Received July 10, 2017, accepted August 1, 2017, date of publication August 17, 2017, date of current version September 19, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2739749

An Efficient Parallel Method for Mining Frequent Closed Sequential Patterns

BAO HUYNH[1,2,3], BAY VO[4,5], (Member, IEEE), AND VACLAV SNASEL[3]
[1]Center for Applied Information Technology, Ton Duc Thang University, Ho Chi Minh City 700000, Vietnam
[2]Faculty of Information Technology, Ton Duc Thang University, Ho Chi Minh City 700000, Vietnam
[3]Faculty of Electrical Engineering and Computer Science, VŠB-Technical University of Ostrava, Ostrava 70800, Czech Republic
[4]Institute of Research and Development, Duy Tan University, Da Nang 550000, Vietnam
[5]College of Electronics and Information Engineering, Sejong University, Seoul 143747, Republic of Korea

Corresponding author: Bay Vo (bayvodinh@gmail.com)

ABSTRACT Mining frequent closed sequential pattern (FCSPs) has attracted a great deal of research attention, because it is an important task in sequences mining. In recently, many studies have focused on mining frequent closed sequential patterns because, such patterns have proved to be more efficient and compact than frequent sequential patterns. Information can be fully extracted from frequent closed sequential patterns. In this paper, we propose an efficient parallel approach called parallel dynamic bit vector frequent closed sequential patterns (pDBV-FCSP) using multi-core processor architecture for mining FCSPs from large databases. The pDBV-FCSP divides the search space to reduce the required storage space and performs closure checking of prefix sequences early to reduce execution time for mining frequent closed sequential patterns. This approach overcomes the problems of parallel mining such as overhead of communication, synchronization, and data replication. It also solves the load balance issues of the workload between the processors with a dynamic mechanism that re-distributes the work, when some processes are out of work to minimize the idle CPU time.

INDEX TERMS Data mining, dynamic bit vectors, dynamic load balancing, multi-core processors, closed sequential patterns.

I. INTRODUCTION

Mining sequential patterns is a core problem in data mining, which was first introduced by Agarwal and Srikant [1]. It has been applied to various domains, such as market basket analysis [1], weblog analysis [12], [37], prediction [24] and bioinformatics analysis [27], [40].

Mining sequential patterns generates an exponential number of patterns when the database contains long sequences which need high computational cost in both time and space. Therefore, in some cases, mining closed sequential patterns is a better solution than mining the complete set of sequential patterns because it is more compact while information can still be fully extracted. A closed sequential pattern is a sequential pattern having no supersequence with the same support and mining of frequent closed sequential patterns has not been extensively studied as the general mining of frequent sequential patterns. Although some algorithms have been proposed, such as CloSpan [38], BIDE [36], ClaSP [11] and CloFS-DBV [32], their performance is not good for processing on large databases. CloSpan is not scalable because it works under candidate maintenance-and-test paradigm. BIDE follows a strict depth-first search order to generate the closed sequential patterns; it does not need to keep track of any single historical frequent closed sequential (or candidate) patterns for a new pattern's closure checking. ClaSP uses a heuristic to prune non-closed sequences. CloFS-DBV uses dynamic bit vector (DBV) structure and a vertical data format to mine frequent closed sequences. However, the computational cost of these algorithms is still high because they have to explore a huge search space. Moreover, they require very expensive computations, especially for long sequence databases.

To solve this problem, developing parallel methods is necessary. Parallel processing has been attracted a lot of research and widely applied to improve processing speed for various problems. In that, multi-core processors architectures have been used to speed up processing, but there is no existing method that uses multi-core processors architecture for FCSP mining. Some of mining methods are used multi-core processors architectures include

PGP-mc [18], GapMis-OMP [8], SW [28]. Also based on multi-core processors architecture, two methods for mining FSPs were proposed, PIB-PRISM [15] used prime theory and pDBV-SPM [17] used dynamic bit vector data structure.

Besides speed-up processing, a multi-core system requires software programmers to make efficient use of the multiple computing cores. These challenges include determining how to divide applications into separate tasks and minimizing CPU idle time. Data partition mechanism can be accessed by the tasks running on separate cores and these tasks must be balanced workload.

Although there have been many parallel methods based on distributed system or shared memory system for mining frequent itemsets [42], frequent closed sequential patterns [7], [43], sub-graph mining [31], there are no methods for mining frequent closed sequential pattern on multi-core processors architecture.

In this paper, we propose an efficient algorithm, called pDBV-FCSP (Parallel Dynamic Bit Vector Frequent Closed Sequential Patterns), for mining frequent closed sequential patterns. This method overcomes disadvantages of existing methods for parallel mining. They do not need to synchronize because the communication between computing nodes are eliminated. They do not require data transfer among processing units either and avoid data replication. The main contributions of this paper are summarized as follows:

1. We propose the first parallel method for mining closed sequential patterns using multi-core processors architecture and a DBV data structure.

2. We partition the work into multiple independent tasks to minimize the overhead of inter-processor communication.

3. Load balance of the workload among cores using a dynamic mechanism is re-distributed the work when some processes are out of work.

4. We early perform closure checking of prefix sequences to prune infrequent and non-closed sequences.

The rest of the paper is organized as follows. Section II gives the problem definition and related works. Section III summarizes the DBV data structure and multi-core processors architecture. The pDBV-FCSP algorithm is proposed in Section IV. Experimental results are discussed in Section V. Finally, conclusions and ideas for future work are given in Section VI.

II. BACKGROUND

A. PRELIMINARIES

Let $I = \{i_1, i_2, \ldots, i_m\}$ be a set of m distinct attributes, also called *items*. An itemset X is a set of items such that $X \subseteq I$, itemset X is called k-itemset or length k if it contains k items. Each itemset is given in brackets, the brackets are omitted for itemsets that contain only a single item. A sequence $S = \langle I_1 I_2 \ldots I_n \rangle$ is an ordered list of itemsets, where $I_k \subseteq I$ $(1 \leq k \leq n)$. A sequence of length l is called a l-sequence. A sequence $\delta = (b_1\ b_2 \ldots b_m)$ is called a subsequence of sequence $\lambda = (a_1\ a_2 \ldots a_n)$ and λ is a super sequence of δ, denoted as $\delta \subseteq \lambda$, if there exist integers $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ such that $b_k \subseteq a_{ik}$, $1 \leq k \leq n$. For example, the sequence $\langle H(GK) \rangle$ is a 3-sequence of size 2 and it is a subsequence of $\langle (GH)M(GKL) \rangle$, but $\langle (GH)M \rangle$ is not a subsequence of $\langle GHM \rangle$.

A sequence database DB is a set of sequences, denoted $DB = \{S_1, S_2, \ldots, S_n\}$, where n is the number of sequences, each sequence has the form (sid, S), where sid is a unique sequence identifier and S is a ordered list of itemsets. A pattern is a subsequence of ordered itemsets, and each itemset in a pattern is called an element. The absolute support of a sequence p in a database DB is defined as the total number of sequences in DB that contain p, denoted as $\sigma(p) = |\{S_i \in DB | p \subseteq S_i\}|$. The relative support of p is ratio (in percent) of the sequences that contain p to the total number of sequences in the database.

$$\sigma(p) = \frac{|\{S_i \in DB | p \subseteq S_i\}|}{|DB|}$$

Sometimes, absolute and relative supports are used interchangeably. A sequence p is said to be a frequent sequence pattern if its support is greater or equal to a given support threshold *minsup*, that is $\sigma\ (p) \geq minsup$. A frequent sequence λ is *closed* if it is not a subsequence of any other frequent sequence with the same support, i.e., there does not exist δ such that $\lambda \subseteq \delta$ and $\sigma\ (\delta) = \sigma(\lambda)$. The problem of mining FCSPs is to find all frequent closed sequences in the database DB.

FCSP $= \{\lambda \in SP | \nexists \delta \colon \lambda \subseteq \delta \wedge sup(\lambda) = sup(\delta)\}$, where SP is the set of sequential patterns.

Example 1: The sequence database DB in Table 1 has a set of items is $\{A, B, C, D, E, F\}$. Assume that $minsup = 3$ (50%), if all frequent sequences patterns in database DB are mined with the given *minsup*, we have result of 30 sequences, FSP $= \{A : 6,\ B : 5,\ C : 5,\ D : 3,\ E{:}5, AA{:}4, BB{:}4, AB{:}5, BA{:}5, (CE){:}3, BE{:}4, (AE){:}3, AE{:}5, AD{:}3, CC{:}3, (BC){:}3, BC{:}4, CB{:}3, AC{:}5, CA{:}3, CAC{:}3, BAB{:}4, BAC{:}4, BCB{:}3, ABA{:}3, AAC{:}3, A(AE){:}3, ABC{:}3, ABE{:}3, A(CE){:}3\}$.

TABLE 1. Example of sequence database.

SID	Sequence database
S_1	*CAAC(CADEF)D*
S_2	*AB(AE)CB*
S_3	*(BC)CABC*
S_4	*ABBCA(BCE)*
S_5	*BA(BCE)D*
S_6	*AB(ADE)A*

In contrast, mining frequent closed sequential patterns has only 19 sequences. *FCSP* $= \{A{:}6, BE{:}4, AE{:}5, AB{:}5, BA{:}5, AD{:}3, AA{:}4, AC{:}5, (BC){:}3, AAC{:}3, CAC{:}3, A(AE){:}3, ABC{:}3, BCB{:}3, A(CE){:}3, ABA{:}3, ABE{:}3, BAB{:}4, BAC{:}4\}$, so FCSP is more compact than FSP in general.

Definition 1: Sequence λ is a prefix of δ if and only if $a_i = b_i$ for all $1 \leq j \leq n$. After eliminating the prefix part λ of sequence δ, the remainder of δ is a postfix of δ. From the above definition, we know that a sequence of size k has $(k-1)$ prefixes. For example, a sequence $\langle A(CD)E\rangle$ has 2 prefixes: $\langle A\rangle$ and $\langle A(CD)\rangle$. Therefore, $\langle (CD)E\rangle$ is the postfix for prefix $\langle A\rangle$, and $\langle E\rangle$ is the postfix for prefix $\langle A(CD)\rangle$.

Definition 2: Let S_a be a sequence. Then, $sub_{k,h}(S_a)$ $(k \leq h)$ is defined as a substring of length $(h-k+1)$ from position k to position h of S_a. For example, $sub_{2,4}(CBDAD)$ is BDA and $sub_{1,3}(CBDBA)$ is CBD.

Definition 3: Let λ and δ be two frequent 1-sequences. $\{t_\lambda, p_\lambda\}$ and $\{t_\delta,\ p_\delta\}$ are the transaction and position of sequences λ and δ, respectively. Two forms of sequence extension.

$$\text{Itemset extension:} \langle(\lambda\delta)\rangle\{t_\delta, p_\delta\},\ \text{if } (\lambda \leq \delta) \wedge (t_\lambda = t_\delta) \wedge (p_\lambda = p_\delta) \quad (1)$$

$$\text{Sequence extension:} \langle\lambda\delta\rangle\{t_\delta, p_\delta\},\ \text{if } (t_\lambda = t_\delta) \wedge (p_\lambda < p_\delta) \quad (2)$$

For example, consider database *DB* in Table 1, *A* and *E* are two frequent 1-sequences. Itemset extension produces (AE) because $A \leq E$ and $(t_A = t_E)$ is 1 and $(p_A = p_E)$ is 5. Sequence extension produces AE because $(t_A = t_B)$ is 1 and the position of sequence E is $\{5\}$, the position of sequence A are $\{2, 3, 5\}$, which is smaller than all positions of sequence E.

Definition 4: Let λ and δ be two frequent k-sequences $(k \geq 1)$, $x = sub_{k,k}(\lambda)$, and $y = sub_{k,k}(\delta)$. $\{t_\lambda,\ p_\lambda\}$ and $\{t_\delta, p_\delta\}$ are the transaction and position of sequences λ and δ, respectively. Two forms of sequence extension.

Itemset extension:

$$\lambda + \delta_i = sub_{1,k-1}(\lambda)(xy)\{t_\lambda, p_\lambda\},\ \text{if } (x \leq y) \wedge (t_\lambda = t_\delta) \wedge (p_\lambda = p_\delta) \wedge (sub_{1,k-1}(\lambda) = sub_{1,k-1}(\delta)) \quad (3)$$

Sequence extension:

$$\lambda + \delta_s = \lambda y\{t_\lambda, p_\lambda\},\ \text{if } (t_\lambda = t_\delta) \wedge (p_\lambda < p_\delta) \wedge (sub_{1,k-1}(\lambda) = sub_{1,k-1}(\delta)) \quad (4)$$

For example, *DE* and *DF* are two frequent 2-sequences. Sequence extension produces *DED, DEE, DEF* and itemset extension produces $D(EF)$ because *DE* and *DF* have the same prefix (D).

Definition 5: Let $S = (p_1 p_2 \ldots p_n)$. An item p_k can be added to one of three positions of sequence S for extension.

$$S^* = (p_1 p_2 \ldots p_n p_k) \wedge (\sigma\ (S^*) = \sigma\ (S)) \quad (5)$$

$$\exists i\ (1 \leq i < n) : S^* = (p_1 p_2 \ldots p_i p_k \ldots p_n) \wedge (\sigma\ (S^*) = \sigma\ (S)) \quad (6)$$

$$S^* = (p_k p_1 p_2 \ldots p_n) \wedge (\sigma(S*) = \sigma(S)) \quad (7)$$

In (5), S^* is called a forward extension sequence and item p_k is called a forward extension because it appears after p_n. In (6) and (7), S^* is called a backward extension sequence and item p_k is called a backward extension because it appears before p_n. For example, sequence *BE*:4 is a forward extension of sequence *B*:4 because sequence *E* is extended after sequence *B*. Sequence *EBE*:2 is a backward extension of sequence *EE*:2 because sequence *B* is extended in the middle of sequence *EE*.

Definition 6: Consider $S = (p_1\ p_2 \ldots p_n)$, the starting position of sequence S is the position of the first appearance of itemset p_1. For example, in the sequence $AB(BC)AD$, the starting position of sequence (BC) is 3, and sequence ABD is 1.

B. RELATED WORKS

Sequential pattern mining was first proposed by Agrawal and Srikant in 1995 [1], the goal of it is to discover the full set of SPs from a sequence database, especially from large databases. The basic method for SP mining is candidate-generation-and-test strategy based on the Apriori-property, stated as follows: "*every nonempty subsequence of a sequential pattern is a sequential pattern*".

The general idea of all existing methods is to begin from short sequences and then extend them to gain long sequences. Three main types for existing methods can be categorized as following.

1) HORIZONTAL METHODS

A sequence database in the horizontal format is a database where each row is a transaction in the form *sid-itemset*, where *sid* is a sequence ID and *itemset* is a set of items. Some particular algorithms use this method were proposed: AprioriAll [1], GSP [2] and PSP [20]. In this type, the runtime and memory usage are high because the database must be scanned many times.

2) VERTICAL METHODS

A sequence database in vertical format is a database where each row has a transaction in the form $\langle item,\ sid\rangle$, where *sid* is a set of sequence IDs containing *item.* The main advantage of this type of method is that it reduces the number of database scans. Some representative approaches are SPAM [4], SPADE [41], PRISM [10], PIB-PRISM [15], pDBV-SPM [17].

3) PROJECTION METHODS

Projection methods are hybrid methods between horizontal and vertical methods. The general idea is to examine only the prefix subsequences and project only their corresponding postfix subsequences into projection databases. Some algorithms based on these methods are FP-growth [14], PrefixSpan [26], an extension of FreeSpan [13], IMSR_PreTree [33] and MNSR_PreTree [25]. Sequential patterns are grown by exploring only local frequent patterns in projection database. The projection is based only on frequent prefixes instead of projecting sequence databases because any frequent subsequence can always be found by growing a frequent prefix. The prefix tree architecture is efficient for organizing and storing candidate sequences.

To reduce the computational cost, instead of mining a full set of frequent patterns, many algorithms for mining frequent closed sequential patterns have been proposed. Popular algorithms for closed sequential pattern mining are CloSpan [38], ClaSP [11], BIDE [36], and CloFS-DBV [32]. CloSpan uses candidate-generation-and-test strategy and combines a hash-index structure with a tree structure for storing sequences. With techniques Common Prefix and Backward Sub-Pattern, this algorithm reduces the search space by prunes the patterns. The ClaSP algorithm uses a heuristic to prune non-closed sequences, this algorithm maintains previous candidates to test the closure of sequences and remove them later. The main disadvantage of maintenance of candidates is that it requires a lot of memory and explosive the number of test candidates.

BIDE uses bi-directional extension techniques to examine frequent closed patterns as candidates before extending a sequence. It reduces mining time by uses a BackScan process to determine candidates that cannot be extended sequence and pseudo projection techniques to reduce database storage space and is efficient for low support thresholds. However, this method is not efficient because it has to project and scan databases many times for each prefix. In addition, using dynamic bit vector (DBV) [35] data structure combined with location information in the structure of the transaction, CloFS-DBV [36] perform prunes of prefix sequences early, checks the backward-extension and forward-extension quickly based on CloFS-DBVPattern structure.

All FSP and FCSP mining methods are implemented based on single-task processing and in a sequential maner. Hence, they are very time-consuming for large databases, especially long or dense databases. To improve performance, some researchers have applied parallel computing is to cut down on the execution time of processor. Based on distributed memory system, some parallel methods such as pSPADE [41], which is based on SPADE [40], Par-CSP [7] and Par-ClosP [43] were proposed. In modern processor architecture, multi-core processors [3], [30] allow for multiple tasks to be executed in parallel to enhance performance. A multi-core processor has many advantages especially for multitasking computing power of system. Some parallel algorithms were proposed based on multi-core processor architectures, such as cache-conscious optimization and lock-free parallel [19] and an effective load balancing strategy [39] for frequent itemsets mining. Parallel mining has been applied to closed itemset mining [19], [21] [29], [39] correlated pattern mining [5], parallel method for CAR mining [23], parallel method for sub-graph mining [16], generic pattern mining [22] and frequent sequential pattern mining [15], [17].

Although the performance of these approaches is better than serial counterpart methods, these approaches still require large storage space and produce redundant results, especially for large or long sequence databases.

III. DATA STRUCTURE FOR PARALLEL MINING

A. MULTI-CORE PROCESSOR ARCHITECTURE

A multi-core processor is a single computing component that has two or more independent cores in the same physical package [30], [34], with each core having its own resources. Multi-core processors allow executed multiple tasks simultaneously to increase performance.

A multi-core processor may have a separate L1 cache and execution unit for each core, while it has a shared L2 cache for the entire processor, this is make the best use of the resources and to make inter-core communication efficient and more resources will be shared between the cores on the die. If multiple processes run on different cores of the same physical package and if they share data that fit in the cache, then the shared last-level cache between cores will minimize the data duplication. Therefore, it is more efficient in communication.

Multi-core processors are an improvement over the deficiencies of single-core processors. Multi-core processors can perform more works in parallel on separate operations, while reducing the power consumption and dissipate the heat [7]. Applications of multi-core processors are to speed up the work of operating systems and to support multithreading. Fig. *1* shows an example of a dual-core, dual-processor system.

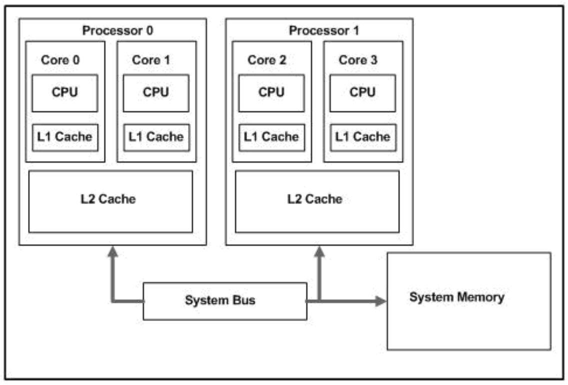


FIGURE 1. A dual-core, dual-processor system.[1]

The main advantage of multi-core processors is lessening the heat coming off CPU and to significantly increase the speed of processing while it is cheaper than multi-processor system so it widely used in many fields including embedded, network, digital signal processing, and graphics.

The present study proposes a method for parallel mining FCSPs to improving the efficiency of systems and reducing computational cost based on a multi-core processor architecture.

B. BIT VECTORS

A bit vector is an array of bits that compactly stores bits. A bit vector is built to represent the positions of an item *X*

[1]https://software.intel.com/en-us/articles/software-techniques-for-shared-cache-multi-core-systems

appearing in a sequence S. If the i-th itemset in S contains item X, then the i-th bit in the bit vector is 1; otherwise, it is 0.

However, the sizes of bit vectors for itemsets are always equal to the number of transactions in the database. It is difficult to store them all in main memory when the number of transactions is large. In addition, a lot of time and memory are needed for computing the intersection between bit vectors. The bit vector of an itemset with many '0' bits can be shortened to reduce storage space and computation time. To address this issue, dynamic bit vector (DBV) were thus proposed [35], it represents a bit vector after removing the '0' bits at the front and end of the vector thus it is more efficiently for itemsets with many '0' bits.

C. DYNAMIC BIT VECTOR DATA STRUCTURE

The DBV data structure [35] is used vertical format layout so it quickly calculates the supports count by perform AND operations on the two DBVs from larger position value of two DBVs. If resulting value is 0, then the position value of the outcome DBV is increased by 1 until the first non-zero resulting value is reached. Next, from the position of non-zero byte, all the resulting bytes by the AND operation are kept unless the last continuous zero bytes.

A DBV consists of two parts:

- *Start bit*: the position of the first appearance of a '1' bit.
- *Bit vector*: a sequence of bits starting from the first non-zero byte to the last non-zero byte.

For example, Table 2 is shows a bit vector of 16 transactions in a sequence database. The bit vector for item i needs 16 bytes because item i exists in transactions 6, 10 and 11. Using the DBV structure, only 8 bytes are required (6 bytes for the bit vector and 2 bytes for position).

TABLE 2. Example of 16-byte bit vector.

0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0

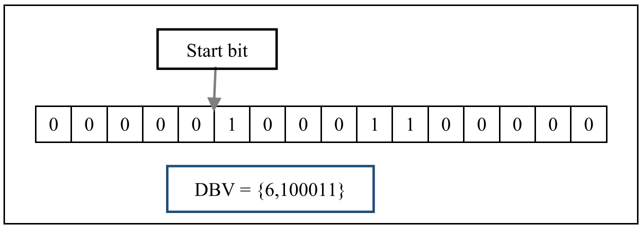
Table 4 is shows the conversion of the bit vector in Table 2 to DBV, the first non-zero byte appears at position 6 so DBV = {6, 100011}.

TABLE 3. Conversion of database DB in Table 1 to DBV format.

Item	*ID*	*Bit-vector*		*Start bit*	*Bit-vector*	*Value*
A	1, 2, 3, 4, 5, 6	111111	*Conversion to DBV*	1	111111	63
B	2, 3, 4, 5, 6	111110		2	11111	62
C	1, 2, 3, 4, 5	011111		1	11111	31
D	1, 5, 6	110001		1	110001	49
E	1, 2, 4, 5, 6	111011		1	111011	59
F	1	000001		1	1	1

Example 2: Consider database DB in Table 1, sequence A exists in transactions 1, 2, 3, 4, 5 and 6, the start bit is 1, the bit vector is 111111, and thus σ $(A) = 6$ because the bit vector has four '1' bits. The similar way, σ $(B) = 5$, σ $(C) = 5$, σ $(D) = 3$, σ $(E) = 5$ and σ $(F) = 1$. Table 3 shows the conversion of database DB in Table 1 to the DBV format.

TABLE 4. Conversion of bit vector to DBV.



IV. PROPOSED ALGORITHM

A. DBV-PATTERN DATA STRUCTURE

The DBV-Pattern [17], [35] data structure combines the DBV structure with a representation of a sequence. It can be briefly summarized as follows: Each DBV-Pattern including sequence S; dynamic bit vector DBV; and list of positions of the occurrence in the sequence of each transaction LP. List positions in the form *startPos*:{*list position*}, where *startPos* is the first appearance of the sequence in each transaction.

Example 3: In database DB, sequence A exists in six transactions. In transaction S_1, sequence A appears at positions {2, 3, 5}. The starting position is 2, so LP is 2:{2, 3, 5}. Similarly, LP is 1:{1, 3} in transaction S_2, LP is 3:{3} in transaction S_3, LP is 1:{1, 5} in transaction S_4, LP is 2:{2} in transaction S_5, and LP is 1:{1, 3, 4} in transaction S_6. Table 5 presents the DBV-Pattern for sequence A in Table 1.

TABLE 5. DBV-Pattern for sequence A.

Sequence	*A*					
Start bit	1					
Value	63					
Index	6	5	4	3	2	1
Positions	1: {1, 3, 4}	2: {2}	1: {1, 5}	3: {3}	1: {1, 3}	2 : {2, 3, 5}

B. PARALLEL DYNAMIC BIT VECTORS FOR MINING FREQUENT CLOSED SEQUENTIAL PATTERNS

This section describes the proposed pDBV-FCSP algorithm, which uses a multi-core processors architecture to mine FCSPs combine with DBV data structure [39].

The *root* of the search tree is labeled as the null sequence. Each node at level k is repeatedly extended by adding one item I to generate a child node at the next level $(k + 1)$-sequence. A $(k + 1)$-sequence can be extended via sequence extension or itemset extension. In itemset extension, an item is added to the last itemset in the pattern. In sequence extension, an item is appended to the sequence pattern to create a new itemset.

In the parallel mining, distributes each branch of the search tree to a single task, the computation at each node becomes an independent task and the overall computation can be parallel by distributing these tasks among the available processor cores which can be processed independently to generate FCSPs. The task parallel formulation distributes the tasks among the processor cores in the following way. First, the tree is expanded using the data-parallel algorithm at level $k + 1$, with $k > 0$. Then, the different nodes at level k are distributed among the processor cores. Once this initial distribution is done, each processor core proceeds to generate the subtrees underneath the nodes to which they have been assigned.

The main challenges in parallel mining is that the candidate generation tree is usually not balanced. This skewness (one subtree is very deep compared to the others) of the tree can reduce the performance of the parallel algorithm. Therefore, dynamically re-distributing the works when some processes are out of work is necessary.

The proposed pDBV-FCSP algorithm uses the DBV data structure and depth first search that relies on dynamic load balancing, where each process can perform a DFS on the subtree, since they are computed independently.

The advantage of pDBV-FCSP is that each task is assigned for searching a branch of the tree and is processed independently. The advantages over using threads are a task can run on multiple cores, requiring less memory and less processing time than threads because a thread runs on only one core and it requires more memory. The operating system has initialization, destruction, and must perform context switching between threads so it requires more processing.

When a process finishes finding all frequent closed sequential patterns in its corresponding part of the tree, it actively requests an unexplored part of the tree from other processes. When a processor becomes idle, it randomly selects a dedicator processor and sends it a work request. The dedicator sends a response indicating whether or not it has additional work. If a response indicates that a dedicator can do any more work, the processor receives nodes to expand, along with the portion of the database associated with those nodes. Otherwise, the processor selects another donor and sends work request to that dedicator. This process continues until every processor completely extended the nodes assigned to it.

Proposition 1 [32]: Consider the prefix $S_p = k_1 k_2 \cdots k_n$. If there exists an item k before the starting position of prefix S_p in each of the transactions containing S_p in sequence database DB, the extension can be pruned by prefix S_p. Consider database DB in Table 1, there exists a sequence E that occurs after A in each transaction that contains prefix E, it is no need to extend prefix E. If we extend prefix E, the results obtained will be absorbed due to the extension of prefix A already containing E and having the same support.

Proposition 2 [32]: If there exists a sequence δ that is a forward-extension or backward-extension of sequence λ, sequence λ is not closed, and λ can be safely absorbed by δ. For example, suppose that $\alpha = BB{:}2$ and $\delta = BCB{:}2$. Then, $BB{:}2$ will be absorbed by BCB because $BB \subseteq BCB$ and $\sigma\ (BB) = \sigma\ (BCB) = 2$.

The main steps of the pDBV-FCSP algorithm are as follows.

1. Convert the sequence database to the DBV-Pattern structure.
2. Identify the frequent 1-sequences.
3. Project the database along each frequent 1-sequences and check the closure of frequent sequences to early eliminate infrequent sequences.
4. Prune the prefix sequences early.
5. Extend sequences.

The mining of FCSPs is divided into separate branches as follows.

1. Build the search tree from the set of frequent 1-sequences.
2. Parallel mine FCSPs.
3. Synchronize results.

The pseudo code of the **pDBV-FCSP** strategy is shown in Fig. 2. The value of *root* is initialized to *NULL*. Next, this procedure finds all frequent 1-sequences from database *DB* that satisfies the *minsup* threshold (lines 3-5). The items in F_1 are sorted in ascending order by support (line 7) to gain more effectively balance of the workload. Next, **pDBV-FCSP** creates new tasks corresponding to each pattern in F_1 (line 10). Each task executes the procedure **FCSP-Ext** (line 11) to extend itemsets and sequences. Tasks run in parallel to generate a partial set of FCSPs. The final set of FCSPs is the union and synchronize of the partial results.

```
Procedure pDBV-FCSP(DB, minsup)
Input Database DB, minsup
Output: All FCSPs that satisfy minsup
1   Begin
2     root = NULL
3     For (each s in DB ) do
4       if ($\sigma(s) \geq minsup$) then
5         $F_1 = F_1 \cup s$
6     End for
7     Sort ($F_1$) in increasing order of support
8     Add $F_1$ to child node of root
9     For (each node i in root) do
10        Create new task $t_i$
11        Call FCSP-Ext (i, minsup)
12    End for
13  End begin
```

FIGURE 2. pDBV-FCSP strategy.

Consider database *DB* in Table 1 and *minsup* = 50%. After this procedure is executed, five frequent 1-sequences are stored, and thus $F_1 = \{A{:}6, B{:}5, C{:}5, D{:}3, E{:}5\}$.

The procedure **FCSP-Ext** is shown in Fig. 3. This procedure expands the search tree by executing procedures **extendItem** (line 6) and **extendSeq** (line 9). For each node in the search tree, the pattern for that node is extended by calling **extendItem** and **extendSeq** to create a new pattern. Before sequence extension, the algorithm tests and eliminates prefixes that cannot extend frequent closed sequences using Proposition 1 (line 4). This process is repeated (line 13) until no frequent closed sequences are generated. Lines 16-22 uses Proposition 2 to check the prefix

S_x. If S_x is not a frequent closed sequence, it will be set to NULL. After each level expansion, the processors communicate between each other to determine whether the work needs to be re-balanced (line 23).

```
Procedure FCSP-Ext (p, minsup)
1  Begin
2  Let list = child nodes of p
3  For each $S_x$ in list do
4      If ($S_x$ is not pruned) then
5          For each $S_y$ in list do
6              If ($\sigma(S_{xy}$ = extendItem($S_x$, $S_y$)) $\geq$ minsup) then
7                  Add $S_{xy}$ to child nodes of $S_x$
8              End if
9              If ($\sigma(S_{xy}$ = extendSeq ($S_x$,$S_y$)) $\geq$ minsup) then
10                 Add $S_{xy}$ to child nodes of $S_x$
11             End if
12         End for
13         Call FCSP-Ext($S_x$, minsup)
14     End if
15     If (checkBackwardExt($S_x$,$S_y$) = true) then
16         $S_x$.isClosed = false
17     Else (checkForwardExt($S_x$,$S_y$) = true) then
18         $S_x$.isClosed = false
19     End if
20     If ($S_x$.isClosed = false) then
21         $S_x$ = NULL
22     End if
23     Load_balancing()
24 End for
25 End begin
```

FIGURE 3. FCSP-Extension procedure.

Consider a pattern with prefix A, the algorithm performs sequence extension to create new frequent closed 2-sequences, pattern with prefix A extends using **extendSeq** by concatenation with A, B, C, D and E to make new patterns AA, AB, AC, AD and AE, respectively, and extends using **extendItem** by concatenation with B, C, D and E to make new patterns (AB), (AC), (AD) and (AE), respectively. It is then checked whether the support count of these patterns satisfies the threshold or not. This process is repeated for each task until all FCSPs are obtained, as shown in Table 6.

TABLE 6. Set of frequent closed sequential patterns from Table 1 with *minsup* = 50%.

No.	Item	FCSPs – Support
1	*A*	*A*:6, *AE*:5, *AB*:5, *AD*:3, *AA*:4, *AC*:5, *AAC*:3, *A(AE)*:3, *ABC*:3, *A(CE)*:3, *ABA*:3, *ABE*:3
2	*B*	*BA*:5, *BE*:4, (*BC*):3, *BAB*:4, *BAC*:4, *BCB*:3
3	*C*	*CAC*:3
4	*D*	∅
5	*E*	∅

V. EXPERIMENTAL RESULTS

To show the effectiveness of the proposed algorithm, the experiments were conducted and performed on a personal computer with an Intel Core i5-6200U 2.3-GHz CPU with 4 cores, 3 MB of L3 cache, and 4 GB of RAM, running 64-bit Windows 10 Pro. The algorithm was implemented in C# and run on .Net Framework 4.5.

The first and second databases (C6T5N1kD10k and T10I4D100K) used for comparison were generated using the IBM synthetic data generator. The third database (Kosarak25k) was provided by Bodon (http://fimi.ua.ac.be/data/). Other databases were provided by Fournier-Viger (http://www.philippe-fournier-viger.com/spmf/). Statistic of the databases is shown in Table 7. The definitions of parameters used to generate the databases are shown in Table 8.

TABLE 7. Databases used in experiments.

Database	#seq	#items	
C6T5N1kD10k	10,000	1,000	Synthetic databases
T10I4D100K	100,000	870	
Kosarak25k	25,000	41,270	Click stream data of a Hungarian online news portal
BMSWebView1	59,601	497	Click stream data from an e-commerce site
BMSWebView2	77,512	3,340	Click stream data from the KDD-CUP 2000
MSNBC	31,790	17	Click stream data from the UCI repository
MSNBC_Full	989,818	17	

TABLE 8. Definitions of parameters for standard databases.

C	Average number of itemsets per sequence
T	Average number of items per itemset
I	Average number of items in maximal sequences
N	Number of distinct items
D	Number of sequences

Some existing methods for parallel mining closed sequential patterns are designed on a distributed memory system and are performed from 4, 8, 16, 32 and 64 nodes as Par-CSP [7] and Par-ClosP [43], each node has a 1GHz Pentium III processor, 1GB main memory. Our proposed method is the first method applied to multi-core processors for mining closed sequential pattern. Therefore, it is very difficult to compare our result with those methods because they are developed in different platforms. Experiments were conducted to compare CloFS-DBV and the proposed pDBV-FCSP for various *minsup* values and the proposed method running on three figures are two, three and four cores. The proposed method is significantly improved the running time compared to the sequential counterpart.

The runtime results are shown in *Fig. 4–Fig. 9* and memory usage results are shown in *Fig. 10–Fig. 15* for the six regarding databases.

Parallel mining improves performance without affected the results. Table 8 shows the mining results of CloFS-DBV and pDBV-FCSP. The set of results of algorithms were always the same for all databases, that proves the correctness of pDBV-FCSP.

We also add a set of results of frequent sequential pattern (FSP) mining for comparison. With these thresholds, the result is similar for all databases among frequent closed sequential pattern (FCSP) and frequent sequential patterns. We just saw that the results are different between FCSP and FSP on the Kosarak25k database and on two databases BMSWebView1 and BMSWebView2 with the lowest threshold.

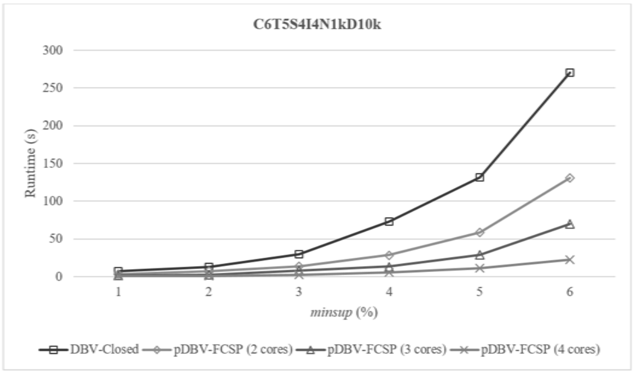


FIGURE 4. **Runtimes on C6T5N1kD10k database.**

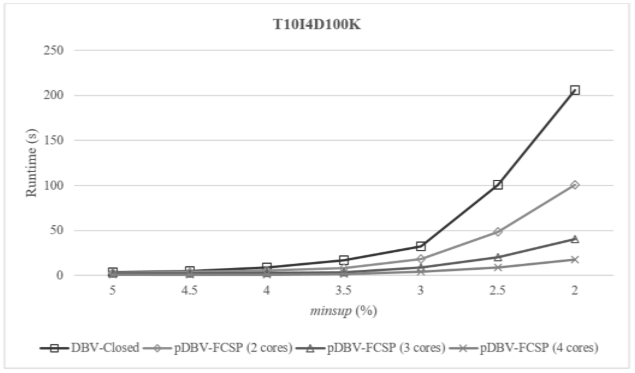


FIGURE 5. **Runtimes on T10I4D100K database.**

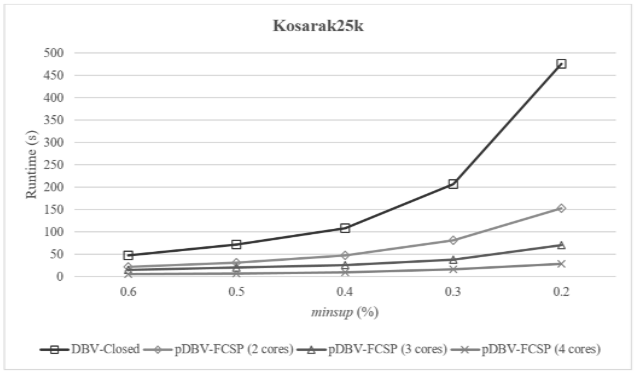


FIGURE 6. **Runtimes on Kosarak25k database.**

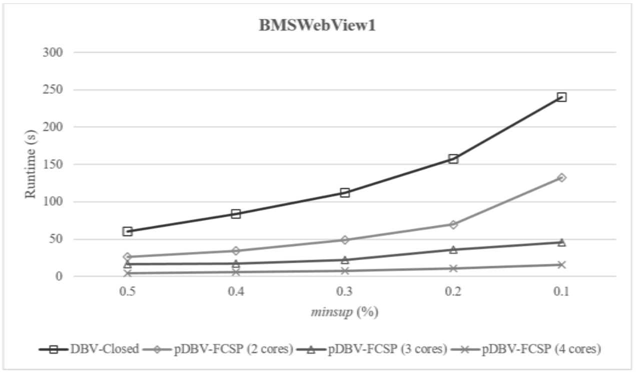


FIGURE 7. **Runtimes on BMSWebView1 database.**

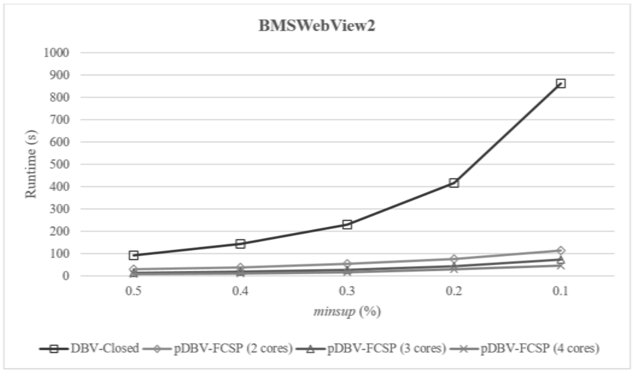


FIGURE 8. **Runtimes on BMSWebView2 database.**

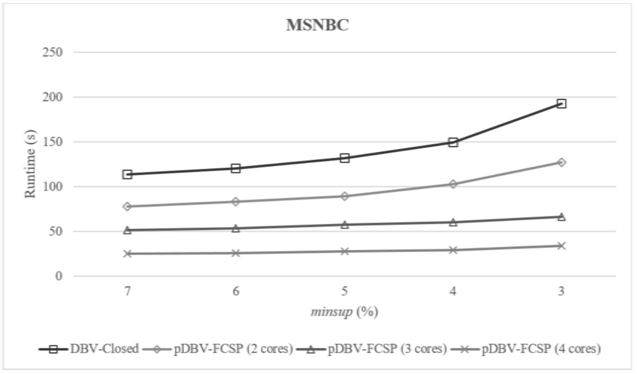


FIGURE 9. **Runtimes on MSNBC database.**

A. RUNTIME

The experimental results show that pDBV-FCSP is faster than CloFS-DBV in most cases, especially, when they are executed on a computer with more cores. With a large *minsup*, pDBV-FCSP is not faster than CloFS-DBV, however, with a small *minsup*, pDBV-FCSP is much faster than CloFS-DBV. In *Fig. 4*, for the C6T5S4I4N1kD10k database, the run-

TABLE 9. The set of results of algorithms for comparison.

BMSWebView1							
minsup(%)	0.5	0.4	0.3	0.2	0.1		
#FSP	201	286	435	798	3991		
#FCSP	201	286	435	798	3974		
BMSWebView2							
minsup (%)	0.5	0.4	0.3	0.2	0.1		
#FSP	408	676	1340	3683	23294		
#FCSP	408	676	1340	3683	22245		
Kosarak25k							
minsup(%)	0.6	0.5	0.4	0.3	0.2		
#FSP	1148	1668	2694	5167	15679		
#FCSP	1026	1458	2248	3972	9005		
MSNBC							
minsup (%)	7	6	5	4	3		
#FSP	721	987	1478	2352	4118		
#FCSP	721	987	1478	2352	4118		
C6T5S4I4N1kD10k							
minsup(%)	6	5	4	3	2	1	
#FSP	83	117	188	281	418	863	
#FCSP	83	117	188	281	418	863	
T10I4D100K							
minsup(%)	5.0	4.5	4.0	3.5	3.0	2.5	2.0
#FSP	10	17	26	40	60	107	155
#FCSP	10	17	26	40	60	107	155

times of CloFS-DBV and pDBV-FCSP were 7.06 s, 3.59 s (two cores), 2.00 s (three cores) and 0.91 s (four cores) seconds, respectively. The runtimes of pDBV-FCSP were always better than the others when executed on four cores while CloFS-DBV increased rapidly when *minsup* was decreased from 6% to 1%. Similar results, the runtimes of pDBV-FCSP were slower than the CloFS-DBV as shown in *Fig. 5* for T10I4D100K database. In *Fig. 6–Fig. 9*, the runtimes of pDBV-FCSP for Kosarak25k, BMSWebView1, BMSWebView2 and MSNBC (a short version) databases were better than CloFS-DBV in most cases, especially with small *minsup* and running on computer that has many cores. When *minsup* was decreased, the runtimes of pDBV-FCSP were always faster.

B. MEMORY USAGE

On all databases, CloFS-DBV is always better than pDBV-FCSP in terms of memory usage, as shown

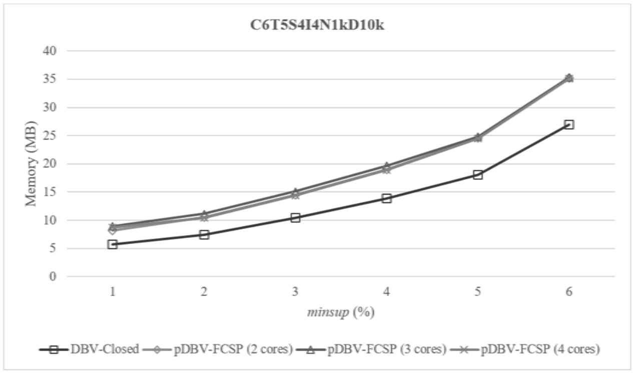


FIGURE 10. Memory usage for C6T5N1kD10k database.

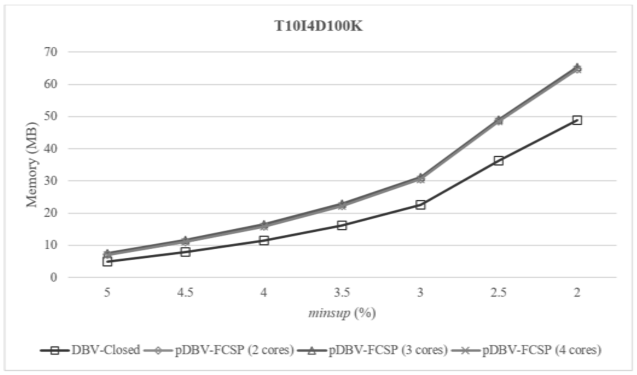


FIGURE 11. Memory usage for T10I4D100K database.

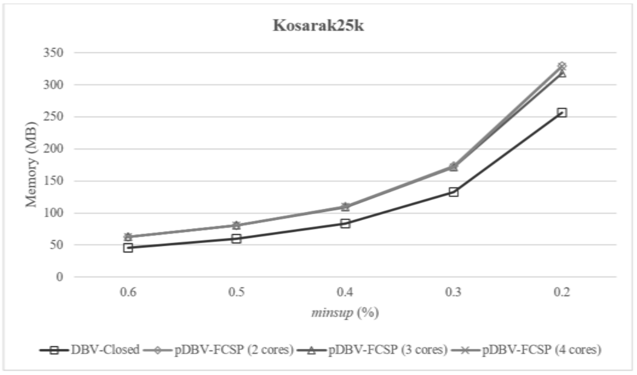


FIGURE 12. Memory usage for Kosarak25k database.

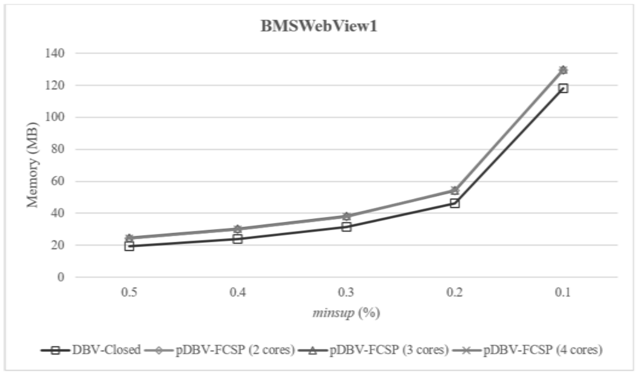


FIGURE 13. Memory usage for BMSWebView1 database.

in *Fig. 10–Fig. 15*. This can be explained as follows. Although, both algorithms used the same DBV data structure for storing sequences information. pDBV-FCSP using more memory usage because parallel processing divided the tasks to be processed into independent branches, needing more memory to store the results. When *minsup* was decreased, more FCSPs were obtained and thus the runtime and memory usage increased. The memory usage of pDBV-FCSP between two, three and four cores was equivalents because the numbers of nodes in the search tree of was the same.

An advantage of pDBV-FCSP is that it helps to balance the search tree using a dynamic mechanism that re-distributes the works when some processes are out of work

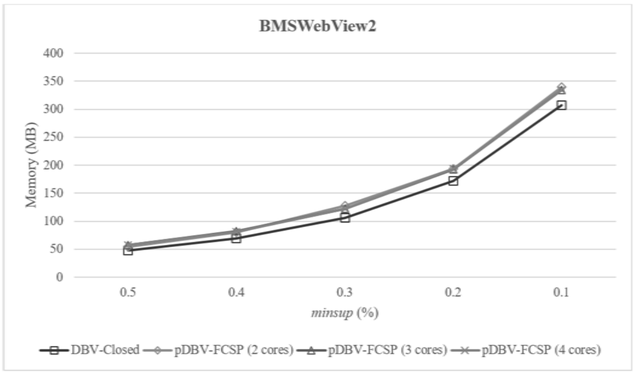


FIGURE 14. Memory usage for BMSWebView2 database.

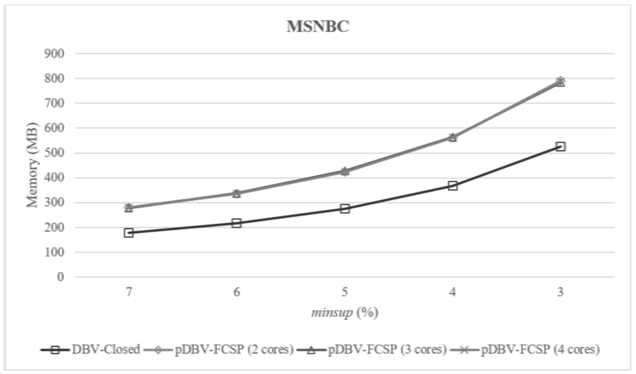


FIGURE 15. Memory usage for MSNBC database.

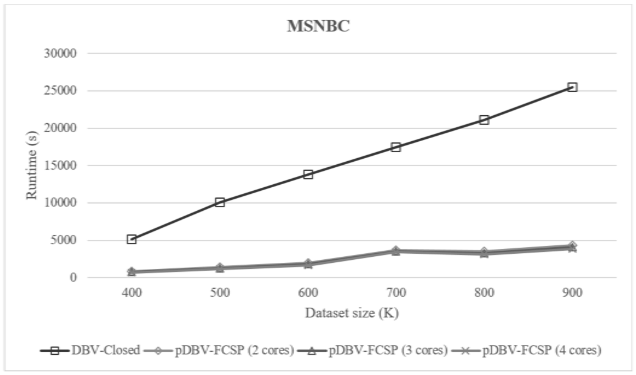


FIGURE 16. Scalability of pDBV-FCSP and CloFS-DBV for MSNBC database with *minsup*=7% and various database sizes.

to minimize the CPU idle time and the set of frequent 1-sequences is sorted in ascending order before task assignment. This strategy balances the search tree for parallel processing, and thus the search times for branches of the search tree are similar.

C. SCALABILITY

In this section, we performed scalability experiments on various number of sequences for MSNBC full database which is the largest one in the experimental databases. The goal of this experiment is to observe the influence of the number of sequences on execution time. The results in *Fig. 16* show that pDBV-FCSP had the best scalability. We tried this database with CM-ClaSP approach proposed by Philippe et al. [9], even the CMClaSP could not run for MSNBC database from 500K sequences on the above computer.

VI. CONCLUSION AND FUTURE WORKS

This article proposed an efficient parallel strategy for mining FCSPs based on multi-core processors architecture and used an efficient DBV data structure for quickly determining support count. With a dynamic load balancing mechanism, the proposed algorithm solved the load balance issues of the workload between processors to minimize the idle CPU time. The experimental results show that the proposed algorithm outperformed the CloFS-DBV algorithm, especially the runtime significantly reduced when the number of cores increased.

In the future, we will study some issues related to maximal patterns, utility pattern and subgraph mining in complex sequence databases using multi-core processor, or hybrid environment that combine between distributed and multi-core processor. We will also study how to use other architectures to improve the efficiency of the mining process.

REFERENCES

[1] R. Agrawal and R. Srikant, "Mining sequential patterns," in *Proc. ICDE*, vol. 95. 1995, pp. 3–14.

[2] R. Agrawal and R. Srikant, "Mining sequential patterns: Generalizations and performance improvements," in *Proc. EDBT*, vol. 96. 1996, pp. 3–17.

[3] A. Vajda, *Multi-core and Many-core Processor Architectures*. Springer, 2011, pp. 9–43. [Online]. Available: http://www.springer.com/gp/book/9781441997388

[4] J. Ayres, J. Gehrke, T. Yiu, and J. Flannick, "Sequential pattern mining using a bitmap representaion," in *Proc. SIGKDD*, vol. 2. 2002, pp. 1–7.

[5] A. Casali and C. Ernst, "Extracting correlated patterns on multicore architectures," in *Proc. CD-ARES*, vol. 13. 2013, pp. 118–133.

[6] L. Chai, Q. Gao, and D. K. Panda, "Understanding the impact of multi-core architecture in cluster computing: A case study with intel dual-core system," in *Proc. CCGRID*, 2007, pp. 471–478.

[7] S. Cong, J. Han, and D. Padua, "Parallel mining of closed sequential patterns," in *Proc. ACM SIGKDD*, vol. 5. 2005, pp. 562–567.

[8] T. Flouri, C. S. Iliopoulos, K. ParkSolon, and P. Pissis, "GapMis-OMP: Pairwise short-read alignment on multi-core architectures," in *Artificial Intelligence Applications and Innovations, IFIP AICT*, vol. 382. Springer, 2012, pp. 593–601.

[9] P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas, "Fast vertical mining of sequential patterns using co-occurrence information," in *Proc. PAKDD*, vol. 14. 2014, pp. 40–52.

[10] K. Gouda, M. Hassaan, and M. J. Zaki, "Prism: An effective approach for frequent sequence mining via prime-block encoding," *J. Comput. Syst. Sci.*, vol. 76, no. 1, pp. 88–102, 2010.

[11] A. Gomariz, M. Campos, R. Marin, and B. Goethals, "ClaSP: An efficient algorithm for mining frequent closed sequences," in *Proc. PAKDD*, 2013, pp. 50–61.

[12] L. K. J. Grace, V. Maheswari, and D. Nagamalai, "Web log data analysis and mining," *Adv. Comput.*, vol. 133, pp. 459–469, Mar. 2011.

[13] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M. C. Hsu, "Freespan: Frequent pattern-projected sequential pattern mining," in *Proc. KDD*, 2000, pp. 355–359.

[14] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," *ACM SIGMOD*, vol. 29, no. 2, pp. 1–12, 2000.

[15] B. Huynh and B. Vo, "Using multi-core processors for mining frequent sequential patterns," *ICIC Exp. Lett.*, vol. 9, no. 11, pp. 3071–3079, 2015.

[16] B. Huynh, D. Nguyen, and B. Vo, "Parallel frequent subgraph mining on multi-core processor systems," *ICIC Exp. Lett.*, vol. 10, no. 9, pp. 2105–2113, 2016.
[17] B. Huynh, B. Vo, and V. Snasel, "An efficient method for mining frequent sequential patterns using multi-Core processors," *Appl. Intell.*, vol. 46, no. 3, pp. 703–716, 2017.
[18] A. Laurent, B. Négrevergne, N. Sicard, and A. Termier, "Efficient parallel mining of gradual patterns on multicore processors," in *Advances in Knowledge Discovery and Management, SCI*, vol. 398. Springer, 2012, pp. 137–151.
[19] L. Liu, E. Li, Y. Zhang, and Z. Tang, "Optimization of frequent itemset mining on multiple-core processor," in *Proc. VLDB*, vol. 7. 2007, pp. 1275–1285.
[20] F. Masseglia, F. Cathala, and P. Poncelet, "The PSP approach for mining sequential patterns," in *Proc. 2nd Eur. Symp. Principles Data Mining Knowl. Discovery (PKDD)*, vol. 1510. 1998, pp. 176–184.
[21] B. Negrevergne, A. Termier, J. F. Méhaut, and T. Uno, "Discovering closed frequent itemsets on multicore: Parallelizing computations and optimizing memory accesses," in *Proc. HPCS*, Jun. 2010, pp. 521–528.
[22] B. Negrevergne, A. Termier, M. C. Rousset, and J. F. Méhaut, "Para miner: A generic pattern mining algorithm for multi-core architectures," *Data Mining Knowl. Discovery*, vol. 28, no. 3, pp. 1–41, 2014.
[23] D. Nguyen, B. Vo, and B. Le, "Efficient strategies for parallel mining class association rules," *Expert Syst. Appl.*, vol. 41, no. 10, pp. 4716–4729, 2014.
[24] M. Norouzi, A. Souri, and M. S. Zamini, "A data mining classification approach for behavioral malware detection," *J. Comput. Netw. Commun.*, vol. 1, pp. 8069672:1–8069672:9, Mar. 2016.
[25] T. T. Pham, J. Luo, T. P. Hong, and B. Vo, "An An efficient method for mining non-redundant sequential rules using attributed prefix-trees," *Engineering Applications of Artificial Intelligence*, vol. 32, pp. 88–99, Jun. 2014.
[26] J. Pei *et al.*, "Mining sequential patterns by pattern-growth: The PrefixSpan approach," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 11, pp. 1424–1440, Nov. 2004.
[27] K. Raza, "Application of data mining in bioinformatics," *Indian J. Comput. Sci. Eng.*, vol. 1, no. 2, pp. 114–118, 2013.
[28] F. Sánchez, F. Cabarcas, A. Ramirez, and M. Valero, "Long DNA sequence comparison on multicore architectures," in *Euro-Par-Parallel Processing*. 2010, pp. 247–259.
[29] B. Schlegel, T. Karnagel, T. Kiefer, and W. Lehner, "Scalable frequent itemset mining on many-core processors," in *Proc. 9th Int. Workshop Data Manage. New Hardw.*, 2013, Art. no. 3.
[30] Y. Solihin, *Fundamentals of Parallel Computer Architecture*. Boca Raton, FL, USA: CRC Press, 2009.
[31] N. Talukder and M. J. Zaki, "Parallel graph mining with dynamic load balancing," in *Proc. IEEE Int. Conf. Big Data*, Dec. 2016, pp. 3352–3359.
[32] M. T. Tran, B. Le, and B. Vo, "Combination of dynamic bit vectors and transaction information for mining frequent closed sequences efficiently," *Eng. Appl. Artif. Intell.*, vol. 38, pp. 183–189, Feb. 2015.
[33] T.-T. van, B. Vo, and B. Le, "IMSR_PreTree: An improved algorithm for mining sequential rules based on the prefix-tree," *Vietnam J. Comput. Sci.*, vol. 1, no. 2, pp. 97–105, 2014.
[34] A. Vajda, "Multi-core and many-core processor architectures," in *Programming Many-Core Chips*. New York, NY, USA: Springer, 2011, pp. 9–43.
[35] B. Vo, T. P. Hong, and B. Le, "DBV-Miner: A dynamic bit-vector approach for fast mining frequent closed itemsets," *Expert Syst. Appl.*, vol. 39, no. 8, pp. 7196–7206, 2012.
[36] J. Wang, J. Han, and C. Li, "Frequent closed sequence mining without candidate maintenance," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 8, pp. 1042–1056, Aug. 2007.
[37] P. Weichbroth, M. Owoc, and M. Pleszkun, "Web user navigation patterns discovery from WWW server log files," in *Proc. FedCSIS*, vol. 12. Sep. 2012, pp. 1176–1177.
[38] X. Yan, J. Han, and R. Afshar, "CloSpan: Mining closed sequential patterns in large databases," in *Proc. SDM*, vol. 3. 2003, pp. 166–177.
[39] K. M. Yu and S. H. Wu, "An efficient load balancing multi-core frequent patterns mining algorithm," in *Proc. TrustCom*, vol. 11. 2011, pp. 1408–1412.
[40] M. J. Zaki, J. T. L. Wang, and H. T. T. Toivonen, "BIOKDD01: Workshop on data mining in bioinformatics," *ACM SIGKDD Explorations*, vol. 3, no. 2, pp. 71–73, 2002.
[41] M. J. Zaki, "SPADE: An efficient algorithm for mining frequent sequences," *Mach. Learn.*, vol. 42, pp. 31–60, Jan. 2010.
[42] M. J. Zaki, "Parallel sequence mining on shared-memory machines," *J. Parallel Distrib. Comput.*, vol. 61, no. 3, pp. 401–426, 2001.
[43] T. Zhu and S. Bai, "A parallel mining algorithm for closed sequential patterns," in *Proc. AINA Workshops*, May 2007, pp. 392–395.

BAO HUYNH received the B.Sc. degree in computer science from the University of Science, Vietnam National University, Ho Chi Minh City, Vietnam, and the M.Sc. degree in computer Science from the Posts and Telecommunications Institute of Technology, Ho Chi Minh City, Vietnam, in 2002 and 2011, respectively. His research interests include data mining and social network analysis.

BAY VO (M'16) received the Ph.D. degree in computer science from the University of Science, Vietnam National University of Ho Chi Minh, in 2011. He was an Associate Professor from 2015. His research interests include association rule mining, classification, incremental mining, distributed databases, and privacy preserving in data mining. He serves as an Associate Editor of the ICIC Express-Letters, Part B: Applications, a member of the review board of the International Journal of Applied Intelligence, and an Editor of the *I*nternational Journal of Engineering and Technology Innovation. He also served as the Co-Chair of several special sessions, such as ICCCI 2012, ACIIDS 2013, 2014, 2015, 2016, KSE 2013, 2014, SMC 2015, as Reviewer of many international journals, such as IEEE-TKDE, KAIS, ESWA, IEEE-SMC: Systems, Information Sciences, Knowledge Based Systems, Soft Computing, and PLOS ONE, the IEEE ACCESS.

VACLAV SNASEL received the M.Sc. degree from Palacky University, Olomouc, Czech Republic, the Ph.D. degree in algebra and geometry from Masaryk University, Brno, Czech Republic. His research and development experience include over 25 years in the Industry and Academia. He was in a multidisciplinary environment involvinrtificial intelligence, multidimensional data indexing, conceptual lattice, information retrieval, semantic web, knowledge management, data compression, machine intelligence, neural network, Web intelligence, data mining and applied to various real-world problems. He has given over ten plenary lectures and conference tutorials in these areas. He has authored or co-authored several refereed journal/conference papers and book chapters. He has authored over 400 papers (147 is recorded at Web of Science). He has supervised many Ph.D. students from Czech Republic, Jordan, Yemen, Slovakia, Ukraine and Vietnam.

From 2001, he has been a Visiting Scientist with the Institute of Computer Science, Academy of Sciences of the Czech Republic. From 2003, he has been the Vice-Dean for Research and Science with the Faculty of Electrical Engineering and Computer Science, VSB-Technical University of Ostrava, Czech Republic. Since 2006, he has been a Full Professor. Before turning into a full-time academic, he was with industrial company, where he was involved in different industrial research and development projects for over eight years.

He is the Editor-in-Chief of two journals, he also serves the editorial board of some reputed International journals. He is actively involved in the International Conference on Computational Aspects of Social Networks; Computer Information Systems and Industrial Management; Evolutionary Techniques in Data Processing series of International conferences. He is a ACM, AMS, and SIAM.