The Reconstruction of User Sessions from a Server Log Using Improved Time-oriented Heuristics

Jie Zhang and Ali. A. Ghorbani
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B 5A3, Canada

E-mail: {jie.zhang,ghorbani}@unb.ca

Abstract

Web usage mining plays an important role in the personalization of Web services, adaptation of Web sites, and the improvement of Web server performance. It applies data mining techniques to discover Web access patterns from Web usage data. In order to discover access patterns, Web usage data should be reconstructed into sessions with or without user identification. However, not all Web server logs contain complete information for constructing user sessions. One approach for solving such a problem is to use time-oriented heuristics to reconstruct user sessions.

This paper describes improved statistical-based time-oriented heuristics for the reconstruction of user sessions from a server log. Comparative analysis are carried out using two similarity measures. The performance results of the proposed improved heuristics are promising and in some cases show reasonable improvements.

keywords: Web Mining, Web Usage Mining, Session Reconstruction, Heuristics.

1. Introduction

Web usage mining applies data mining techniques to mine Web access patterns [7, 8]. Mining Web access patterns is useful when building user profiles which in turn are used for the personalization and tuning of Web services, the presentation of promotional contents, and other applications for which user interests, preferences, requirements, and behavioral conventions must be assessed and served [6]. Mining Web access patterns is also useful when improving Web structure and Web server performance [9]. Users' access to pages of the Website should be separated into user sessions. Each session is the group of activities performed by a user from the moment she enters the site to the moment she leaves it.

User sessions are extracted from the Web server log, the primary source of data in which the activities of Web users are captured. More reliable Web usage mining results need more reliable reconstructed user session results. However, it is difficult to tell when a user has left a Web site because there is no record of users leaving. Aside from the lack of information, some other problems also exist. For example, an IP sharing problem exists because several users may access a site through the same host or proxy and may employ the same software agent. An empty referrer may appear inside a session due to the following reasons: a) the user has typed the URL directly; b) requests are made by agents, and agents do not necessarily follow the page links; and c) some frames belong to the same page [10].

Several approaches are devised to address these problems. Two time-oriented heuristics for session identification are described in [1, 10]: the session duration heuristic (h1) and the page-stay time heuristic (h2). The heuristic h1 states that the duration of a session must not exceed a threshold. The heuristic h2 is based on the assumption that the duration of a visited page must not exceed a threshold. Due to users' irregular navigation behavior, the performance of the time-oriented heuristics (h1 and h2) with fixed thresholds in reconstructing the sessions have not been satisfactory. In this paper we propose three extensions to h1 and h2 heuristics in order to improve their performance.

The rest of the paper is organized as follows. Section 2 briefly introduces the proposed heuristics. Section 3 presents the framework of the system developed in this work. Section 4 gives the detailed implementation of the proposed system including its four main phases. Subsequently, experimental results are presented in Section 5. The conclusion of the present study and future work are given in Section 6.

2. Proposed Heuristics

Commonly used time thresholds for h1 and h2 are 30 and 10 minutes, respectively. A 30-minute cutoff time for session duration is proposed by [3] and used commonly in many applications. A 10-minute threshold for page-stay time is mentioned by [10] as a very conservative maximum cutoff. We believe that different Web site structures and different user groups should have different thresholds for h1 and h2. In this paper, a statistical-based approach is employed to determine appropriate thresholds for h1 and h2. The main aim of the proposed approach is to improve the performance of h1 and h2. In the implementation we have used the following notations:

- **Fh1**: h1 heuristic with 30 minutes fixed threshold.
- **Fh2**: h2 heuristic with 10 minutes fixed threshold.
- **Dh1**: h1 heuristic with variable threshold.
- **Dh2**: h2 heuristic with variable threshold.

The heuristics Fh2 and Dh2 put a limit on the time spent on a page. We in turn propose the heuristic Mh2 which is based on the time difference between two visited pages. A page in the middle of two pages will be assigned to the session to which it is closer (i.e., has smaller time difference). This heuristic is based on the assumption that closer pages are more likely to belong to the same session.

Heuristics Fh1, Fh2, Dh1, Dh2 and Mh2 are all time-oriented heuristics. [4] points out that Web topology can help user session identification. In addition, clearly Web access patterns result from reasons such as underlying structure of Web sites, users' habits, users' interests in topics, and association of concepts. Many types of access patterns can be extracted with different meanings and usages. One typical type of access pattern is Maximal Frequent Sequence (MFS), which is defined by [12] as "frequently used contiguous sequences of page references". Based on the assumption that MFSs extracted from one place of Web usage data may likely exist in other places, we use MFSs to pre-separate an access sequence. Then we apply other heuristics on pre-separated access sequences. Moreover, applying MFSs into session reconstruction may somewhat solve lack of information problems.

3. The Reconstruction Framework

Figure 1 illustrates the framework of the system developed in this work. In this figure, ellipses represent entities or databases, rectangles represent engines or components in the system, and arrows represent data flow from/to entities to/from system components. Real sessions are captured by cookies or other information such as IPs from the Web server log. The Statistical Analyzer calculates two time thresholds for Dh1 and Dh2 using real sessions. The MFS Discoverer identifies MFSs from a generalized suffix tree built from real sessions (training examples). The discovered MFSs are used by the Session Reconstructor to separate the given long sequence of users' accesses (testing data) into smaller sequences. Finally, the Session Reconstructor combines smaller sequences into sessions by applying individual heuristic or a combination of different heuristics. A detailed explanation of each system component is given in the following subsections.

3.1. Dynamic Heuristics

The Statistical Analyzer calculates threshold values α_1 and α_2 for Dh1 and Dh2 as follows:

$$\alpha_1 = \mu_1 + \lambda\sigma_1 \qquad 0 \leq \lambda \leq 5 \tag{1}$$

$$\alpha_1 = \mu_2 + \lambda\sigma_2 \qquad 0 \leq \lambda \leq 5 \tag{2}$$

where μ_1 and μ_2 represent the average duration of all sessions and the average page-stay time, respectively. σ_1 and σ_2 denote standard deviations of session duration and page-stay time, respectively.

Let x_i and y_j represent duration of ith session and page-stay time of jth page. Let S and P represent the total number of sessions and pages in the data set, respectively. We calculated the averages (μ_1, μ_2) after removing the smallest and largest values from the set of session durations and the set of page-stay times. Experiments are carried out using values 0 to 5 for λ. The results are presented in Subsection 5.2.

3.2. Maximal Frequent Sequences

A frequent sequence is defined as the frequently used contiguous sequence of page references [12]. A frequent sequence is maximal if it is not a subsequence of any other frequent sequence. The technique of detecting MFS, online adaptive traversal (OAT) pattern mining, is presented in [12].

A large sequence can be represented by a suffix tree. In the suffix tree the nodes that have only one child are ignored. The subsequences represented in the suffix tree by each edge are shown as $x : y$, where x represents the position of the first character in a subsequence, and y is the length of that subsequence. Each internal node represents a sequence of characters that start from the root. The suffix link at the internal node points to the node that represents the longest suffix of the subsequence. The suffix links pointing to the root are ignored. Suffix links are used to help construct a suffix tree.

Ukkonen's method for constructing a suffix tree is a linear time algorithm [5, 11]. It uses suffix links to speed up the implementation. However, the training examples that are used to discover MFSs are multiple sequences of accessed

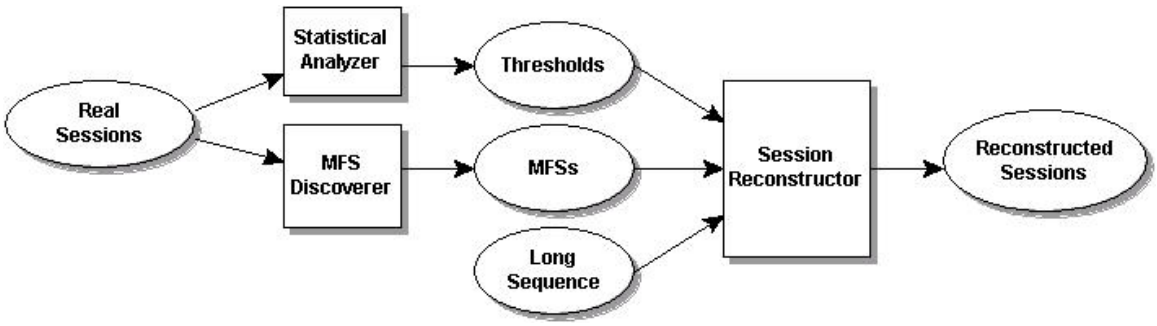

Figure 1. Simplified framework of the system

pages. Thus, the suffix tree for multiple sequences, called generalized suffix tree [2], should be constructed. To construct a generalized suffix tree, a unique symbol is appended to each sequence, and the database is regarded as a large sequence. A suffix tree for the first sequence of characters is built first. Then, starting at the root of this tree, the second sequence is matched against a path in the tree until a mismatch occurs. At that point, the remaining characters of the suffix for the second sequence are added to the current suffix tree. When the second sequence is fully processed, it encodes all the suffixes of the first sequence and all the suffixes of the second sequence. Following this process, the generalized suffix tree for the string set is built.

After constructing the generalized suffix tree, the MFSs are extracted by the OAT algorithm. The OAT algorithm was implemented in C++ and works properly with experimental data. The experimental results are clearly described in [12]. The implemented OAT algorithm outputs MFSs and their suffixes instead of only MFSs since the latter needs more memory. Fortunately, this fact makes implementation of our system much easier. To separate a given long sequence based on those MFSs, it is more efficient to match the long sequence with the suffix tree. Once we have MFSs and their suffixes, it is not necessary to apply Ukkonen's algorithm again.

3.3. Session Reconstructor

Two processes, pre-separation of access sequence and session reconstruction, are performed by the Session Reconstructor.

3.3.1. Pre-separation of Access Sequences MFSs produced by the OAT algorithm are sequences that frequently appear in the training examples. A given access sequence is pre-separated into a number of smaller sequences by MFSs. These smaller sequences are later used for session reconstruction.

The output of the OAT algorithm is the MFSs and their suffixes. A simple way to separate the long sequence is matching the sequence to the MFSs by scanning all the MFSs one by one. One difficulty found in this approach is that the separation of the access sequence cannot be decided until all the MFSs are scanned and compared. Furthermore, MFSs are frequent, and all their subsequences are frequent too. Thus, every subsequence should also be scanned and compared. The time complexity of this approach, of course, is extremely large. Using the suffix tree of MFSs is a more efficient way and is implemented in our system.

Since the long sequence is separated by scanning one character after another from left to right, only MFSs and their suffixes should be compared in order to get maximal length of shorter sequences. We first build the suffix tree for the MFSs. All the suffixes of the MFSs are already found by the OAT algorithm. It is unnecessary to apply Ukkonen's algorithm again to build a suffix tree for the MFSs.

The tree for all the MFSs and their suffixes is, of course, the suffix tree of the MFSs. Once the tree is built, the long sequence can be separated by walking down this tree from root to the deepest node. The deepest node here means that there is no further character that can be matched by the children of the node. This node can be an internal node, a leaf, or even the root. By repeating this process, we finally get a set of shorter sequences, which will be used for session reconstruction.

3.3.2. Session Reconstruction Heuristics Fh1, Fh2, Dh1, Dh2, Mh2 and their combinations are used to reconstruct the sessions. They are applied after separating long access sequences using the discovered MFSs. Note that there is no difference between Fh1 and Dh1 as well as Fh2 and Dh2, except Fh1 and Fh2 use fixed thresholds whereas Dh1 and Dh2 use variable thresholds.

The pseudo code for the Session Reconstruction (SR) algorithm is shown in Figure 2. The sequences resulting from the pre-separation stage are stored in a stack. From the stack, each time at most three sequences are removed and possibly merged. A simple case happens when there is only one sequence left in the stack, and this sequence will directly become a single session. Two complex types

of merging sequences can happen.

- If there are two sequences left in the stack that satisfy the constraints of the heuristics, they are merged and become a single session. Otherwise, each becomes a session.
- The most complex case is when there are three sequences to be merged (see Figure 2 for the details). Note that there are only two cases that a new session will be created: a) when the first two sequences cannot be merged or the second sequence is closer to the third sequence, the first sequence becomes a new session; b) when only the first two sequences can be merged or the second sequence is closer to the first sequence, the first two sequences are merged and become a new session.

```
Procedure reconstruct (Stack S)
//S stores sequences after pre-separation
while S is not empty
    pop up (at most) three sequences
    from the top of the stack
    if three sequences are popped up
        if the first two sequences statisfy constraints
            if all three sequences statisfy constraints
                merge all of them, push back to the stack
            else if the last two sequences statisfy constraints
                if the second one is closer to the first one
                    merge the first two sequences,
                    becomes one session
                else
                    the first sequence becomes one session
            else
                merge the first two sequences,
                becomes one session
        else the first sequence becomes one session
    else if two sequences are popped up
        if they satisfy constraints
            merge them and becomes one session
    else //only one sequence is popped up
        it becomes one session
end while
end Procedure
```

Figure 2. Session reconstruction algorithm

4. Implementation

Implementation of the system consists of four phases: data preparation, data preprocessing, real session generation, and evaluation.

4.1. Data Preparation

In the data preparation phase, data are collected and cleaned, and real sessions are generated.

4.1.1. Data Collection All the experiments are carried out using a large data set. The set consists of Web log data collected from January 6, 2003 to January 9, 2003 at the University of New Brunswick. Part of the data set is reserved for testing. The rest is used to calculate thresholds for Dh1 and Dh2 heuristics and find maximal frequent sequences. The MFSs are used for pre-separation of the test data. The training and test data are obtained after cleaning the Web log data and generating real sessions.

4.1.2. Data Cleaning The cleaning process removes graphic/multimedia entries as well as information such as graphic-maps. Most related works also remove the entries produced by executing CGI scripts and 'POST' commands. Because these entries contain valuable information for session reconstruction, we decided to keep them. However, we remove entries that are generated by CGI scripts but do not have direct HTML references. Entries with status code '4xx', '5xx', and '301' are removed as well as entries with the 'HEAD' method.

4.1.3. Real Session Generation The real sessions are generated by simply counting the source IPs. In this case, we assume that the IP sharing problem does not exist or is at its minimum effect. For a single user with multiple sessions, we use the referrer information to assign visited pages to the sessions properly.

4.2. Data Preprocessing

In this phase, we select useful information from the real session data including visited pages and time stamps for each page. Each page is assigned a unique ID. The time stamp of a page is converted into an integer number which represents the time difference in seconds between this page and the earliest visited page. The result of this process is a file containing real sessions with visited pages and corresponding time stamps. Subsequently, we split the real sessions into two parts: training data and test data (see Subsection 5.1 for details).

4.3. Session Reconstruction

The Session Reconstructor uses the log data to build sessions. It consists of a number of components. One component builds the suffix tree. It uses the discovered MFSs and their suffixes as input and returns the suffix tree. Every node in the suffix tree is a hashtable, which contains different visited pages.

Information	Data Set for Test 1				Data Set for Test 2			
	N	μ (s)	σ	M (s)	N	μ (s)	σ	M (s)
Page	153066	54	199.2526	3726	152373	54	199.2526	3726
Session	24127	269	558.7291	19608	24055	268	557.1731	19608

Table 1. Statistical Information for the Two Training Data Sets

Another component of the Session Reconstructor is a procedure that separates the log data by matching the suffix tree. Starting from the root of the suffix tree, nodes are visited within a path until there are no further visited pages that can be matched. The procedure repeats the process to match the following visited pages. The smaller sequences produced by this procedure will be stored in a stack.

The third component validates discovered MFSs and discards the ones which contain only one visited page or are not sequential in time.

4.4. Evaluation

The degree of similarity between the real sessions and the generated sessions is used to evaluate the performance of the SR algorithm. Two measures are used to calculate the degree of the similarity. In the first similarity measure, S, the degree of similarity between a real session $R = \{p_1, p_2, ..., p_n\}$ and a generated session $G = \{g_1, g_2, ..., g_n\}$, where p_i and g_i denote visited pages, is given as follows:

$$S = \frac{|R \bigcap G|}{|R \bigcup G|}$$

Another similarity measure, S', is calculated as follows:

$$S' = 1 - (d)^n (d')^{1-n}$$

where d represents the percentage of extra pages generated and d' represents the percentage of pages missed. n and $n-1$ represent the weights of d and d', respectively. d and d' are calculated as follows:

$$d = \frac{|\{R \bigcup G\} - R|}{|G|}$$

$$d' = \frac{|\{R \bigcup G\} - G|}{|R|}$$

5. Experimental Results

The experiments show the results of session reconstruction based on two sets of training and test data. the results of different stages are presented in the following subsections.

5.1. Data for Training and Testing

Two sets of training and test data are created from real sessions. The first set is created by selecting x number of sessions starting from the first session in every 100 real sessions. The second set is created by selecting x number of sessions starting from the twentieth one in every 100 real sessions. The value of x is randomly selected from a set of integer numbers less than 20. Information about the two sets of training data is given in Table 1. In this table, N represents the total number of pages or sessions, μ is average session duration or page-stay time in seconds, σ denotes standard deviation, and M represents maximal session duration or page-stay time.

5.2. Results of Statistical Analysis

The Statistical Analyzer produces different thresholds for the Dh1 and Dh2 heuristics. These thresholds are summarized in Table 2. In this table, λ represents the number of standard deviations.

λ	Data Set for Test 1		Data Set for Test 2	
	Dh1	Dh2	Dh1	Dh2
0	269	54	268	51
1	828	253	825	253
2	1386	453	1382	452
3	1945	652	1940	652
4	2504	851	2497	851
5	3063	1050	3054	1050

Table 2. Thresholds for Dh1 and Dh2 Using Equations 1 and 2 (All numbers are in seconds)

5.3. Selecting the Best Similarities

Tables 3 and 4 give the similarities between real sessions and generated sessions according to different thresholds summarized in Table 2. Note that in Tables 3-7, we use the symbol '&' to represent combinations of heuristics. For example, Mh2&Dh1&Dh2 represents the combination of Mh2, Dh1 and Dh2 heuristics. In Tables 3 and 4, Dh1

Heuristics	Similarity S					
	0	1	2	3	4	5
Dh1	0.7653	0.7858	0.7719	0.7636	0.7606	0.7593
Dh2	0.6992	0.7912	0.8045	0.7866	0.7779	0.7734
Dh1&Dh2	0.6976	0.7900	0.8044	0.7864	0.7779	0.7733
Mh2&Dh1	0.7650	0.7856	0.7721	0.7635	0.7608	0.7593
Mh2&Dh1&Dh2	0.6976	0.7900	0.8044	0.7864	0.7779	0.7733

Table 3. Similarity, S, for Test 1

Heuristics	Similarity S					
	0	1	2	3	4	5
Dh1	0.7777	0.7997	0.7832	0.7785	0.7758	0.7750
Dh2	0.6999	0.8058	0.8205	0.8064	0.7933	0.7877
Dh1&Dh2	0.6987	0.8041	0.8198	0.8065	0.7931	0.7876
Mh2&Dh1	0.7773	0.8001	0.7837	0.7786	0.7757	0.7749
Mh2&Dh1&Dh2	0.6987	0.8041	0.8198	0.8065	0.7931	0.7876

Table 4. Similarity, S, for Test 2

and the combination of Mh2 and Dh1 show the best results when their thresholds are set to $\mu + 1\sigma$. Dh2, the combination of Dh1 and Dh2, and the combination of Mh2, Dh1 and Dh2 produce the best results when their thresholds are set to $\mu + 2\sigma$. These results will be used for comparison between different ways of session reconstruction in the following subsections.

Heuristics	Data Set for Test 1		Data Set for Test 2		n
	S	S'	S	S'	
Fh1	0.7649	0.7735	0.7794	0.7879	1.0
Fh2	0.7925	0.8084	0.8118	0.8257	1.0
Fh1&Fh2	0.7926	0.8090	0.8114	0.8257	1.0

Table 5. Closest Similarity S' to Similarity S

5.4. Comparison of Two Measures

Different weights of d and d' produce different values of S'. Experiments are carried out to find an n that produces the closest S' to S. The results are presented in Table 5 for Fh1, Fh2, and their combination. The results show that for $n = 1$, the S' is the closest to S, which indicates that the factor of missing pages, d', in our data may be ignored. This is further illustrated in details in Table 6. Table 6 shows the values of the two distance measures d and d' for different heuristics. The distance d' is very small compared to d; thus S' is mainly affected by the distance d (i.e., extra pages in the reconstructed sessions).

Heuristics	Test 1		Test 2	
	d	d'	d	d'
Fh1	0.2265	0.0107	0.2121	0.0087
MFS&Fh1	0.2378	0.0102	0.2238	0.0082
Fh2	0.1916	0.0176	0.1743	0.0145
MFS&Fh2	0.2053	0.0113	0.1901	0.0086
Fh1&Fh2	0.1910	0.0181	0.1743	0.0150
MFS&Fh1&Fh2	0.2034	0.0137	0.1882	0.0116
Dh1	0.1880	0.0266	0.1736	0.0265
Dh2	0.1681	0.0289	0.1529	0.0268
Dh1&Dh2	0.1677	0.0293	0.1525	0.0278
Mh2&Dh1	0.1882	0.0267	0.1732	0.0264
Mh2&Dh1&Dh2	0.1677	0.0293	0.1525	0.0278

Table 6. Test Results for d and d'

5.5. Comparison of Different Heuristics

The results for different heuristics are presented in Table 7 and Figure 3. From the results, it is seen that Dh2 provides the best performance, Dh1 performs better than Fh1, and the combination of Dh1 and Dh2 produces better results than the combination of Fh1 and Fh2. In other words, the simulation results indicate that overall heuristics with dynamic thresholds perform better than heuristics with fixed thresholds. The heuristics Dh1 and the combination of Mh2 and Dh2 produce similar results. Also, the combination of Dh1 and Dh2 and the combination of Mh2, Dh1 and Dh2 produce similar results. Therefore, we can state that in our simulation MF2 did not improve the performance of session reconstruction. Also, we have observed that MFS slightly de-

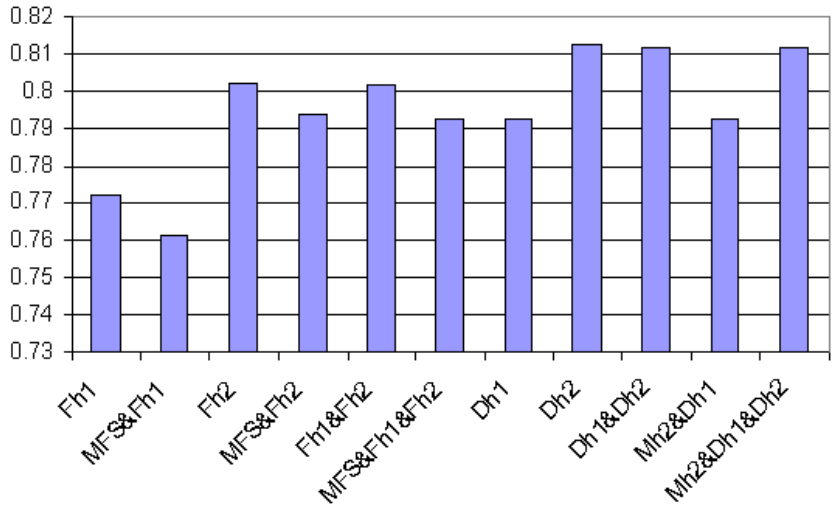

Figure 3. Comparison of different heuristics (Similarity Versus the Type of Heuristic)

creases the accuracy of session reconstruction (see the results of Fh1 and MFS&Fh1 in Table 7 and Figure 3).

Heuristics	Similarity S		
	Test 1	Test 2	Average
Fh1	0.7649	0.7794	0.7722
MFS&Fh1	0.7541	0.7683	0.7612
Fh2	0.7925	0.8118	0.8022
MFS&Fh2	0.7852	0.8020	0.7936
Fh1&Fh2	0.7926	0.8114	0.8020
MFS&Fh1&Fh2	0.7847	0.8009	0.7928
Dh1	0.7858	0.7997	0.7928
Dh2	0.8045	0.8205	0.8125
Dh1&Dh2	0.8044	0.8198	0.8121
Mh2&Dh1	0.7856	0.8001	0.7929
Mh2&Dh1&Dh2	0.8044	0.8198	0.8121

Table 7. Similarity S for Different Heuristics

6. Conclusion and Future Work

Session reconstruction reconstructs Web usage data into user sessions. Two time-oriented heuristics, Fh1 and Fh2, are commonly used for session reconstruction. We used statistical analysis and usage mining techniques to improve Fh1 and Fh2. The new improved heuristics are called Dh1, Dh2 and Mh2. Experimental results show that statistical analysis is useful and improves the performance of Fh1 and Fh2 heuristics. Even though the application of usage mining (i.e., finding maximal frequent sequences) was not advantageous in our experiments, we believe that with proper combination of time-oriented heuristics and MFS better performance might be possible. We will further explore this in our future work. Other future works will include: exploring ways of improving the performance of Mh2, extracting Web access patterns other than MFSs from sufficient historic data to improve the session reconstruction performance, and using other measurements to evaluate the performance of the proposed session reconstruction heuristics.

7. Acknowledgments

This work was funded by the Atlantic Canada Opportunity Agency (ACOA) through the Atlantic Innovation Fund (AIF) and through grant RGPIN 227441-00 from the National Science and Engineering Research Council of Canada (NSERC) to Dr. Ali A. Ghorbani.

References

[1] B. Berendt, B. Mobasher, M. Nakagawa, and M. Spiliopoulou. The impact of site structure and user environment on session reconstruction in web usage analysis. In *Proceedings of the 4th WebKDD 2002 Workshop, at the ACM-SIGKDD Conference on Knowledge Discovery in Databases*, Edmonton, Alberta, Canada, July 2002.

[2] P. Bieganski, J. Riedl, and J. Carlis. Generalized suffix trees for biological sequence data: Applications and implementation. In *Proceedings of the 27th Annual Hawaii International Conference on System Sciences*, pages 35–44. IEEE, IEEE Computer Society Press, 1994.

[3] L. Catledge and J. Pitkow. Characterizing browsing behaviors on the World-Wide Web. *Computer Networks and ISDN Systems*, 27(6):1065–1073, 1995.

[4] R. Cooley, B. Mobasher, and J. Srivastava. Data preparation for mining World Wide Web browsing patterns. *Knowledge and Information Systems*, 1(1):5–32, 1999.

[5] D. Gusfield. *Linear-Time Construction of Suffix Trees*. Cambridge University Press, New York, NY, USA, 1997.

[6] M. Kilfoil, D. A. Ghorbani, W. Xing, Z. Lei, J. Lu, J. Zhang, and X. Xu. Toward an adaptive web: The state of the art and science. In *Proceedings of Communication Network and Services Research (CNSR) 2003 Conference*, pages 108–119, Moncton, NB, Canada, May 15–16 2003.

[7] R. Kosla and H. Blockeel. Web mining research: A survey. *SIG KDD Explorations*, 2(15):1–15, July 2000.

[8] S. K. Pal, V. Talwar, and P. Mitra. Web mining in soft computing framework : Relevance, state of the art and future directions. *IEEE Trans. Neural Networks*, 13(5):1163–1177, 2002.

[9] J. Pei, J. Han, B. Mortazavi-asl, and H. Zhu. Mining access patterns efficiently from web logs. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 396–407, 2000.

[10] M. Spiliopoulou, B. Mobasher, B. Berendt, and M. Nakagawa. A framework for the evaluation of session reconstruction heuristics in web usage analysis. *INFORMS Journal of Computing, Special Issue on Mining Web-Based Data for E-Business Applications*, 15(2):171–190, 2003.

[11] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[12] Y. Xiao and M. H. Dunham. Efficient mining of traversal patterns. *Data and Knowledge Engineering*, 39(2):191–214, November 2001.