ORIGINAL RESEARCH

Heavy path based super-sequence frequent pattern mining on web log dataset

Xinran Yu [*], Turgay Korkmaz

Department of Computer Science, The University of Texas at San Antonio, San Antonio, Texas, USA

Received: January 20, 2015 **Accepted:** March 2, 2015 **Online Published:** March 12, 2015
DOI: 10.5430/air.v4n2p1 **URL:** http://dx.doi.org/10.5430/air.v4n2p1

Abstract

Mining web log datasets has been extensively studied using Frequent Pattern Mining (FPM) and its various other forms. Identifying frequent patterns in different sequences can help in analyzing the most common sub-sequences (e.g., the pages visited together). However, this approach would not be able to identify general structures spanning over multiple sequences. In response to understanding general structures, we introduce a new form of sequential pattern mining called super-sequence frequent pattern mining (SS-FPM). In contrast to sub-sequences determined by FPM, SS-FPM determines the super-sequences that can contain the common parts from different sequences. This can be useful in understanding the general behavior/flow of users in web usage mining, classifying web pages and users, making predictions etc. In essence, finding frequent super-sequence patterns turns out to be related to the well-known heaviest (longest) path problem in graphs, which is known to be NP-hard. Accordingly, we transform a given sequential dataset into a sequence graph and formulate the problem as k-hop heaviest path problem. We then propose an efficient heuristic called sequence matrix method using dynamic programming techniques. We compared our method to the existing Heavypath method. The results show that our method is more efficient especially on large datasets.

Key Words: Super-sequence mining, Pattern mining, Web log dataset, Heavy paths

1 Introduction

Sequential pattern mining has been a popular research topic due to its importance in many applications (e.g., shopping basket analysis, bioinformatics, web usage mining and so on) where data is given in the form of a set of sequences or a long sequence.[1–4,39] One form of the sequential pattern mining is to find Frequent Patterns (FPM). It was first introduced by Agrawal et al[1,5] and was to analyze the item sets that appear frequently in the customers' shopping baskets. As we review in the next section, researchers have proposed many algorithms to this problem and investigated various other forms of FPM. In this paper, we focus on investigating a new form of sequential pattern mining that we call super-sequence frequent pattern mining (SS-FPM) and the application of it on Web Log datasets. Various types of sequential datasets can also be applied to accordingly.

A *sequential dataset* is a collection of sequence(s) of ordered elements or events.[6] As an example, let us consider web log sessions in Table 1. Each web session is an ordered sequence of web clicks, i.e., two consecutive pages AB means that a user first visited page A and then page B. In this example, traditional FPM algorithms (e.g., Apriori, downward closure) would identify AB and CD as the most frequent sub-sequence patterns as they appear three times in the dataset. Finding such frequent sub-sequences can be helpful for analysing the most common structures and improving the performance. For example, by analysing the traversal patterns in a web server's log, one can gather im-

[*]**Correspondence:** Xinran Yu; Email: yxr_8011@163.com; Address: Department of Computer Science, The University of Texas at San Antonio, San Antonio, Texas, USA

portant information such as the most popular pages which are likely to be visited together.[7]

Table 1: An example of web log sequences

Session Id	Session Sequence
1	ABEB
2	BED
3	AB
4	AB
5	BCD
6	CD
7	CD

In contrast to finding the most common sub-structures, we take a different approach and consider how to idendentify/analyse the underlying general structures among multiple sequences in the database. For example, suppose the example in Table 1 is from a news web-site. To analyse the general behaviour/flow of users (e.g., in what order the users generally read the news items, do they follow the suggested links or jump to different news etc.), we need to find a relatively longer visiting sequence of pages that have received the most clicks from different users. We call such sequences as super-sequences and the process of finding them as super-sequence frequent pattern mining (SS-FPM). The key distinguishing characteristic of super-sequences is that they may contain several parts from different sequences. For example, in the above sample dataset, the most frequent super-sequence pattern with sequence length 4 would be ABCD because the total number of occur-ring times of AB, BC, and CD is 7, which is higher than that of any other super-sequences with sequence length 4. This super-sequence shows the popular flow of the visited news which is from A to B then to C and then D. As we can see that only analysing the frequent sub-sequences cannot give a flow like super-sequence does. In addition to analysing general structures, web designers can use supper-sequence patterns to predict future users' behaviour and dynamically make recommendations in interactive learning and shopping sites. For example, if new user visits news A and B, according to the super-sequence found, C and D can be recommended.

In a closely related work,[8] the authors have analysed the research filed revolution by considering heavy paths which are similar to our super-sequences. The authors have considered a citation database where each citation relationship from paper A to paper B can be viewed as a length 2 sequence. By associating each paper with its topic, they generated a network where the nodes are research topics and the weight on an edge is the frequency of citations. Heavy-paths (i.e., super-sequences) in such a graph capture strong flows of ideas across topics. The authors in Ref.[8] also considered heavy paths for music recommendation system and itinerary recommendation system for visiting as many as popular places of interest in a limited travel time.

Extracting super-sequences in a sequential dataset would also be important in various other areas and applications such as system call analysis, bioinformatics, social networks, etc. For example, biologists can better understand the relationships between gene structures and functional elements by determining the frequent super-sequences in the DNA sequences[2] of closely related species. Also, in protein association networks, super-sequences can help seek the most related proteins.[40] Another example would be malware detection,[9] where researchers analyse the sequence of system calls.[3, 10] By identifying frequent super-sequence system call patterns from many program traces, researchers can determine unusual sequence of calls that might be invoked by malicious programs.

The SS-FPM problem is related to the heaviest (a.k.a. the longest) path problem in directed graphs, which is known to be NP-hard.[11] Accordingly, we first transform the given sequential dataset into a sequence graph, and then search for heaviest paths as patterns. More specifically, we are interested in finding all the k-hop paths ((k + 1)-length super-sequence frequent patterns) that have a larger weight than a given threshold. Note that k-hop heaviest path is the same as (k + 1)-length super-sequence frequent pattern (SS-FP) and we use these terms interchangeably in the rest of the paper.

In response to solving this new form of the heaviest path problem in a directed weighted graph, we propose a sequence matrix method that basically uses some heuristics and dynamic programming techniques to gradually compute k-hop heaviest paths from the initial one-step sequence matrix representing the underlying sequence graph and the (k-1)-hop heavies paths computed in previous iteration. As we discuss later in detail, the worst-case complexity of the proposed sequence matrix method is $O(kn^4)$, where k is the number of hops (length of the super-sequence) and n is the number of nodes in the sequence graph . However, through experiments in Section 5, we will show that the average running time of our solution grows more like a quadratic function of n rather than quartic function as in the worst-case. We also show that it is significantly better than the running time of the heavy path heuristic proposed in Ref.[8]

In addition to the above algorithmic contribution to solve SS-FPM problem, we apply our solution to analysing some actual sequential web log datasets and identify interesting super-sequence patterns in these datasets. In this direction, we specifically consider an actual web-log dataset called BMS-WebView-1,[12] which consists of multiple sequences.

The remaining part of the paper is organized as follows. In Section 2 we give related work in frequent pattern mining area. We formally define super-sequence frequent pattern mining (SS-FPM) problem and give the graph model in Section 3. In Section 4 we present our proposed heuristic method using dynamic programming techniques for solving the SS-FPM problem. We then evaluate the performance of our solution both compared to Heavypath method and through experiments on actual web log dataset consisting of multiple sequences in Section 5. Finally, we conclude this paper and point out some issues for future research in Section 6.

2 Related work

FPM is to seek frequently occurring patterns or relationships in a large database. It was first introduced by Agrawal et al[1] to analyse the item sets that appear frequently in the customers' shopping baskets. After the introduction of their solution called Apriori algorithm, researchers have proposed many new algorithms and improvements for the same FPM problem while investigating its various other forms as well. For example, Park et al. have tried to improve Apriori using hashing technique,[13] Savasere et al. used partitioning technique,[14] Toivonen et al. proposed sampling approach.[15] In addition, researchers considered other methods such as dynamic item set counting,[16] incremental mining,[17] parallel and distributed mining[13,17,18] and so on.[19] Researchers have also developed new algorithms for the FPM problem such as FP-growth,[20] which uses FP-tree to store the itemset association information, and Eclat,[21] which uses an Equivalence CLAss Transformation method. Moreover, researchers have been studying various other types of FPM problem including multilevel and multidimensional association rules mining,[22] closed frequent pattern mining,[23] colossal pattern mining,[24] sequential pattern mining,[5] graphs, trees and lattices mining.[19]

Our work is related to sequential mining, where the frequent item sets mining is extended with the consideration of ordered items. Specifically, sequential mining tries to find the set of frequent subsequences in a given sequential dataset. One of the first solutions to this problem was AprioriAll,[5] which extends the similar techniques in Apriori[1] to find frequent patterns in transactions of the customers. It generates candidate sequences by measuring the support of them in passing over the database while using the downward-closure property of sequential patterns.

A representative list of the well-known sequential mining algorithms include Apriori-based GSP (Generalized Sequential Patterns),[25] Pattern-growth based FreeSpan[26] and PrefixSpan,[27] Vertical format-based SPADE[28] and Constraint-based SPIRIT.[29] All of these solutions mainly focus on the sub-sequence mining, which is to find the common part of all the sequences in a dataset. In our problem, we are looking for a general structure (i.e., the super-sequence). The most frequent super-sequence is the one with the largest total support of each consecutive length two sequence which is a link in it. Although part of the most frequent super-sequence may be the most frequent sub-sequence, this is not necessary.

Finding supper-sequences is related to the heaviest (longest) path problem, which is known to be NP-hard.[11] To cope with the NP-hardness of the heaviest path problem, researchers have proposed various approximation algorithms or developed exact algorithms for specific classes of graphs. Karger et. al proposed an approximation method for the heaviest path in a graph using greedy algorithm.[30] Murat et. al proposed probabilistic heaviest path problem.[31] Others have considered solutions for specific type of graphs. For example, researchers proposed heaviest path problems on directed acyclic graph,[32] interval graphs,[33] co-comparability graphs,[34] ptolemaic graphs[35] and so on. Most of these studies try to find the heaviest path starting from a source without any constraints. In our case, we are interested in any k-hop heaviest path whose weight is greater than a threshold. A similar work has been done and shown in Ref.[8] Their strategy is to use rank join method and go through each edge with each possible expansion from (k-1)-hop to k-hop. Since their work can find the accurate top-N heaviest paths in a graph, it is very time consuming. We will show some comparisons between our method and theirs in the experiment section later.

3 Problem definition and graph model

In this paper, a sequential dataset is denoted by $S = \{S_1, S_2, \cdots, S_N\}$, where $S_i = \{w_1, w_2, \cdots, w_{li}\}$ is a sequence of ordered elements or events (e.g., sequences of web pages in Table 1). The union of all the events is $W = \{w_1, w_2, \cdots, w_n\}$, (i.e., $W = \bigcup_{i=1}^{N} S_i$). Let $p = \{w_1, w_2, \cdots, w_k\}$ be a k-length super-sequence pattern (or (k-1)-hop path). Let support ($w_x w_y$) be the total number of occurrences of the sequence $w_x w_y$ in S. Similarly, we can define the support for a pattern p as follows:

$support(p) = \sum_{i=1}^{k-1} support(w_i w_{i+1})$

We say p is k-length frequent if $support(p)$ is greater than a given threshold δ. We can now formally define SS-FPM problem as follows:

Definition 1. Given a sequential dataset S, a sequence length k, and a threshold δ, the Super-Sequence Frequent Pattern Mining (SS-FPM) problem is to find the set $P = \{p_1, p_2, \cdots, p_r\}$, where each pattern $p_j = \{w_1, w_2, \cdots, w_k\}$ satisfies the following condition:

$$support(p_j) = \sum support(w_i w_{i+1}) > \delta \qquad (1)$$

SS-FPM problem is actually related to the heaviest path problem in a graph. So we transform a given sequential dataset S into a sequence graph, which can be defined as follows.

Definition 2. A Sequence Graph is a directed weighted graph $G = (V, E)$, where $V = \{w_1, w_2, \cdots, w_n\}$ and $E = \{e_1, e_2, \cdots, e_m\}$. Each edge e_i connects w_a and w_b if there is a consecutive sequence $w_a w_b$ in any sequence in S. The weight of edge e_i is set to $support(w_a w_b)$, which is the total number of occurrences of consecutive sequence $w_a w_b$ in all sequences in S.

According to this definition, we can easily construct the sequence graph for a given sequential dataset. For example, the sequence graph for the sequential dataset in Table 1 would be as shown in Figure 1. We use the adjacency matrix, which we call the initial one-step sequence matrix, to represent the sequence graph. Since there are n elements in the set of all sequences, the sequence matrix would be $n \times n$. For example, the sequence matrix for the above graph would be as in Figure 2.

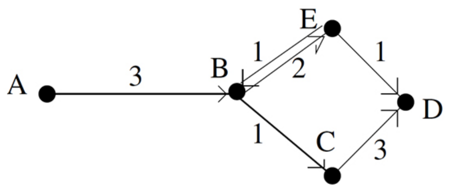

Figure 1: The sequence graph for the sequential dataset in Table 1

The sequence matrix denoted by M_1 can directly be computed from the given sequential dataset S. For this, we first set $M_1[w_i][w_j]$ to 0 for every $w_i w_j$. We then go through each sequence S_i and add one to $M_1[w_i][w_j]$ for each $w_x w_y$ in S_i.

The reason for using the notation M_1 is to indicate that $M_1[w_i][w_j]$ actually represents the support of 1-hop path (2-length sequence) from w_i to w_j In the rest of the paper, we will continue to generalize this notation as M_k to store the supports of the k-hop heaviest path ((k+1)-length most frequent super-sequence). Note that $M_k[w_i][w_j]$, not the path from w_i to w_j. The actual k-hop paths will be stored in a hash table, as we discuss later in detail.

$$M_1 = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{pmatrix} 0 & 3 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 2 \\ 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Figure 2: The sequence matrix for the sequence graph in Figure 1

We assume that the super-sequences are simple paths (e.g., sequential patterns without duplications). The dataset may have duplicates such as the first entry in Table 1. But when we search for a frequent super-sequence, we do not consider loops since the path may not be able to get out of the loop. So the frequent super-sequences got from the sequence graph are simple paths.

4 Proposed sequence matrix method for SSFPM

In this section we are going to give the details of our solution that computes M_k and determine the heaviest paths (i.e., the most frequent super-sequence patterns). To determine the (k+1)-length most frequent super-sequence patterns, we actually try to compute all the k -hop heaviest paths whose total weights are greater than a given threshold. For this, we mainly use dynamic programming techniques to gradually compute the matrix M_k from M_1 and M_{k-1} and we call this method the sequence matrix method. As we discussed in last section, $M_1[i][j]$ is the support of the 1-hop heaviest path (direct link) from node i to node j. On the other hand, for $k > 1$, $M_k[i][j]$ is the support of the k-hop heaviest path starting with sequence ij rather than being the weight of a path from i to j.

The formula is given below:

$$M_k[i][j] = M_1[i][j] + M_{k-1}[j][maxC] \tag{2}$$

where maxC is the column index of the maximum value in row j.

In addition, we maintain a hash table H_k that stores k-hop heaviest paths ((k + 1)-length frequent super-sequences) starting with ij. So the keys in H_k are $[ij]$ for all the non-zero elements in M_k while the values are the k-hop heaviest paths starting with the corresponding ij. Here there can be multiple paths starting with the sequence ij that have the same weights. In our heuristic algorithm, we choose one arbitrarily. Actually, this may cause missing the actual heaviest path. However, our experiments show that the difference is not significant, and thus we keep our current algorithm to choose one path. Nevertheless, at the cost of increasing computational and space complexity, one may simply extend our solutions to store multiple paths with the same weight and with the same starting sequence in the hash table. The formula that computes the hash table H_k is:

$$H_k[XY] = concat(H_1[XY], H_{k-1}[Y \cdots Z]) \tag{3}$$

where $H_{k-1}[Y \cdots Z]$ has the maximum weight in H_{k-1} starting with Y.

We will now illustrate the details of the proposed solution through an example. We then give its pseudo code and analyse its computational complexity. Specifically, we will consider the sequence graph in Figure 1 and illustrate the process of getting k-hop heaviest paths ((k + 1)-length frequent super-sequence patterns) when k is 3. So, we need to gradually calculate M_3 and H_3. First, let us generate H_1. This can easily be done by using the non-zero elements from the original one-step sequence matrix M_1. Resulting keys and 1-hop paths in H_1 will be as shown below:

H_1 =

Key	Value
AB	<AB>
BC	<BC>
BE	<BE>
CD	<CD>
EB	<EB>
ED	<ED>

We then compute M_2 by using M_1 twice as follows. For each i and j, we first search the largest value in row j of M_1 and identify its column index as $maxC$. We then simply compute $M_2[i][j]$ by taking the summation of $M_1[i][j]$ and $M_1[j][maxC]$. If $M_1[i][j]$ is 0, we keep it 0 in $M_2[i][j]$, which means we do not have a path starting from ij. Also, if the maximum in row j of M_1 is 0, we again set $M_2[i][j]$ to 0, which means that there is no outlet from node j.

Now we are going to explain why we need to check if there is a 0. When $M_1[i][j]$ is zero, there is no visits on page j after page i. Thus, there is no need to calculate the super-sequence path starting with ij in the sequence. Similarly, when the largest value on row j is zero, there is either no outlet in the (k-1)-sequence matrix for page j or they could generate a circle if we keep adding pages in the path. Thus, we set the corresponding $M_k[i][j]$ to zero. In summary, if either one of these conditions is true, then we cannot find a k-hop path starting with ij; thus, we set $M_k[i][j]$ to 0. Accordingly, the resulting M_2 will be as shown in Figure 3.

$$M_2 = \begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 5 & 0 & 0 & 0 \\ B & 0 & 0 & 4 & 0 & 3 \\ C & 0 & 0 & 0 & 0 & 0 \\ D & 0 & 0 & 0 & 0 & 0 \\ E & 0 & 2 & 0 & 0 & 0 \end{array}$$

Figure 3: The 2-sequence matrix for the sequence graph in Figure 1

While we generate M_2, we can derive the corresponding hash table H_2 and make sure that the paths in hash table will not have loops. After computing $M_2[i][j]$, we check if it is 0 or not. If it is not 0, then we copy the value of the key $[jmaxC]$ in H_1 to $[ij]$'s value in H_1 after the first element. For example, the new value in H_2 for key [AB] will be <ABE>. This is obtained by copying <BE> to <AB> after A since $M_1[B][E]$ has the largest value on row B. Accordingly, the resulting H_2 will be as follows:

H_2=

Key	Value
AB	<ABE>
BC	<BCD>
BE	<BED>
EB	<EBC>

From H_2 we can see that 3-length super-sequence patterns are ABE, BCD, BED and EBC with the supports of 5, 4, 3 and 2, respectively. Note that we did not choose EBE as a 2-hop path since it has a loop. We keep track of the paths and check if there is a loop when a path is extended with a new node. If so, we ignore that node and consider the next node.

Finally, M_3 is going to be computed by using M_1 and M_2 as follows. For each i and j, we first search the largest value in row j of M_2 and identify its column index as maxC. We then simply compute $M_3[i][j]$ by taking the summation of $M_1[i][j]$ and $M_2[j][maxC]$. Accordingly, the resulting M_3 will be as shown in Figure 4.

$$M_3 = \begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 7 & 0 & 0 & 0 \\ B & 0 & 0 & 0 & 0 & 0 \\ C & 0 & 0 & 0 & 0 & 0 \\ D & 0 & 0 & 0 & 0 & 0 \\ E & 0 & 5 & 0 & 0 & 0 \end{array}$$

Figure 4: The 3-sequence matrix for the sequence graph in Figure 1

For H_3, since $M_2[B][C]$ has the largest value on row B, we copy the value of [BC]: <BCD> to the value of [AB]: <ABE>, starting from the second element. Consequently, we obtain <ABCD> and save it in H_3 with the key value of [AB]. Using the same process, we also obtain <EBCD>. As a result, H_3 will be as follows:

H_3=

Key	Value
AB	<ABCD>
EB	<EBCD>

In summary, we computed ABE as the 2-hop heaviest path (3-length most frequent sequence) by combining 1-hop heaviest paths AB and BE from M1 in Figure 1. Since AB has the support of 3 and BE has the support of 2, the support of ABE will be 5. Continuing this way, we determined ABCD as the 3-hop heaviest path (4-length most frequent super-sequence) with the support of 7. We can now present the above steps in a pseudo code form, as shown in Algorithm 1.

Algorithm 1 Sequence Matrix Algorithm Computing k-hop Longest Path

```
Input: $M_1$, $M_{k-1}$, and $H_{k-1}$
Output: $M_k$ and $H_k$
for $1 \le i \le n$ do
  for $1 \le j \le n$ do
    if $M_1[i][j] \neq 0$ then
      Find maxC such that
      $M_{k-1}[j][maxC] \ge M_{k-1}[j][l]$ for $1 \le l \le n$
      while there is a loop in the new path do
      $M_{k-1}[j][maxC] = 0$,
      Find maxC such that
      $M_{k-1}[j][maxC] \ge M_{k-1}[j][l]$ for $1 \le l \le n$
      end while
      if $M_{k-1}[j][maxC]! = 0$ then
        $M_k[i][j] = M_1[i][j] + M_{k-1}[j][maxC]$,
        update hash table $H_k$
      else
        $M_k[i][j] = 0$
      end if
    end if
  end for
end for
```

4.1 Accuracy analysis of sequence matrix algorithm

When there are multiple same weights links existing for length k+1 super-sequences, our solution will miss some super-sequences, e.g., if there are 2 most frequent super-sequences starting from ij, we will only get one of them. This can also mislead the most frequent length k+1 super-sequence. There is also another assumption in this algorithm which is that the heaviest hop-k paths are distributed evenly for all the paths starting with any two nodes' permutation. This is because we can find at most n^2 top-N paths no matter what number is k. So any two nodes' permutation can only contribute to one possible heaviest path. Now we will show that for searching top-N heaviest paths, two of these paths come from one single two-node as starting edge is in small probability.

In a directed weighted graph with n nodes and m edges, each node has m/n edges in average. Suppose we are searching for top-N hop-k paths. The number of possible hop-k paths is $P_k^m = m!/(m-k)!$. The number of possible paths starting from link ab is $(m/n)^{k-1}$. Suppose there are two hop-k paths that are both started with ab and they in the top-N hop-k path-set, the probability is:

$$Prob = (m/n)^{k-1}/(m!/(m-k)!) \qquad (4)$$

If the graph is dense, this number can be very small, e.g., searching a hop-5 heavy path in a 10 node graph with 50 edges, the probability is

$$Prob = (5^4)/(50\times49\times48\times47\times46\times45) \approx 1/(25000000) \qquad (5)$$

which is very small. This shows that using our sequence matrix method has large chance to get most of the top-N heaviest paths.

In conclusion, as we mentioned at the beginning of this section, our solution can find the most frequent super-sequence with length k+1 starting from ab with small probability of errors when the weight on each link is different from each other.

4.2 Time complexity analysis of sequence matrix algorithm

In Algorithm 1, if there are n nodes in the graph, it computes the $n \times n$ matrix M_k by performing the above mentioned operations on two $n \times n$ matrices, namely M_1 and M_{k-1}. The actual paths are stored in a hash table H_k by extending the paths in H_k-1 while making sure that there will be no loops in them. To compute each element $M_k[i][j]$, the heuristic algorithm obtains the maximum in row j of M_{k-1}, which costs $O(n)$, and checks if the new path starting with ij would have a loop, which costs $O(k)$. If there is a loop, the heuristic algorithm ignores that path and finds another maximum until it finds a path without a loop. In the worst case, the heuristic algorithm may end up checking all the elements in row j, which costs $O(n)$. So, for each element ij, the heuristic algorithm would perform $O(n(n+k))$ operations, resulting in overall complexity of $O(n^4+kn^3)$. Since we are interested in simple paths, k will be less than n and thus the worst-case complexity of the heuristic algorithm for a given k will be $O(n^4)$. Since we gradually compute M_k by calling the heuristic algorithm k times, the overall worst-case complexity of our solution will be $O(kn^4)$. However, in practice, since the sequence graphs are sparse and the matrix M_k storing the weights of the heaviest k-hop paths gets more and more zero elements as the k increases, we would not need to find maximum or search for loops in paths for every element in the matrix. This results in reasonable average case complexity, as we demonstrate through experiments in Section 5.

5 Experiments and results

For the experiments, we have two parts. In the first part, we compare the accuracy of our sequence matrix method as well as their actual running time with the Heavypath algorithm, which is the name used in Ref.[8] for seeking top-N heaviest paths in a graph. In the second part, we use an actual web log dataset consisting of multiple sequences (sessions) and analyse the supper-sequence patterns in it. All of the experiments are run on 2.40 GHz Intel Xeon CPU with 24GB memory.

5.1 Accuracy of sequence matrix method and the running time comparison with existing heavy-path algorithm

Heavypath algorithm is proved to be able to find the accurate top-N paths given a number N. We used Cora (`http://www.cs.umass.edu/_mccallum/data`) as the real world graph data and generated a 70-node graph as was shown in Ref.[8] We also generated six synthetic directed weighted graphs of 100 nodes with 100 edges, 200 edges and 300 edges randomly distributed between two nodes. The length k of the seeking path is ranging from 2 to 3 (the running time for longer length takes huge amount of time using Heavypath) for Cora dataset and 2 to 7 for the synthetic dataset. Figure 5 to Figure 8 shows the results.

In Figure 5 we can see that the sequence matrix method runs much faster than the Heavypath method while they can both get the correct heaviest path in the graph. In Figure 6, Heavypath runs a little bit faster than our sequence matrix method, but as the graph getting denser, the running time for the Heavypath algorithm increases faster, while our algorithm still keeps almost the same running time as shown in Figure 7 and Figure 8 . This is because that as the number of edges increases, it takes more time to go through each edge in the rank join strategy to expand a path, while our method is based on the matrix which is only related to the number of nodes in the graph. Also, in our sequence matrix method, as the length k grows, the matrices that are going to be calculated are going to contain more 0s, which takes less and less time each round of adding up an edge to the paths.

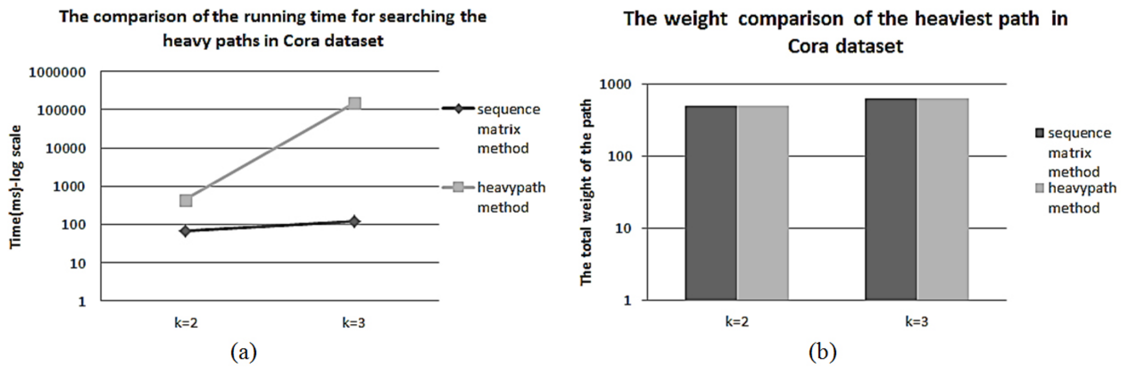

Figure 5: The comparison of the running time and the heaviest path weight on Cora Dataset

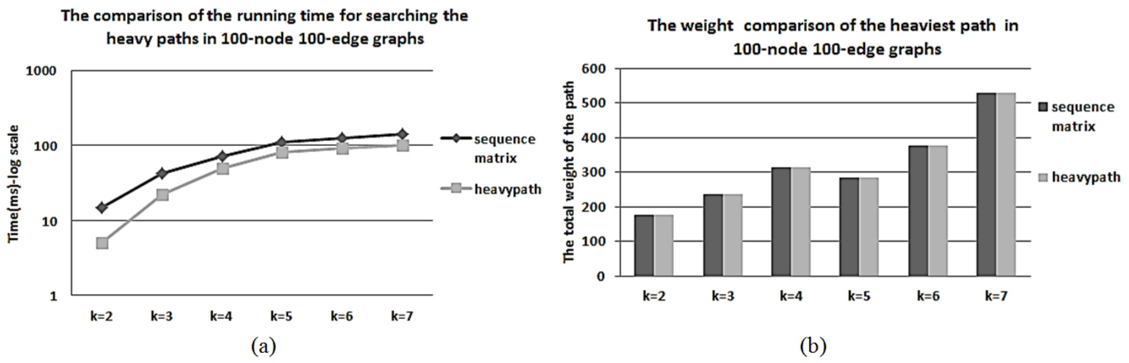

Figure 6: The comparison of the running time and the heaviest path weight on 100-node 100-edge dataset

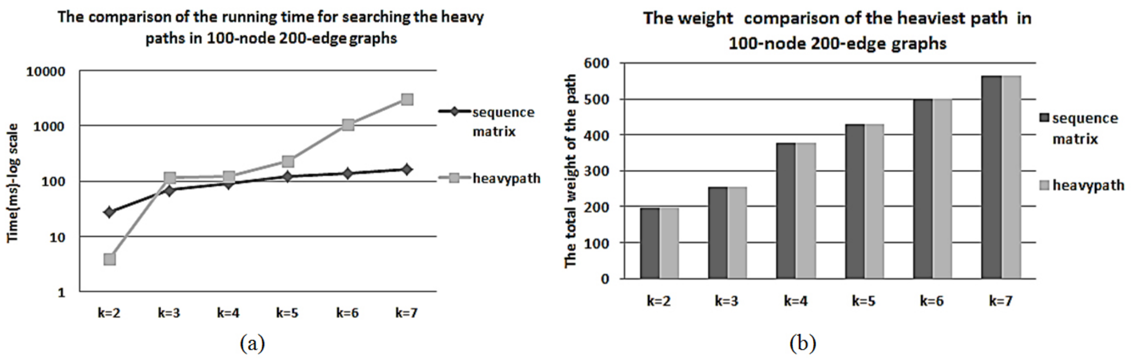

Figure 7: The comparison of the running time and the heaviest path weight on 100-node 200-edge dataset

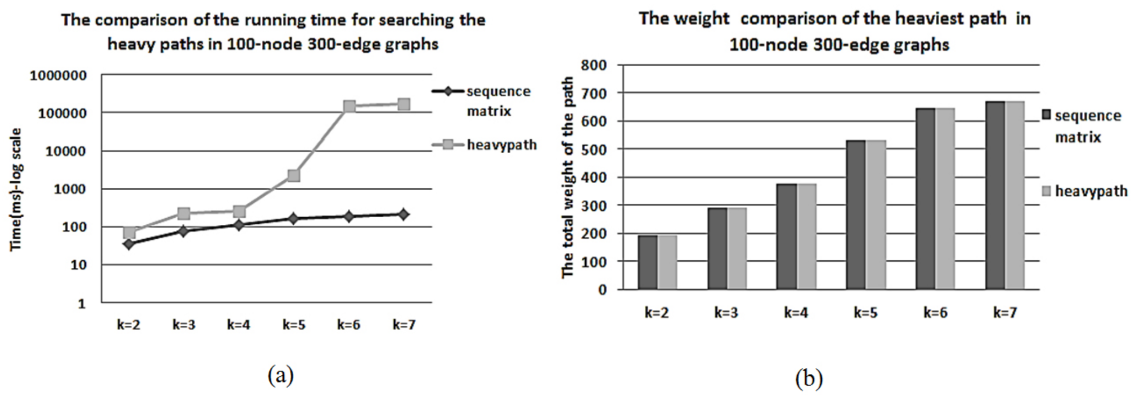

Figure 8: The comparison of the running time and the heaviest path weight on 100-node 300-edge dataset

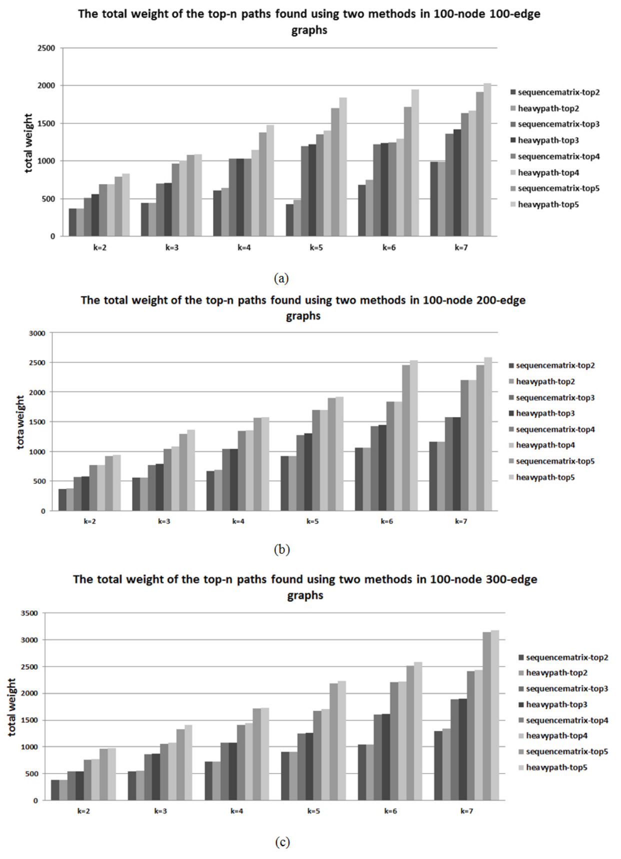

Figure 9: The comparison of the total weight of the top-N heavy paths found

To test our matrix method's accuracy, we also searched the heaviest path using both of the methods. The results are shown in Figure 6 (b), Figure 7 (b) and Figure 8 (b). We can see that our algorithm almost has the same accuracy as the Heavypath algorithm. Heavypath can find top-N heaviest paths, while our method can return a set of paths with larger weight than a given threshold. This means that if given a threshold δ, and the number of returned heavy paths is large or equal to N, there is a chance that N of these paths can be the top-N paths. To test the ability of our method seeking the top-N heavy paths, we did experiments on the two algorithms to find the top-2 to top-5 paths. Here the way to seek top-N heavy paths of using sequence matrix method is that we need to try some threshold numbers to get one that can return more than N heavy paths, and then from these paths pick the top-N heaviest. To show the accuracy, we compare the total weight of the N paths found by using the two methods to get two groups A and B of top-3 heavy paths. Then calculate the total weight of the paths in A and B respectively. The results are shown in Figure 9. We can see that our algorithm has very high accuracy.

From the experiment we can see that the sequence matrix method is more efficient and can give relatively accurate result. For the next experiment on real-world dataset, we are going to use it as the method to find the frequent super-sequences.

5.2 Super-sequence pattern analysis on web log dataset

The actual web log dataset that we use is called BMS-WebView-1 (BMS_WebView-1 has been used as datasets n KDDCup 2000 competition and can be downloaded from The Data Mining Forum: `http://forum.ai-direcotry.com`). It contains several months of clicking stream data from an E-commerce website. As discussed in Section 1, the web log data is transformed into a set of sequences. Each sequence is a session showing a sequence of web pages visited by a user. The details on web log pre-processing are out of the scope of this paper and the details can be found.[12,37,38] In BMS-WebView-1, there are 59602 sessions and 497 web pages in it. For simplicity, we give each web page an ID starting from 0 to 496.

In Figure 10, we show the number of sequences (paths) that we found with visiting numbers (total weight) from 2000 to 5000 and the sequence length (k) varies from 5 to 10. Similarly, Figure 11 shows the number of sequences that have visiting numbers from 5000 to 8000 and the sequence length goes from 12 to 20. Both figures confirm that, as expected, our algorithm finds more paths under the same visiting numbers as the sequence length increases. This is due to the fact that the total weight of a path increases as the sequence length increases.

From Figure 10 and Figure 11, we can see that there are a lot of paths (e.g., close to 30000) when path length is relatively long and visiting threshold is relatively small. Instead of such commonly appearing many paths, it would be useful to focus on the small number of paths having a relatively large weight (Nummer of visiting). For example, in this data set, one should closely analyse the 6-length paths with visiting numbers larger than 2200, or the 10-length paths with visiting numbers larger than 4000.

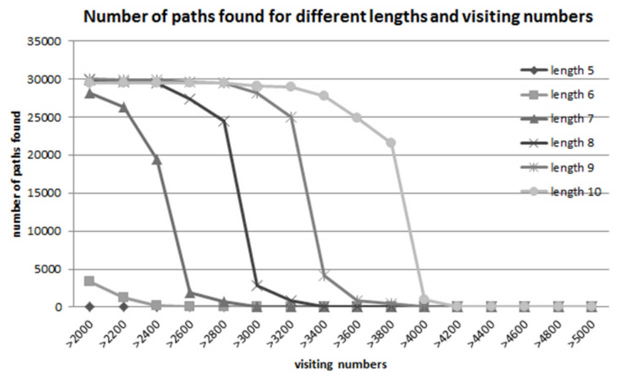

Figure 10: Number of paths found for different lengths (5-10) with different visiting numbers

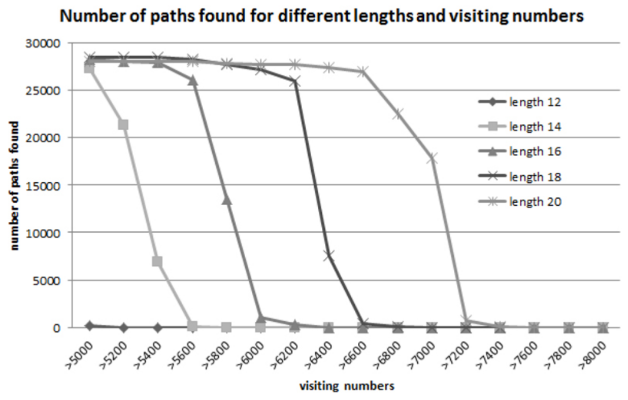

Figure 11: Number of paths found for different lengths (12-20) with different visiting numbers

For example, let us consider such distinguished paths to investigate if there is any weakest link, which has significantly small number of visitings compared to other links in a frequent super-sequence pattern. If there are any such links, web designers can delete or regroup them. To identify such links, we calculated the deviations of the distinguished paths with length 3, 4 and 5. From Table 2, 3 and 4 we can see that the number of paths significantly decreases as the visiting threshold increases. Accordingly, we pick the visiting numbers of 1000, 1500, and 2000 to get 18, 7, and 10 distinguished paths, respectively.

Table 2: The number of paths with length 3

Number of total visitings	Number of paths found
>900	169
>1000	18
>1100	5
>1200	3
>1300	1
>1400	0

Table 3: The number of paths with length 4

Number of total visitings	Number of paths found
>1400	9
>1500	7
>1600	5
>1700	4
>1800	2
>1900	1
>2000	0

Table 4: The number of paths with length 5

Number of total visitings	Number of paths found
>1900	11
>2000	10
>2100	6
>2200	2
>2300	1
>2400	0

Figure 12 shows the ordered standard deviation for the 18 paths found (total visitings > 1000) in Table 2. We can see that there are 3 paths with deviation less than 200 and 5 paths with standard deviation larger than 600. The rest of the paths have standard deviation between 200 and 600. After checking the path that has the smallest standard deviation, we found that the path has the web page sequence <163, 0, 1>. The visitings for page 0 after visiting page 163 is 590 and then number of visitings to page 1 is 621. The largest deviation exists in path <265, 281, 277>, where the visitings are 48 and 953, respectively. Figure 13 shows the ordered standard deviation for the 7 paths found (total visitings > 1500) in Table 3. The smallest deviation exists at the path with web page sequence <163, 0, 1, 97>. We can see that it is the same path before but extended with web page 97 and the visitings after page 1 to page 97 is 771. The largest deviation exists in path <45, 2, 281, 277> and the visitings are 877, 8, 953, respectively. Figure 14 shows the ordered standard deviation for the 10 paths found (total visitings > 2000) in Table 4. We found that the path that has the largest standard deviation has the web page sequence

<1, 45, 2, 281, 277> and the visitings are 193, 877, 8, 953, respectively. This means that visitings on web page 281 after users visiting page 2 has only 8 times occurring in the dataset. This is very few compared to other visitings in this super-sequence. So the designer might want to remove page 281 or combine its content with page 277.

To analyse the super-sequences found we looked into the 6 length-5 paths found which has total visitings greater than 2100 and they are listed in Table 5 in the page IDs. From the table we can see that the super-sequences mainly have two centres, one is related to page 0, 1, 97 and the other one is 45, 2, 281 and 277. We also looked into the top visited single pages and the top frequent occurring length-5 sub-sequences (each page in the sub-sequence occurs equal or more than 50 times) they are shown in Table 6 and Table 7.

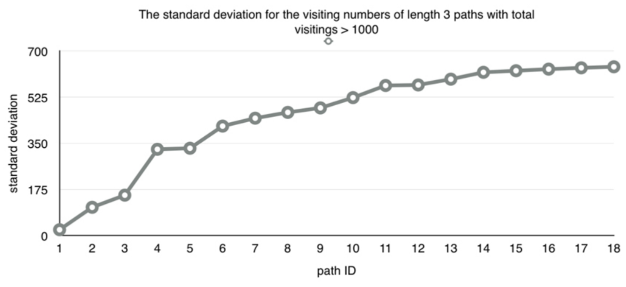

Figure 12: Visiting numbers standard deviation for length 3 paths with total visitings larger than 1000

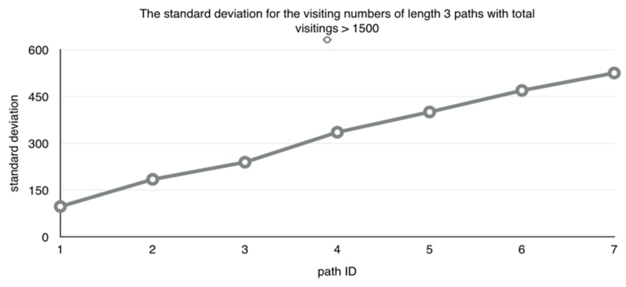

Figure 13: Visiting numbers standard deviation for length 4 paths with total visitings larger than 1500

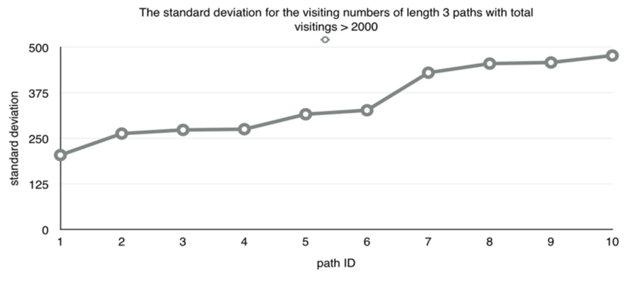

Figure 14: Visiting numbers standard deviation for length 5 paths with total visitings larger than 2000

Table 5: The 6 length-5 paths found with total vistings > 2100

Paths found	Total visiting number
{0, 1, 94, 45, 2}	2407
{17, 163, 0, 1, 97}	2140
{44, 45, 2, 281, 277}	2128
{45, 2, 293, 281, 277}	2154
{163, 0, 1, 97, 176}	2285
{163, 164, 0, 1, 97}	2120

Table 6: The top 5 most visted webpages

Webpage ID	Visiting numbers
5	1948
163	2009
45	2049
2	2268
1	2371
0	2797
97	3449
277	3612
35	3623
281	3658

Table 7: The top-6 most visited sub-sequences

Paths found	Total visiting number
167,60,9,4,5	1252
73,167,60,9,4	999
163,164,0,1,97	2120
293,308,266,276,281	386
308,266,276,281,285	551
292,293,308,266,276	416

From comparing Table 6 and Table 5 we can see that although page 5 and 35 are most visited single pages; they are not appearing in the super-sequences. This can imply that the pages connecting with 5 and 35 are spread out and then the link linking them are with less weight that the ones that can be selected as part of the super-sequences. This also means that page 5 and 35 may be hubs in the web page networks but not as the mainly behaviour flow of the users. So they may just intermediate pages to connect other pages. On the other hand from comparing Table 6 and Table 7 we can see that there is little overlap between the super-sequences and the frequent sub-sequences. This shows that frequent sub-sequences cannot reflect the problems that super-sequences can. For example, in Table 7, although each sub-sequence as a whole appears relatively frequent, where the weight on each link is between 20 and 30, they reflect the behaviour of a small amount of users. In Table 5, the average weight on each link is around 500, although some of the links may have a lot more visitings than the other, it still contains a lot more users' information than just using sub-sequences. Through the super-sequences we have better understanding of the users' behaviour flow and different trends of those behaviours.

6 Conclusion and future work

In this paper, we formulated a new form of sequential pattern mining, namely SS-FPM. We then proposed a heuristic sequence matrix method using dynamic programming techniques which gave high accuracy with a reasonable running time. We compared our method to the existing Heavy-path method in the experiment section and showed that our method is more efficient. Although the sequence matrix method can lose some of the actual paths during calculation, it can still give reasonable results. Accordingly, we used the matrix method to analyse actual web log dataset from a different perspective and found interesting facts based on the identified super-sequences in these datasets.

In this paper, we only consider patterns without loops, in our future research we would like to extend our algorithm on finding super-sequence patterns with loops. In the future, we would like to scale up our method on large datasets by developing their distributed and parallel versions. We also plan to apply our methods on other areas such as bioinformatics and so on.

Disclosure

An abridged version of this paper was presented in Workshop on Knowledge Management and Big Data Analytics(KMBA), to be held as part of the IEEE International Conference on Big Data, Oct 6-9, 2013 at Santa Clara, CA, USA.

References

[1] R. Agrawal, R. Srikant, et al. Fast algorithms for mining association rules." Proc. 20th Int. Conf. Very Large Data Bases. VLDB. 1994; 1215: 487-499.

[2] F. Sanger, S. Nicklen, A. R. Coulson. Dna sequencing with chain-terminating inhibitors. Proceedings of the National Academy of Sciences. 1977; 74(12): 5463-5467. http://dx.doi.org/10.1073/pnas.74.12.5463

[3] S. A. Hofmeyr, S. Forrest, A. Somayaji. Intrusion detection using sequences of system calls. Journal of computer security. 1998; 6(3): 151-180.

[4] B. Chikhaoui, S. Wang, H. Pigot. A frequent pattern mining approach for adls recognition in smart environments. Advanced Information Networking and Applications (AINA). 2011 IEEE International Conference on IEEE. 2011: 248-255. http://dx.doi.org/10.1109/aina.2011.13

[5] R. Agrawal, R. Srikant. Mining sequential patterns. Data Engineering. Proceedings of the Eleventh International Conference on IEEE. 1995: 3-14.

[6] J. Han, M. Kamber, J. Pei. Data mining: concepts and techniques. Morgan kaufmann. 2006.

[7] R. Ivancsy, I. Vajk. Frequent pattern mining in web log data. Acta Polytechnica Hungarica. 2006; 3(1): 77-90.

[8] M. Khabbaz, S. Bhagat, L. V. Lakshmanan. Finding heavy paths in graphs: A rank join approach. arXiv preprint arXiv:1112.1117, 2011.

[9] N. Idika, A. P. Mathur. A survey of malware detection techniques. Purdue University. 2007: 48.

[10] I. Sato, Y. Okazaki, S. Goto. An improved intrusion detecting method based on processprofiling. Transactions of the Information Processing Society of Japan. 2002; 43(11): 3316-26.

[11] E. L. Lawler. Combinatorial optimization: networks and matroids. Courier Dover Publications. 2001.

[12] W. Bin, L. Zhijing. Web mining research. Computational Intelligence and Multimedia Applications. 2003. ICCIMA 2003. Proceedings. Fifth International Conference on IEEE. 2003: 84-89.

[13] J. S. Park, M.-S. Chen, P. S. Yu. An effective hash-based algorithm for mining association rules. ACM. 1995; 24(2). http://dx.doi.org/10.1145/568271.223813

[14] A. Savasere, E. R. Omiecinski, S. B. Navathe. An efficient algorithm for mining association rules in large databases. 1995.

[15] H. Toivonen, et al. Sampling large databases for association rules. Proceedings of the International Conference on Very Large Data Bases. IEEE. 1996: 134-145.

[16] S. Brin, R. Motwani, J. D. Ullman, et al. Dynamic itemset counting and implication rules for market basket data. ACM SIGMOD Record. 1997; 26(2): 255-264. http://dx.doi.org/10.1145/253260.253325

[17] D. W. Cheung, J. Han, V. T. Ng, et al. Maintenance of discovered association rules in large databases: An incremental updating technique. Data Engineering, 1996. Proceedings of the Twelfth International Conference on IEEE. 1996: 106-114. http://dx.doi.org/10.1109/icde.1996.492094

[18] R. Agrawal, J. C. Shafer. Parallel mining of association rules. Knowledge and Data Engineering. IEEE Transactions. 1996; 8(6): 962-969.

[19] J. Han, H. Cheng, D. Xin, et al. Frequent pattern mining: current status and future directions. Data Mining and Knowledge Discovery. 2007; 15(1): 55-86. http://dx.doi.org/10.1007/s10618-006-0059-1

[20] J. Han, J. Pei, Y. Yin. Mining frequent patterns without candidate generation. ACM SIGMOD Record. 2000; 29(2): 1-12. http://dx.doi.org/10.1145/335191.335372

[21] M. J. Zaki, S. Parthasarathy, M. Ogihara, et al. New algorithms for fast discovery of association rules. in 3rd Intl. Conf. on Knowledge Discovery and Data Mining. 1997; 20: 283-286.

[22] R. Srikant, R. Agrawal. Mining generalized association rules. IBM Research Division, 1995.

[23] N. Pasquier, Y. Bastide, R. Taouil, et al. Discovering frequent closed itemsets for association rules. Database TheoryICDT99. Springer. 1999: 398-416. http://dx.doi.org/10.1007/3-540-49257-7_25

[24] F. Zhu, X. Yan, J. Han, et al. Mining colossal frequent patterns by core pattern fusion. Data Engineering. 2007. ICDE 2007. IEEE 23rd International Conference on IEEE. 2007: 706-715. http://dx.doi.org/10.1109/icde.2007.367916

[25] R. Srikant, R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. Springer. 1996. http://dx.doi.org/10.1007/bfb0014140

[26] J. Han, J. Pei, B. Mortazavi-Asl, et al. Freespan: frequent pattern-projected sequential pattern mining. Proceedings of the sixth ACM SIGKDD internationalconference on Knowledge discovery and data mining. ACM. 2000: 355-359. http://dx.doi.org/10.1145/347090.347167

[27] J. Han, J. Pei, B. Mortazavi-Asl, et al. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. Proceedings of the 17th International Conference on Data Engineering. 2001: 215-224. http://dx.doi.org/10.1109/icde.2001.914830

[28] M. J. Zaki. Spade: An efficient algorithm for mining frequent sequences. Machine learning. 2001; 42(1-2): 31-60.

[29] M. Garofalakis, R. Rastogi, K. Shim. Spirit: Sequential pattern mining with regular expression constraints. Proceedings of the international conference on very large data bases. 1999: 223-234.

[30] D. Karger, R. Motwani, G. Ramkumar. On approximating the longest path in a graph. Algorithmica. 1997; 18(1): 82-98. http://dx.doi.org/10.1007/bf02523689

[31] C. Murat, V. T. Paschos. The probabilistic longest path problem. Networks. 1999; 33(3): 207-219. http://dx.doi.org/10.1002/(sici)1097-0037(199905)33:3<207::aid-net7>3.3.co;2-z

[32] G. Fertin, H. M. Babou, I. Rusu. Algorithms for subnetwork mining in heterogeneous networks. Experimental Algorithms. Springer. 2012: 184-194. http://dx.doi.org/10.1007/978-3-642-30850-5_17

[33] K. Ioannidou, G. B. Mertzios, S. D. Nikolopoulos. The longest path problem has a polynomial solution on interval graphs. Algorithmica. 2011; 61(2): 320-341.

[34] K. Ioannidou, S. D. Nikolopoulos. The longest path problem is polynomial on cocomparability graphs. Algorithmica. 2013; 65(1): 177-205. http://dx.doi.org/10.1007/s00453-010-9411-3

[35] Y. Takahara, S. Teramoto, R. Uehara. Longest path problems on ptolemaic graphs. IEICE transactions on information and systems. 2008; 91(2): 170-177. http://dx.doi.org/10.1093/ietisy/e91-d.2.170

[36] X. Yu, T. Korkmaz. Super-sequence frequent pattern mining on sequential dataset. Big Data, 2013 IEEE International Conference on IEEE. 2013: 52-59. http://dx.doi.org/10.1109/bigdata.2013.6691783

[37] R. Cooley, B. Mobasher, J. Srivastava. Grouping web page references into transactions for mining world wide web browsing patterns. Knowledge and Data Engineering Exchange Workshop. Proceedings. IEEE. 1997: 2-9. http://dx.doi.org/10.1109/kdex.1997.629824

[38] P. Pirolli, J. Pitkow, R. Rao. Silk from a sow's ear: extracting usable structures from the web. Proceedings of the SIGCHI conference on Human factors in computing systems: common ground. ACM. 1996: 118-125. http://dx.doi.org/10.1145/238386.238450

[39] Wu, Fei, Jayant Madhavan, Alon Halevy. Identifying aspects for web-search queries. Journal of Artificial Intelligence Research. 2011: 677-700. http://dx.doi.org/10.1007/978-3-642-25631-8_23

[40] Yu, Xinran, et al. Heavy path mining reveals novel protein-protein associations in the malaria parasite plasmodium falciparum. Bioinformatics and Biomedicine (BIBM), 2014 IEEE International Conference on IEEE. 2014.