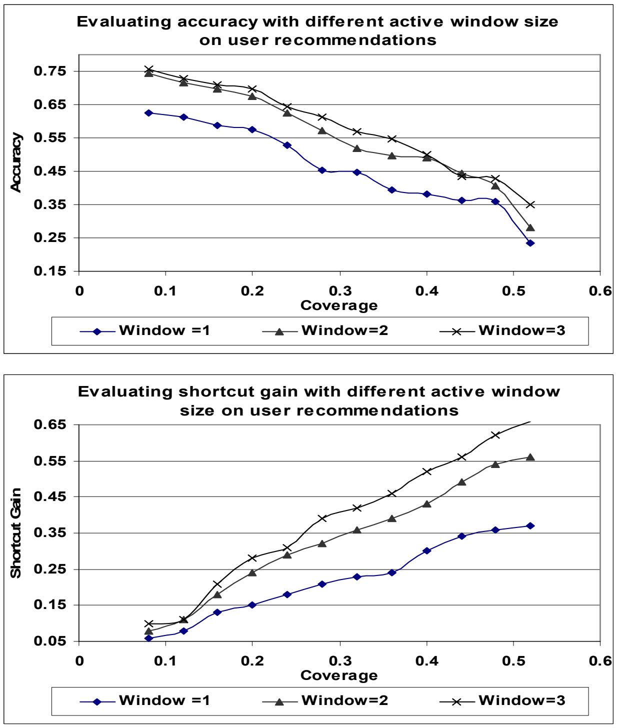

Figure 4: System performance with different active recommendation windows

4.2.3 Evaluating Different Reward Functions

Next we changed the effect of parameters constituting our reward function. First we began by not considering the *Dist* parameter, described in section 3, in our rewards. We gradually increased it's coefficient in steps of 5% and recorded the results as shown in Table 2. These results show that increasing the impact of this parameter in our rewards up to 15% of total reward can result both in higher accuracy and higher shortcut gain. Using values greater than 15% has a slight negative effect on accuracy with a slight positive effect on shortcut gain and keeping it almost constant. This seems a natural consequence since although we're paying more attention to pages that tend to appear later in the user sessions, the system's vision into the future is bounded by the size of window on recommendations. This limited vision also explains why our accuracy is not decreasing as expected.

The next set of experiments tested system performance with the reward function that considers *next state Q-value* of each action in rewarding the action performed in the previous state, as described in section 3. We began by increasing the coefficient of this factor (γ) in the reward function the same way we did for the *Dist* parameter as shown in Table 1. In the beginning increasing this value, lead to higher accuracy and shortcut gains.

After reaching an upper bound, the accuracy began to drop. In these settings, recommendations with higher values were those targeted toward the pages that occurred more frequently in the end of user sessions. These recommended pages, if recommended correctly, were only successful in predicting the last few pages in the user sessions. As expected, shortcut gain increased steadily with increase in this value up to a point where the recommendations became so inaccurate that rarely happened anywhere in the user sessions.

Table 2: System performance with varying α in the reward function (AC=Accuracy, SG=Shortcut Gain)

Coverage	Performance									
	α = 0.1		α = 0.15		α = 0.20		α = 0.25		α = 0.30	
	AC	SG	AC	SG	AC	SG	AC	SG	AC	SG
10	.75	.15	.78	.17	.76	.17	.73	.18	.69	.18
15	.71	.28	.73	.33	.72	.35	.69	.34	.65	.35
20	.69	.37	.68	.40	.67	.41	.67	.41	.61	.41
25	.65	.40	.66	.44	.65	.44	.61	.46	.58	.46
30	.55	.43	.57	.50	.54	.53	.52	.54	.49	.57
40	.48	.48	.50	.54	.45	.57	.40	.58	.36	.57
50	.36	.51	.39	.57	.33	.58	.29	.58	.27	.59

Table 3: System performance with next state Q-Values in the reward function (AC=Accuracy, SG=Shortcut Gain)

Coverage	Performance									
	γ = 0.1		γ = 0.15		γ = 0.20		γ = 0.25		γ = 0.30	
	AC	SG	AC	SG	AC	SG	AC	SG	AC	SG
10	.73	.10	.77	.12	.80	.16	.74	.17	.60	.20
15	.69	.19	.71	.23	.73	.35	.65	.33	.54	.24
20	.63	.26	.66	.36	.67	.40	.60	.43	.49	.30
25	.60	.34	.63	.43	.62	.45	.53	.49	.40	.33
30	.52	.40	.57	.50	.58	.54	.40	.62	.32	.37
40	.45	.45	.51	.56	.50	.59	.33	.64	.20	.42
50	.36	.52	.41	.59	.42	.59	.26	.64	.19	.42

4.2.4 A Comparison with other Methods

Finally we observed our system performance in comparison with two other methods: association rules, one of the most common approaches in web mining based recommender systems [2,11,17,15], and collaborative filtering which is commonly known as one of the most successful approaches for recommendations. We chose item-based collaborative filtering with probabilistic similarity measure [3], as a baseline for comparison because of the promising results it had shown. In Figure 5 you can see the performance of these systems in the sense of their accuracy and shortcut gain in different coverage values.

At lower coverage values we can see although our system still has superior results especially over association rules, accuracy and shortcut gain values are rather close. As the coverage increases, naturally accuracy decreases in all systems, but our system gains much better results than the other two systems. It can be seen the rate in which accuracy decreases in our system is lower than other two systems; at lower coverage values where the systems made their most promising recommendations (those with higher values), pages recommended were mostly the next immediate page and as can be seen had an acceptable accuracy. At lower coverage rates, where recommendations with lower values had to be made our system began recommending pages occurring in the session some steps ahead, while the other approaches also achieved greater shortcut gains, as the results show their lower valued recommendations were not as accurate and their performance declined more intensely.

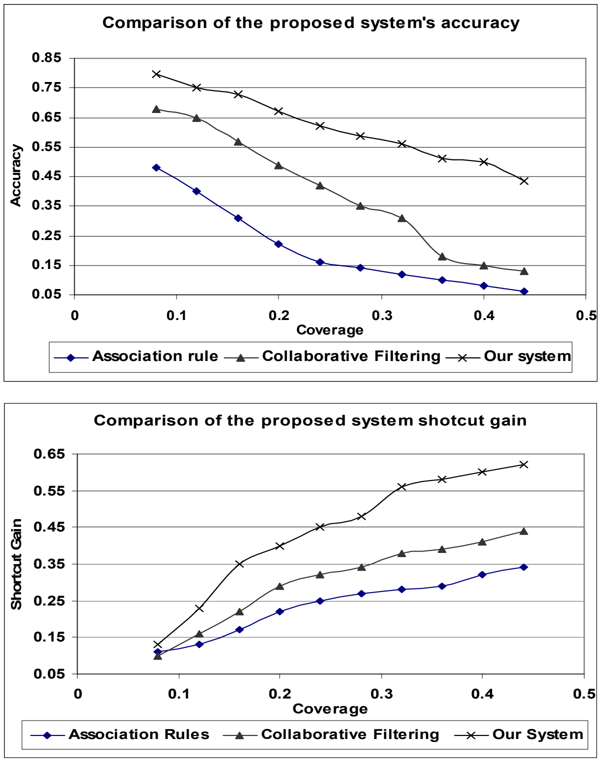

Figure 5: Comparing our system's performance with two other common methods

5. CONCLUSION

We presented a new web page recommendation system based on reinforcement learning in this paper. This system learns to make recommendations from web usage data as the actions it performs rather than discovering explicit patterns from the data and inherits the intrinsic characteristic of reinforcement learning which is being in a constant learning process. We modeled web page recommendation as a Q-Learning problem and trained the system with common web usage logs. System performance was evaluated under different settings and in comparison with other methods. Our experiments showed promising results achieved by exploiting reinforcement learning in web recommendation based on web usage logs. However, there are other alternatives that can potentially improve the system and constitute our future work. Regarding our evaluation, we tend to use an on-line evaluation of users instead of the off-line method based on web usage logs. In the case of the reward function used, various implicit feedbacks from the user rather than just the fact that the user had visited the page can be used, such as those proposed in [22]. Another option is using a more complicated reward function rather than the linear combination of factors; a learning structure such as neural networks is an alternative. Finally, in a broader sense, instead of making recommendations merely on web usage, it can be beneficial to take into account evidence from other sources of information, such as web content and structure, as has been the trend in recent web recommendation systems.

6. REFERENCES

[1] Burke, R. Hybrid recommender systems: Survey and experiments. User Modeling and User-Adapted Interaction, 2002.

[2] Cooley, R., Tan, P. N., Srivastava, J. WebSIFT: The Web Site Information Filter System. In Proceedings of the Web Usage Analysis and User Profiling Workshop (WEBKDD'99), 1999.

[3] Deshpande, M., Karypis, G. Item-based top-N recommendation algorithms. ACM Transactions on Information Systems (TOIS), 2004.

[4] Fu, X., Budzik, J., Hammond, K. J. Mining navigation history for recommendation. Intelligent User Interfaces, 2000.

[5] Golovin, N., Rahm, E. Reinforcement Learning Architecture for Web Recommendations. Proceedings of the ITCC2004. IEEE, 2004.

[6] Herlocker, J., Konstan, J., Brochers, A., Riedel, J. An Algorithmic Framework for Performing Collaborative Filtering. Proceedings of 200 Conference on Research and development in Information Retrieval, 2000.

[7] Joachims, T., Freitag, D., Mitchell, T. M. WebWatcher: A tour guide for the world wide web. Proceedings of International Joint Conference on Artificial Intelligence, 1997.

[8] Konstan, J., Riedl, J., Borchers, A., Herlocker, J. Recommender Systems: A GroupLens Perspective. In: Recommender Systems: Papers from the 1998 Workshop (AAAI Technical Report WS-98-08), 1998.

[9] Li, J., Zaiane, O. R. Combining Usage, Content and Structure Data to Improve Web Site Recommendation, 5th International Conference on Electronic Commerce and Web, 2004

[10] Liu, J., Zhang, S., Yang, J. Characterizing Web usage regularities with information foraging agents. IEEE Transactions on Knowledge and Data Engineering, 16(5), 566-584. 2004.

[11] Mobasher, B., Cooley, R., Srivastava, J. Automatic Personalization based on Web Usage Mining. Communications of the ACM. 43 (8), pp. 142-151, 2000.

[12] Mobasher, B., Dai, H., Luo, T., Sun, Y., Zhu, J. Integrating web usage and content mining for more effective personalization. In EC-Web, pages 165–176, 2000.

[13] Nakagawa M., Mobasher, B. A Hybrid Web Personalization Model Based on Site Connectivity. Proc. 5th WEBKDD workshop, 2003.

[14] Resnick, P., Varian, H.R. Recommender Systems. Communications of the ACM, 40 (3), 56-58, 1997.

[15] Shahabi, C., M. Zarkesh, A., Abidi, J., Shah, V. Knowledge discovery from user's Web-page navigation. In Proceedings of the 7th IEEE Intl. Workshop on Research Issues in Data Engineering, 1997.

[16] Shany, G., Heckerman, D., Barfman, R. An MDP-Based Recommender System. Journal of Machine Learning Research, 2005.

[17] Srivastava, J., Cooley, R., Deshpande, M., Tan, P.N. Web Usage Mining: Discovery and Applications of Usage Patterns from Web Data. SIGKDD Explorations, 1(2):12–23, 2000.

[18] Srivihok, A., Sukonmanee, V. E-commerce intelligent agent: personalization travel support agent using Q Learning. ACM International Conference Proceeding Series; Proceedings of the 7th international conference on Electronic commerce, 2005

[19] Su, Z., Yang, Q., Lu, Y., Zhang, H. What next: A prediction System for Web Requests Using N-gram Sequence Models. In Proceedings of the First International Conference on Web Information Systems and Engineering Conference.2000.

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

[21] Wasfi, A. M. Collecting User Access Patterns for Building User Profiles and Collaborative Filtering. In: IUI '99: Proceedings of the 1999 International Conference on Intelligent User Interfaces. 1999.

[22] Zhang, B., Seo, Y. Personalized web-document filtering using reinforcement learning. Applied Artificial Intelligence, 15(7):665-685, 2001.