A Large-Itemset-Based Method for the Incremental Update of Supporting Personalized Information Filtering on the Internet*

Ye-In Chang, Jun-Hong Shen and Tsu-I Chen
Dept. of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, Taiwan, R.O.C
changyi@cse.nsysu.edu.tw

ABSTRACT

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, 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. Indexing user profiles can reduce the costs of storage space and the processing time for modifying the user profiles. However, when someone's interests are often changed, we must care about the way to provide the low update cost of the index structure. Therefore, in this paper, we propose a large-itemset-based method for the incremental update of the index structure for storing keywords to reduce the update cost. According to our simulation results, our method really can reduce the update cost as needed by Wu and Chen's method.

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 developed on the Web to retrieve useful information. Information filtering is one of the techniques to help users find what they want [2][3][9]. Each user has his (her) profile which stores a set of keywords that can present his (her) interests [6][9]. For a profile to match the document, every word which it contains must be in the document. 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 [4][5][8].

In order to match data with profiles efficiently, a profile index is built on these profiles. Indexing the user profiles can reduce the costs of storage space and the processing time for modifying the user profiles. 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 [7]. Two kinds of index structures are used on the information filtering service, and the users' profiles can be expressed in Vector Space and Boolean models. In the Vector Space model, users' 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. Instead of using the Vector Space model, the user may use Boolean model to specify keywords that he wants in documents received [10].

In [9], 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 [10], 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 wasting much space on storing the index and too much time on matching process in [10], Wu and Chen [7] have proposed four methods to improve the performance in terms of the storage space: index path with path signatures, index graph with path signatures, index path with profile sets, and index graph with profile sets.

Among those methods for information filtering, Wu and Chen's methods [7] can expect to minimize the storage space at the cost of the processing time. Although Wu and Chen's methods [7] can reduce the large storage space as needed by Yan and Garcia-Molina's methods [10], their methods require long time on the matching process. Moreover, when someone's interests are changed, Wu and Chen's data structures [7] need to be restructured from the root. That is, their data structure will be affected globally.

Therefore, in this paper, in order to reduce the update cost as needed by Wu and Chen's methods [7], we propose a large-itemset-based method for the incremental update. We adopt the vector space model. Each keyword can be distinguished the long-term interest which has weight above the threshold from the

* This research was supported in part by the National Science Council of Republic of China under Grant No. NSC95-2221-E-110-101 and by National Sun Yat-Sen University.

short-term interest which has weight below the threshold. Moreover, we also use the idea of the Apriori algorithm [1] to get the large itemset, the long-term interest. Owing to 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 simulation results, our method really can reduce the update cost as needed by Wu and Chen's method [7].

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

2: The Large-Itemset-Based Method for the Incremental Update

Based on Wu and Chen's methods [7], when someone's interests are changed, the profile in their data structures need to be restructured from the root. In this section, we present a large-itemset-based method for the incremental update of the index structure for storing keywords to reduce the update cost.

2.1: The Update 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. Therefore, in the proposed method, we take the weight of each keyword into consideration.

A threshold α is given to distinguish how importance of those keywords is. If the weight of the keyword is larger 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 smaller 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. According to the property of the keywords which are assigned with some weight, we can reduce the update cost in our proposed method.

In our proposed method, we use the idea of the Apriori algorithm [1] to get the large itemset. Because, the large itemsets in our update 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. In our revised Apriori algorithm, the minimum support is dynamically decided by $\frac{\sum_{i=1}^{|Cn|} the\ support\ of\ Ci}{|Cn|}$, where C_n represents the candidate itemset in the *n*'th round.

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)

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

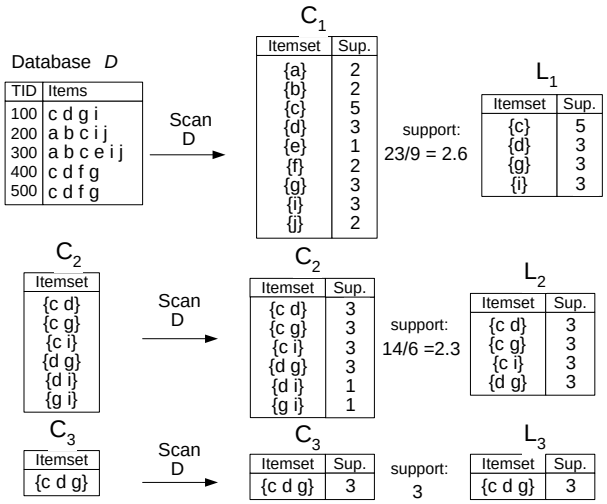

Figure 2: An example of getting the large itemset

Take an example in Figure 1 to illustrate the way to get the large itemset. In Figure 1, there are five profiles which contain a list of keywords with weights and the threshold α = 0.5. Those keywords with the weight < α are shown in Figure 1-(a), and those keywords with the weight $\geq \alpha$ are shown in Figure 1-(b). Because the keyword with the weight $\geq \alpha$ will become the candidate item, we only use those profiles as shown in Figure 1-(b) to be the input data. That is, as shown in Figure 2, the large itemset can be chosen from those profiles shown in Figure 1-(b). The minimum support of C_1 is calculated by $Support = \frac{(2+2+5+3+1+2+3+3+2)}{9} = 2.6$. So, we choose only those itemsets $\{c\}$, $\{d\}$, $\{g\}$, $\{i\}$ which have support larger than 2.6 to be L_1. The final result of the large itemset is $L_3 = \{c, d, g\}$. We choose the large itemset with the largest 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 above 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. Those steps are repeated until no keywords are in the profiles which have the weight of each

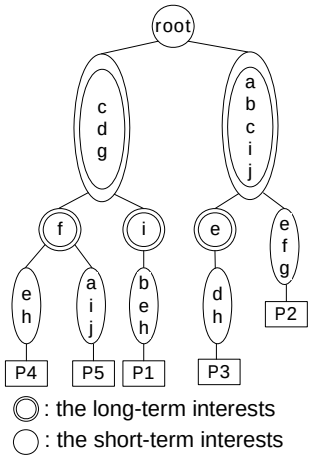

Figure 3: The updatable tree

Profile	Keywords
1	c, d, g, i
2	a, b, c, i, j
3	a, b, c, e, i, j
4	c, d, f, g
5	c, d, f, g

does not contain {c, d, g}

Profile	Keywords
2	a, b, c, i, j
3	a, b, c, e, i, j

(a)

contains {c, d, g}

Profile	Keywords
1	c, d, g, i
4	c, d, f, g
5	c, d, f, g

(b)

Figure 4: Those profiles are divided into two parts according to the large itemset {*c*, *d*, *g*}: (a) profiles P_2 and P_3 do not contain the large itemset; (b) profiles P_1, P_4 and P_5 contain the large itemset.

keyword above the threshold. Finally, we insert the identifier of the profiles and those keywords which have the weight below the threshold to the index structure according to the path from the root that those profiles own by themselves.

Let's use an example shown in Figure 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 Figure 1-(b), which have the weight of each keyword above the threshold to get the large itemset. At the first time, the large itemset is {*c*, *d*, *g*} as shown in Figure 2. We add the large itemset {*c*, *d*, *g*} to the updatable tree, as shown in Figure 3. Then we divide those profiles into two parts: one part does not contain the large itemset {*c*, *d*, *g*} as shown in Figure 4-(a), and the other part contains it as shown in Figure 4-(b).

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 Figure 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 Figure 5-(a), and the other part contains it as shown in Figure 5-(b). Then, we add the keyword {*f*} to the updatable tree, following keywords {*c*, *d*, *g*}. Therefore, there are no keywords with the weight $\geq \alpha$ in profiles P_4 and P_5. So, we add keywords {*e*, *h*} as

Profile	Keywords
1	i
4	f
5	f

does not contain {f}

Profile	Keywords
1	i

(a)

contains {f}

Profile	Keywords
4	f
5	f

(b)

Figure 5: Those profiles are divided into two parts according to the large itemset {*f*}: (a) profile P_1 does not contain the large itemset; (b) profiles P_4 and P_5 contain the large itemset.

shown in Figure 1-(a), which have the weight below the threshold in profile P_4, to the updatable tree, following keyword {*f*}. And we add the identifier of profile P_4 to the updatable tree, following keywords {*e*, *h*}. Similar to the previous step, we also add keywords {*a*, *i*, *j*} as shown in Figure 1-(a), which have the weight below the threshold in profile P_5, to the updatable tree, following keyword {*f*}. And we add the identifier of profile P_5 to the updatable tree, following keywords {*a*, *i*, *j*}. Next, there is only one profile P_1 that contains one keyword *i* which has the weight above the threshold. Therefore, we add keyword *i* to the updatable tree, following keywords {*c*, *d*, *g*}. There are no keywords with the weight $\geq \alpha$ in profile P_1. Then, we add keywords {*b*, *e*, *h*} as shown in Figure 1-(a), which have weight below the threshold in profile P_1, to the updatable tree, following keyword {*i*}. And we add the identifier of profile P_1 to the updatable tree, following 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 Figure 4-(a). Then, we add the large itemset {*a*, *b*, *c*, *i*, *j*} to the updatable tree, following the node of root. 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 add the keyword {*e*} to the updatable tree, following keywords {*a*, *b*, *c*, *i*, *j*}. Next, we add keywords {*d*, *h*} as shown in Figure 1-(a), which have the weight below the threshold in profile P_3, to the updatable tree, following keyword {*e*}. Next, we add the identifier of profile P_3 to the updatable tree, following keywords {*d*, *h*}. Finally, similar to the previous steps, we add keywords {*e*, *f*, *g*} as shown in Figure 1-(a), which have the weight below the threshold in profile P_2, to the updatable tree, following keywords {*a*, *b*, *c*, *i*, *j*}. And we add the identifier of profile P_2 to the updatable tree, following keywords {*e*, *f*, *g*}. Therefore, the final result for the input shown in Figure 1 is shown in Figure 3.

2.2: The Update Process

According to our large-itemset-based method for the incremental update as described above, we can reduce the update cost as needed by Wu and Chen's methods [7]. Owing to 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 below the threshold to the leaf nodes of the tree, we can update the short-term interests locally. For example, the weight of keyword *f* in profile P_2 is 0.1, it is one of the short-term interests which have the higher probability to be changed over a short period. According to the index tree as shown in Figure 3, if the user with profile P_2 is not interested in keyword *f*, we can remove 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*}.

For the deletion of the long-term interest in profile P_i, if the keyword, *l_key*, of the deletion is contained only in the node leading to P_i, it could be deleted directly. On the other hand, if keyword *l_key* is contained in the node leading to P_i and the other profiles, keywords in this node and its child nodes should be reallocated. That is, the reallocation is locally operated.

If the user with profile P_2 is interested in keyword *d* over a short period, we will insert keyword *d* to the node which contains the short-term interest. That is, keyword *d* is inserted to the node containing {*e, g*}, as shown in Figure 3.

For the insertion of the long-term interest in profile P_i, the keyword, *l_key*, of the insertion will be put into the last long-term node leading to P_i. Moreover, if keyword *l_key* is exclusively belonging to P_i, a new long-term node containing this keyword should be created, and attached to the original last long-term node. Furthermore, if the siblings of the last long-term node containing *l_key* have *l_key*, these nodes should be reallocated. That is, keywords in these nodes should be locally reallocated.

3: Performance

In this section, we make a comparison with Wu and Chen's method [7] and our proposed method.

3.1: The Simulation Model

We generated synthetic profiles to evaluate the performance [10]. 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*. The first profile is called "base profile." In our assumption, the users with similarity 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 word in the new profile, there is a probability *Q* that it is the same as the corresponding word in the base profile. If it is not, then the keyword in the new profile is picked at random from the set of keywords *D*. There are no duplicated keywords in the profile. Hence, by varying *Q* from 0 to 1, we can control the similarity among the profiles. If *Q* is 0, the keywords in all profiles are randomly chosen from the set of keywords *D*.

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

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

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

3.2: Simulation Results

The data which we choose to update contains four parameters: $N = 500$, $K = 5$, $D = 50$, and $Q = 80\%$. That is, we cluster 500 users with similarity interests into the same group. The length of all profile is 5. The set of keywords is composed of 50 keywords. Moreover, we choose 80% to decide the similarity among profiles.

In our simulation, four parameters and their default settings are listed in Table 1. Owing to that the update process contains the deletion operation and the insertion operation, we can observe the impact of the ratio in the deletion and insertion operation for the update cost. Moreover, we can adjust the ratio of the probability of modifying the short-term interests and that of modifying the long-term interests. First, we define a base case, (*PD, PI*) = (50%, 50%) and (*PS, PL*) = (80%, 20%).

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 which we have passed through in the index structure. According to those parameters in the base case, a comparison of the update cost in Wu and Chen's method and our method is shown in Table 2. Note that all our experimental results are the average of 100 executions. From this result, we show that Wu and Chen's method [7] needs more update cost than our method. On the average, our method can reduce about 68% update cost as compared with Wu and Chen's method.

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 number of *PD*, a comparison of the update cost in Wu and Chen's method and our method is

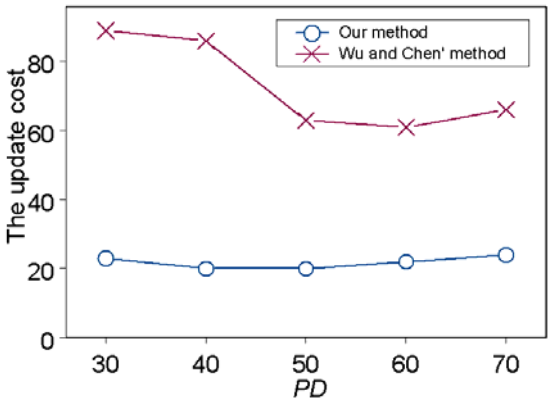

Figure 6: A comparison of the update cost (under the probability of doing the deletion operation)

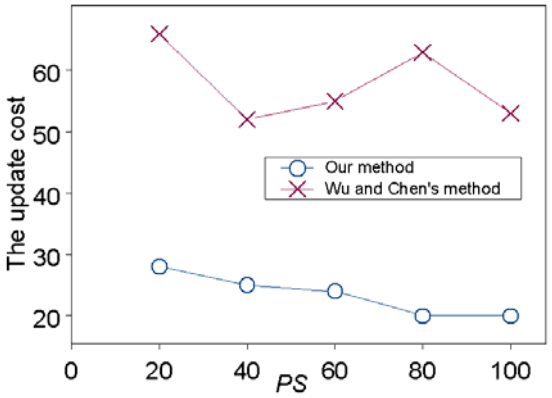

Figure 7: A comparison of the update cost (under the probability of modifying the short-term interests)

shown in Figure 6. From this result, we show that Wu and Chen's method [7] needs more update cost than our method. Because the line of our method shown in Figure 6 is close to a straight line, the probability of doing the deletion operation does not influence the performance in our method. But, in Wu and Chen's method, when the probability of doing the deletion operation is small, they need more update cost. That is, their method needs larger update cost when they do the insertion operation. On the average, our method can reduce about 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 number of *PS*, a comparison of the update cost in Wu and Chen's method and our method is shown in Figure 7. From this result, we show that Wu and Chen's method [7] needs also more update cost than our method. Because in Wu and Chen's method, they do not consider whether the keyword is the long-term interest or the short-term interest, the pseudo code of Wu and Chen's method shown in Figure 7 does not relate to the probability of modifying the short-term interests. But, 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 interest. Therefore, our method can reduce more update cost, when the probability of modifying the short-term interests is large. On the average, our method can reduce about 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 methods [7][10], we have proposed the large-itemset-based method for the incremental update. We take the weight of each keyword into consideration. The long-term interests have the weight above the threshold and the short-term interests have the weight below the threshold. Owing to 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 simulation, we have shown that our method really requires smaller update cost than Wu and Chen's method.

REFERENCES

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules in Large Databases," *Proc. of the 20th Int. Conf. Very Large Data Bases*, pp. 490-501, 1994.

[2] 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, No. 12, pp. 97-102, Dec. 2001.

[3] M. Hammami, Y. Chahir, and L. Chen, "Webguard: A Web Filtering Engine Combining Textual, Structural, and Visual Content-Based Analysis," *IEEE Trans. on Knowledge and Data Eng.*, Vol. 18, No. 2, pp. 272-284, Feb. 2006.

[4] S. Jung, J. Kim, and J. L. Herlocker, "Applying Collaborative Filtering for Efficient Document Search," *Proc. of IEEE/WIC/ACM Int. Conf. on Web Intelligence*, pp. 640-643, 2004.

[5] Y. W. Park and E. S. Lee, "A New Generation Method of an User Profile for Information Filtering on the Internet," *Proc. of IEEE Int. Conf. on Data Eng.*, pp. 337-347, 1994.

[6] D. H. Widyantoro, T. R. Ioerger and J. Yen, "An Adaptive Algorithm for Learning Changes in User Interests," *Proc. of the 8th Int. Conf. on Information and Knowledge Management*, pp. 405-412, 1999.

[7] Y. H. Wu and A. L. P. Chen, "Index Structures of User Profiles for Efficient Web Page Filtering Services," *Proc. of the 20th IEEE Int. Conf. on Distributed Computing Systems*, pp. 644-653, 2000.

[8] Y. H. Wu, Y. C. Chen and A. L. P. Chen, "Enabling Personalized Recommendation on the Web Based on User Interests and Behaviors," Proc. of the 11th IEEE Workshop Research Issues in Data Eng., pp. 17-24, 2001.

[9] T. W. Yan and H. Garcia-Molina, "Index Structures for Information Filtering Under the Vector Space Model,"

Proc. of IEEE Int. Conf. on Data Eng., pp. 337-347, 1994.

[10] T. W. Yan and H. Garcia-Molina, "Index Structures for Selective Dissemination of Information Under the Boolean Model," *ACM Trans. on Database Systems*, Vol. 19, No. 2, pp. 322-364, Nov. 1994.