Improving User Profiles for E-Commerce by Genetic Algorithms*

Yi-Shin Chen and Cyrus Shahabi

Integrated Media Systems Center and Computer Science Department
University of Southern California, Los Angeles, CA 90089-2564

E-mail:[*shahabi*, *yishinc*]@*usc.edu*

Abstract. Recommendation systems are widely adopted in e-commerce businesses for helping customers locate products they would like to purchase. The major challenge for these systems is bridging the gap between the physical characteristics of data with the users' perceptions. In order to address this challenge, employing user profiles to improve accuracy becomes essential. However, the system performance may degrade due to inaccuracy of user profiles. Therefore, an effective system should offer learning mechanisms to correct erroneous user inputs. In this paper, we extend an existing recommendation system, Yoda, to improve the profiles automatically by utilizing users' relevance feedback with genetic algorithms (GA). Our experimental results indicate that the retrieval accuracy is significantly increased by using the GA-based learning mechanism.

Keywords: e-commerce, recommendation systems, fuzzy logic, soft query, clustering, genetic algorithm, relevance feedback

1 Introduction

As the amount of available products in e-commerce businesses is burgeoning, searching for desired products among enormous offerings is becoming increasingly difficult. As a result, e-commerce users frequently suffer from information overload. To alleviate this problem, recommendation systems are being widely adopted to help customers locate products they would like to purchase. In essence, these systems apply data analysis techniques to provide a list of recommended products for each online customer. The most famous example in e-commerce businesses is the "*Customers who bought*" feature used in Amazon.com™, which is basically applied to every product page on its websites. With the help of this feature, Amazon.com™'s system recommends similar products to the current buyer based on the purchase histories of previous customers who bought the same product.

* This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC) and ITR-0082826, NIH-NLM R01-LM07061, DARPA and USAF under agreement nr. F30602-99-1-0524, and unrestricted cash/equipment gifts from NCR, IBM, Intel and SUN.

Systems such as Amazon.com™ employ filtering techniques which fall into two classes: *content-based filtering* and *collaborative filtering*. Both types of systems have inherent strengths and weaknesses, where content-based approaches directly exploit the product information, and the collaboration filtering approaches utilize specific user rating information.

The content-based filtering approach generates recommendation lists based on comparisons between the feature vectors of products (e.g., artist, style) in the database with those in the user's profile. Hence, the accuracy of the user's profile is very important. To keep the user profile accurate, various learning techniques, such as *Bayesian classifiers*, *neural networks*, and *genetic algorithms* (GA), are utilized for revising user profiles [14–16,2].

Despite these improvements, this approach has several other weaknesses. One is *content limitation*, i.e., lexical fragment methods can only be applied to text content. The other is *over-specialization*, i.e., users can only obtain the information indicated in their profiles and have no chance of exploring new information they might desire. Moreover, because of the complexity of user profiles, the learning processes are always computationally costly and unable to adapt to frequently changing user preferences.

On the other hand, the collaborative filtering (CF) approach, does not use any information regarding the actual content of the products. The approach is based on the assumption that people having similar interests will possibly like the same objects. Typically, CF-based recommendation systems utilize users' rating of products to generate recommendation lists. Therefore, the over-specialization problem is avoided since a user could explore new items listed in other users' profiles.

The nearest-neighbor algorithm is the earliest CF-based technique used in recommendation systems [12]. With this algorithm, the similarity between users is evaluated based on their ratings of products, and the recommendation is generated considering the items visited by nearest neighbors of the user. In its original form, CF-based recommendations suffer from the problems of, **scalability**, **sparsity**, and **synonymy** (i.e., latent association between items is not considered for recommendations.)

In order to alleviate or even eliminate these problems, more recently, researchers introduced a variety of different techniques into collaborative filtering systems, such as *content analysis* [11] for avoiding the synonymy and sparsity problems; *categorization* [13] to alleviate the synonymy and sparsity problems; *Bayesian network* [9,8] for lightening the scalability problems; *clustering* [9] to lessen sparsity and scalability problems; and Singular Value Decomposition (SVD) [10,7] to ease all three problems. However, all these techniques have limitation and do not work well in all general cases.

In an earlier work [1], we introduced a hybrid recommendation system - *Yoda*, which simultaneously utilizes the advantages of *clustering*, *content analysis*, and *collaborate filtering* (CF) approaches. Basically, Yoda is a two-step approach recommendation system. Basically, during the offline process,

Yoda maintains numerous recommendation lists obtained from human experts, clusters of user evaluations, and web navigation patterns analyzed by clustering and content analysis techniques. During the online process, the confidence values of an active user to the experts are estimated and kept in the user profile. By utilizing the user profile and experts' recommendations, Yoda finally generates customized recommendations for the user. As a result, we reduce the time complexity through model based and clustering approaches, alleviate the synonymy problem with content analysis, and address the sparsity problem by implicit identification of the users interests [5,6].

Since Yoda relies on user profiles to recommend products, the accuracy of recommendation results will decline if the user profiles are inaccurate. In practice, obtaining user profiles by explicit acquisitions has been challenging. We utilized the users' relevance feedback and improved the profiles automatically using GA [4]. To the best of our knowledge, only a few studies [3,2] incorporate GA for improving the user profiles. In these studies, users are directly involved in the evolution processes. Because users have to enter data for each product inquiry, they are often frustrated with this method. On the contrary, in our design, users are not required to offer additional data to improve the confidence values. These confidence values are corrected by the GA-based learning mechanisms using users' future navigation behaviors. That is, Yoda assumes positive feedback from a user when the user actually navigates through Yoda's recommended items. Our experimental results indicate a significant increase in the accuracy of recommendation results due to the integration of the proposed learning mechanism.

The remainder of this paper is organized as follows. Section 2 describes the concept of genetic algorithms. In Section 3, we provide an overview on the functionality of Yoda. In Section 4, we discuss the detailed design of Yoda and the learning mechanism. The results of our evaluations as well as the details of the system implementation and our benchmarking method are described in Section 5. Section 6 concludes the paper.

2 Genetic Algorithms

Genetic algorithms (GAs), which were introduced by Holland [4], are iterative search techniques based on the spirit of natural evolution. By emulating biological selection and reproduction, GAs can efficiently search through the solution space of complex problems. Basically, a GA operates on a population of candidate solutions called *chromosomes*. A chromosome, which is composed of numerous genes, represents an encoding of the problem and associates it with a fitness value evaluated by the fitness function. This fitness value determines the goodness and the survival ability of the chromosome.

Generally, GA starts by initializing the population and evaluating its corresponding fitness values. Before it terminates, GA produces newer generations iteratively. At each generation, a portion of the chromosomes is selected

according to the survival ability for reproducing offspring. The offspring are generated through crossover and mutation processes and are used for replacing some chromosomes in the population with a probability consistent with their fitness values. In other words, with the help of the fitness function to point out the correct direction, GA could construct better and better chromosomes from the best partial genes of past samplings. Please see reference [17] for mathematical foundations.

In summary, GA is composed of a fitness function, a population of chromosomes and three operators - selection, crossover and mutation. The parameter settings of the operators can be chosen depending on the applications or remain unchanged even when the applications are varied. However, the fitness function and the coding method are required to be specially designed for each problem. The design of fitness function and encoding method for Yoda will be described in Section 4.3.

3 Overview

The primary objective of a web-based recommendation system can be stated as follows:

Problem 1. Suppose the *item-set* $I = \{i | i$ is an item presented in a web-site $\}$ and u is a user interactively navigating the Web-site. The recommendation problem is to obtain the *wish-list* $I_u \in I$, which is a list of items that are ranked based on u's interests.

In general, to acquire a wish-list for a user, a recommendation process goes through three steps/phases:

1. Obtaining User Perceptions: Data about user perceptions such as navigation behaviors are collected. In some systems [8,7], these data need further processing for abstracting data which are used in the later phases.
2. Ranking the Items: The predicted user interests are utilized to provide the predicted user wish-list.
3. Adjusting user settings: The system acquires relevance feedback (or follow-up navigation behaviors) from the user and employs it to refine the user settings/profiles, which represent the user perceptions. On occasion, this phase is integrated into phase one.

Figure 1 illustrates the processing flow of Yoda. Suppose that music CDs are the items presented in a web-site. Yoda provides each active user a list of recommended CDs by analyzing his/her navigation behaviors. To generate the recommendations, during an offline process, Yoda obtains experts' recommendation (termed *experts' wish-lists*), which could be from human experts, clusters of user evaluations, or clusters of navigation patterns. Later, during the on-line recommendation process, the system first acquires the initial user profile, which is composed of a list of confidence values and a fuzzy

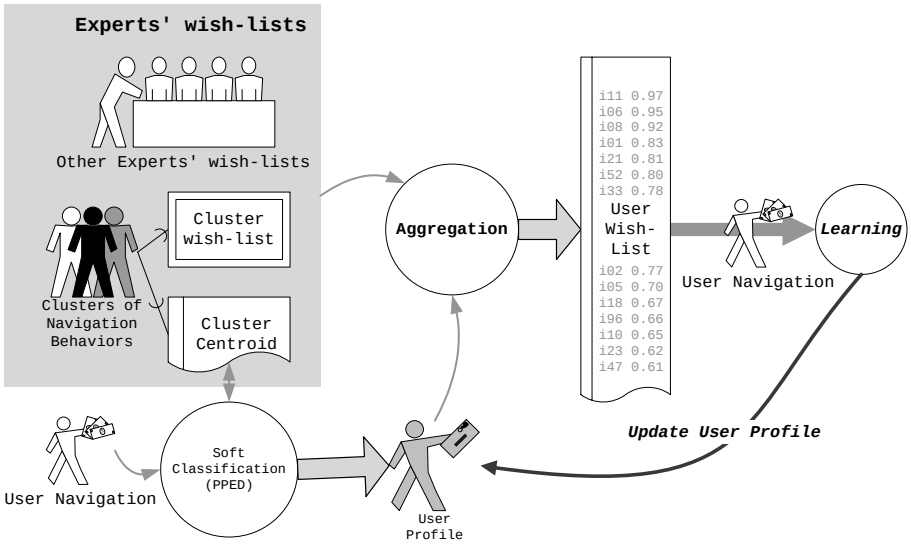


Fig. 1. Processing Flow of Yoda

cut, by softly classifying the user with clusters of navigation patterns. Note that the confidence values of the user toward other experts are not estimated at this step because the estimation process is comparably time consuming. Subsequently, Yoda uses the user profile to generate the customized recommendation (termed *user wish-list*) by weighted aggregation of the experts' wish-lists. Thereafter, the system improves and updates the user profiles as a background process by utilizing the follow-up user navigation behaviors and GA.

4 System Design

In this section, we provide a detailed description of Yoda's components. Since phase I is based on our previous work [18], here we elaborate more on phase II and III of Yoda.

4.1 Phase I - Obtaining User Perception

Yoda uses the client-side tracking mechanism proposed in [6] to capture view-time, hit-count, and sequence of visiting the web-pages (items) within a web-site. These features reflect users' interests on items. To analyze these features and infer the user interests, Yoda employs the *Feature Matrices (FM)* model, which we introduced in [18]. FM is a set of hyper-cube data structures that

can represent various aggregated access features with any required precision. With FM, the patterns of both a single user and a cluster of users are modeled.

Here, Yoda uses FM to model the navigation patterns of the active users individually, and then the aggregated navigation pattern of each cluster is generated by clustering a collection of user navigation behaviors. Yoda also applies a similarity measure, termed *Projected Pure Euclidean Distance (PPED)* [18], to evaluate the similarity of a user navigation to a cluster navigation pattern.

Thus, Yoda quantifies the confidence value a user to each navigation-pattern cluster and initializes the corresponding user profile, which contains a list of confidence values to experts and a user perception standard (fuzzy cut). However, because the PPED method can only apply to FM model, Yoda cannot acquire the confidence values of a user's interests to the recommendation lists of other experts at this step.

4.2 Phase II -Ranking the Items

Two types of work in Yoda involve ranking the items. The first type is generating the experts' recommendations, which are lists of ranked items produced by either human experts, clusters of users, or clusters of navigation patterns. In our previous work [1], a content analysis technique was proposed to abstract common interests from navigation patterns. With this technique, the system can generate a list of ranked items for each navigation-pattern cluster. However, for the sake of simplicity, we briefly describe this technique and only focus on another type of ranking work - generating the user wish-list online. In order to properly describe this method, we first formally define some necessary terms.

Definition 1. An *item* is an instance of product, service, etc. that is presented in a web-site. Items are described by their *properties*, which are abstract perceptual features.

$$i = \{(p, \tilde{p}_i) \mid p \text{ is a property} \in P, \tilde{p}_i \text{ is a fuzzy set} \in F\} \in I \tag{1}$$

For example, for a music CD as an item, "styles" of the music, "ratings", and "popularity" can be considered as properties of the item. Since properties are perceptual we use fuzzy-sets to evaluate properties. ∎

Definition 2. A wish-list, I_x, for user/expert x is defined as:

$$I_x = \{(i, v_x(i)) | i \text{ is an item}, v_x(i) \in [0, 1]\} \tag{2}$$

where the *preference* value $v_x(i)$ measures the probability of item i be of interest to user/expert x. ∎

Definition 3. A cluster browse-list, B_k, for navigation-pattern cluster k is a list of items visited by all users in this cluster. ∎

A user profile is composed of two parts: user confidence data and user fuzzy cut value. The formal definition of user confidence data is as follows:

Definition 4. E denotes a set of experts in the system. U represents the set of users who have assigned reference confidence values to experts. π is a confidence value for a user u to an expert e; $\pi : o \in O,\ e \in E \rightarrow b_{o,e}$. Note that the value of $b_{o,e}$ is a form of human judgment and is represented as a fuzzy term.

Generating Navigation-Pattern Cluster Wish-lists Yoda represents the aggregated interests of the users in each cluster by a set of property values (PVs), termed *favorite PVs* of the cluster. The favorite PV, $F_p(k)$, identifies likelihood of the cluster k being interested in property p of the items and is extracted by applying a *voting procedure* to the *browse-list* of the cluster as follows:

$$\begin{aligned} C_{p,f}(k) &= \|\{i \mid i \in B_k, \tilde{p}_i = f\}\| \\ F_p(k) &= \max\{f \mid f \in F, C_{p,f}(k) = \max_{\forall f' \in F}\{C_{p,f'}(k)\}\} \end{aligned} \tag{3}$$

Example 1. Suppose the browse-list of cluster K is {A, B, G, K, Y, Z}, and the values of property "Rock" for the corresponding CDs are { (A, high), (B, high), (G, low), (K, medium), (Y, high), (Z, high) }. Because "high" has the maximum vote, the favorite PV of cluster K, $F_{Rock}(K)$, is "high".

Based on these extracted favorite PVs of the cluster k, Yoda can evaluate $v_k(i)$, preference value of an item i for cluster k, by quantifying the similarity between favorite PVs and property values associated with item i. The aggregation function used to compute $v_k(i)$ is:

$$\begin{aligned} G_f(k) &= \{p \mid f \in F,\ p \in P,\ F_p(k) = f\} \\ E_{k,f}(i) &= f \times \max\{\tilde{p}_i \mid p \in G_f(k)\} \\ v_k(i) &= \max\{E_{k,f}(i) \mid \forall f \in F\} \end{aligned} \tag{4}$$

Example 2. Suppose properties are grouped as $G_{medium}(K) = \{\text{Vocal, Soundtrack}\}$, $G_{high}(K) = \{\text{Rock, Pop}\}$, and $G_{low}(K) = \{\text{Classic}\}$, and the item i is defined as { (Rock, low), (Pop, low), (Vocal, low), (Soundtrack, high), (Classic, medium)}. According to the equations above, the preference value $v_K(i) = \max$ { (high $\times$ low), (medium $\times$ high), (low $\times$ low) } = (medium $\times$ high) = 0.75.

Generating User Wish-lists During the on-line recommendation process, Yoda aggregates the experts' wish-lists to generate the predicted user wish-list for the active user u. A fuzzy aggregation function is employed to measure

and quantify the preference value $v_u(i)$ of each item i for the user u based on the user profile of user u. We use an optimized aggregation function with a triangular norm [19]. A triangular norm aggregation function g satisfy the following properties:

$$\begin{aligned} \text{Monotonicity: } & g(x,y) \leq g(x',y') \text{ if } x \leq x' \text{ and } y \leq y' \\ \text{Commuatativity: } & g(x,y) = g(y,x) \\ \text{Associativity: } & g(g(x,y),z) = g(x,g(y,z)) \end{aligned}$$

With these properties, the query optimizer can replace the original query with a logically equivalent one and still obtain the exact same result. The optimized aggregation function we propose for Yoda is:

Definition 5. First, experts are grouped based on their reference confidence values assigned by user u.

$$G_f(u) = \{e \mid f \text{ is a fuzzy set} \in F,\ \pi_{u,e} = f\} \tag{5}$$

Then, the preference value $v_u(i)$ for item i is computed as:

$$\begin{aligned} E_{u,f}(i) &= f \times \max\{v_e(i) \mid e \in G_f(u)\} \\ v_u(i) &= \max\{E_{u,f}(i) \mid \forall f \in F\} \end{aligned} \tag{6}$$

∎

Basically, this aggregation function partitions the preference values into $\|F\|$ different subgroups according to the confidence values of the expert e. Subsequently, the system maintains a list of maximum preference values for all subgroups. Finally, the system computes the preferences of all items in the user wish-list by iterating through all subgroups. As compared to a naive weighted aggregation function with time complexity $O(\|E\| \times \|I\|)$ (where $\|E\|$ is the number of experts in the system) the complexity of the proposed aggregation function is $O(\|F\| \times \|I\|) = O(\|I\|)$, where $\|F\|$ is a small constant number representing the number of fuzzy terms.

To reduce the time complexity of generating the user wish-lists further, we apply a cut-off point on the expert wish-lists. Each shorten wish-list includes the N best-ranked items according to their preference values for the corresponding expert. In [19], Fagin has proposed an optimized algorithm, the A_0 algorithm, to retrieve N best items from a collection of subsets of items with time complexity proportional to N rather than total number of items. Here, by taking the subgroups of items (as described above) as the subsets, the A_0 algorithm can be incorporated into Yoda[1]. Applying the A_0 algorithm to generate a user wish-list with cut-off point N, we reduce the time complexity to $O(\|F\| \times \|N\|) = O(\|N\|)$, where $\|N\| \ll \|I\|$.

[1] Since our aggregation function is in triangular norm form, it satisfies the requirements of the A_0 algorithm.

4.3 Phase III -Adjusting User Settings

This learning mechanism is a background process performed at the same time that the users navigate the web-site. It employs GA for improving the list of confidence values by decoding the best chromosome to replace existing one in the system after its evolution. Users are not required to make additional effort to improve these confidence values. Yoda collects users' follow-up navigation behaviors and employs these behaviors as the goal of GA prior to the beginning of evolution[2]. Note that the learning mechanism is only triggered after receiving enough navigation data. In our implementation, it is activated when the number of navigated items is the same as that of the recommended items in the wish-list.

We first describe the method for transforming the navigation data to the relevance feedback needed in GA. Let FN be the set of follow-up navigated items and i be an item in FN. As described in Section 4.1, Yoda can capture the view-time, the hit-count and sequence about the items. Therefore, FN can be formally defined as:

$$\begin{aligned} FN = \{(i, \upsilon_t(i), \upsilon_s(i), \upsilon_h(i)) | i :\in I, \upsilon_t(i) : \text{view-time of } i, \\ \upsilon_s(i) : \text{sequence of } i \text{ in reverse order}, \upsilon_h(i) : \text{hit-count of } i\} \end{aligned} \tag{7}$$

Assuming that users only navigate potentially desired items, the preferences of items can be estimated from navigation behaviors. That is, the users are more interested in the items that are navigated earlier, accesses more often, or viewed for longer periods of time. As a result, the feedback preference $\bar{v}_u(i)$ of the navigated item i from user u's perspective could be estimated based on the navigation data by using Equation (9).

$$\begin{aligned} \omega_f &: \text{the importance weight of feature } f \\ \mu_f &: \text{the mean of navigation data in feature } f \\ \sigma_f &: \text{the standard deviation of navigation data in feature } f \\ \psi_f &= \begin{cases} \frac{\upsilon_f(i)}{\mu_f + 3 \times \sigma_f} & \text{if } f(i) \leq (\mu_f + 3 \times \sigma_f) \\ 1 & \text{if } \upsilon_f(i) > (\mu_f + 3 \times \sigma_f) \end{cases} \qquad (8) \\ \bar{v}_u(i) &= (\omega_s \times \psi_s) \times (\omega_h \times \psi_h) \times (\omega_t \times \psi_t) \qquad (9) \end{aligned}$$

Note that for the normalization purpose, we use $(\mu_f + 3 \times \sigma_f)$ as the upper bound in Equation (8). This upper bound can prevent the affect from outliers [21] of the navigation data.

[2] Note that GA would converge per evolution process and there is no guarantee that it converges across several evolutions if the user's navigation behavior is inconsistent. However, in general, no learning mechanism can deal with inconsistent behaviors

Subsequently, we explain the coding design for GAs in our learning mechanism. The chromosomes represent a possible user profile. Two types of records are involved in the genes. One is user confidence information with k records, where k is the number of experts in the system. The value of the ith gene is an integer in $[0, L-1]$, where L is the number of fuzzy terms used in the system, and denotes the user's confidence level to expert i. The other is a user fuzzy cut value which is associated with the $(k+1)$th gene. The value of fuzzy cut is $(t+1)/L$, where $t \in [0, L-1]$ is the value of this gene.

For example, suppose that there are 50 experts and 8 different fuzzy terms in the system, there will be 51 genes per chromosome where the first 50 genes represent the corresponding confidence values to the experts, and the last gene represents the value of the user fuzzy cut. Additionally, after decoding, the value of 0 in gene i indicates that the confidence level to user i is "none" and the value of 6 in gene 51 indicates that the value of fuzzy cut is $(6+1)/8 = 0.875$. Likewise, after encoding, "full" confidence level to user i is represented by a number 7 in gene i and the 0.75 fuzzy cut is denoted by a number 5 in gene 51.

This coding method can guarantee a one-to-one mapping of profiles to chromosomes. That is, a chromosome will be decoded to one and only one legal user profile, and a user profile will be encoded to one and only one chromosome. Consequently, the solution space will be equal to the searching space in GA. This implies that our coding method is effective.

Next, we describe our GA fitness function, which heavily utilizes the preference list B estimated by our converting method. The fitness function first decodes the chromosome into a confidence list and a fuzzy cut value. Then, it obtains the user wish-list Q according to the profile using Equation (6). In other words, this process needs to interact with the system for obtaining experts' wish-lists. Finally, it generates the fitness value by measuring the similarity between Q and B. The similarity values are computed by Equation (12) which is based on two measurements. Equation (10) evaluates the similarity on ranking, and Equation (11) measures the average satisfaction of the user wish-list.

$$Q = \{(i, v_u(i)) | v_u(i)) \in [0,1]\}$$

$$B = \{(i, \bar{v}_u(i)) | \bar{v}_u(i) \in [0,1]\}$$

$$\text{Cos } \theta(Q,B) = \frac{\sum_i v_u(i) \times \bar{v}_u(i)}{\sqrt{\sum_i v_u(i)^2 \times \sum \bar{v}_u(i)^2}} \tag{10}$$

$$Avg(Q,B) = \frac{\sum_{i \in Q} \bar{v}_u(i)}{\|Q\|} \tag{11}$$

$$\text{Similarity}(Q,B) = \text{Cos } \theta(Q,B) + 3 \times Avg(B,Q) \tag{12}$$

In summary, once the learning process is triggered, the learning mechanism first converts the navigation behaviors to relevance feedback. Next, it

encodes the corresponding confidence list to a chromosome and randomly generates other chromosomes as the initial population. Subsequently, GA iteratively discover better user profiles until it achieves the terminal condition such as the fitness value of one chromosome being 1 or the generation number being 200. In the end, the learning mechanism decodes the best chromosome to a confidence list and a fuzzy cut value for replacing the current data.

5 Performance Evaluation

In this section, we first describe our experimental setup and benchmarking method in Section 5.1. Subsequently, the details of our experimental results are discussed in Section 5.2.

5.1 Experimental Methodology

We developed a GA for Yoda using SUGAL [20] for its wide range of operators and data types. Yoda is implemented in C and on top of Microsoft Access 2000, which is running on a Pentium II 233MHZ processor with Microsoft NT4.0.

Parameter	Definition
M	Number of items in the web-site
N	The cut-off point (number of items in a user wish-list)
F	Number of fuzzy terms
E	Number of experts
O	Number of known experts ($E \geq O$)
K	Number of navigation-pattern clusters ($O \geq K$)
P	Percentage of interesting items within the item set
U	Number of users

Table 1. Benchmarking Parameters

In order to populate data for evaluation purposes, we propose a parametric algorithm to simulate various benchmarks (see Table 5.1). The benchmarking method maintains a list of preference values (denoted as $\mathring{A}_0$) that contains the perfect knowledge about items of interest to the active users. This is performed as follows. First, the algorithm randomly generates E experts and populates the preference information to an expert matrix $\bar{E}_0$. The cell $\bar{e}(i,j) \in \bar{E}_0$ represents the preference values for item j by expert i. Note that among these experts, there are K navigation-pattern clusters. Each cluster comprises a browse-list, a list of favorite PVs, and a pattern of navigation as the cluster centroid.

Next, the system randomly generates a list of confidence values $\ddot{\pi}$ and a fuzzy cut value for each active user. Each confidence value is represented by a fuzzy term, which is an integer in the range of $[0, 7]$. Finally, the system populates the preference values to $\mathring{A}_0$ by aggregating $\ddot{\pi}$ and $\bar{E}_0$ using Equation (6).

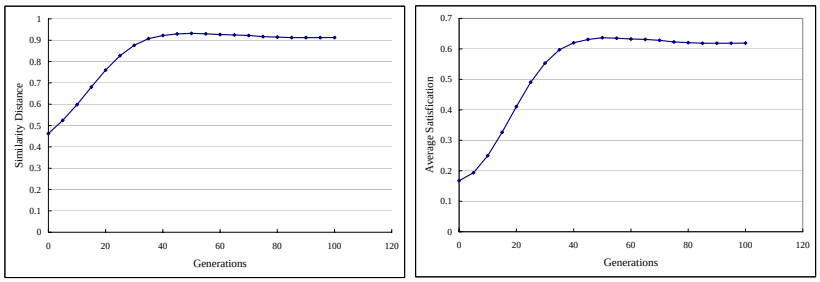
To simulate imperfect user feedback during the GA, all perfect knowledge will be tuned in a noisy process. Based on the preference values in this imperfect knowledge, the system then randomly selects a set of items and assigned feature values as the user navigation behaviors. These feature values are generated by a reverse procedure of Equation (11) . For example, the higher the preference value of an item is, the longer periods of view time is assigned to this item. Finally, the system reinitializes the user profile by assigning K imperfect confidence values of navigation-pattern clusters and a fuzzy cut value to the imperfect user profile.

5.2 Experimental Results

We conducted several sets of experiments to verify our system performances and to compare the results of different GA parameter settings. In these experiments, we observed a significant margin of improvement after incorporating the GA in matching the user expectations in various settings. It is also shown that the performance improvement of our learning mechanism is independent of the number of users. Moreover, the improvement is linearly increased with the number of items. However, due to the space limit, in this paper, we only stress the improvements achieved by applying our learning mechanism.

The results shown for each set of experiments are averaged over twenty runs, where each run is executed with different seeds for the random generator functions. The parameter settings of GA operators [20,17] are: population size=30, one-point crossover, tournament selection, keep the elitism, and mutation rate = 0.25. The benchmark settings of the following figures, i.e., the number of items, the number of cut-off point, the number of fuzzy terms, the number of experts, and the number of navigation-pattern clusters, are fixed at $M = 5000$, $N = 100$, $F = 8$, $E = 50$, $K = 10$, and $P = 5\%$, respectively.

Figure 2 demonstrates the improvements achieved by our learning mechanism. The X-axis of Figure 2 depicts the number of generations. The Y-axis of Figure 2.a illustrates the similarity distance (between query results and perfect user feedback) computed by Equation (10). The Y-axis of Figure 2.b represents average satisfaction computed by Equation (11), respectively. In this experiment, Figure 2 indicates that the accuracy of wish-lists improves drastically after incorporating the learning mechanism. As observed, the average satisfaction increases nearly 50% within ten generations of the evolution, and the similarity on ranking improves 100% after 40 generations of learning. Overall, in the majority of our experiments, GA can acquire nearly ideal user profiles, i.e, the rankings in the retrieval wish-list has 90% similarity to those in the perfect wish-list, within 40 generations. It should be noticed that Yoda



a. Similarity distance b. Average satisfaction

Fig. 2. The system improvement

only recommends the top 100 items for users and also only receives feedbacks about 100 items from users, where the set of feedback items and the set of recommended items are not identical. In other words, GA needs to search for ideal user profiles based on 2% of information. These results suggest that our learning mechanism is efficient and can greatly improve user profiles.

Since GA only has 2% information for learning, the limited information might restrict the learning ability. This problem was also observed in the experiments. In Figure 2, the accuracy of wish-lists slightly decreases after 50 generations of learning. The reason is that the learning mechanism is constrained by the limited feedbacks. To illustrate, consider the following example. Assume user U likes most of the items recommended by expert A. However, because the recommendation system only recommends few items and user U does not have the chance to know expert A's recommendations, the learning mechanism has no ability to discover the relationship between A and U. Generally, this problem can be alleviated by several evolutions.

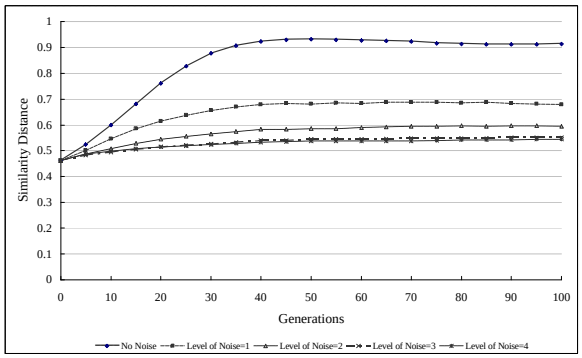


Fig. 3. Impact of noise

In order to compare the system performance when the user relevance feedbacks are imperfect, we introduce five different noise levels in the experiments

of Figure 3, where noise level 0 represents perfect feedback and noise level 10 represents complete chaos. The X-axis is the number of generations and the Y-axis depicts the similarity distance (between query results and perfect user feedback) computed by Equation (10). As revealed by Figure 3, although the noise levels affect the accuracy of results, our learning mechanism still improves the quality of user profiles in the range of 20% to 50%. This figure indicates that our learning mechanism has the ability to tolerate noises during the learning process.

6 Conclusion

We proposed a recommendation system to reduce the information overload problem in e-commerce. However, this system heavily relies on user profiles for providing accurate recommendation lists. The system accuracy may decline if user profiles are inaccurate. Therefore, we introduced a learning mechanism that utilizes the users' navigation behaviors to improve the profiles automatically using genetic algorithms. The experimental results indicated that the accuracy of results significantly increased up to 100% by our GA-based learning mechanism. It also demonstrated that our learning mechanism has the ability of tolerating noises during learning processes and improvement is in the range of 20% to 50% depending on the noise level.

References

1. Shahabi C., Banaei-Kashani F., Chen Y.-S., McLeod D., (2001) Yoda: An Accurate and Scalable Web-based Recommendation System. In Proceedings of Sixth International Conference on Cooperative Information Systems (CoopIS 2001)
2. Moukas A., (1996) Amalthea: Information discovery and filtering using a multiagent evolving ecosystem. In Proceedings of 1st Int. Conf. on The Practical Applications of Intelligent Agents and MultiAgent Technology (PAAM)
3. Sheth B., (1993) Evolving Agents for Personalized Information Filtering. In Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications
4. Holland J., (1975) Adaption in Natural and Artificial Systems. University of Michigan Press, Ann Arbor, Michigan
5. Konstan J., Miller B., Maltz D., Herlocker J., Gordon L., Riedl J., (1997) Applying Collaborative Filtering to Usenet News. In: Communications of the ACM (40) 3
6. Shahabi C., Zarkesh A.M., Adibi J., Shah V., (1997) Knowledge Discovery from Users Web Page Navigation. In Proceedings of the IEEE RIDE97 Workshop
7. Sarwar B., Karypis G., Konstan J., Riedl J. (2000) Application of Dimensionality Reduction in Recommender System – A Case Study. In Proceedings of ACM WebKDD 2000 Web Mining for e-Commerce Workshop
8. Kitts B., Freed D., Vrieze M., (2000)Cross-sell, a fast promotion-tunable customer-item recommendation method based on conditionally independent probabilities. In Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 437-446

9. Breese J., Heckerman D., Kadie C. (1998) Empirical Analysis of Predictive Algorithms for Collaborative Filtering. In Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, pp. 43-52
10. Sarwar B., Karypis G., Konstan J., Riedl J., (2000) Analysis of Recommendation Algorithms for e-Commerce. In Proceedings of ACM e-Commerce 2000 Conference, 2000.
11. Balabanovi M., Shoham Y., (1997) Fab, content-based, collaborative recommendation. In: Communications of the ACM, Vol 40(3), pp. 66-72
12. Resnick P., Iacovou N., Suchak M., Bergstrom P., Riedl J., (1994) GroupLens, An Open Architecture for Collaborative Filtering of Netnews. In Proceedings of ACM conference on Cumputer-Supported Cooperative Work, pp. 175-186
13. Good N., Schafer J., Konstan J., Borchers J., Sarwar B., Herlocker J., Riedl J., (1999) Combining Collaborative Filtering with Personal Agents for Better Recommendations. In Proceedings of the 1999 Conference of the American Association of Artifical Intelligence, pp. 439-446
14. Pazzani M., Billsus D., (1997) Learning and Revising User profiles: The Indentification of Interesting Web Sites. In: Machine Learning, Vol 27, pp. 313-331
15. Tan A., Teo C., (1998) Learning User Profiles for Personalized Information Dissemination. In Proceedings of Int'l Joint Conf. on Neural Network, pp. 183-188
16. Lam W., Mukhopadhyay S., Mostafa J., Palakal M., (1996) Detection of Shifts in User Interests for Personalized Information Filtering, In Proceedings of the 19th Int'l ACM-SIGIR Conf on Research and Development in Information Retrieval, pp. 317-325
17. Goldberg D.E., (1989) Genetic Algorithms in Search, Optimisation, and Machine Learning. Addison-Wesley, Wokingham, England
18. Shahabi C., Banaei-Kashani F., Faruque J., Faisal A., (2001) Feature Matrices: A Model for Efficient and Anonymous Web Usage Mining, In Proceedings of EC-Web 2001
19. Fagin R., (1996) Combining Fuzzy Information from Multiple Systems. In Proceedings of Fifteenth ACM Symposyum on Principles of Database Systems
20. Hunter A., (1995) Sugal Programming manual. http://www.trajan-software.demon.co.uk/sugal.htm
21. Knorr, E., R. Ng, and V. Tucakov., (2000) Distance-Based Outliers: Algorithms and Applications. The VLDB Journal, 8(3),pp. 237-253