Optimal Algorithms for Finding User Access Sessions from Very Large Web Logs

Zhixiang Chen[1], Ada Wai-Chee Fu[2], and Frank Chi-Hung Tong[3]

[1] Department of Computer Science, University of Texas-Pan American, Edinburg TX 78539 USA. chen@cs.panam.edu

[2] Department of Computer Science, Chinese University of Hong Kong, Shatin, N.T., Hong Kong. adafu@cse.cuhk.edu.hk

[3] Department of Computer Science and Information Systems, the University of Hong Kong, Pokfulam Road, Hong Kong. ftong@eti.hku.hk

Abstract. Although efficient identification of user access sessions from very large web logs is an unavoidable data preparation task for the success of higher level web log mining, little attention has been paid to algorithmic study of this problem. In this paper we consider two types of user access sessions, *interval sessions* and *gap sessions*. We design two efficient algorithms for finding respectively those two types of sessions with the help of new data structures. We present both theoretical and empirical analysis of the algorithms and prove that both algorithms have optimal time complexity.

1 Introduction

Web log mining has been receiving extensive attention recently (e.g., [1, 3–5, 13]) because of its significant theoretical challenges and great application and commercial potentials. The goal of web log mining is to discover user access behaviors and interests that are buried in vast web logs that are being accumulated every day, every minute and every second. The discovered knowledge of user access behaviors and interests will certainly help the construction and maintenance of real-time intelligent web servers that are able to dynamically tailor their designs to satisfy users' needs [7, 10, 11]. The discovered knowledge will 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 [3].

The *definition of* **session** *or* **visit** *is the group of activities performed by a user from the moment she enters a server site to the moment she leaves the site* [12]. A user access session determines the set of all the web pages a user accessed from the beginning of the session to the end. This kind of understanding is similar to the *"market basket"* concept in data mining and knowledge discovery [5], the set of all items in the market basket a customer buys at a retail store. As such, the existing data mining techniques, such as the popular association rule mining, may be applied to web log mining [13]. Some researchers [6] have suggested that user access sessions are too coarse grained for mining tasks such as discovery of association rules so that it is necessary to refine single user access sessions into smaller transactions. But such refining process is also based on identification of user access sessions. The difficulty of finding users and user access sessions from web logs have been addressed in [6, 4, 8, 9]. Researchers have proposed cut-off thresholds to approximate the start and the end of a session (e.g., [6]). One may also propose a 30-minutes gap limit between any two pages accessed by the user. If the gap limit is exceeded, then a session boundary should be inserted between the two pages [2, 10, 11].

There is little algorithmic study of efficient identification of user access sessions in literature. Web logs are typically large, from hundreds of mega bytes to tens of giga bytes of user access records. The logs are usually stored in hard disks. Certainly, after eliminating irrelevant records from the raw web log, sorting algorithms can be applied to group the same user's access records together. After that a one-pass sequential reading process can be used to cut each group of the same user's access records into sessions according to some threshold. This sorting approach may be efficient for smaller web logs, but it is not practically efficient for finding sessions from very large web logs, say, logs with hundreds of mega bytes or tens of giga bytes of records. When real-time applications based on dynamic discovery of the users' hidden access patterns is concerned, faster algorithms that have the ability to find user access sessions in several seconds or several minutes is highly preferred. Without such faster algorithms, any higher level mining tasks would be slowed down.

This paper is organized as follows. In section 2, we give formal definitions of α-interval sessions and β-gap sessions. In section 3, we design data structures and two session finding algorithms. We given theoretical and empirical analysis about the two algorithms in sections 4 and 5, and conclude the paper in section 6.

2 Web Logs and User Access Sessions

The web log for any given web server is a sequential file with one user access record per line. Each user access record consists of the following fields: (1) User's IP address or host name, (2) access time, (3) request method (e.g., "GET", "POST"), (4) URL of the page accessed, (5) Protocol (e.g., HTTP/1.0), (6) return code, and (7) number of bytes transmitted. Other fields may also be included when the server is specially configured. One should note that user access records in a web log are ordered according to record access time with the oldest access record placed on the top and the newest at the bottom.

Given any user access record r, let $t(r)$ be the number of seconds starting at `01/Jan/1900:00:00:00` and ending at the time when r was performed. Let $\mathcal{L}$ be a web log and α and β two positive values. We propose the following two ways to define user access sessions. Recent studies show that the two ways lead to high accuracy in web usage analysis [2].

α-interval sessions: the duration of a session may not exceed a threshold of α. Given any user u, let $r_1, \ldots, r_m$ be the ordered list of u's access records in $\mathcal{L}$. The α-interval sessions for u is a list of subsets $s_1 = \{r_{i_0+1}, \ldots, r_{i_1}\}, s_2 = \{r_{i_1+1}, \ldots, r_{i_2}\}, \ldots, s_k = \{r_{i_{k-1}+1}, \ldots, r_{i_k}\}$ such that for $1 \leq j \leq k$ we have

$$0 \leq t(r_{i_j}) - t(r_{i_{j-1}+1}) \leq \alpha,\ t(r_{i_j+1}) - t(r_{i_{j-1}+1}) > \alpha,$$

where $r_{i_0+1} = r_1$ and $r_{i_k} = r_m$.

β**-gap sessions**: the time between any two consecutively assessed pages may not exceed a threshold of β. Given any user u, let $r_1, \ldots, r_m$ be the ordered list of u's access records in $\mathcal{L}$. The β-gap sessions for u is a list of subsets $s_1 = \{r_{i_0+1}, \ldots, r_{i_1}\}, s_2 = \{r_{i_1+1}, \ldots, r_{i_2}\}, \ldots, s_k = \{r_{i_{k-1}+1}, \ldots, r_{i_k}\}$ such that for $1 \leq j \leq k$ and for $i_{j_1} + 1 \leq l < i_j$ we have

$$0 \leq t(r_{l+1}) - t(r_l) \leq \beta,\ t(r_{i_j+1}) - t(r_{i_j}) > \beta,$$

where $r_{i_0+1} = r_1$ and $r_{i_k} = r_m$.

3 Data Structures and Algorithms

Our algorithms maintain a data structure as follows. 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; 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 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 relevant records that have been processed so far.

The interval session finding algorithm, called ISessionizer, is given in Figure 1. Definitions of the functions used in the algorithm should be inferred from their names. E.g., `purgeNodes(outfile, S, t(r))` searches every user node n in the linked list *S.head*. If it finds that $t(r) - n.startTime > \alpha$, then it outputs the session consisting of the URLs in *n.listPtr* to *outfile*, and then deletes the URL linked list *n.listPtr* and the user node n via memory deallocation.

The gap session finding algorithm, called GSessionizer, is shown in Figure 2. Function *purge2Nodes(outfile, S, t(r))* checks, for every user node in *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 ISessionizer does.

4 Theoretical Analysis

We present two theorems for algorithm ISessionizer only and the proofs are omitted due to space limit but will be included in the full version of the paper.

Similar theorems can be obtained for algorithm GSessionizer. Several notations are introduced: T_1 = the time to read one record from the web log $\mathcal{L}$. T_2 = the time to check whether a record is relevant or not. T_3 = the time to write the URL and the access time of a relevant record from RAM to an output file. and T_4 = the time to search once the user linked list *S.head* and the URL linked list at each user node.

```
Algorithm ISessionizer (Interval Sessionizer):
    input:
        infile: input web log file
        outfile: output user access session file
        α > 0: threshold for defining interval sessions
        γ > 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 (isRecordRelevant(r))
6.                      n = findRecord(S, r)
7.                      if (n is null)
8.                              addRecord(S, r)
9.                      else if (t(r) − n.startTime ≤ α)
10.                             n.currentTime = t(r)
11.                             addURL(n.urlListPtr, r)
12.                     else
13.                             writeSessionAndReset(outfile, n, r)
14.             S.counter = S.counter + 1
15.             if (S.counter > γ)
16.                     purgeNodes(outfile, S, t(r)); S.counter=1
17.     cleanList(outfile,S);
18.     close infile and outfile
    end
```

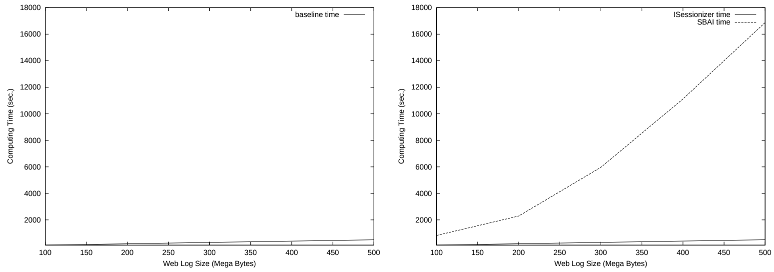
Figure 1: Algorithm ISessionizer

```
Algorithm GSessionizer (Gap Sessionizer):
    input:
        infile: input web log file
        outfile: output user access session file
        β > 0: threshold for defining gap sessions
        γ > 0: threshold for removing old user nodes
    This algoprithm is the same as algorithm ISessionier except that
    Lines 9 and 16 are replaced respectively by
9.                      else if (t(r) − n.currentTime ≤ β)
16.                     purge2Nodes(outfile, S, t(r)); S.counter=1
```

Figure 2: Algorithm GSessionizer

Theorem 1. *Given $\alpha > 0$, let $\mathcal{S}$ denote the set of all the α-interval sessions in $\mathcal{L}$. Then, algorithm ISessionizer finds exactly the set $\mathcal{S}$.*

Theorem 2. *Assume that the web log $\mathcal{L}$ have N user access records with M relevant ones. Then, the time complexity of algorithm ISessionizer with a γ-threshold for purging user nodes is at most $(T_1 + T_2)N + T_3M + T_4(1 + \frac{1}{\gamma}))M$.*



(a)Baseline Performance (b) ISessionizer vs. SBAI

Fig. 1. Performances of Algorithm ISessionizer vs. SBAI vs. Baseline Time

5 Empirical Analysis

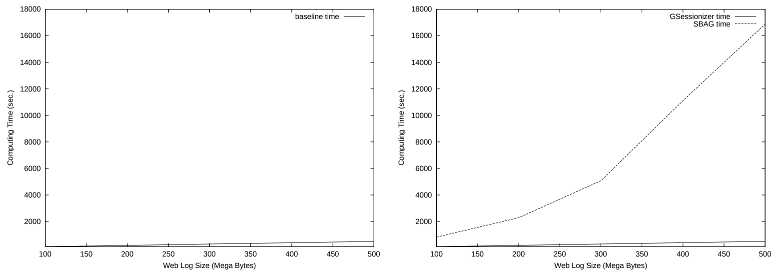
We report empirical performance analysis of algorithms ISessionizer and GSessionizer. We use five web logs with respectively 100, 200, 300, 400, and 500 mega bytes of user access records that were collected from the web server of the Department of Computer Science, the University of Texas - Pan American. The computing environment is a Dell OptiPlex GX1 PC with a 300 MHz processor, 512 mega bytes of RAM and 8 Giga bytes of hard disk. The baseline time is the time needed for reading a Web log once sequentially from disk to RAM, testing whether each record is pertinent or not, and writing each pertinent record back to disk. The sorting based algorithm, denoted by *SBAI*, for finding α-interval sessions is the one that sorts the web log to group the same user's access records together and then reads the sorted log sequentially once from disk to RAM to divide the relevant access records in each group into α-interval sessions. The sorting based algorithm, denoted by *SBAG*, for finding β-gap sessions works similarly as algorithm SBAI, but it divides the relevant access records in each group into β-sessions. We choose $\beta = \alpha = 30$ minutes and $\gamma = 500$. The same *isRecordRelevant( )* function was used for the four algorithms and the baseline time testing. Performance comparisons are illustrated in Figures 3 and 4.

6 Conclusions

We have settled the problem of efficient identification of interval sessions and gap sessions from very large Web logs. However, we feel that more robust session definitions may be needed because of the vast diversity of the web users and wide applications of cashing and proxy servers, and that machine learning may help the design of new algorithms.

References

1. B. Berendt and M. Spiliopoulou. Analysis of navigation behavior in web sites integrating multiple information systems. *The VLDB Journal*, 9:56-75, 2000.



(a)Baseline Performance (b) GSessionizer vs. SBAG

Fig. 2. Performances of Algorithm GSessionizer vs. SBAG vs. Baseline Time

2. Bettina Berendt, Bamshad Mobasher, Myra Spiliopoulou, and Jim Wiltshire. Measuring the accuracy of sessionizers for web usage analysis. *Proceedings of the Workshop on Web Mining at the First SIAM International Conference on Data Mining*, pages 7-14, April 2001.
3. Alex G. Buchner and Maurice D. Mulvenna. Discovering internet marketing intelligence through online analytical web usage mining. *ACM SIGMOD RECORD*, pages 54-61, Dec. 1998.
4. L. Catledge and J. Pitkow. Characterizing browsing behaviors on the world wide web. *Computer Networks and ISDN Systems*, 27, 1995.
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. Robert Cooley, Bamshad Mobasher, and Jaidep Srivastava. Data preparation for mining world wide web browsing patterns. *Journal of Knowledge and Information Systems*, 1:1, 1999.
7. M. Perkowitz and O. Etzioni. Adaptive web pages: Automatically synthesizing web pages. *Proceedings of AAAI/IAAI'98*, pages 727-732, 1998.
8. J. Pitkow. In search of reliable usage data on the WWW. *Proceedings of the Sixth World Wide Web Conference.* pages 451-463, Santa Clara, CA, 1997.
9. P. Pirolli, J. Pitkow, and R. Rao. Silk from sow's ear: Extracting usable structures from the Web. *Proceedings of the 1996 Conference on Human Factors in Computing Systems (CHI'96).* Vancouver, British Columbia, Canada, 1996.
10. Myra Spiliopoulou and Lukas C. Faulstich. Wum: A tool for web utilization analysis. *Proceedings of EDBT Workshop WebDB'98*, LNCS1590, pages 184-203. Springer Verlag, 1999.
11. Myra Spiliopoulou, Carsten Pohle, and Lukas C. Faulstich. Improving the effectiveness of a web site with web usage mining. *KDD'99 Workshop on Web Usage Analysis and User Profiling WEBKDD'99*, Aug, 1999.
12. W3C. World wide web committee web usage characterization activity. *W3C Working Draft: Web Characterization Terminology and Definitions Sheet*, pages www.w3.org/1999/05/WCA-terms/, 1999.
13. Osmar Zaïane, Man Xin, and Jiawei Han. Discovering web access patterns and trends by applying olap and data mining technology on web logs. *Advances in Digital Libraries*, pages 19-29, April, 1998.