

Short Paper

A Data Mining-Based Method for the Incremental Update of Supporting Personalized Information Filtering*

YE-IN CHANG, JUN-HONG SHEN AND TSU-I CHEN

Department of Computer Science and Engineering

National Sun Yat-Sen University

Kaohsiung, 804 Taiwan

E-mail: changyi@cse.nsysu.edu.tw

Information filtering is an area of research that develops tools for discriminating between relevant and irrelevant information. Users first give descriptions about what they need, *i.e.*, user profiles represented by a set of keywords, to start the services. A profile index is built on these profiles. Then, the Web page will be recommended to the users whose profiles belong to the filtered results. Therefore, a critical issue of the information filtering service is how to index the user profiles for an efficient matching process. Among previous proposed methods, Wu and Chen's graph-based index method can expect to minimize the storage space. However, when the users often change their interests, the index structure of Wu and Chen's method needs to be reconstructed, resulting in the high update cost. Therefore, in this paper, we propose a data mining-based method for the incremental update of the index structure, the *updatable tree*, to reduce the update cost. In fact, each keyword could have a weight representing the degree of importance to a user. We apply this feature to distinguish between long-term and short-term interests. By making use of the property that the short-term interest has a higher probability to be changed than the long-term one, our proposed method can locally update the short-term interest, resulting in the low update cost. According to our experimental results, our method really can reduce the update cost as needed by Wu and Chen's method.

Keywords: data mining, incremental update, information filtering, personalization, profile

1. INTRODUCTION

The growth of the Web has brought about the rapid accumulation of data and the increasing possibility of information sharing. When searching the Web, a user can be overwhelmed by thousands of results retrieved by a search engine, and few of which are valuable. Therefore, many techniques have been developed on the Web to retrieve useful information. Information filtering is one of the techniques to help users find what they want [4, 6, 7, 15]. It is classified into two kinds of approaches, collaborative filtering and content-based information filtering. Collaborative filtering identifies the relevant users

Received February 2, 2007; accepted July 13, 2007.

Communicated by K. Robert Lai, Yu-Chee Tseng and Shu-Yuan Chen.

* This research was supported in part by the National Science Council of Taiwan, R.O.C. under grant No. NSC 95-2221-E-110-101 and by National Sun Yat-Sen University.

who own similar interests and provides the data they like to each other [3, 8]. However, it is not well-suited to locating information for a specific content information need [8]. On the other hand, content-based information filtering identifies and provides the relevant data for the users based on the similarity between data and their interests [14].

In this paper, we focus on content-based information filtering. Each user has his (her) profile which stores a set of keywords that can present his (her) interests [1, 12, 15]. For a profile to match the document, every keyword that it contains must be in the document [13]. The matched Web pages are also presented with the associated set of keywords. Comparing data with profiles, the users who are interested in the data are identified and informed. That is, information filtering can find good matches between the Web pages and the users' information needs [9-11, 14].

In order to match data with profiles efficiently, a profile index is built on these profiles. Indexing user profiles can reduce the costs of storage space and the processing time for comparing the user profiles with incoming Web pages. We can use a proxy server which is regarded as a mechanism to produce Web pages. That is, the Web pages fetched by the proxy server will form the incoming Web pages for the information filtering service [13]. Two kinds of models that index structures are based on, vector space and boolean models, are used on the information filtering service. In the vector space model, user profiles and documents are identified by keywords which are associated with the weight that can represent its statistical importance, such as its frequency in the document. Since each keyword has a weight, the vector space model can provide the best match with relevance ranking. On the other hand, the user may use the boolean model to specify keywords that he (she) wants in documents received [5, 16]. The boolean model is used to provide the exact match, and simple to implement. Note that, in fact, the boolean model is a special case of the vector space model in which all the keywords have the same weight. Both models have their applicable problem domains.

In [15], Yan and Garcia-Molina have proposed three methods based on the vector space model: the brute force method, the profile indexing method and the selective profile indexing method. In [16], Yan and Garcia-Molina have proposed four methods based on the boolean model: the brute force method, the counting method, the key method and the tree method. To improve the performance in terms of the storage space on storing the index in [16], Wu and Chen [13] have proposed four methods: index path with path signatures, index graph with path signatures, index path with profile sets, and index graph with profile sets.

Among these methods for information filtering, Wu and Chen's index graph with profile sets [13] can expect to minimize the storage space at the cost of the processing time. However, their method does not concern about the issue of updates. With the concern of updates, the reorganization of an index structure should be locally operated so that the update cost for reorganizing the structure is minimized. When someone's interests are changed, Wu and Chen's graph-based index structure [13] needs to be reconstructed from the root. That is, their index structure will be affected globally, resulting in the high update cost.

Therefore, in this paper, in order to reduce the update cost as needed by Wu and Chen's method [13], we propose a data mining-based method for constructing the index structure, the *updatable tree*, which supports the incremental update. We adopt the vector space model. Each keyword can be distinguished between the long-term interest which

has the weight greater than or equal to the threshold and the short-term interest which has the weight less than the threshold. Moreover, we use a revised version of the Apriori algorithm [2], a well-known data mining method for mining association rules, to get the large itemset, *i.e.*, the long-term interest, by taking weights of keywords in profiles into consideration. By making use of the property that the probability of modifying the short-term interests is higher than that of modifying the long-term interests, we can update the short-term interests locally to reduce the update cost. According to our experimental results, our method really can reduce the update cost as needed by Wu and Chen's method [13].

The rest of the paper is organized as follows. Section 2 presents a data mining-based method for the incremental update. In section 3, we show the performance study and make a comparison of the proposed method and Wu and Chen's method. Finally, section 4 gives the conclusion.

2. THE DATA MINING-BASED METHOD FOR THE INCREMENTAL UPDATE

Based on Wu and Chen's method [13], when someone's interests are changed, their graph-based index structure needs to be reconstructed from the root. In this section, we present a data mining-based method for the incremental update of the index structure for storing keywords to reduce the update cost.

2.1 The Proposed Method

We adopt the vector space model. Basically, a profile in the vector space model contains a list of keywords and each keyword is weighted according to its degree of importance. Hence, each keyword in the profile is given a weight that signifies its statistical importance. The weight of a keyword is bounded by $(0..1]$ and weights among keywords are independent. For example, in profile $\{car, stock\} = \{0.9, 0.2\}$, keyword *car* has a weight 0.9, and keyword *stock* has a weight 0.2. Therefore, in the proposed method, we take the weight of each keyword into consideration. Furthermore, we assume that user profiles are clustered so that in each cluster, the user profiles have similar interests. Our proposed method is then applied in a cluster.

A threshold α is given to distinguish how importance of those keywords is. If the weight of the keyword is greater than or equal to threshold α , it can be regarded as the *long-term interests*. The long-term interests are interests that result from an accumulation of experiences over a long time. On the other hand, if the weight of the keyword is less than threshold α , it can be regarded as the *short-term interests*. The short-term interests are interests in events on a day-to-day basis which change over a short period. By making use of the property of the keywords which are assigned with the weight, we can reduce the update cost in our proposed method.

In our proposed method, we use a revised version of the Apriori algorithm [2] to get large itemsets. The Apriori algorithm constructs a candidate set of large itemsets, counts the number of occurrence of each candidate set, and then determines large itemsets based on a predetermined minimum support. Because the large itemsets in our proposed method represent the long-term interests which are not often modified, we modify the

definition of candidate itemsets in the Apriori algorithm. That is, the count of the keyword will be increased only when the weight of the keyword is greater than or equal to threshold α . Moreover, in the original Apriori algorithm, if the value of the predetermined minimum support is set too large, it may not generate large itemsets. Therefore, in our revised Apriori algorithm, the minimum support is dynamically decided to guarantee

that we can get large itemsets. It is calculated by $\frac{\sum_{i=1}^{|C_n|} Sup(C_i)}{|C_n|}$, where C_n represents the candidate itemset in the n 'th round, and $Sup(C_i)$ represents the number of occurrence of candidate itemset C_i among profiles.

Profile	Keywords/Weight
P1	b, e, h = {0.2, 0.1, 0.2}
P2	e, f, g = {0.3, 0.1, 0.4}
P3	d, h = {0.2, 0.2}
P4	e, h = {0.3, 0.4}
P5	a, i, j = {0.2, 0.4, 0.3}

(a)

Profile	Keywords/Weight
P1	c, d, g, i = {0.9, 0.7, 0.8, 0.6}
P2	a, b, c, i, j = {0.7, 0.9, 0.5, 0.6, 0.8}
P3	a, b, c, e, i, j = {0.8, 0.6, 0.6, 0.7, 0.9, 0.5}
P4	c, d, f, g = {0.8, 0.7, 0.8, 0.5}
P5	c, d, f, g = {0.8, 0.6, 0.5, 0.7}

(b)

Fig. 1. Profiles for the running example: (a) the profiles containing those keywords with the weight $< \alpha$; (b) the profiles containing those keywords with the weight $\geq \alpha$.

Take an example in Fig. 1 to illustrate the way to get the large itemset. In Fig. 1, there are five profiles which contain a list of keywords with weights and the threshold $\alpha = 0.5$. Those keywords with the weight $< \alpha$ are shown in Fig. 1 (a), and those keywords with the weight $\geq \alpha$ are shown in Fig. 1 (b). Because the keyword with the weight $\geq \alpha$ will become the candidate item, we only use those profiles as shown in Fig. 1 (b) to be the input data. That is, as shown in Fig. 2, the large itemset can be chosen from those profiles shown in Fig. 1 (b). Those profiles are scanned once to generate the one-item candidate itemset, C_1 , with the support that is the number of occurrence among them.

The minimum support of C_1 is calculated by $\frac{(2+2+5+3+1+2+3+3+2)}{9} = 2.6$. So, we choose only those itemsets $\{c\}$, $\{d\}$, $\{g\}$, $\{i\}$ which have support greater than or equal to 2.6 to be L_1 . Next, itemsets in C_2 are generated from the combination of any two itemsets in L_1 . The similar process is preceded until no more large itemsets are generated. The final result of the large itemset is $L_3 = \{c, d, g\}$. We choose the large itemset with the longest length, the same as that in the Apriori algorithm.

By using our revised Apriori algorithm, we can get the large itemset from the profiles which have the weight of each keyword greater than or equal to the threshold. After we get the large itemset, we divide those profiles into two parts according to the result of the large itemset. One part contains the large itemset and the other part does not contain the large itemset. Next, those profiles in the two parts keep on getting the large itemset by using our revised Apriori algorithm, respectively. We use each result of the large itemsets to construct the index structure, the *updatable tree*. Those steps are repeated until no keyword is in the profiles which have the weight of each keyword greater than or equal to the threshold. Note that the construction of the updatable tree requires a number

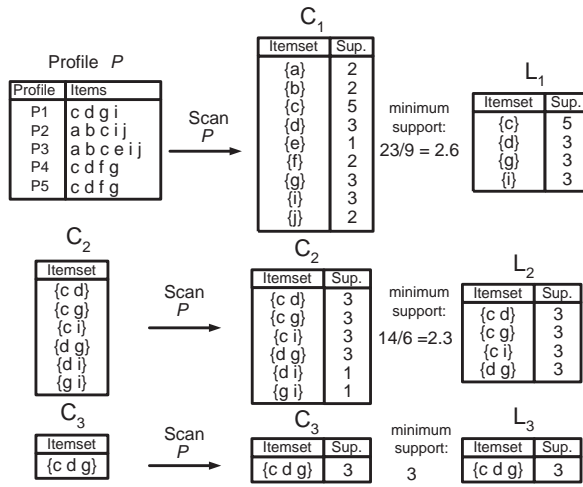


Fig. 2. An example of getting the large itemset.

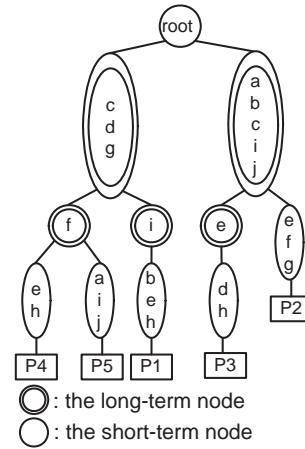


Fig. 3. The updatable tree.

of iteration on finding the large itemsets. This may be time-consuming, but it operates in the off-line manner. Finally, we insert the identifiers of the profiles and those keywords which have the weight less than the threshold to the index structure according to the path from the root that those profiles own by themselves. By making use of the property that the probability of modifying the short-term interests is higher than that of modifying the long-term interests, and that we always put the short-term interests which have the weight less than the threshold to the leaf nodes of the tree, we can update the short-term interests locally.

Let's use an example shown in Fig. 1 to illustrate those steps of constructing the updatable tree. As described above, we divide those profiles into two parts. We use the profiles as shown in Fig. 1 (b), which have the weight of each keyword greater than or equal to the threshold, to get the large itemset. At the first time, the large itemset is $\{c, d, g\}$ representing long-term keywords, as shown in Fig. 2. We create a long-term node to contain the large itemset $\{c, d, g\}$ to the updatable tree, following the root node, as shown in Fig. 3. Next, we divide those profiles into two parts: one part does not contain the large itemset $\{c, d, g\}$ as shown in Fig. 4 (a), and the other part contains it as shown in Fig. 4 (b). Note that, in this way, therefore, only one unique path from the root node will lead to each profile.

Then, we use those profiles P_1, P_4, P_5 which have already removed the large itemset $\{c, d, g\}$ as shown in the left part of Fig. 5 to get the large itemset $\{f\}$ again. We can then divide profiles P_1, P_4, P_5 into two parts again: one part does not contain the large itemset as shown in Fig. 5 (a), and the other part contains it as shown in Fig. 5 (b). After that, we create a long-term node to contain keyword $\{f\}$ to the updatable tree, following the node containing keywords $\{c, d, g\}$. At this point, there is no keyword with the weight $\geq \alpha$ in profiles P_4 and P_5 . So, we create a short-term node to contain keywords $\{e, h\}$ as shown in Fig. 1 (a), which have the weight less than the threshold in profile P_4 , to the updatable tree, following the long-term node containing keyword $\{f\}$. Moreover, we add the identifier of profile P_4 to the updatable tree, following the node containing keywords $\{e, h\}$.

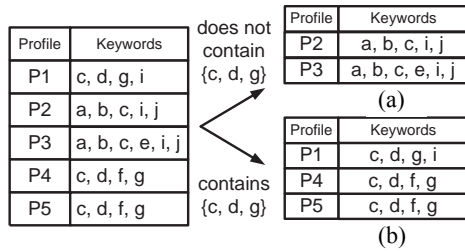


Fig. 4. The profile division based on the large itemset $\{c, d, g\}$: (a) the profiles not containing the large itemset; (b) the profiles containing the large itemset.

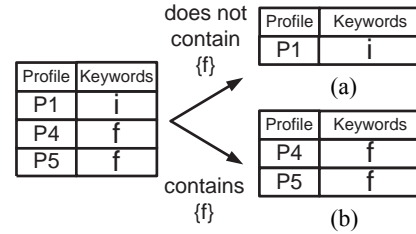


Fig. 5. The profile division based on the large itemset $\{f\}$: (a) the profile not containing the large itemset; (b) the profiles containing the large itemset.

Note that, obviously when the short-term interest is changed, it can be locally updated. Similar to the previous step, we also create a short-term node to contain keywords $\{a, i, j\}$ as shown in Fig. 1 (a), which have the weight less than the threshold in profile P_5 , to the updatable tree, following the node containing keyword $\{f\}$. Next, we add the identifier of profile P_5 to the updatable tree, following the node containing keywords $\{a, i, j\}$.

Then, there is only one profile P_1 that contains one keyword i which has the weight greater than or equal to the threshold. Therefore, we create a long-term node to contain keyword i to the updatable tree, following the node containing keywords $\{c, d, g\}$. After that, there is no keyword with the weight $\geq \alpha$ in profile P_1 . So, we create a short-term node to contain keywords $\{b, e, h\}$ as shown in Fig. 1 (a), which have the weight less than the threshold in profile P_1 , to the updatable tree, following the node containing keyword $\{i\}$. Moreover, we add the identifier of profile P_1 to the updatable tree, following the node containing keywords $\{b, e, h\}$.

Similar to the previous steps, we get the large itemset $\{a, b, c, i, j\}$ from those profiles as shown in Fig. 4 (a). Next, we create a long-term node to contain the large itemset $\{a, b, c, i, j\}$ to the updatable tree, following the root node. After the large itemset $\{a, b, c, i, j\}$ is removed from profiles P_2 and P_3 , there is only one keyword $\{e\}$ in profile P_3 . So, we create a long-term node to contain keyword $\{e\}$ to the updatable tree, following the node containing keywords $\{a, b, c, i, j\}$. Then, we create a short-term node to contain keywords $\{d, h\}$ as shown in Fig. 1 (a), which have the weight less than the threshold in profile P_3 , to the updatable tree, following the node containing keyword $\{e\}$. Moreover, we add the identifier of profile P_3 to the updatable tree, following the node containing keywords $\{d, h\}$. Finally, similar to the previous steps, we create a short-term node to contain keywords $\{e, f, g\}$ as shown in Fig. 1 (a), which have the weight less than the threshold in profile P_2 , to the updatable tree, following the node containing keywords $\{a, b, c, i, j\}$. Moreover, we add the identifier of profile P_2 to the updatable tree, following the node containing keywords $\{e, f, g\}$. Consequently, the final result for the input shown in Fig. 1 is shown in Fig. 3.

2.2 The Matching Process

To find a match for a Web page, the breadth first search from the root in the updatable tree should be conducted. If all the keywords in a node are completely matched

with those of the Web page, the children of this node are then traversed; otherwise, the children of this node are not further traversed. If the identifier of a profile is reached, it is a match and this Web page is recommended to the corresponding user.

For example, a Web page contains keywords $\{a, c, d, e, f, g, h, i, j\}$. Since, in Fig. 3, keywords $\{c, d, g\}$ of the left node of the root are completely contained in this page, its children are then traversed. On the other hand, keywords $\{a, b, c, i, j\}$ of the right node of the root are not completely contained in this page, its children are not further traversed. Then, the similar process is conducted. Finally, this page will be recommended to the users having profiles P_4 and P_5 , respectively.

2.3 The Update Process

According to our data mining-based method for the incremental update as described above, we can reduce the update cost as needed by Wu and Chen's method [13]. For example, in Fig. 1 (a), the weight of keyword f in profile P_2 is 0.1, it is one of the short-term interests which have the high probability to be changed over a short period. According to the updatable tree as shown in Fig. 3, if the user with profile P_2 is not interested in keyword f , we can delete keyword f from the node containing $\{e, f, g\}$. That is, the node containing $\{e, f, g\}$ is changed to the node containing $\{e, g\}$.

```

1: procedure Delete ( $l\_key, P_i$ )
2: begin /*  $l\_key$  is the long-term keyword (interest) for profile  $P_i$  to be deleted. */
3:   locate the node,  $W$ , containing keyword  $l\_key$  of profile  $P_i$  in the updatable tree
   by using keywords of profile  $P_i$ ;
4:   if  $W$  is the node leading to not only profile  $P_i$  then
5:     begin
6:       create a new long-term node  $X$  to contain  $l\_key$ ;
7:       attach node  $X$  to node  $W$ ;
8:       attach the children of node  $W$  not leading to profile  $P_i$  to node  $X$ ;
9:       if node  $X$  has only one child and this child is a long-term node then
10:        combine node  $X$  with its child;
11:     end;
12:   delete  $l\_key$  from node  $W$ ;
13:   if there is no keyword contained in node  $W$  then combine node  $W$  with its parent;
14: end;

```

Fig. 6. Procedure *Delete*.

For the deletion of the long-term keyword (interest), l_key , in profile P_i , we use procedure *Delete* shown in Fig. 6 to deal with it. In procedure *Delete*, we first use keywords of profile P_i to locate the node, W , containing l_key in the updatable tree. If this node is leading to profile P_i and the other profiles, a new long-term node, X , is created to contain l_key , l_key is deleted from node W , and the children of node W should be re-located. Otherwise, keyword l_key is directly deleted from node W . For example, consider that long-term keyword c , referred to as l_key , of profile P_2 is being deleted from

the tree shown in Fig. 3. The node containing $\{a, b, c, i, j\}$, referred to as node W , is located. Since this node is leading to not only profile P_2 but also profile P_3 , a new node, X , is created to contain keyword c . Then, node X is attached to node W and the child of node W not leading to profile P_2 , *i.e.*, the node containing $\{e\}$, is attached to node X . Since node X has only one child that is a long-term node, node X is combined with its child. Next, keyword c is deleted from node W . After that, since node W still contains keywords, no further process is preceded. The final result of this deletion is shown in Fig. 7 (a).

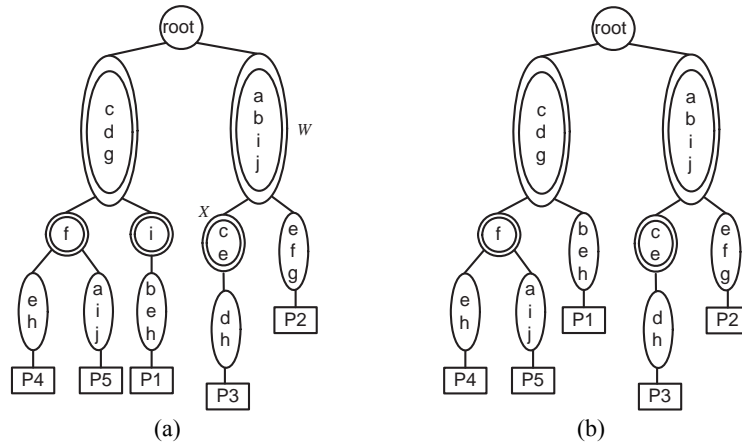


Fig. 7. The updatable tree: (a) after the deletion of long-term keyword c of profile P_2 from the tree; (b) after the deletion of long-term keyword i of profile P_1 from the tree.

Another example is that long-term keyword i of profile P_1 is being deleted from the tree shown in Fig. 7 (a). The node containing $\{i\}$ leading to profile P_1 , referred to as node W , is located. Since it is the only node leading to only profile P_1 , keyword i is directly deleted from node W . After that, since there is no keyword contained in node W , this node is combined with its parent. The final result of this deletion is shown in Fig. 7 (b).

In Fig. 3, if the user with profile P_2 is interested in keyword d over a short period, we will insert keyword d to the node containing the short-term interest. That is, keyword d is inserted to the node containing $\{e, f, g\}$, as shown in Fig. 3.

For the insertion of the long-term keyword (interest), l_key , in profile P_i , we use procedure *Insert* shown in Fig. 8 to deal with it. In procedure *Insert*, the last long-term node, W , leading to profile P_i in the updatable tree is located by using keywords of profile P_i . If this node is leading to only profile P_i , keyword l_key is directly inserted into it. Otherwise, a new long-term node is created to contain l_key , and inserted between node W and its child node leading to profile P_i . Moreover, the other children of node W are further checked whether they contain keyword l_key . If yes, they will merge with node X . For example, a long-term keyword, g , is inserted into the updatable tree shown in Fig. 7 (b) for profile P_3 . The last long-term node containing $\{c, e\}$ leading to profile P_3 , referred to as node W , is located. Since this node is leading to only profile P_3 , keyword g is directly inserted into this node. The result of this insertion is shown in Fig. 9 (a).

```

1: procedure Insert (l_key, Pi)
2: begin /* l_key is a new long-term keyword (interest) for profile Pi. */
3:   locate the last long-term node, W, leading to profile Pi in the updatable tree by
   using keywords of profile Pi;
4:   if W is the node leading to only profile Pi then
5:     insert l_key into node W
6:   else
7:     begin
8:       create a new long-term node X to contain l_key;
9:       attach the child of node W leading to profile Pi to node X;
10:      attach node X to node W;
11:      if children of node W except node X contain l_key
        and are the long-term nodes then
12:        begin
13:          delete l_key from these children of node W containing l_key;
14:          attach them to node X;
15:          if there is no keyword contained in these children of node W then
16:            combine them with node X;
17:          end;
18:          if node X is the only child of node W then combine node X with node W;
19:          end;
20:        end;

```

Fig. 8. Procedure *Insert*.

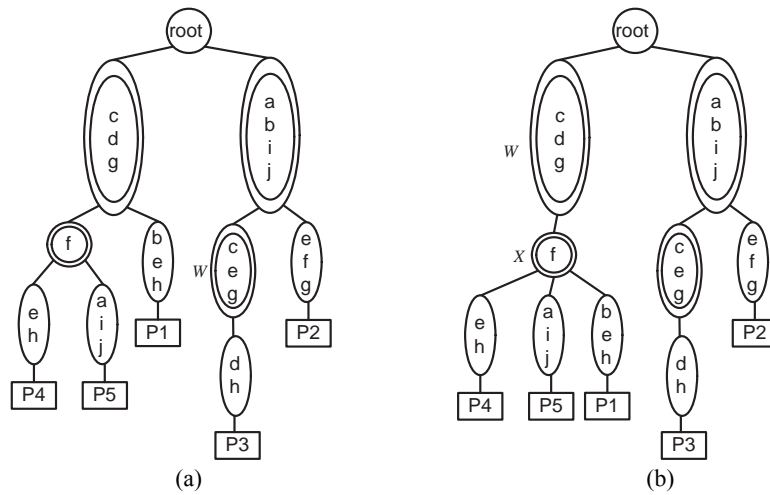


Fig. 9. The updatable tree: (a) after the insertion of long-term keyword *g* into the tree for profile *P₃*; (b) after the insertion of long-term keyword *f* into the tree for profile *P₁*.

Another example is that a long-term keyword, *f*, is inserted into the updatable tree shown in Fig. 9 (a) for profile *P₁*. The last long-term node containing {*c*, *d*, *e*} which is leading to profile *P₁*, referred to as node *W*, is located. Since this node is leading to not

only profile P_1 but also profiles P_4 and P_5 , we create a new long-term node X to contain keyword f and check whether the other child of node W contains keyword f (lines 7-19 in procedure *Insert*). This process is preceded as follows. First, a new long-term node, X , is created to contain keyword f and the node containing $\{b, e, h\}$ leading to profile P_1 is attached to node X . Next, since the other child of node W contains keyword f and is the long-term node, this child deletes keyword f and is attached to node X . After that, since there is no keyword contained in this child, it is combined with node X . The combined result is shown in Fig. 9 (b). Finally, since node X is the only child of node W , it is combined with its parent, node W .

Note that if a number of insertions and deletions are operated on the short-term nodes in the updatable tree, these operations modify only the keywords of the corresponding profiles and do not modify the shared keywords in the long-term nodes among the profiles. Therefore, we do not need to reorganize the updatable tree. If a large number of insertions and deletions are operated on the long-term nodes in the updatable tree, these operations will cause the long-term nodes to be split, resulting in the increase in the number of nodes in the tree and the size of the tree. That will increase the storage space. At this point, to reduce the storage space, we will reorganize the updatable tree. Since users' long-term interests are rarely changed as mentioned before, we do not need to reorganize the updatable tree frequently.

3. PERFORMANCE

In this section, we make a comparison of our proposed method and Wu and Chen's index graph with profile sets [13].

3.1 The Simulation Model

We generate synthetic profiles to evaluate the performance [16]. The number of profiles is N . To simplify the study of the effect of the profile size on performance, all profiles have the same length, K ; that is, K is fixed for all profiles. The keywords that all profiles choose are composed of the set of keywords D . So, keywords in the first profile are chosen randomly from the set of keywords D . Moreover, the weight of each keyword is chosen with uniform distribution from $(0, 1]$. The first profile is called "base profile." In our assumption, the users with similar interests are clustered into the same group. Therefore, in order to model the similarity among profiles, the similarity parameter Q controls how similar the new profile and the base profile are. That is, for each keyword in the new profile, there is a probability Q that it is the same as the corresponding keyword in the base profile. If it is not, then the keyword in the new profile is picked up at random from the set of keywords D . There is no duplicated keyword in the profile. Hence, by varying the value of Q from 0 to 1, we can control the similarity among the profiles. If the value of Q is 0, the keywords in all profiles are randomly chosen from the set of keywords D .

3.2 Experimental Results

We generate the profiles used in our simulation based on the setting: $N = 500$, $K = 5$,

$D = 50$, and $Q = 80\%$. That is, we cluster 500 users with similar interests into the same group. The length of each profile is 5. The set of keywords is composed of 50 keywords. Moreover, we choose 80% to decide the similarity among profiles. Furthermore, we have threshold $\alpha = 0.5$ that is used to determine whether a keyword is a long-term interest. That is, if the weight of a keyword is greater than or equal to 0.5, the keyword is a long-term interest; otherwise, it is a short-term interest.

Table 1. Parameters and their default settings used in the simulation.

Parameter	Default value
(PD, PI)	(30%, 70%), (40%, 60%), (50%, 50%), (60%, 40%), (70%, 30%)
(PS, PL)	(20%, 80%), (40%, 60%), (60%, 40%), (80%, 20%), (100%, 0%)

PD : The probability of doing the deletion operation

PI : The probability of doing the insertion operation

PS : The probability of modifying the short-term interests

PL : The probability of modifying the long-term interests

In our simulation, four parameters and their default settings are listed in Table 1. Owing to that the update process contains the deletion and insertion operations, we can observe the impact of the ratio between the deletion and insertion operations for the update cost. Moreover, we can adjust the ratio of the probability of modifying the short-term interests to that of modifying the long-term interests. Note that in our simulation, there are 100 update operations applied to each index structure. First, we define a base case, $(PD, PI) = (50\%, 50\%)$ and $(PS, PL) = (80\%, 20\%)$. The first pair means that the probability of doing the deletion operation (PD) is 50% and that of doing the insertion operation (PI) is also 50%. That is, among 100 update operations, there are 50 deletions and 50 insertions. The latter pair means that the probability of modifying the short-term interests (PS) is 80% and that of modifying the long-term interests (PL) is 20%. That is, there are 80 out of 100 update operations applied to the short-term interests and the remaining 20 update operations applied to the long-term ones.

When we do the update operation of the keywords which the user is (not) interested in, first, we must pass through the index structure to find the profile which the user has. Then, we do the update operation of the keywords for the user in the index structure. Therefore, the update cost which we care in the simulation is the number of edges passed through in the index structure during the update process. According to those parameters in the base case, a comparison of the update cost in our method and Wu and Chen's method is shown in Table 2. From this result, we show that Wu and Chen's method [13] needs more update cost than our method. On the average, our method can reduce about the 68% update cost as compared with Wu and Chen's method.

Table 2. A comparison of the update cost (under the base case).

Methods	The update cost
Wu and Chen's method	63
Our method (reduced %)	20 (68%)

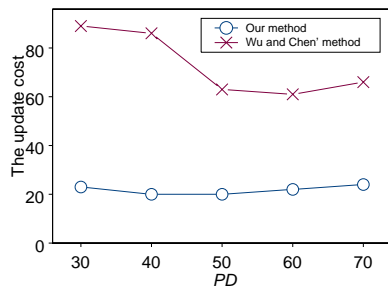


Fig. 10. A comparison of the update cost (under the probability of doing the deletion operation).

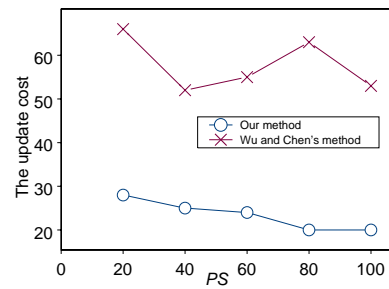


Fig. 11. A comparison of the update cost (under the probability of modifying the short-term interests).

Next, we study the impact of those parameters on the performance. The first parameter that we vary is PD , the probability of doing the deletion operation. The range of PD is set to 30%, 40%, 50%, 60%, 70%. The PS and PL parameters are kept as their base values. Under the change of the value of PD , a comparison of the update cost in our method and Wu and Chen's method is shown in Fig. 10. From this result, we show that Wu and Chen's method [13] needs more update cost than our method. Because the performance result of our method shown in Fig. 10 is close to a straight line, the probability of doing the deletion operation does not influence the performance in our method. By contrast, when the probability of doing the deletion operation is low, Wu and Chen's method needs high update cost. That is, their method needs high update cost when doing the insertion operation. On the average, our method can reduce about the 64% update cost of Wu and Chen's method.

The second parameter that we vary is PS , the probability of modifying the short-term interests. The range of PS is set to 20%, 40%, 60%, 80%, 100%. The PD and PI parameters are kept as their base values. Under the change of the value of PS , a comparison of the update cost in our method and Wu and Chen's method is shown in Fig. 11. From this result, we show that Wu and Chen's method [13] needs also more update cost than our method. Because Wu and Chen's method does not consider whether the keyword is the long-term interest or the short-term interest, the performance result of Wu and Chen's method shown in Fig. 11 does not relate to the probability of modifying the short-term interests. By contrast, as the value of PS increases, the update cost decreases in our method. In fact, the probability of modifying the short-term interests is higher than that of modifying the long-term interests. Therefore, our method can reduce a lot of the update cost, when the probability of modifying the short-term interests is high. On the average, our method can reduce about the 52% update cost of Wu and Chen's method.

4. CONCLUSION

In this paper, to reduce the update cost as needed by Wu and Chen's method [13], we have proposed a data mining-based method for the incremental update. We take the weight of each keyword into consideration. The long-term interests have the weight greater than or equal to the threshold and the short-term interests have the weight less

than the threshold. By making use of the property that the probability of modifying the short-term interests is higher than that of modifying the long-term interests, we can update the short-term interests locally. From our experimental results, we have shown that our method really requires less update cost than Wu and Chen's method.

ACKNOWLEDGMENT

The authors also would like to thank "Aim for Top University Plan" project of NSYSU and Ministry of Education, Taiwan, for partially supporting the research.

REFERENCES

1. G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 17, 2005, pp. 734-749.
2. R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, 1994, pp. 490-501.
3. J. S. Breese, D. Heckerman, and C. Kadie, "Empirical analysis of predictive algorithms for collaborative filtering," in *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence*, 1998, pp. 43-52.
4. B. L. D. Bezerra and F. A. T. Carvalho, "A symbolic approach for content-based information filtering," *Information Processing Letters*, Vol. 92, 2004, pp. 45-52.
5. Y. I. Chang, T. I. Chen, and J. H. Shen, "A large-itemset-based index structure for supporting personalized information filtering on the internet," in *Proceedings of the 8th World Multiconference on Systemics, Cybernetics and Informatics*, 2004, pp. 194-199.
6. E. J. Glover, S. Lawrence, M. D. Gordon, W. P. Birmingham, and C. L. Giles, "Web search – Your way," *Communications of the ACM*, Vol. 44, 2001, pp. 97-102.
7. M. Hammami, Y. Chahir, and L. Chen, "Webguard: A web filtering engine combining textual, structural, and visual content-based analysis," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, 2006, pp. 272-284.
8. J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, "An algorithmic framework for performing collaborative filtering," in *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1999, pp. 230-237.
9. J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, "Evaluating collaborative filtering recommender systems," *ACM Transactions on Information Systems*, Vol. 22, 2004, pp. 5-53.
10. S. Jung, J. Kim, and J. L. Herlocker, "Applying collaborative filtering for efficient document search," in *Proceedings of IEEE/WIC/ACM International Conference on Web Intelligence*, 2004, pp. 640-643.
11. Y. W. Park and E. S. Lee, "A new generation method of an user profile for information filtering on the internet," in *Proceedings of IEEE International Conference on Data Engineering*, 1994, pp. 337-347.

12. D. H. Widyantoro, T. R. Ioerger, and J. Yen, "An adaptive algorithm for learning changes in user interests," in *Proceedings of the 8th International Conference on Information and Knowledge Management*, 1999, pp. 405-412.
13. Y. H. Wu and A. L. P. Chen, "Index structures of user profiles for efficient web page filtering services," in *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, 2000, pp. 644-653.
14. Y. H. Wu, Y. C. Chen, and A. L. P. Chen, "Enabling personalized recommendation on the web based on user interests and behaviors," in *Proceedings of the 11th IEEE Workshop Research Issues in Data Engineering*, 2001, pp. 17-24.
15. T. W. Yan and H. Garcia-Molina, "Index structures for information filtering under the vector space model," in *Proceedings of IEEE International Conference on Data Engineering*, 1994, pp. 337-347.
16. T. W. Yan and H. Garcia-Molina, "Index structures for selective dissemination of information under the Boolean model," *ACM Transactions on Database Systems*, Vol. 19, 1994, pp. 322-364.

Ye-In Chang (張玉盈) received the B.S. degree in Computer Science and Information Engineering from National Taiwan University, Taipei, Taiwan, R.O.C., in 1986, and M.S. and Ph.D. degrees in Computer and Information Science from the Ohio State University, Columbus, OH, in 1987 and 1991, respectively. From August 1991 to July 1999, she joined the faculty of the Department of Applied Mathematics at National Sun Yat-Sen University, Kaohsiung, Taiwan. Since August 1997, she has been a Professor in the Department of Applied Mathematics at National Sun Yat-Sen University, Kaohsiung, Taiwan. Since August 1999, she has been a Professor in the Department of Computer Science and Engineering at National Sun Yat-Sen University, Kaohsiung, Taiwan. Her research interests include database systems, distributed systems, multimedia information systems, and mobile information systems.

Jun-Hong Shen (沈俊宏) received the B.S. degree in Information Engineering from I-Shou University, Kaohsiung, Taiwan, R.O.C., in 1999, and the M.S. degree in Computer Science and Engineering from National Sun Yat-Sen University, Kaohsiung, Taiwan, in 2001. He is currently a Ph.D. candidate in the Department of Computer Science and Engineering at National Sun Yat-Sen University. His research interests include mobile information systems, data mining and peer-to-peer systems.

Tsu-I Chen (陳姿伊) received the B.S. degree in Applied Mathematics and the M.S. degree in Computer Science and Engineering from National Sun Yat-Sen University, Kaohsiung, Taiwan, R.O.C., in 2001 and 2003, respectively. She is currently a mathematics teacher at Tainan Municipal Tu Cheng High School, Taiwan.