Linear Time Algorithms for Finding Maximal Forward References

Zhixiang Chen
Department of Computer Science
University of Texas-Pan American
chen@cs.panam.edu

Richard H. Fowler
Department of Computer Science
University of Texas-Pan American
fowler@panam.edu

Ada Wai-Chee Fu
Department of Computer Science
Chinese University of Hong Kong
adafu@cse.cuhk.edu.hk

Abstract

In this paper, two algorithms are designed for finding maximal forward references from very large Web logs, longest sequences of Web pages visited by a user without revisiting some previously visited page in the sequence, and their performance is comparatively analyzed. It is shown that the two algorithms have linear (hence optimal) time complexity.

1 Introduction

Because of its significant theoretical challenges and great application and commercial potentials, Web mining has recently attracted extensive attention. Surveys of recent research in Web mining can be found in [7]. One of major concerns in Web mining is to discover user traversal (or navigation) path patterns that are hidden in vast Web logs. Such discovered knowledge can be used to predict where the Web users are going, i.e., what they are seeking for, so that it helps the construction and maintenance of real-time intelligent web servers that are able to dynamically tailor their designs to satisfy users' needs [11]. It has significant potential to reduce, through prefetching and cashing, Web latencies that have been perceived by users year after year [12, 8, 10, 13]. It can also help the administrative personnel to predict the trends of the users' needs so that they can adjust their products to attract more users (and customers) now and in the future [1].

Traversal path pattern mining is based upon the availability of traversal paths that must be obtained from raw Web logs. A maximal forward reference of a Web user, a longest sequence of Web pages visited by the user without revisiting some previously visited page in the sequence, is a typical traversal path pattern [5, 6]. In this paper, we study the problem of efficient identification of maximal forward references. Our work here aims for designing algorithms with optimal time complexity for identifying maximal forward references from very large Web logs. The best known algorithm reported in literature is a sorting-based algorithm with sublinear time complexity [5]. In this paper we present two algorithms, one is designed for *interval sessions* of user accesses and the other for *gap sessions*. We show that the two algorithms have linear, hence optimal, time complexity. We conduct empirical performance analysis and show that both algorithms need just several seconds more than the baseline time, i.e., the time needed for reading the Web log once sequentially from disk to RAM, testing whether each user access record is valid or not, and writing each valid user access record back to disk. The empirical performance analysis also shows that both algorithms are substantially faster than the sorting based algorithm. Finally, both algorithms are extended to the case of distributed Web logs.

The rest of the paper is organized as follows. The problem formulation is given in section 2. The related work is reviewed in section 3. Algorithms with linear time complexity are designed in section 4. Section 5 presents empirical performance analysis. Section 6 discusses extensions of the algorithms to distributed Web logs. Conclusions are given in section 7.

2. Problem Formulation

The Web has a natural graph structure: pages are in general linked via hyperlinks. When a user surfs the Web, she may move forward along the graph via selecting a hyperlink in the current page. She may also move backward to any page visited earlier in the same session (the definition of a session will be given later in section 4) via selecting a "backward" icon. A forward reference may be understood as that the user is looking for her desired information. A backward reference may mean that the user has found her desired information and is going to looking for something else. A sequence of consecutive forward references may indicate what the user is looking for. A maximal forward reference is defined as the longest consecutive sequence of forward references before the first backward reference is made to visit some previously visited page. As thus, the last reference in a maximal forward sequence indicates a content page [5, 6] that is desired by the user. Under such understanding , when a user searches for desired information, her information needs can be modeled by a set of maximal forward references occurred during her search process.

Suppose, like in [5], the Web log contains the traversal path $\{$A,B,C,D,C,B,E,G,H,G,W, A,O,U,O,V$\}$ for a user as shown in Fig. 1. Starting at A, we have consecutive forward reference sequences AB, ABC and ABCD. Then, at D a backward reference to C is made, meaning that ABCD is a maximal forward reference. Now, some caution shall be paid to C. Here, even though a backward reference is made to B, we shall not consider ABC as a maximal forward reference, because the current backward reference is the second backward reference (the first is D$\rightarrow$C. At B, a forward reference is made to E, thus begins new consecutive forward reference sequences ABE, ABEG and ABEGH. ABEGH is a maximal forward reference, because the first backward reference occurs at H after the forward reference E. Similarly, one can find the other maximal forward references ABEGW, AOU and AOV. When D, H, W, U and V are the pages desired by the user, the set of the maximal forward references $\{$ABCD, ABEGH, ABEGW, AOU, AOV$\}$ precisely describes the user's needs and her actual search behaviors as well.

The traversal path in the above example is recorded in the Web log, and so are traversal paths of all the other users. The goal of this paper is to design algorithms with optimal time complexity for finding maximal forward reference sequences from Web logs.

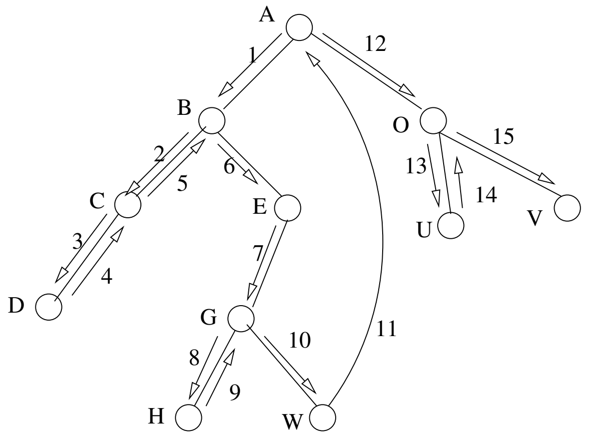

Figure 1. An illustration of traversal path patterns

3. Related Work

The problem of mining most frequent maximal forward references was investigated in [5]. As the first step, an algorithm for finding maximal forward references from a traversal log was designed in that paper. That algorithm works in two phrases. First, the log is sorted according to user IDs to group every user's references as a traversal path. Next, each traversal path is examined to find maximal forward reference. The time complexity of the algorithm is $O(n \log n)$, where n denotes the number of records in the log. [6] has detailed discussions of various data preparation tasks. The maximal forward references was also examined as a finer level characterization of user access sessions, but no explicit algorithm was given to find those sequences. To our best knowledge, the algorithm in [5] is the best reported in literature for finding maximal forward references.

Many researchers have been investigating traversal or surfing path patterns in order to model users' behaviors and interests. The uncovered model can be used in many applications such as prefetching and caching to reduce Web latencies. Examples of such work include [12, 8, 10, 13, 14]. The N-gram approaches was used to find most frequent subsequences of the paths, and sometimes the most frequent longest subsequences are the most interested. Once those subsequences were obtained, methods such as first order or higher order Markov-chains can be applied to predict users' behaviors or interests. Other researchers have also considered user access sessions as ordered sequences, such as in [9]. They also studied how to identify the longest repeating subsequences of the ordered sessions so as to find associations of those sequences. In all those papers, a user's surfing path is considered as an ordered sequences of consecutive references. In other words, a user's surfing path is the user's access session with order. This is different from the maximal forward references studied in [5, 6] and in the present paper.

In [4], optimal algorithms were designed for finding user access sessions from very large Web logs. As carefully examined in [6], maximal forward references are finer level characterizations of sessions, the algorithms in [4] can not be applied directly to the problem of finding maximal forward references. However, based on those algorithms, we will design linear time algorithms for finding maximal forward references, and the new algorithms only need to read the Web log once sequentially.

4. Algorithms

Our basic idea of finding maximal forward references is as follows. The algorithm reads the Web log once sequentially, and generates sessions on the fly. While a session is generating, maximal forward references within that session are also generated. As in [4], we consider two types of sessions in the following.

4.1. Interval Sessions and Gap Sessions

α**-interval sessions**: the duration of a session may not exceed a threshold of α. That is, we assume that a user should not spend too much time for each session. So we give a value α to limit the time a user can have for a session. Usually, we may set α to 30 minutes.

β**-gap sessions**: the time between any two consecutively assessed pages may not exceed a threshold of β. That is, we assume that a user should not be "idle" too much time between any two consecutive accesses in any session. So we give a value β to limit the time a user can be "idle" between two consecutive accesses in a session. Usually, we may set β to 30 minutes.

Certainly, there are exceptions to the α and β limits. For example, a person may need to spend a few hours to find a right tour map of Hong Kong over the web, so the session exceeds the α limit. However, recent studies show that these two kinds of limits are quite accurate in Web usage analysis.

4.2. Data Structures

Our algorithms maintain the following data structures. We define a URL node structure to store the URL and the access time of a user access record and a pointer to point to the next URL node. We define a user node to store the following information for a user: the user ID that is usually the hostname or IP address in the access record; the start time that is the time of the first access record entered into the user node; the current time that is the time of last access record entered into the user node; the URL node pointer that points to the linked list of URL nodes; a forward reference tag to indicated whether the last reference of the user is forward reference or not, the counter to record the total number of URL nodes added; and two user node pointers to respectively point to the previous user node and the next. Finally, we define a head node structure with two members, one pointing to the beginning of the two-way linked list of user nodes and the other storing the total number of valid records that have been processed so far.

4.3. Finding Maximal Forward Reference Sequences from Interval Sessions

At a higher level, the algorithm repeats the following tasks until the end of the web log: read a user access record r from disk to RAM; test whether r is valid or not; if r is valid then add r to its corresponding user node D in the user node linked list; compare the start time of D with the time $t(r)$ to see whether a new session of the user begins or not. If yes then output the URLs in D's URL node linked list in order as a maximal forward reference sequence, and and reset D with information in r. Otherwise use r to search D's URL linked list to find maximal forward reference sequences; finally, check if sufficient valid records have been processed so far. If yes, then purge all the old user nodes and output their sessions. The algorithm, called ISMaxFRS, is given in Figure 2.

4.4. Finding Maximal Forward Reference Sequences from Gap Sessions

The algorithm, called GSMaxFRS, is similar to algorithm ISMaxFRS. They differ at steps 9 and 16. At step 9 GSMaxFRS checks, in order to decide whether a new session begins or not, whether the "time gap" between the current record and the last record of the same user is beyond the threshold β or not. At step 16, algorithm GSMaxFRS will call function *purge2Nodes(outfile, S, t(r))* to check, for every user node in the user node linked list $S.head$, whether the "time gap" between the current record and the last record of the user node is beyond the threshold β or not. If so, it purges the user node in a similar manner as algorithm ISMaxFRS does. The description of GSMaxFRS is given in Figure 3.

4.5. Complexity

Algorithm ISMaxFRS has a main loop controlled by the number of user access records in the web log $\mathcal{L}$. When the two parameters α and γ are given, each loop iteration can be done in constant time. Hence, the time complexity of algorithm ISMax has an upper bound $O(n)$, where n denoted the number of user access records in the Web log. Obviously, in order to find maximal forward references from $\mathcal{L}$, any algorithm must process every user access record in $\mathcal{L}$. Thus, the time complexity of algorithm ISMaxFRS has a lower bound $\Omega(n)$. Therefore, its time complexity is optimally $\theta(n)$. Because algorithm GSMaxFRS has a similar structure as algorithm ISMaxFRS, it follows from the same analysis as above that GSMaxFRS also has linear time optimal time complexity.

Algorithm ISMaxFRS (Interval Session Maximal Forward Reference Sequence):

input:
infile: input web log file
outfile: output user access session file
$\alpha > 0$*: threshold for defining interval sessions*
$\gamma > 0$*: threshold for removing old user nodes*
begin
1. *open infile and outfile*
2. *createHeadNode(S); S.head = null; S.counter=0*
3. *while (infile is not empty)*
4. *readRecord(infile, r)*
5. *if (isRecordValid(r))*
6. *n = findRecord(S, r)*
7. *if (n is null)*
8. *addRecord(S, r)*
9. *else if* $(t(r) - n.startTime \leq \alpha)$
10. *n.currentTime = t(r)*
11. *findMaxFRS(n.urlListPtr, n.forwardReferenceTag, r)*
12. *else*
13. *writeMaxFRSAndReset(outfile, n, r)*
14. $S.counter = S.counter + 1$
15. *if* $(S.counter > \gamma)$
16. *purgeNodes(outfile, S, t(r)); S.counter=1*
17. *cleanList(outfile,S);*
18. *close infile and outfile*

end

Figure 2: Algorithm ISMaxFRS

Algorithm GSMaxFRS (Gap Session Maximal Forward Reference Sequences):

input: the same as ISMaxFRS except
$\beta > 0$ *is used to replace* α.
The body is also the same as ISMaxFRS except that Line 9 is replaced by
9. *else if* $(t(r) - n.currentTime \leq \beta)$

and Line 16 is replaced by
16. *purge2Nodes(outfile, S, t(r)); S.counter=1*

Figure 3: Algorithm GSMaxFRS

5. Empirical Performance Analysis

As mentioned before, to our best knowledge, the only reported algorithm for finding maximal forward references from Web logs is the sorting-based algorithm in [5], denoted as SortMF. We shall compare our algorithms with that algorithm. We use five web logs *L100MB, L200MB, L300MB, L400MB* and*L500MB* with respectively 100, 200, 300, 400 and 500 mega bytes of user access records that were collected from the Web server of the Department of Computer Science, the University of Texas - Pan American. The computing environment is a Gateway Dell E-5400 PC with 512 mega bytes of RAM and 20 Giga bytes of hard disk. We utilize the sort command in the DOS environment, which supports a standard external file sorting. We choose $\beta = \alpha = 30$ minutes and $\gamma = 500$. We use the same $isRecordValid()$ function for the three algorithms and the baseline time testing as well. Performance comparisons are illustrated in Figures 4 and 5. It is clear that ISMaxFRS and GSMaxFRs have almost the same performance and are substantially much faster than the sorting based algorithm in [5].

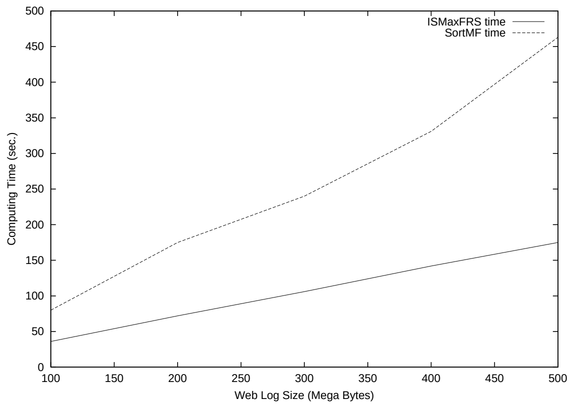

Figure 4. Execution Times of ISMaxFRS vs. SortMF

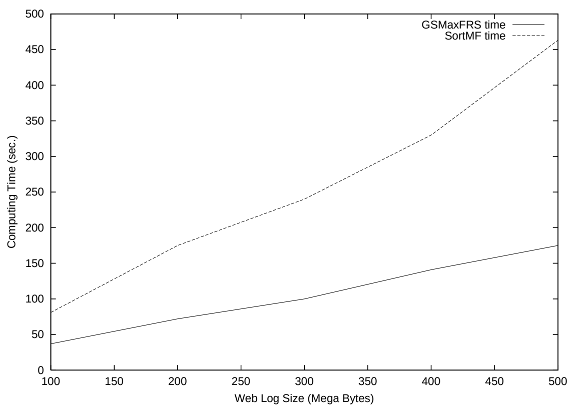

Figure 5. Execution Times of GSMaxFRS vs. SortMF

6. Extensions

A heavily accessed web server may have several distributed logs of user access records for many good reasons. One is to reduce the workload of the server. Another is that the architecture of the server itself may be distributed: the main server has several distributed proxy servers so that when a user access the main server, the user's request may be automatically dispatched to one of the proxy servers. Thus, in this case, user's access records are logged in several distributed files. Let $\mathcal{L}_1, \ldots, \mathcal{L}_d$ be d distributed web logs. We can design two linear algorithms to identify user access sessions from the union of those distributed logs $\mathcal{L}_1 \cup \cdots \cup \mathcal{L}_d$. The difficulty we are facing now is that records in a given user access session may be scattered among those d logs. But we can overcome the difficulty by means of the fact that records in each log are sorted in time. Precisely, we can read one record from each of those distributed logs, and find the oldest and feed it to algorithm ISMaxFRS or GSMaxFRS.

7. Conclusions

Finding maximal forward references is a fundamental data preparation task in Web Mining. We design two algorithms for finding maximal forward references from respectively interval sessions and gap sessions. We show that both algorithms have linear time optimal time complexity. To our best knowledge, no such optimal algorithms have been reported in literature. When the two algorithms are combined with our work on fast construction of suffix trees over a very large alphabet [2], we obtain algorithms for mining frequent maximal forward references from very large Web logs with best possible efficiency [3].

ACKNOWLEDGMENT. The work in this paper is supported in part by the UTPA Center of Computing and Information Technology.

References

[1] A.G. Buchner and M.D. Mulvenna. Discovering internet marketing intelligence through online analytical web usage mining. *ACM SIGMOD RECORD*, pages 54-61, Dec. 1998.

[2] Z. Chen, R. Fowler, A. Fu, and C. Wang, Fast construction of generalized suffix trees over a very large alphabet, submitted for publication, 2003.

[3] Z. Chen, R. Fowler, A. Fu, and C. Wang, Linear and sublinear time algorithms for mining frequent traversal path patterns from very large Web logs, submitted for publication, 2003.

[4] Z. Chen, A. Fu, and F. Tong, Optimal algorithms for finding user access sessions from very large Web logs, Advances in Knowledge Discovery and Data Mining/PAKDD'02, Lecture Notes in Computer Science 2336, pages 290-296, 2002. (The full version will appear in Journal of World Wide Web: Internet and Information Systems, 2003.)

[5] M.S. Chen, J.S. Park, and P.S. Yu. Efficient data mining for path traversal patterns. *IEEE Transactions on Knowledge and Data Engineering*, 10:2:209-221, 1998.

[6] R. Cooley, B. Mobasher, and J. Srivastava. Data preparation for mining world wide web browsing patterns. *Journal of Knowledge and Information Systems*, 1:1, 1999.

[7] R. Kosala and H. Blockeel, Web mining research: A survey, SIGKDD Explorations, 2(1), pages 1 - 15, 2000.

[8] T.M. Kroeger, D.D.E. Long, and J.C. Mogul, Exploring the bounds of Web latency reduction from cashing and prefetching, Proc. of the USENIX Symposium on Internet Technologies & Systems, 1997.

[9] F. Masseglia, P. Poncelet, and R. Cicchetti, An efficient algorithm for Web usage mining, Networking and Information Systems Journal, 2(5-6), pages 571-603, 1999.

[10] V.N. Padmanabhan and J.C. Mogul, Using predictive prefetching to improve World Wide Web latency, Computer Communication Review, 26, 22-36, 1996.

[11] M. Perkowitz and O. Etzioni. Adaptive web pages: Automatically synthesizing web pages. *Proceedings of AAAI/IAAI'98*, pages 727-732, 1998.

[12] J. Pitkow and P. Pirolli, Mining longest repeating subsequences to predict World Wide Web Surfing, Proc. of the Second USENIX Symposium on Internet Technologies & Systems, pages 11-14, 1999.

[13] S. Schetcher, M. Krishnan, and M.D. Smith, Using path profiles to predict HTTP requests, Proc. of the Seventh International World Wide Web Conference, Computer Networks, 30(1-7), 457-467, 1998.

[14] Z. Su, Q. Yang, Y. Lu, and H. Zhang, WhatNext: A prediction system for Web requests using N-gram sequence models, Proc. of the First International Conference on Web Information Systems Engineering, pages 200-207, 2000.