Index Structures of User Profiles for Efficient Web Page Filtering Services

Yi-Hung Wu and Arbee L. P. Chen✞
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan 300, R.O.C.
E-mail: alpchen@cs.nthu.edu.tw

Abstract

Searching information from the WWW efficiently and effectively has become a very important issue. In this paper, we apply the information filtering concept to finding good matches of web pages for what the users need. Furthermore, we propose four new methods for indexing the users' information needs and show their efficiency in comparison with two well-known approaches.

Keywords: *information filtering, index structure, user profile, web page, signature.*

1. Introduction

Owing to the booming development of the WWW, users have to face a variety and a large number of web pages and often waste much time on searching or browsing the web pages. To alleviate the difficulty, many tools have been developed and used on the WWW. They can be classified into two types [10]:

Search engine: a centralized processing approach. A search engine often collects the representative descriptions of the web pages before the users' information needs appear. Two approaches are used to get the descriptions of the web pages. One is to provide a registration form for a web page author to describe what the web page contains, e.g., *Yahoo* [25]. The other is to fetch the web pages and to extract the representative information periodically, e.g., *Google* [2] and *WebCrawler* [16]. Most of the systems extract the keywords from web pages. Some systems also consider the authors' names or the URLs, e.g., *HotBot* [6]. After that, the descriptions of the web pages are properly organized as an index for future retrieval.

To find the most related web pages against the index, the user may use two basic approaches: a subject guide or a keyword search. A subject guide is useful in browsing the general topics, while for the specific topics, a keyword search is usually used [8]. Some of the popular search engines are *InfoSeek* [7], *Lycos* [11], *Northern Light* [14] and *Alta Vista* [20]. The detailed comparisons among the search engines can be found in [18] and [22].

Meta-search engine: a distributed processing approach. In contrast to the search engine, a meta-search engine collects the information of the web pages when the users make their requests. The descriptions of the web pages are distributed in a set of search engines. A meta-search engine only keeps the summary information of the individual search engines, such as the index size, the frequency of updates, the search capability, and the expected response time. When the user issues a query, a meta-search engine selects the best set of search engines to answer the query. Some systems also consider the way to use the selected search engines. After that, the query is sent to individual sites for processing and the results are integrated. Some of the meta-search engines are *SavvySearch* [5], *MetaCrawler* [19], *Discover* [21], and *HyPursuit* [23].

The search engines process the users' requests in a centralized way. On the contrary, the distributed processing techniques are employed in the meta-search engines. Obviously, the performance of the search engines will get worse if the number of web pages grows rapidly [9]. As for the meta-search engines, there exists a tradeoff between the processing time and the quality of query results. If the summary information of the individual search engines is rich, the quality of query results will be promoted but the processing time worsened. Therefore, a compromise between satisfying the users' requests and promoting the system performance should be made.

In this paper, we apply the concept of *information filtering* to find good matches between the web pages and the users' information needs. A typical architecture of the information filtering service is illustrated in Figure 1. Users first give descriptions about what they need, which are called *user profiles*, to start the services. A *profile index* is built on these profiles. A series of incoming web pages will be put into the *matching process*. Each incoming web page is represented in the same form of the user profile. In this way, the users who are interested in an

✞Contact author

incoming web page can be identified by comparing the descriptions of the web page with each user profile. At last the web page will be recommended to the users whose profiles belong to the *filtered result*.

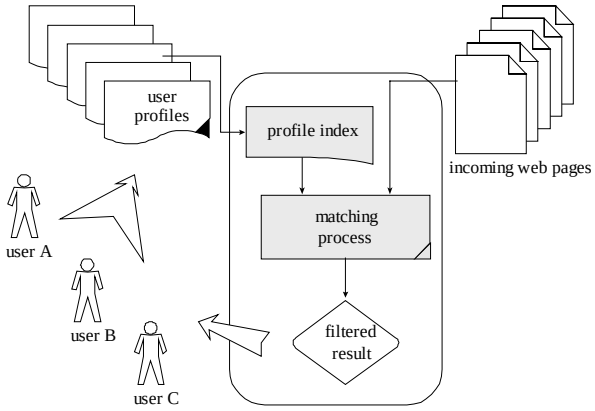

Figure 1. An information filtering service

Compared with the search engine and the meta-search engine, the filtering service can reduce the efforts on building the index of the web pages. Therefore, the filtering service can perform as usual no matter how the number of web pages grows.

Many research results on the information filtering services have been presented in the literature [4]. Most of the works focus on two topics. One is to provide an efficient method for indexing and matching the user profiles automatically, such as [26], [27], and [28]. The other is to devise an effective method for creating and modifying the user profiles, such as [13] and [15]. In addition, [1], [12], [17] and [29] also consider the issues on the system design and demonstrate the feasibility of the information filtering services.

In our approach, a proxy server 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. In a web page filtering service, users first specify their profiles as a set of keywords, named the *keyword profiles*. All the keyword profiles are stored together and organized into an index. Similarly, an incoming web page is represented by the set of keywords extracted from it. After the comparisons between these sets of keywords are made, the users who are interested in the incoming web page can be identified.

Because the proxy servers only fetch the web pages based on the users' requests, the scope of incoming web pages is limited. Another mechanism is provided to broaden the scope of the incoming web pages. This mechanism allows the users to specify the *URL profiles* [3]. A URL profile is a partial specification of URLs, which can be used to represent a set of URLs that the users may be interested. For instance, a URL profile can be *http://db.nthu.edu.tw/** or *http://db.*.edu.tw/* (the symbol * stands for the *don't care* condition). When a web page is fetched by the proxy server, the complete URLs contained in the web page, such as the URLs specified in the hyperlinks, will be collected. These URLs will be compared with the URL profiles to find the matches. For example, the URLs *http://db.nthu.edu.tw/alp/alpchen.html* and *http://mx.nthu.edu.tw/~secwww/president/index.html* are both the matches for the URL profile *http://*.nthu.edu.tw/**. By considering the matched URLs as users' requests, the proxy server will fetch more web pages. In this way, the scope of the incoming web pages can be broadened.

As described above, a critical issue of the information filtering service is how to index the user profiles for an efficient matching process. Several approaches to profile indexing were proposed in [27] and [28]. In this paper, we will propose new indexing methods, which can reduce the costs of storage space and the processing time for modifying the user profiles.

The rest of the paper is organized as follows. In section 2, two well-known indexing methods are introduced. Section 3 presents four new indexing methods. The performance analysis and comparison among these methods are shown in section 4. In section 5, we give the conclusions and future work.

2. Related approaches

Several indexing methods for keyword profiles have been proposed in [27] and [28]. In the following subsections, we first describe an example that will be used to illustrate the structure of profile index and the matching process. Afterwards, two well-known approaches will be introduced.

2.1. An example

Profile	Keyword
P1	a b
P2	a d
P3	a d e
P4	b f
P5	c d e f
Example Page	a b c f

Table 1. An example

This paper will consider the tables shown in Table 1 as an example for profile indexing. A set of profiles and an example page are presented with the associated sets of keywords. The alphabets in the Keyword column denote the keywords specified in the profiles or extracted from the example page. For instance, the profile P_1 is represented by two keywords a and b. That is, the user specifying P_1 is interested in the web pages which contain the two keywords. Consider the example page that contains four keywords a, b, c, and f. After the matching process terminates, only P_1 and P_4 will belong to the

filtered result because both the keyword sets {a b} and {b f} are contained in {a b c f}.

Among the previous works on profile indexing, the approaches proposed by Yan and Garcia-Molina [26] [27] [28] [29] are well recognized. Two basic methods used in [27] will be introduced in the following two subsections. Due to the lack of space, we omit the detailed algorithms and the complexity analyses from this paper.

2.2. The counting method

In the first method, all the profiles containing a specific keyword are put together in the form of *inverted lists*. To illustrate, the right-hand side in Figure 2 shows the inverted lists derived from all the profiles in Table 1. Note that P_1 appears in both the inverted lists of a and b, indicating that P_1 contains these two keywords. Similarly, P_1, P_2, and P_3 all appear in the inverted list of a, indicating that all the three profiles contain the keyword a.

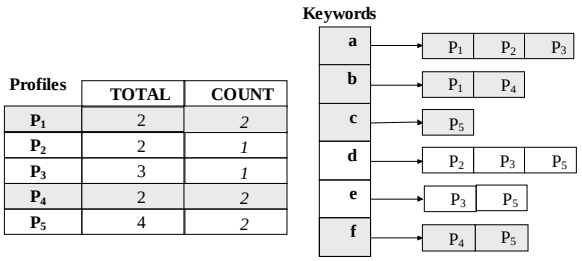

Figure 2. The counting method

At the left-hand side in Figure 2, two auxiliary arrays, called TOTAL and COUNT, are constructed for the matching process. TOTAL is used to keep the number of keywords in the profile. For example, P_2 appears in the inverted lists of a and d, and the corresponding entry in TOTAL is 2. COUNT is used to keep the number of occurrences for a profile during traversing the inverted lists in the matching process. Before the matching process for a web page starts, all the entries in COUNT are set to 0. After that, the inverted lists of the keywords extracted from the web page will be traversed one by one. Consider the example page in Table 1. The inverted lists, which are shaded at the right-hand side in Figure 2, will be traversed during the matching process.

During the traversal of the inverted lists, the number of occurrences for each profile will be accumulated in the corresponding entries of COUNT. At last, a profile will be regarded as a match if the value of the COUNT entry equals the value of the corresponding entry in TOTAL. Consider the shaded blocks at the left-hand side in Figure 2. P_1 is a match because the values in the corresponding entries of TOTAL and COUNT are equal. Similarly, P_4 is also a match.

2.3. The tree method

As described above, a profile may be repeatedly stored in the inverted lists. Therefore, the storage space and the matching time may get worse with the increase of repetitions. The other approach considers a tree structure. The internal nodes in a tree structure can be shared by their descendants. In this way, the repetitions of storing the same information can be reduced to a certain extent.

A tree derived from a set of profiles is called an *index tree*, which is composed of nodes and edges. There are two types of nodes: one keeps a keyword and the other records a profile, which are called the *k-node* and *p-node*, respectively. The root is a pseudo node. As shown in Figure 3, all the internal nodes are k-nodes, while all the leaf nodes are p-nodes. Furthermore, a keyword may appear in several k-nodes, while a profile can only appear in one p-node. In this way, an *external path* can be defined as a path that starts from the root, passes a consecutive sequence of k-nodes, and ends with a p-node. In the index tree, each profile is represented by an external path. For example, the profile P_1 is represented by the external path that starts from the root and ends with the p-node P_1. The keywords on the k-nodes, passed by this path, are the same as the keywords specified in P_1. Therefore, a profile can be inserted by traversing and creating the corresponding external path. Obviously, the repetitions of the k-nodes can be further reduced by promoting the most common used ones. For instance, the keyword f appears in both the profile P_4 and P_5. Therefore the promotion of the k-node f to the first layer can save the space of a k-node.

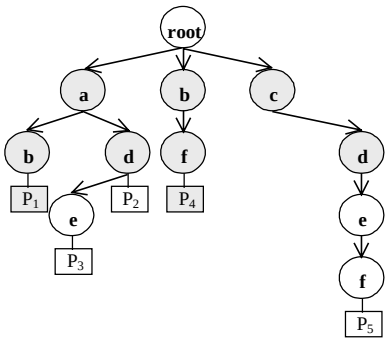

Figure 3. An index tree

To find the matches for a web page, a traversal of the index tree from the root has to be conducted. That is, all the external paths, which may cover a set of profiles, have to be examined. If the keyword in a k-node is not contained in the set of keywords extracted from a web page, all the external paths containing this k-node will not cover any matched profiles. For example, the keyword d is not contained in the keyword set {a b c f} of the example page. Therefore, all the profiles P_2, P_3, and P_5 covered by the external paths containing k-node d will not be the matches. Furthermore, only the k-nodes and p-nodes, which are shaded in Figure 3, have to be accessed during the matching process.

3. Our approaches

Since the counting method and the tree method result in the repetitions of profiles and keywords, respectively, these methods waste too much space on storing the index and too much time on matching. On the contrary, our methods focus on removing the repetitions in the profile index. Therefore, we can expect to minimize the storage space with acceptable costs for the matching process.

In this section, we present four new approaches to profile indexing. The example given in Table 1 will be used to illustrate all these approaches. Due to the lack of space, we omit the detailed algorithms and the complexity analyses from this paper. In [24], the reader can find the inserting algorithm, the matching algorithm, and the complexity analysis for each approach.

3.1. Method 1: index path with path signatures

This method has an index structure which looks like a linear path, named an *index path*, as illustrated in Figure 4. There are also two types of nodes: k-nodes and p-nodes. Because a keyword and a profile can only be put in a k-node and a p-node, respectively, this method guarantees that no repetition will occur in the index.

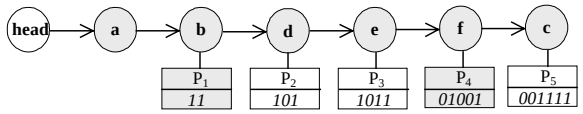

Figure 4. An index path with path signatures

The index path starts at the head, which is a pseudo node, and consists of a series of k-nodes. Among the keywords contained in a profile, there exists a keyword whose associated k-node is located at the rightmost position in the index path. We put the p-node associated with the profile under this k-node. For example, P_1 contains two keywords a and b. Moreover, node b is on the right side of node a in the index path. Therefore node P_1 is put under node b as shown in Figure 4. In this way, different p-nodes may be put under the same k-node.

Each profile is represented by a particular signature, called a *path signature*, in the associated p-node. A path signature is in the form of a bit string, indicating the locations of the associated k-nodes in the index path from left to right. In the bit string, two numbers 1 and 0 are used to denote whether the keyword of a k-node is contained in the profile or not. For example, the path signature of P_1 is 11, indicating that P_1 is associated with the leftmost two nodes a and b in the index path.

To insert a profile, we only have to traverse the index path from the head and build the path signature based on the comparisons with the k-nodes. After all the associated k-nodes are passed, a p-node for the profile will be created under the rightmost k-node. As shown in Figure 4, P_3 is put under the node e because all the other associated k-nodes, such as node a and d, appear before node e.

As for the matching process, the keywords extracted from a web page are compared with the k-nodes during traversing the index path from the head. A path signature for the web page can also be built as a bit string. However, the path signature for the web page may be recomputed as the traversal of the index path moves to different k-nodes. For instance, the path signature of the example page is changed from 1100 to 11001 when the traversal of the index path moves from node e to node f. In this way, the matched profiles can be found by comparing the path signature of the web page with the one of each p-node under the matched k-nodes when the matching process proceeds.

Considering the example page in Table 1, its path signature equals 11 at node b and 110 at node d. Therefore P_1 is a match, but P_2 is not. Because the example page is associated with the k-node c, which is at the end of the index path, all the k-nodes have to be accessed during the matching process and are shaded in Figure 4. After that, only P_1 and P_4 are the matches, which are also shaded in Figure 4.

3.2. Method 2: index graph with path signatures

Method 1 adopts a linear structure to reduce the storage space to a certain degree. However, it may waste too much time on traversing the whole path. To overcome the difficulty, a directed graph structure without any cycle, named an *index graph*, is proposed to store the profiles. The graph in Figure 5 illustrates the index derived from the set of profiles in Table 1. There are also two types of nodes in the graph: k-nodes and p-nodes. Because a keyword and a profile can only be put in a k-node and a p-node, respectively, this method also guarantees that no repetition will occur in the index.

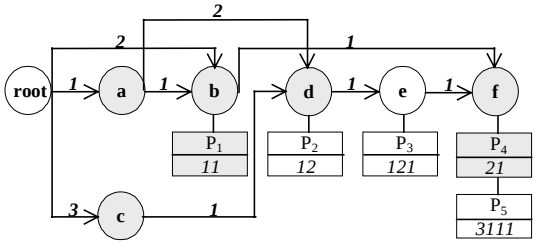

Figure 5. An index graph with path signatures

In an index graph, a link that points from a k-node to another indicates a direction for traversing the graph. The root is regarded as the starting node of the traversal. All the links are associated with a number such that the numbers of any two links from the same k-nodes are different. For example, there are two links from node a in Figure 5, which are associated with numbers 1 and 2 respectively. Furthermore, the sequence of numbers

associated with the links in a path from the root can be used to specify the path. No different paths can be specified by the same sequence of numbers. For example, consider the two paths ending at node d. The one passing node a is specified as 12, and the other passing node c as 31. Obviously, no other paths can be specified by 12 or 31.

Each profile can be represented by a path from the root, which only passes all the associated k-nodes of the profile. Therefore, the sequence of numbers for specifying a path can also be used to represent a profile. For instance, the path root→a→d→e is specified as 121, which can also be used to represent P_3. In this way, each profile will be associated with a *path signature*, which is formed by a sequence of numbers for the corresponding path in the index graph.

To insert a profile, we have to traverse the index graph from the root and build the corresponding path. After all the associated k-nodes are linked into a path, a p-node for the profile is created under the ending k-node of the path. As shown in Figure 5, P_3 is put under node e, which is at the end of the path root→a→d→e. Moreover, the path signature is computed by gathering the sequence of numbers that appear on the links in the path. For example, the path signature of P_3 equals 121 because the corresponding path consists of three links, i.e. root→a, a→d, and d→e.

As for the matching process, the keywords extracted from a web page are compared with the k-nodes during traversing the index graph from the root. A path signature for the web page can also be built as a sequence of numbers. However, the path signature for the web page may be recomputed as the traversal of the index graph produces different paths from the root. Consider the example page in Table 1. The path signature equals 111 when the traversal moves to node f along the path root→a→b→f, while it becomes 21 if the traversal produces the path root →b→f. In this way, the matched profiles can be found by comparing the path signature of the web page with the one of each p-node under the matched k-nodes when the matching process proceeds.

If the keyword on a k-node is not contained in the set of keywords extracted from the web page, all the paths starting from this k-node can be ignored. For instance, the paths starting from node d can be ignored considering the example page. In this way, all the k-nodes except node e have to be accessed during the matching process and are shaded in Figure 5. After that, only P_1 and P_4 are the matches, which are also shaded in Figure 5.

3.3. Method 3: index path with profile sets

In method 1 and method 2, we attach a path signature to each p-node for identifying the matched profiles. However, both of them may waste too much time on comparing with the keywords in the k-nodes, which are not associated with any matched profiles. In this method and the next, we will introduce the ways to reduce the overheads and improve the overall performance. First of all, we refine the index path used by method 1.

As shown in Figure 6, the new index structure is similar to the one used by method 1 except that each k-node is associated with a set of profile identifiers, named a *profile set*. For each k-node, a profile set can be used to indicate the set of profiles containing the keyword in it. For example, the profile set of node a in Figure 6 equals $\{P_1P_2P_3\}$, indicating that all these profiles contain the keyword a. In addition, we put a profile identifier instead of a path signature in each p-node.

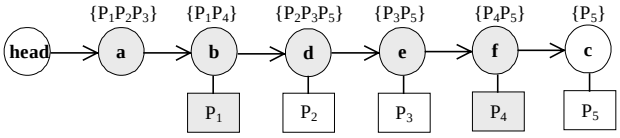

Figure 6. An index path with profile sets

As described in method 1, different p-nodes may be put under the same k-node. Furthermore, this method also guarantees that no repetition will occur in the index because a keyword and a profile can only be put in a k-node and a p-node, respectively. To insert a profile, we can follow the same algorithm of method 1 except the operations on the k-nodes. If the keyword in a k-node is contained in the profile, the profile identifier will be added into the profile set of the k-node. After all the associated k-nodes are passed, a p-node of the profile will be created under the rightmost k-node. As shown in Figure 6, P_3 is contained in the profile sets of node a, d, and e. Moreover, the p-node for P_3 is put under node e, which is the rightmost one.

As for the matching process, we can also follow the same algorithm of method 1 except the use of path signature. First of all, we define a *candidate profile* as a profile, which may be a match. Moreover, a *candidate set* is defined as the set of all candidate profiles during the matching process. For example, all the profiles are included in the candidate set when the matching process starts. In addition, we define a *target set* as the intersection of the candidate set and the profile set of a k-node. As method 1 describes, the keywords extracted from a web page are compared with the keyword of each k-node during traversing the index path from the head. Furthermore, the candidate set is also compared with the profile set of each k-node. After these comparisons are made, two kinds of results will be obtained.

- ✧ If the keyword of a k-node is a match, all the p-nodes under this k-node may be matches. Furthermore, we can regard a p-node as a match if it contains a profile identifier belonging to the target set. After that, all the matched profiles are removed from the candidate set before the matching process continues.
- ✧ On the contrary, all the p-nodes under a k-node are not matches if the keyword of the k-node is not a match.

Furthermore, a profile containing the keyword can not be a match. Therefore, all the profiles in the target set can also be removed from the candidate set. The following lemma explains this property.

Lemma 1.

Assume that a profile P_j contains a keyword X and KEY_{web} is the set of all keywords extracted from a web page Y. Then, **$X \notin KEY_{web} \Rightarrow P_j$ is not a match of Y.**

Consider the example in Table 1. The candidate set is set to $\{P_1P_2P_3P_4P_5\}$ when the matching process starts. At node a, the target set equals $\{P_1P_2P_3\}$, while there is no p-node under it. The candidate set keeps all the profiles. Similarly, the target set is $\{P_1P_4\}$ at node b. Therefore P_1 is a matched profile and the candidate set becomes $\{P_2P_3P_4P_5\}$. At node d, the target set is $\{P_2P_3P_5\}$ and the keyword is not contained in the keyword set of the example page. By lemma 1, all the three profiles are removed from the candidate set (i.e. $\{P_4\}$). In this way, another matched profile P_4 can be found at node f. Moreover, the process also terminates at node f because all the associated k-nodes have been passed.

3.4. Method 4: index graph with profile sets

In this method, we apply the concept of the profile set to the index graph. As shown in Figure 7, the new index structure is similar to the one used by method 2 except that each k-node is associated with a profile set. As described in method 3, the profile set of a k-node indicates the set of profiles containing the keyword in it. In addition to the profile identifier, a p-node is also associated with the number of keywords contained in the profile, named a *path length*. For example, the p-node of P_1 in Figure 7 records its profile identifier and a number 2, which stands for the path length of P_1. Note that no sequence number is associated with the link.

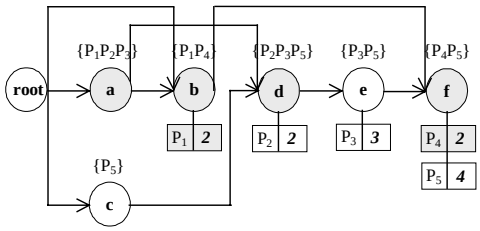

Figure 7. An index graph with profile sets

As described in method 2, different p-nodes may be put under the same k-node. Furthermore, this method also guarantees that no repetition will occur in the index because a keyword and a profile can only be put in a k-node and a p-node, respectively. To insert a profile, we can follow the same algorithm of method 2 except the operations on the k-nodes. If the keyword in a k-node is contained in the profile, the profile identifier will be added into the profile set of the k-node. After all the associated k-nodes are passed, a p-node for the profile will be created under the ending k-node of the traversed path. As shown in Figure 7, P_3 is contained in the profile sets of node a, d, and e. Therefore, the p-node for P_3 is put under node e, which is at the end of the path root→a→d→e. Moreover, a p-node is also associated with the path length of the profile. For example, the path length of P_3 equals 3.

As for the matching process, we can follow the same algorithm of method 2 except the use of path signature. Furthermore, this method also adopts the concepts of the candidate set and the target set defined in method 3. During traversing the index graph, a target set of a k-node is derived by the intersection of the candidate set and the profile set of the k-node. Moreover, this target set can be regarded as the candidate set for the next k-node along the same path if the comparison of keywords succeeds. Take the index graph in Figure 7 as an example. The candidate set is set to $\{P_1P_2P_3P_4P_5\}$ when the matching process starts. At node a, the target set is $\{P_1P_2P_3\}$. Because the keyword a is contained in the keyword set of the example page, the candidate set for node b becomes $\{P_1P_2P_3\}$. On the contrary, the target set can also be used to reduce the candidate set if the comparison of keywords fails. By lemma 1, the profiles shown in the target set can be removed from the candidate set.

A path length for the web page is accumulated during traversing the index graph. In this way, we can find the matched profiles if all the following conditions are satisfied:

- ✧ The keyword of the k-node is contained in the keyword set of the example page.
- ✧ The profile identifier associated with the p-node is contained in the target set of this k-node.
- ✧ The path length associated with the p-node is equal to the path length of the web page at this k-node.

Consider the example page in Table 1 and the index graph in Figure 7. As described above, the candidate set of node b becomes $\{P_1P_2P_3\}$ considering the path root→a→b. Therefore, P_1 belongs to the filtered result because all the three conditions are satisfied. In this way, another matched profile P_4 can also be found by traversing the path root→b →f. At last, the matching process terminates when the candidate set for the web page is set to ∅.

4. Comparisons

In Table 2, some notations are listed with the associated values based on the example in Table 1. For instance, P refers to the set of all profiles and the notation |P| stands for the number of profiles, which equals 5 in the example. Moreover, only six distinct keywords are used in the example. Therefore the notation |K|, which means the number of distinct keywords, equals 6. In the following, these notations will be used to denote the results of the complexity analyses for the indexing methods. The complexity analyses are based on the average case.

Symbol	Value	Description
P	|P|=5	The set of all profiles
K	|K|=6	The set of all distinct keywords
n	2.6	Average number of keywords specified in a profile
f	2.16	average number of profiles in which a specific keyword is contained
m	4	average number of keywords extracted from a web page

Table 2. Some notations

As shown in Table 3, the characteristics of the six approaches to profile indexing are listed under six criteria.

- **Duplication of information**: Both the counting method and the tree method incur the repetitions of the same information in the indexes. In the inverted lists constructed by the counting method, each profile will be duplicated as many times as the number of keywords it contains. For the index tree constructed by the tree method, two profiles may share some keywords if they happen to contain the same keywords that compose the same path from the root. However, there are still many keywords duplicated in the index tree. In our approaches, all the four methods guarantee that there is no duplication of information.
- **Sorting of keywords**: Only the tree method and the graph based methods need to keep the order of keywords shown in the indexes. That is, the keywords are inserted to the indexes in a specific order, such as the alphabetical order. In the index tree, the order for inserting keywords has a significant impact on the number of duplicated keywords. Therefore, the performance of the tree method will degenerate if the order for inserting keywords results in more duplicated keywords. On the other hand, the order for inserting keywords has no impact on the performance of the graph based methods. All we have to do is to insert keywords in an order, which can be chosen arbitrarily. Regarding the other three methods, the keywords can be inserted to the index in an arbitrary order. Therefore the overheads for sorting the keywords in advance can be saved.
- **Space to store the profile index**: As shown in Table 3, the counting method costs the most to store the whole index. Further, the tree method also wastes much space on the repetitions of keywords. All of our approaches guarantee that there is no duplication of information in the indexes. Therefore, we claim that the proposed approaches can save much storage space as compared to the counting method and the tree method.
- **Time to insert or delete a profile**: From Table 3, we have that method 2 is similar to the counting method and the tree method regarding the time to insert or delete a profile. The other methods spend less time on the insertion or deletion of a profile. Especially method 3 and method 4 can provide efficient ways to find unused k-nodes.
- **Time to match the profile index**: As shown in Table 3, the costs to match the profile indexes are similar for all the methods except the counting method. The extra expenditure of the counting method is to find a set of matched profiles by checking all the entries in TOTAL and COUNT.
- **Time to modify a profile**: The time complexity of modifying the content of a profile is the same as the one of inserting or deleting a profile. That is, method 1, method 3 and method 4 perform better than the other methods.

5. Conclusions

In this paper, we consider a web page filtering service utilizing the proxy servers on the WWW. Four index structures of user profiles and the related algorithms for the web page filtering service are proposed. Moreover, the comparisons with other representative approaches to profile indexing are discussed according to different criteria of performance. To sum up, our indexing methods take advantage of both the graph structure and the signature to save the storage space and reduce the overheads of profile updates. Therefore, we claim that the proposed approaches to profile indexing are effective and efficient for the web page filtering services.

In general, the performance evaluation can be done in three ways: formal analysis, simulation modeling, and experiments on real data. In this paper, we only obtain a primitive result of the performance analysis. A simulation model and the test bed for the experiment can be considered in the future. A web page filtering service consists of three phases: input of user profiles, filtering of incoming web pages, and dissemination of the filtered result. An effective and efficient way to disseminate and display the filtered result is urgently required in a low bandwidth mobile environment.

To provide better flexibility and precision on specifying the user profiles, we will also consider more types of query predicates and different kinds of matching functions in the future. These works include the multi-field queries, truncation or thesauruses of keywords, and vector-space queries.

Reference

[1] Balabanovic, M., "An adaptive web page recommendation service," *Proceedings of International Conference on Autonomous Agents*, 1997.

[2] Brin, S. and L. Page, "The anatomy of a large-scale hypertextual Web search engine," *Proceedings of seventh International WWW Conference*, 1998.

[3] Chen, J. P., Y. H. Wu, and A. L. P. Chen, **"**Index structures of URL profiles for efficient web page filtering services," *NTHU Technical Report*, 1999.

[4] *Communications of the ACM*, 35(12), 1992.

[5] Dreilinger, D. and A. E. Howe, "Experiences with selecting search engines using metasearch," *ACM Transactions on Information Systems*, 15(3), 1997.
[6] HotBot, *http://www.hotbot.com/*, 1999.
[7] InfoSeek, *http://www.infoseek.com/*, 1999.
[8] Kansas City Public Library, "Introduction to search engines," *http://www.kcpl.lib.mo.us/search/srchengines.htm*, 1999.
[9] Kosmynin, A., "From bookmark managers to distributed indexing: an evolutionary way to the next generation of search engines," *IEEE Communication Magazine*, June 1997.
[10] Lawrence, S. and C. L. Giles, "Searching the Web: general and scientific information access," *IEEE Communication Magazine*, January 1999.
[11] Lycos, *http://www.lycos.com/*, 1999.
[12] Mostafa, J., and et al., "A multilevel approach to intelligent information filtering modeling, system, and evaluation," *ACM Transactions on Information Systems*, 15(4): 368-399, October 1997.
[13] Moukas, A., "Amalthaea: information discovery and filtering using a multi-agent evolving ecosystem," *International Journal of Applied Artificial Intelligence*, 1997.
[14] Northern Light, *http://www.northernlight.com/*, 1999.
[15] Park, Y. W. and E. S. Lee, "A new generation method of an user profile for information filtering on the Internet," *Proceedings of IEEE Conference on Information Networking*, pp.261-264, 1998.
[16] Pinkerton, B., "Finding what people want: experience with the WebCrawler," *Proceedings of first International WWW Conference*, 1994.
[17] Rhodes, B. J. and T. Starner, "Remembrance agent: a continuously running automated information retrieval system," *Proceedings of PAAM*, 1996.
[18] Search Engine Watch, "Search engine features for webmasters," *http://searchenginewatch.internet.com/webmasters/features.html*, Revised 1999.
[19] Selberg, E. and O. Etzioni, "Multi-service search and comparison using the MetaCrawler," *Proceedings of third International WWW Conference*, 1995.
[20] Seltzer, R., E. Ray and D. Ray, *The AltaVista search revolution: how to find anything on the Internet*, McGraw-Hill, 1997.
[21] Sheldon, M. A. et al., "Discover: a resource discovery system based on content routing," *Proceedings of third International WWW Conference*, 1995.
[22] Slot, M., "Web matrix: overview matrix," *http://www.ambrosiasw.com/~fprefect/matrix/overview.html*, 1996.
[23] Weiss, R. et al., "HyPursuit: a hierarchical network search engine that exploits content-link hypertext clustering," *Proceedings of ACM Conference on Hypertext*, pp. 16-20, 1996.
[24] Wu, Y. H., and A. L. P. Chen, **"**Index structures of user profiles for efficient web page filtering services," *NTHU Technical Report*, 1999.
[25] Yahoo, *http://www.yahoo.com/*, 1999.
[26] Yan, T. W. and H. Garcia-Molina, "Index structures for information filtering under the vector space model," *Proceedings of International Conference on Data Engineering*, pp.337-347, 1994.
[27] Yan, T. W. and H. Garcia-Molina, "Distributed selective dissemination of information," *Proceedings of Parallel and Distributed Information Systems*, pp.89-98, 1994.
[28] Yan, T. W. and H. Garcia-Molina, "Index structures for selective dissemination of information under the boolean model," *ACM Transactions on Database Systems*, 19(2): 332-364, 1994.
[29] Yan, T. W., "SIFT--A Tool for Wide-Area Information Dissemination," *Proceedings of USENIX*, 1995.

Approaches / Criteria	Counting Method	Tree Method	Method 1	Method 2	Method 3	Method 4
Duplication of information	profiles	keywords	No	no	no	no
Sorting of keywords	no	yes	No	yes	no	yes
Storage space	O(|P|+f|K|)	O(n|P|)	O(|P|+|K|)	O(|P|+|K|)	O(|P|+|K|)	O(|P|+|K|)
Insertion/Deletion time	O(nf)	O(nf)	O(n+f)	O(nf)	O(n+f)	O(n+f)
Matching time	O(mf+|P|)	O(mf)	O(mf)	O(mf)	O(mf)	O(mf)
Modification time	O(nf)	O(nf)	O(n+f)	O(nf)	O(n+f)	O(n+f)

Table 3. Summary of the six approaches to profile indexing