



# Mining inter-sequence patterns with Itemset constraints

Anh Nguyen<sup>1</sup> · Ngoc-Thanh Nguyen<sup>1,2</sup> · Loan T.T. Nguyen<sup>3,4</sup> · Bay Vo<sup>5</sup>

Accepted: 6 February 2023 / Published online: 18 March 2023

© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2023

## Abstract

Nowadays, raw data is rarely used directly. In real world applications, data is often processed, and the necessary knowledge extracted, depending on the purpose of the user. Applying constraints in pattern mining is a major factor in reducing the resulting patterns to help decision support systems work efficiently. In 2018, a constraint-based approach was developed to discover inter-sequence patterns. However, this method only focused on the constraints with single items. The task of discovering constraint-based inter-sequential patterns is our target in this work. We propose the DBV-ISP MIC algorithm, a DBV-PatternList based structure, for mining inter-sequential patterns with itemset constraints. The proposed algorithm utilizes an organized search tree structure stored as dynamic bit vectors to quickly compute the support of patterns. In addition, we also develop a property and, based on it, an improved algorithm is proposed to reduce checking candidates. Finally, we develop the *p*DBV-ISP MIC algorithm as a parallel method of the DBV-ISP MIC algorithm. Empirical evaluations show that DBV-ISP MIC has better performance than the post-processing algorithms in experimental databases and *p*DBV-ISP MIC is better than DBV-ISP MIC with regard to the runtime.

**Keywords** Data mining · Frequent sequence · Frequent inter-sequence · Constraints

## 1 Introduction

Sequential pattern mining is used to reveal a set of frequent patterns from a sequence database  $t$  [1–6] while mining inter-sequence patterns will find the frequent patterns across multiple sequences in the sequence database [7–12].

Among the important tasks in the field of data mining, *inter-sequence pattern mining* is an especially practical one. Its purpose is to find the frequent sequences that are common to many sequences in the database, and to find the relationships between these sequence patterns over time. Therefore, it brings very important and meaningful information for rule

generation to support decision-making, management, and prediction.

### 1.1 Motivation

In recent years, various methods of mining constraint-based (inter)-sequential patterns or rules have been developed [8, 13, 14]. However, most of these solved the problem of mining sequential patterns and mining sequential rules with constraints [13, 14]. The ISP-IC algorithm, suggested by Vo et al. in 2018 [8], focusing in discovering constraint-based inter-sequence patterns. Inter-sequence pattern mining with itemset constraints, however, is not addressed by any of the mentioned approaches. We want to mine patterns such as: “If a customer visits URL1, URL2, URL3 today, then he/she will visit URL2, URL3, URL5 two days later. The constraint is that he/she must visit {URL2, URL3}”. We can solve this problem by applying the proposed algorithm from [7] to mine all mining inter-sequence patterns, after that, we check the constraints for each mined pattern to get the result. However, this work will take more time and memory to mine and store patterns.

As far as we know, the complexity of the mining inter-sequence patterns task is much higher than that of mining frequent sequences. This is because we must mine all

---

✉ Bay Vo  
vd.bay@hutech.edu.vn

<sup>1</sup> Department of Applied Informatics, Wrocław University of Science and Technology, Wrocław, Poland

<sup>2</sup> Faculty of Information Technology, Nguyen Tat Thanh University, Ho Chi Minh City, Vietnam

<sup>3</sup> School of Computer Science and Engineering, International University, Ho Chi Minh City, Viet Nam

<sup>4</sup> Vietnam National University, Ho Chi Minh City, Viet Nam

<sup>5</sup> Faculty of Information Technology, HUTECH University, Ho Chi Minh City, Vietnam

sequences appearing across multiple sequences, and solving the itemset constraints needs more complex processing to check the patterns that satisfy them. Therefore, the major challenge of this study is to address the problem of inter-sequence pattern mining in combination with itemset constraints.

## 1.2 Contributions

In this study, our goal is to solve the task of inter-sequence pattern mining with itemset constraints. This mining task, unlike that of using itemset constraints for mining sequence patterns, requires more complex processing because many candidates are generated during the mining process. Our major contributions are as follows.

1. Based on the EISP-Miner algorithm [7] and a using dynamic bit vector data structure [9], we state the problem of inter-sequence pattern mining in combination with itemset constraints.
2. We then suggest a proposition to help reduce candidate checking during sequence expansion according to the EISP-Miner algorithm, thus reducing the search space for inter-sequence pattern mining with itemset constraints.
3. Next, an algorithm named DBV-ISPMIC is developed to discover constraints-based inter-sequence patterns. A parallel version of the DBV-ISPMIC algorithm, named *p*DBV-ISPMIC algorithm, was also presented to improve the runtime.
4. Finally, we conduct experiments with various databases to evaluate the proposed method.

The rest of this paper is organized as follows: The next section carries out our literature review on sequence pattern mining, sequence pattern mining with constraints, inter-sequence patterns and inter-sequence patterns with constraints. The third section offers the basic concepts and problem statement. The proposed algorithms are presented in the fourth section. In this section, we introduce a structure, namely the DBV-PatternList, and propose the DBV-ISPMIC and *p*DBV-ISPMIC algorithms, to solve the problem. A proposition is also developed to reduce checking candidates in DBV-ISPMIC-IMPROVING. The experimental results are discussed in the fifth section. The last section then presents the conclusion and some directions for future studies.

## 2 Related work

### 2.1 Mining frequent (inter-)sequences

Mining frequent sequences has been explored by many researchers, and many algorithms have thus been developed for mining frequent sequences. AprioriAll was first proposed

by Agrawal and Srikant [2] in 1995 for mining sequential patterns from a sequence database. AprioriAll is based on two basic principles for finding the frequent sequence, candidate generation and testing [2].

However, candidate generation and testing are time-consuming and need many database scans, and thus Zaki et al. introduced the SPADE algorithm in 2001 [3]. This algorithm is based on lattice theory to divide the search space to reveal frequent sequences. It utilizes the vertical database format and equivalence class to find frequent sequences. Therefore, the SPADE algorithm outperforms AprioriAll in most experiments. Han et al. suggested the FreeSpan algorithm in 2000 [4]. This algorithm uses projected sequence databases to confine the search and the growth of subsequence fragments. Pei and Han proposed the PrefixSpan approach in 2001 [5]. The algorithm is based on a specific divide-and-conquer method, which mines frequent sequential patterns based on a prefix database projection (like the FP-Growth algorithm). PrefixSpan can generate frequent sequences with candidate generation. Therefore, it has better execution time than either SPADE or AprioriAll. Thanks to the divide-and-conquer technique, PrefixSpan significantly reduces the size of the database as the length of the common patterns increase, leading to savings in memory and computation time. Gouda et al. used prism encoding to compress sequence IDs to save memory storage and runtime [15, 16].

In 2020, Huynh et al. [17] proposed the CUP algorithm to mine clickstream patterns. In their study, the authors employed a structure called the pseudo-IDList. In addition, the algorithm also adopts the DUB pruning technique to help CUP run faster than state-of-the-art algorithms, by from 13 to 45%. Huynh et al. also proposed efficient parallel algorithms using multi-core processors for mining clickstream patterns [18], as well as a weight-based method for mining clickstream patterns. The method is based on the average weight measure and the authors proposed two algorithms, named CM-WSPADE and Compact-SPADE, for efficient mining of weighted clickstream patterns.

In 2009, a novel method for inter-sequence pattern mining based on vertical database format was introduced by Wang and Lee [7]. The authors developed an ISP-tree to generate candidates that satisfy the minimum support threshold. With the method of inter-sequence pattern mining, the authors defined a new method for the mining sequences which ensures that it can still exploit the traditional sequences like the algorithms SPADE [3], CM-SPADE [6], and PRISM [15, 16], but also mine patterns through multiple sequences in the sequence database. This approach stores sequence IDs (SIDs) and uses them to compute the support of patterns. Therefore, it needs a large amount of memory to store SIDs and time to compute the intersection of SIDs. In 2012, Vo et al. introduced a novel approach, which employed a highly efficient structure call the dynamic bit vector (DBV-PatternList) [9], to replace the pattern-list structure used by the EISP-Miner approach. This method thus significantly reduces

the search space and time in mining inter-sequence patterns, as well as for mining closed inter-sequence patterns [10].

### 2.2 Mining frequent (inter-)sequences with constraints

In 2018, Van et al. proposed the MSPIC-DBV algorithm to mine sequential patterns with itemset constraints [14]. To efficiently discover sequential patterns, MSPIC-DBV adopts the dynamic bit vector and the prefix tree structure. A method for early pruning the candidates to help reduce processing time is also introduced. The authors then continued to improve their work in mining sequential rules with itemset constraints through two new algorithms, namely the MSRIC-R and MSRIC-P, which were introduced in 2021 [13]. The constraints are combined into the rule generating phase for the MSRIC-R algorithm, while the latter algorithm combined it into the pattern mining phase. The authors suggested a technique to integrate the mining process with itemset constraints, to help the algorithm only create constraint-satisfying patterns and thus increase the speed. These methods solved the problem of itemset constraints but can only be applied on mining sequential patterns or sequential rules, so the generated subsequences do not satisfy the inter-sequence mining requirements. Therefore, these algorithms cannot be applied to the problem examined in the current study.

An algorithm named ISP-IC was developed by Le et al. [8] for constraints-based inter-sequence pattern mining, as well as its improvements, iISP-IC and piISP-IC. The authors also applied a parallel processing method to speed up the runtime. As stated in the introduction, ISP-IC, iISP-IC, and piISP-IC focus on the condition of items in the sequence, and we cannot apply this approach to the itemset problem.

## 3 Basic concepts and problem statement

### 3.1 Inter-sequence pattern mining

Let  $t$  be the set of items,  $t = \{u_1, u_2, u_3, \dots, u_m\}$  where  $u_i$  is an item ( $1 \leq i \leq m$ ). A sequence  $s = \langle t_1, t_2, t_3, \dots, t_n \rangle$  is an ordered list of itemsets where  $t_i \subseteq t$  ( $1 \leq i \leq n$ ) is an itemset. A sequential

database  $D = \{s_1, s_2, s_3, \dots, s_w\}$  where  $w = |D|$  is the number of sequences in  $D$  and  $s_i$  ( $1 \leq i \leq w$ ) is a pair of values  $\langle Dat, Sequence \rangle$ , in which  $Dat$  is the property of  $s_i$  used to describe contextual information based on the time of the transaction.

What follows is an example with the sequence database in Table 1. An example of creating a sequential database from a customer dataset.

With  $DAT = 1$  ( $DAT$  is an attribute describing the time of the transaction): First buy item C then buy itemset AB.

Assuming the sequences  $s_1$  and  $s_2$  have  $d_1$  and  $d_2$ , respectively, be their domain attributes ( $DAT$ ). If we take  $d_1$  as the reference point, the span between  $s_2$  and  $s_1$  is defined as  $[d_2 - d_1]$ . The sequence  $s_2$  at domain attribute  $d_2$  with respect to  $d_1$  is called an extend sequence and denoted as  $s_2[d_2 - d_1]$ . Take the database given in Table 1. An example of creating a sequential database from a customer dataset.

For instance – if  $DAT$  from the 1st transaction is used as the reference point, then the extended sequence from the 2nd transaction equals to  $\langle C(ABC)A \rangle [1]$ .

Let  $s[d] = \langle t_1, t_2, t_3, \dots, t_n \rangle [d]$  be a sequence where  $t_i$  is an itemset ( $1 \leq i \leq n$ ) and  $[d]$  is span of  $s.t_i$  associated with  $[d]$  was defined as an extended itemset (e-itemset) denoted by  $\langle t_i \rangle [d]$ . If  $t_i = \{u_1, u_2, u_3, \dots, u_m\}$  where each  $u_i$  ( $1 \leq i \leq m$ ) is an item, then  $u_i$  associated with  $[d]$  is an extended item (e-item) denoted by  $(u_i)[d]$ . For example,  $\langle C(ABC)A \rangle [1]$  has 3 e-itemset  $\langle C \rangle [1]$ ,  $\langle (ABC) \rangle [1]$ ,  $A[1]$  and 3 e-item:  $(C)[1]$ ,  $(A)[1]$ ,  $(B)[1]$ .

**Definition 1 (Megasequence) [7]** Given a list of  $k$  sequences  $\langle d_1, s_1 \rangle, \langle d_2, s_2 \rangle, \dots, \langle d_k, s_k \rangle$  in the sequential database. A megasequence with  $k \geq 1$  is denoted as  $\Psi = s_1[0] \cup s_2[d_2 - d_1] \cup \dots \cup s_k[d_k - d_1]$ .

From the example database shown in Table 1. An example of creating a sequential database from a customer dataset.


With  $maxspan = 1$  and  $DAT = 1$  as the reference point, we have a list of megasequences shown in Table 2.

**Definition 2 [8]** Let  $a = (i_m)[x]$  and  $b = (i_n)[y]$  be 2 e-items.

- (1)  $a = b$  if and only if  $(i_m = i_n) \wedge (x = y)$  and
- (2)  $a < b$ , if and only if  $x < y$  or  $((x = y) \wedge (i_m < i_n))$ .

**Table 1** An example of creating a sequential database from a customer dataset

Date	Time	CustomerId	Transaction
12.12.1998	9:00	1	C
12.12.1998	10:00	2	AB
13.12.1998	9:00	3	C
13.12.1998	14:00	1	ABC
13.12.1998	15:00	4	A
14.12.1998	10:00	6	A
14.12.1998	11:00	4	D
15.12.1998	15:00	5	A
15.12.1998	16:00	4	D



DAT	Sequences
1	$\langle C(AB) \rangle$
2	$\langle C(ABC)A \rangle$
3	$\langle AD \rangle$
4	$\langle AD \rangle$

**Table 2** Converting a sequential database to megasequences

DAT	Sequences	DAT	Megasequences
1	$\langle C(AB) \rangle$	1	$\langle C(AB) \rangle [0] \langle C(ABC)A \rangle [1]$
2	$\langle C(ABC)A \rangle$	2	$\langle C(ABC)A \rangle [0] \langle AD \rangle [1]$
3	$\langle AD \rangle$	3	$\langle AD \rangle [0] \langle AD \rangle [1]$
4	$\langle AD \rangle$	4	$\langle AD \rangle [0]$

For example,  $(A)[0] = (A)[0]$ ,  $(A)[1] < (A)[2]$ , and  $(B)[1] < (D)[1]$ .

**Definition 3 [8]** Given a pattern  $p$ . A function  $sub_{ij}(p)$  is defined as  $(j - i + 1)$  subset e-items of  $p$  from position  $i$  to  $j$ . For example,  $sub_{1,3}(\langle (BC) \rangle [0] \langle (AD)B \rangle [1]) = \langle (BC) \rangle [0] \langle (A) \rangle [1]$  and  $sub_{4,4}(\langle (BC) \rangle [0] \langle (AD)B \rangle [1]) = (D)[1]$ .

**Definition 4 [8]** Given two frequent 1-patterns  $\alpha = \langle u \rangle [0]$  and  $\beta = \langle v \rangle [0]$ .  $\alpha$  is joinable to  $\beta$  in any instance and there are three types of join operation: (1) itemset-join:  $\alpha \cup_i \beta = \{ \langle (uv) \rangle [0] \}$ ; (2) sequence-join:  $\alpha \cup_s \beta = \{ \langle (uv) \rangle [0] \}$ ; (3) inter-join:  $\alpha \cup_t \beta = \{ \langle u \rangle [0] \langle v \rangle [x] \mid 1 \leq x \leq maxspan \}$ . For example, given  $maxspan = 2$ ,  $\langle C \rangle [0] \cup_i \langle D \rangle [0] = \langle (CD) \rangle [0]$ ;  $\langle C \rangle [0] \cup_s \langle D \rangle [0] = \langle CD \rangle [0]$ ; and  $\langle C \rangle [0] \cup_t \langle D \rangle [0] = \{ \langle C \rangle [0] \langle D \rangle [1], \langle C \rangle [0] \langle D \rangle [2] \}$ .

**Definition 5 [8]** Given 2 frequent  $k$ -patterns  $\alpha$  and  $\beta$ ,  $k > 1$ , then  $sub_{k,k}(\alpha) = (u)[i]$ , and  $sub_{k,k}(\beta) = (v)[j]$ .  $\alpha$  is joinable to  $\beta$  if  $sub_{1,k-1}(\alpha) = sub_{1,k-1}(\beta)$  and  $i \leq j$ , which yields three types of join operation: (1) itemset-join:  $\alpha \cup_i \beta = \{ \alpha +_i (v)[j] \mid (i = j) \wedge (u < v) \}$ ; (2) sequence-join:  $\alpha \cup_s \beta = \{ \alpha +_s (v)[j] \mid (i = j) \}$ ; (3) inter-join:  $\alpha \cup_t \beta = \{ \alpha +_t (v)[j] \mid (i < j) \}$ . For example,  $\langle BC \rangle [0] \cup_i \langle BD \rangle [0] = \langle B(CD) \rangle [0]$ ,  $\langle BC \rangle [0] \cup_s \langle BD \rangle [0] = \langle BCD \rangle [0]$ , and  $\langle BC \rangle [0] \cup_t \langle B \rangle [0] \langle D \rangle [2] = \langle BC \rangle [0] \langle D \rangle [2]$ .

**Definition 6 (Prefix) [14]** A pattern  $\beta = \langle b_1 b_2 \dots b_m \rangle$  is called a prefix of pattern  $\alpha = \langle \alpha_1 \alpha_2 \dots \alpha_n \rangle$  if and only if  $b_i = \alpha_i$  for all  $1 \leq i \leq m - 1$ ,  $b_m \subseteq \alpha_m$ ,  $m < n$ . For instance, the prefixes of pattern  $\langle D(BC)B \rangle$  are:  $\langle D \rangle$ ,  $\langle DB \rangle$  and  $\langle D(BC) \rangle$ . Thus, any sequence would be the prefix of its extended sequences based on this definition.

**3.2 Problem statement**

Given a sequence database  $D$ , the minimum support ( $minsup$ ), and a set of constraint itemsets  $IC = \{c_1, c_2, c_3, \dots, c_k\}$ . The

task of inter-sequence pattern mining with an itemset constraint is to discover all frequent sequences  $\alpha = \alpha_1[w_1], \alpha_2[w_2], \dots, \alpha_m[w_m]$  such that  $\exists \alpha_i[w_i] \in \alpha, \exists b_j \in IC: b_j \subseteq \alpha_i$ .

For instance, let  $IC = \{ (C), (E) \}$ , the sequence  $\langle C(AB) \rangle [0] \langle C(ABC)A \rangle [1]$  satisfies the constraint whereas the sequence  $\langle AD \rangle [0] \langle A \rangle [1]$  does not.

**4 Proposed algorithm**

**4.1 DBV-PatternList structure**

Instead of using the PatternList data structure according to the EISP-Miner algorithm [7], we use the DBV-PatternList data structure [9]. This structure uses dynamic bit vectors to store the t-values and p-values of an e-item in the database. It is used to minimize the space and time needed to extend the e-item according to inter-sequence patterns. The DBV-PatternList structure is presented as follows:

- The index of the first non-zero value in the bit vector.
- Bit vector: the array of values after trimming zeroes at the start and end index.

Figure 1 Presents the use of PatternList and DBV-PatternList structures.

e-item	$\langle A \rangle [0]$						
t-value	1	2	3	4	5	7	
	↓	↓	↓	↓	↓	↓	↓
p-value	2	2 3	1	1	1	1	

(a) PatternList structure

e-item	$\langle A \rangle [0]$						
start	1						
bit-vector	15				10		
t-value	1	2	3	4	5	7	
	↓	↓	↓	↓	↓	↓	↓
p-value	2	2 3	1	1	1	1	

(b) DBV-PatternList structure

**Fig. 1** Structures of (a) PatternList and (b) DBV-PatternList

**Fig. 2** The DBV-ISPMIC algorithm

---

**DBV-ISPMIC algorithm**

---

**Input:** A sequence database  $D$ , minimum support ( $minsup$ ), and maximum span ( $maxspan$ ),  $IC = \{c_1, c_2 \dots c_n\}$

**Output:** A complete set of frequent inter-sequence patterns  $FP$  satisfying  $minsup$  and itemset constraints.

1. Scan  $D$  to generate a set of all frequent 1-DBV-PatternList,  $T|NULL$ , as the extended group of the root node of an ISP-tree  $T$ ;
2. **For each** frequent 1-DBV-PatternList  $\alpha_{list}$  in  $T|NULL$  **do**
3. Call  $ISP-Join1(T|NULL, FP, IC, \alpha_{list})$  to get  $T|\alpha_{list}$ ;
4. Call  $ISP-Joink(T|\alpha_{list}, FP, IC)$ ;
5. **End for**
6. Output  $FP$ ;

---

As in structure Fig. 1a, if we store information for an e-item using the PatternList data structure we need to use 26 bytes (12 bytes for t-values and 14 bytes for p-values). But if we use the DBV-PatternList (structure Fig. 1b) data structure we only use 20 bytes (2 bytes for the start position of e-items, 4 bytes for the t-values and 14 bytes for the p-values). Because we have converted the t-values to bit vectors (Definition 6), storage space is reduced.

### 4.2 DBV-ISPMIC algorithm

Our proposed method relies on the DBV-PatternList structure and the DBV-PatternList joining methods. The model of the DBV-ISPMIC algorithm is shown in Fig. 2.

In this algorithm, there are five functions –  $ISP-Join1$ ,  $ISP-Joink$ ,  $ISP-Join1-Extension$ ,  $ISP-Joink-Extension$  and  $Check$  – as shown in Figs. 3, 4, 5, 6 and 7, respectively. The algorithm computes on a given sequential database with  $minsup$ ,  $maxspan$  and a set of itemset constraints defined by the user. The result is a set of frequent inter-sequence patterns that satisfy the conditions of  $minsup$  and itemset constraints. The DBV-ISPMIC algorithm has three main steps, as presented below.

**Step 1** The sequential database is scanned once to find all frequent 1-patterns in which there is not less than the user-specified minimum support threshold  $minsup$  (Fig.

2, line 1). Then, we will create a tree  $T$ , with the root being  $NULL$  and leaves including the found DBV-PatternList. The algorithm goes over all frequent 1-DBV-PatternList results to start expanding the nodes according to the itemset, sequence and inter extension (Fig. 2, line 2).

**Step 2** The algorithm calls the  $ISP-Join1$  function (Fig. 3) for extending an  $\alpha_{list}$  node with the remaining children of the  $T|NULL$  tree (Definition 4). In this function, we will have a new DBV-PatternList in three ways that was extended from the itemset, sequence and inter (based on  $maxspan$  value) extension. We check the DBV-PatternList result with the  $minsup$  condition and itemset constraints. The  $Check$  function in Fig. 7 is used to check if a new DBV-PatternList satisfies the constraint. If satisfied, we will add this DBV-PatternList as a child node of  $T|\alpha_{list}$  (Fig. 5, lines 1–9).

**Step 3** The algorithm calls the  $ISP-Joink$  function (Fig. 4) for extending the child node of the  $\alpha_{list}$  node with the remaining children according to the  $k$ -pattern (Definition 5). In this function, we have been given a new DBV-PatternList in three ways that extend according to itemset, sequence and inter and then we will check whether the DBV-PatternList result satisfies the  $minsup$  condition. The  $Check$  function is used to check if a new DBV-PatternList satisfies the constraint. If it is satisfied,

**Fig. 3** The  $ISP-Join1$  function

---

**Function 1:**  $ISP-Join1(T, FP, IC, \alpha_{list})$

---

1. **For each** frequent 1-DBV-PatternList  $\gamma_{list}$  in  $T|NULL$ , where  $\alpha = \langle u \rangle [0]$  and  $\gamma = \langle v \rangle [0]$ , **do**
2. **For**  $x = 0$  to  $maxspan$  **do**
3.  $ISP-Join1-Extension(T|\alpha_{list}, FP, IC, \alpha_{list}, \gamma_{list}, u, v, x)$ ;
4. **End for**
5. **End for**

---

Fig. 4 The ISP-Joink function

**Function 2:** ISP-Joink( $T, FP, IC$ )

1. **For** each frequent  $k$ -DBV-PatternList  $\beta_{list}$  in  $T|\alpha_{list}$ , where  $sub_{k,k}(\beta)=\langle u \rangle[i]$  **do**
2.     **For** each frequent  $k$ -DBV-PatternList  $\gamma_{list}$  in  $T|\alpha_{list}$ , where  $sub_{k,k}(\gamma)=\langle v \rangle[j]$  **do**
3.         ISP-Joink-Extension( $T|\beta_{list}, FP, IC, \beta_{list}, \gamma_{list}, u, v, i, j$ );
4.     **End for**
5.     **Call** ISP-Joink( $T|\beta_{list}, FP, IC$ );
6. **End for**
7. Delete  $T|\alpha_{list}$  from  $T$ ;

Fig. 5 The ISP-Join1-Extension function

**Function 3:** ISP-Join1-Extension( $T, FP, IC, \alpha_{list}, \gamma_{list}, u, v, x$ )

1. **If** ( $x=0$ ) and ( $u < v$ ) **then**
2.      $\theta_{list} = \alpha_{list} \cup_i \gamma_{list}$ ;
3.     **If**  $support(\theta_{list}) \geq minsup$  and  $Check(\theta_{list}, IC)$  **then** add  $\theta_{list}$  to  $T|\alpha_{list}$  and  $\theta$  to  $FP$ ;
4. **If** ( $x=0$ ) **then**
5.      $\rho_{list} = \alpha_{list} \cup_s \gamma_{list}$ ;
6.     **If**  $support(\rho_{list}) \geq minsup$  and  $Check(\rho_{list}, IC)$  **then** add  $\rho_{list}$  to  $T|\alpha_{list}$  and  $\rho$  to  $FP$ ;
7. **If** ( $x > 0$ ) **then**
8.      $\sigma_{list} = \alpha_{list} \cup_t \gamma_{list}$ ;
9.     **If**  $support(\sigma_{list}) \geq minsup$  and  $Check(\sigma_{list}, IC)$  **then** add  $\sigma_{list}$  to  $T|\alpha_{list}$  and  $\sigma$  to  $FP$ ;

Fig. 6 The ISP-Joink-Extension function

**Function 4:** ISP-Joink-Extension( $T, FP, IC, \beta_{list}, \gamma_{list}, u, v, i, j$ )

1. **If** ( $i=j$ ) and ( $u < v$ ) **then**
2.      $\theta_{list} = \beta_{list} \cup_i \gamma_{list}$ ;
3.     **If**  $support(\theta_{list}) \geq minsup$  and  $Check(\theta_{list}, IC)$  **then** add  $\theta_{list}$  to  $T|\beta_{list}$  and  $\theta$  to  $FP$ ;
4. **If** ( $i=j$ ) **then**
5.      $\rho_{list} = \beta_{list} \cup_s \gamma_{list}$ ;
6.     **If**  $support(\rho_{list}) \geq minsup$  and  $Check(\rho_{list}, IC)$  **then** add  $\rho_{list}$  to  $T|\beta_{list}$  and  $\rho$  to  $FP$ ;
7. **If** ( $i < j$ ) **then**
8.      $\sigma_{list} = \beta_{list} \cup_t \gamma_{list}$ ;
9.     **If**  $support(\sigma_{list}) \geq minsup$  and  $Check(\sigma_{list}, IC)$  **then** add  $\sigma_{list}$  to  $T|\beta_{list}$  and  $\sigma$  to  $FP$ ;

Fig. 7 The function Check

**Function 5:** Check( $\alpha, IC$ )

1. **For** each  $a \in \alpha$  **do**
2.     **For** each  $b \in IC$  **do**
3.         **If**  $b \subseteq a$  **then** //  $a'$  is  $a$  after removing span
4.         **Return** true;
5.     **End for**
6. **End for**
7. **Return** false;

we will add this DBV-PatternList as a child node of  $T|_{\alpha_{list}}$  (Fig. 6, lines 1–9).

### 4.3 Improved DBV-ISPMIC algorithm

We can see that if a pattern satisfies itemset constraints, then the new pattern that is extended from this node with the itemset, sequence, and inter extension will also satisfy the itemset constraints. This helps to reduce the algorithm’s DBV-PatternList node expansion time. We propose a proposition to verify this, as follows.

**Proposition 1** (Checking itemset constraints) Given an inter-sequence  $\alpha$  and a set of itemset constraints  $IC$ , if  $\alpha$  satisfies  $IC$ , then the sequence  $\beta$ , generated from  $\alpha$ , also satisfies constraint  $IC$ .

**Proof** Let  $\alpha = \langle \alpha_1[w_1] \alpha_2[w_2] \dots \alpha_m[w_m] \rangle$ ,  $\beta = \langle \beta_1[w_1] \beta_2[w_2] \dots \beta_u[w_u] \rangle$  be an inter-sequence, whereas each  $\alpha_i, \beta_i$  corresponds to an itemset. Because  $\alpha$  satisfies  $IC$ , based on the problem statement, this condition holds:  $\exists \alpha_i[w_i] \in \alpha, \exists \beta_j \in IC: \beta_j \subseteq \alpha_i$ .

Based on Definition 5, there are three cases to consider:

**Itemset-join:** In  $\beta$  always exists  $\beta_i[w_i]$  such that  $\alpha_i \subseteq \beta_i \Rightarrow \beta_j \subseteq \alpha_i \subseteq \beta_i$  or  $\beta_j \subseteq \beta_i$ . It means  $\beta$  satisfies  $IC$ .

**Sequence-join:** Because itemset of  $\alpha_i$  does not change. It means  $\beta_i = \alpha_i$  and therefore, we have  $\beta_j \subseteq \beta_i$  or  $\beta$  satisfies  $IC$ .

**Inter-join:** Based on the inter-join, all itemsets from  $\alpha$  always exist in  $\beta$ , therefore, in  $\beta$  always exists  $\beta_k[w_k]$  such that  $\alpha_i \subseteq \beta_k \Rightarrow \beta_j \subseteq \alpha_i \subseteq \beta_k$  or  $\beta_j \subseteq \beta_k$ . It means  $\beta$  satisfies  $IC$ .

In the ISP-Join1-improving function, we first check the  $\alpha_{list}$  pattern with the itemset constraints. If it is true, all frequent DBV-PatternList which are extended from  $\alpha_{list}$  also satisfy the itemset constraints (Fig. 8, lines 3–12). This is the same with the ISP-Joink-improving function (Fig. 9, lines 2–13).

Consider the example shown in Table 1. An example of creating a sequential database from a customer dataset.

Where  $minsup = 2, maxspan = 1$  and itemset constraints  $IC = \{(AB), CA, AD\}$ . The frequent patterns generated in the mining phase are presented in Fig. 10.

**Step 1** the sequential database is scanned once by the algorithm to enumerate all frequent 1-patterns, which is  $\{ \langle A \rangle, \langle B \rangle, \langle C \rangle, \langle D \rangle \}$  with the support  $\{4, 2, 2, 2\}$ , respectively (Fig. 10, level 0). In this database scan, the 1-DBV-PatternLists are also generated.

**Step 2** The algorithm generates candidates  $\{ \langle A \rangle [0] \langle A \rangle [1], \langle (AB) \rangle [0], \langle AD \rangle [0], \langle A \rangle [0] \langle D \rangle [1] \}$  that shares the 1-prefix  $\langle A \rangle$  by joining  $\langle A \rangle$  with  $\langle A \rangle, \langle B \rangle, \langle C \rangle$  and  $\langle D \rangle$  based on itemset, sequence and inter extension (Fig. 8). Other generated candidates  $\langle B \rangle [0] \langle A \rangle [1], \langle CA \rangle [0], \langle C \rangle [0] \langle A \rangle [1]$  and  $\langle CB \rangle [0]$  that share 1-prefix  $\langle B \rangle, \langle C \rangle$  are also created at the same time during the combination (Fig. 10, level 1).

**Step 3** The algorithm traverses the children of each  $(k-1)$ -pattern by depth first search to generate  $k$ -patterns. If a  $(k-1)$ -pattern satisfies the constraints  $IC$  then all its super patterns should not be checked. For instance,  $\langle (AB) \rangle [0]$  satisfies the itemset constraints, so we do not need to check the itemset constraints for  $\langle (AB) \rangle [0] \langle A \rangle [1]$ . Due to this checking case, we reduce the checking time of the algorithm. Otherwise,  $\langle A \rangle [0] \langle A \rangle [1]$  do not satisfy the itemset constraints, so we check itemset constraints for its child nodes. The algorithm backtracks to step 3 and complete the full candidates of  $\langle B \rangle, \langle C \rangle$  then  $\langle D \rangle$ . As no more candidates can be found in branch  $\langle D \rangle$ , the algorithm terminates. We have  $FP = \{ \langle AA \rangle [0] \langle AD \rangle [1]: support = 2, \langle (AB) \rangle [0]: support = 2, \langle (AB) \rangle [0] \langle A \rangle [1]: support = 2, \langle AD \rangle [0]: support = 2, \langle CA \rangle [0]: support = 2, \langle CA \rangle [0] \langle A \rangle [1]: support = 2, \langle C(AB) \rangle [0]: support = 2, \langle C(AB) \rangle [0] \langle A \rangle [1]: support = 2 \}$ .

### 4.4 Parallel DBV-ISPMIC algorithm

As shown in Fig. 2, the DBV-ISPMIC algorithm is a sequential algorithm. The complexity time of the DBV-ISPMIC algorithm is calculated by  $t_{node\_1} + t_{node\_2} + t_{node\_3} + \dots + t_{node\_n}$ , whereas the set  $\{node\_1, node\_2, node\_3, \dots, node\_n\}$  contains child nodes of an ISP-tree  $T$ ,  $t_{node_i}$  is the extension time of a node extension of the ISP-tree  $T$  in 1-pattern and  $k$ -pattern (in ISP-Join1 and ISP-Joink functions, respectively). This is because the ISP-Joink extension function independently expands the sub-nodes of the ISP-tree  $T$ . If each sub-branch of the ISP-tree  $T$  is expanded for each task, the running time of the algorithm improves. Therefore, the overall runtime of the algorithm can be determined as  $Max\{node\_1, node\_2, node\_3, \dots, node\_n\}$ .

An example for the above proposal is given in Fig. 11. Based on the database in Table 2 with  $minsup = 2$ , the frequent patterns that were generated at the first level by 1-pattern extension included  $\langle A \rangle, \langle B \rangle, \langle C \rangle$  and  $\langle D \rangle$ . For each frequent pattern, the  $k$ -pattern extension is processed at each individual task. As stated, the time of the algorithm is calculated in  $Max\{t_{Task1}, t_{Task2}, t_{Task3}\}$ , since one task runs in parallel with the others. The allocation of the number of tasks that can be executed simultaneously is

**Fig. 8** The function ISP-Join1-improving

---

**Function 6:** ISP-Join1-Improving( $T, FP, IC, \alpha_{list}$ )

---

1. **For** each frequent 1-DBV-PatternList  $\gamma_{list}$  in  $T \setminus NULL$ , where  $\alpha = \langle u \rangle [0]$  and  $\gamma = \langle v \rangle [0]$ , **do**
  2.   **For**  $x = 0$  to  $maxspan$  **do**
  3.     **If** Check( $\alpha_{list}, IC$ ) **then**
  4.       **If** ( $x=0$ ) and ( $u < v$ ) **then**
  5.          $\theta_{list} = \alpha_{list} \cup_i \gamma_{list}$ ;
  6.         **If**  $support(\theta_{list}) \geq minsup$  **then** add  $\theta_{list}$  to  $T \setminus \alpha_{list}$  and  $\theta$  to  $FP$ ;
  7.         **If** ( $x=0$ ) **then**
  8.          $\rho_{list} = \alpha_{list} \cup_s \gamma_{list}$ ;
  9.         **If**  $support(\rho_{list}) \geq minsup$  **then** add  $\rho_{list}$  to  $T \setminus \alpha_{list}$  and  $\rho$  to  $FP$ ;
  10.        **If** ( $x > 0$ ) **then**
  11.          $\sigma_{list} = \alpha_{list} \cup_t \gamma_{list}$ ;
  12.         **If**  $support(\sigma_{list}) \geq minsup$  **then** add  $\sigma_{list}$  to  $T \setminus \alpha_{list}$  and  $\sigma$  to  $FP$ ;
  13.     **Else**
  14.        ISP-Join1-Extension( $T \setminus \alpha_{list}, FP, IC, \alpha_{list}, \gamma_{list}, u, v, x$ );
  15.     **End for**
  16. **End for**
- 

**Fig. 9** The function ISP-Joink-improving

---

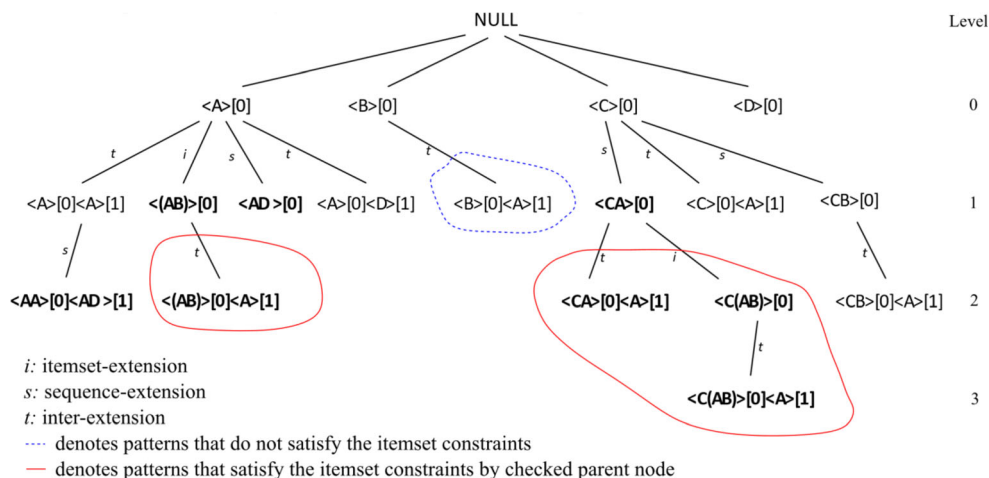
**Function 7:** ISP-Joink-Improving( $T, FP, IC$ )

---

1. **For** each frequent  $k$ -DBV-PatternList  $\beta_{list}$  in  $T \setminus \alpha_{list}$ , where  $sub_{k,k}(\beta) = \langle u \rangle [i]$ , **do**
  2.   **If** Check( $\beta_{list}, IC$ ) **then**
  3.     **For** each frequent  $k$ -DBV-PatternList  $\gamma_{list}$  in  $T \setminus \alpha_{list}$ , where  $sub_{k,k}(\gamma) = \langle v \rangle [j]$ , **do**
  4.       **If** ( $i=j$ ) and ( $u < v$ ) **then**
  5.          $\theta_{list} = \beta_{list} \cup_i \gamma_{list}$ ;
  6.         **If**  $support(\theta_{list}) \geq minsup$  **then** add  $\theta_{list}$  to  $T \setminus \beta_{list}$  and  $\theta$  to  $FP$ ;
  7.         **If** ( $i=j$ ) **then**
  8.          $\rho_{list} = \beta_{list} \cup_s \gamma_{list}$ ;
  9.         **If**  $support(\rho_{list}) \geq minsup$  **then** add  $\rho_{list}$  to  $T \setminus \beta_{list}$  and  $\rho$  to  $FP$ ;
  10.        **If** ( $i < j$ ) **then**
  11.          $\sigma_{list} = \beta_{list} \cup_t \gamma_{list}$ ;
  12.         **If**  $support(\sigma_{list}) \geq minsup$  **then** add  $\sigma_{list}$  to  $T \setminus \beta_{list}$  and  $\sigma$  to  $FP$ ;
  13.     **End for**
  14.   **Else**
  15.     **For** each frequent  $k$ -DBV-PatternList  $\gamma_{list}$  in  $T \setminus \alpha_{list}$ , where  $sub_{k,k}(\gamma) = \langle v \rangle [j]$ , **do**
  16.        ISP-Joink-Extension( $T \setminus \beta_{list}, FP, IC, \beta_{list}, \gamma_{list}, u, v, i, j$ );
  17.     **End for**
  18.    Call ISP-Joink-Improving( $T \setminus \beta_{list}, FP, IC$ );
  19. **End for**
  20. Delete  $T \setminus \alpha_{list}$  from  $T$ ;
-



**Fig. 10** The extended tree of patterns corresponding to the example database



determined by the computer processor’s cores. This can also be extended to distributed systems, where each task is processed on a separate system and then final the result is gathered and combined.

The *p*DBV-ISMIC algorithm is based on the DBV-ISMIC with the algorithm parallelized. Figure 12 shows with a flowchart the main steps of the sequential DBV-ISMIC algorithm and its parallel (*p*DBV-ISMIC) counterpart. The *p*DBV-ISMIC algorithm has two main steps:

- Step 1:** Loading the database and finding the frequent 1-pattern sets that satisfy the *minsup*.
- Step 2:** Allocating execution tasks, with each task handling one *k*-pattern. The algorithm search space exploration is DFS-based (depth-first search), which is recursive. When no more candidates can be generated, the algorithm terminates.

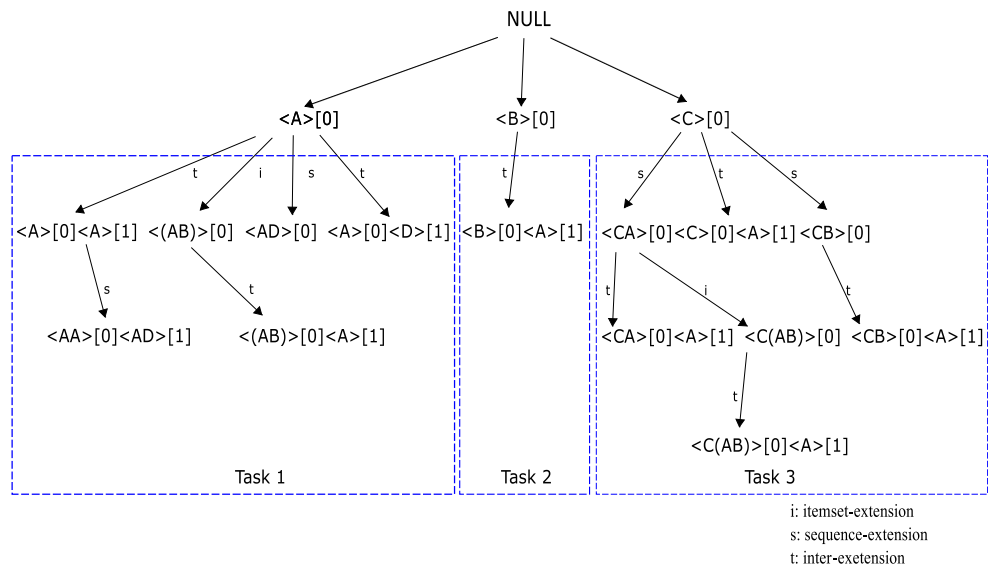
### 5 Experimental results

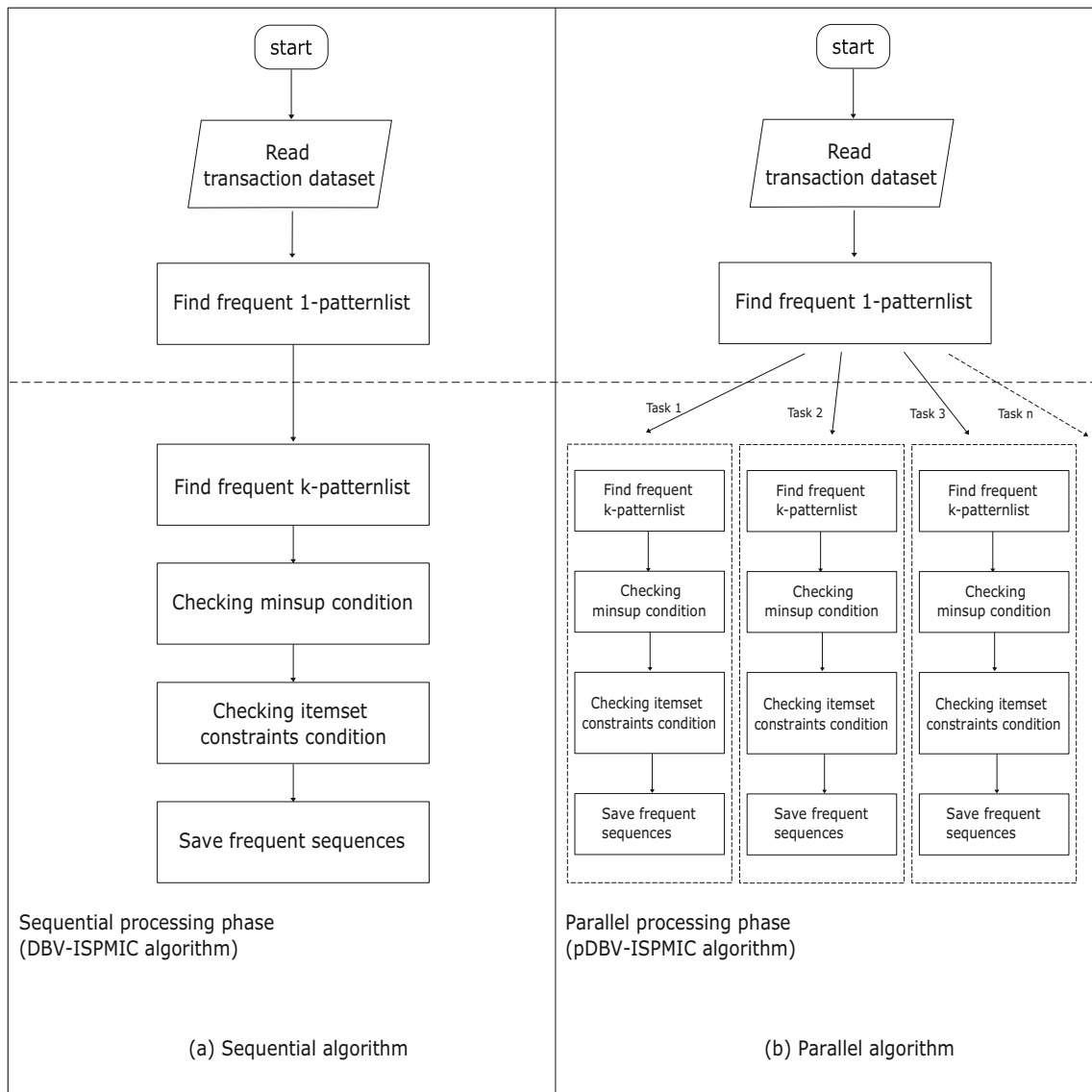
In evaluating the performance of the DBV-ISMIC algorithm and its improvement in runtime, all the experiments were carried out on a PC with an Intel® Core™ i7 10th gen processor (10,510 U) @ 1.8–4.9 GHz, and 20 GB RAM. The operating system used is Windows 10 64-bit. The algorithms were implemented in Visual Studio 2017 C#.

#### 5.1 Experimental databases

We ran tests on five databases, namely C6T5S4I4N1kD1k, C6T5S4I4N1kD10k, Gazelle, BIKE and BMSWebView1. The synthetic databases used for comparison were generated using the IBM synthetic data generator. These databases are available at <https://www.mediafire.com/folder/id3p3z6b9g8kj>. Their characteristics are shown in Table 3.

**Fig. 11** Example of using parallel processing for ISP-tree extension





**Fig. 12** The figure shows the difference between sequential and parallel flow chart. **a** The main steps of a sequential algorithm, and **b** the main steps of a parallel processing algorithm

**5.2 Runtime**

For the C6T5S4I4N1kD1k database, we evaluated the algorithms with *minsup* = 0.5% and *maxspan* = {1,2,3,4,5}, the C6T5S4I4N1kD10k database with *minsup* = 5% and

*maxspan* = {1, 2, 3, 4, 5}, the Gazelle database with *minsup* = 1% and *maxspan* = {1,2,3,4,5}, the BIKE database with *minsup* = 0.5% and *maxspan* = {1, 2, 3, 4, 5}, and the BMSWebView1 database with *minsup* = 0.5% and *maxspan* = {1, 2, 3, 4, 5}. We use an EISP-Miner algorithm

**Table 3** Test database characteristics

Database	#Sequence	#Item	Type of data
C6T5S4I4N1kD1k	1000	1000	Synthetic databases
C6T5S4I4N1kD10k	10,000	1000	Synthetic databases
Gazelle	59,602	497	Clickstream data
BIKE	21,078	67	Bike Share data from LA Metro
BMSWebView1	59,601	497	Clickstream data

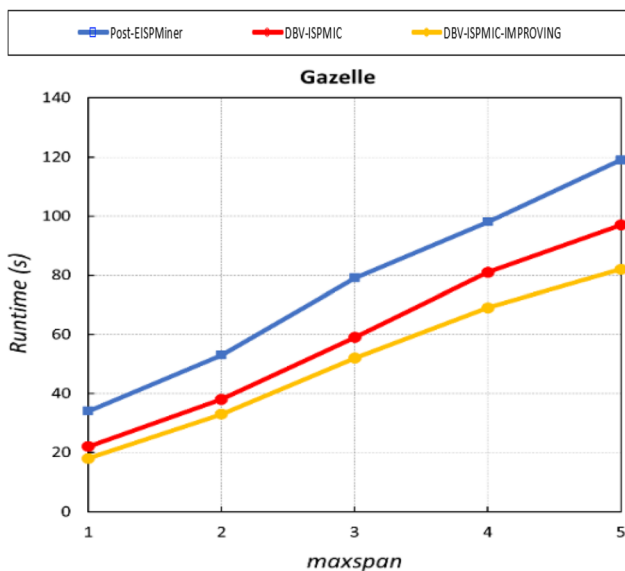


Fig. 13 Execution times of Post-EISPMiner, DBV-ISMIC and DBV-ISMIC-IMPROVING for the Gazelle dataset

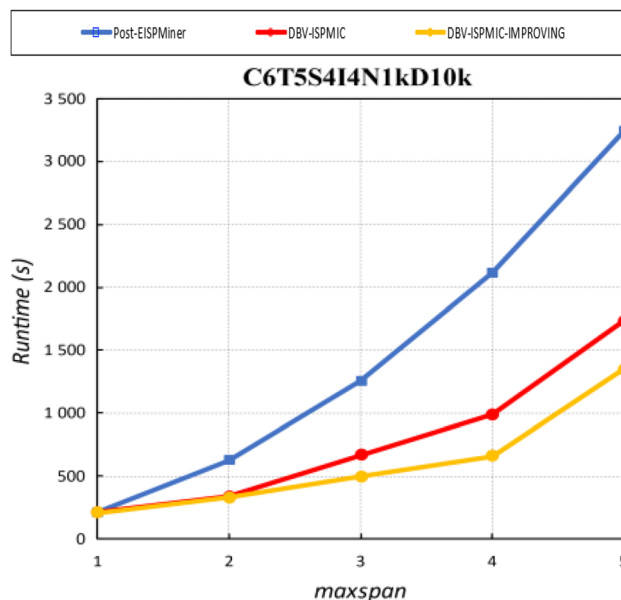


Fig. 15 Execution times of Post-EISPMiner, DBV-ISMIC and DBV-ISMIC-IMPROVING for the C6T5S4I4N1kD10k dataset

to evaluate all the proposed algorithms [7], and add a check constraint to it, with this approach called the Post-EISPMiner algorithm.

Based on the experimental results in Figs. 13, 14, 15, 16 and 17, we can see that the DBV-ISMIC-IMPROVING algorithm runs faster than the other two algorithms, Post-EISPMiner and DBV-ISMIC. In Fig. 13, we compare the runtime of Post-EISPMiner, DBV-ISMIC and DBV-

ISMIC-IMPROVING for the Gazelle dataset. When the value of *maxspan* is increasing, the running time of all three algorithms increases relatively evenly.

Figures 14, 15, 16 and 17 show the results for the C6T5S4I4N1kD1k, C6T5S4I4N1kD10k, BIKE and BMSWebView1 datasets. It is clear that the runtimes for DBV-ISMIC-IMPROVING and DBV-ISMIC are much better than that of Post-EISPMiner.

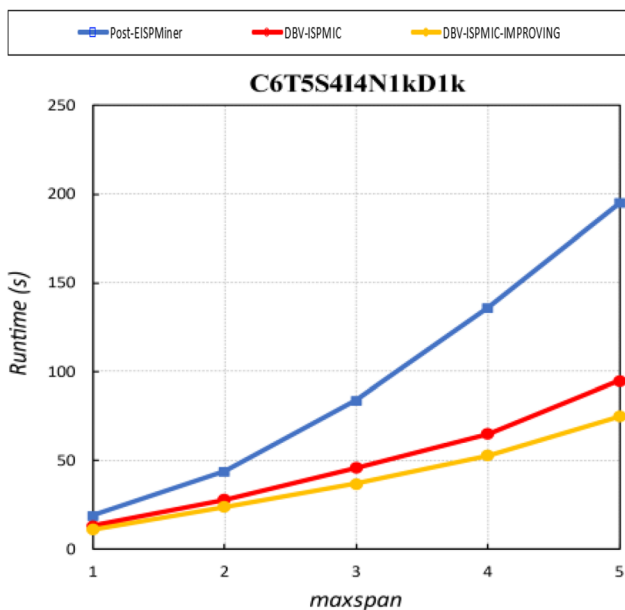


Fig. 14 Execution times of Post-EISPMiner, DBV-ISMIC and DBV-ISMIC-IMPROVING for the C6T5S4I4N1kD1k dataset

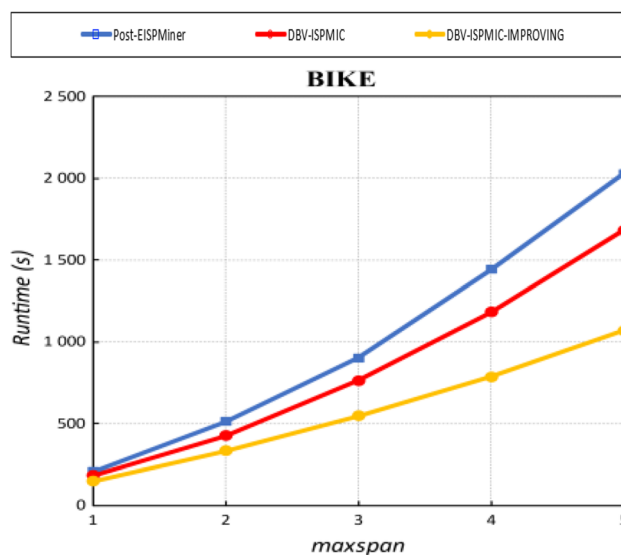


Fig. 16 Execution times of Post-EISPMiner, DBV-ISMIC and DBV-ISMIC-IMPROVING for the BIKE dataset

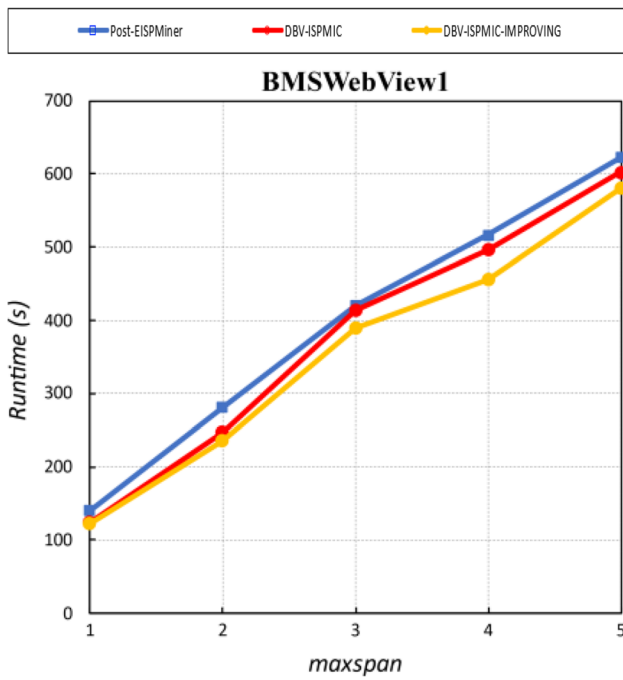


Fig. 17 Execution times of Post-EISPMiner, DBV-ISP MIC and DBV-ISP MIC-IMPROVING for the BMSWebView1 dataset

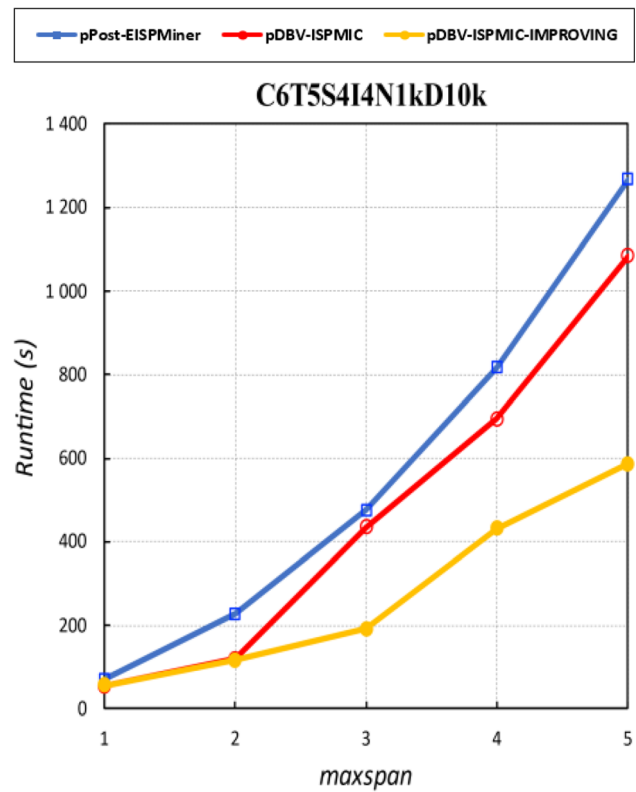


Fig. 19 Execution time in a parallel evaluation of *p*Post-EISPMiner, *p*DBV-ISP MIC and *p*DBV-ISP MIC-IMPROVING for the C6T5S4I4N1kD10k dataset

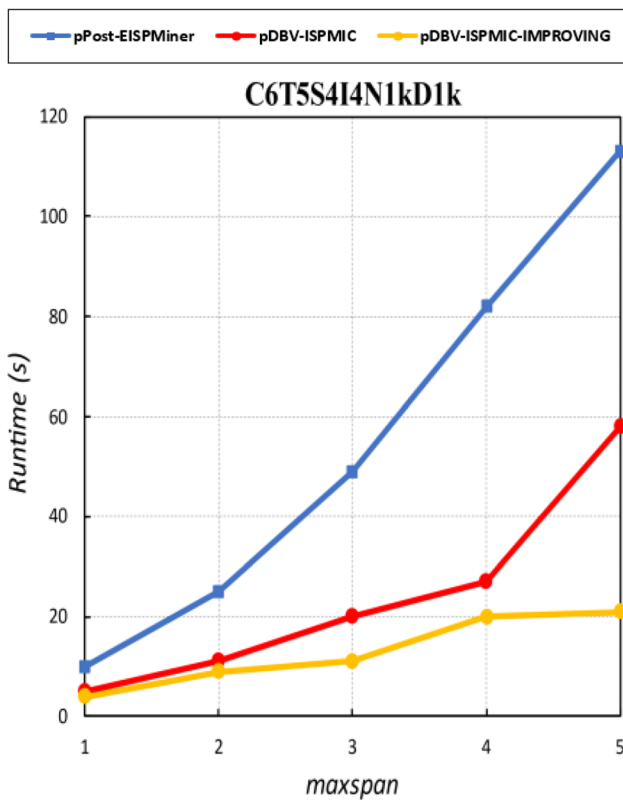


Fig. 18 Execution times in a parallel evaluation of *p*Post-EISPMiner, *p*DBV-ISP MIC and *p*DBV-ISP MIC-IMPROVING for the C6T5S4I4N1kD1k dataset

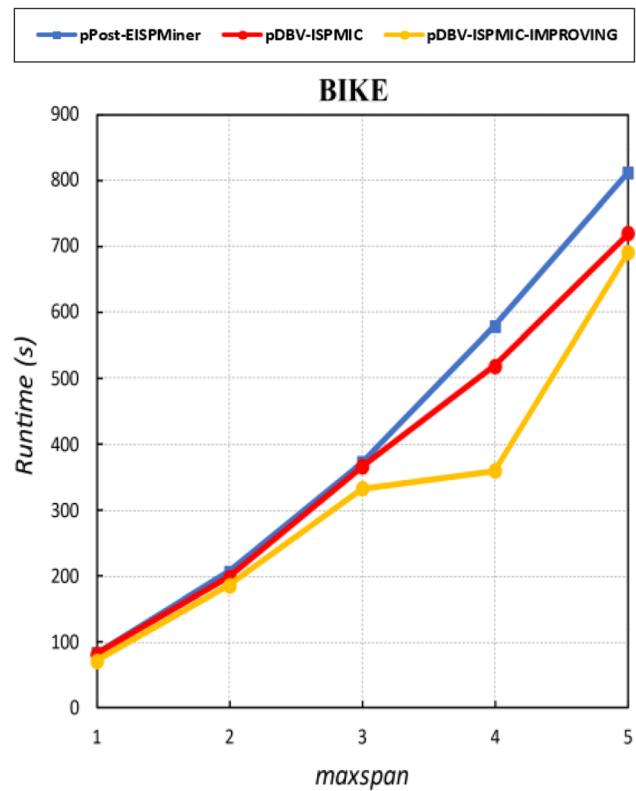


Fig. 20 Execution times in a parallel evaluation of *p*Post-EISPMiner, *p*DBV-ISP MIC and *p*DBV-ISP MIC-IMPROVING for the BIKE dataset

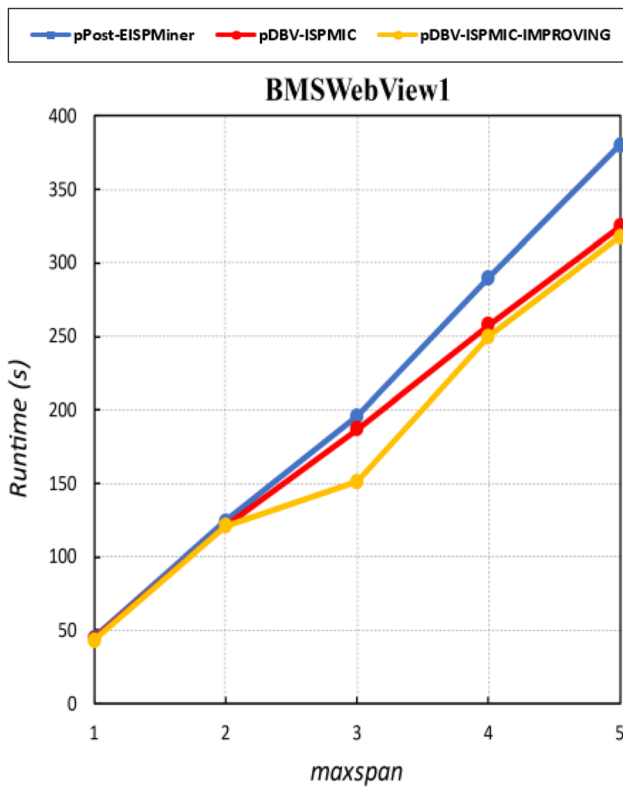


Fig. 21 Execution times in a parallel evaluation of *p*Post-EISPMiner, *p*DBV-ISP MIC and *p*DBV-ISP MIC-IMPROVING for the BMSWebView1 dataset

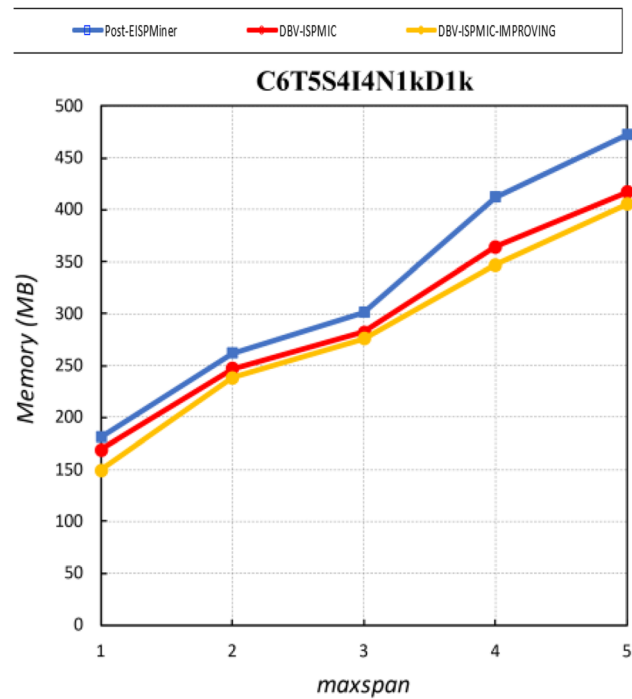


Fig. 23 Memory usage of Post-EISPMiner, DBV-ISP MIC and DBV-ISP MIC-IMPROVING for the C6T5S4I4N1kD1k dataset

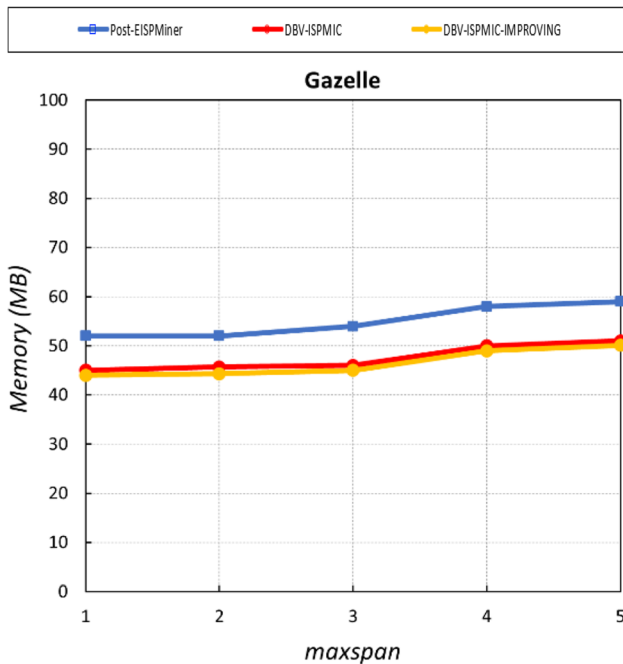


Fig. 22 Memory usage of Post-EISPMiner, DBV-ISP MIC and DBV-ISP MIC-IMPROVING for the Gazelle dataset

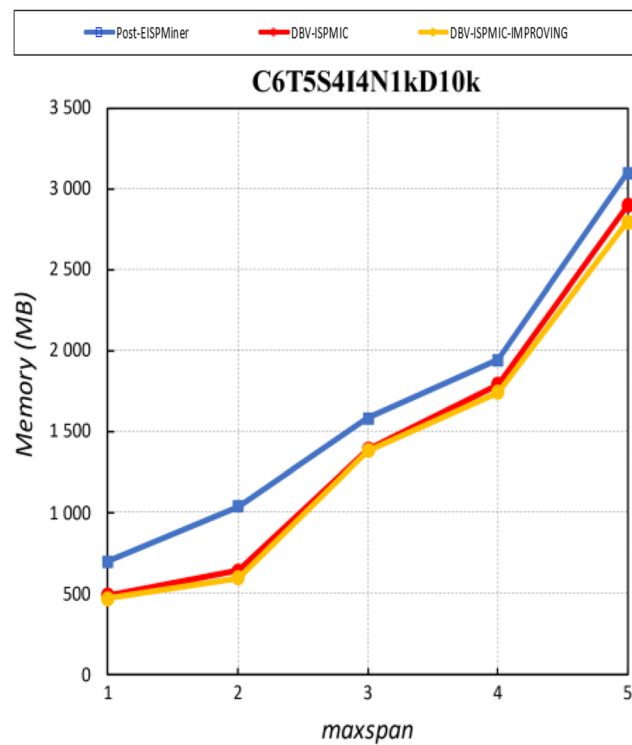


Fig. 24 Memory usage of Post-EISPMiner, DBV-ISP MIC and DBV-ISP MIC-IMPROVING for the C6T5S4I4N1kD10k dataset

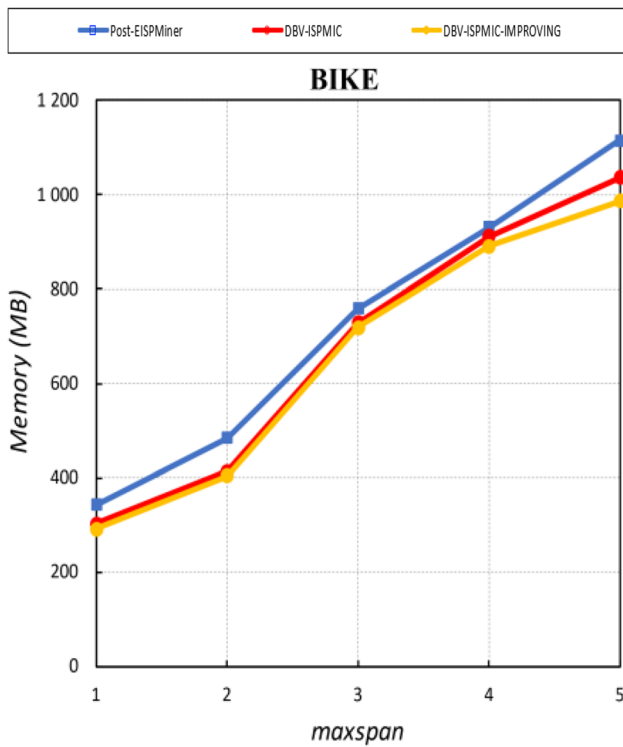


Fig. 25 Memory usage of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the BIKE dataset

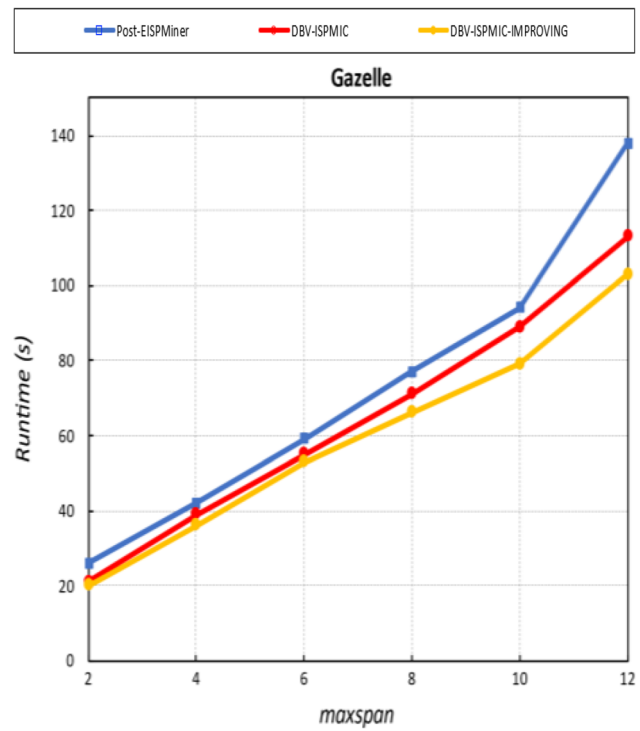


Fig. 27 Execution times of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the Gazelle dataset, with maxspan from 2 to 12

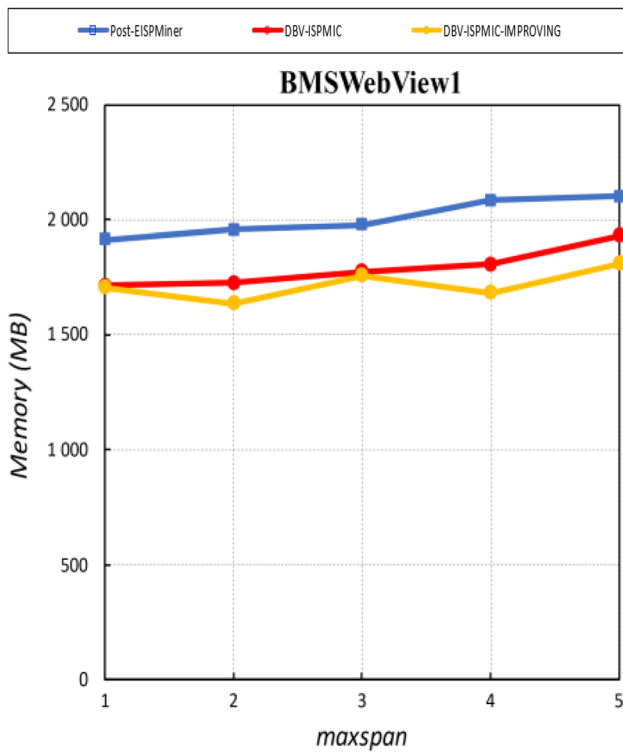


Fig. 26 Memory usage of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the BMSWebView1 dataset

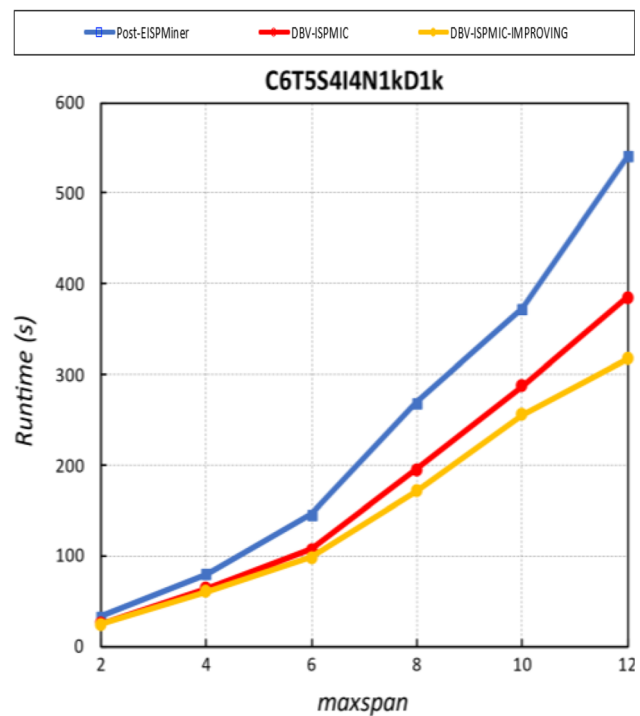


Fig. 28 Execution times of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the C6T5S4I4N1kD1k dataset, with maxspan from 2 to 12

### 5.3 Parallel method for efficient mining of inter-sequence patterns with itemset constraints

Because DBV-ISPMIC-IMPROVING is the best algorithm for mining inter-sequence patterns with itemset constraints, we develop a parallel version of it, *p*DBV-ISPMIC-IMPROVING, by using the C#.NET software library to improve the performance. The performance of *p*DBV-ISPMIC-IMPROVING algorithm is evaluated by comparing it with that of the DBV-ISPMIC-IMPROVING algorithm. The results are shown in Figs. 18, 19, 20 and 21, and it can be seen that when *maxspan* increases, the runtime of *p*DBV-ISPMIC-IMPROVING is much less than the runtime of DBV-ISPMIC-IMPROVING algorithm.

### 5.4 Memory usage

Figures 22, 23, 24, 25 and 26 show the peak memory consumption of the three algorithms, Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING. The results show that the memory needed by DBV-ISPMIC and DBV-ISPMIC-IMPROVING is less than that needed by the Post-EISPMiner algorithm for almost all database parameter values. Because the two proposed algorithms reduce the time needed to check the child nodes generated, they have less memory usage compared to the Post-EISPMiner algorithm.

### 5.5 Impact of *maxspan*

For mining inter-sequence patterns, when we increase the *maxspan* value, the number of candidates generated will also increase. Therefore, if we use proposition 1 to reduce the itemset constraints checking, the processing time will be better. For instance, we use two databases, Gazelle and C6T5S4I4N1kD1k, to evaluate this. The Gazelle database (*minsup* = 3%) and C6T5S4I4N1kD1k database (*minsup* = 0.8%) were tested with the *maxspan* value increasing from 2 to 12. The results show that the proposed algorithms (DBV-ISPMIC and DBV-ISPMIC-IMPROVING) always work well (Fig. 27 and 28).

## 6 Conclusions and future work

With this study, we introduced an algorithm to solve the problem of mining inter-sequence patterns with itemset constraints. This algorithm, named DBV-ISPMIC, is based on the EISP-Miner algorithm to mine inter-sequence patterns, and uses a dynamic bit vector structure to store data, which helps to increase the processing speed and reduce the storage space when compared to EISP-Miner. Based on the DBV-

ISPMIC algorithm, we also propose its improvement to help reduce processing time.

In the future, we will apply distributed computing to the improved algorithm to help optimize the running time. We will also study how to put the constraints into mining frequent closed inter-sequences. Finally, algorithms for mining high utility sequences have been proposed in recent years [19–25], and we will study how to mine high utility inter-sequences and high utility inter-sequences with constraints.

## References

1. Huynh HM, Nguyen LTT, Vo B, Nguyen A, Tseng VS (Mar. 2020) Efficient methods for mining weighted clickstream patterns. *Expert Syst Appl* 142:112993. <https://doi.org/10.1016/j.eswa.2019.112993>
2. Agrawal R, Srikant R (1995) Mining sequential patterns. In: *Proceedings - International Conference on Data Engineering*, pp. 3–14. <https://doi.org/10.1109/icde.1995.380415>
3. Zaki MJ (2001) SPADE: an efficient algorithm for mining frequent sequences. *Mach Learn* 42(1–2):31–60. <https://doi.org/10.1023/A:1007652502315>
4. Han J, Pei J, Mortazavi-Asl B, Chen Q, Dayal U, Hsu M-C (2000) FreeSpan: frequent pattern-projected sequential pattern mining. In: *Proceedings - the sixth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD '00*, pp. 355–359. <https://doi.org/10.1145/347090.347167>
5. Pei J et al (2001) PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: *Proceedings - International Conference on Data Engineering*, pp. 215–224. <https://doi.org/10.1109/icde.2001.914830>
6. Fournier-Viger P, Gomariz A, Campos M, Thomas R Fast vertical mining of sequential patterns using co-occurrence information. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, vol. 8443 LNAI, no. PART 1, pp. 40–52. [https://doi.org/10.1007/978-3-319-06608-0\\_4](https://doi.org/10.1007/978-3-319-06608-0_4)
7. Wang CS, Lee AJT (May 2009) Mining inter-sequence patterns. *Expert Syst Appl* 36(4):8649–8658. <https://doi.org/10.1016/j.eswa.2008.10.008>
8. Le T, Nguyen A, Huynh B, Vo B, Pedrycz W (May 2018) Mining constrained inter-sequence patterns: a novel approach to cope with item constraints. *Appl Intell* 48(5):1327–1343. <https://doi.org/10.1007/s10489-017-1123-9>
9. Vo B, Tran MT, Hong TP, Nguyen H, Le B (2012) A dynamic bit-vector approach for efficiently mining inter-sequence patterns. In: *Proceedings - 3rd International Conference on Innovations in Bio-Inspired Computing and Applications, IBICA 2012*, pp. 51–56. <https://doi.org/10.1109/IBICA.2012.31>
10. Le B, Tran MT, Vo B (Jul. 2015) Mining frequent closed inter-sequence patterns efficiently using dynamic bit vectors. *Appl Intell* 43(1):74–84. <https://doi.org/10.1007/s10489-014-0630-1>
11. Wang CS, Liu YH, Chu KC (Jun. 2013) Closed inter-sequence pattern mining. *J Syst Softw* 86(6):1603–1612. <https://doi.org/10.1016/J.JSS.2013.02.010>
12. Liao W, Wang Q, Yang L, Ren J, Davis DN, Hu C (Apr. 2018) Mining frequent intra-sequence and inter-sequence patterns using bitmap with a maximal span. *Proc. - 2017 14th web Inf. Syst Appl Conf WISA 2017*, vol 2018-January, pp 56–61. <https://doi.org/10.1109/WISA.2017.70>

13. Van T, Le B (Mar. 2021) Mining sequential rules with itemset constraints. *Appl Intell* 51:1–13. <https://doi.org/10.1007/s10489-020-02153-w>
14. Van T, Vo B, Le B (Nov. 2018) Mining sequential patterns with itemset constraints. *Knowl Inf Syst* 57(2):311–330. <https://doi.org/10.1007/s10115-018-1161-6>
15. Gouda K, Hassaan M, Zaki MJ (2007) PRISM: A prime-encoding approach for frequent sequence mining. In: *Proceedings - IEEE International Conference on Data Mining, ICDM*, pp. 487–492. <https://doi.org/10.1109/ICDM.2007.33>
16. Gouda K, Hassaan M, Zaki MJ (Feb. 2010) Prism: an effective approach for frequent sequence mining via prime-block encoding. *J Comput Syst Sci* 76(1):88–102. <https://doi.org/10.1016/J.JCSS.2009.05.008>
17. Huynh HM, Nguyen LTT, Vo B, Yun U, Oplatková ZK, Hong TP (Jun. 2020) Efficient algorithms for mining clickstream patterns using pseudo-IDLists. *Futur Gener Comput Syst* 107:18–30. <https://doi.org/10.1016/j.future.2020.01.034>
18. Huynh HM, Nguyen LTT, Vo B, Oplatková ZK, Fournier-Viger P, Yun U (Jan. 2022) An efficient parallel algorithm for mining weighted clickstream patterns. *Inf Sci (NY)* 582:349–368. <https://doi.org/10.1016/J.INS.2021.08.070>
19. Gan W, Lin JCW, Zhang J, Fournier-Viger P, Chao HC, Yu PS (Feb. 2021) Fast utility mining on sequence data. *IEEE Trans Cybern* 51(2):487–500. <https://doi.org/10.1109/TCYB.2020.2970176>
20. Gan W et al (May 2021) Utility Mining Across Multi-Dimensional Sequences. *ACM Trans Knowl Discov Data* 15(5):1–24. <https://doi.org/10.1145/3446938>
21. Lin JCW, Li Y, Fournier-Viger P, Djenouri Y, Zhang J (2020) Efficient chain structure for high-utility sequential pattern mining. *IEEE Access* 8:40714–40722. <https://doi.org/10.1109/ACCESS.2020.2976662>
22. Gan W, Lin JCW, Zhang J, Chao HC, Fujita H, Yu PS (Mar. 2020) ProUM: Projection-based utility mining on sequence data. *Inf Sci (NY)* 513:222–240. <https://doi.org/10.1016/J.INS.2019.10.033>
23. Wu Y, Geng M, Li Y, Guo L, Li Z, Fournier-Viger P, Zhu X, Wu X (Oct. 2021) HANP-miner: high average utility nonoverlapping sequential pattern mining. *Knowledge-Based Syst* 229:107361. <https://doi.org/10.1016/J.KNOSYS.2021.107361>
24. Chun-wei Lin J et al (Nov. 2021) Scalable Mining of High-Utility Sequential Patterns with Three-Tier MapReduce model. *ACM Trans Knowl Discov Data* 16(3):1–26. <https://doi.org/10.1145/3487046>
25. Truong T, Duong H, Le B, Fournier-Viger P, Yun U, Fujita H (Aug. 2021) Efficient algorithms for mining frequent high utility sequences with constraints. *Inf Sci (NY)* 568:239–264. <https://doi.org/10.1016/J.INS.2021.01.060>

**Publisher's note** Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.