A Performance Comparison of Pattern Discovery Methods on Web Log Data

Murat Ali Bayır, Ismail H. Toroslu, Ahmet Coşar
Department of Computer Engineering
Middle East Technical University
E-Mail: {ali.bayir, toroslu, cosar}@ceng.metu.edu.tr

Abstract

One of the popular trends in computer science has been development of intelligent web-based systems. Demand for such systems forces designers to make use of knowledge discovery techniques on web server logs. **Web usage mining** has become a major area of knowledge discovery on World Wide Web. **Frequent pattern discovery** is one of the main issues in web usage mining. These frequent patterns constitute the basic information source for intelligent web-based systems. In this paper; frequent pattern mining algorithms for web log data and their performance comparisons are examined. Our study is mainly focused on finding suitable pattern mining algorithms for web server logs.

1. What is Web Mining?

Extracting valuable knowledge from Web data has become more popular. There has been huge interest towards web mining [Cooley97]. Two different approaches are proposed on the definition of web mining. One approach is process-based and the other is data-based. Data-based definition is more widely accepted today. In this perspective, web mining is the application of data mining techniques to extract knowledge from Web data, where at least one of structure (hyperlink) or usage (Web log[CERN]) data is used in the mining process (with or without other types of Web data) [Srivastava04] . There are no differences between web mining and data mining compared in general.

Web mining is broadly classified in three categories according to the data-based approach.

1.1 Web Content Mining: Web content mining is the extraction of useful information from content of web documents. These documents contain data in both multimedia and text format. Web content mining is the most widely researched area nowadays. Some of the research issues are topic discovery, extracting association patterns, clustering of web documents and classification of web pages. These research areas contain methods from other fields like information retrieval and NaturalLanguageProcessing(NLP).

1.2 Web Structure Mining: A web graph [Cooper01, Kumar00] represents web pages as nodes, and hyperlinks are represented with edges connecting two graph nodes (Figure 1). Web Structure mining can be described as discovering web structure and link topology information from web. This type of mining can be divided into two categories with respect to the examined structure element. One approach is hyperlink based mining. A hyperlink connects one page to another page which could be either in the same web server (local) or at another site (remote). Second approach is document based mining. Any web page can be organized in a tree structure based on HTML and XML tags in the web page.

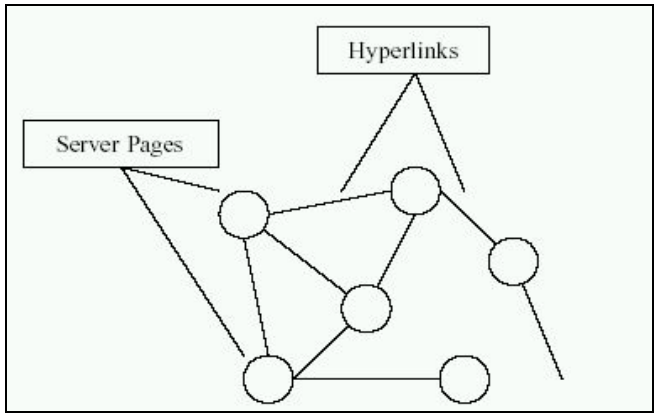


Figure 1: Graph model of www.

1.3 Web Usage Mining: Web usage mining is application of data mining techniques to discover user access patterns from web data. Web usage data captures web-browsing behavior of users from a web site. Web usage mining can be classified according to kinds of usage data examined. In our context, the usage data is Access logs on server side, which keeps information about user navigation. Our work is mainly focused on web usage mining including discovery of user navigation patterns from access logs. In the next section, the structure of access logs as our main data source is introduced.

2. Data Source for Web usage Mining

The basic information source for web usage mining is the access log files at server side. When any user agent (IE, Mozilla, Netscape, etc) hits an URL in a domain, the information related to that operation is recorded in an access log file. The web log data can be preprocessed in order to obtain session information of any user. Data reduction techniques can be applied to web log data in order to obtain raw sets for data mining.

Basically an access log file contains its information in Common Log Format [CERN]. In this format, each user request to any URL corresponds to a record in access log file. Each record is a tuple containing 7 attributes. Session information is 2-tuple containing an IP address of user and a sequential list of web pages that are visited in this session.

$S_i = (IP_i, PAGES_i)$
$PAGES_i = \{ (URL_i)_1 , (URL_i)_2 , \ldots\ldots\ldots\ldots\ldots\ldots (URL_i)_k \}$

After applying data preprocessing and data reduction, session information is obtained from web log data. In the next section data preprocessing step is given.

3) Data Preprocessing of Access Log Data

Access log file on the server side contains log information of user that opened a session. These logs include the list of items that a user agent has accessed. The log format of the file is CERN (Common Log Format) [CERN]. , which includes special record formats. These records have some common fields, which are:

1. User's IP address
2. Access date and time
3. Request method (GET or POST),
4. URL of the page accessed,
5. Transfer protocol (HTTP 1.0, HTTP 1.1,),
6. Success of return code.

7. Number of bytes transmitted.

The information in this record is sufficient to obtain session information. In our context, the information that is going to be extracted is defined as the click - stream in a user session for a particular web server. A click - stream is defined as a series of page view requests. Also the parser [Martinez02] described below transforms a set of logs, L, expressed as,

$L = \{L_1\ L_2, L_3\ \ldots L_i, \ldots\ldots L_{|L|}\}\ \ , \forall i \leq |L|.$
$|L|$: The number of records in the Log file.
$L\ \{IP_i, TIME_i, METHOD_i, URL_i, PROT_i, CODE_i, BYTES_i\}$

Into a set of session S such that
$S = \{ S_1\ S_2, S_3\ \ldots S_i, \ldots\ldots S_{|S|}\}\ \ , \forall i \leq |S|.$
$|S|$: The number of discrete sessions
Each session S_i contains IP_i , $PAGES_i$ that are navigated in that session.
$S_i = \{ IP_i\ PAGES_i\}$
$PAGES_i = \{$URL i-1, URL i-2, …….. URL i-k $\}$

Each URL represents a vertex in graph model; *k* is the number of pages requested by user agent connected from IP_i in session S_i. The set of URLs that form a session must satisfy the requirement that the time elapsed between two consecutive requests is smaller than a given *t*, which is accepted as 30 minutes [Catledge95]. The algorithm used by the parser [Martinez02] is as follows:

L : The set of input logs
|L| : the number of input logs
Δt : time interval
S : The set of sessions
|S| : The number of sessions

input : L, |L| , Δt
Output : S, |S|

```
Function Log_Parser ( |L|,L, Δt)
    For each Li of L
        If Methodi is 'GET' AND Urli is 'WEBPAGE'  // 1. if
            If ∃ Sk ∈ Open_sessions with IPk = IPi then // 2. if

                If ( (Timei – END_TIME(Sk) ) < Δt)  then  // 3. if
                    Sk = (IPk , PAGESk ∪ Urli)
                Else
                    Close_session(Sk)
                    Open_session(IPi, Urli)
                End if                          // end of 3. if
            Else
                    Open_session(IPi, Urli)
            End if                              // end of 2. if
        End if                              // 1. if
    End For
```

Function END_TIME returns the maximum time detected in the page set of corresponding session. Function CLOSE_SESSION removes corresponding session from Open_sessions set. Function OPEN_SESSION adds corresponding session to open sessions set. When a URL_i is being considered the algorithm checks whether the given URL is in valid format. That means, if any incomplete or invalid format URL is found, the algorithm discards the corresponding log entry.

Clearly the parser above transforms data definition of logs containing 7 attribute tuples into
L_i = {IP_i, $TIME_i$, $METHOD_i$, URL_i, $PROT_i$, $CODE_i$, $BYTES_i$ }
Another data format
S_i = (IP_i $PAGES_i$) having two tuple. Simultaneously data cleaning is performed in
URL_i → $PAGES_i$ conversion. After that each session is reduced to S_i = ($PAGES_i$) format. All sessions are in enumerated form. We have only a sequential list of pages that were visited in session S. This phase completes the data reduction step. .

4) Frequent Pattern Mining Algorithms on Raw Data

Pattern discovery is the main issue in both web usage mining and data mining. Many algorithms have been proposed on capturing frequent user navigation patterns. Finding desired patterns are quite challenging in very large data. The search space increases exponentially as pattern length increases. Also, discovered patterns must be interpreted and understandable knowledge must be extracted from them.

In this section, we give the algorithms for four frequent pattern mining algorithms . These algorithms are SPADE [Zaki01], GSP [Srikant96], and Bread First Search and Depth First Search algorithms. Their comparisons and performance analysis will be presented.

Except GSP, the algorithms referred above are executed on lattice model. All patterns are classified with respect to their lengths. These patterns will form a lattice based on pattern-length and pattern-frequency.

In the next section, construction of pattern lattice based on length is discussed.

4.1 Basic Definitions

Before construction of pattern lattice is discussed, some basic definitions are given.

Definition of Lattice: An algebra (L; $\wedge$;$\vee$) is called a lattice, if L is a nonempty set and $\wedge$ and $\vee$ are binary operations on L. Both $\wedge$ and $\vee$ are idempotent, commutative, associative and they satisfy absorption law.

Idempotent	Law: If A is a set, $A \wedge A = A$, $A \vee A = A$.
Commutative	Law: If A and B are sets, $A \wedge B = B \wedge A$, $A \vee B = B \vee A$,
Associative	Law: If A, B and C are sets $A \vee (B \vee C) = (A \vee B) \vee C$ $A \wedge (B \wedge C) = (A \wedge B) \wedge C$
Absorption	Law: If A and B are sets $A \wedge (A \vee B) = A \vee (A \wedge B) = A$

Atoms: Each single webpage in web domain called atom corresponding to a single node in web graph.

Pattern: Any Pattern consists of a sequential list of atoms.

Session id-timestamp list: Session id- timestamp list is the basic structure for our purposes. After data preprocessing step we obtain a series of web pages visited in each session. Each session is in the format given below:

$S_i = Page_{i-1} \rightarrow Page_{i-2} \rightarrow$ ……..$Page_{i-k} \rightarrow$ …….. $\rightarrow Page_{i-|Si|}$

The number k is the number of pages in the corresponding session. It can be used as a timestamp as follows. Any pattern P_j is visited before P_k in session S_n if $j < k$. Session id timestamp list is a list which keeps session id and timestamp information for any patterns in all sessions. The timestamp information keeps the the order of last atom for patterns with length > 1. It keeps the order of atoms with length=1.

For example we have 4 web pages. $Page_1$, $Page_2$, $Page_3$, $Page_4$ are our atoms. We have 3 sessions given below.

$S_1 = Page_1 \rightarrow Page_2 \rightarrow Page_4 \rightarrow Page_1 \rightarrow Page_3$
$S_2 = Page_4 \rightarrow Page_3 \rightarrow Page_1 \rightarrow Page_2$
$S_3 = Page_3 \rightarrow Page_4 \rightarrow Page_1$

Session id-timestamp list of these four pages are given below.

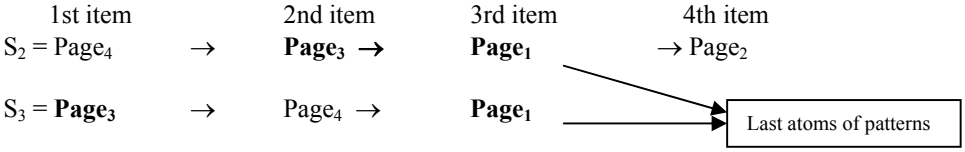
Table 1: Session id-timestamp list.

$Page_1$		$Page_2$		$Page_3$		$Page_4$	
Session	Timestamp	Session	Timestamp	Session	Timestamp	Session	Timestamp
1	1	1	2	1	5	1	3
1	4	2	4	2	2	2	1
2	3			3	1	3	2
3	3						

The *Session id-timestamp list* of pattern $Page_3 \rightarrow Page_1$ is

Table 2: Session id-timestamp list of pattern $Page_3 \rightarrow Page_1$

$Page_3 \rightarrow Page_1$	
Session	Timestamp
2	3
3	3



Support of Pattern: The support of any length-k pattern

$P = Page_1 \rightarrow Page_2 \rightarrow$ ……..$Page_k$. A session S supports pattern P if session S contains all atoms in pattern P and . $\forall P_i \in P$, $P_j \in P$ satisfying $i \leq j$ implies $\exists P_k$, P_l, S (Session) where $P_k = P_i \wedge P_j = P_l \wedge P_k \in S \wedge P_l \in S \wedge k \leq l$. That means if $Page_i$ comes before $Pages_j$ in pattern P, same thing holds in all sessions S which support pattern P.

The support of Pattern P is defined as:

Support(P) = (# of instances of sessions support P) / (# of all sessions).
The Support of pattern with length *n* can be calculated

Prefix based n-legth equivalanece class : The Pattern P_1 and P_2 with length *n* are said to be in the same equivalance class if
$P_1 = P \rightarrow Atom_x$ and $P_2 \rightarrow Atom_y$. Where |P| = n-1. The notation for prefix- based equivalence class is given as: $P_1 \in [P]$ and $P_2 \in [P]$.

The support of pattern P with length n+1 can be calculated by using two length-n patterns in the same prefix based class the union of which equal to P.
For example The pattern $P = Page_4 \rightarrow Page_1 \rightarrow Page_3$ can be written as

$P = Page_4 \rightarrow Page_1 \vee Page_4 \rightarrow Page_3$. The possible outcomes of this operation are $Page_4 \rightarrow Page_1 \rightarrow Page_3$ and $Page_4 \rightarrow Page_3 \rightarrow Page_1$. Input patterns $P_1 = Page_4 \rightarrow Page_1$ and $P_2 = Page_4 \rightarrow Page_3$ are both in the same prefix class such that $Page_4 \rightarrow Page_1 \in [Page_4]$ and $Page_4 \rightarrow Page_3 \in [Page_4]$.
The support of P can be calculated by using *Session id-timestamp list* of P_1 and P_2 .

For example, two lists for P_1 and P_2 are given in table 3.

Table 3: Two example session id – timestamp lists.

P_1		P_2	
Session	Timestamp	Session	Timestamp
2	6	2	16
5	12	6	4
4	3	9	7
9	10		

Let's say we have 10 session Support (P_1)=0.4 and Support (P_2)=0.3

For P_1 session=1 and Timestamp=6 means that if Session 1 is given as:
$S_1 = Url_{1-1} \rightarrow Url_{1-2} \rightarrow Url_{1-3} \rightarrow Url_{1-4} \rightarrow Url_{1-5} \rightarrow Url_{1-6} \rightarrow \ldots\ldots\ldots\ldots\ldots Url_{1-|S1|}$
$P_1 = Page_4 \rightarrow Page_1$ Then

$Page_4 \subset Url_{1-1} \rightarrow Url_{1-2} \rightarrow Url_{1-3} \rightarrow Url_{1-4} \rightarrow Url_{1-5}$ and $Url_{1-6} = Page_1$ The timestamp gives the order of last atom of corresponding patterns.

Now calculate *Session id-timestamp list* of
$Page_4 \rightarrow Page_1 \rightarrow Page_3 = Page_4 \rightarrow Page_1 \vee Page_4 \rightarrow Page_3$. The pattern P_1 and P_2 are common in sessions 2,9. However, the last element of P is $Page_3 \in P_2$ so in the common sessions the timestamp (P_2) > timestamp (P_1) The session 2 satisfies, but the session 9 does not satisfy. So we only take session 2 and timestamp of P becomes 16.

Session id-timestamp list of P is shown in Table 4.

Table 4: Session id – timestamp list of $P = Page_4 \rightarrow Page_1 \rightarrow Page_3$.

P	
Session	Timestamp
2	16

Support (P) = (1/10) = 0.1

4.2 Construction of Pattern Lattice

Since the basic element of pattern is atom, the bottom elements of patterns lattice are composed of single atoms. Each single atom stands for the length-1 prefix equivalence class. Beginning from bottom elements the frequency of upper elements with length *n* can be calculated by using two n-1 length patterns belonging to the same class.

Example:
As an example, consider that we three atoms corresponding to three web pages P_1 , P_2 and P_3 . Example lattice up to some length-3 patterns and complete length-2 patterns are given below.

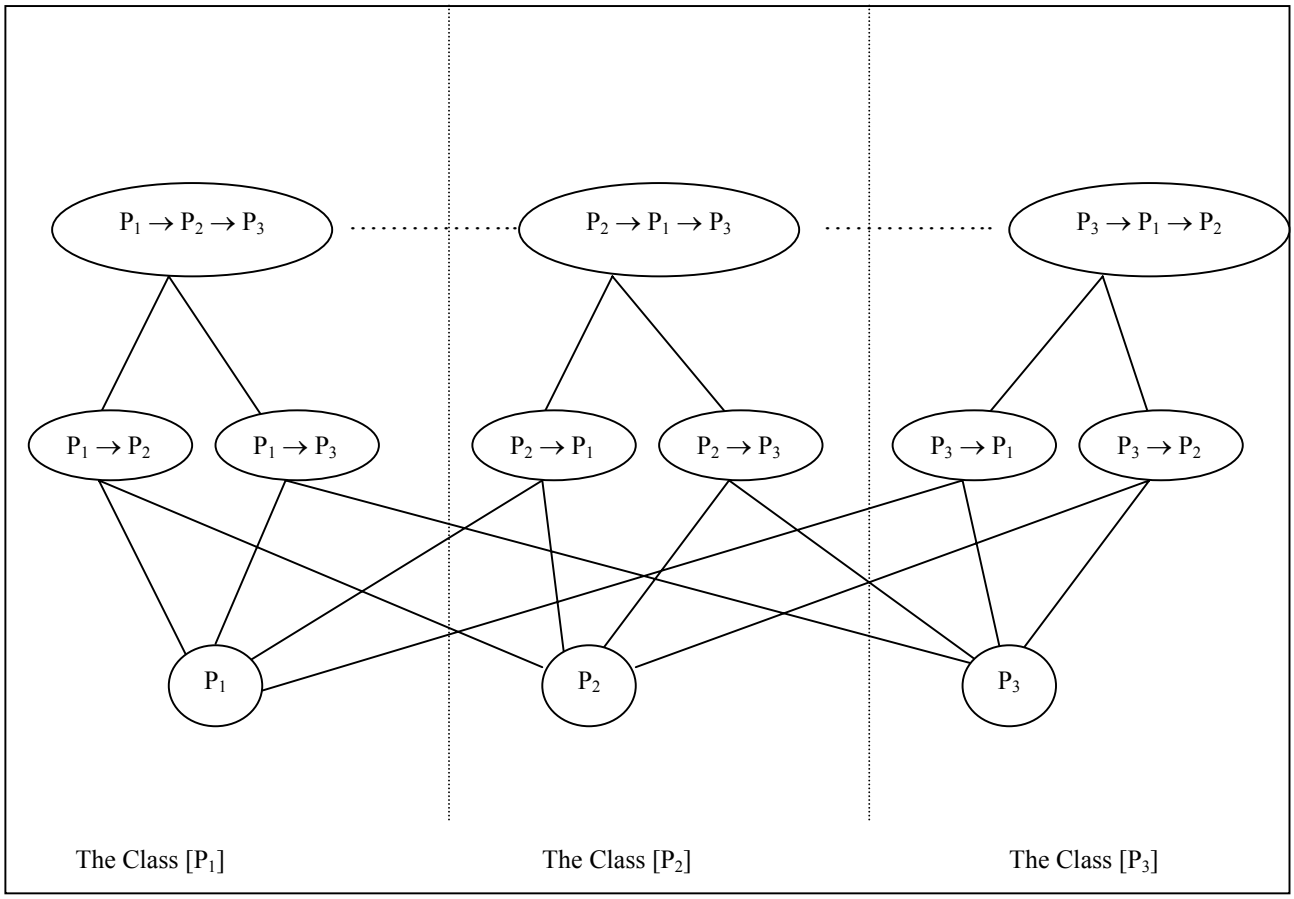


Figure 2: Prefix based classification of pattern lattice.

4.3 Search For Frequent Patterns

Breadth First Search: In breadth first search, the lattice of equivalence classes is generated by the recursive application, exploring the whole lattice in a bottom up manner. All child length-n patterns are generated before moving into parent patterns with length n+1. For example the patterns $P_1 \rightarrow P_2$ and $P_1 \rightarrow P_3$ are generated before $P_1 \rightarrow P_2 \rightarrow P_3$ and $P_1 \rightarrow P_3 \rightarrow P_2$. In each generation, the user defined threshold is checked to see whether it is satisfied or not. If the frequency of current pattern is below the user defined threshold *t*, the current pattern does not participate in constructing parent patterns. If the child pattern is not frequent then its parent patterns cannot be frequent. Pruning strategy applies here.

Algorithm:

Construct Session id-timestamp list for all atoms
Calculate support of each atom.

For n=1 to max length session
Eliminate patterns having frequency < user threshold

For each pattern pairs with length-n in same prefix (n-1) class
Construct Session id-timestamp list of parent patterns length n+1
Calculate support of each pattern
End For

End For

Depth First Search: In the depth first search, all patterns are explored by a single path coming from its child before exploring all patterns having smaller length from corresponding pattern. In breadth first search we only need to keep the id lists of current patterns with length *n* in the memory.

Algorithm:

Pattern_list : = all atoms

Function explore_patterns (Pattern list)
Pattern x := first_element(Pattern_list)
Remove x from pattern_list
Construct Session id-timestamp list for x
Calculate support of x
If support(x) < threshold
Prune and return
Else
For each Pattern with P with length = length(X) + 1 and P ∈ [X]
explore_patterns (P)
End for
End function

GSP Algorithm: GSP makes multiple passes over the session set. Given a set of frequent n-1 patterns, the candidate set for next generation are generated from input set according to the thresholds. Only frequent patterns in the current set are considered for generating the next candidate sequence. A pruning phase eliminates subsets of infrequent patterns. For all patterns P in the candidate set with length k, all sessions are processed once and the count is incremented for each detected pattern in the candidate set.

Algorithm:

$Pattern\_list_1$: = Frequent atoms;

For k=2 ; $Pattern\_list_k$!= empty ; k = k + 1
C_k = set of candidate
For all sessions in S do
Increment count of $X \in C_k$ contained in S
End For
$Pattern\_list_k = \{ X \in C_k /$ support(X) > threshold $\}$
End For
Set of all frequent Patterns := $\cup_k Pattern\_list_k$

SPADE Algorithm: In the SPADE algorithm, firstly *Session id-timestamp list* of atoms created. Then these lists are sorted with respect to the support of each atom. Then, if support of any atom is below the

input threshold it is eliminated automatically. Next, frequent patterns from single atoms are generated according to union operation $\vee$ based on prefix-based approach defined above. Finally, all frequent items with length n > 2 are discovered in their length-1 prefix class [Atom] independently. While exploring search space both depth first and breadth first search can be used, however breadth first search has the advantage of keeping only current length-k pattern in the memory.

Algorithm:

SPADE(min_sup,S)

P_1 = all frequent atoms with their Session id-timestamp list
P_2 = all frequent length-2 patterns with their Session id-timestamp list
E = All equivalance classes of Atoms $\in P_1$

For all [X] $\in$ E do
Construct pattern lattice of [X]
Explore frequent patterns in [X] by
Using either Depth first or Breadth first search.
End For

End function

---

5) Experimental Results

In this section performance comparison of four algorithms on web logs of our departmental web server is given.

5.1 Frequent patterns of a web server (www.ceng.metu.edu.tr)

All experiments are done using the access logs of our web server at the Computer Engineering Department. These logs contain requests of web pages in our departmental web server, The log files contain records of one month. General statistics of server pages that we worked on is given in the Table 5.

Table 5: Web usage statistics.

Successful requests:	439,241
Average successful requests per day	15,494
Successful requests for pages	172,005
Average successful requests for pages per day	6,067
Failed requests	49,507
Redirected requests	18,612
Distinct files requested	11,614
Distinct hosts served	15,726
Data transferred	9.59 gigabytes
Average data transferred per day	346.32 megabytes

In the next step we have performed preprocessing to extract only web page records from log file in order to apply our algorithm. Table 6 shows the information about file types from the log files.

Table 6: Log file types.

File Type	**# of requests**	**Proportion**
Web Pages (html,htm,php..)	155,779	%35.46
Gif	152,458	%34.70
Jpg	31,715	%7.22
Pdf	17,046	%3.88
Other types	82,243	%18.74
Total	439,241	%100

The algorithm is going to be applied on %35.46 of total requests including web pages. Among these records of web pages, we evaluate applicability and scalability of our algorithm. In Table 7 we give the most popular 15 web pages (ATOMS) with respect to the number of requests.

Table 7: Hit ratios of popular pages.

URL of web page	**# of requests**	**Proportion**
/	19988	%12.83
/course/	8081	%5.18
/courses/ceng230/	3528	%2.26
/~ceng140/	3481	%2.23
/course/2xx.html	3217	%2.06
/~ceng140/nav/	3004	%1.92
/courses/ceng230/announcements.html	2971	%1.90
/courses/ceng230/left.html	2968	%1.90
/courses/ceng230/top.html	2895	%1.85
/courses/ceng230/students/std.info.php3	2521	%1.61
/course/3xx.html	2253	%1.44
/~ceng140/homeworks.html	1814	%1.16
/people/	1792	%1.15
/~ceng140/index_files	1462	%0.93
/tanitim/jpegs/sld.php3	1418	%0.91

After that we apply four algorithms to find the most frequent patterns with length k=2 and k=3. In Table 4 and Table 5, most frequent patterns with their frequencies are given.

Patterns with length k =2 and frequency_threshold = 0.05 (lower bound for total frequency of patterns with length k=2)

Table 8: Patterns with length 2 and high frequency.

Patterns	Frequency > 0.05
Pattern1 Item List / → /course/	0.110
Pattern2 Item List / → /~ceng140/	0.077
Pattern3 Item List / → /courses/	0.073
Pattern4 Item List / → /people/	0.051
Total Frequency	**0.311**

Patterns length k=3 :
fitness_infima = 0.02 (lower bound for frequency of most frequent patterns), frequency_threshold = 0.01 (lower bound for total frequency of patterns with length k=3)

Table 9: Patterns with length 3 and high frequency

Patterns	Frequency > 0.02
Pattern1 Item List / → /~ceng140/→/~ceng140/nav/	0.049
Pattern2 Item List / → /courses/ → /courses/ceng230/	0.038
Pattern3 Item List / → /course/→/course/3xx.html	0.022
Total Frequency	0.109

When pattern lengths becomes k=4 the frequency of most frequent patterns remains below 0.001. In that case total frequency of frequent patterns is below the threshold for total frequency . So there is no need to search for frequent patterns with length $k \geq 4$.

5.2 Comparion of algorithms based on discovered frequent patterns

Figure3 compares discovery time of all frequent patterns with length k= 1 to 5. The threshold for length k is given as 10^{-k} .

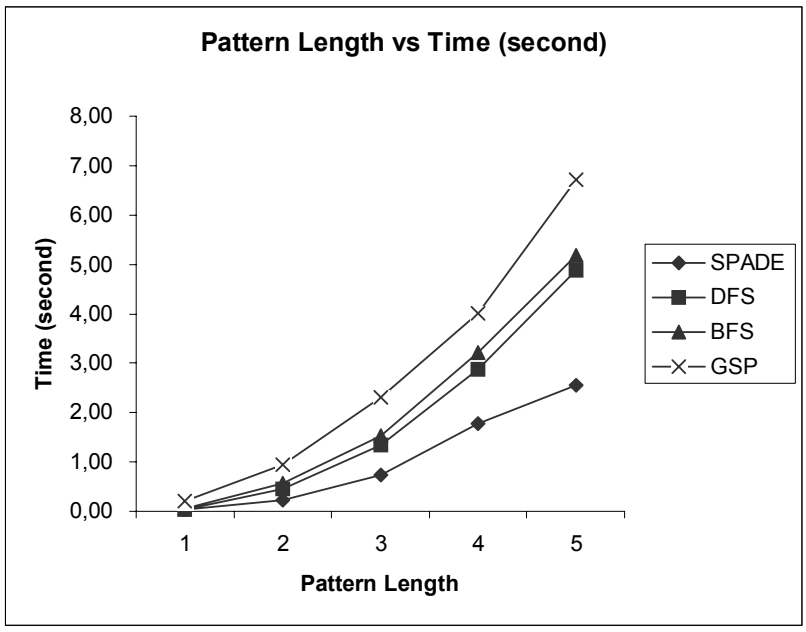


Figure 3: Comparison of mining methods' execution times.

In our experiments GSP has given the worst result because it does not use pattern lattice structure and at each step it has to perform a session scan. DFS is better than BFS because it eliminates infrequent patterns at each level and in the memory it keeps less patterns at each step. SPADE is the best one, because it works on prefix-based classes which is a much smaller search-space. Also, in the first step it calculates all frequent length-2 patterns which eliminates most of the infrequent patterns that are possible to come across during exploration of the search space. Another observation is that, session-id time stamp list structure provides unneccessary database scans for evaluating frequency of length 1 and 2 patterns.

References

[Catledge95] Catledge, L.,Pitkow, J., *"Characterizing browsing behaviors on the world wide web"*, in Computer Networks and ISDN Systems, 27(6), 1995.

[CERN] CERN Common Log Format,
http://www.w3.org/Daemon/User/Config/Logging.html#common-logfile-format

[Cooper01] Cooper, C. Frieze A., *"A general model of Web graphs"*, In ESA, pages 500-511, 2001.

[Cooley97] Cooley, R., Mobasher, B., and Srivastava, J. *"Web mining: Information and pattern discovery on the world wide web"*. In proceedings of the 9th IEEE International conference on Tools with Artifical Intelligence (ICTAI' 97), Newposrt Beach, CA, 1997.

[Kumar00] Kumar R., Prabhagar R., Sridhar R., *"The Web As a Graph"*, 19th ACM SIGACT-SIGMOD-AIGART Symp, 2000.

[Martinez02] Martinez E., Karamcheti V., *"A prediction Model for User Access Sequences"*, In WEBKDD Workshop: Web Mining for Usage Patterns and User Profiles, July 2002.

[Srikant96] Srikant R., and Agrawal R. *"Mining Sequential Patterns: Generalizations and performance improvements"*, In Proc. 5th Int'l Conference Extending Database Technology (EDBT), Avignon, France, March 1996.

[Srivastava04] Srivastava J., Desikan P., Kumar V., *"Web Mining- Accomplishments & Future Directions"* IASTED Conference Series, St. Thomas, U.S. Virgin Islands, 2004.

[Zaki01] Zaki M. J., *"SPADE: An Efficient Algorithm for Mining Frequent Sequences"*, in Machine Learning Journal, special issue on Unsupervised Learning (Doug Fisher, ed.), pages 31-60, Vol. 42 No. 1/2, 2001.