Finding User Navigation Patterns to Improve Website Usability

Nariman Farsad
nariman@cse.yorku.ca

April 21, 2010

Abstract

In the past two decades Internet has grown exponentially. As the result the demand for making more user friendly websites has also grown. Huge amount of data about user navigation patterns are available in server log files. These data need to be mined in order to extract useful patterns that can be used to improve user experience. In this paper we look at different methods of extracting user navigation patters from web server log files. In particular we compare the performance of GSP, WAP-Tree, Pre-Order WAP-Tree, and HPG algorithms using a real website, called music sheet exchange. Our experimental results show that HPG is a better mining algorithm for user navigation pattern compared to the other algorithms.

1 Introduction

The Internet has evolved significantly over the past few decades. With higher bandwidths available due to improvements in telecommunication and widespread availability of Internet, concepts such as cloud computing and web based applications are now a reality. For example, Google Inc. has introduced its own set of web-based applications such as a word processor and a spreadsheet program. With the grows of web-based applications and Internet in general, the design of web-based user interface is becoming extremely important.

One of the greatest advantages of designing web-based user interfaces over traditional user interfaces is the ability to keep track of user interactions with the site. Thanks to the simple (yet extremely useful) concept of server log files, users' interaction with a website is kept in a raw format that can be easily processed by automated tools. This information is stored on most web servers by default. Additionally, Unix-based web servers already include all the tools needed to extract the information from these raw log files. The extracted information can then be used for finding user navigation patterns.

By finding frequent user navigation sequences or user navigation sessions from server logs, we can compare actual user navigation trails with the designer's expected navigation trails and try to improve the interface of the site accordingly. This involves: (1) pre-processing the log files and converting it to sequential data and (2) finding user navigation patterns from the sequential data using different pattern finding algorithms.

In this paper we find user navigation patterns of a music sheet sharing service called Sheet Exchange (the website's address is sheetexchange.com). The raw log files from the Apache web server on which the Sheet Exchange website resides are first simplified and converted into sequential data. Then a number of pattern finding algorithms, namely Generalized Sequential Pattern (GSP) [1], Web Access Pattern tree (WAP-Tree) [2], Pre-order Linked WAP-Tree (PLWAP) [3] as well as Hypertext Probabilistic Grammars (HPG) [4], an algorithm designed specifically for mining user navigation patterns, are applied to this sequential data and the results are compared. Finally a few examples of resulting patterns is examined and the benefits of our finding to usability improvements is discussed.

To compare these algorithms we use two measuring factors. First, we consider the number of patters that are found using each algorithm. Since, the number of patterns do not indicate the quality, we also consider the inputs from the owner of the website as a second measure for quality. The owner is asked to look at the different results form different algorithms and decide which algorithm found more meaningful and non-trivial patterns. Based on our results HPG is a superior algorithm and can find more meaningful patterns.

The remainder of this paper is organized as follows. In Section 2, we present the raw log file structure and discuss the preprocessing step required for conversion to sequential data. The pattern finding algorithms are then discussed in Section 3. In Section 4 and the Appendix B and C we discuss the implementation of the algorithms. The results are presented and discussed in Section 5. Finally we conclude the report in Section 6.

2 Preprocessing

We use real logs from a music sheet exchange service website (http://www.SheetExchange.com). This website allows musicians to share information regarding the music sheets (notes) that they own and would like to share. Users are able to browse or search for sheets shared by other users using different criteria and if they find a sheet they are looking for, they can contact the owner of the sheet to arrange an exchange. Similarly the site keeps track of the sheets that are requested by a user to arrange for a better exchange.

In order to allow certain pages and commands to be bookmarked the site uses GET arguments in the URLs. This will require a great deal of pre-processing on the data before the raw log data can be converted into sequential data. First, the log file is purified and all the irrelevant lines are removed. We then simplify the URLs. Since our goal is to extract information related to the structure of the site, we need to generalize arguments passed to URLs that are too specific. For example, if an argument-value pair such as sheet_id=20 is passed to the page, we need to filter the values to get the same URL regardless of the ID number of a sheet. One easy solution is to simply delete all arguments and use page names only. However, since for this specific dataset file names are passed as modules (e.g. index.php?m=search instead of search.php), we need to convert these module arguments to file names. Additionally, some arguments could potentially be important because they specify through which link the page was accessed.

Once the log file is in its purest form, we need to find a good criteria such as the referring page or time threshold to cluster page requests into user sessions. Note that in a log file users are identified only by their IP address and since the IP addresses might change between sessions, we can only extract one visitor session. In other words we can't accurately attach these sessions to an actual user of the site or track returning users accurately.

Converting logs to sequential data can be done manually, using web-log analyzer or data mining software. In the next section we describe the details of preprocessing step.

2.1 Detailed Explanation

The target site uses GET arguments and modules are passed to a single page (index.php) as GET arguments (e.g. index.php?m=search loads the search module). Our goal is to extract the navigation pattern related to the interface of the site so we need to consider users' navigation between these modules (pages). There is a large number of arguments passed to some pages. Some arguments are very useful because they tell us how a page is accessed by the users and some are ignorable. If we keep all argument-value pairs, similar pages would be identified as different items just because they have a different value passed to them.

We use a Linux scripts along with a web data mining software called WUM [5] for preprocessing. First, WUM is used to remove the irrelevant lines. These include lines that are not related to the target website, all lines that are not web pages (images, css files, javascript files, web icons, etc.). Invalid requests passed by bots (or hackers) are also removed. Since, web URLs use %20 instead of space, all %20 are converted to space.

Since the website uses GET arguments and modules, we remove either values of arguments or complete argument-value pairs as necessary. For some arguments (for example user_id) we needed to know the argument that was passed but not the actual value of the arguments. For example, when a user_id is passed to sheets.php, it shows sheets specific to a user. This is different from calling sheets.php without this argument which shows the latest added sheets sorted in reverse chronological order. Thus, it is important to keep the argument itself. However, if the value of user_id is kept, two similar requests with different user_id values would be considered two different items. Since we don't want that we remove the value to make the items similar. Some arguments-value pairs are completely unnecessary and are removed. For example sheet_title is a redundant argument (perhaps for efficiency) because whenever sheet_title was passed to a page another argument (sheet_id) was also passed: sheet_id is the unique identifier of a sheet while sheet_title is the user friendly name of that sheet. We further simplify the results by converting module arguments to page names. For example, index.php?m=sheets was changed to sheets.php.

All the modifications described in the previous two paragraphs are made using Linux scripts followed by WUM software. The resulting output is a comma separated sequence of web pages visited in one user session. We enabled the option in WUM to use referring URL in addition to the default IP address and time threshold pair to extract user sessions. We used a time threshold of 30 minutes for each session and converted the simplified log (over 18000 lines) to 3415 user session sequences. In the final output file, WUM stores a sequence ID and one user session per line. In order to make this file readable by different pattern finding algorithms we also created a simple Linux script that would include a sequence ID and the number of items in the sequence followed by a tab-delimited user session per line.

An example of each step of the preprocessing procedure is presented Appendix A at the end of the report.

3 Pattern Finding Algorithms

In this section we will consider four pattern finding algorithms that are applicable to user navigation patter mining. These algorithms are GSP [1], WAP-Tree [2], Pre-order Linked WAP-Tree [3] and Hypertext Probabilistic Grammars (HPG) [4].

The first algorithm we will use is the Generalized Sequential Pattern (GSP) mining algorithm. Since GSP is a well known algorithm in sequential mining, we skip the detailed description of this algorithm. In the next sub sections we describe the other three pattern finding algorithms in details.

3.1 WAP-Tree

The GSP algorithm was one of the first sequential patter mining algorithms developed. It uses Apriori like algorithm to find frequent sequences. However just like Apriori algorithm GSP has the problem of generating a lot of candidates. It also requires multiple scans of the database. To solve these issues Pei and et al. [2] have proposed a new method very similar to FP-grow [6] algorithm that needs to scan the dataset twice. The following is a brief description of how the algorithm works.

The database is scanned once to find frequent 1-element sequences, then a second time to construct the WAP-tree structure. For example given the sequential data in Table 1 we can construct the WAP-tree structure in Figure 1.

Table 1: A database of web access sequences.

TID	Web access sequence	Frequent subsequence
100	*abdac*	*abac*
200	*eaebcac*	*abcac*
300	*babfaec*	*babac*
400	*afbacfc*	*abacc*

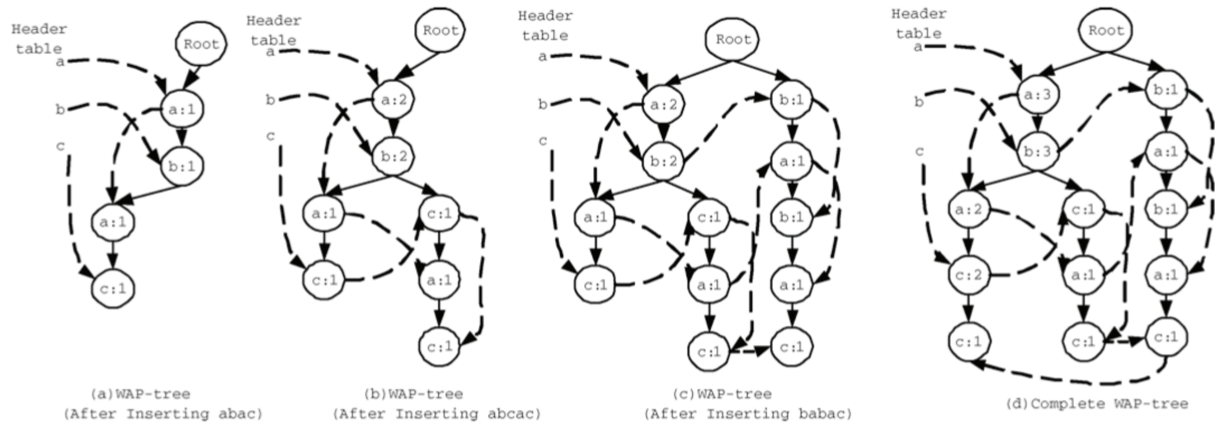
Header table
a
b
c
Root
a:1
b:1
a:1
c:1
(a)WAP-tree (After Inserting abac)
a:2
b:2
c:1
a:1
(b)WAP-tree (After Inserting abcac)
(c)WAP-tree (After Inserting babac)
a:3
b:3
c:2
(d)Complete WAP-tree

Figure 1: Construction of WAP Tree.

WAP-tree is constructed by scanning the frequent subsequences and adding a new node to the tree if a path with the same subsequence does not exist. Finding the frequent sequences can also be achieved by using conditional WAP-tree just like the FP-grow algorithm. Figure 2 shows example of conditional WAP-trees.

3.2 Pre-order Linked WAP-Tree

As shown in the previous section, the weakness of WAP-Tree algorithm is in the mining step where multiple conditional WAP trees need to be generated. This could be costly when dealing with large

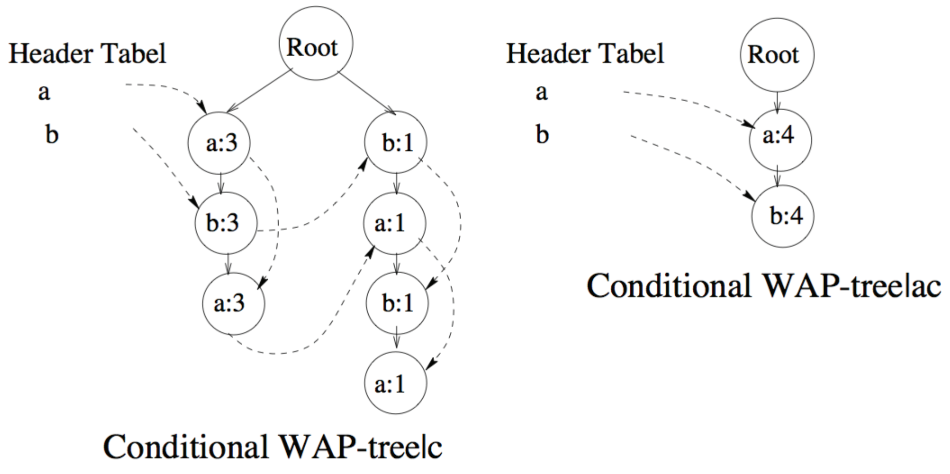

Figure 2: Conditional WAP Trees.

databases. As the result Ezeife and et al. [3] propose a new method similar to that of WAP that does not require multiple conditional WAP-tree constructions. They add a new variable to each node called the position. The position variable is a binary position code that will uniquely identifies each node. It can also be used to relate the position of a node relative to another node. The construction of this new WAP-tree is the same as the one shown in section 3.1 with the addition of position codes. Notice that the order of the links are also different in Pre-order Linked WAP-Tree (PLWAP). Figure 3 shows the construction of PLWAP tree using the dataset given in Table 1.

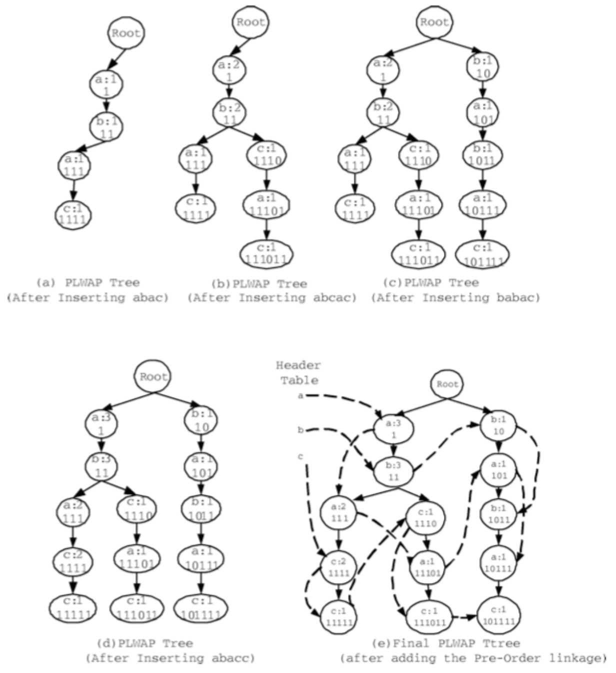
(a) PLWAP Tree (After Inserting abac)
(b) PLWAP Tree (After Inserting abcac)
(c) PLWAP Tree (After Inserting babac)
(d) PLWAP Tree (After Inserting abacc)
Header Table
(e) Final PLWAP Ttree (after adding the Pre-Order linkage)

Figure 3: Construction of PLWAP Tree.

The mining step for PLWAP tree is also different. Instead of starting from least frequent header and using conditional suffix, PLWAP uses the most frequent and conditional prefix to mine the frequent sequences. Figure 4 shows how the PLWAP tree can be used to find the frequent sequence starting with *aa*.

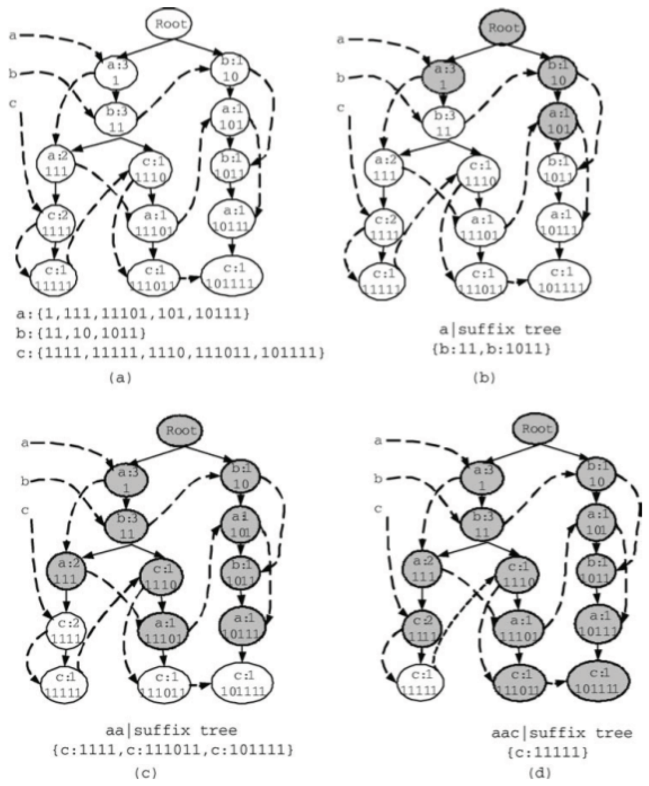

Figure 4: Finding frequent sequence starting with *aa*.

3.3 HPG

None of the previously mentioned algorithms is designed specifically for finding user navigation patterns. In fact, they can be thought of as general sequential mining algorithms. As the result, Borges and et al. [4] proposed a method for finding user navigation patterns based on the hypertext probabilistic grammar (HPG). In this scheme user navigation sessions are modelled as a hypertext probabilistic language generated by a hypertext probabilistic grammar. HPG is a probabilistic regular grammar which has a one-to-one mapping between the set of non-terminal symbols, representing different pages in the website and a set of terminal symbols. Two other states, S and F, indicate the beginning and the end of a user navigation session. In HPG a production rule corresponds to a link between pages.

Formally, HPG can be defined as a four-tuple $< V,\ \Sigma,\ S,\ P >$ where $V = \{S, A_1, \cdots, A_k, F\}$ is the set of non-terminal symbols, $\Sigma = \{a_1, \cdots, a_k\}$ is the set of terminal symbols, S is a unique start symbol and P is the set of production rules. Here, the pages in the website are represented by A_1 to A_k as non-terminal symbols and again by a_1 to a_k as terminal symbols.

The production rules can then be broken down in to three parts. Productions with S on the left-hand side are calculated as

$$p(S \rightarrow a_i A_i) = \alpha \frac{\text{number of times page } A_i \text{ is visited}}{\text{total number of pages visited}} + (1-\alpha) \frac{\text{number of times } A_i \text{ is the first page visited}}{\text{total number of user sessions}} \tag{3.1}$$

where α is used to give proper weight to a page being the first page in the user navigation pattern. The second set of productions are calculated as

$$p(A_i \rightarrow a_j A_j) = \frac{\text{number of times page } A_i \text{ is visited followed by page } A_j}{\text{total number of times page } A_i \text{ is visited}}. \tag{3.2}$$

The third and the final set of productions with state F on the right hand side is calculated as

$$p(A_i \rightarrow F) = \frac{\text{number of times } A_i \text{ is the last visited page}}{\text{total number of times page } A_i \text{ is visited}}. \tag{3.3}$$

Table 2 shows an example of set of user navigation sessions. In this example, there are 6 user sessions with a total of 24 page requests. As an example we show the calculation of productions involving page A_4. The page A_4 is visited 4 times. It is once the start state and once the end state. The productions involving A_4, using $\alpha = 0.5$, are calculated as follows

$$\begin{aligned}
p(S \rightarrow a_4 A_4) &= 0.5\frac{4}{24} + (1-0.5)\frac{1}{6} = 0.17 \\
p(A_4 \rightarrow a_1 A_1) &= \frac{2}{4} = 0.5 \\
p(A_4 \rightarrow a_6 A_6) &= \frac{1}{4} = 0.25 \\
p(A_4 \rightarrow F) &= \frac{1}{4} = 0.25
\end{aligned}$$

Table 2: An example of user navigation sessions.

Session ID	User trail
1	$A_1 \rightarrow A_2 \rightarrow A_3 \rightarrow A_4$
2	$A_1 \rightarrow A_5 \rightarrow A_3 \rightarrow A_4 \rightarrow A_1$
3	$A_5 \rightarrow A_2 \rightarrow A_4 \rightarrow A_6$
4	$A_5 \rightarrow A_2 \rightarrow A_3$
5	$A_5 \rightarrow A_2 \rightarrow A_3 \rightarrow A_6$
6	$A_4 \rightarrow A_1 \rightarrow A_5 \rightarrow A_3$

Similarly the production for all the other pages can be calculated. HPG can be represented using a finite state automaton (FSA) where the states are the pages in the website and the edges represent the production probabilities. Figure 5 shows the resulting FSA for the user sessions given in Table 2.

In a HPG the probability of the first derivation step of a string is evaluate against the support threshold, θ, and is not factored into the derivation probability. Thus, the support threshold is used to prune out the strings which may otherwise have high probability but correspond to a subset of the hypertext system rarely visited. Moreover, a string is included in the grammar's language if its derivation probability is above the cut-point, λ, where the cut-point corresponds to the grammar confidence threshold. The values of the support and confidence thresholds give the user control over the quantity and quality of the trails to be included in the rule set. The strings generated by the

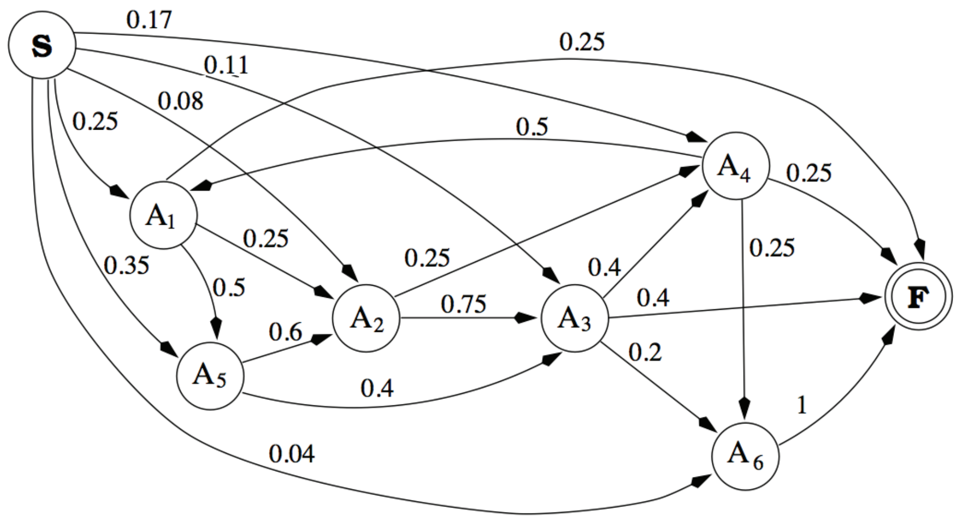

Figure 5: The hypertext grammer for N=1 and $\alpha = 0.5$.

grammar correspond to user navigation trails, and the aim is to identify the subset of these strings that best characterise the user behaviour when visiting the site.

The algorithm used to mine rules having confidence and support above the specified thresholds is a special case of a directed graph Depth-First Search which performs an exhaustive search of all the strings with the required characteristics. Figure 6 shows the rules obtained using this algorithm for various values of θ and *lambda* based on the graph given in Figure 5.

rule-set 1		rule-set 2		rule-set 3		rule-set 4	
$\alpha = 0.5$						$\alpha = 1.0$	
$\theta = 0.1$				$\theta = 0.15$		$\theta = 0.1$	
$\lambda = 0.2$		$\lambda = 0.3$		$\lambda = 0.3$		$\lambda = 0.3$	
rule	conf.	rule	conf.	rule	conf.	rule	conf.
A_1A_2	0.25	$A_1A_5A_2$	0.3	$A_1A_5A_2$	0.3	$A_1A_5A_2$	0.3
$A_1A_5A_3$	0.2	A_3A_4	0.4	A_4A_1	0.5	$A_2A_3A_4$	0.3
$A_1A_5A_2A_3$	0.23	A_4A_1	0.5	A_5A_3	0.4	A_3A_4	0.4
$A_3A_4A_1$	0.2	A_5A_3	0.4	$A_5A_2A_3$	0.45	A_4A_1	0.5
A_3A_6	0.2	$A_5A_2A_3$	0.45			A_5A_3	0.4
$A_4A_1A_5$	0.25					$A_5A_2A_3$	0.45
A_4A_6	0.25						
A_5A_3	0.4						
$A_5A_2A_3$	0.45						

Figure 6: The rules obtained with various model configurations.

HPG algorithm can be extended to an N-gram model. In this case the N-gram concept is used to determine the assumed user memory when navigating within the site where N is the history depth ($N \geq 1$). For a given N, it is assumed only the last N pages visited influence user's choice of the next page. For the purpose of this project we will use a simplified version of the algorithm where $N = 1$. However this is an important feature of the algorithm since we would like to study user behaviour and needs to be implemented in future expansions of this work.

4 Implementation

In this section we will discuss the algorithm implementations. We spent a long time looking for existing sequential mining software, however the few implementations that we found did not do a good job. For example WUM was a full data mining software but it only represented the resulting sequences as graphs, so we only used it to convert the log to sequential data. For GSP, WAP Tree and Pre-order Linked WAP Tree we found existing C++ implementations on the web [7]. However the implementations only worked on sequences of numbers, so the code was slightly modified to use URLs instead. We implemented a simple version of the HPG algorithm in Java. We discuss HPG in greater details next.

Since we needed to create a directed graph representing the HPG we used the JGraphT [8], a free Java graph library. Using this graph library for each page a graph node is generated. A start node (S) and an end (F) are also generated. The values on the edges (probabilities) are calculated using Equations (3.1), (3.2), and (3.3), respectively. The code written for this part of the project is presented in Appendix C at the end of the report.

5 Results

We ran all four algorithms on the final formatted sequential data based on the sequences WUM created from our simplified log. While we did not formally study execution time, we noticed GSP ran much slower than the other three algorithms almost by 10 times. We ran the algorithms using five different support thresholds (2% to 10%). For the HPG algorithm, we tried two values for each of α and λ. For α we used 1 to distribute the weights regardless of whether the state was the first state and 0.5 to distribute the weight proportionally between all states and states that were the first state. For the confidence (λ) of HPG we used 1% and 2% values. Figure 7 shows the relationship between the number of sequences mined and the support threshold for each algorithm. Note that GSP, WAP Tree and Pre-order Linked WAP Tree all returned the same number of sequences and thus the graph overlaps for them.

By increasing the support threshold, the number of sequences mined decreases more rapidly for sequential algorithms than HPG. It is interesting to note that for a given support (θ) and confidence (λ) threshold, decreasing α (giving more weight to pages that were the first state), increases the number of sequences mined up to a certain threshold and then decreases them (comparing to $\alpha = 1$ where all states are weighted equally).

To study the effect of the confidence threshold on the number of sequences mined using the HPG algorithm, we ran the algorithm with fixed a support threshold (θ) of 1% and variable confidence thresholds (λ) from 2% to 20%. The results presented in Figure 8 show an exponential decrease in the number of rules as confidence threshold increases.

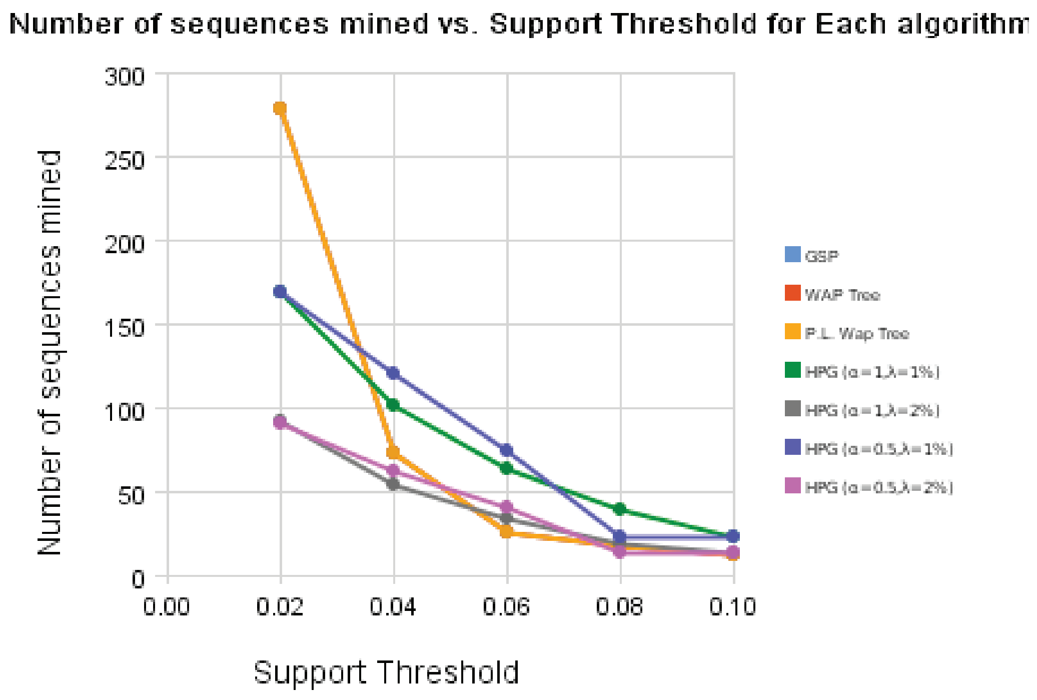

Figure 7: the relationship between the number of sequences mined and the support thresholds.

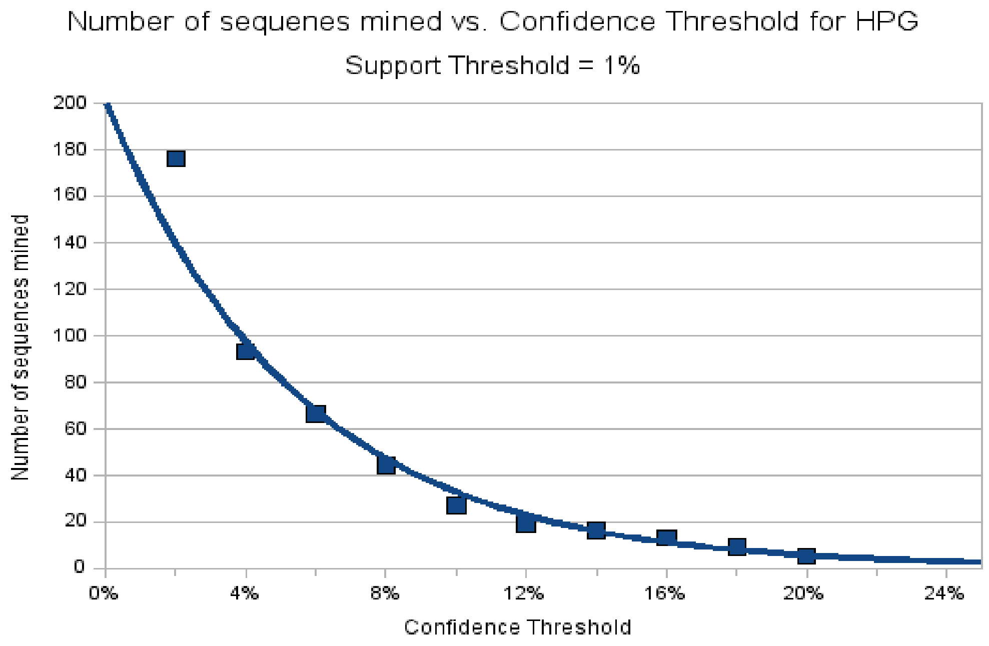

Figure 8: Number of sequences mined vs confidence thresholds.

5.1 Quality of results

We found that with support thresholds of higher than 4% not many useful sequences were returned using the sequential mining algorithms. This is because the most frequent sequences (top 30) returned were generally the most visited pages of the site or generally sequences with only one or at most two items. Although this information is useful, we could already get this information with simple web log analyzers. Thus, longer sequences are of more interest to us. On the other hand while for small support thresholds HPG returned less results than the other algorithms, it returned more interesting results even with higher support thresholds. For example with $\theta = 10\%$, $\lambda = 2\%$ and $\alpha = 0.5$, 13 sequences were returned that included some interesting sequences up to 3 items long.

5.2 Usefulness

To check the /quality and the usefulness of the results we consulted the owner of the website. Based on his comments the sequences we found (especially those found using HPG) did in fact gave very useful information regarding the usability of the site. These are a few examples of the frequent sequences found, their description and their application to web site usability.

- `"/home.php" "/search.php"`

  `"/search.php&search_by=title&query=&category_id=&submit"`

  The first page is the home page (first page after users log in), followed by the search page and a ”search by title” request where a category is selected to filter the search results. This tells us the search feature is used frequently and that most are done by the title. More importantly it tells us when searching, a large number of users limit their search to a specific category of music, *thus it is important to inform users to select a correct category when adding sheets*.

- `"/login.php" "/error.php&error_code=15"`

  An error page (error code 15 means the account is not active) is shown after the user tries to login. This means users try to login without activating their account using the activation email. *We need a better way of informing the users that they need to activate their account before they can log in.*

- `"/register.php" "/activation.php&username=&activation_id=" "/login.php"`

  `"/home.php"`

  This sequence shows registration, followed by activation, followed by a successful login. This is one of the most frequent sequences (top 20). This is good news. It means users receive the activation email sent by us, activate their accounts and login. Most importantly it means the activation email is not lost in users' SPAM folder and they do not forget to come back and complete the registration

- `"/activation.php&username=&activation_id=" "/login.php"`

  `"/error.php&error_code=15"`

  This sequence indicates that the users visited the activation page to log in, however the error page told them that their account was not active yet. This could be an error in the site. Although some users have already visited the activation page, their account is not activated. This could potentially mean that some bots or hackers are sending invalid code to the activation page to hack it and auto-create user accounts to SPAM the site or add links to improve their own website's rank.

- `"/error.php&error_code=13" "/register.php"`

  This sequence indicates error code 13 (invalid username or password) followed by the registration page. This means some users try to re-register when their password does not work. We have a module that resets the passwords but according to this some people do not use it. Interesting enough, the link to this module on the site is right next to the link to register (although the link to register a new account is in bold).

- `"/sheets.php" "/user.php&user_id=&show_requests=1"`

  `"/user.php&user_id=&sheet_id=&request_id="`

  This sequence indicates that the page with the list of sheets is followed by a page showing the music sheets that are requested by a user. The last page opens up a message box to contact a user with the given user_id to arrange an exchange of the sheet with the given sheet_id and request_id. This is also good news. It means when user A tried to ask for a sheet that user B owned, instead of just asking for the sheet with nothing in return (aka leeching), he or she looks at the list of sheets that user B is looking for and offers them to user B. It is also important that users noticed and used the ”requests” feature since this feature was also recently integrated into the site.

- `/tell_friends.php&sheet_id=`

  This page allows users to email or recommend a sheet to others. This is one of the most frequent sequences (top 20), which means the recommendation feature is used frequently. This is free advertisement for the site, so it's good news for the site's owner.

6 Conclusion

We extracted raw server logs from a music sheet exchange website. Due to the structure of the site a great deal of pre-processing work was applied to the log file which cut the size of the file in almost half and simplified the URLs to a format best describing general user interaction with the site. The resulting log file was converted to sequential user navigation sessions using the IP addresses of the users, a time threshold of 30 minutes per session and the referring URL information from the log file.

We implemented and applied four data mining algorithms. The general GSP algorithm for sequential pattern mining and three algorithms designed specifically for web logs: WAP Tree, Pre-order Linked WAP Tree and Hypertext Probabilistic Grammar. We found while HPG returned a smaller number of sequences for smaller thresholds than the other algorithms (which all returned the same sequences), the sequences mined using HPG were of more interest for our specific purpose of mining user navigation patterns.

As some examples suggested, the resulting user navigation sessions can be studied and compared with the designers' expected user navigation pattern and help improve the structure of the site. This can tell the site owner what the users prefer to do most and where the users are confused the most.

In general, these methods can help find errors and usability issues using information that is freely available for virtually any web site. We did not implement the N-gram (history depth for $N > 1$) feature of the HPG algorithm. Using such advanced feature of HPG we can take advantage of even more accurate user navigation pattern mining. This feature will be part of the future work in this area.

Appendix A: Sample Input/Output

In this section we show the sample input and output of every step from the preprocessing to the final pattern found.

Original web server log (before preprocessing)

```
85.185.81.5 - - [01/Oct/2007:06:12:16 -0400] "GET /index.php?m=sheets&lang=fa HTTP/1.1" 200
20985 www.sheetexchange.com
"http://www.google.com/ie?q=%D9%85%D9%88%D8%B3%DB%8C%D9%82%DB%8C+%D8%AC%D8%AF%DB%8C%D8%AF&hl=
fo&btnG=Leita" "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)" "-"
85.185.81.5 - - [01/Oct/2007:06:12:24 -0400] "GET /default.css HTTP/1.1" 200 3367
www.sheetexchange.com "http://www.sheetexchange.com/index.php?m=sheets&lang=fa" "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1)" "-"
85.185.81.5 - - [01/Oct/2007:06:12:27 -0400] "GET /includes/form_verification.js HTTP/1.1" 200
886 www.sheetexchange.com "http://www.sheetexchange.com/index.php?m=sheets&lang=fa"
"Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)" "-"
85.185.81.5 - - [01/Oct/2007:06:12:29 -0400] "GET /images/title_fa.jpg HTTP/1.1" 200 23614
www.sheetexchange.com "http://www.sheetexchange.com/index.php?m=sheets&lang=fa" "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1)" "-"
```

Purified and simplified web server log

```
71.118.245.109 - - [01/Oct/2007:11:08:19 -0400] "GET /sheets.php HTTP/1.1" 200 18163
www.sheetexchange.com "http://www.google.com/search?q=javad maroufi sheet
music&hl=en&start=10&sa=N" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.7)
Gecko/20070914 Firefox/2.0.0.7" "-"
71.118.245.109 - - [01/Oct/2007:11:09:15 -0400] "GET /search.php HTTP/1.1" 200 5848
www.sheetexchange.com "http://www.sheetexchange.com/sheets".php "Mozilla/5.0 (Windows; U;
Windows NT 5.1; en-US; rv:1.8.1.7) Gecko/20070914 Firefox/2.0.0.7" "-"
71.118.245.109 - - [01/Oct/2007:11:09:17 -0400] "GET /faq.php HTTP/1.1" 200 6408
www.sheetexchange.com "http://www.sheetexchange.com/search".php "Mozilla/5.0 (Windows; U;
Windows NT 5.1; en-US; rv:1.8.1.7) Gecko/20070914 Firefox/2.0.0.7" "-"
```

Simple sequential input: paths extracted with web log analyzers

(Here the first number is the number of repetitions)

```
34,/register.php,/sheets.php
31,/home.php,/register.php,/home.php,/search.php,/home.php,/search.php,/sheets.php,/home.php,/
add.php,/register.php,/home.php,/search.php,/home.php,/search.php
30,/register.php,/login.php,/sheets.php
```

Actual sequential input extracted using data mining software (Web Utilization Miner)

(One page per line. First number is sequence number (transaction ID) connecting the pages. The other numbers were unused in this experiment)

```
100003,1001391,1,5000716,1,"-"
100003,1001391,2,3000720,1,"/sheets.php"
100004,1001391,1,5000717,1,"/sheets.php"
```

```
100004,1001391,2,3000747,1,"/sheets.php&sheet_id="
100004,1001391,3,3000747,2,"/sheets.php&sheet_id="
100004,1001391,4,3000721,1,"/user.php&user_id="
100004,1001391,5,3000748,1,"/user.php&user_id=&sheet_id=&show_requests=1"
100004,1001391,6,3000744,1,"/user.php&user_id=&sheet_id=&request_id="
100004,1001391,7,3000721,2,"/user.php&user_id="
100005,1001391,1,5000718,1,"/user.php&user_id="
100005,1001391,2,3000747,1,"/sheets.php&sheet_id="
100005,1001391,3,3000747,2,"/sheets.php&sheet_id="
100005,1001391,4,3000747,3,"/sheets.php&sheet_id="
100005,1001391,5,3000721,1,"/user.php&user_id="
100005,1001391,6,3000721,2,"/user.php&user_id="
100005,1001391,7,3000721,3,"/user.php&user_id="
100005,1001391,8,3000721,4,"/user.php&user_id="
```

Final processed sequential input in TAB-separated format

(First number is the sequence number (transaction ID), the second number is the number of items in the sequence)

```
0       6       "/sheets.php&sheet_id=" "/sheets.php&sheet_id=" "/user.php&user_id="
"/user.php&user_id=&sheet_id=&show_requests=1"  "/user.php&user_id=&sheet_id=&request_id="
 "/user.php&user_id="
0       7       "/sheets.php&sheet_id=" "/sheets.php&sheet_id=" "/sheets.php&sheet_id="
"/user.php&user_id="    "/user.php&user_id="    "/user.php&user_id="    "/user.php&user_id="
0       7       "/contact.php"  "/sheets.php&sheet_id=" "/user.php&user_id="
"/sheets.php&sheet_id=" "/user.php&user_id="    "/user.php&user_id="
"/sheets.php&sheet_id="
0       6       "/user.php&user_id="    "/sheets.php&sheet_id=" "/user.php&user_id="
"/user.php&user_id="    "/user.php&user_id="    "/sheets.php&sheet_id="
```

Sample frequent sequences returned

```
"/user.php&user_id=&sheet_id="  "/user.php&user_id=&sheet_id=&request_id="
"/user.php&user_id=&sheet_id="  "/user.php&username="
"/user.php&user_id=&sheet_id=&request_id="      "/sheets.php&sheet_id="
"/user.php&user_id=&sheet_id=&request_id="      "/user.php&user_id="
"/user.php&user_id=&sheet_id=&request_id="      "/user.php&user_id=&sheet_id="
"/user.php&user_id=&sheet_id=&request_id="      "/user.php&user_id=&sheet_id=&request_id="
"/user.php&user_id=&sheet_id=&request_id="      "/user.php&user_id=&sheet_id=&show_requests=1"
"/user.php&user_id=&sheet_id=&show_requests=1"  "/user.php&user_id=&sheet_id=&request_id="
"/user.php&username="   "/user.php&user_id=&sheet_id="
"/activation.php&username=&activation_id="      "/home.php"     "/search.php"
"/activation.php&username=&activation_id="      "/home.php"     "/sheets.php"
"/activation.php&username=&activation_id="      "/home.php"     "/user.php&user_id=&sheet_id="
```

Appendix B: Algorithm Implementation and Usage

Sequential Algorithms Implementations

C++ implementations of PLWAP, WAP tree and GSP algorithms used. The original code from `http://cs.uwindsor.ca/~cezeife/codes.html` has been modified to take in html pages (strings) instead of numeric sequences.

Sequential Algorithms Usage

The code reads the data from a file named ”test.data”. See below. Once started the program asks for a frequency (minimum support) between 0 and 1. The results are saved in ”result_[GSP/WAP/PLWAP].data” (TAB-separated). The execution time is printed for each algorithm.

Sequential Algorithms Input requirements

The required format of TAB-separated test.data is: id seq_count seq_item1 seq_item2 ... Where id is a unique sequence id seq_count is the number of elements in the sequence seq_item[n] can be any string (here html page names and URLs)

Sequential Algorithms Limitations

No known limitations have been reported by the author.

HPG Algorithm Implementations

We implemented the HPG algorithm in Java. Source code is presented in Appendix C.

HPG Algorithm Usage

The code reads the data from a file named ”test.data”. And outputs the results to result.txt The values of alpha, theta and lambda must be changed in the code. To compile the code the type package is needed. Need to copy the file type.jar to jre
lib
ext folder.

HPG Algorithm Input requirements

Same as above.

HPG Algorithm Limitations

No knows limitation at this time.

Appendix C: HPG code

Here we present the code we wrote for HPG. Since the code for GSP, WAP, PLWAP were obtained from the Internet, we do not present them here and the user is refereed to the [7].

```java
/**
 * sequenceMine.java
 *
 *
 * @author Nariman Farsad
 * @version 1.00 2010/4/15
 */
package org.jgrapht.graph;
//package org.jgrapht.traverse.DepthFirstIterator;

import java.util.*;
import org.jgrapht.*;
import org.jgrapht.graph.*;
import org.jgrapht.traverse.*;
import type.lib.*;
import java.io.*;
class Counter
{
    private int counter;
    public Counter()
    {
        counter=0;
    }
    public int getCounter () {return counter;}
    public void setCounter (int c) {counter=c;}
    public void increment () {counter++;}
    public void decrement () {counter--;}
}

class PageCount
{
    private String page;
    private int count;

    public PageCount ()
    {
        page = "";
        count = 0;
    }
    public PageCount (String p, int c)
    {
        page = p;
        count = c;
    }
    public String getPage() {return page;}
    public int getCount () {return count;}
    public void setPage (String p) {page = p;}
    public void setCount (int c) {count = c;}
    public void incrementCount () {count++;}
}
class EdgePairs
{
    private String page1;
    private String page2;
    private int count;
    public EdgePairs ()
    {
        page1 = "";
        page2= "";
        count = 0;
    }
    public EdgePairs (String p1, String p2, int c)
    {
        page1 = p1;
        page2 = p2;
        count = c;
    }
    public String getPage1()    {return page1;}
    public String getPage2()    {return page2;}
    public int getCount () {return count;}
    public void setPage1 (String p1) {page1 = p1;}
    public void setPage2 (String p2) {page1 = p2;}
    public void setCount (int c) {count = c;}
    public void incrementCount () {count++;}
    public String toString ()   {return page1 + "->"+ page2+"\t"+count;}

}


public class sequenceMine
{


    public static void main (String [] arg) throws IOException
    {
        // HPG Graph
        DirectedWeightedMultigraph<String, DefaultWeightedEdge> weightedGraph =
                            new DirectedWeightedMultigraph<String, DefaultWeightedEdge>(DefaultWeightedEdge.class);
        Scanner FileIn = new Scanner(new File ("test.data"));
        PrintStream FileOut = new PrintStream("result.txt");
        ArrayList<PageCount> pageLists = new ArrayList<PageCount>();
        ArrayList<EdgePairs> edgePairList = new ArrayList<EdgePairs>();
        ArrayList<String> candVertix = new ArrayList<String>();
        int totalPages = 0;
        int totalSessions = 0;
        final String startPage = "~~####";
        final String endPage = "####~~";
        String prevPage;
        double alpha = 1;
        double theta = 0.10;
        double lambda = 0.02;
        // Add the start and end vertix to the graph
        if (!weightedGraph.addVertex(startPage))
                        System.out.println("Failed␣to␣add␣vertix:␣"+startPage);
        if (!weightedGraph.addVertex(endPage))
                        System.out.println("Failed␣to␣add␣vertix:␣"+endPage);

        // read the file line by line
        while (FileIn.hasNextLine())
        {
            StringTokenizer tokens = new StringTokenizer(FileIn.nextLine(), "\t");
            prevPage = startPage; // set previous page to start page
            totalSessions++; // add the session
            String session = tokens.nextToken(); // skip session
            String number = tokens.nextToken(); // skip number
            while (tokens.hasMoreTokens())  // go through the tokens (pages)
            {
                String page = tokens.nextToken();
                totalPages ++;
                boolean flagExists = false;
                // add the new page to approprieta edge pair
                for (int i = 0; i < edgePairList.size();i++)
                {
                    EdgePairs tempEdge = edgePairList.get(i);
                    if (tempEdge.getPage1().equals(prevPage)&&tempEdge.getPage2().equals(page))
                    {
                        tempEdge.incrementCount(); // increment the edge pai count if the edge already exists
                        flagExists=true;
                        break;
                    }
                }
                // create the edge pair if it is new
                if (!flagExists)
                {
                    edgePairList.add (new EdgePairs (prevPage, page, 1));
                }

                prevPage = page;
                flagExists = false;
                // add the pages to page counter
                for (int i = 0; i < pageLists.size();i++)
                {
                    PageCount temp = pageLists.get(i);
                    if (temp.getPage().equals(page))
                    {
                        temp.incrementCount(); // increment the count if page already added
                        flagExists=true;
                        break;
                    }
                }
                // create the new page if did nor exist and add it to graph as vertix
                if (!flagExists)
                {
                    pageLists.add (new PageCount (page,1));
                    if (!weightedGraph.addVertex(page))
                        System.out.println("Failed␣to␣add␣vertix:␣"+page);
                }
            }
            // add the end edge to the edge pai list
            boolean flagExists = false;
            String page = endPage; // set next page to end page
            // check if the end edge pai exists
            for (int i = 0; i < edgePairList.size();i++)
            {
                EdgePairs tempEdge = edgePairList.get(i);
                if (tempEdge.getPage1().equals(prevPage)&&tempEdge.getPage2().equals(page))
                {
                    tempEdge.incrementCount(); //increment the cost is the edge exists
                    flagExists=true;
                    break;
                }
            }
            // create the edge pair if it does not exists
            if (!flagExists)
            {
                edgePairList.add (new EdgePairs (prevPage, page, 1));
            }

        }
        // add the start edges that did not exist
        for (int i = 0; i < pageLists.size();i++)
        {
            boolean exists = false;
            String p = pageLists.get(i).getPage();
            // check is start edge exists
            for (int j = 0; j <  edgePairList.size(); j++ )
            {
                EdgePairs ep = edgePairList.get(j);
                if (ep.getPage1().equals(startPage) && ep.getPage2().equals(p) )
                    exists = true;
            }
            if (!exists) // add the edge if it did not exits
                edgePairList.add (new EdgePairs (startPage, p, 0));
        }
        // print pages
        /*for (int i = 0; i < pageLists.size();i++)
        {
            System.out.println (pageLists.get(i).getPage() +"\t"+pageLists.get(i).getCount());
        }
        System.out.println (totalPages); // print total pages
        // print edge pairs
        for (int i = 0; i < edgePairList.size();i++)
        {
            System.out.println (edgePairList.get(i));
        }*/

        // add the edges with their weights calculated to graph
        for (int i = 0; i < edgePairList.size();i++)
        {
            EdgePairs tempEdge = edgePairList.get(i); // get the current edge pair
            //add the edge to graph
            DefaultWeightedEdge edge = weightedGraph.addEdge(tempEdge.getPage1(), tempEdge.getPage2());

            // calculate the edge weight if it is a start edge
            if (tempEdge.getPage1().equals(startPage))
            {
                double wieght = 0.0;
                double pageCount = 0.0;
                for (int j = 0; j<pageLists.size(); j++)
                {
                    PageCount pc = pageLists.get(j);
                    if (pc.getPage().equals(tempEdge.getPage2()))
                    {
                        pageCount = pc.getCount();
                        break;
                    }
                }
                wieght = alpha * pageCount / totalPages + (1-alpha) * tempEdge.getCount()/ totalSessions;
                weightedGraph.setEdgeWeight(edge, wieght);
            }
            else // calculate the wieght if it is not a start edge
            {
                for (int j = 0; j<pageLists.size(); j++)
                {
                    PageCount pc = pageLists.get(j);
                    if (pc.getPage().equals(tempEdge.getPage1()))
                    {
                        weightedGraph.setEdgeWeight(edge, (double)(tempEdge.getCount())/pc.getCount() );
                        break;
                    }
                }
            }
        }
        // check which first derivation pages pass the support criteria
        Set <DefaultWeightedEdge> edgeS = weightedGraph.edgeSet();
        Iterator it = edgeS.iterator();
        while (it.hasNext())
        {
            DefaultWeightedEdge edge = (DefaultWeightedEdge)it.next();
            double weight = weightedGraph.getEdgeWeight(edge);
            //System.out.println (edge+"\t"+weight);
            if (weightedGraph.getEdgeSource(edge).equals(startPage) && weight > theta)
            {
                candVertix.add(weightedGraph.getEdgeTarget(edge)); // come up with candidate vertix
                //System.out.println ("this edge passed the test" + weightedGraph.getEdgeTarget(edge));
            }
        }
        //System.out.println (weightedGraph);
        for ( int i = 0; i < candVertix.size(); i++)
        {
            ArrayList <String> ruleList = new ArrayList <String>();
            String oldVertix = candVertix.get(i);
            ruleList.add(oldVertix);
            sequenceMine(oldVertix, weightedGraph, ruleList, 1.0, lambda,FileOut);
        }
        System.out.println ("DONE!");
        FileOut.close();

    }

    static public void sequenceMine(String vertix, DirectedWeightedMultigraph<String, DefaultWeightedEdge> dg,
                        ArrayList <String> ruleList, double prevWeight, double support, PrintStream FileOut)
    {
        //System.out.print (vertix +"\t");
        final String endPage = "####~~";
        Set <DefaultWeightedEdge> edgeS = dg.edgesOf(vertix);
        //System.out.println (edgeS);
        DirectedWeightedMultigraph<String, DefaultWeightedEdge> ndg =
                (DirectedWeightedMultigraph<String, DefaultWeightedEdge>)dg.clone();
        ndg.removeVertex(vertix);
        Iterator it = edgeS.iterator();
        while (it.hasNext())
        {
            DefaultWeightedEdge currentEdge = (DefaultWeightedEdge)it.next();
            String target = dg.getEdgeTarget(currentEdge);
            if (! target.equals(vertix) && ! target.equals(endPage))
            {
                //System.out.println ("the edge here is"+currentEdge);
                double curWeight = dg.getEdgeWeight (currentEdge);
                double newWeight = prevWeight*curWeight;
                if ( newWeight >= support)
                {
                    ruleList.add(target);
                    sequenceMine(target, ndg, ruleList, newWeight, support, FileOut);

                }
            }
        }
        if (ruleList.size()>1)
        {

            for (int i = 0; i<ruleList.size();i++ )
            {
                FileOut.print (ruleList.get(i));
                if (i!=ruleList.size()-1)
                    FileOut.print ("\t");
                else
                    FileOut.println ("");
            }
            FileOut.flush();
            ruleList.remove (ruleList.size()-1);
        }
    }
}
```

References

[1] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," *Advances in Database TechnologyEDBT'96*, pp. 1–17, 1996.

[2] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu, "Mining access patterns efficiently from web logs," *Knowledge Discovery and Data Mining. Current Issues and New Applications*, pp. 396–407, 2000.

[3] C. Ezeife and Y. Lu, "Mining web log sequential patterns with position coded pre-order linked wap-tree," *Data Mining and Knowledge Discovery*, vol. 10, no. 1, pp. 5–38, 2005.

[4] J. Borges and M. Levene, "Data mining of user navigation patterns," *Web usage analysis and user profiling*, pp. 92–112, 1999.

[5] "The Web Utilization Miner WUM." `http://hypknowsys.sourceforge.net/wiki/`, April 2010.

[6] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data mining and knowledge discovery*, vol. 8, no. 1, pp. 53–87, 2004.

[7] "Downloadable Research Source Codes." `http://cs.uwindsor.ca/~cezeife/codes.html`, April 2010.

[8] "JGraphT." `http://jgrapht.sourceforge.net/`, April 2010.