Applied Intelligence (2019) 49:150–171
https://doi.org/10.1007/s10489-018-1259-2

A novel mapreduce algorithm for distributed mining of sequential patterns using co-occurrence information

Sumalatha Saleti[1] · **R. B. V. Subramanyam**[1]

Published online: 20 August 2018

Abstract
Sequential Pattern Mining (SPM) problem is much studied and extended in several directions. With the tremendous growth in the size of datasets, traditional algorithms are not scalable. In order to solve the scalability issue, recently few researchers have developed distributed algorithms based on MapReduce. However, the existing MapReduce algorithms require multiple rounds of MapReduce, which increases communication and scheduling overhead. Also, they do not address the issue of handling long sequences. They generate huge number of candidate sequences that do not appear in the input database and increases the search space. This results in more number of candidate sequences for support counting. Our algorithm is a two phase MapReduce algorithm that generates the promising candidate sequences using the pruning strategies. It also reduces the search space and thus the support computation is effective. We make use of the item co-occurrence information and the proposed Sequence Index List (SIL) data structure helps in computing the support at fast. The experimental results show that the proposed algorithm has better performance over the existing MapReduce algorithms for the SPM problem.

Keywords Big data · Co-occurrence map · Data mining · MapReduce framework · Sequential pattern mining

1 Introduction

Sequential Pattern Mining is an important research direction in the area of data mining. It has been introduced by Agrawal and Srikant [1] to extract the frequent sequences among the given set of customer sequences. A customer sequence is defined as an ordered collection of transactions that belongs to the same customer. The process of finding the frequent sequences is useful in many applications such as analyzing the customer buying habits, predicting the events such as web usage patterns and natural disasters, identification of disease symptoms, DNA sequence analysis etc. The traditional sequential pattern mining algorithms [9, 19] are not scalable for big data [5]. Hence, there is a need to redesign the existing sequential pattern mining algorithms using a distributed framework that best suits the big data.

✉ Sumalatha Saleti
katam.suma@gmail.com

R. B. V. Subramanyam
rbvs66@gmail.com

[1] National Institute of Technology, Warangal, India

MapReduce is a popular distributed framework developed by Google [7] to distribute the data over multiple machines in the cluster. A programmer has to define the map and reduce functions to process the data and the runtime environment of MapReduce is responsible for the overall execution of the program. Recently, few researchers have developed sequential pattern mining algorithms using MapReduce [5, 15, 16, 31, 32]. However, they adopt an iterative MapReduce framework that leads to multiple MapReduce rounds. A long frequent sequence contains a huge number of candidate subsequences, and these candidate subsequences must be generated and tested for support count. It leads to the evaluation of patterns that do not occur in the input sequences.

Redesigning the traditional approaches to suit the distributed frameworks such as MapReduce is of recent research interest. Distributed Progressive Sequential Pattern mining (DPSP) [16] algorithm is one such initiative to provide solution to the SPM problem for progressive databases using the MapReduce framework on a cloud computing environment. Later, Sequential PAttern Mining algorithm based on the MapReduce model on the Cloud (SPAMC) [5] is proposed and it is a parallel version of SPAM [3] algorithm. Recently, Distributed Sequential

Pattern using the Dynamic Bit Vector (DSPDBV) [15] is proposed and it uses a dynamic bit vector data structure to convert the given input sequences into a vertical representation. DPSP, SPAMC and DSPDBV are the three sequential pattern mining algorithms based on MapReduce and suitable for mining big data. However, these are iterative MapReduce algorithms that employ multiple MapReduce rounds and involve high communication and scheduling overhead between Mappers and Reducers.

Recently, Chen et al. [4] proposed sequential pattern mining in the cloud-uniform distributed lexical sequence tree algorithm (SPAMC-UDLT). The novelty is that they avoided the overhead of reloading the data by using the two phase MapReduce model. However, SPAMC-UDLT is derived from SPAM [3] and it uses a bitmap representation of the items and proved to be prominent only when the number of distinct items is smaller. Also, the authors of SPAMC-UDLT have not dealt with the issue of reducing the number of candidate sequences, early prune strategies and handling long sequences. To address these issues, we proposed a two phase MapReduce algorithm that efficiently generates the candidate sequences using co-occurrence information and effectively prunes the candidate sequences prior to support count. The objectives of this paper include:

1. Designing an efficient MapReduce algorithm that avoids iterative MapReduce model, thereby reducing the communication and scheduling overhead between mappers and reducers.
2. Developing an efficient mechanism to handle possible large search space and handling early prune of infrequent sequences in a distributed environment.
3. Proposing the data structure that helps in computing the support count of a sequence in an efficient way and avoiding multiple scans of the data.
4. Providing empirical analysis and comparison with the recent approaches of mining sequential patterns based on the MapReduce framework.

To address the above issues, a two phase MapReduce algorithm has been proposed based on the Sequence Index List (SIL) and Co-occurrence MAP ($CMAP$) data structures. The Sequence Index List data structure is derived from the sparse id-list data structure proposed in [23]. However, the size of the sparse id-list is fixed to the number of sequences in the given dataset. Sparse id-list of each item stores a list of transaction ids of the sequence. If a sequence does not contain the item, then the corresponding row is assigned a null value. Thus, the sparse-id list may waste large amounts of memory for the items which do not occur in a sequence. In the proposed SIL data structure, its size is not fixed and it depends on the number of input sequences in which a frequent sequence is present. To handle the pruning of infrequent candidates before support counting, $CMAP$ data structure introduced in [8] has been used in the proposed algorithm. $CMAP$ is a simple data structure that maintains the item co-occurrence information.

Compared with the recent works in distributed mining of sequential patterns using MapReduce, the contributions of the paper include:

1. Two phase MapReduce algorithm is proposed for distributed mining of sequential patterns in a cloud computing environment. It reduces the problem of communication and scheduling overheads of iterative MapReduce algorithms. In this two phase algorithm, we show how SIL and $CMAP$ data structures can be integrated to find the frequent sequences in an efficient manner.
2. To reduce the large search space, we adapted the $CMAP$ data structure for novel candidate generation and early prune of the infrequent sequences. Two algorithms, $CREATE\_CMAP_i$ and $CREATE\_CMAP_s$ are proposed to map the given input sequences into a co-occurrence map in a distributed way. The main purpose of this design is to transform the serial construction of $CMAP$ to distributed construction, so that multiple mappers can construct the $CMAP$ in parallel on multiple machines simultaneously. Therefore, each machine in the second phase can access the required co-occurrence information and early prune the infrequent candidates.
3. Since support counting is more costly in finding the frequent sequences, a memory efficient Sequence Index List data structure and two novel algorithms, $CREATE\_SIL$ and $CREATE\_SIL\_SEQUENCE$ are proposed for support counting. These algorithms, converts each candidate into an SIL format and the number of rows in SIL is the support of the candidate. The main advantage of this approach is, it avoids multiple scans of the input sequences while support counting.
4. The proposed algorithm is tested on both the synthetic and real datasets. Its performance is compared with the recent MapReduce algorithms SPAMC-UDLT [4], DSPDBV [15], SPAMC [5] and DPSP [16].

The rest of the paper is organized as follows. Section 2 describes the related work. In Section 3, the basic concepts to understand the problem of sequential pattern mining are presented. In Section 4, the data structures SIL and $CMAP$ are discussed. The candidate generation and pruning methods are also discussed. The proposed DSPC algorithm is also presented in Section 4. The experimental results are presented in Section 5. Finally, the conclusions are given in Section 6.

2 Related work

The solutions for the problem of sequential pattern mining fall into two categories: 1. Candidate generation and test algorithms [1–3, 20, 25, 29, 34] 2. Pattern growth algorithms [6, 13, 22, 30]. AprioriAll [1] and its improvement, namely, GSP [25] are the two well-known Apriori-based sequential pattern mining algorithms that employ Breadth-First search approach. In a Breadth-First search approach, sequential patterns are generated based on their length, i.e. sequences of length n are generated from the sequences of length $n-1$. These algorithms repeatedly scans the database for support counting. The reason for the repeated scan is due to the horizontal representation of the database. PSP algorithm proposed in [20] resumes the principles of GSP and proposed a prefix tree structure to speed up the mining process.

SPAM [3] and SPADE [34] are the improvements of apriori based approaches. They convert the horizontal representation of the database to a vertical format by scanning the database only once. The vertical format allows to calculate the support without performing multiple database scans. However, vertical algorithms follow candidate generation and test approach that can generate more number of infrequent candidates that do not occur in the input database. To address this problem, candidate pruning based on co-occurrence information is proposed in [8]. As proposed in [8], the three algorithms CM-SPAM, CM-SPADE and CM-CLASP use the common pruning strategy named Co-occurrence-Based Pruning. All the three algorithms are an extension to three state-of-the-art algorithms SPAM [3], SPADE [34] and ClaSP [11].

Pattern growth algorithms such as FreeSpan [13], PrefixSpan [22] and UDDAG [6] avoid the problem of generating the candidates that do not exist. They project the database into smaller databases based on the prefix of a pattern and scans the projected databases to find the support count of the pattern. However, the efficiency of pattern growth algorithms is reduced due to the repeated construction and scanning of the projected databases. Recently, a number of variations of sequential pattern mining problem, closed sequential pattern mining [10], mining top-k co-occurrence items with sequential pattern [18], mining high utility sequential patterns [26] have been proposed. However, all the above mentioned algorithms are the serial implementations of the sequential pattern mining.

To make the mining process more efficient and scalable, several parallel algorithms [12, 17, 24, 33] have been proposed in the literature. In [12], the authors proposed a tree-projection algorithm for finding the sequential patterns. The authors parallelized the algorithm on a distributed-memory parallel computing architecture. Parallel algorithms for mining sequential patterns in a shared nothing environment are proposed in [24]. Zaki proposed pSPADE [33], a parallel algorithm that decomposes the search space into suffix-based classes. Each class is solved independently by each processor on a shared memory system. Recently, Bao et al. [17] proposed Parallel Dynamic Bit Vector Sequential Pattern Mining (pDBV-SPM) for mining frequent sequential patterns on a multi-core processor architecture.

The existing parallel algorithms, demands the user to have the knowledge of data partitioning, workload balancing, job scheduling and fault tolerance. In [7], an abstract programming model known as MapReduce [28] has been proposed by Google to overcome the aforementioned issues. Many MapReduce based algorithms [4, 5, 15, 16, 21, 27, 31, 32] have been designed that suits the distributed mining of big data. DPSP [16] is the first MapReduce based algorithm to mine sequential patterns. It divides the input data into multiple progressive windows and generates the candidates within the current period of interest. Later, it assembles the support count of all the candidates and reports only the frequent sequential patterns. DPSP repeats this process of candidate computing and support assembling jobs for the next time stamp.

SPAMC [5] is an iterative MapReduce algorithm, it parallelized the SPAM [3] algorithm proposed by Ayres et al. In SPAMC, there exist two phases, namely, scanning phase and mining phase. A single MapReduce round is designed in the scanning phase to find the frequent items. Also, the input database is converted into a vertical bitmap format and it is stored in a distributed hash table. The mining phase constructs a lexical sequence tree at the mapper, each node in the tree represents a pattern. After the construction of the local sub-tree, at each node of the sub-tree, bit-AND operation is computed to find the support count of the pattern. Finally, the reducer combines the intermediate result from the mappers and computes the actual support count of the pattern. DPSP and SPAMC involves multiple rounds of MapReduce jobs thereby increasing the communication and scheduling overhead. Recently, SPAMC-UDLT [4] have been proposed as an alternative to the iterative MapReduce approach and improved the cloud based sequential pattern mining algorithm proposed in [5]. However, these works generate the candidates that do not exist in the input sequences and incurs more time in the support counting phase. To solve this issue, we adapt the CMAP data structure to generate the candidates that actually occur in the input sequences.

DGSP [32] and MR-PrefixSpan [31] algorithms represent the data in a horizontal format. DGSP is a MapReduce version of the GSP algorithm. It follows the "two-jobs" structure and reduces the communication and scheduling overheads. MR-PrefixSpan is proposed as an improvement to the PrefixSpan [22] algorithm. Each map task corresponding to a prefix sequence, recursively constructs and scans the projected databases before generating the complete set of frequent sequences. In [27], transactions are divided into smaller partitions and PrefixSpan algorithm is applied to each partition using the MapReduce framework. MG-FSM [21] makes use of FP-growth [14] and constructs the projected database using MapReduce. Due to the horizontal representation of the data, these works need to scan the database multiple times to find the frequent sequences. In contrast, the proposed algorithm follows the vertical database format by creating the SIL data structure and provides the advantage of calculating the support counts without multiple database scans. Moreover, the existing MapReduce solutions have not dealt with the issue of handling long sequences and they are not able to prune the infrequent candidates before support counting. In this paper, a novel MapReduce algorithm has been proposed to generate the candidates efficiently based on the item co-occurrence information and prunes them before support counting.

3 Preliminaries

This section presents the overview of MapReduce model and the formal definition of sequential pattern mining.

3.1 MapReduce model

MapReduce is a distributed programming model developed by the Google [7]. It tackles the problem of Big Data by following the divide and conquer strategy. It hides the system level details such as the decomposition of a problem into smaller sub-problems, assigning the data and tasks to workers in a distributed way, achieving the synchronization between the workers and sharing the partial results among the workers. The programmer can focus on the algorithmic details of the problem instead of the above mentioned system details. MapReduce programming consists of two stages, namely, map and reduce. The $\langle Key, Value\rangle$ pair is the basic data structure of MapReduce programming. The programmer has to define the map and reduce methods with input and output $\langle Key, Value\rangle$ pairs. Depending on the problem, the programmer has to define the type of key and value. The map and reduce methods together is called as a MapReduce job. The objects that implement the map and reduce methods are called mappers and reducers respectively. The map method processes an input $\langle Key, Value\rangle$ pair depending on the computation specified by the programmer. The output from a mapper is aggregated and sorted by the implicit shuffle and sort stage to form intermediate $\langle Key, Value\rangle$ pairs. The list of values associated with the same key arrives at the same reducer. The reduce method finally generates the output $\langle Key, Value\rangle$ pairs and writes them to a file on HDFS (Hadoop Distributed File System). HDFS is a distributed file system that provides scalable and reliable data storage in a MapReduce environment. Figure 1 illustrates the MapReduce model.

3.2 Problem definition

Definition 1 Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of items and $s \subseteq I$ be an itemset denoted as $(x_1 x_2 \ldots x_l)$, where x_j is an item, $1 \leq j \leq l$. The length of an itemset is l if it contains l items and it is denoted as $|s| = l$. Let $SD = \{S_1, S_2, \ldots, S_n\}$ be a set of sequences stored in the Hadoop Distributed File System, where each sequence $S_i = \langle s_1 s_2 \ldots s_k\rangle$ and $1 \leq i \leq k$ is an ordered list of itemsets.

Definition 2 A sequence $S_a = \langle a_1 a_2 \ldots a_n\rangle$ is said to be contained in another sequence $S_b = \langle b_1 b_2 \ldots b_m\rangle$, if there exists integers $1 \leq i_1 < i_2 < \cdots < i_n \leq m$ such that $a_1 \subseteq b_{i1}, a_2 \subseteq b_{i2}, \ldots, a_n \subseteq b_{in}$. If a sequence S_a is

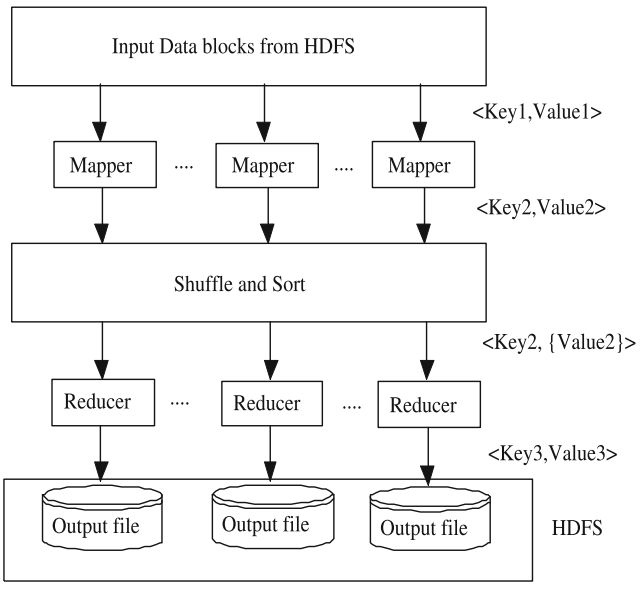


Fig. 1 MapReduce model

contained in another sequence S_b, then S_a is said to be a subsequence of S_b and S_b is said to be a supersequence of S_a.

For example, the sequence $S_a = \langle(1\ 2)(4)\rangle$ is a subsequence of $S_b = \langle(1\ 2\ 3)(3\ 4)\rangle$ and S_b is a supersequence of S_a.

Definition 3 Given a sequence $S = \langle s_1 s_2 \ldots s_k\rangle$, the length of the sequence S is l if it contains l items, i.e., $|S| = |s_1| + |s_2| + \cdots + |s_k|$. A sequence of length l is called a l-sequence.

For example, length of the sequence $\langle(1\ 2\ 3)(3\ 4)(3\ 8)\rangle$ is 7.

Definition 4 The support of a sequence S is the number of sequences that contain S in SD and it is denoted by $sup(S)$.

For example, consider the sample dataset given in Table 1, support of a sequence $\langle(1\ 2)(3\ 4)\rangle$ is 2 as it is contained in the sequences 1 and 2.

Definition 5 Given a sequence dataset SD and a positive integer min_sup as the support threshold, a sequence S is called a sequential pattern or a frequent sequence if $sup(S) \geq min\_sup$. A sequential pattern is called a l-pattern if its length is l.

For example, consider a sequence $S = \langle(1\ 2)(3)\rangle$ and $min\_sup = 2$, S is a subsequence of the sequences $\langle(1\ 2\ 3)(3\ 4)(3\ 8)\rangle$ and $\langle(1\ 2)(1\ 4\ 6)(3\ 4\ 7)\rangle$. Therefore, $sup(S) = 2$ and it satisfies the min_sup. Hence, S is a sequential pattern of length 3 i.e. 3-pattern.

Problem statement Let SD be a set of sequences stored in the Hadoop Distributed File System and min_sup be the user given minimum support threshold, the problem of sequential pattern mining is to find all the sequential patterns in SD.

Table 1 Sample dataset

Sequence Id	Sequence
1	$\langle(1\ 2\ 3)(3\ 4)(3\ 8)\rangle$
2	$\langle(1\ 2)(1\ 4\ 6)(3\ 4\ 7)\rangle$
3	$\langle(3\ 5\ 7)(8)\rangle$
4	$\langle(3)(1\ 4\ 7)\rangle$
5	$\langle(7)(8)\rangle$

4 Distributed sequential pattern mining using the co-occurrence information

In this section, we introduce the proposed algorithm for Distributed mining of Sequential Patterns using Co-occurrence information (DSPC). This section is divided into three subsections. The first subsection describes the terminology used in the DSPC algorithm. The second subsection introduces the first phase of the algorithm, which includes construction of the $CMAP$ data structure. The third subsection introduces the second phase of the algorithm, it includes SIL creation, support count calculation, candidate generation and early prune properties.

4.1 Terminology used in the DSPC algorithm

Definition 6 A sequence is said to be an itemset extension if it is generated by adding an item to the last itemset of the given sequence. The itemset extension of $S = \langle s_1 s_2 ... s_k\rangle$ with an item x is denoted as $\langle s_1 s_2 ... \{s_k \cup x\}\rangle$.

For instance, the sequence $S = \langle(1\ 2\ 3)(3\ 4)\rangle$ is generated by adding an item 4 to the last itemset of $\langle(1\ 2\ 3)(3)\rangle$. Hence, S is an itemset extension.

Definition 7 A sequence is said to be a sequence extension if it is generated by adding a sequence of length 1 to its end. The sequence extension of $S = \langle s_1 s_2 ... s_k\rangle$ with an item x is denoted as $\langle s_1 s_2 ... s_k x\rangle$.

For instance, the sequence $S = \langle(1\ 2\ 3)(3\ 4)(3)\rangle$ is generated by adding a sequence $\langle(3)\rangle$ to the end of $\langle(1\ 2\ 3)(3\ 4)\rangle$. Hence, S is a sequence extension.

Definition 8 Given a sequence $S = \langle s_1 s_2 ... s_l\rangle$, an item j is said to succeed by itemset extension to an item k, if $j \in s_i$ and $k \in s_i$, $1 \leq i \leq l$, and j is located after k in S_i.

For instance, consider a sequence $S = \langle(1\ 2\ 3)(3\ 4)\rangle$, item 3 succeed to items 1 and 2, whereas, item 2 succeed to item 1. Similarly, item 4 succeed to item 3 by an itemset extension.

Definition 9 Given a sequence $S = \langle s_1 s_2 ... s_l\rangle$, an item j is said to succeed by sequence extension to an item k, if $j \in S_x$ and $k \in S_y$, $1 \leq x < y \leq l$.

For instance, consider a sequence $S = \langle(1\ 2\ 3)(3\ 4)\rangle$, items 3 and 4 succeed to the items 1, 2 and 3 by a sequence extension.

Definition 10 Given a sequence dataset SD, the Co-occurrence List of an item j that is succeeded by an itemset extension is defined as a list of items that succeed to j by an itemset extension in no less than min_sup sequences of SD, denoted as $CL_i(j)$.

For example, consider the sample dataset given in Table 1 and let $min\_sup = 2$, item 2 succeed to item 1 in the sequences 1 and 2 by an itemset extension. Similarly, item 4 succeed to item 1 in the sequences 2 and 4. Hence, $CL_i(1) = \{2, 4\}$.

Definition 11 Given a sequence dataset SD, the Co-occurrence list of an item j that is succeeded by a sequence extension is defined as a list of items that succeeds j by a sequence extension in no less than min_sup sequences of SD, denoted as $CL_s(j)$.

For example, consider the sample dataset given in Table 1 and let $min\_sup = 2$, items 3 and 4 succeed to item 1 in the sequences 1 and 2 by a sequence extension. Hence, $CL_s(1) = \{3, 4\}$.

Even though an item succeed to an item by itemset/sequence extension more than once in the same sequence, it is counted once.

Definition 12 Given a sequence dataset SD, the Co-occurrence Map of SD with respect to itemset extension is defined as the mapping of each item j to its Co-occurrence list $CL_i(j)$, denoted as $CMAP_i$, where $CL_i(j) \neq \phi$.

For instance, from Definition 10, $CL_i(1) = \{2, 4\}, CL_i(3) = \{4, 7\}, CL_i(4) = \{7\}$ and for the remaining items, CL_i is empty. Hence, $CMAP_i$ of the given sample dataset is the mapping of items $1, 3$ and 4 to their corresponding Co-occurrence lists and it is shown in Table 2.

Definition 13 Given a sequence dataset SD, the Co-occurrence Map of SD with respect to sequence extension is defined as the mapping of each item j to its Co-occurrence list $CL_s(j)$, denoted as $CMAP_s$, where $CL_s(j) \neq \phi$.

For instance, from Definition 11, $CL_s(1) = \{3, 4\}, CL_s(2) = \{3, 4\}, CL_s(3) = \{4, 8\}, CL_s(4) = \{3\}, CL_s(7) = \{8\}$ and for the remaining items, $CL_s = \phi$. Hence, $CMAP_s$ of the given sample dataset is the mapping of items $1, 2, 3, 4$ and 7 to their corresponding Co-occurrence lists and it is shown in Table 3.

Lemma 1 *An itemset extensible sequence of length 2, $\langle(x\ y)\rangle$, is identified as frequent if and only if $y \in CMAP_i(x)$.*

Proof Based on Definition 10 and Definition 12, if an item y does not exist in $CL_i(x)$, then y succeed to x by an itemset extension in less than min_sup sequences and y cannot be mapped to $CMAP_i(x)$. Hence $\langle(x\ y)\rangle$ cannot be frequent. □

Lemma 2 *A sequence extensible sequence of length 2, $\langle(x)(y)\rangle$, is identified as frequent if and only if $y \in CMAP_s(x)$.*

Proof Based on Definition 11 and Definition 13, if an item y does not exist in $CL_s(x)$, then y succeed to x by a sequence extension in less than min_sup sequences and y cannot be mapped to $CMAP_s(x)$. Hence $\langle(x)(y)\rangle$ cannot be frequent. □

DSPC algorithm is divided into two MapReduce phases. The first phase involves the creation of $CMAP$ data structure by scanning the dataset once. Each mapper in the first phase reads the input split of sequences and finds the items that co-occur with each item by an itemset extension (CL_i of item) and a sequence extension (CL_s of item). Finally, it emits the three $\langle Key, Value\rangle$ pairs.

1. $\langle item, CL_i(item)\rangle$
2. $\langle item, CL_s(item)\rangle$
3. $\langle Sequenceid, Sequence\rangle$

The reducer of the first phase counts the item co-occurrences and emits only the $CMAP$ of items that satisfy the minimum support threshold. During the

Table 2 $CMAP_i$

Item	CL_i
1	{2,4}
3	{4,7}
4	{7}

Table 3 $CMAP_s$

Item	CL_s
1	{3,4}
2	{3,4}
3	{4,8}
4	{3}
7	{8}

computation at reducer side, infrequent items are eliminated. Finally, reducer emits the frequent 1-sequence and its co-occurrences as output. The $CMAP$ thus created is stored in a distributed cache and it is accessed by the second MapReduce phase. In addition to the aggregation of $CMAP$ received from the mapper, it also processes a sequence by noting the position of items and emits the $\langle Sequenceid, \langle item, position \rangle \rangle$ as an output $\langle Key, Value \rangle$ pair.

The second phase makes use of the $CMAP$ to efficiently generate the candidate sequences and creates their SIL to implement the early prune of the candidates before support counting. In the next section, we describe the steps involved in the first phase. Then, we introduce candidate generation and early prune properties in Sections 4.3.3 and 4.3.4. The overall workflow of DSPC is shown in Fig. 2.

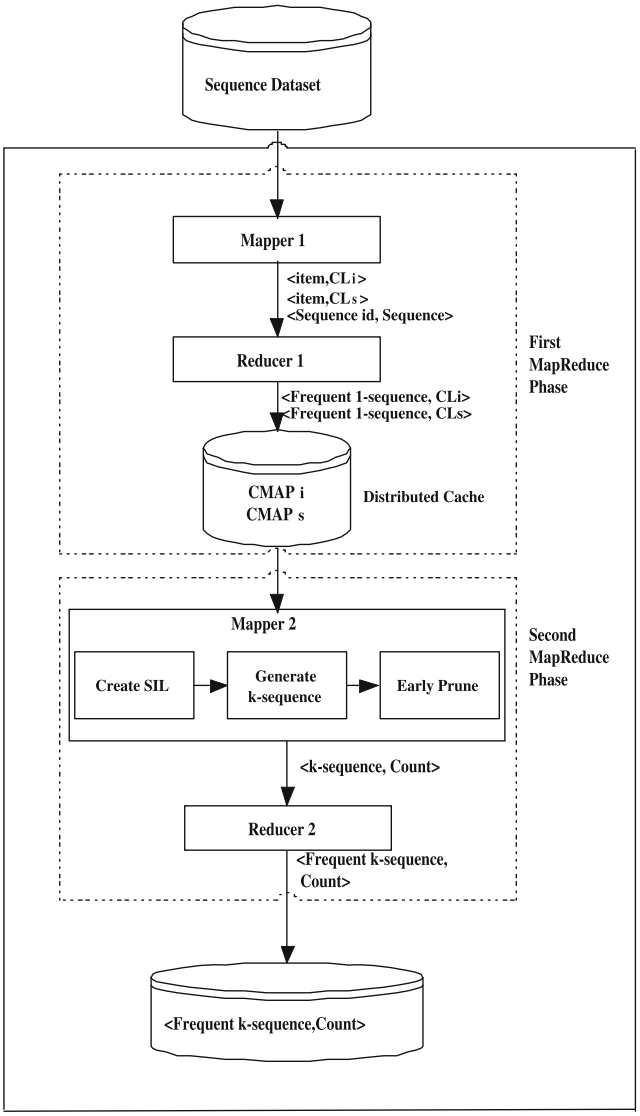


Fig. 2 Workflow of DSPC

Algorithm 1 First MapReduce Phase

Input:
SD_i ▷ Sequence dataset of i^{th} split
min_sup ▷ Minimum support
Output:
$CMAP$ ▷ Co-occurrence Map

Mapper

1: Create Multimap datastructures $CMAP_i$ and $CMAP_s$
2: **function** MAP(key *offset*, values S)
3: $CMAP_i \leftarrow CREATE\_CMAP_i(S)$
4: $CMAP_s \leftarrow CREATE\_CMAP_s(S)$
5: output(*S.sid, S.value*)
6: **end function**
7: **function** CLOSE
8: **for** each $item_i \in CMAP_i$ **do**
9: output($item_i$, $CMAP_i.get(item_i)$)
10: **end for**
11: **for** each $item_s \in CMAP_s$ **do**
12: output($item_s$, $CMAP_s.get(item_s)$)
13: **end for**
14: **end function**

Reducer

15: **function** REDUCE(key S, values $values$)
16: **for** each $value$ in $values$ **do**
17: **if** S is a sequence-id **then**
18: **for** each $item$ in the sequence **do**
19: output($sequence\text{-}id$,pair($item$, pos))
20: **end for**
21: **else if** S is $item_i$ **then**
22: **for** each co-occurrence item $item_c$ in $value$ **do**
23: $CMAP_i = CMAP_i \cup item_c$
24: Increment the count of $item_c$ by 1
25: **end for**
26: **else if** S is $item_s$ **then**
27: **for** each co-occurrence item $item_c$ in $value$ **do**
28: $CMAP_s = CMAP_s \cup item_c$
29: Increment the count of $item_c$ by 1
30: **end for**
31: **end if**
32: **end for**
33: **for** each co-occurrence item $item_c$ in $CMAP_i$ **do**
34: **if** count of $item_c \geq min\_sup$ **then**
35: output(S, $item_c$)
36: **end if**
37: **end for**
38: **for** each co-occurrence item $item_c$ in $CMAP_s$ **do**
39: **if** count of $item_c \geq min\_sup$ **then**
40: output(S, $item_c$)
41: **end if**
42: **end for**
43: **end function**

Algorithm 2 Create $CMAP_i$

Input:
$CurrentSeq$ ▷ Input sequence
Output:
$CMAP_i$ ▷ List of items that succeeds an item by itemset extension
1: **function** $CREATE\_CMAP_i(CurrentSeq)$
2: **for** each $itemset$ in $CurrentSeq$ **do**
3: **for** $i = 0$ to the size of $itemset$ **do**
4: **for** $j = (i + 1)$ to the size of $itemset$ **do**
5: Let $item_i$ be the i^{th} item in the $itemset$ and $item_j$ be the j^{th} item in the $itemset$
6: **if** $item_i \in CMAP_i$ **then**
7: $CL(item_i) \leftarrow CMAP_i(item_i)$
8: **if** $item_j \in CL(item_i)$ **then**
9: Increment the count of $item_j$ in $CL(item_i)$
10: **else**
11: $CL(item_i) \leftarrow CL(item_i) \cup \{item_j\}$
12: **end if**
13: **else**
14: $CL(item_i) \leftarrow \{item_j\}$
15: **end if**
16: $CMAP_i(item_i) \leftarrow CL(item_i)$
17: **end for**
18: **end for**
19: **end for**
20: **end function**

Algorithm 3 Create $CMAP_s$

Input:
$CurrentSeq$ ▷ Input sequence
Output:
$CMAP_s$ ▷ List of items that succeeds an item by sequence extension
1: **function** $CREATE\_CMAP_s(CurrentSeq)$
2: **for** $i = 0$ to the number of itemsets in the $CurrentSeq$ **do**
3: **for** $j = 0$ to the size of $itemset_i$ **do**
4: **for** $k = 0$ to the size of $itemset_{i+1}$ **do**
5: **if** $item_j \in CMAP_s$ **then**
6: $CL(item_j) \leftarrow CMAP_s(item_j)$
7: **if** $item_k \in CL(item_j)$ and $item_k$ is the first occurrence in $CL(item_j)$ for the $CurrentSeq$ **then**
8: Increment the count of $item_k$ in $CL(item_j)$
9: **else if** $item_k \notin CL(item_j)$ **then**
10: $CL(item_j) \leftarrow CL(item_j) \cup \{item_k\}$
11: **end if**
12: **else**
13: $CL(item_j) \leftarrow \{item_k\}$
14: **end if**
15: $CMAP_s(item_j) \leftarrow CL(item_j)$
16: **end for**
17: **end for**
18: **end for**
19: **end function**

4.2 The first MapReduce phase

4.2.1 Constructing Co-occurrence MAP

The first phase of DSPC algorithm is described in Algorithm 1. Firstly, a mapper reads the input sequence and finds the co-occurrence information, namely, $CMAP_i$ (line 3) and $CMAP_s$ (line 4). The mapper outputs the item and its local $CMAP_i$, $CMAP_s$ (line 9 and line 12). Depending on the key, reducer accumulates the support count of $CMAP_i$ (lines 22-25) and $CMAP_s$ (lines 27-30). The reducer outputs the co-occurrence items of each entry in $CMAP_i$ and $CMAP_s$ whose support count is no less than the min_sup (lines 33-42).

Based on Definition 12, $CMAP_i$ is a mapping of each $item_i \in I$ to its Co-occurrence List i.e. $CL(item_i)$ succeeded by an itemset extension, where $CL(item_i)$ is non empty. For convenience, we used the notation CL to represent the Co-occurrence List in both Algorithm 2 and Algorithm 3. The detailed algorithm to construct $CMAP_i$ is shown in Algorithm 2. If $CMAP_i$ contains an $item_i$ mapping to its Co-occurrence List (line 6 of Algorithm 2), then we check whether the $item_j$ is an element in $CL(item_i)$ (line 8). If it is found that $item_j$ already exists in $CL(item_i)$, then its count is incremented by 1 (line 9). Otherwise, $item_j$ is included in $CL(item_i)$ (line 11). The $item_i$ is included into $CMAP_i$ for the first time by creating its Co-occurrence List with the $item_j$ (line 14).

Based on Definition 13, $CMAP_s$ is a mapping of each $item_i \in I$ to its Co-occurrence List i.e. $CL(item_i)$ of items succeeded by a sequence extension, where $CL(item_i)$ is non empty. The detailed algorithm to construct the $CMAP_s$ is shown in Algorithm 3. If $CMAP_s$ contains an $item_j$ mapping to its Co-occurrence List (line 5 of Algorithm 3), then we check whether $item_k$ occurs in $CL(item_j)$ for the first time in the $CurrentSeq$ (line 7) and its count is incremented by 1 (line 8). Otherwise, $item_k$ is included in $CL(item_j)$ (line 10). The $item_j$ is included into $CMAP_s$ for the first time by creating its Co-occurrence List with $item_k$ (line 13).

Example Consider the sample dataset in Table 1 and $min\_sup = 2$. For sequence 1 i.e. $S_1 = \langle(1\ 2\ 3)(3\ 4)(3\ 8)\rangle$, first we check the successors of item 1 within the first itemset (1 2 3). The successors of 1 are found to be 2 and 3 and the successor of 2 is 3. Now, we move to the next itemset in the same sequence which is (3 4). Successor of 3 is 4. Similarly, successor of 3 is 8 in the last itemset (3 8). From sequence 1, it is found that the successors of 1 are 2 and 3, the successor of 2 is 3 and the successors of 3 are 4 and 8. This procedure is repeated for all the input sequences. Finally, only the list of items that succeed to an item by itemset extension in no less than min_sup sequences are remained in $CMAP_i$ and it is as shown in Table 2.

Similarly, for sequence 1, first we check the successors of item 1 based on a sequence extension as described in Definition 9. The successors of 1 are found to be 3 and 4 from itemset (3 4), 3 and 8 from itemset (3 8). The successors of 2 are 3 and 4 from itemset (3 4), 3 and 8 from itemset (3 8). Similarly, the successors of 3 are 3 and 4 from itemset (3 4), 3 and 8 from itemset (3 8). The successors of 4 are 3, 8 from itemset (3 8). This procedure is repeated for all the input sequences. Finally, only the list of items that succeed to an item by a sequence extension in no less than min_sup sequences are remained in $CMAP_s$ and it is as shown in Table 3.

4.3 The second MapReduce phase

The second phase of DSPC algorithm is described in Algorithm 4. The main steps involved in the second phase are creating the SIL of frequent items (Algorithm 5), generating the candidate sequences (Algorithm 7) and their SIL (Algorithm 6), and early prune of the infrequent candidates (Algorithm 8).

Algorithm 4 Second MapReduce Phase

Input:
Output from first MapReduce phase
min_sup ▷ Minimum support
Output:
S_n ▷ Frequent sequence
$count$ ▷ Support of a frequent sequence

1: **function** MAP(key *offset*, values S)
2: **for** each $S.sid$ in its input split **do**
3: $CREATE\_SIL(S.sid, S.value)$ ▷ $S.value$ is the pairs(frequent item,position)
4: **end for**
5: **for** each length-1 sequence S_1 in SIL **do**
6: $GENERATE\_SEQUENCE(S_1, SIL(S_1))$
7: **end for**
8: **end function**
9: **function** REDUCE(key S, values $local\_count$)
10: Intitalize global_count to 0
11: **for** each c in $local\_count$ **do**
12: $global\_count = global\_count + c$
13: **end for**
14: **if** $global\_count \geq min\_sup$ **then**
15: write(S, $global\_count$)
16: **end if**
17: **end function**

4.3.1 Sequence index list

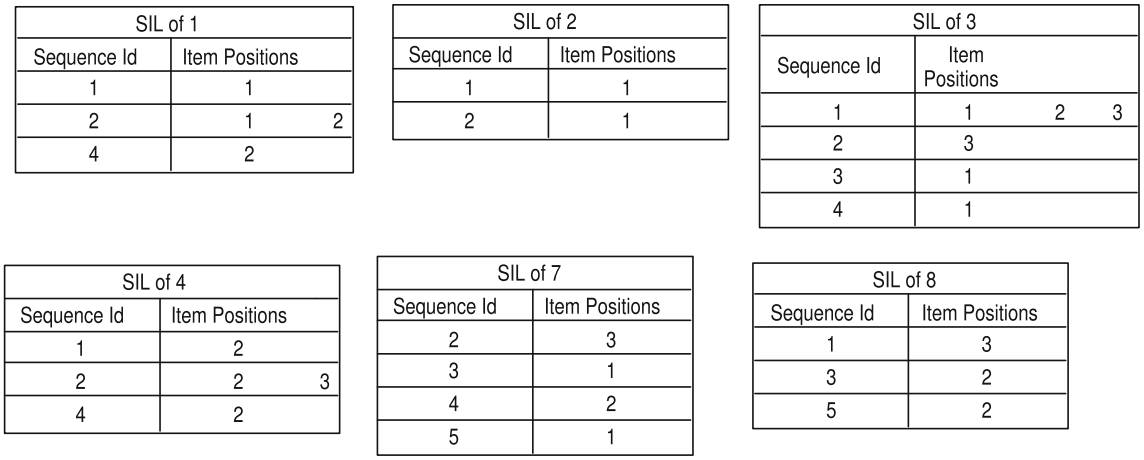
Definition 14 The Sequence Index List (SIL) of a sequence $S(|S| \geq 1)$ is defined as a mapping of input sequence α_i to a list of positions of S in α_i, such that S is a subsequence of α_i, where $i \leq n$ and n is the number of input sequences. SIL is implemented using a list of lists, where the size of the SIL is not fixed and it is equal to the number of sequences in which S occurs. Each row in the SIL is a list which stores the position of S in α_i. The number of rows depends on the number of sequences in which S is a subsequence of α_i and it is called support count of S. Moreover, a subsequence can occur multiple times in the same supersequence. Hence, the number of columns in SIL is not fixed and it depends on the number of occurrences of S in α_i. For example, given the sample dataset as shown in Table 1, the sequence $S = \langle(1)\rangle$ is a subsequence of the sequences 1, 2 and 4. The position of S in sequence 1 is 1. In sequence 2, S occurs two times in positions 1 and 2. Similarly, in sequence 4, S occurs at position 2. The SIL of the sequences $\langle(1)\rangle$, $\langle(2)\rangle$, $\langle(3)\rangle$, $\langle(4)\rangle$, $\langle(7)\rangle$ and $\langle(8)\rangle$ is shown in Fig. 3. In this paper, we used the following notation to represent a SIL: $\langle\alpha_a p_l p_m ... - \alpha_b p_x p_y ... - ...\rangle$, where α_a and α_b are the sequence ids, p_l, p_m, p_x and p_y are the positions such that $1 \leq \alpha_a \leq \alpha_b ... \leq n$, $1 \leq p_l \leq p_m ... \leq |\alpha_a|$ and $1 \leq p_x \leq p_y ... \leq |\alpha_b|$, where n is the number of input sequences. The SIL representation of 1-patterns is shown in Table 4.

Each mapper in the second phase receives the sequence id and the pairs(frequent item,position) as input from the first phase. Firstly, it creates the SIL of frequent items (line 3 of Algorithm 4) and this information is used to create the SIL of k-sequences, where $k \geq 2$. Later, it invokes the candidate generation procedure $GENERATE\_SEQUENCE$ (line 6) for each frequent 1-sequence. This is a recursive procedure which emits the $\langle Key, Value\rangle$ pair $\langle Sequence, Count\rangle$ as an output. The second phase reducer is responsible for finding the global count of each sequence (line 12) received from the mapper and emits only the sequences whose support satisfies the minimum support threshold (line 14).

Algorithm 5 Create SIL of an item

Input:
$SequenceID$
pairs($item$, $position$)
Output:
SIL

1: **function** $CREATE\_SIL(SequenceID$,pairs ($item$, $position$))
2: create a multi hash data structure H to store item and its position in the $CurrentSeq$
3: **for** each $item$ in pairs($item$, $position$) **do**
4: $H \leftarrow \langle item, position\rangle$
5: **end for**
6: **for** each $item$ in H **do**
7: $itemset\_pos\_list \leftarrow H$.get($item$)
8: $SIL \leftarrow \{item, \langle SequenceID, itemset\_pos\_list\rangle\}$
9: **end for**
10: **end function**



SIL of 1

Sequence Id	Item Positions
1	1
2	1 2
4	2

SIL of 2

Sequence Id	Item Positions
1	1
2	1

SIL of 3

Sequence Id	Item Positions
1	1 2 3
2	3
3	1
4	1

SIL of 4

Sequence Id	Item Positions
1	2
2	2 3
4	2

SIL of 7

Sequence Id	Item Positions
2	3
3	1
4	2
5	1

SIL of 8

Sequence Id	Item Positions
1	3
3	2
5	2

Fig. 3 SIL of 1-patterns found in the sample dataset of Table 1

Algorithm 5 describes the process of creating the SIL of frequent items in the sequence dataset. According to Definition 14, the SIL of an item i is computed by checking the position of i in each input sequence (lines 3-5 of Algorithm 5). However, an item may occur multiple times in a sequence, so each item is mapped to its position in the input sequence (lines 6-8). If an item does not occur in a sequence, then the sequence entry is not included in $SIL(i)$.

4.3.2 Creating the *SIL* of a sequence and support counting

Algorithm 6 describes the procedure of creating the SIL of a sequence. Given X i.e. SIL of a sequence to be extended and Y i.e. SIL of an item that extends X, Algorithm 6 creates SIL of a new sequence. First, it starts comparing the sequence ids of X and Y (line 6). Once the $sids$ are matched, then it starts comparing the positions. Even though, a sequence occurs at multiple positions in the same input sequence, it is counted only once. If the new sequence is itemset extensible, then it finds the matched positions of X and Y (lines 11-18). Similarly, if the new sequence is sequence extensible, then it finds the positions where the position of Y is greater than the position of X (line 20-27). The matched sequence id is mapped to the newly found position list and the sequence entry is included in Z (line 30). The number of rows in the SIL of a sequence is its support count.

Table 4 SIL representation of 1-patterns

1-pattern	SIL
1	$\langle 1\ 1 -2\ 1\ 2 -4\ 2\rangle$
2	$\langle 1\ 1 -2\ 1\rangle$
3	$\langle 1\ 1\ 2\ 3 -2\ 3 -3\ 1 -4\ 1\rangle$
4	$\langle 1\ 2 -2\ 2\ 3 -4\ 2\rangle$
7	$\langle 2\ 3 -3\ 1 -4\ 2 -5\ 1\rangle$
8	$\langle 1\ 3 -3\ 2 -5\ 2\rangle$

Example Let X be the $SIL[3] = \langle 1\ 1\ 2\ 3 -2\ 3 -3\ 1 -4\ 1\rangle$ and Y be the $SIL[4] = \langle 1\ 2 -2\ 2\ 3 -4\ 2\rangle$. $\langle(3\ 4)\rangle$ is an itemset extensible sequence and its SIL is created as described in Algorithm 6. The sequence ids matched in X and Y are 1, 2 and 4. The positions matched for $sids$ 1, 2 are 2, 3 respectively. Thus, SIL of the sequence $\langle(3\ 4)\rangle$ is $\langle 1\ 2 -2\ 3\rangle$ and its support is the number of rows in its SIL i.e. 2. Similarly, for the sequence extensible sequence $\langle(3)(4)\rangle$, the matched $sids$ are 1 and 4. Since SIL is calculated for the sequence $\langle(3)(4)\rangle$ ($\langle 4\rangle$ occurs after $\langle 3\rangle$), position of $\langle 4\rangle$ that is greater than the position of $\langle 3\rangle$ is noted in the resultant SIL according to Algorithm 6. Hence SIL of $\langle(3)(4)\rangle$ is $\langle 1\ 2 -4\ 2\rangle$ and $\sup(\langle(3)(4)\rangle)$ is the number of rows in its SIL i.e. 2. The process of constructing the SIL is illustrated in Fig. 4.

4.3.3 Candidate generation

Case 1 Given a sequential pattern S of length $k-1$, where $S = \langle s_1 s_2 \ldots s_{k-1}\rangle$, generate the candidate $C_k = \langle s_1 s_2 \ldots \{s_{k-1} U j\}\rangle$, for all $j \in CMAP_i(l)$, where l is the last item in the itemset s_{k-1}.

Example Consider the sequential pattern $\langle(1\ 2)(3)\rangle$ and from Table 2, we have $CMAP_i(3) = \{4, 7\}$. Thus, the candidates generated from the given sequence based on the itemset extension are $\langle(1\ 2)(3\ 4)\rangle$, $\langle(1\ 2)(3\ 7)\rangle$.

Case 2 Given a sequential pattern S of length $k-1$, where $S = \langle s_1 s_2 \ldots s_{k-1}\rangle$, generate the candidate $C_k = \langle s_1 s_2 \ldots s_{k-1} j\rangle$, for all $j \in CMAP_s(l)$, where l is the last item in the itemset s_{k-1}.

Example Consider the sequential pattern $\langle(1\ 2)(3)\rangle$ and from Table 3, we have $CMAP_s(3) = \{4, 8\}$. Thus, the candidates generated from the given sequence based on the sequence extension are $\langle(1\ 2)(3)(4)\rangle$, $\langle(1\ 2)(3)(8)\rangle$.

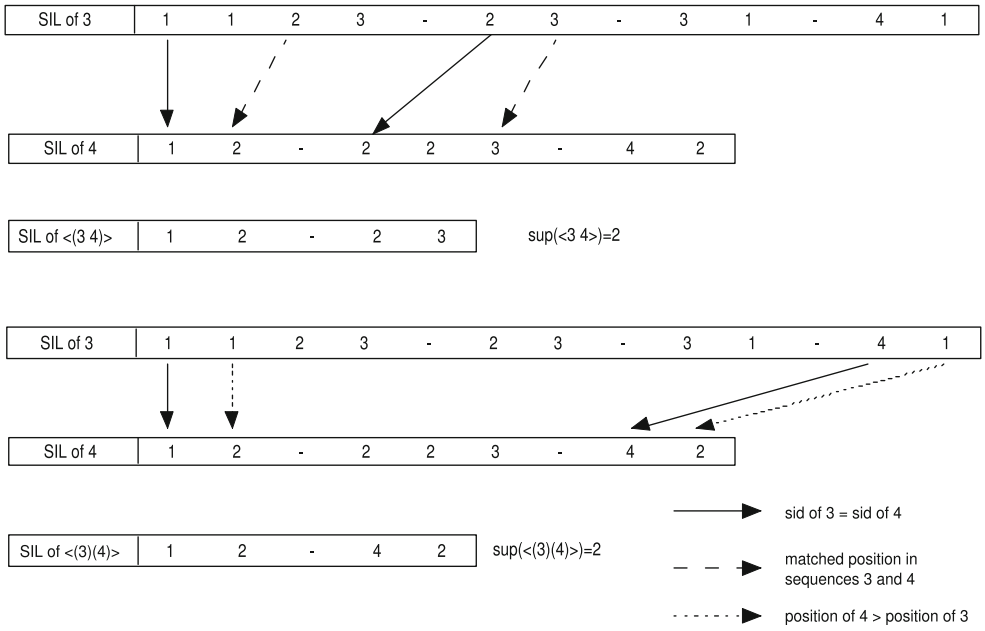


Fig. 4 An example to construct SIL of the sequences $\langle(3\ 4)\rangle$ and $\langle(3)(4)\rangle$

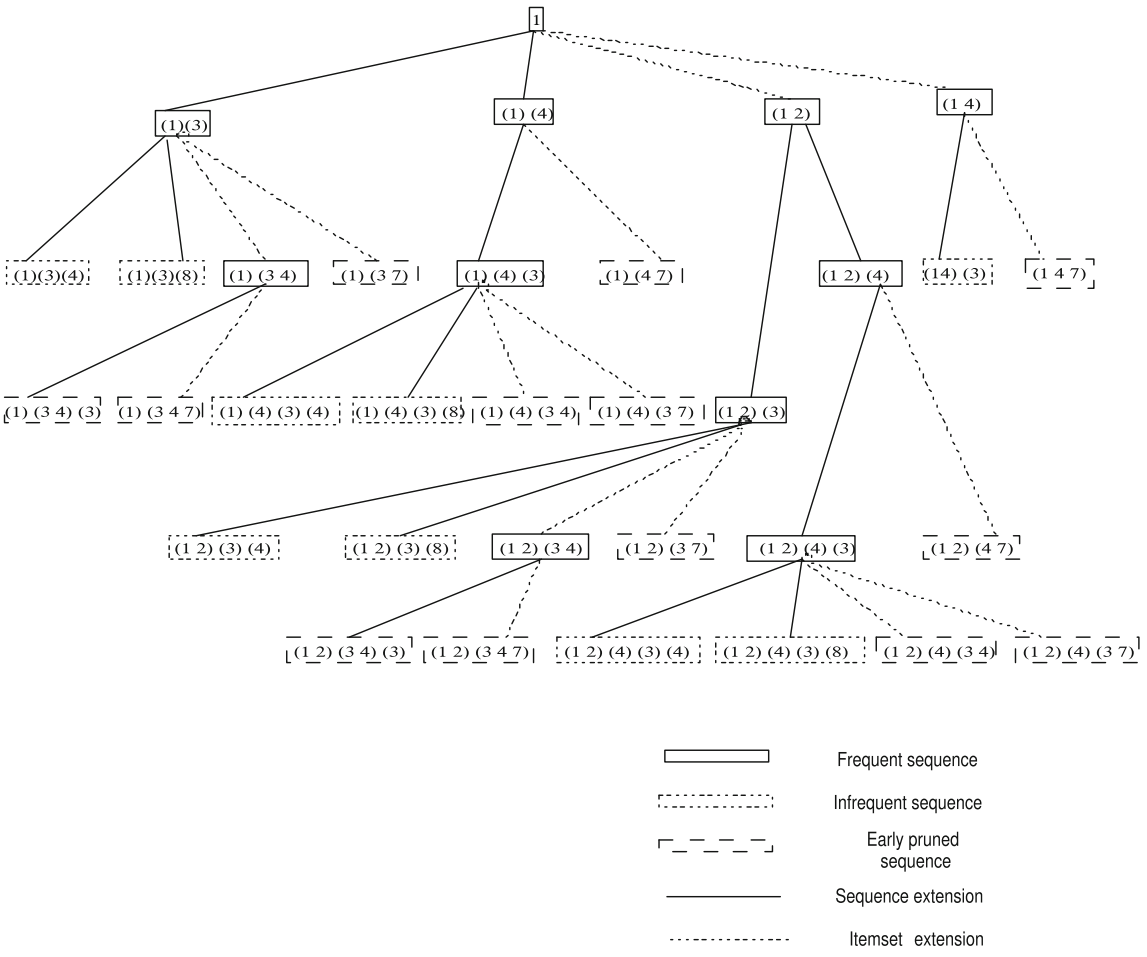


Fig. 5 Example search space for sequence $\langle(1)\rangle$

Algorithm 7 describes the procedure of generating the candidate sequences. The input to the algorithm is a sequence to be extended and its SIL. Candidate sequences are generated by combining the sequence and co-occurrence items of the last item in the sequence (lines 7-11 and lines 21-25). Before finding the support count of the sequence (line 14 and line 28), $EARLY\_PRUNE$ procedure is invoked and it is described in Algorithm 8. The sequence that satisfy the minimum support threshold (line 15 and line 29) is considered as the input for the next recursive call (line 17 and line 31). Thus, the process of recursion is repeated until all the frequent sequences are generated. Before doing a recursive call, each mapper emits the sequence and its support as a $\langle Key, Value \rangle$ pair.

Algorithm 6 Create SIL of a sequence

Input:
X ▷ SIL of a sequence
Y ▷ SIL of an item
Output:
Z ▷ SIL of a new sequence
1: **function** $CREATE\_SIL\_SEQUENCE(X, Y)$
2: **for** each sid in X **do**
3: $Polist_x \leftarrow X.get(X.sid)$ ▷ Get the position list of $X.sid$
4: Create a list $Poslist_z$ to store the position list of the new sequence
5: **for** each sid in Y **do**
6: **if** $X.sid \neq Y.sid$ **then**
7: break
8: **end if**
9: $Polist_y \leftarrow Y.get(Y.sid)$ ▷ Get the position list of $Y.sid$
10: **if** X is an itemset extensible sequence **then**
11: **for** each $Pos_x \in Poslist_x$ **do**
12: **for** each $Pos_y \in Poslist_y$ **do**
13: **if** $Pos_x = Pos_y$ **then**
14: $Poslist_z \leftarrow Pos_x$
15: break
16: **end if**
17: **end for**
18: **end for**
19: **else if** X is a sequence extensible sequence **then**
20: **for** each $Pos_x \in Poslist_x$ **do**
21: **for** each $Pos_y \in Poslist_y$ **do**
22: **if** $Pos_y > Pos_x$ **then**
23: $Poslist_z \leftarrow Pos_y$
24: break
25: **end if**
26: **end for**
27: **end for**
28: **end if**
29: **end for**
30: $Z.put(X.sid, Poslist_z)$
31: **end for**
32: return Z
33: **end function**

For example, $CMAP_s(1) = \{3, 4\}$, Hence, the possible 2-sequences generated from $\langle(1)\rangle$ are $\langle(1)(3)\rangle$, $\langle(1)(4)\rangle$, $\langle(1\ 3)\rangle$ and $\langle(1\ 4)\rangle$. In DSPC, the depth first search procedure is followed to extend a frequent sequence of length k to $k + 1$. Hence, the 2-sequence that is extended first is $\langle(1)(3)\rangle$. It is extended to $\langle(1)(3)(4)\rangle$ and found infrequent. Similarly, the next generated sequence $\langle(1)(3)(8)\rangle$ is found to be infrequent. The next sequence $\langle(1)(3\ 4)\rangle$ is frequent and is extended to $\langle(1)(3\ 4)(3)\rangle$ and $\langle(1)(3\ 4\ 7)\rangle$. These newly generated sequences are early pruned (refer Algorithm 8) as 3 does not belong to $CMAP_s(3)$ and 7 does not belong to $CMAP_i(3)$. This process of generating the candidates and early prune is illustrated for the sequence $\langle(1)\rangle$ in Fig. 5 and the frequent sequences generated for the sample dataset are shown in Fig. 6.

Algorithm 7 Candidate Generation

Input:
S ▷ Sequence
SIL_s ▷ SIL of sequence S
Output:
$new\_sequence$ ▷ New sequence extended from S
$count$ ▷ Support of the sequence
1: **function** $GENERATE\_SEQUENCES, SIL_s$
2: **if** S is length-1 sequence **then**
3: $last\_item \leftarrow S$
4: **else**
5: $last\_item \leftarrow$ Find the last item in the sequence S
6: **end if**
7: **for** each co-occurrence item $item_c$ in $CMAP_s$ $(last\_item)$ **do**
8: **if** $item_c \notin SIL$ **then**
9: break
10: **end if**
11: $new\_sequence \leftarrow S \cup item_c$
12: **if** $EARLY\_PRUNE(new\_sequence)$=FALSE **then**
13: $SIL[new\_sequence] \leftarrow CREATE\_SIL\_SEQUENCE\ (SIL[S], SIL[item_c])$
14: $count \leftarrow$ Find the support of $new\_sequence$ ▷ sup($new\_sequence$) is the number of rows in $SIL[new\_sequence]$
15: **if** $count \geq min\_sup$ **then**
16: output($new\_sequence, count$)
17: $GENERATE\_SEQUENCE$ ($new\_sequence$, $SIL[new\_sequence]$)
18: **end if**
19: **end if**
20: **end for**
21: **for** each co-occurrence item $item_c$ in $CMAP_i$ $(last\_item)$ **do**

```
22:        if item_c ∉ SIL then
23:            break
24:        end if
25:        new_sequence ← S ∪ item_c
26:        if EARLY_PRUNE(new_sequence)=FALSE then
27:            SIL[new_sequence] ← CREATE_SIL_SEQUENCE (SIL[S], SIL[item_c])
28:            count ←Find the support of new_sequence
29:            if count ≥ min_sup then
30:                output(new_sequence, count)
31:                GENERATE_SEQUENCE (new_sequence, SIL [new_sequence])
32:            end if
33:        end if
34:    end for
35: end function
```

4.3.4 Early prune properties

Property 1 An itemset extension candidate $C_k = \langle s_1 s_2 \ldots s_{k-2}\{s_{k-1} \cup j\}\rangle$ cannot be frequent if either of the following two conditions holds.

(i) If there exists an item $l \in s_{k-1}$ and j is not a member of $CMAP_i(l)$.

(ii) If there exists an item $l \in s_{k-2}$ and j is not a member of $CMAP_s(l)$.

Proof According to property 1 in [8]. For example consider the sequence $\langle(1\ 4\ 7)\rangle$ that is extended from $\langle(1\ 4)\rangle$. Since item 7 is not a member of $CMAP_i(1)$, $\langle(1\ 4\ 7)\rangle$ is pruned before calculating its support count. Similarly, the sequence $\langle(1\ 2)(4\ 7)\rangle$ is pruned as 7 is not a member of $CMAP_s(2)$. □

Property 2 A sequence extension candidate $C_k = \langle s_1 s_2 \ldots s_{k-2} s_{k-1} j\rangle$ cannot be frequent if there exists an item $l \in s_{k-1}$ and j is not a member of $CMAP_s(l)$.

Proof According to property 2 in [8]. For example, consider the sequence $\langle(1\ 2)(3\ 4)(3)\rangle$ that is extended from $\langle(1\ 2)(3\ 4)\rangle$. It is early pruned as 3 is not a member of $CMAP_s(3)$. □

The early prune procedure is described in Algorithm 8. It returns TRUE if the sequence S cannot be frequent, otherwise it returns FALSE. An itemset extensible sequence cannot be frequent if its last item is not a member of $CMAP_i(x)$ (line 7) or not a member of $CMAP_s(y)$ (line 12), where x represents the items in the last itemset and y represents the items in the last but one itemset. Similarly, a

Fig. 6 Frequent sequences

Sample Dataset	(1 2 3) (3 4) (3 8), (1 2) (1 4 6) (3 4 7), (3 5 7) (8), (3) (1 4 7), (7) (8)	
	Itemset extensible sequences	Sequence extensible sequences
Frequent 2-sequences	(1 2), (1 4), (3 4), (3 7), (4 7)	(1)(3), (1)(4), (2)(3), (2)(4), (3)(4), (3)(8), (4)(3), (7)(8)
Candidate 3-sequences	(1 4 7), (1) (3 4), (1) (3 7), (1) (4 7), (2) (3 4), (2) (3 7), (2) (4 7), (3) (4 7) , (4) (3 4), (4) (3 7) Early pruned sequences (1 4 7), (1) (3 7), (1) (4 7), (2) (3 7), (2) (4 7), (3) (4 7), (4) (3 4), (4) (3 7)	(1 2) (3), (1 2) (4), (1 4) (3), (1) (3) (4), (1) (3) (8), (1) (4) (3), (2) (4) (3), (3 4) (3), (3 7) (8), (4 7) (8), (4) (3) (4), (4) (3) (8) Early pruned sequences (3 4) (3), (4 7) (8)
Frequent 3-sequences	(1) (3 4), (2) (3 4)	(1 2) (3), (1 2) (4), (1) (4) (3), (2) (4) (3)
Candidate 4-sequences	(1) (3 4 7), (2) (3 4 7), (1 2) (3 4), (1 2) (3 7), (1 2) (4 7), (1) (4) (3 4), (1) (4) (3 7), (2) (4) (3 4), (2) (4) (3 7) Early pruned sequences (1) (3 4 7), (2) (3 4 7), (1 2) (3 7), (1 2) (4 7), (1) (4) (3 4), (1) (4) (3 7), (2) (4) (3 4), (2) (4) (3 7)	(1) (3 4) (3), (2) (3 4) (3), (1 2) (3) (4), (1 2) (3) (8), (1 2) (4) (3), (1) (4) (3) (4), (1) (4) (3) (8), (2) (4) (3) (4), (2) (4) (3) (8) Early pruned sequences (1) (3 4) (3), (2) (3 4) (3)
Frequent 4-sequences	(1 2) (3 4)	(1 2) (4) (3)
Candidate 5-sequences	(1 2) (3 4 7), (1 2) (4) (3 4), (1 2) (4) (3 7) Early pruned sequences (1 2) (3 4 7), (1 2) (4) (3 4), (1 2) (4) (3 7)	(1 2) (3 4) (3), (1 2) (4) (3) (4), (1 2) (4) (3) (8) Early pruned sequences (1 2) (3 4) (3)
Frequent 5-sequences	None	None

sequence extensible sequence cannot be frequent if its last item is not a member of $CMAP_s(y)$ (line 20).

Theorem 1 *Given a sequence dataset SD and the minimum support threshold min_sup, DSPC finds all the frequent sequences whose support is greater than or equal to the min_sup.*

Algorithm 8 Early Prune

Input:
S ▷ Sequence
Output:
TRUE/FALSE

```
1:  function EARLY_PRUNE(S)
2:      I_k ← Find the last itemset in S
3:      I_{k-1} ← Find the last but one itemset in S
4:      last_item ← Find the last item in S
5:      if S is itemset extensible sequence then
6:          for each item x ∈ I_k do
7:              if last_item ∉ CMAP_i(x) then
8:                  return TRUE
9:              end if
10:         end for
11:         for each item y ∈ I_{k-1} do
12:             if last_item ∉ CMAP_s(y) then
13:                 return TRUE
14:             end if
15:         end for
16:         return FALSE
17:     end if
18:     if S is sequence extensible sequence then
19:         for each item y ∈ I_{k-1} do
20:             if last_item ∉ CMAP_s(y) then
21:                 return TRUE
22:             end if
23:         end for
24:         return FALSE
25:     end if
26: end function
```

Proof The theorem is proved by showing that the proposed pruning properties in DSPC algorithm will never miss the frequent sequences.

1. Pruning the infrequent 2-length sequences:
   As described in Algorithm 1, the first phase of DSPC algorithm constructs $CMAP_i$ and $CMAP_s$. According to Lemma 1 and Lemma 2, the $CMAP$ consists of all the 2-length frequent sequences and hence we do not miss any 2-length frequent sequence in the first phase.
2. Pruning the infrequent k-length itemset extensible sequences ($k > 2$):
   Let C_k be the k-length itemset extensible sequence generated by concatenating item y to the last itemset of $(k-1)$-length sequence C_{k-1}. Now, according to early prune property 1, C_k can be pruned if any one of the following two conditions hold.
   - If there exists an item x, where $y \notin CMAP_i(x)$, such that x belongs to the last itemset of C_k, then, according to Lemma 1, $sup(\langle(xy)\rangle) < min\_sup$. Since $\langle xy\rangle$ is a subsequence of C_k, $sup(C_k) \leq sup(\langle(xy)\rangle)$ and it can be pruned without further extension.
   - If there exists an item x, where $y \notin CMAP_s(x)$, such that x belongs to the last but one itemset of C_k, then, according to Lemma 2, $sup(\langle(x)(y)\rangle) < min\_sup$. Since $\langle(x)(y)\rangle$ is a subsequence of C_k, $sup(C_k) \leq sup(\langle(x)(y)\rangle)$ and it can be pruned without further extension.

   Consequently, pruning the itemset extensible sequence whose support is less than the min_sup will not miss any frequent sequence.
3. Pruning the infrequent k-length sequence extensible sequences ($k > 2$):
   Let C_k be the k-length sequence extensible sequence generated by concatenating item y to the last itemset of $(k-1)$-length sequence C_{k-1}. If there exists an item x, where $y \notin CMAP_s(x)$, such that x belongs to the last but one itemset of C_k , then, according to Lemma 2, $sup(\langle(x)(y)\rangle) < min\_sup$. Since $\langle(x)(y)\rangle$ is a subsequence of C_k, $sup(C_k) \leq sup(\langle(x)(y)\rangle)$ and it can be pruned without further extension. Consequently, pruning the sequence extensible sequence whose support is less than the min_sup will not miss any frequent sequence.

   Therefore, all the frequent sequences whose support is greater than or equal to the min_sup will be returned by the DSPC algorithm.

□

5 Results

To assess the performance of the proposed algorithm, various experiments have been performed on our in-house Hadoop cluster equipped with 8 data nodes, each node having Intel Xeon CPU E5-2640 (2.5GHZ/6-core/15MB/95W) processor, 16 GB RAM, 2*300 HDD and each node runs on a CentOS 6.5 server with Hadoop 1.2.1. All the algorithms were implemented using JDK 1.8.0_31. The experiments have been performed on both the real datasets and synthetic datasets. The three real-world datasets that are used in the experiments, namely, $Kosarak$,

Table 5 The real datasets

Dataset	Sequence count	Item count	Average length of the sequence
$Kosarak$	990,002	41,270	8.099
$BMSWebView2$	77,512	3,340	4.62
$MSNBC$	989,818	17	5.7

$MSNBC$, and $BMSWebview2$, are previously used in SPM studies [8, 10, 17, 18]. The characteristics of the real datasets are shown in Table 5 and they are described as follows.

- $Kosarak$ is a large dataset containing click-stream data of a Hungarian online news portal. It consists of 990,002 sequences and 41,270 distinct items. The dataset is downloaded from FIMI repository.[1] The 10k version of the $Kosarak$ dataset having 10,000 sequences was previously used in [8]. Also, the 25k version having 25,000 sequences was studied in [17]. To test the performance of the proposed algorithm on large datasets, the full version of the dataset having approximately one million sequences is used in the current experiments.
- $MSNBC$ dataset[2] contains log information of Internet Information Server (IIS) for $MSNBC$ website[3] and news-related portions of the website for the entire day of September 28, 1999. It consists of 989,818 sequences, where each sequence represents the page views of a user on that particular day and the average number of visits per user is 5.7. It has been used in the previous studies [10, 18].
- $BMSWebView2$ dataset contains several months of clickstream data from an e-commerce website.[4] Each transaction is a list of web pages browsed by a customer in a browsing session. Different sessions of the same user are considered as a sequence. The dataset is downloaded from SPMF open source data mining library[5] and it has been used in the recent studies [17, 18].

Like most of the studies in sequential pattern mining [1, 3–5, 13, 22], we generated the synthetic datasets using the major parameters that are specified during the data generation procedure described in [25]. The synthetic dataset generation and their meaning are given in Table 6. The proposed algorithm is compared with the state-of-the-art sequential pattern mining algorithms implemented using the MapReduce framework, namely, SPAMC-UDLT [4], DSPDBV [15], SPAMC [5] and DPSP [16].

5.1 Performance comparison

To evaluate the performance of the DSPC algorithm, experiments have been done on the three real datasets and four synthetic datasets. While generating the synthetic datasets, the parameters D and C are kept constant and the values of N and T are changed. Firstly, we have tested the performance of the algorithms on the datasets $C20D10N1T15$, $C20D10N1T20$, $C20D10N10T15$ and $C20D10N10T20$ which contains 10 million sequences and the number of transactions per sequence is 20. The minimum support is varied from 0.1% to 0.3%. Figure 7 shows the execution time of the five algorithms on the synthetic datasets with respect to different values of the minimum support. It can be observed that the execution time of the algorithms decrease with an increase in the minimum support. Due to the iterative MapReduce implementation, DSPDBV, SPAMC and DPSP spends more time for scanning and reloading the data. Hence, the performance of these algorithms is less compared to SPAMC-UDLT and DSPC. Moreover, SPAMC and SPAMC-UDLT represent the data in bit format and they take more time for bitwise AND operation for large values of N. The amount of computation performed during bitwise AND increases with the number of transactions and the number of distinct items. For example, if there are 1,000,000 transactions in a dataset, then 1,000,000 bit-AND operations are required for testing each candidate during the support count calculation. It becomes worse, especially for large values of N. Hence, it can be observed that SPAMC-UDLT takes more time for completion while testing on the datasets $C20D10N10T15$ and $C20D10N10T20$ compared to $C20D10N1T15$, $C20D10N1T20$ due to increased number of bitwise AND operations.

Table 6 Parameters used in the synthetic data generation

Parameter	Description
D	Number of sequences in million
C	Average number of itemsets per sequence
T	Average number of items per itemset
N	Number of distinct items in thousand

[1] http://fimi.ua.ac.be/data/

[2] http://kdd.ics.uci.edu/databases/msnbc/msnbc.html

[3] http://www.msnbc.com

[4] https://www.gazelle.com/

[5] http://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php

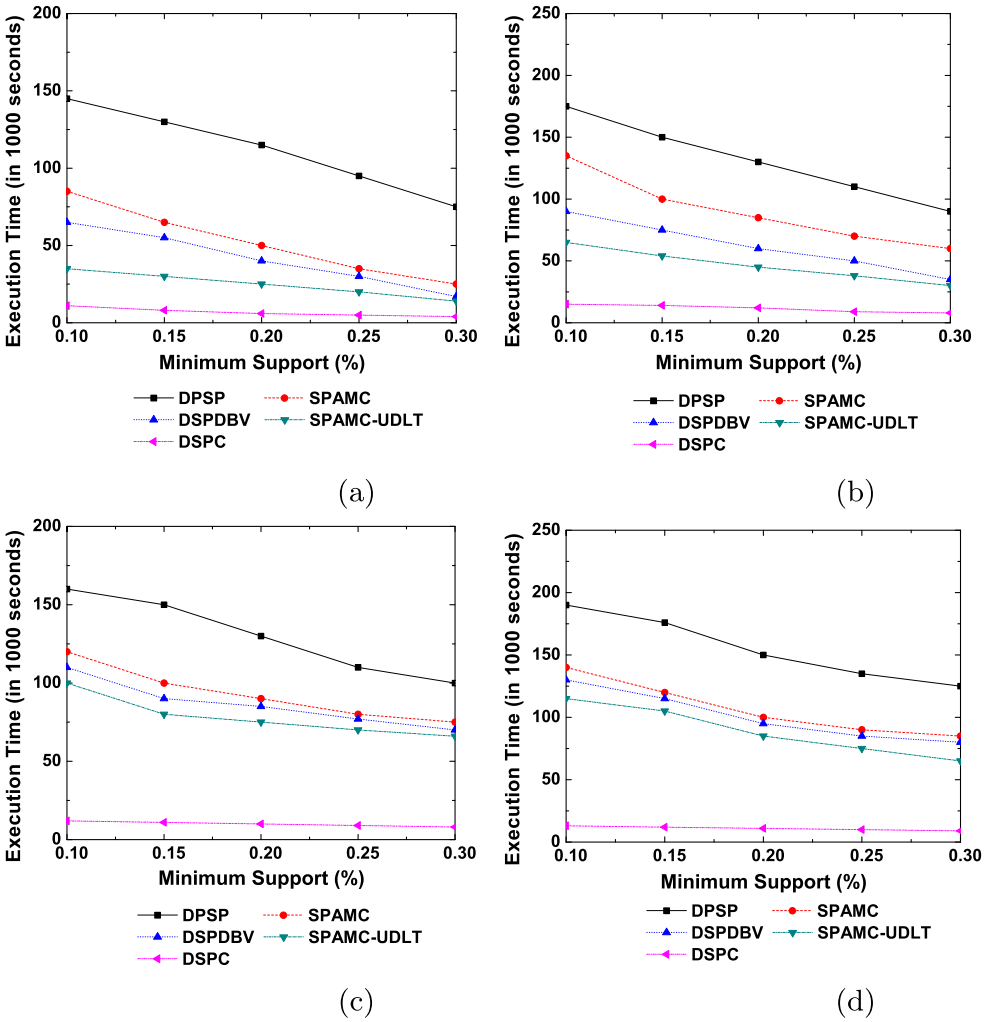


Fig. 7 Performance comparison of five algorithms on the datasets **a** $C20D10N1T15$ **b** $C20D10N1T20$ **c** $C20D10N10T15$ **d** $C20D10N10T20$

Figure 8 shows the execution time and distribution of frequent sequences on the real datasets. From Fig. 8a, c and e, it is observed that for higher values of the minimum support, SPAMC-UDLT and DSPDBV have similar performance and DSPDBV works faster than SPAMC and DPSP. Figure 8b shows the distribution of frequent sequences of $Kosarak$ dataset for different values of the minimum support. It can be observed that frequent sequences of longer length are obtained when minimum support is less than 0.2%. Figure 8d shows the distribution of frequent sequences of $BMSWebView2$ dataset for different values of minimum support. It shows that the frequent sequences of longer length are obtained when the minimum support is less than 0.15%. From Fig. 8f, we can observe that the number of frequent 1-sequences obtained for the minimum supports 0.1%, 0.15% and 0.2% is the same (17 sequences) and for the minimum supports 0.25% and 0.3%, 16 frequent 1-sequences are obtained. As shown in Fig. 8a, c and e, the proposed DSPC algorithm has better performance than the other four algorithms and is more capable of dealing with large number of distinct items and large number of items per transaction. The main difference between the proposed DSPC algorithm and the previous algorithms is the size of the search space. In DSPC algorithm, we consider the patterns that actually appear in the dataset and reduce the search space by efficiently generating the candidate sequences using the $CMAP$ data structure and following the depth first search.

In summary, the following are the reasons why DSPC outperforms the other existing MapReduce solutions for SPM problem.

1. DSPC uses a simple comparison operation between two SIL's. The size of the SIL decreases as the length of the frequent sequence increase. As a result, the support counting will be done in less time.
2. For lower values of the minimum support, more frequent sequences of greater length will be found and it leads to high execution time. This problem is solved by adapting the $CMAP$ data structure to early prune

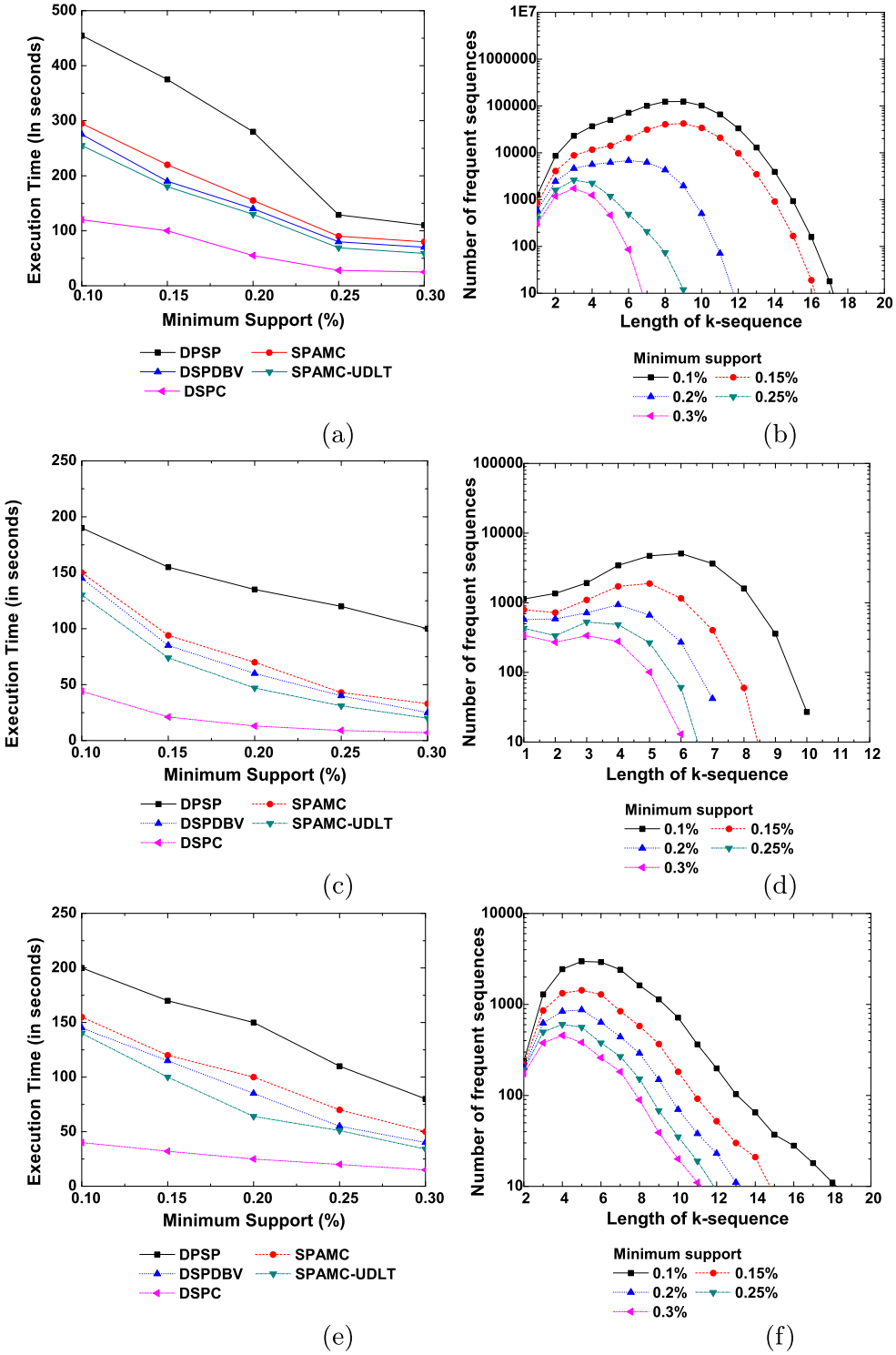


Fig. 8 Real datasets: Execution time and distribution of frequent sequences (**a**, **b**) $Kosarak$ (**c**, **d**) $BMSWebView2$ (**e**, **f**) $MSNBC$

the infrequent sequences and extend only the frequent sequences in a depth first manner.

3. Two phase MapReduce model is proposed with only one MapReduce job in each phase. This overcomes the problem of communication and scheduling overhead.

5.2 Scalability

To evaluate the scalability of the five algorithms, a series of datasets have been generated ranging from 2 million transactions to 10 million transactions. The parameters used

in this series of datasets are $N = 0.02$, $T = 20$ and $C = 100$. Also, the scalability with respect to the varying number of items is tested on a dataset with $D = 2$, $T = 20$ and $C = 100$. During this experiment, the number of distinct items is increased from 20 to 100. Figure 9 shows the scalability results. The scalability study is done in two ways: 1) the size of the dataset is kept constant and we increased the number of distinct items and 2) the parameters N, T and C are kept constant and increased the number of sequences. During these experiments, the minimum support is set to 0.1%.

From Fig. 9a, it can be observed that the execution time of all the algorithms increases with the increase in the number of sequences. The execution time of DPSP, DSPDBV and SPAMC increased drastically after 4 million sequences and the execution time of SPAMC-UDLT increased after 6 million sequences. Since DSPDBV, SPAMC and DPSP are iterative MapReduce algorithms, as the number of sequences increases, they involve more number of MapReduce rounds and leads to more execution time. SPAMC-UDLT implements a streaming MapReduce model and there is no need to reload the data. Hence, SPAMC-UDLT is more scalable compared to DSPDBV, SPAMC and DPSP. However, the mining phase of SPAMC-UDLT incurs more overhead due to the increased search space as it implements the breadth first search strategy to evaluate the candidate sequential patterns. Also, SPAMC and SPAMC-UDLT are derived from SPAM and they consume more memory to represent the items in bitmap form when the number of distinct items is large. They take more time for tree construction and bit-AND operation, and do not scale well for large values of N. In the second phase of DSPC, at each mapper, only the SIL of the items present in that particular input split is calculated and then we follow the depth first search strategy while generating the candidate sequential patterns. Hence, as shown in Fig. 9a and b, DSPC scales more linearly compared to the other four algorithms.

5.3 Speedup

The speedup of the algorithms is tested on a synthetic dataset with the parameters $D = 2$, $N = 1$, $T = 20$ and $C = 20$. The minimum support is set to 0.1%. In this experiment, to find the speedup, runtime of the algorithms is computed on the nodes varying from 2 to 8. The speedup is defined as the increase in speed by increasing the number of nodes. Dividing the runtime of an algorithm on a single node by the runtime on multiple nodes (in our case, 2, 4, 6 and 8 nodes) gives the speedup of the algorithm. From Fig. 10a, it can be observed that the speedup of all the algorithms tends to increase with the number of nodes. But, this linear speedup may not be true in a further increase in the number of nodes due to the increase in communication cost between the mappers and reducers. The communication cost is analyzed on the same dataset as described above. The total amount of data transferred over the network between mapper node and reducer node is known as the communication cost. In the iterative MapReduce models, the frequent k-sequences generated by the k^{th} round MapReduce job are passed to the next round job to generate the $k + 1$ frequent sequences. The master node has to schedule the jobs in each round. Moreover, the output of each round is written to the HDFS from which the next round MapReduce job reads. Therefore, SPAMC-UDLT and DSPC significantly outperforms the iterative approaches, DPSP, SPAMC and DSPDBV in terms of the communication cost as shown in Fig. 10b. Due to the early prune properties applied in the second phase of DSPC, the number of candidates will be less compared to SPAMC-UDLT and this reflects the communication cost as shown in Fig. 10b.

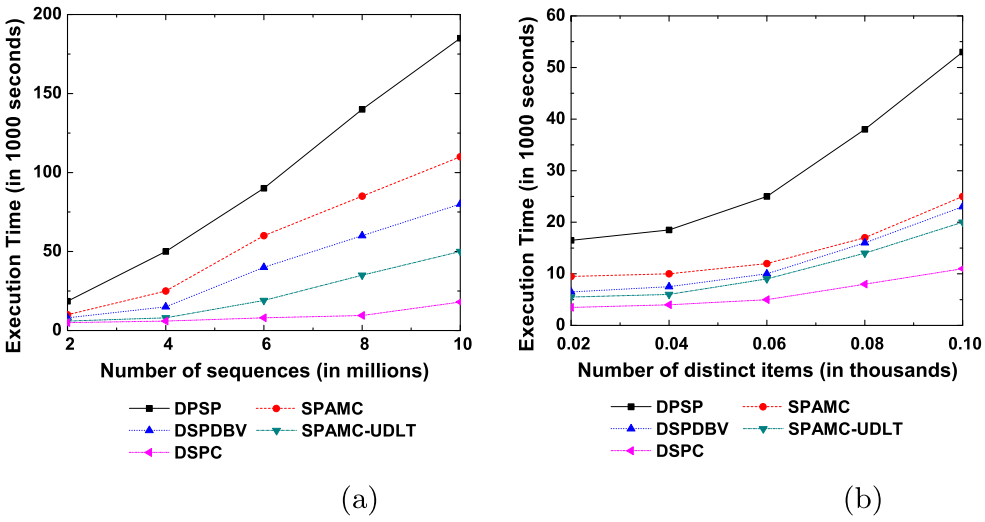


Fig. 9 Scalability of five algorithms with respect to change in number of **a** sequences **b** items

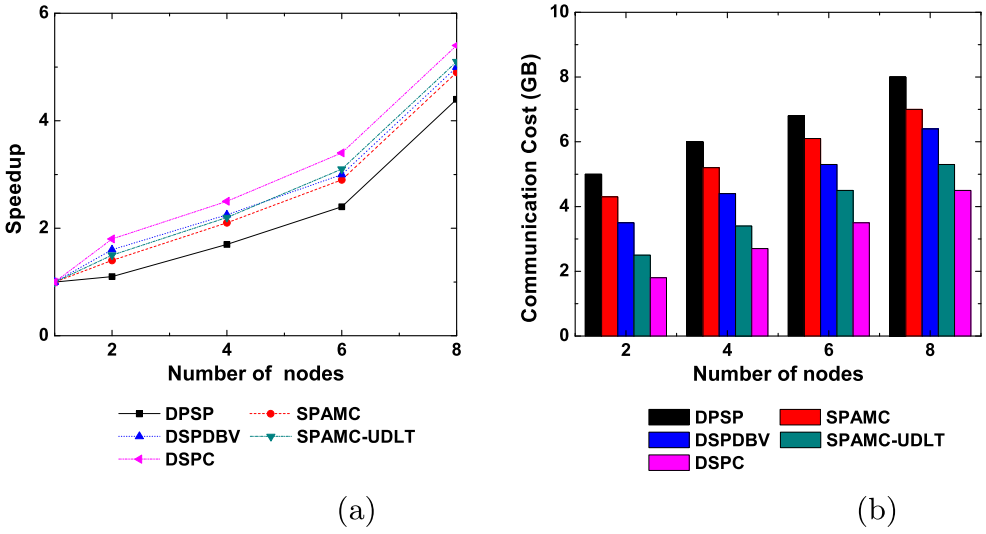


Fig. 10 **a** Speedup **b** Communication cost

5.4 Memory usage

The memory usage is compared among all the five algorithms using both the real datasets and a synthetic dataset. The parameters used in the synthetic data generation are $D = 2$, $N = 1$, $T = 20$ and $C = 20$. Figure 11 shows the memory usage for $Kosarak$, $BMSWebView2$, $MSNBC$ and $C20D2N1T20$. The memory overhead involved in

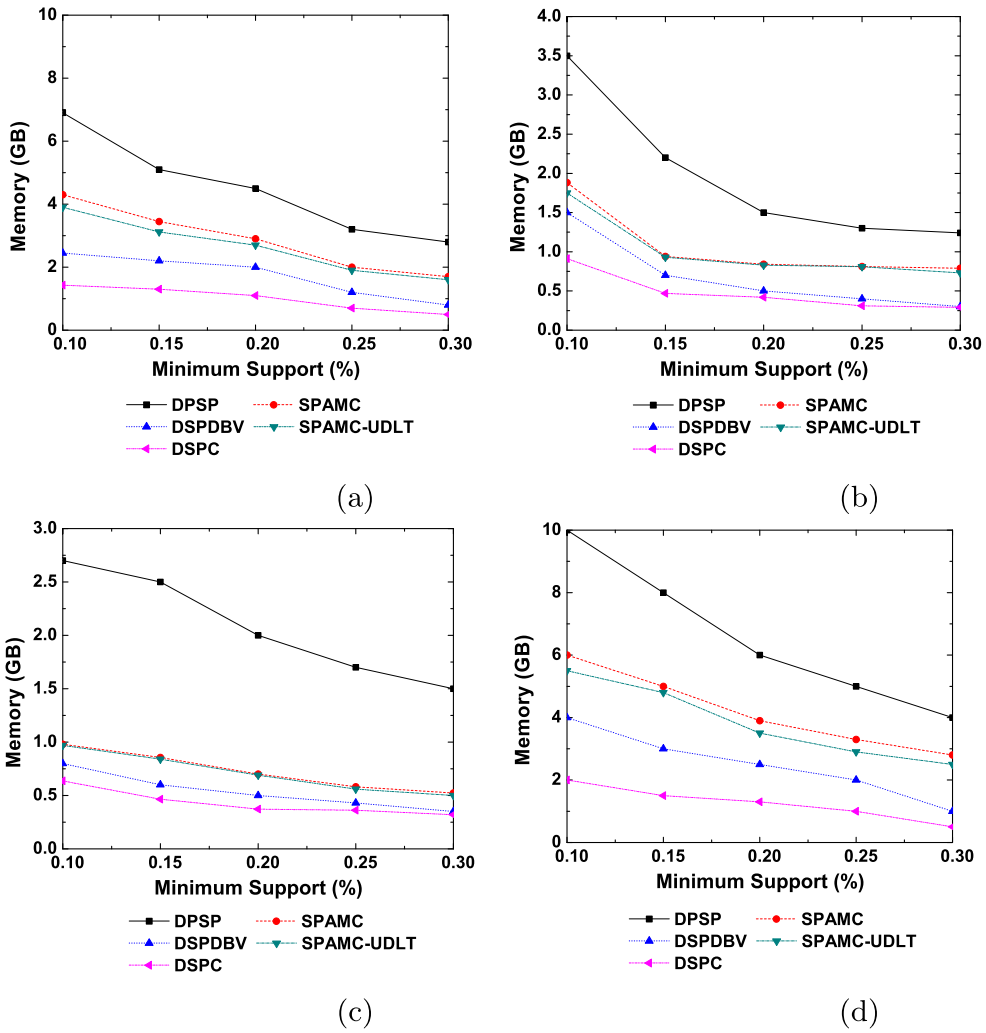


Fig. 11 Memory usage of five algorithms on the datasets **a** $Kosarak$ **b** $BMSWebView2$ **c** $MSNBC$ **d** $C20D2N1T20$

Table 7 $CMAP$ size

(a)					
Minimum Support (%)	0.1	0.15	0.2	0.25	0.3
Size (KB)	50.6	27.2	12.3	7.45	5.2
(b)					
Minimum Support (%)	0.1	0.15	0.2	0.25	0.3
Size (KB)	0.75	0.74	0.73	0.72	0.7
(c)					
Minimum Support (%)	0.1	0.15	0.2	0.25	0.3
Size (KB)	0.63	0.58	0.53	0.5	0.46
(d)					
Minimum Support (%)	0.1	0.15	0.2	0.25	0.3
Size (MB)	112	108	105.4	101.3	98

(a) $Kosarak$ (b) $BMSWebView2$ (c) $MSNBC$ (d) $C20D2N1T20$

using the co-occurrence information is also studied. In Table 7, the $CMAP$ size for different datasets is given. It is observed that the $CMAP$ size decreases with an increase in the minimum support and it is 50.6 KB for $Kosarak$, 0.75 KB for $BMSWebView2$, 0.63 KB for $MSNBC$ and 112 MB for $C20D2N1T20$ datasets. Since DSPDBV use compressed bit vector representation, its memory usage is less compared to SPAMC, SPAMC-UDLT and DPSP. As shown in Fig. 11, the memory usage of SPAMC and SPAMC-UDLT is similar and less compared to DPSP. However, the breadth first search strategy of SPAMC-UDLT needs to store all the frequent sequences of length k in memory in order to generate the sequences of length $k+1$. It is also observed that, the memory usage of DSPDBV is similar to the proposed algorithm for higher values of the minimum support. However, when the minimum support is decreased, more number of candidate sequences are generated, which increases the number of MapReduce rounds in DSPDBV. DSPC use co-occurrence information while generating the candidate sequences and then applies the early prune properties which cuts down the memory usage to a great extent. This explains why DSPC consumes less memory when compared to the other four algorithms.

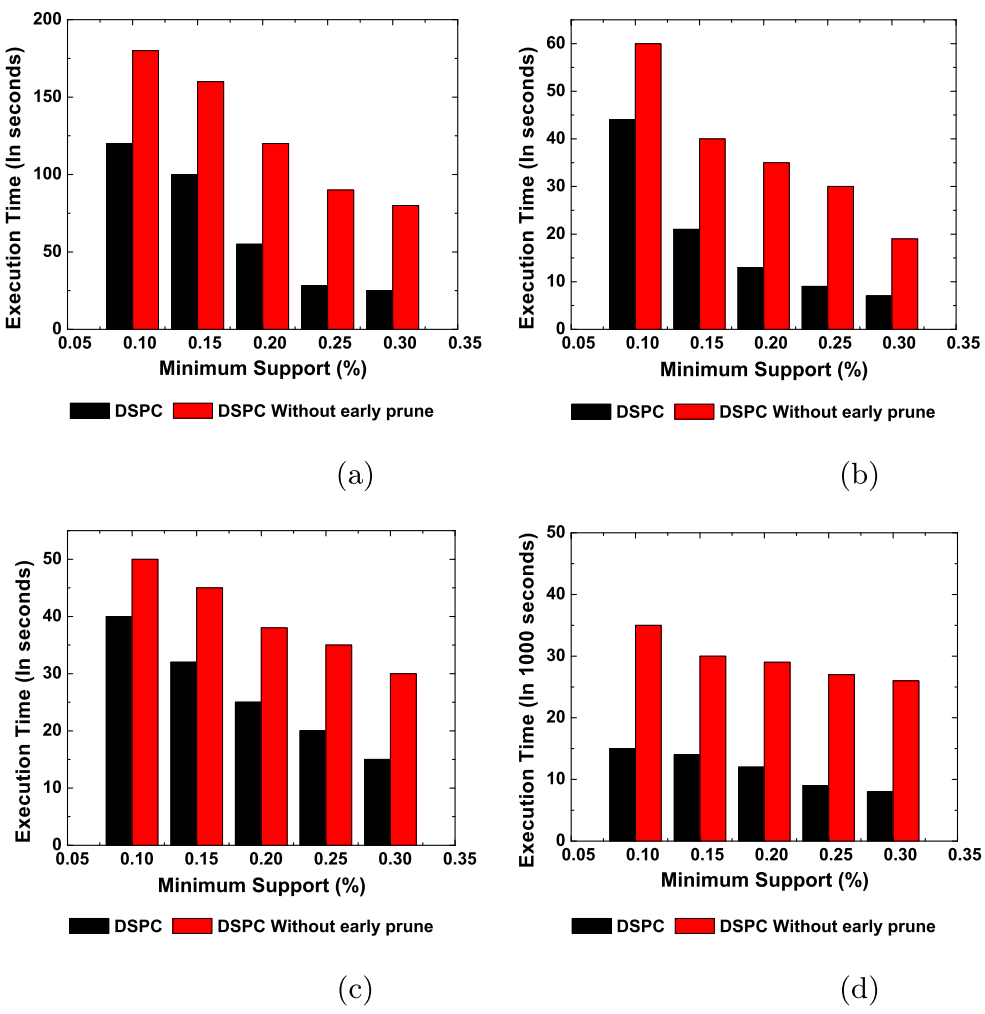


Fig. 12 Performance evaluation of early prune properties on the datasets **a** $Kosarak$ **b** $BMSWebView2$ **c** $MSNBC$ **d** $C20D10N10T20$

5.5 Evaluation of the early prune properties

In this section, to evaluate the performance of the proposed pruning properties, we compare the DSPC algorithm with early prune and without prune. The experiment is performed on the three real datasets and on the synthetic dataset $C20D10N10T20$. Figure 12 shows the performance of the algorithms with respect to the minimum supports varying from 0.1% to 0.3%. As shown in Fig. 12, the early prune improves the performance of the DSPC algorithm. It is observed that, the performance gain is higher for higher values of the minimum support. The reason is that the $CMAP$ size gets lower for higher values of the minimum support and it increases the percentage of the pruned candidates. The performance gain range from 40% to 70% on the $Kosarak$ dataset, 30% to 70% on the $BMSWebView2$ dataset, 30% to 50% on the $MSNBC$ dataset, and 60% to 70% on the synthetic dataset.

6 Conclusion

In this paper, two phase MapReduce based DSPC algorithm is proposed and it reduces the overheads of communication and scheduling in comparison with the existing sequential pattern mining algorithms. The current work also aims to narrow the search space by efficiently generating the candidates using the co-occurrence information. DSPC uses an efficient SIL data structure rather than the conventional bitmap representation that requires to store a zero in the bitmap even if the item is not present in the transaction. Moreover, SIL allows efficient support counting. DSPC also avoids the necessity to scan the input data multiple times. Most of the previous MapReduce based sequential pattern mining algorithms, have not aimed with the issues such as pruning the candidates that do not appear frequently in the input sequences. To address these issues, we used the $CMAP$ data structure which helps in the efficient candidate generation and early prune of the infrequent candidates. We implemented DSPC on a real world Hadoop cluster having eight nodes. The experimental results prove that the proposed approach efficiently handles the large datasets with long sequences and huge number of distinct items in terms of both time and space. In future, the authors would like to investigate the frequent sequences that include time intervals between successive items.

References

1. Agrawal R, Srikant R (1995) Mining Sequential Patterns. In: Proceedings of the Eleventh international conference on data engineering, pp 3–14
2. Aseervatham S, Osmani A, Viennet E (2006) bitSPADE: a lattice-based sequential pattern mining algorithm using bitmap representation. In: Proceedings of the Sixth international conference on data mining
3. Ayres J, Flannick J, Gehrke J, Yiu T (2002) Sequential PAttern mining using a bitmap representation. In: Proceedings of the Eighth ACM SIGKDD international conference on knowledge discovery and data mining
4. Chen CC, Shuai HH, Chen MS (2017) Distributed and scalable sequential pattern mining through stream processing. Knowl Inf Syst 53(2):365–390
5. Chen CC, Tseng CY, Chen MS (2013) Highly scalable sequential pattern mining based on MapReduce model on the cloud. In: Proceedings of IEEE international congress on big data, pp 310–317
6. Chen J (2010) An UpDown directed acyclic graph approach for sequential pattern mining. IEEE Trans Knowl Data Eng 22(7):913–928
7. Dean J, Ghemawat S (2008) MapReduce: simplified data processing on large clusters. Commun ACM 51(1):107–113
8. Fournier-Viger P, Gomariz A, Campos M, Thomas R (2014) Fast vertical mining of sequential patterns using co-occurrence information. In: Tseng VS, Ho TB, Zhou ZH, Chen ALP, Kao HY (eds) Advances in knowledge discovery and data mining. Springer, Cham, pp 40–52
9. Fournier-Viger P, Lin JCW, Kiran RU, Koh YS, Thomas R (2017) A survey of sequential pattern mining. Data Science and Pattern Recognition 1(1):54–77
10. Fumarola F, Lanotte PF, Ceci M, Malerba D (2016) cloFAST: closed sequential pattern mining using sparse and vertical id-lists. Knowl Inf Syst 48(2):429–463
11. Gomariz A, Campos M, Marin R, Goethals B (2013) claSP: an efficient algorithm for mining frequent closed sequences. In: Pei J, Tseng VS, Cao L, Motoda H, Xu G (eds) Advances in knowledge discovery and data mining, vol 7818. Springer, Heidelberg, pp 50–61
12. Guralnik V, Karypis G (2004) Parallel tree-projection-based sequence mining algorithms. Parallel Comput 30(4):443–472
13. Han J, Pei J, Mortazavi-Asl B, Chen Q, Dayal U, Hsu MC (2000) FreeSpan: frequent pattern-projected sequential pattern mining. In: Proceedings of the Sixth ACM SIGKDD international conference on knowledge discovery and data mining, pp 355–359
14. Han J, Pei J, Yin Y, Mao R (2004) Mining frequent patterns without candidate generation: a Frequent-Pattern tree approach. Data Min Knowl Disc 8(1):53–87
15. Hoang T, Le B, Tran MT (2017) Distributed algorithm for sequential pattern mining on a large sequence dataset. In: Proceedings of the Ninth international conference on knowledge and systems engineering, pp 18–23
16. Huang JW, Lin SC, Chen MS (2010) DPSP: distributed progressive sequential pattern mining on the cloud. In: Zaki MJ, Yu JX, Ravindran B, Pudi V (eds) Advances in knowledge discovery and data mining. Springer, Berlin, pp 27–34
17. Huynh B, Vo B, Snasel V (2017) An efficient method for mining frequent sequential patterns using multi-Core processors. Appl Intell 46(3):703–716
18. Kieu T, Vo B, Le T, Deng ZH, Le B (2017) Mining top-k co-occurrence items with sequential pattern. Expert Syst Appl 85(1):123–133
19. Mabroukeh NR, Ezeife CI (2010) A taxonomy of sequential pattern mining algorithms. ACM Comput Surv 43(1):3:1–3:41
20. Masseglia F, Cathala F, Poncelet P (1998) The PSP approach for mining sequential patterns. In: Proceedings of the Second European symposium on principles of data mining and knowledge discovery, Lect Notes Comput Sci, vol 1510, pp 176–184

21. Miliaraki I, Berberich K, Gemulla R, Zoupanos S (2013) Mind the gap: large-scale frequent sequence mining. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data, pp 797–808
22. Pei J, Han J, Mortazavi-Asl B, Wang J, Pinto H, Chen Q, Dayal U, Hsu MC (2004) Mining sequential patterns by pattern-growth: the PrefixSpan approach. IEEE Trans Knowl Data Eng 16(11):1424–1440
23. Salvemini E, Fumarola F, Malerba D, Han J (2011) FAST sequence mining based on sparse Id-Lists. In: Kryszkiewicz M, Rybinski H, Skowron A, Ras ZW (eds) Foundations of intelligent systems. Springer, Berlin, pp 316–325
24. Shintani T, Kitsuregawa M (1998) Mining algorithms for sequential patterns in parallel : hash based approach. In: Wu X, Kotagiri R, Korb KB (eds) Research and development in knowledge discovery and data mining, vol 1394. Springer, Berlin, pp 283–294
25. Srikant R, Agrawal R (1996) Mining sequential patterns: generalizations and performance improvements. In: Proceedings of the Fifth international conference on extending database technology, vol 1057, pp 3–17
26. Wang J, Huang JL, Chen YC (2016) On efficiently mining high utility sequential patterns knowledge information systems. https://doi.org/10.1007/s10115-015-0914-8
27. Wang X, Wang J, Wang T, Li H, Yang D (2010) Parallel sequential pattern mining by transaction decomposition. In: Proceedings of the Seventh international conference on fuzzy systems and knowledge discovery, pp 1746–1750
28. White T (2015) Hadoop: The Definitive guide, fourth edn O'Reilly Media
29. Yang Z, Kitsuregawa M (2005) LAPIN-SPAM: an improved algorithm for mining sequential pattern. In: Proceedings of the 21st international conference on data engineering
30. Yang Z, Wang Y, Kitsuregawa M (2007) LAPIN: Effective sequential pattern mining algorithms by last position induction for dense databases. In: Kotagiri R, Krishna PR, Mohania M, Nantajeewarawat E (eds) Advances in databases: concepts, systems and applications, vol 4443. Springer, Berlin, pp 1020–1023
31. Yong-qing W, Dong L, Lin-shan D (2012) Distributed prefixspan algorithm based on MapReduce. In: Proceedings of 2012 internatioanl symposium on information technology in medicine and education, pp 901–904
32. Yu X, Liu J, Liu X, Ma C, Li B (2015) A MapReduce reinforced distributed sequential pattern mining algorithm. In: Wang G, Zomaya A, Martinez G, Li K (eds) Algorithms and architectures for parallel processing, vol 9529. Springer, Cham, pp 183–197
33. Zaki MJ (2001) Parallel sequence mining on Shared-Memory machines. J Parallel Distrib Comput 61(3):401–426
34. Zaki MJ (2001) SPADE: An efficient algorithm for mining frequent sequences. Mach Learn 42(1-2):31–60

Sumalatha Saleti is currently a PhD student in the department of Computer Science and Engineering, National Institute of Technology, Warangal, India. Her research areas of interest include Data mining, Big data analytics and Cluster computing.

Dr. R. B. V. Subramanyam is a Professor in the department of Computer Science and Engineering, National Institute of Technology, Warangal, India. He received his Master of Technology and Doctor of Philosophy degrees from Indian Institute of Technology, Kharagpur. His areas of interest include Data mining, Distributed data mining, Graph databases, Fuzzy data mining, Big data analytics, Cluster computing, Pattern recognition, High performance computing, Soft computing, Outlier analysis.