AN EFFICIENT-DISTRIBUTED MODEL FOR MINING SEQUENTIAL PATTERNS ON A LARGE SEQUENCE DATASET

Tran Minh Thai
Department of Information Technology, HUFLIT
minhthai@huflit.edu.vn

ABSTRACT

Sequential pattern mining is an active research area because of its many different applications. There have been many studies suggesting efficient mining algorithms. With the current trend, the size of the sequence dataset is growing, and research has been applying the distributed processing model on the problem of sequential patterns on sequence databases. One of the algorithms that apply the distributed modeling to the efficient sequential pattern mining algorithm is the sequential pattern mining algorithm based on the MapReduce model on the cloud (SPAMC). However, SPAMC is still limited in mining datasets that have a large number of distinct items. This article proposes a distributed algorithm to deal with this problem, called the distributed algorithm for sequential pattern mining on a large sequence dataset using dynamic vector bit structures on the MapReduce distributed programming model (DSPDBV). In addition, the algorithm uses different techniques for early prune redundant candidates and reduce the amount of memory usage. Experimental results show that DSPDBV is highly efficient and scalable for large sequence datasets. Moreover, DSPDBV is more efficient than SPAMC handling datasets have a large number of distinct items.

Keywords: *Distributed sequential pattern mining, Big data, Data mining, Dynamic bit vector, Sequential pattern, MapReduce.*

1. Introduction

A sequential pattern mining is a process of finding sequential patterns in a sequence dataset. It is an important stage in sequence data mining. This issue plays a significant role in data mining and has many useful applications in a variety of fields, such as economics, biology, e-commerce, or software [1]. For example, in the field of economics, the common behavior of an object or group of objects can determine the relationship between them. In the field of biology or medicine, these patterns can be used to detect abnormalities of protein or DNA structures. In addition, in the software field, these patterns can help predict inconsistent or unusual codes. The patterns can also be used in the field of phrase analysis in text documents.

The sequential pattern mining problem has been extensively discussed and researched since Agrawal and Srikant first proposed the AprioriAll [2] algorithm in 1995. Since then, many authors have proposed other techniques, such as A priori-based algorithms including GSP [3], SPADE [4], SPAM [5], LAPIN-SPAM [6], FreeSpan [7] and PrefixSpan [8]. However, these algorithms only mine locally on a single computer with a small dataset, and these algorithms still have limitations, such as difficulty expanding and communication costs in the system. To improve the performance and scalability of sequential pattern mining on large datasets, many parallel mining techniques based on distributed programing model MapReduce [9] have been proposed such as PTDS [10] and SPAMC [11]. Although these algorithms have somewhat overcome the performance and scalability issues, there is still the limitation of the runtime as the number of items increases.

To address this problem, an algorithm is proposed that can mine the distributed sequential pattern using the dynamic bit vector based on the MapReduce framework called DSPDBV. The rest of this article is constructed as follows. Section 2 contains the relevant preliminary concepts. Section 3 gives a brief description of the related works. Section 4 shows the proposed algorithm. Section 5 presents the experimental results, and the conclusions and future studies are included in Section 6.

2. Problem Definition

This section defines some basic concepts of sequence dataset mining and the issue of sequential pattern mining.

Let $I = \{i_1, i_2, i_3, \ldots, i_n\}$ be a set of distinct items (or events), where $\{i_j\}$ represents an item and $1 \leq j \leq n$. A set of unordered items is called an itemset, and each itemset is represented in brackets. The lexicographical order denoted by $\succ_{lex}$ is defined as any total on I. All items in each itemset are ordered based on $\succ_{lex}$. A sequence $S = (e_1, e_2, e_3, \ldots, e_m)$ represents an ordered list of itemsets, where

e_j represents an itemset ($1 \leq j \leq m$). The size of a sequence represents the number m of itemsets in the sequence. A sequence with length k, called a k-sequence, represents the number of items in the sequence. A sequence dataset SDB is a list of sequences and is denoted as SDB = {S_1, S_2, S_3, ⋯, $S_{|SDB|}$}, where |SDB| represents the number of sequences in SDB, and S_i ($1 \leq i \leq |SDB|$) represents the i-th sequence in SDB.

This article determines the full set of sequential patterns in a large sequence dataset SDB with a given minSup value.

3. Related Work

Since Agrawal and Srikant proposed the AprioriAll [2] algorithm for the sequential mining problem, sequential pattern mining has frequently studied. Initially, these algorithms performed linear processing. Then, when the size of the dataset increase, the sequential pattern mining has become inefficient. Thus, some studies have proposed algorithms for parallel mining sequential patterns. Based on the characteristics of linear sequential pattern mining, these algorithms can be classified in some approaches: the horizontal dataset format algorithms, such as AprioriAll [2] and GSP [3]; the vertical dataset format algorithms such as SPADE [4], SPAM [5], and LAPIN-SPAM [6]; the pattern growth algorithms such as FreeSpan [7] and PrefixSpan [8].

Algorithms that used the data structure in a vertical format have proven to be more effective than others because they only scan the dataset one or two times. Furthermore, during the mining process, they only use the information in the vertical format table to generate new candidates and count their support by logic and bitwise operators. The SPADE [4] is a typical algorithm designed in this approach. The SPADE represents the data as an IDList structure. To reduce the memory usages of SPADE, SPAM, bitSPACE [12], and LAPIN-SPAM [6] used a bit vector data structure to represent the data, where the value 1 corresponds to the occurrence of items in each transaction; otherwise, the values are set to 0. To eliminate redundant bits (0), Vo et al. proposed a dynamic bit vector structure (DBV) in DBV-Miner [13]. Then, Tran proposed the CloFS-DBV algorithm [14].

In addition, to eliminate the redundant candidates early, Philippe Fournier-Viger et al. proposed the CMAP structure [15] and CM-SPAM algorithm [15] in 2014. Data is stored in CMAP to map each item *k* in *I* to a set of items that can be associated with item *k* in two types of pattern extensions to get new candidates. With a given minSup, CMAP is considered as a reference table of frequent 2-sequence that satisfies the minSup.

The sequence datasets tend to increase quickly and be more complex. Consequently, these datasets lead to a large space of mining being used. Therefore, the problem of mining sequential patterns consumes more resources and time. Thus, a parallel model is the ideal solution for mining sequential patterns. Some parallel algorithms were proposed, such as pSPADE [16]. In addition, PIB-PRISM [17] and pDBV-SPM [18] algorithms have been proposed based on a multi-core processor. Although these methods have good performance and scalability, they still suffer from limitations, such as limited resources, communication costs, and load balancing.

Hadoop MapReduce [19] is based on the Google MapReduce platform, allowing substantial amounts of distributed data across computer clusters with high reliability and availability. The algorithms that have been designed on this platform include PTDS [10] and SPAMC [11]. The PTDS divides the original dataset into small datasets and executes the mining based on PrefixSpan. The limitation of PTDS is that it mines all the set of candidates in the subset of the dataset, so it does not efficiently work on long sequences. Alternatively, SPAMC is based on SPAM. Instead of mining on a full lexigraphy dictionary, it divides patterns into sub-trees and uses loops based on the MapReduce model for each sub-tree to find patterns. However, the SPAMC algorithm is not effective for large distinct items, since it takes a long time to transfer data from the MapReduce function.

4. The Proposed Algorithm

4.1. Data representation

The data structure of the proposed algorithm uses a dynamic bit vector structure that combines the location information of the patterns on each transaction that was first proposed in the CloFS-DBV algorithm [14], known as the CloFS-DBVPattern. In the DSPDBV algorithm, we make improvements to the data structure to increase the algorithm's performance. Instead of being represented as a list of positions, we use a bit vector to represent the positions of the sequential patterns on the transactions.

To extend the patterns, we prebuilt the reference matrix for the positioning process of the candidate pattern on the transactions, called the sMATRIX and iMATRIX.

4.2. DSPDBV Algorithm

Given a sequence dataset SDB and a sequence S ∈ SDB, suppose SDB is divided into *n* partitions. Thus, SDB = $p_1 + p_2 + \ldots + p_n$. Then, $sup(S/SDB) = sup(S/p_1) + sup(S/p_2) + \ldots + sup(S/p_n)$. There are three remarks: (1) Given a minimum threshold support (minSup), the sequence S is called a pattern if $sup(S/p_1) + sup(S/p_2) + \ldots + sup(S/p_n) \geq minSup$; (2) Assume S is any pattern of SDB; if S exists in p_i ($1 \leq i \leq n$), then S is also considered a pattern in p_i; (3) Given two patterns S_A and S_B, where $|S_A| < |S_B|$, the candidate generation time of S_A is less than that of S_B because S_A has fewer candidates than S_B.

❖ **Distributed mining using MapReduce**

Table 1: *DSPDBV algorithm*

Algorithm 1: DSPDBV

Input: an *SDB*, a *minSup* and the maximum depth of child prefix tree *depth*

Output: complete set of frequent sequential patterns

```
1.let isExists;
2.let partitions = split SDB into n partitions;
3.call DistributedDBVConversion (partitions, minSup);
4.let isExists = check if exist output for listDBVItems;
5.while (isExists)
6.    call DistributedPatMining(listDBVPattens, minSup, depth);
7.    let isExists = check if exist output for listDBVPatterns;
```

MapReduce programing model is used for large data processing based on the theory of parallel computing and distributed data processing model on clusters of computers. The proposed algorithm (Algorithm 1) is implemented on a MapReduce model for parallel and distributed sequential pattern mining. In which, a Map function performs a data conversion and generates candidates. A Reduce function calculates the support of candidates, and candidates that satisfy the minSup value are stored. The implementation process is then divided into two phases: (1) DBV Conversion Phase: It is done by a Map and a Reduce function. The Map function performs Step 2, and the Reduce function performs Step 3 of the algorithm; (2) Sequential Pattern Mining Phase: It is also done by a Map and Reduce function. It performs Steps 4 and 5. In the mining process, this phase calls back based on the loop of the execution of MapReduce process. MapReduce works on data defined by a pair <key, value>, where key represents an item or a sequential pattern, and value represents a BlockInfo or a support. Thus, while the execution of MapReduce process, the algorithm must convert the data into (key, value). The details of this process are described in the following section.

Table 1 shows the pseudocode of DSPDBV algorithm. First, it divides SDB into n partitions (line 2). Then, it scans these partitions to convert sequences into the DBVItem structure and find 2-sequences (line 3). Lines 5-7 perform the sequential patterns by mining subtrees in the same level, the candidates are generated by extending nodes on these subtrees. This process is repeated until there are no new candidates.

❖ **Distributed DBV Conversion**

The dataset after being partitioned is distributed and stored in HDFS. In the DBV Converting Phase, the data conversion process is performed in parallel by executing a MapReduce job process (Table 2), and a Map function is called to do a partition. Each Map function builds the DBVItem dataset. The Map function also finds sets of 2-sequences to build CMAP (line 1). There are two types of key-value pairs that are output to the Reduce function, and key-value pairs with the same key are transferred to the same Reduce function. If the data is DBVItems then key-value is <item, blockinfo>. In which, the key is an item in DBVItem, and the value is blockinfo of this item (lines 2-3). The other

is <pattern, support> if data is 2-sequences where the 2-sequence key and value support this sequence (lines 4-5).

Table 2: *DistributedDBVConversion algorithm*

```
Algorithm 2: DistributedDBVConversion
Mapper Side
Input: a partition of sequence database p
   1.scan partition data only one time to
     a. DBVItems<item,blockInfo>←construct DBVItem for each item
     b. sequences<pattern,support>←find 2-sequences and its support
   2.for each (dbv in DBVItems)
   3.     output <dbv.item, dbv.blockInfo>
   4.for each (item in sequences)
   5.     output <item.pattern, item.support>
Reducer Side
Input: mapper output pairs <key, values> and a minSup
Output: complete set of frequent items, 2-sequences and set of DBVItems
divide by partition (listDBVItem)
   1.let support;
   2.let blockInfos<BlockInfo>;
   3.if (key is pattern) // pattern is 2-sequence
   4.     for each (value in values)
   5.         let support = sum of sup(value);
   6.     if (support >= minSup)
   7.         output <key, support>;
   8.else if (key is item)
   9.     for each (value in values)
  10.         let support = sum of sup(value);
  11.         add value to blockInfos;
  12.     if (support >= minSup)
  13.         output <key, support>;
  14.         for each (blockInfo in blockInfos)
  15.             output <key, blockInfo> divide by partition;
```

Reduce function calculates the sum support of items or 2-sequences. With the key is 2-sequences, sum support is calculated by the sum of the value and eliminates the 2-sequences that do not satisfy the minSup. (lines 8-12). With the key is the item, the support is counted by DBV in BlockInfo, and items that meet the minSup condition are retained in each corresponding partition (lines 13–15).

❖ **Distributed Sequential Pattern Mining**

In the sequential pattern mining process, the exploitation of the entire candidate patterns on the DBVTree prefix tree for large data sets is ineffective due to the wide search space that requires a lot of resources. In addition, branches of the tree are independent of each other, so the process of generating candidates can be performed independently.

Thus, the proposed algorithm is designed to perform a tree level to make the mining process more flexible and require fewer resources. Candidates in the sub-prefix tree are exploited independently in parallel by implementing MapReduce processes. Furthermore, the branches of the tree have different heights, with the parallel exploitation of the sub-prefix trees high enough to make the mining process achieve better load balancing. In addition, the algorithm is designed to optimize the data transfer between Map and Reduce, and details of the implementation of distributed mining are described in Algorithm 3 (Table 3).

Table 3: *Distributed Sequence Pattern Mining algorithm*

Algorithm 3: DistributedPatMining

Mapper Side

Input: a list of root nodes of child prefix tree *roots*, a *minSup* and the maximum depth of child prefix tree *depth*

```
1.scan listDBVItem of particular partition only one time to
  a. load list of DBVItems into DBV<item,blockInfo>
  b. extract list item frequent items into items<item>
2.scan 2-sequence data only one time to construct CMAP
3.for each (root in roots)
4.  call DBVPattenExt(root, DBV, CMAP, items, items, 1, depth);
```

Reducer Side

Input: mapper output pairs *<key, values>*, a *minSup* and the maximum depth of child prefix tree *depth*

Output: complete set of frequent sequential patterns and set of DBVPatterns divide by partition (listDBVPattern)

```
5.let support;
6.let blockInfos<BlockInfo>;
7.for each (value in values)
8.    let support = sum of sup(value);
9.    if (key is a node at lowest depth in child prefix tree)
10.       add value to blockInfos
11.if (support >= minSup)
12.    output <key, support>;
13.    if (key is a node at lowest depth in child prefix tree)
14.       for each (blockInfo in blockInfos)
15.           output <key, blockInfo> divide by partition;
```

Generate Candidate Pattern: Candidates are generated in parallel by the Map function. The algorithm uses the prefix tree structure, where nodes arranged in the order of the dictionary, helping the process of generating the candidate pattern is highly effective. Each node in the tree is a candidate pattern, expressed in the DBVPattern structure. Data is then divided into subsets so that each node in the tree is independently exploited in different Map functions to improve execution time. The extending subtree process is done in three main steps: (1) building a frequent DBV set, (2) constructing a CMAP structure, and (3) expanding the subtree (Table 3).

Find the sequential pattern: The Reduce function calculates the total support of the candidate patterns. In addition, if the candidate pattern is the node at the last level of the tree, the BlockInfo information of the candidate pattern is retained (lines 7-10). Candidates that meet the minSup condition are retained (lines 10-11), and the DBVPatten information of these candidates are stored and distributed corresponding to the input data (listDBVPattern datasets) to perform a sequential mining phase (lines 13-15) (Table 4).

Table 5 describes the OutputDBVPattern algorithm called in the DBVPatternExt algorithm to perform the resulting transfer from the Map function to the Reduce function. Instead of transmitting all

the BlockInfo information of each node to the Reduce function, the algorithm only passes BlockInfo information to the Reduce function if the node is the node at the last level of the tree (lines 1-2). Otherwise, only the support of the node is passed (lines 3-4). This reduces the amount of data transfer between the Map function and the Reduce function, limits the bottleneck, and increases the processing efficiency of the Reduce function by processing fewer data.

Table 4: *DBVPatternExt algorithm*

Algorithm 4: DBVPattenExt

Input: a root node *root*, a set of frequent items *DBV*, a co-occurrence map *CMAP*, list of extendable items *sItems*, *iItems*, and depth information *depth*, *maxDepth*

```
1.if (depth > maxDepth)
2.    return;
3.let sTemp<item> = NULL;
4.let sNode<DBVPattern> = NULL;
5.for each (item in sItem) // extendable item for sequence extension
6.    if (item is not pruned by CMAP-sequence-extension)
7.        if (sup(let node = SequenceExt(root, DBV(item)))> 0)
8.            add item to sTemp;
9.            add node to sNode;
10.for each (node in sNode)
11.    call OutputDBVPatten(node, depth == maxDepth)
12.    call DBVPattenExt(node,DBV,CMAP,sTemp,iItem,depth+1,maxDepth);
13.let iTemp<item> = NULL;
14.let iNode<DBVPattern> = NULL;
15.for each (item in iItem)
16.    if (item is not pruned by CMAP-itemset-extension)
17.        if (sup(let node = ItemsetExt(root, DBV(item))) > 0)
18.            add item to iTemp;
19.            add node to iNode;
20.for each (node in iNode)
21.    call OutputDBVPatten(node, depth == maxDepth)
22.    call DBVPattenExt(node,DBV,CMAP,sTemp,iTemp,depth+1,maxDepth);
```

Table 5: *OutputDBVPattern algorithm*

Algorithm 5: OutputDBVPatten

Input: a node of prefix tree *node* and flag to check if node is at lowest depth *flag*

```
1.if (flag)
2.    output <node.pattern, node.BlockInfo>;
3.Else
4.    output <node.pattern, sup(node)>;
```

5. Experiment Results

This section presents the experimental results of the proposed algorithm. The datasets generated from an IBM synthetic data generator. The characteristics of synthetic datasets are described in Table 6. The source codes used in the experiment were written in JAVA programing language, and the empirical process consists of two phases. The first stage compares the execution time of the DSPDBV with SPAMC. The second stage then tests the DSPDBV extensibility.

In the first phase, the SPAMC and DSPDBV algorithms were implemented on synthetic datasets with different parameters and minimum support thresholds. The algorithms were implemented

in the Hadoop Cluster on four computers: one master and three slaves. All computers were running Ubuntu Server 13.04 (64bit version) installed Hadoop 2.7.3 and Open JDK 1.7. All computers communicated with each other through the SSH protocol, and the configuration details of the machines are described in Table 7. For each experiment, the algorithm execution time was recorded. Each experiment was conducted three times on each dataset with the same set of parameters and support thresholds. The execution time of the algorithm was calculated as the average time of three executions, and the execution times of these algorithms were compared.

Table 6: *Characteristics of synthetic datasets*

Parameter	Description
D	Number of sequences
C	Average number of transactions per sequence
T	Average number of items per itemset
N	Number of distinct items

The second stage performs similar results in the first stage. In particular, the stage implemented the DSPDBV algorithm with different support thresholds and a different number of computers in the Hadoop Cluster.

Table 7: *Configuration of computers in the experiments*

Component	Master	Slave
CPU	Intel(R) Xeon(R) CPU E3-1220 V2 @ 3.10GHz	Intel(R) Xeon(R) CPU E3-1230 V2 @ 3.30GHz
Memory	10GB	8GB
Storage	500GB, 7200RPM	500GB, 7200RPM
Network	2 X Ethernet 1000MB	2 X Ethernet 1000MB

5.1. Compared to SPAMC

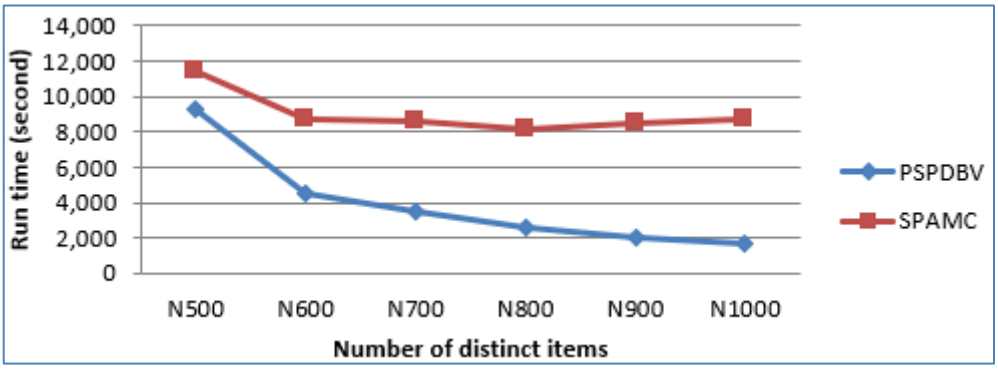

Figure 1: *Runtime comparison of DSPDBV and SPAMC*

This experiment performed a runtime comparison on the D500kC5T5 dataset with the number of distinct items *N* with values of 500, 600, 700, 800, 900, and 1000. The minSup value was set as 0.1% on a cluster with four computers. When increasing *N*, DSPDBV decreases the execution time. While SPAMC has reduced execution time when *N* increases from 500 to 800, the execution time increased as *N* increases to 900 and 1000 (Figure 3).

5.2. Scalability and Extensibility

To test the scalability of the algorithm, the size of datasets was increased with different support thresholds on a cluster with a different number of computers. First, the experiment on the C5T5N1000 dataset with *D* increased from 1 M to 5 M (the gap is 1 M).

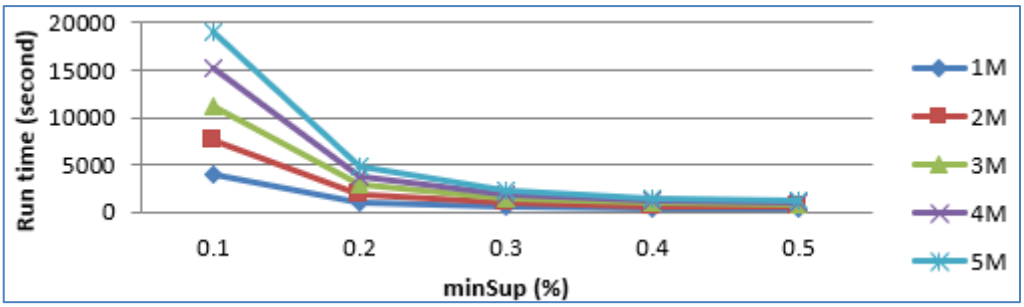

Figure 2: *Effect of the dataset on mining efficiency*

These datasets were duplicated from D500kC5T5N1000. For example, dataset D1000kC5T5N1000 was obtained by joining two datasets (D500kC5T5N1000). The minSup support threshold then varied from 0.1% to 0.5% (the gap is 0.1%). The algorithm executed on a cluster consists of four computers (Figure 4).

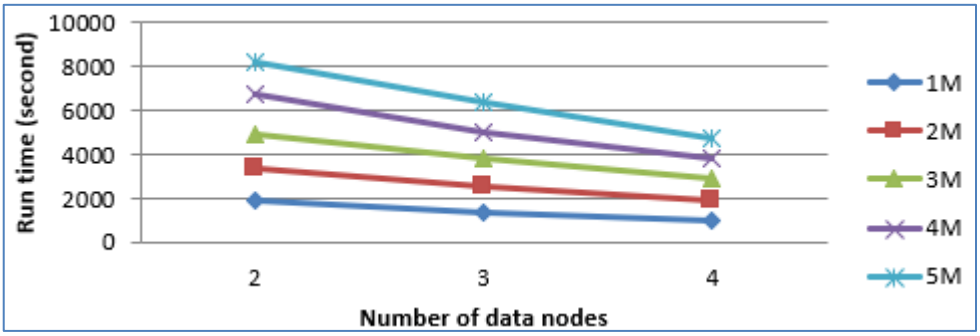

Figure 3: *Effect of the number of nodes on mining efficiency*

Next, the algorithm was performed on a cluster with a different number of computers (nodes) to determine the effect of the number of nodes when performing a parallel mining. The experiment was performed with a minSup support threshold of 0.2% on dataset C5T5N1000 with *D* changed from 1 M to 5 M (the gap is 1 M) with 2, 3, and 4 nodes (Figure 5).

6. Conclusion and future work

In this article, a parallel and distributed sequential mining algorithm on a large sequence dataset was proposed. The experiments were deployed on a MapReduce distributed programing environment, and the proposed algorithm scans the dataset only one time for the entire mining process. The results show that the DSPDBV algorithm is highly effective when parallel mining sequential patterns are performed on cluster computers. The algorithm applies in-depth mining and parallel execution on datasets and subtrees with the same level. Mining on subtrees instead of whole trees at the same time improved the load balancing performance between processes. In addition, using the dynamic bit vector structure combined with iMATRIX and sMATRIX reduced the memory cost and computation time. Moreover, the algorithm detected and early pruned redundant candidates by using the co-occurrence map structure for frequent itemsets. The cost of the checking operation is lower than that of the candidate generation, allowing the algorithm to execute with optimum time.

Due to the compactness of closed sequential patterns, this approach can be further developed to mine these patterns. In addition, based on the efficient data structure discussed in this article, this approach could be extended and applied to additional methods in the MapReduce model for large sequence datasets.

REFERENCES

[1] Dong G., Pei J. Sequence Data Mining, New York: Springer Science+Business Media, LLC, 2007.

[2] R. Agrawal and R. Srikant. Mining sequential patterns, *Proc. of IEEE International Conference on Data Engineering*, pp. 3-14, 1995.

[3] R. Srikant and R.Agrawal. Mining sequential patterns: Generalizations and performance

improvements, *Springer Berlin Heidelberg*, pp. 1-17, 1996.

[4] M. Zaki. SPADE: An efficient algorithm for mining frequent sequences, *Machine Learning*, **42:**1-2:31–60, 2001.

[5] Ayres. J, Gehrke. J, Yiu. T, Flannick. J. Sequential PAttern Mining using A Bitmap Representation, *SIGKDD '02 Edmonton*, Alberta, Canada, 2002.

[6] Yang, Z., Kitsuregawa, M. LAPIN-SPAM: An Improved Algorithm for Mining Sequential Pattern, *The International Conference on Data Engineering Workshops*, 2005.

[7] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M.-C. Hsu. FreeSpan: Frequent Pattern-Projected Sequential Pattern, *ACM SIGKDD Int'l Conf. Knowledge Discovery*, 2000.

[8] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal and M. C. Hsu. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth, *2013 IEEE 29th International Conference on Data Engineering (ICDE). IEEE Computer Society.*, pp. 0215-0215, 2001.

[9] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters, *Communications of the ACM*, **51**:1:107-113, 2008.

[10] Wang. X, Wang. J, Wang. T, Li. H, Yang. D. Parallel Sequential Pattern Mining by Transaction Decomposition, *The Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD 2010)*, 2010.

[11] Chen CC, Tseng CY, Chen MS. Highly scalable sequential pattern mining based on MapReduce model on the cloud, *IEEE international congress on big data (BigData Congress'13),* p. 310–317, 2013.

[12] S. Aseervatham, A. Osmani, E. Vienne. bitSPADE: A Lattice-based Sequential Pattern Mining Algorithm Using Bitmap Representation, *The International Conference on Data Mining*, 2006.

[13] Vo, B., Hong, T.P., Le, B. DBV-Miner: a dynamic bit-vector approach for fast mining frequent closed itemsets, *Expert Systems with Applications, 39,* p. 7196–7206, 2012.

[14] M. Tran, B. Le and B. Vo. Combination of dynamic bit vectors and transaction information for mining frequent closed sequences efficiently, *Engineering Applications of Artificial Intelligence*, **38**:183-189, 2015.

[15] P. G. A. C. M. T. R. Fournier-Viger. Fast Vertical Mining of Sequential Patterns Using Co-occurrence Infomation, *PAKDD 2014,* no. LNAI 8443, pp. 40-52, 2014.

[16] M. J. Zaki. Parallel Sequence Mining on Shared-Memory Machines, *Journal of Parallel and Distributed Computing,* **61**:401-426, March 2001.

[17] Huynh. B, Vo. B, Using multi-core processors for mining frequent sequential patterns, *ICIC Express Letters*, **9**:11:3071-3079, 2015.

[18] Huynh. B, Vo. B, Snasel. V. An efficient method for mining frequent sequential patterns using multi-Core processors, *Applied Intelligence*, **46**:703-716, 2017.

[19] The Apache Software Foundation, *Apache Hadoop*.

[20] Guralnik V., Karypis G. Parallel tree-projection-based sequence mining algorithms, *Parallel Computing,* **30**:4:443–472, 2004.

MÔ HÌNH PHÂN TÁN HIỆU QUẢ CHO KHAI THÁC MẪU TUẦN TỰ TRÊN TẬP DỮ LIỆU CHUỖI KÍCH THƯỚC LỚN

Trần Minh Thái
Khoa Công nghệ thông tin, Đại học Ngoại ngữ - Tin học TP. HCM
minhthai@huflit.edu.vn

Tóm tắt: Khai thác mẫu tuần tự là một lĩnh vực nghiên cứu tích cực bởi vì chúng có nhiều ứng dụng khác nhau và đã có nhiều nghiên cứu đề xuất các thuật toán khai thác hiệu quả. Với xu hướng hiện tại, kích thước của tập dữ liệu chuỗi đang gia tăng, và nghiên cứu áp dụng mô hình xử lý phân tán trong bài toán mẫu tuần tự trên dữ liệu chuỗi. Một trong những thuật toán áp dụng mô hình phân tán vào

thuật toán khai thác mẫu tuần tự hiệu quả là thuật toán khai thác mẫu tuần tự dựa trên mô hình MapReduce (SPAMC). Tuy nhiên, SPAMC vẫn còn giới hạn trong khai thác tập dữ liệu có số lượng sự kiện phân biệt lớn. Bài báo này đề xuất một thuật toán phân tán nhằm giải quyết vấn đề trên, được gọi là thuật toán phân tán cho khai thác mẫu tuần tự trên tập dữ liệu chuỗi lớn sử dụng cấu trúc vector bit động trên mô hình lập trình phân tán MapReduce (DSPDBV). Ngoài ra, thuật toán đề xuất sử dụng các kỹ thuật khác nhau để tỉa sớm ứng viên dư thừa và cắt giảm dung lượng bộ nhớ sử dụng. Các kết quả thực nghiệm cho thấy rằng DSPDBV hiệu quả và khả năng mở rộng cao cho các tập dữ liệu lớn. Hơn nữa, DSPDBV hiệu quả hơn SPAMC cho việc xử lý các tập dữ liệu có số lượng sự kiện phân biệt lớn.

Từ khóa: *Khai thác mẫu tuần tự phân tán, Dữ liệu lớn, Khai thác dữ liệu, Vector bit động, Mẫu tuần tự, MapReduce.*

TS. Trần Minh Thái tốt nghiệp cử nhân ngành Công nghệ Phần mềm vào năm 2001 và thạc sỹ Tin học vào năm 2006 tại trường Đại học Khoa học Tự nhiên thành phố Hồ Chí Minh, nhận bằng Tiến sỹ Khoa học Máy tính vào năm 2017 do Đại học Quốc gia thành phố Hồ Chí Minh cấp. Anh ta từng là giảng viên và quản lý khoa Công nghệ Thông tin trường Cao đẳng Công nghệ Thông tin thành phố Hồ Chí Minh từ năm 2002 đến 2015. Từ năm 2015 đến hiện tại, anh ta là giảng viên và là Trưởng bộ môn Hệ thống Thông tin thuộc khoa Công nghệ Thông tin trường Đại học Ngoại ngữ - Tin học thành phố Hồ Chí Minh. Lĩnh vực nghiên cứu chính của anh ta liên quan đến vấn đề khai thác dữ liệu và xử lý dữ liệu lớn.