Efficient Web Log Mining for Product Development

Yew-Kwong Woon  Wee-Keong Ng
Nanyang Technological University
Nanyang Avenue
Singapore 639798
{ps7420440f, awkng}@ntu.edu.sg

Xiang Li  Wen-Feng Lu
Singapore Institute of Manufacturing Technology
71 Nanyang Drive
Singapore 638075
{xli, wflu}@SIMTech.a-star.edu.sg

Abstract

With the new global economy, manufacturing companies are focusing their efforts on the product development process which is fast emerging as a new competitive weapon. Several product development solutions allow engineers, suppliers, business partners and even customers to collaborate throughout the entire product lifecycle via the Internet. To gain an additional edge over competitors, it is vital that companies utilize web logs to discover hidden knowledge about trends and patterns in such a cyberworld. However, existing web log mining techniques are not designed for web logs generated by product data management processes. In this paper, we propose a method termed Product Development Miner (PDMiner) to mine such web logs efficiently and effectively using a trie structure and sequential mining techniques. Experiments involving real web logs show that PDMiner is both fast and practical.

1. Introduction

According to Parametric Technology Corporation, global manufacturing investment in product development solutions is expected to reach more than $25 billion dollars [8]. This is largely due to a revolutionary change of mindset of manufacturing companies. Instead of just focusing on developing great products, companies are looking into the prospect of converting product development processes into a competitive weapon. This strategy is made possible by the wide range of product development solutions that are currently available.

Product development solutions allow engineers, suppliers, business partners and even customers to collaborate throughout the entire product lifecycle via the Internet through thin web-based clients. Hence, there exists abundant raw data in the form of web logs. For brevity, we shall henceforth refer to web logs as logfiles. Much valuable knowledge about the trends and patterns of the product development processes can be found by applying data mining techniques on logfiles. Such knowledge is hidden and previously-unknown and thus may well become the keen competitive edge that manufacturers are urgently seeking. This knowledge can be put to great use by streamlining the product development processes or simply improve the speed and convenience of web accesses.

Unfortunately, as logfiles are originally meant for debugging purposes, they are in a form that is unsuitable for data mining [10]. However, due to their wide adoption by existing web servers, we contend that they will not be replaced as the de facto web data source in the near future. A standard logfile has the following fields:

- **remotehost**: remote hostname or IP address
- **logname**: remote logname of the user
- **username**: username with which the user has authenticated himself
- **date**: date and time of the request
- **request**: exact request line as it came from the client
- **status**: HTTP status code returned to the client
- **bytes**: content-length of the document transferred

The following is a small fragment of a sample logfile:

```
ntu.edu.sg - - [19/May/2003:00:01:15 -0430]
"GET /html/faq.html HTTP/1.0" 200 4865
155.69.181.254 - - [1/Jun/2003:00:04:22 -1420]
"GET /public/tile/home.html HTTP/1.0" 200 195
```

As seen in the above example, the fields logname and username are usually not recorded. Hence, it is difficult to identify the activities of individual users. This issue is not important if our main concern is in the sequences of web pages accessed and it will be discussed in subsequent sections. There are several existing work on logfile mining and but they deal separately on specific issues of mining unrelated to product development with various assumptions [4, 5, 13, 14, 18]. Some take the web site structure into
consideration while some focus only on traversal patterns and others consider the amount of time spent on a page.

In this paper, we formulate a novel logfile mining technique termed **Product Development Miner** (PDMiner) to efficiently mine logfiles in the cyberworld of product development. We use a trie structure defined as the **WebTrie** to efficiently hold preprocessed data that is important for fast mining of sequences of web pages. The WebTrie is developed in our previous work and is both incremental and practical [18]. PDMiner takes into consideration, the unique characteristics of product development processes and extracts knowledge pertaining to how products can be designed faster by discovering the relationships among parts and assemblies. In addition, PDMiner discovers how people involved in the product development process interact with one another and this helps to improve workflow processes. Together with the WebTrie, PDMiner has been shown to be practical by experiments conducted on real logfiles.

The rest of the paper is organized as follows. The next section looks at related work done in the areas of logfile mining and product development. Section 3 formally describes the WebTrie and PDMiner. Experiments are conducted on real logfiles to assess our new technique in Section 4. Finally, the paper is concluded in Section 5.

2. Related Work

In this section, we survey and describe works on logfile mining and briefly outline the product development process.

2.1. Logfile Mining

Logfile mining is the discovery of user access patterns from web servers. The following are the kinds of mining that can be performed [17]:

- **Statistical Analysis**: Used to obtain useful statistical information such as the most frequently accessed pages and average page view duration
- **Association Rule Mining** [1]: Used to discover correlations among page references for structuring web sites or page prefetching
- **Sequential Pattern Mining** [2]: Used to discover inter-session patterns for predicting visit patterns
- **Clustering**: Used to group together users with similar characteristics for market segmentation or web content personalization
- **Classification**: Used to group together users into predefined classes based on their characteristics for customer profiling

The above tasks help in the design of better web sites and in the development of marketing strategies. However, it is difficult to obtain user information and the logfiles do not capture such data. Hence, it is more practical to assume the anonymity of users for logfile mining and disregard clustering and classification. In product development, we need only focus on association rule mining and sequential mining. The following are some preprocessing tasks [6]:

1. **Data Cleaning**: The server log is first examined to remove irrelevant items.
2. **User Identification**: The presence of proxy servers complicates user identification because several users may share a single machine name and thus, heuristics are used to identify users [15].
3. **Session Identification**: After a user is identified, his page accesses must be divided into individual sessions.
4. **Transaction Identification**: Once a session of completed paths is determined, page references must be grouped into logical units representing web transactions before mining can be performed. This grouping process is known as transaction identification or transactionization.

Chen et al. pioneered work in logfile mining [4]. The procedure for mining sequential access of web pages consists of three steps:

1. **Determine transactions**: It is assumed that backward references (page revisitations) are used only for ease of travelling. The moment a backward reference occurs, a forward reference path is considered terminated and is termed a **maximal forward reference** (MFR) which represents a transaction.
2. **Determine large reference sequences**: A large reference sequence is a MFR that appears frequent enough to satisfy a minimum support threshold. From the set of MFRs, they are determined using algorithms based on the ideas of association rule mining [1].
3. **Determine maximal reference sequences**: A maximal reference sequence is a large reference sequence that is not contained in any other large reference sequence.

This method may discover too many sequences and give preference to initial pages. For example, the resultant set of MFRs for the sequence \( \{ABC\} \) is \( \{ABC, ABDE\} \). Although the subsequence \( \{AB\} \) appears only once in the original sequence, it appears twice in the set of MFRs.

To group web page references into more meaningful transactions, Cooley et al. proposed a way of identifying transactions based on the assumption that a user uses a page for either navigation or content viewing. Hence, a transaction can be defined in the following two ways:

1. **Navigation-Content**: A navigation-content transaction contains all navigation references leading to content references and the content references themselves for
product development process as represented by the bidirectional arrows in Figure 1. Some commercial product development systems include Windchill [7] and eMatrix [12].

In global manufacturing, designers and engineering no longer contribute most in product design; all collaborators play substantial roles and in some cases, suppliers contribute up to 70% of the final product design [8].

As interactions and activities of collaborators are carried out via web-based clients, large amounts of data are captured in the form of log file entries. Hence, there is valuable knowledge waiting to be unearthed in such log files. The following are some previously unknown knowledge that can be extracted and used as competitive weapons:

1. **Part-Assembly relationships**: Although such relationships can be obtained by querying the part-assembly database, log file mining can help sift out significant relationships with greater speed and lesser supervision. Such knowledge is critical for fast product design.
2. **Frequently-accessed web pages**: Patterns of sequential accesses of web pages allow webmasters to improve structure of web client for faster access.
3. **Collaboration relationships**: Collaborators access the product development system at various timepoints of the product development process. By understanding how collaboration is usually carried out, workflow processes can be fine-tuned and optimized. For example, if it is discovered that customers usually step in to alter design even after suppliers have confirmed shipment of parts, then the workflow should include time buffers to cushion sudden changes of customer requirements. Due to the complexity and security of login mechanisms of product development software, such relationships are still difficult to mine and will be handled only in our future work. Fortunately, some knowledge can be discovered about such relationships by looking at the frequent access patterns of web pages as discussed in Section 3.2.2.

Figure 2 shows the traditional phases of the product development process and here is a brief description of each
phase in proper sequence [16]:

1. Requirements Definition: Involves identifying the needs of the customer and defining the business and design objectives for the product
2. Conceptual Design: Involves identifying possible design approaches based on requirements
3. Detailed Design: Involves evaluating design approaches and finalizing the design of the product
4. Test and Evaluation: Involves correcting problems, identifying areas for improvement and reducing risks
5. Manufacturing: Involves producing the product
6. Logistics: Involves planning and controlling the flow and storage of products and their components

The bidirectional arrows in Figure 2 indicate that the above phases constantly interact with one another at various stages of the product development process. For example, even when a product is already being shipped to customers (last phase), transportation problems such as the unforeseen fragility of the product may force engineers to go back to the drawing board (second phase). Therefore, it is highly desirable to use logfile mining to discover trends in the complex mechanics of the product development process for streamlining purposes.

3. Data Structure and Algorithm

In this section, we describe and formally define our trie structure and the algorithm that uses it to discover knowledge for product development from logfiles.

3.1. WebTrie

The WebTrie is a trie structure conceived in our previous work for the objective of storing page sequences of lengths 1 and 2 and we shall use it in conjunction with PDMiner [18]. For completeness, we will briefly describe it here. Here are some formulation preliminaries:

- Let the universal navigation set $U = \{p_1, p_2, \ldots, p_n\}$ be a set of navigation pages $p_i$ where $1 \leq i \leq n$ found in a web site.
- Let $D \subseteq 2^U$ be the set of sessions found in a logfile.
- Let $S_m \in D$ be a session identified from the logfile with $m$ navigation pages. Sessions are found using a 30-minute session duration heuristic which has been proven to be reliable [3].
- A page sequence $X_k \subseteq U$ is an ordered sequence of $k$ navigation pages.

The following is a formal definition of the WebTrie.

**Definition 1 (WebTrie)** A WebTrie consists of a tree node $w$ which has a label representing a navigation page and an integer representing its support. We use $w_{p_i}$ to refer to a tree node that corresponds to a page $p_i \in U$ and $\beta_{w_{p_i}}$ to represent its support count. Let $C(w_{p_i})$ be the support-ordered set of child nodes of node $w_{p_i}$. If $C(w_{p_i}) \neq \emptyset$, then $C(w_{p_i}) \subseteq \{w_m, \ldots, w_n\}$ where $1 \leq m \leq n$ and $\beta_{w_m} \geq \beta_{u_{m+1}} \land m \neq i$.

Note that an implicit virtual root node $R$ is needed to link all the WebTries together. The WebTries under $R$ are sorted in support-descending order where the leftmost WebTrie has the highest support count. For brevity, we shall refer to the set of WebTries as WebTrie henceforth. The WebTrie $\mathcal{W}$ is constructed in the following steps from a set of sessions $D$:

1: for each session $S_m \in D$ do
2: for each page $p_i \in S_m$ where $i = 1, ..., m$ do
3: if $\exists j$ s.t. $p_i = p_j$ where $j < i$ then
4: if $w_{p_i}$ exists then
5: increment $\beta_{w_{p_i}}$ by 1
6: else
7: create new node $w_{p_i}$ under root node $R$ with $\beta_{w_{p_i}} = 1$
8: end if
9: end if
10: if $\exists j$ s.t. $\langle p_i, p_{i+1} \rangle = \langle p_j, p_{j+1} \rangle$ where $j < i$ then
11: if $i \neq m$ then
12: if $w_{p_{i+1}} \in C(w_{p_i})$ then
13: increment $\beta_{w_{p_{i+1}}}$ by 1
14: else
15: create child node $w_{p_{i+1}}$ under parent node $w_{p_i}$ with $\beta_{w_{p_{i+1}}} = 1$
16: end if
17: end if
18: end if
19: end for
20: end for

For every page $p$ in a session, we first check if there is a first-level tree node representing $p$ (step 4). If there is, we simply increment its support count by 1. Otherwise, we will create a node under $R$ to represent $p$. If $p$ is not the last page, we check the page that comes immediately after it to see if a child node is needed to be created or if we can simply increment its support count (step 11). With this construction algorithm, information about every page sequence of lengths 1 and 2 is stored in $\mathcal{W}$. Note that steps 3 and 10 are necessary to ensure that each session increments the support count of any page sequence by 1 only to avoid giving preference to page sequences that appear very often in a particular session.
Table 1. A sample logfile.

<table>
<thead>
<tr>
<th>Session ID</th>
<th>Page Sequences</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>ABCBDE</td>
</tr>
<tr>
<td>200</td>
<td>BDBCE</td>
</tr>
<tr>
<td>300</td>
<td>CDE</td>
</tr>
<tr>
<td>400</td>
<td>ABD</td>
</tr>
</tbody>
</table>

Though not explicitly mentioned above, all tree nodes and their child nodes are sorted in a support-descending order and thus, every change to a WebTrie may require a shifting of nodes to maintain the order. Note that \( W \) is created/updated on-the-fly each time a session arrives. Hence, the WebTrie is support-independent and thus, do not require any user input to begin construction. In addition, it is also incremental because it need not be rebuilt from scratch whenever new sessions arrive or when different support thresholds are used. This makes the WebTrie very suitable for mining large logfiles with fast arrival of sessions.

Figure 3 shows the WebTrie constructed from the page sequences of the sample logfile shown in Table 1. The alphabet beside the square node represents the label of the node and the bracketed number beside it is its support count. Ignore the arrows and the numbers beside them for the moment. Notice that the constructed WebTrie is ordered differently from an alphabet trie with page \( D \) at the leftmost node. To find the support count of any page sequence of lengths 1 and 2, the WebTrie need only be traversed; for example, the support count of \( \langle B, C \rangle \) is 2 which is correct because \( \langle B, C \rangle \) appears in sessions 100 and 200. The support-ordered nature of the WebTrie allows for fast pruning during mining.

3.2. PDMiner

PDMiner is an efficient logfile mining algorithm specifically designed for the domain of product development. It uses the WebTrie to discover frequent page sequences and association rule mining techniques to discover correlations among parts, assemblies and documents. It consists of two main steps discussed as follows.

3.2.1. Preprocessing

The preprocessing phase consists of the following steps:

1. Unimportant entries for product development such as HTTP requests for image, audio and video files in the logfile are removed.
2. All requests that do not return the HTTP status code of 200 (means successful request) are removed.
3. Scripts must be examined for relatedness. Most product development software use scripts to communicate with its remote servers and unimportant/unrelated scripts should be removed.
4. All remaining entries are converted into a simpler form for mining in the following way: Each unique entry is converted into a unique integer. In our sample logfile shown in Table 1, we use alphabets instead for clarity.
5. For each entry with the same IP address, group them together into a session. A 30-minute time window is used here which means that a session consists of entries assessed within 30 minutes of the first entry.
6. Create another dataset \( D_u \) consisting of unique entries in sessions representing parts and assemblies. Duplicated pages in a session are also removed. Note that scripts must be examined prior to this step to identify such entries which may include the creation, updating, deletion and viewing of parts and assemblies. As before, these entries are mapped to unique integers. This is done to allow the application of association rule mining methods to find correlations among entries containing parts and assemblies.

An important point to note here is that transactionization is not carried out. This is because we would want to accept sessions in their original form instead of modifying them using assumptions that are not applicable to product development. For example, backward references should not be disregarded as they reveal access patterns in product development. In addition, the effect of noise in a product development web client is non-existent because such clients are usually more well-designed than web sites. Finally, navigation and content pages are difficult to differentiate even with their access times because access times may be affected by several other factors like web traffic and speed of the system. Hence, PDMiner does not make such assumptions and proceed to mine the session entries in their entirety.

3.2.2. Mining

There are two types of knowledge that can be extracted by PDMiner:

1. Association Rules: Association rule mining techniques can be applied on the dataset \( D_u \) with transactions
containing no duplicated items to discover correlations among parts and assemblies. Any association rule mining algorithm can be used here and in our previous work, we have conducted experiments using latest association rule mining methods to affirm the feasibility of this type of mining on logfiles [18]. Hence, due to the lack of space, we would not be discussing such mining here but would focus on the mining of sequential patterns which is less straightforward. An example of an association rule in the context of product development is \( P_1 \land P_2 \land A_3 \rightarrow P_3 \) where \( P_x \) are parts and \( A_3 \) is an assembly; means that there is a frequent trend of \( P_3 \) being accessed whenever \( P_1, P_2 \) and \( A_3 \) are accessed.

2. Sequential Patterns: Frequent sequential patterns of page accesses can be discovered using sequential mining techniques [2]. Such patterns not only help webmasters to improve the design of product development web clients, they also reveal pertinent knowledge about the relationship of various types of information that can be accessed through the web clients. For example, much is known via the frequent sequential pattern \((P_1, D_1, D_2, M_1)\) where \(P_1\) is a part, \(D_x\) are documents and \(M_1\) is a modification command executed for \(P_1\). The webmaster can use this pattern to prefetch documents \(D_1\) and \(D_2\) when \(P_1\) is accessed. If it is known that the documents are generated only after a certain process, say, after the final design of a product is confirmed, and that they can only be accessed by certain collaborators such as the customer, we may be able to deduce that customers usually change parts even after the product design is finalized by engineers. Of course, the knowledge genre depends on the organization’s context as well as the customized product development software and thus, it is still necessary that the patterns be further analyzed by the software administrator and project leader before they can be translated into valuable knowledge.

Before we move on, let us look at a case study involving Hewlett-Packard (HP) and Windchill to see how the above knowledge can be utilized. In this study, HP’s Imaging and Printing Group (IPG) used Windchill [7] as the product lifecycle collaboration framework to coordinate back-end design centers with front-end regions’ product requirements in Europe, Asia and America [11]. More than 1,500 diverse users (includes business partners, suppliers, designers, third parties) were able to collaborate globally resulting in a reduction in change order cycle time, heightened levels of design and process leverage, reductions in stopped shipments and higher-volume production.

With so many users accessing the cyberworld of Windchill, much valuable knowledge is embedded in the logfiles. PDMiner can mine the logfiles for association rules to reveal implicit relationships among parts and assemblies to allow designers to further speed up the design process. More importantly, PDMiner can reveal sequential patterns that are useful for improving the access time and user-friendliness of Windchill web clients for all its 1,500 users through dynamic customization according to their frequent access patterns. In addition, the pertinent relationships among these users can be discovered through sequential mining as well. Currently, IPG is able to set up a new partner in 24 hours using Windchill. We believe that with knowledge unveiled by PDMiner, this process can be shortened significantly by utilizing past history/experience hidden in the logfiles. The efficiency of PDMiner would certainly play an important role here because of the immense size of the logfiles.

We shall now focus on how sequential patterns can be discovered efficiently and here are some formulation preliminaries:

- A page sequence \( X_k = \langle x_1, x_2, \ldots, x_k \rangle \) is said to be a consecutive subsequence \( \triangleright \) of session \( S_m = \langle s_1, s_2, \ldots, s_m \rangle \), iff \( \exists i \) s.t. \( s_{i+j} = x_j \) for \( 1 \leq j \leq k \).
- A session \( S_m \) is said to support page sequence \( X_k \) iff \( X_k \triangleright S_m \).
- The support count \( \beta_{X_k} \) of page sequence \( X_k \) is the number of sessions that support it.
- A page sequence \( X_k \) is termed frequent if \( \beta_{X_k} \geq |D| \times s\% \) where \( s \) is a user-defined support threshold.

We are interested only in page sequences that occur frequently because they represent interesting and important trends. As seen in Section 2.1, several techniques exist for mining page sequences but they are mostly carried out after transactionization which means that the sequences are made shorter. However, in PDMiner, the sessions are not transactionized and thus, more efficient techniques are needed to mine longer sequences. We have previously introduced a method called the Web Traversal Blueprint (WTB) algorithm to mine sequences together with the WebTrie but it is not efficient at lower support thresholds where extremely long patterns are discovered. This is due to the large number of candidate (potentially-frequent) patterns generated and the need to scan the logfile to obtain the actual support counts of each of them. To eliminate this candidate generation problem, we introduce an improved version of WTB:

1. Let \( \ell \) be size of the largest pattern in \( D \)
2. Initialize frequent sequence set \( F_j = \emptyset \) for \( 1 \leq j \leq \ell \)
3. where \( j \) is the length of the sequences in the set
4. for each tree node \( p_i \in C(R) \) do
5. if \( \beta_{w_{p_i}} \geq |D| \times s\% \) then
6. \( F_1 = F_1 \cup p_i \)
6: Invoke \texttt{FindTrail}(\{w_{p_i}\}, w_{p_i}, 2)
7: else
8: Exit foreach loop
9: end if
10: end for
11: Result = \bigcup_{j=1}^{N} F_j

\textbf{FindTrail}(sequence X, tree node parent, int k)

1: for each tree node \(w_{p_i} \in C(\text{parent})\) do
2: if \(\beta_{w_{p_i}} \geq |D| \times s\)% then
3: Append \(\{p_i\}\) to \(X\) and assign new \(X\) to sequence \(Y\)
4: if \((k == 2) \lor (\beta_Y \geq |D| \times s\)% then
5: \(F_k = F_k \cup Y\)
6: Invoke \texttt{FindTrail}(Y, w_{p_i}, k + 1)
7: end if
8: else
9: Exit foreach loop
10: end if
11: end for

The main idea is to reduce the number of database scans by earlier database scans to eliminate the need for candidate pattern generation. Unlike WTB, PDMiner only invokes the new procedure \texttt{FindTrail} recursively on frequent sequences (steps 7–9 of \texttt{FindTrail}). This is intuitive because if a sequence has been found to be infrequent, it will still be infrequent even if pages are appended to it. Therefore, by scanning the logfile earlier (step 4 of \texttt{FindTrail}), we can avoid unnecessary calls to \texttt{FindTrail} as well as candidate pattern generation.

As the WebTrie is support-ordered, the search for frequent sequences can be stopped the moment a node containing an infrequent sequence is found at the same tree level (step 3 of PDMiner and step 2 of \texttt{FindTrail}). One obvious important consideration here is the proper setting of the support threshold. Currently, in all known logfile mining literature, there is no a priori technique for determining the optimal support threshold. Thus, the user has to experiment with various support thresholds in order to find the optimal threshold. As such, the WebTrie is useful here because it is support-independent and can thus be utilized (without being rebuilt) for various thresholds. The following example shows how PDMiner works together with the WebTrie.

\textbf{Example} Suppose the given support threshold is 75% which means that a sequence must have a minimum support count of 3 for the sample logfile shown in Table 1. The frequent sequences found using PDMiner are \(\{D, B, BD, C, E\}\). The paths of traversal are shown by the arrows in Figure 3. The numbers beside the arrows denote the sequence of traversal (left-first, depth-first). Notice that path \(DB\) is never taken; this is because the sequence \(DE\) is not a frequent sequence and thus, there is no need to traverse its sibling node. The same goes for paths \(CD, CE\). The paths \(RD\) and \(DE\) are traversed twice (steps 5 and 6) because the sequence \(\{BD\}\) is frequent and the algorithm tries to form longer sequences. However, the candidate sequence \(\{BDE\}\) only has a support count of 1 (appears in session 100 only) and is thus not frequent.

\section{4. Performance Evaluation}

This section evaluates the performance and practicality of PDMiner by conducting experiments on a Pentium-IV machine with a CPU clock rate of 2.4 GHz, 1 GB of main memory and running on a Windows 2000 platform. The algorithms are implemented in Java and the real logfiles \(D_1\) and \(D_2\) can be freely downloaded [9]. \(D_1\) is about 168 MB, contains 8436 unique pages and is captured over a week by an Internet access provider for the Metro Baltimore while \(D_2\) is about 200 MB, contains 941 unique pages and is captured over a month by NASA Kennedy Space Center. Although they do not belong to the product development domain, they contain real data and would suffice to allow us to assess the practicality of PDMiner. Real product development logfiles are still unavailable at this time. It is not possible to compare the performance of PDMiner with that of other existing logfile mining algorithms, including our original WTB algorithm, because they all use different assumptions to transactionize the logfiles. As such, only PDMiner is used in the experiments here.

Figure 4 shows the execution times of PDMiner for the logfiles at varying minimum support thresholds. The paths \(RD\) and \(DE\) are traversed twice (steps 5 and 6) because the sequence \(\{BD\}\) is frequent and the algorithm tries to form longer sequences. However, the candidate sequence \(\{BDE\}\) only has a support count of 1 (appears in session 100 only) and is thus not frequent.
quences can be found and the WebTrie has to be traversed many more times recursively. This shortcoming is insignificant because at lower thresholds, the number and length of frequent sequences found are much larger and may be more difficult to analyze as seen in Figure 5. For $D_2$ at a support threshold of 0.1%, there are more than 400 frequent sequences of length 2 and several frequent sequences of lengths 8 and above whereas at 0.25%, there are much lesser frequent sequences and the longest sequence is of length 8 only. The same observation can be seen for $D_1$ except that it contains lesser and shorter frequent sequences and thus requires much shorter computation time as seen in Figure 4. This is because it has many more unique pages which means that generally, sequences have lower support counts than those of $D_2$; its sessions contain more diversified combinations of pages. All in all, we contend that users would probably use higher thresholds to obtain lesser frequent sequences for ease of analysis without significant loss of knowledge. In any case, spending a few hundred seconds is certainly worthwhile if PDMiner discovers valuable knowledge that would result in savings of days or even weeks.

5. Conclusions

The need for efficient product development becomes increasingly important as manufacturing becomes globalized. As product development solutions are mostly web-based, there exists an abundance of raw data in the form of web logs. We have proposed an efficient technique called PDMiner to mine web logs from the product development cyberworld and have verified its viability through experiments with real logfiles. In our future work, we will conduct more experiments involving web logs generated by product development operations to further assess the usefulness of PDMiner and to fine-tune it for actual operations.

References