Distributed and Parallel Databases, 14, 173–192, 2003

An Adaptive Recommendation System without Explicit Acquisition of User Relevance Feedback*

CYRUS SHAHABI shahabi@usc.edu
YI-SHIN CHEN yishinc@usc.edu
Integrated Media Systems Center and Computer Science Department, University of Southern California, Los Angeles, CA 90089-2564, USA

Recommended by: Boualem Benatallah

Abstract. Recommendation systems are widely adopted in e-commerce businesses for helping customers locate products they would like to purchase. In an earlier work, we introduced a recommendation system, termed *Yoda*, which employs a hybrid approach that combines collaborative filtering (CF) and content-based querying to achieve higher accuracy for large-scale Web-based applications. To reduce the complexity of the hybrid approach, Yoda is structured as a tunable model that is trained off-line and employed for real-time recommendation on-line. The on-line process benefits from an optimized aggregation function with low complexity that allows the real-time aggregation based on confidence values of an active user to pre-defined sets of recommendations. In this paper, we extend Yoda to include more recommendation sets. The recommendation sets can be obtained from different sources, such as human experts, web navigation patterns, and clusters of user evaluations. Moreover, the extended Yoda can learn the confidence values automatically by utilizing implicit users' relevance feedback through web navigations using genetic algorithms (GA). Our end-to-end experiments show while Yoda's complexity is low and remains constant as the number of users and/or items grow, its accuracy surpasses that of the basic nearest-neighbor method by a wide margin (in most cases more than 100%). The experimental results also indicate that the retrieval accuracy is significantly increased by using the GA-based learning mechanism.

Keywords: e-commerce, recommendation systems, 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 IIS-0082826, NIH-NLM R01-LM07061, DARPA and USAF under agreement nr. F30602-99-1-0524, and unrestricted cash gifts from Microsoft, NCR, and Okawa Foundations.

In an earlier work [24], 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. During the offline process, Yoda maintains numerous recommendation lists obtained from typical navigation patterns analyzed by clustering and content analysis techniques. During the online process, the confidence value of an active user to each navigation-pattern cluster is estimated using the PPED distance measure [25] by comparing the user's navigation pattern with centroid of each cluster. Finally, Yoda generates customized recommendations for the user by aggregating across recommendation lists (one list for each cluster) using the confidence values as the weights.

To expedite the aggregation step, we proposed an optimized fuzzy aggregation function that reduces the time complexity of aggregation from $O(N \times E)$ to $O(N)$, where N is the number of recommended items and E is the number of clusters. Besides the advantage of efficiency, our aggregation function (similar to FALCON [31]) can manage disjunctive queries while the traditional weighted average method cannot. For example, assume that the system knows all the information about users and user U is interested in the items recommended by expert *A or B*. The traditional weighted average method can only generate the list of items recommended by both expert *A and B* while our aggregation method can retrieve the expected list.

In sum, the time complexity is reduced through a model-based technique, a clustering approach, and the optimized aggregation method. Additionally, due to the utilization of content analysis techniques, Yoda can detect the latent association between items and therefore provides better recommendations. Moreover, Yoda is able to collect information about user interests from implicit web navigation behaviors while most other recommendation systems [3, 9, 20, 23, 28] do not have this ability and therefore require explicit rating information from users. Consequently, Yoda puts less overhead on the users.

Since content analysis techniques only capture certain characteristics of products, some desired products might not be included in the recommendation lists produced by analyzing the content. For example, picking wines based on brands, years, and descriptors might not be adequate if "smell" and "taste" are more important characteristics. In order to remedy for this problem, we extend Yoda to incorporate more recommendation lists than just web navigation patterns. These recommendation lists can be obtained from various experts, such as human experts and clusters of user evaluations.

Meanwhile, because PPED is specially designed for measuring the similarity between two web navigation patterns including related data such as browsed items, view time, and sequences information, it can only be used for estimating confidence values to navigation-pattern clusters. Therefore, a learning mechanism is needed for obtaining the complete confidence values of an active user toward all experts. We propose a learning mechanism that utilizes users' relevance feedback to improve confidence values automatically using genetic algorithms (GA) [7].

To the best of our knowledge, only a few studies [18, 29] 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. 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 summarizes the related work. In Section 3, we provide an overview on the functionality of Yoda. In Section 4, we discuss the detailed design of Yoda . Section 5 depicts the learning mechanism of Yoda. Section 6 describes the results of our evaluations as well as the details of the system implementation and our benchmarking method. Section 7 concludes the paper.

2. Related work

Recommendation systems are designed either based on *content-based filtering* or *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.

Content-based filtering approaches are derived from the concepts introduced by the Information Retrieval (IR) community. Content-based filtering systems are usually criticized for two weaknesses:

1. *Content limitation*: IR methods can only be applied to a few kinds of content, such as text and image, and the extracted features can only capture certain aspects of the content.
2. *Over-specialization*: Content-based recommendation system provides recommendations merely based on user profiles. Therefore, users have no chance of exploring new items that are not similar to those items included in their profiles.

The collaborative filtering (CF) approach remedies for these two problems. Typically, CF-based recommendation systems do not use the actual content of the items for recommendation. Moreover, since other user profiles are also considered, user can explore new items. The nearest-neighbor algorithm is the earliest CF-based technique used in recommendation systems [20, 28]. 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, the nearest-neighbor algorithm uses a two-dimensional user-item matrix to represent the user profiles. This original form of CF-based recommendation systems suffers from three problems:

1. *Scalability*: The time complexity of executing the nearest-neighbor algorithm grows linearly with the number of items and the number of users. Thus, the recommendation system cannot support large-scale applications such as Amazon.com™, which provides more than 18 million unique items for over 20 million users.
2. *Sparsity*: Due to large number of items and user reluctance to rate the items, usually the profile matrix is sparse. Therefore, the system cannot provide recommendations for some users, and the generated recommendations are not accurate.

3. *Synonymy*: Since contents of the items are completely ignored, latent association between items is not considered for recommendations. Thus, as long as new items are not rated, they are not recommended; hence, false negatives are introduced.

In order to solve these problems, a variety of different techniques have been proposed. Some of techniques, such as dimensionality reduction [22, 23], clustering [16], and Bayesian Network [3, 9], mainly are remedies for the scalability problem. These techniques extract characteristics (patterns) from the original dataset in an offline process and employ only these patterns to generate the recommendation lists in the online process. Although this approach can reduce the online processing cost, it often reduces the accuracy of the recommending results. Moreover, the online computation complexity keeps increasing with the number of patterns.

Some other techniques, such as association rules [17, 23], content analysis [1, 2, 15], categorization [6, 12], are emphasized on alleviating the sparsity and synonymy problems. Basically, these techniques analyze the Web usage data (from Web server logs) to capture the latent association between items. Subsequently, based on both item association information and user ratings, the recommendation systems can thus generate better recommendation to users. However, the online computation time concurrently increases, as more data are incorporated into the recommendation progress. Additionally, because Web usage data from the server side are not reliable [26], the item association generated from Web server logs might be wrong.

With Yoda [24], we introduced a hybrid recommendation model, which consists of two steps. During the offline process, Yoda generates cluster recommendation lists based on the Web usage data from the client-side through clustering and content analysis techniques. This approach not only can address the scalability problem by the preprocessing work, but also can alleviate the sparsity and synonymy problems by discovering latent association between items. Since the Web usage data from the client-side can capture real user navigation behaviors, the item association discovered by the Yoda system would be more accurate. Beside the cluster recommendation lists, Yoda also maintains numerous recommendation lists obtained from different experts, such as human experts of the Website domain, and the cluster representatives of the user ratings. By these additional recommendation lists, Yoda is less impacted by the preprocessing work as compared to other systems.

During the online process, for each user who is using the system, Yoda estimates his/her confidence values to each expert, who provides the recommendation list, based on his/her current navigation behaviors through the PPED distance measure [25] and our GA-based learning mechanism. Subsequently, Yoda generates customized recommendations for the user by aggregating across recommendation lists using the confidence value as the weight. In order to expedite the aggregation step, Yoda employs an optimized fuzzy aggregation function that reduces the time computation complexity of aggregation from $O(N \times E)$ to $O(N)$, where N is the number of recommended items in the final recommendation list to users and E is the number of recommendation lists maintained in the system. Consequently, the online computation complexity of Yoda remains the same even if number of recommendation lists increases.

3. Overview

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

Problem 1. Suppose the *item-set* $I = \{i \mid i$ 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 u's *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 [9, 22], these data need further processing for inferring information which is used in the later phases.
2. *Ranking the items*: The inferred 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, 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 recommending items in the Yoda web-site. The objective of Yoda is to provide each active user, who is using the system, a satisfactory-and-customized recommendation list of CDs by

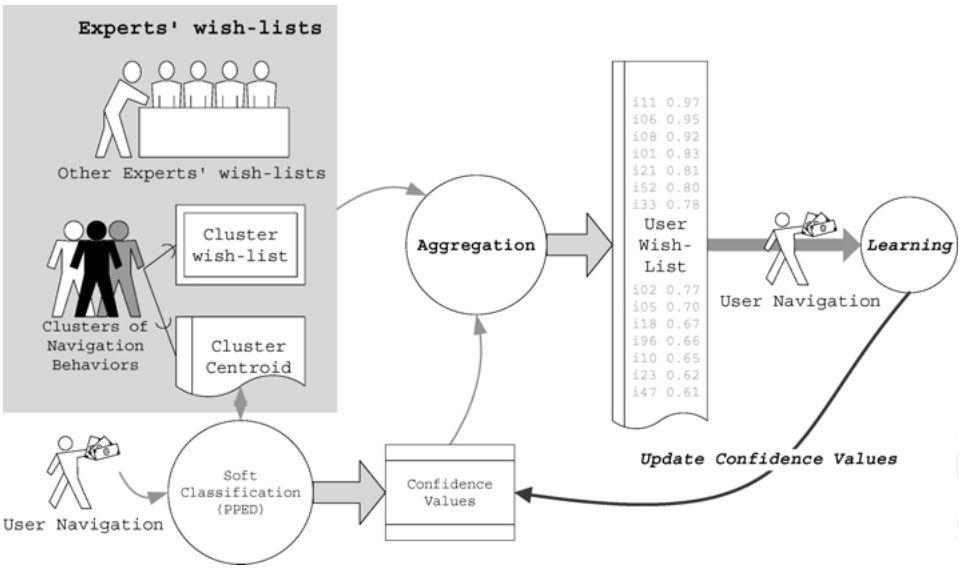


Figure 1. Processing flow of Yoda.

analyzing his/her navigation behaviors. To accomplish this goal, Yoda performs a two-step process. During the offline process, Yoda clusters the collected Web usage data from the browser side and generates the corresponding cluster recommendation list (termed *experts' wish-lists*) and the centroid for each navigation-pattern cluster. Moreover, Yoda also maintains other experts' wish-lists obtained from different experts, such as human experts, and the cluster representatives of user ratings.

Later, during the on-line recommendation process, the system first acquires the initial confidence values by *softly classifying*[1] the active user into navigation-pattern clusters. Note that the confidence values between the user and other experts are not estimated at this step because the estimation process is comparably time consuming. Subsequently, Yoda uses these confidence values to generate the customized recommendation (termed *user wish-list*) by weighted aggregation of the experts' wish-lists. After the user receives the user wish-list and continue to navigate the website, Yoda will correct the confidence values in a background process by utilizing the follow-up user navigation behaviors and the GA-based learning mechanism.

4. System design

In this section, we provide a detailed description of Yoda's components. Since phase I is based on our previous work [25], here we elaborate more on phase II and III of Yoda. The phase III is described in Section 5.

4.1. Phase I—Obtaining user perception

Yoda uses the client-side tracking mechanism proposed in [27] 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 [25]. 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*) [25], to evaluate the similarity of a user navigation to a cluster navigation pattern.

Thus, Yoda can quantify the confidence value of a user to each navigation-pattern cluster. However, because PPED can only apply to the navigation pattern behaviors modeled by 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 [24], 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 4.1. An *item* i 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, and the corresponding degrees of the properties.

$$i = \{(p, \tilde{p}_i) \mid p \text{ is a property} \in P, \tilde{p}_i \text{ is the degree of } i \text{ to } p\} \in I \tag{1}$$

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 most of properties are perceptual and asking for precise values to rate these perceptual properties is inappropriate, we use limited fuzzy-sets ($f \in F$) to evaluate proprieties. By this design, we can also ease the difficulties of the content analysis in the later process.

Definition 4.2. A wish-list, I_x, for user/expert x is defined as:

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

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

Definition 4.3. A cluster browse-list, B_k, for navigation-pattern cluster k is a list of items visited by all users in this cluster.

Definition 4.4. A confidence value π for a user u to an expert e is formally defined as: $\pi : u, e \in E \rightarrow b_{u,e}$, where E denotes a set of experts in the system and U represents the set of users who have assigned reference confidence values to experts. Note that the value of $b_{u,e}$ is represented as a fuzzy term for two reasons. First, using fuzzy terms to describe forms of human judgment is more appropriate. Second, by the limited set of fuzzy terms, the online computation process can be expedited.

4.2.1. 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. These favorite PVs indicate the emphasized degree of the properties by the majority of users in this cluster. In order to extract these values from the navigation data in this cluster, we design a voting procedure defined as:

Definition 4.5. The favorite PV, $F_p(k)$, identifies likelihood of the cluster k being interested in property p of the items and is extracted from the cluster *browse-list* B_k through

Eq. (4).

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

Example 4.1. 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:

$$v_k(i) = \max\{F_p(k) \times \tilde{p}_i\} \tag{4}$$

Example 4.2. Suppose the favorite PV of cluster K is (Rock, high), (Pop, high), (Vocal, medium), (Soundtrack, medium), (Classic, low), and the item i is defined as {(Rock, low), (Pop, low), (Vocal, low), (Soundtrack, high), (Classic, low)}. According to the equations above, the preference value $v_K(i) = \max$ {(high $\times$ low), (high $\times$ low), (medium $\times$ low), (medium $\times$ high), (low $\times$ low)} = (medium $\times$ high) = 0.75.

4.2.2. 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 [4]. A triangular norm aggregation function g satisfy the following properties:

Monotonicity: $g(x, y) \leq g(x', y')$ if $x \leq x'$ and $y \leq y'$

Commuatativity: $g(x, y) = g(y, x)$

Associativity: $g(g(x, y), z) = g(x, g(y, z))$

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 4.6. 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 Fagin [4], 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. However, the A_0 algorithm can only be employed by the aggregation function that satisfy triangular norm form.

Here, by taking the subgroups of items (as described above) as the subsets, the A_0 algorithm can be incorporated into Yoda.[2] 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 $\|N\| \ll \|I\|$.

5. The learning mechanism of Yoda

In this section, we provide a detailed description of the learning mechanism. We first describe the background knowledge of GA. Next, we elaborate the details on phase III of Yoda in Section 5.2.

5.1. Background on genetic algorithms

Genetic algorithms (GAs), which were introduced by Holland [7], 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 survivability 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 survivability 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 [5] 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 5.2.

Comparing to other learning techniques, GA has four major differences [5]:

- The learning processes of GA do not perform on the parameters directly but on a coding of the parameter set. As a result, unlike other learning techniques, GA can perform learning procedures for all types of parameters without limiting the learning ability. For example, the solution space can be discrete or continuous in GA, while the Rocchio-based learning algorithms [21] only allow continuous solution space.
- GA searches the optimal solution from a population of points. Since these points can simultaneously seek the characteristics of the optimal solution, the learning process is more efficient than other common learning techniques, such as neural network and reinforcement learning.
- The learning processes of GA are based on the payoff information from the fitness function only. Contrasting to GA, many learning techniques require auxiliary information during learning. For example, derivatives are essential for gradient-based learning techniques, and the environment information is necessary for dynamic programming method of the reinforcement learning techniques. In relevance feedback problems, since GA does not directly employ the relevance feedback but employ the payoff information as an objective direction, the GA-based learning mechanism can tolerant incorrect relevance feedback. Moreover, GA can also learn from the relevance feedback where the majority of items receive negative scores.
- GA uses probabilistic transition rules to lead the search direction as well. Hence, GA has less chance of stopping in local optimal answers.

Overall, given the facts that: 1) the solution space is discrete in Yoda, 2) the relevance feedback data are usually imperfect, 3) the majority of items may indeed receive negative scores, 4) the learning processes should be efficient, and 5) the chance of finding local optimal should be minimized, the GA-based learning technique is more appropriate for Yoda than other learning techniques.

5.2. Phase III—Adjusting user settings

Yoda's 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 efforts 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.[3] 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 navigation data and i be a navigated 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:

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

Assume that users only navigate potentially desired items and they only browse the pages of uninterested items for a comparably shorter time; therefore, 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. Conversely, the items whose view-time is much shorter than the average view-time of all navigation data will be considered as detested items and thus receive negative preference values. 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 Eq. (10) (Table 1).

$$\epsilon_t = \begin{cases} \left(\dfrac{\upsilon_t(i) - (\tilde{\mu_t} - 3\tilde{\sigma_t})}{\tilde{\mu_t} - 3\tilde{\sigma_t}} \times \dfrac{\mu_f}{\tilde{\mu_t}}\right) & \text{if } \left(\dfrac{\upsilon_t(i) - (\tilde{\mu_t} - 3\tilde{\sigma_t})}{\tilde{\mu_t} - 3\tilde{\sigma_t}} \times \dfrac{\mu_f}{\tilde{\mu_t}}\right) < 1 \\ -1 & \text{else} \end{cases}$$

$$\psi_t = \begin{cases} \epsilon_t & \text{if } \upsilon_t(i) \le (\tilde{\mu_t} - 3\tilde{\sigma_t}) \\ \dfrac{\upsilon_t(i)}{\tilde{\mu_t} + 3\tilde{\sigma_t}} & \text{if } (\tilde{\mu_t} - 3\tilde{\sigma_t}) < \upsilon_t(i) \le (\tilde{\mu_t} + 3\tilde{\sigma_t}) \\ 1 & \text{if } \upsilon_t(i) > (\tilde{\mu_t} + 3\tilde{\sigma_t}) \end{cases} \tag{8}$$

$$\psi_{(f=(h\vee s))} = \begin{cases} \dfrac{\upsilon_f(i)}{\mu_f + 3\sigma_f} & \text{if } \upsilon_f(i) \le (\mu_f + 3\sigma_f) \\ 1 & \text{if } \upsilon_f(i) > (\mu_f + 3\sigma_f) \end{cases} \tag{9}$$

$$\bar{v}_u(i) = \begin{cases} (\omega_s \times \psi_s) + (\omega_h \times \psi_h) + (\omega_t \times \psi_t) & \text{if } \psi_t \ge 0 \\ \psi_t & \text{if } \psi_t < 0 \end{cases} \tag{10}$$

Table 1. Parameters for Eqs. (8), (9), and (10).

Parameter	Definition
ω_f	Importance weight of feature f , where $\sum_f \omega_f = 1$
μ_f	Mean of user's current navigation data in feature f
σ_f	Standard deviation of user's current navigation data in feature f
$\tilde{\mu_t}$	Mean of view-time from all users' navigation data
$\tilde{\sigma_t}$	Standard deviation of view-time from all users' navigation data

Note that for the normalization purpose, we use $(\mu_f + 3\sigma_f)$ as the upper bound in Eq. (9). This upper bound can prevent the effect from outliers [10] of the navigation data. Moreover, in order to distinguish the uninterested items from other desired items, we employ $(\tilde{\mu_f} - 3\tilde{\sigma_f})$ as the threshold. The navigated item whose view-time is lower than the threshold will be assigned a negative preference value.

Subsequently, we explain the coding design for GAs in our learning mechanism. The chromosomes represent a possible confidence list of a specific user and each gene corresponds to a confidence value. 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, g_i, in $[0, L-1]$, where L is the number of fuzzy terms used in the system, and denotes the user's confidence value $(\pi(u, e) = \frac{g_I+1}{L})$ to expert e. The other is a user fuzzy cut value which is associated with the $(k+1)$th gene. The value of fuzzy cut is $(\alpha + 1)/L$, where $\alpha \in [0, L-1]$ is the value of this gene.

Example 5.1. 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 Eq. (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 Eq. (13) which is based on two measurements. Equation (11) evaluates the similarity on ranking, and Eq. (12) measures the average satisfaction of the user wish-list.

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

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

$$\text{Cos}\,\theta(Q, B) = \frac{\sum_i v_u(i) \times \bar{v}_u(i)}{\sqrt{\sum_i v_u(i)^2 \times \sum \bar{v}_u(i)^2}} \tag{11}$$

$$Avg(Q, B) = \frac{\sum_{i \in Q} \bar{v}_u(i)}{\| Q \|} \tag{12}$$

$$\text{Similarity}(Q, B) = \text{Cos}\,\theta(Q, B) + 3 \times Avg(B, Q) \tag{13}$$

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. Example 5.2 illustrates the processing flow of our GA-based learning mechanism.

Example 5.2. Suppose the number of items in the user wish-list is 5 and the GA-based learning processes will be terminated after 50 generations. The learning mechanism is triggered when the user navigates 5 items. Assume $FN, \omega_h, \omega_t, \omega_s, \tilde{\mu}_t, \tilde{\sigma}_t, L$, and the number of chromosomes in the population are {($i5$, 10, 5, 2), ($i7$, 1, 3, 1), ($i2$, 78, 2, 1), ($i8$, 120, 5, 1), ($i3$, 5, 1, 1)}, 0.3, 0.4,0.3, 145, 40, 8, and 10, respectively. First, based on Eq. (10), the learning mechanism converts FN to relevance feedback B where the items and their corresponding preferences are {($i5$, 0.66), ($i7$, −0.51), ($i2$, 0.30), ($i8$, 0.46),($i3$, −0.42)}.

Next, assuming there are five experts in the system and we know the confidence values of the first two experts as {0.5, 0.625} through the PPED measurement, Yoda encodes the original confidence list and the standard fuzzy set value 0.5 to a chromosome, whose genes are {4, 3, 0, 0, 0, 3}. Yoda also randomly generates other 9 chromosomes and employ these 10 chromosomes as the initial population.

During the learning processes, each chromosome will be decoded to a list of confidence values and a fuzzy cut, and employed to generate the corresponding user wish-list. Based on the user wish-list, each chromosome in the population is evaluated its fitness degree (survivability) through Eq. (13).

Subsequently, the learning mechanism produces the next generation of the population through crossover and mutation operations on the current population, where the higher the survivability of the chromosome the higher the possibility it will be picked to generate its offspring. This breeding procedure will be performed until the number of the next-generation chromosomes is 10. These evaluation and breeding procedures will be repeatedly performed for producing the newer generation until the termination of the GA-based learning processes.

Before terminating the learning processes, the learning mechanism will select one chromosome that has highest survivability from the final population, and decode the chromosome to replace the old setting. This updated setting (an updated confidence list and an updated fuzzy cut) can then provide better suggestion in the later recommendation processes.

6. Performance evaluation

We conducted two different sets of experiments in this paper. The first one is an end-to-end simulation to compare accuracy and scalability of Yoda with the basic nearest-neighbor (BNN) method [22]. As far as we know, BNN is the most accurate method for recommendation among current techniques; however, it suffers from low performance. Since we have shown the complexity of Yoda in Section 4.2 for the performance comparison, we emphasis on the accuracy comparison between BNN and Yoda in our first set of experiments.

The second set is a simulation to illustrate the accuracy improvement by using our GA-based learning mechanism. Both Yoda and BNN are implemented in C and on top of Microsoft Access 2000, which is running on a Pentium II 233 MHz processor with Microsoft NT4.0. Moreover, we developed a GA for Yoda using SUGAL [8] for its wide range of operators and data types.

This section is structured as follows. In Section 6.1, we describe our benchmarking methods for the two sets of experiments. Section 6.2 discusses the details of our experimental results.

6.1. Experimental methodology

Theoretically, preference values and navigation behaviors collected from real users are the best source of benchmarks. However, we have observed that the navigation behaviors of volunteers, who are not the real users of an application, are usually inconsistent. Moreover, it is difficult to acquire correct user feedback after the long duration of the experiments. Therefore, we conduct all the experiments by utilizing synthetic data generated from our benchmarking method, which can ensure the consistency of data, and therefore allow the experimental results to be compared fairly.

In order to populate data for evaluation purposes, we propose a parametric algorithm to simulate various benchmarks (see Table 2). 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[4] 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.

Subsequently, the algorithm generates PVs for each item i based on the favorite PVs of the cluster that has the highest preference value for item i, say cluster k'. The higher the

Table 2. Benchmarking parameters.

Parameter	Definition
d	Number of properties
M	Number of items in the web-site
$L_{\min}$	Minimum size of user browse-lists
$L_{\max}$	Maximum size of user browse-lists
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

preference value of item i in cluster k', the more similar PVs of i to favorite PVs of cluster k'.

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 Eq. (6).

To simulate imperfect user feedback, 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 assigns feature values as the user navigation behaviors.[5] These feature values are generated by a reverse procedure of Eq. (10). For example, the higher the preference value of an item is, the longer is the view-time assigned to this item. The corresponding preference values of browsed items in $\mathring{A}_0$ are equivalent to the rating values to be used by BNN. We use Yoda and BNN to construct wish-list I_u for each user u and compare the average satisfaction with Eq. (12) to evaluate the accuracy of these systems. Finally, the system generates the wish-lists after learning processes and evaluates the accuracy improvement of Yoda.

6.2. *Experimental results*

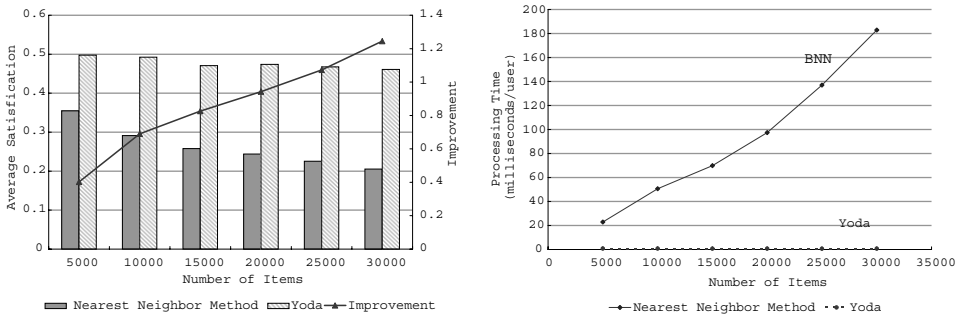
6.2.1. Yoda and BNN comparison. We conducted several experiments to compare Yoda with BNN. In these experiments, we observed a significant margin of improvement over BNN in matching the user expectations for various settings. It also shows that performance of Yoda is independent of the number of users.

The results shown for each set of experiments are averaged over many runs, where each run is executed with different seeds for the benchmarking procedure. The benchmark settings of the following parameters, i.e., d, $L_{\min}$, $L_{\max}$, N, F, E, O, K, and U, are fixed at 20, 20, 40, 50, 10, 50, 10, 10, and 500, respectively.

$$\text{Improvement (Yoda, BBN)} = \frac{(\text{Yoda} - \text{BBN})}{\text{BBN}} \tag{14}$$

Figure 2 illustrates the improvements of Yoda over BNN. The X-axis depicts the number of items varying from 5000 to 30000. Y-axis of figure 2(a) depicts average satisfaction computed by Eq. (12) and improvement computed by Eq. (14). The Y-axis of figure 2(b) is the system processing time. The average portion of interested items among item set P is 25%. In other words, users are only interested in 25% of the items in the item sets and thus only 25% of the items have positive preference values.

Figure 2(b) verifies that Yoda is scalable. As the number of items grows, the system processing time of Yoda remains unchanged while the processing time of BNN increases linearly. This is because Yoda is a model-based recommendation system. Figure 2(a) indicates that Yoda always outperforms BNN in accuracy. Although performances of both systems decrease as the number of items grow, the margin of improvement between Yoda and BNN expands. We attribute this improvement to the incorporation of content-based filtering into the Yoda framework.



a. Average satisfaction

b. Processing time

Figure 2. Impacts of number of items.

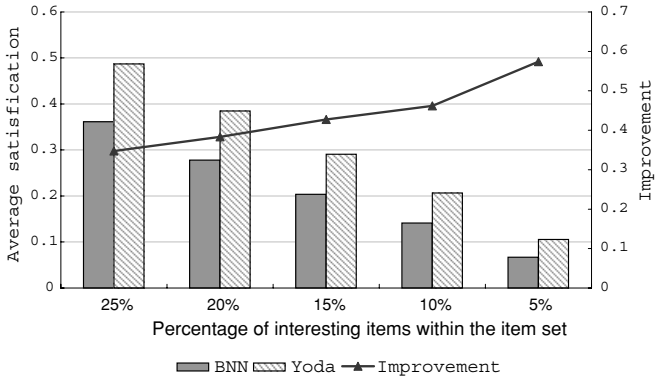


Figure 3. Impacts of the percentage of interesting items within the item set.

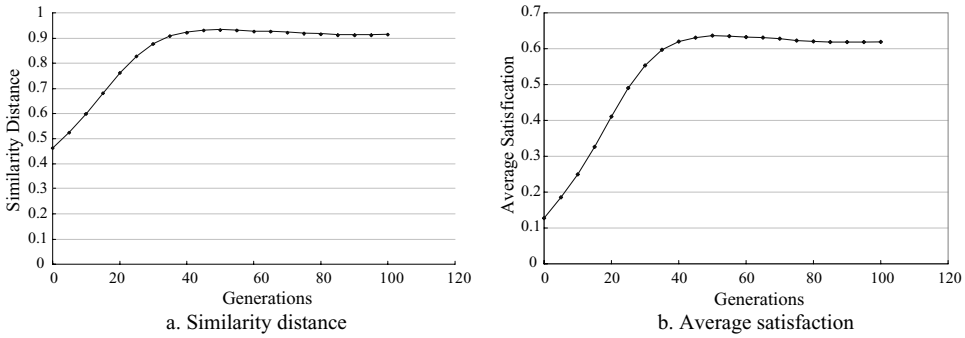
With figure 3, we demonstrate the impacts of P in the average satisfaction of the two systems. The X-axis depicts the percentage of interesting items within the item set from 25 to 5%. The Y-axis depicts the average satisfaction computed and the improvement. The size of item sets is $M = 5000$. In figure 3, the average satisfaction values of both Yoda and BNN decrease as users become more picky. However, the margin of improvement between Yoda and BNN grows as P decreases. Again, adopting content-based filtering enables Yoda to recognize the latent associations between items and hence locate more items that would be desired by users. Note that the value of P in practical applications such as e-commerce stores could be much smaller than in the above example. This observation shows that Yoda is a more appropriate recommendation system in the real world. In the next section, the experimental results indicate that the performance of Yoda can be further improved from 0.13 to 0.6 (more than 350% improvement) by incorporating the GA-based learning mechanism.

6.2.2. System improvement after learning. We conducted several 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 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 limitation, 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 the GA operators [5, 8] are: population size = 30, one-point crossover, tournament selection, keep the elitism, and mutation rate = 0.25. The benchmark settings of the following parameters, 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 4 demonstrates the improvements achieved by our learning mechanism. The X-axis of figure 4 depicts the number of generations. The Y-axis of figure 4(a) illustrates the similarity distance (between query results and perfect user feedback) computed by Eq. (11). The Y-axis of figure 4(b) represents average satisfaction computed by Eq. (12). Figure 4 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 noted that Yoda 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 4, the



a. Similarity distance b. Average satisfaction

Figure 4. The system improvement.

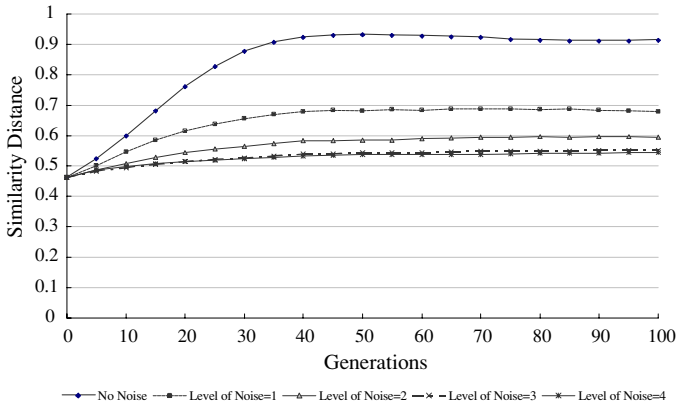


Figure 5. Impact of noise.

accuracy of wish-lists slightly decreases after 50 generations of learning. The reason is that the learning mechanism is constrained by the limited feedback. 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.

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 5, 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 Eq. (11). As revealed by figure 5, 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.

7. Conclusion

We proposed a hybrid recommendation system that combines content-based and collaborative filtering techniques in order to reduce the information overload problem in e-commerce applications. Our 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. The results 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.

Acknowledgments

This research has been funded in part by NSF grants EEC-9529152 (IMSC ERC), IIS-0082826 (ITR), IIS-0238560 (CAREER) and IIS-0307908, and unrestricted cash gifts from Okawa Foundation and Microsoft. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Notes

1. That is, the active user can be classified into different navigation-pattern clusters with the corresponding possibilities (confidence values) between 0 to 1.
2. Since our aggregation function is in triangular norm form, it satisfies the requirements of the A_0 algorithm.
3. 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.
4. The random number is generated through a linear congruential formula [11].
5. In other words, currently, we only consider product description pages as navigated pages in this experiment.

References

1. M. Balabanovi, "An adaptive web page recommendation service," in Proceedings of Autonomous Agents, 1997, pp. 378–385.
2. M. Balabanovi and Y. Shoham, "Fab, content-based, collaborative recommendation, Communications of the ACM, vol. 40, no. 3, pp. 66–72, 1997.
3. J. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence, 1998, pp. 43–52.
4. R. Fagin, "Combining fuzzy information from multiple systems," in Proceedings of Fifteenth ACM Symposyum on Principles of Database Systems, 1996.
5. D.E. Goldberg, Genetic Algorithms in Search, Optimisation, and Machine Learning, Addison-Wesley: Wokingham, England, 1989.
6. N. Good, J. Schafer, J. Konstan, J. Borchers, B. Sarwar, J. Herlocker, and J. Riedl, "Combining collaborative filtering with personal agents for better recommendations," in Proceedings of the 1999 Conference of the American Association of Artifical Intelligence, 1999, pp. 439–446.
7. J. Holland, Adaption in Natural and Artificial Systems, University of Michigan Press: Ann Arbor, Michigan.
8. A. Hunter, Sugal Programming manual. http://www.trajan-software.demon.co.uk/sugal.htm, 1995.
9. B. Kitts, D. Freed, and M. Vrieze. "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, 2000, pp. 437–446.
10. E. Knorr, R. Ng, and V. Tucakov, "Distance-based outliers: Algorithms and applications," The VLDB Journal, vol. 8, no. 3, pp. 237–253, 2000.
11. D. Knuth, "Seminumerical algorithm," The Art of Computer Programming vol. 2, 1997.
12. A. Kohrs and B. Merialdo, "Using category-based collaborative filtering in the active WebMuseum," in Proceedings of IEEE International Conference on Multimedia and Expo, 2000, vol. 1, pp. 351–354.
13. J. Konstan, B. Miller, D. Maltz, J. Herlocker, L. Gordon, and J. Riedl, "Applying collaborative filtering to usenet news," Communications of the ACM, no. 4, p. 3, 1997.
14. W. Lam, S. Mukhopadhyay, J. Mostafa, and M. Palakal, "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, 1996, pp. 317–325.

15. H. Lieberman, N. Dyke, and A. Vivacqua, "Let's Browse, a collaborative Browsing Agent," Knowledge-Based Systems, vol. 12, pp. 427–431, 1999.
16. B. Mobasher, R. Cooley, and J. Srivastava, "Automatic personalization based on Web usage mining," Communications of the ACM, vol. 43, no. 8, pp. 142–151, 2000.
17. B. Mobasher, H. Dai, T. Luo, and M. Nakagawa, "Web data mining: Effective personalization based on association rule discovery from web usage data," in Proceeding of the Third International Workshop on Web Information and Data Management, 2001.
18. A. Moukas, "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), 1996.
19. M. Pazzani and D. Billsus, "Learning and revising user profiles: The indentification of interesting web sites," Machine Learning, vol. 27, pp. 313–331, 1997.
20. P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, "GroupLens, an open architecture for collaborative filtering of netnews," in Proceedings of ACM conference on Cumputer-Supported Cooperative Work, 1994, pp. 175–186.
21. Y. Rui, T. Huang, M. Ortega, and S. Mehrotra, "Relevance feedback: A power tool for interactive content-based image retrieval," IEEE Transactions on Circuits and Systems for Video Technology, vol. 8, no. 5, pp. 644–655, 1998.
22. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of dimensionality reduction in recommender system—A case study," in Proceedings of ACM WebKDD 2000 Web Mining for e-Commerce Workshop, 2000.
23. B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Analysis of recommendation algorithms for e-Commerce," in Proceedings of ACM e-Commerce 2000 Conference, 2000.
24. C. Shahabi, F. Banaei-Kashani, Y.-S. Chen, and D. McLeod, "Yoda: An accurate and scalable web-based recommendation system," in Proceedings of Sixth International Conference on Cooperative Information Systems (CoopIS 2001), 2001.
25. C. Shahabi, F. Banaei-Kashani, J. Faruque, and A. Faisal, "Feature matrices: A model for efficient and anonymous web usage mining," in Proceedings of EC-Web, 2001.
26. C. Shahabi, F. Banaei-Kashani, and J. Faruque, "A reliable, efficient, and scalable system for web usage data acquisition," in WebKDD'01 Workshop in Conjunction with the ACM-SIGKDD, 2001.
27. C. Shahabi, A.M. Zarkesh, J. Adibi, and V. Shah, "Knowledge discovery from users web page navigation," in Proceedings of the IEEE RIDE97 Workshop, 1997.
28. U. Shardanand and P. Maes, "Social information filtering, algorithm for automating word of mouth," in Proceedings on Human Factors in Computing Systems, 1995, pp. 210–217.
29. B. Sheth and P. Maes, "Evolving agents for personalized information filtering," in Proceedings of the Ninth IEEE Conference on Artificial Intelligence for Applications, 1993.
30. A. Tan and C. Teo, "Learning user profiles for personalized information dissemination," in Proceedings of Int'l Joint Conf. on Neural Network, 1998, pp. 183–188.
31. L. Wu, C. Faloutsos, K. Sycara, and T. Payne, "FALCON: Feedback adaptive loop for content-based retrieval," in Proceedings of Int'l. Conf. on Very Large Data Bases, 2000.