Politechnika Wrocławska

FIELD OF SCIENCE: DEPARTMENT OF APPLIED INFORMATICS

DISCIPLINE OF SCIENCE: FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

DOCTORAL DISSERTATION

MINING SEQUENCE AND INTER-SEQUENCE PATTERNS IN LARGE DATABASES

Anh Nguyen

Supervisor/Supervisors:
Prof. D.Sc. Ngoc Thanh Nguyen

Assistant supervisor:
Assoc. Prof. Ph.D. Bay Vo

Keywords: Data Mining, Inter-sequence Pattern Mining

WROCŁAW 2023

ACKNOWLEDGMENTS

First and foremost, I would like to express my deepest gratitude to Professor Ngoc Thanh Nguyen, Associate Professor Bay Vo for their invaluable guidance, unwavering support, and enthusiastic mentorship throughout my journey as a PhD student in the field of scientific research. Their expertise and advice have played a pivotal role in shaping my academic development during my tenure at Wrocław University of Science and Technology.

I would also like to extend my heartfelt appreciation to my esteemed colleagues in the Department of Applied Informatics, Faculty of Information and Communication Technology. Their insightful suggestions, informative discussions, and shared knowledge on the research topic, especially through seminars, have greatly enriched my understanding and perspective.

Furthermore, I would like to express my sincere gratitude to the Doctoral School of Wrocław University of Science and Technology for their unwavering support and encouragement during every phase of my doctoral studies. From the day of my admission and throughout each academic term and year, their enthusiastic assistance has been invaluable.

Finally, I would like to convey my deepest thanks to my family for their unwavering support, encouragement, and facilitation throughout my pursuit of the PhD degree. Their continuous belief in my abilities and provision of a conducive environment have been instrumental in my academic journey.

Wrocław, June 11, 2023

INDEX

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CHAPTER 1: INTRODUCTION

1.1 Background

1.1.1 Data Mining

Data mining is the process of identifying meaningful patterns, trends, and relationships in large datasets using various statistical and computational techniques. It aims to extract valuable information from data and transform it into rules that can be used to draw conclusions or make predictions to assist the user. Pattern mining involves discovering useful, interesting, and unexpected patterns in the database. Some of the basic data mining tasks include clustering, classification, outlier analysis, and pattern mining. The Apriori algorithm, proposed by Agrawal and Srikant in the 1990s, is designed to identify frequent item sets and extract association rules (R. Agrawal and R. Srikant, 1994). Frequent itemsets refer to groups of symbols that appear together often in a database of customer transactions.

Data mining algorithms analyze data from various perspectives, uncover hidden patterns and correlations, and provide insights that can help organizations and individuals make informed decisions in the future. The data mining process involves several steps, including data cleaning, integration, selection, transformation, mining, evaluation, and representation. These steps are crucial to ensure that the data is accurate, complete, and relevant to the current business problem.

Data mining has numerous applications in various industries, such as finance, healthcare, retail, and marketing. Retailers often use data mining techniques to analyze customer shopping data and identify patterns and trends in their purchasing behavior. By doing so, they can better understand customer preferences and develop marketing strategies based on customer habits to drive sales. For example, a retailer may use data mining to analyze customer transaction data and determine which products are often purchased together. If they find that customers who purchase personal computers are also likely to buy accompanying electronic devices such as computer keyboards, computer mice, and computer monitors, they may use this information to create targeted marketing campaigns that suggest or offer discounts on these products. For example, Table 1.1 represents a simple retail store database that contains customer purchase records, including transaction ID (TID), purchase date, time, customer ID, and the products purchased by customers. Each customer's purchasing behavior can be viewed as a sequence of events occurring at different times. As an illustration, customer 1 purchased a laptop and mouse

on March 20, 2023, at transaction T_1. On March 21, 2023, the same customer purchased a keyboard at transaction T_4.

Table 1.1. An original customers database with reference to shopping behavior.

TID	Date	Time	Customer ID	Items
T_1	20.03.2023	09.00	1	Laptop, Mouse
T_2	20.03.2023	10.00	2	iPhone, Speaker
T_3	20.03.2023	11.00	3	Cab, USB
T_4	21.03.2023	16.00	1	Keyboard
T_5	21.03.2023	09.00	3	Mouse
T_6	22.03.2023	09.00	4	Samsung phone, Speaker, Printer
T_7	23.03.2023	09.00	5	Monitor

Association rule mining is a data mining technique utilized to discover relationships and associations between items or sets of items in a database. It involves analyzing data to find patterns, associations, correlations, or co-occurrence between items. Association rules are created to describe the relationships that exist between different items in a data set. These rules consist of a premise (antecedent) and a consequence (conclusion) and are typically written in the form "if the premise, then the consequence." The strength of an association rule is evaluated based on two primary metrics: support and confidence. Support indicates the frequency of occurrence of antecedents and consequences in a data set, while confidence measures the likelihood that the consequences will occur with the premises. Association rule mining has many applications in various fields, including market basket analysis, bioinformatics, web mining, and recommendation systems.

To illustrate, suppose a grocery store wants to analyze its sales data to determine which items are frequently purchased together. By employing association rule mining, the store can identify sets of items that were purchased together more often than expected by chance. For instance, the store may discover that customers who buy bread and milk are also likely to purchase eggs. In this case, the association rule would be:

$$IF\ \{Laptop, Keyboard\}\ THEN\ \{Mouse\}$$

Suppose this item set appears in 20% of all transactions in the store's data. Then the support of the rule is 20%. The confidence of the rule represents the percentage of transactions that contain $\{Laptop, Keyboard\}$ and also contain $\{Mouse\}$. Assuming the confidence of this rule is 80%, this implies that of all transactions containing $\{Laptop, Keyboard\}$, 80% also contain $\{Mouse\}$. Based on this information, the store

can take measures to increase sales by placing these items close together or offering them at a discount when purchased together.

1.1.2 Mining Sequence Patterns

The sequential pattern mining problem, proposed by Agrawal and Srikant (Agrawal & Srikant, 1995), is an interesting problem of subsequence mining in a set of sequences. Sequential pattern mining is a data mining technique that aims to discover interesting patterns or sequences of events in sequential data. It involves identifying subsequences that frequently occur together in a sequence of events or transactions, seeking to uncover patterns that describe the relationships between events or items that occur in a specific order.

Two common types of sequential data used in data mining are time series and sequences. A time series is an ordered list of numerical values, while a sequence is an ordered list of nominal (symbolic) values. For example, Figure 1.1 shows a time series representing quantities, while Figure 1.2 depicts a sequence of symbols (letters). Both time series and sequences are used in many fields. Time series are often used to represent data such as stock prices, temperature readings, and electricity consumption indexes, while sequences are used to represent data such as sentences in text (word strings), sequences of items purchased by customers in retail stores, and sequences of websites visited by users. Sequential pattern mining is commonly used in applications where time ordering is significant, such as analyzing customer purchasing behavior over time or identifying patterns in web clickstream data. It can also be applied in areas such as healthcare, where it can be used to detect trends and patterns in medical data over time.

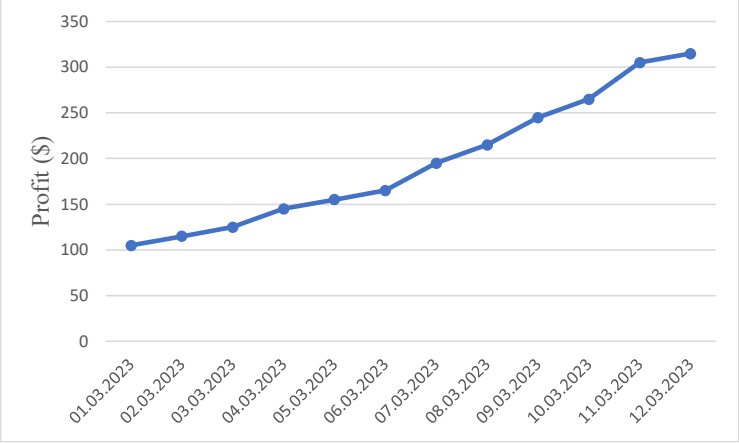


Figure 1.1. An example of time-series

$\langle \{a, b\} \{a, b, c\} \{a, d\} \{c, d\} \rangle$

Figure 1.2. An example of a sequence

Sequential pattern mining involves finding all frequent subsequences in a database of sequences. A sequence S is considered a frequent sequence or a sequential pattern if and only if its support value $\big(support(S)\big)$ is greater than or equal to a user-defined minimum support value $minsupport$. The task of sequential pattern mining involves enumerating all patterns (subsequences) whose support is greater than or equal to the minimum support threshold set by the user. As such, the number of frequent patterns always has a single, true value for the sequential pattern mining problem.

However, discovering sequential patterns is a difficult problem. The naive approach is to compute the support of all possible subsequences in the sequence database and output only those that satisfy the user-specified minimum support constraint. However, this approach is inefficient due to the potentially large number of subsequences. A sequence containing n items in a sequence database can have up to $2^n - 1$ distinct subsequences (Fournier-Viger et al., 2017). Therefore, using a naive approach to solve the sequential pattern mining problem is impractical for most practical sequence databases. Efficient algorithms must be designed to solve the sequential pattern mining problem while avoiding exploring the search space of all possible subsequences. This is necessary to speed up the algorithm and optimize storage space.

Several algorithms have been proposed to solve the sequential pattern mining problem in a sequence database, including AprioriAll (Agrawal & Srikant, 1995), FreeSpan (Han et al., 2000), SPADE (Zaki, 2001), PrefixSpan (Pei et al., 2001), PRISM (Gouda et al., 2007, 2010), CM-SPADE (Fournier-Viger et al., 2014), MSRIC-R and MSRIC-P (Van & Le, 2021), MSPIC-DBV (Van et al., 2018a), CUP (Huynh et al., 2020), CM-WSPADE (Huynh et al., 2020), and SUI (Huynh et al., 2022). These algorithms take a sequence database and a minimum support threshold (selected by the user) as input and output a set of frequent sequential patterns. It's worth noting that there is always only one correct answer to the sequential pattern mining task for a given sequence database and threshold value. Therefore, all sequential pattern mining algorithms return the same set of sequential patterns if they are run with the same parameters on the same database. The difference between the various algorithms lies in their approach to detecting sequential

patterns, with each algorithm using different strategies and data structures to efficiently search for these patterns.

1.1.3 Mining Inter-sequence Patterns

Inter-sequence pattern mining is an extension of sequential pattern mining that involves discovering common patterns, associations, and dependencies between sequences in a sequential database. It identifies patterns that are common not only within the same transaction but also between transactions. The inter-sequential pattern mining problem was first proposed by Wang et al. in 2009 (C. S. Wang & Lee, 2009), and several algorithms have been developed since then, including DBV-ISP (Vo et al., 2012), ISP-IC, *i*ISP-IC and *pi*ISP-IC (T. Le et al., 2018).

One of the applications of inter-sequence pattern mining is in web usage mining, where it is used to analyze the behavior of website visitors. By analyzing web log data, inter-sequence pattern mining can identify common patterns in user behavior, such as the most frequently visited pages or the sequence of pages the user visits before making a purchase. This information can be used to optimize website design, improve user experience, and increase usage rates. In addition to web usage mining, inter-sequence pattern mining has applications in sales, where it can make predictions based on information about purchased items and different times of purchase. It can also be used in other fields such as healthcare and finance.

Overall, inter-sequence pattern mining is a valuable tool for extracting useful information from sequential databases. Its importance is expected to increase as technology continues to advance and users demand more innovative solutions.

1.2 Motivation

Most of the previous studies on mining inter-sequence patterns have shown that these methods still possess disadvantages that result in high computational costs and memory usage during the mining process:

Mining inter-sequence patterns

- The EISP-Miner (C. S. Wang & Lee, 2009) algorithm is the first proposed algorithm for mining inter-sequence patterns. The algorithm is divided into two phases. In the first phase, the algorithm discovers frequent 1-patterns from the sequence database. Next, EISP-Miner utilizes the frequent patterns mined in the first phase to generate frequent k-patterns (where $k > 1$). The algorithm employs a depth-first search extension method. The data structure used is PatternList, which uses a collection of integers to store position information of

frequent patterns. Consequently, the algorithm consumes a significant amount of storage resources during execution, as depicted in Figure 3.7. Furthermore, instead of retrieving information directly from the frequent patterns that created it, all information about pattern positions is stored, leading to the algorithm using more storage space than necessary (as detailed in Chapter 3). Because the algorithm employs integer types to store information, calculating the candidate's support or combining to expand the candidate also requires additional computing time.

- The DBV-ISP (Vo et al., 2012) algorithm was an extension of the EISP-Miner algorithm. DBV-ISP also operates in two phases: first, it identifies the set of frequent 1-patterns, and then it identifies sets of frequent k-patterns $(k > 1)$. DBV-ISP employs a data structure known as DBV-PatternList (as detailed in Section 4.2), which utilizes a bit-vector data type to store pattern positions. The use of the bit-vector data type helps reduce the algorithm's storage space compared to the integer data type. However, the DBV-PatternList data structure has not yet addressed the limitations of the PatternList data structure concerning data duplication and redundancy. Furthermore, DBV-PatternList stores more redundant data than necessary. For example, in a database with 100 transactions and a pattern that appears only in the 1st and 100th positions, DBV-PatternList will store the position for the pattern as a consecutive sequence of bit-vectors with a length of 100, where the 1st and 100th positions are set to the value 1, and the remaining positions are set to the value 0. Thus, memory is consumed to store a bit-vector with a value of 0, having a length of 98, which is unnecessary.

Mining inter-sequence patterns with constraints

- The ISP-IC (T. Le et al., 2018) algorithm was initially introduced for mining inter-sequence patterns with item constraints. ISP-IC represents an extension of both the EISP-Miner and DBV-ISP algorithms. To store frequent pattern positions, ISP-IC utilizes the DBV-PatternList data structure. An enhanced version of the ISP-IC algorithm, known as *i*ISP-IC, was proposed to improve process performance. Additionally, an advanced iteration, *pi*ISP-IC, was introduced by implementing parallel execution techniques to expedite the algorithm. These algorithms all harness the advantageous features of the DBV-PatternList data structure. Nevertheless, the challenge of addressing duplicate data remains unresolved. Furthermore, ISP-IC currently only handles item conditions, while other conditions, such as itemsets, have yet to be introduced.

The research methods mentioned above for mining inter-sequence patterns primarily focus on small databases and have not been tested on larger databases containing 1,000,000 transactions or more. For instance, the EISP-Miner algorithm examines databases with a maximum of 100,000 transactions and 1,000 distinct items. The DBV-ISP algorithm is tested with databases containing 1,000 transactions and 1,000 distinct items. Lastly, the ISP-IC algorithm has only been tested with a maximum of 100,000 transactions and 1,000 distinct items.

Furthermore, in recent years, new data structures have been proposed and applied to clickstream pattern mining and sequential pattern mining, leading to significantly improved efficiency.

The advantages of the pseudo-IDList structure are as follows

- Compact Information: The IDList pseudo-structure contains highly condensed information, encompassing only three values: P, DIP, and a pattern location matrix (detailed in Chapter 3).
- Efficient Support Calculation: Support values for patterns can be rapidly computed based on the matrix's row count.
- Optimized Storage: The matrix solely retains the occurrence positions of patterns relative to the frequent patterns that generate them. This selective storage minimizes data redundancy and reduces storage space utilization within the algorithm.
- Broad Applicability: To date, numerous authors have applied the pseudo-IDList structure to various frequent pattern mining methods, resulting in notable efficiency enhancements. Notable examples include CUP (Huynh et al., 2020b), CM-WSPADE and Compact-SPAD (Huynh et al., 2020a), SUI (Huynh et al., 2022), PF-CUP (Huynh et al., 2023).

Based on the developments and challenges encountered in inter-sequence pattern mining, the main motivations of this thesis are as follows:

- Addressing growing data volume challenges: As the volume of data collected and analyzed continues to grow rapidly, there is an increasing need for scalable and efficient inter-sequence pattern mining algorithms. This thesis presents and develops algorithms, techniques, and recommendations that can process large-scale datasets while maintaining high performance in terms of both time and storage space.

- Introducing the inter-sequence pattern mining problem with itemset constraints: Inter-sequence pattern mining generates a significantly larger number of candidates compared to sequential pattern mining. Users often require specific knowledge, rather than a complete overview of the database, which necessitates mining under certain search conditions. Therefore, this thesis proposes the problem of inter-sequence pattern mining with itemset constraints.
- Overcoming limitations of existing inter-sequence pattern mining algorithms: Existing inter-sequence pattern mining algorithms have limitations related to data duplication, storage space, and processing time. This thesis aims to address these limitations by proposing new and suitable data structures to eliminate data duplication during the mining process and introducing pruning methods to speed up the inter-sequence pattern mining algorithm.
- Applying parallel processing models to inter-sequence pattern mining problems: Inter-sequence pattern mining models often require lengthy processing times. Parallel mining techniques have the potential to significantly speed up the data mining process. This thesis explores the application of parallel processing models to inter-sequence pattern mining problems to accelerate the algorithm while ensuring the accuracy of the results obtained.
- Contributing to the theoretical basis for inter-sequence pattern mining: The thesis provides a comprehensive overview of data mining models, such as sequential pattern mining, clickstream pattern mining, and inter-sequence pattern mining. From this foundation, the thesis presents propositions, conclusions, and suggestions to help improve the efficiency of inter-sequence pattern mining algorithms.

1.3 Research Problem

The problem of inter-sequence pattern mining is an extension of sequential mining, with three extensions including itemset, sequence, and inter. When dealing with large sequential databases and a high user-specified *maxspan* value, a large number of candidate patterns are generated. Therefore, the inter-sequence pattern mining problem requires optimization of the candidate generation process as well as optimization of storage space.

In the process of frequent inter-sequence pattern mining, algorithms such as EISP-Miner (C. S. Wang & Lee, 2009) and DBV-ISP (Vo et al., 2012) generate a large number of patterns. This poses challenges in knowledge discovery and requires significant storage resources. To address these issues and restrict knowledge search based on user-defined

criteria, we introduce, for the first time in Chapter 4, the application of itemset constraints to the inter-sequence pattern mining problem. This innovation helps reduce the volume of frequent patterns generated, resulting in faster mining and the rapid delivery of essential knowledge to users.

Furthermore, while the DBV-ISP (Vo et al., 2012) algorithm has introduced methods for optimizing storage space, it remains limited in its ability to address the problem of duplicate data. To overcome these shortcomings and further enhance storage optimization, we propose the novel application of the pseudo-IDList data structure to the inter-sequence pattern mining problem, which is detailed in Chapter 3.

1.4 Aim of this Thesis

The aim of this thesis is to address the limitations of inter-sequence pattern mining in terms of processing time and storage space. The thesis proposes a novel storage data structure for the inter-sequence pattern mining problem, aiming to minimize data duplication during the mining process. Additionally, it introduces an inter-sequence pattern mining model with itemset constraints to reduce the number of generated candidates, thus accelerating the search and processing of relevant information. Moreover, the thesis presents additional propositions to enhance the efficiency of the proposed methods and algorithms.

1.5 Objectives of this Thesis

The objectives of this thesis are as follows

1. Proposing a solution to address the problem of mining inter-sequence patterns with itemset constraints, introducing the DBV-ISPMIC algorithm.
2. Developing an optimal approach for solving the inter-sequence pattern mining problem with itemset constraints, presenting the *p*DBV-ISPMIC algorithm.
3. Introducing a method for optimizing storage space in the context of the inter-sequence mining problem, utilizing the ISP-PI algorithm.
4. Proposing a candidate pruning technique for the inter-sequence pattern mining problem, incorporating the ISP-IC (Inter-Sequence Pattern mining with Index Intersection Checking) method.
5. The proposed algorithms and methods will be evaluated through experiments using real-world databases sourced from the data mining community's data warehouse. The experimental results will be compared based on the algorithm's running time and memory usage requirements.

1.6 Thesis Contributions

Based on the aim of the thesis, the main contributions are presented in two sections and are briefly outlined as follows

A. The First Contribution Addresses Objectives 1 and 2 of the Thesis

Drawing from proposed inter-sequence mining problems such as EISP-Miner (C. S. Wang & Lee, 2009), DBV-ISP (Vo et al., 2012), and ISP-IC (T. Le et al., 2018), as well as sequential pattern mining problems with itemset constraints like MSPIC-DBV (Van et al., 2018a), we introduce a problem of inter-sequence pattern mining with itemset constraints, named DBV-ISPMIC (Nguyen et al., 2023). The algorithm employs a data structure called DBV-PatternList to store candidates, along with a tree structure named ISP-Tree to store frequent patterns. Additionally, we propose a method for quickly checking the condition of generated candidate itemsets and apply parallel mining to accelerate the algorithm.

The DBV-PatternList data structure optimizes candidate information storage. Instead of using a numeric data type to represent pattern information, DBV-PatternList utilizes a bit-vector data structure. Pattern information is indicated by turning bits on or off, allowing a numeric data type to store more candidate information, thus reducing the space needed for candidates.

Checking itemset constraints for all generated samples is time-consuming for the algorithm. We suggest a method to rapidly verify that the generated candidate meets itemset constraints, using the condition information from the parent patterns that created it. This decreases the algorithm's running time.

The inter-sequence pattern mining problem employs the ISP-Tree structure to store generated frequent patterns, processing the algorithm according to the depth-first traversal method. As the handling of branches on the tree is separate, we present a parallel processing technique for branches on the tree. This enables the algorithm to optimize runtime by processing multiple branches simultaneously.

To evaluate the proposed algorithm, we used five datasets: C6T5S4I4N1kD1k, C6T5S4I4N1kD10k, Gazelle, BIKE, and BMSWebView1. C6T5S4I4N1kD1k and C6T5S4I4N1kD10k are two databases generated by the IBM synthetic data generator tool. Gazelle and BMSWebView1 are two clickstream databases, while BIKE is a database of Bike Share data from LA Metro. The tests were conducted to compare the latest cross-chain pattern mining algorithm in terms of runtime and memory usage. The input setup parameters for running the algorithm on the databases are outlined in Table 1.2. The user-defined maximum span value is denoted by $1 \leq maxspan \leq 5$.

Table 1.2. Test database information for evaluation of the DBV-ISPMIC algorithm.

Database name	*Minsupport* (%)
C6T5S4I4N1kD1k	0.5
C6T5S4I4N1kD10k	5
Gazelle	1
BIKE	0.5
BMSWebView1	0.5

Experimental results demonstrated that the proposed algorithm outperforms previous ones. For example, with the C6T5S4I4N1kD1k database ($minsupport = 0.5\%$, $maxspan = 5$), the DBV-ISPMIC algorithm runs 51% faster and uses 12% less memory than the Post-EISPMiner algorithm. When the quick test method for itemset constraints is applied, the DBV-ISPMIC algorithm runs 62% faster and uses 14% less memory than the Post-EISPMiner algorithm.

With the C6T5S4I4N1kD10k database ($minsupport = 5\%,\ maxspan = 5$), the DBV-ISPMIC algorithm runs 47% faster and uses 6% less memory than the Post-EISPMiner algorithm. When the quick test for itemset constraints is applied, the DBV-ISPMIC algorithm runs 58% faster and uses 10% less memory than the Post-EISPMiner algorithm.

With the Gazelle database ($minsupport = 1\%,\ maxspan = 5$), the DBV-ISPMIC algorithm runs 18% faster and uses 14% less memory than the Post-EISPMiner algorithm. When the quick test for itemset constraints is applied, the DBV-ISPMIC algorithm runs 31% faster and uses 15% less memory than the Post-EISPMiner algorithm.

With the BIKE database ($minsupport = 0.5\%,\ maxspan = 5$), the DBV-ISPMIC algorithm runs 17% faster and uses 7% less memory than the Post-EISPMiner algorithm. When the quick test for itemset constraints is applied, the DBV-ISPMIC algorithm runs 47% faster and uses 12% less memory than the Post-EISPMiner algorithm.

With the BMSWebView1 database ($minsupport = 0.5\%,\ maxspan = 5$), the DBV-ISPMIC algorithm runs 3% faster and uses 8% less memory than the Post-EISPMiner algorithm. When the quick test for itemset constraints is applied, the DBV-ISPMIC algorithm runs 7% faster and uses 14% less memory than the Post-EISPMiner algorithm.

B. The Second Contribution Addresses Objectives 3 and 4 of the Thesis

Building upon inter-sequence pattern mining algorithms like EISP-Miner and DBV-ISP, we propose a novel algorithm called ISP-PI (Inter-Sequence Pattern mining

based on Pseudo-Index). This algorithm aims to optimize data mining models in the context of inter-chain mining using a data structure known as pseudo-IDList. The ISP-PI addresses the shortcomings of previous algorithms concerning data duplication. Instead of requiring storage for all the information of a candidate, we can retrieve its information from the original pattern. This method compresses the position values of the generated candidates, allowing for the retrieval of values from the original patterns that produced the candidates and eliminating the need to save all positions. Furthermore, the algorithm incorporates a pruning method named ISP-IC (Inter-Sequence Pattern mining with Index intersection Checking) to effectively reduce the number of generated candidates. This optimization enhances processing time and storage space, which is crucial due to the growing volume of collected data. The ISP-PI algorithm efficiently compresses data to minimize storage space and employs candidate pruning to accelerate the algorithm's runtime in inter-sequence pattern mining.

Experimental results indicate that the proposed ISP-PI algorithm surpasses the most advanced algorithms for mining inter-sequence patterns (MISPs) in terms of processing time and storage space utilization. Consequently, this contribution signifies a considerable advancement in the field of data mining, particularly for inter-chain pattern mining.

The algorithm evaluation was conducted on six sample datasets, including C150S40T2, C200S12T5, FIFA, BMSWebView2, Kosarak, and MSNBC. Kosarak and MSNBC are two large databases, each containing nearly 1 million rows of data. C150S40T2 and C200S12T5 were generated using the standard generator in (Agrawal & Srikant, 1995), while FIFA, BMSWebView2, Kosarak, and MSNBC are actual databases. C150S40T2, C200S12T5, and FIFA are dense databases with average sequence lengths of 76.64, 51.57, and 36.24, respectively. The input configuration parameters for executing the algorithm on the databases are presented in Table 1.3. The maximum span value, defined by the user, ranges from 1 to 5 inclusively.

Table 1.3. The database information for evaluation of the ISP-PI algorithm.

Database name	*Minsupport* (%)
C150S40T2	6
C200S12T5	3
BMSWebView2	0.02
FIFA	9
Kosarak	0.6
MSNBC	0.2

Experimental results indicate that the proposed ISP-PI algorithm outperforms the two previous inter-chain data mining algorithms, Post-EISPMiner and Post-DBV-ISP.

For the C150S40T2 database $(minsupport = 6\%, maxspan = 5)$, the ISP-PI algorithm is 66% faster than Post-EISPMiner and 11% faster than Post-DBV-ISP. ISP-PI uses 78% less memory than Post-EISPMiner and 76% less than Post-DBV-ISP.

For the C200S12T5 database $(minsupport = 3\%, maxspan = 5)$, the ISP-PI algorithm is 60% faster than Post-EISPMiner and 29% faster than Post-DBV-ISP. ISP-PI uses 88% less memory than Post-EISPMiner and 87% less than Post-DBV-ISP.

For the BMSWebView2 database $(minsupport = 0.02\%, maxspan = 5)$, the ISP-PI algorithm is 76% faster than Post-EISPMiner and 75% faster than Post-DBV-ISP. ISP-PI uses 92% less memory than Post-EISPMiner and 92% less than Post-DBV-ISP.

For the FIFA database $(minsupport = 9\%, maxspan = 5)$, the ISP-PI algorithm is 90% faster than Post-EISPMiner and 87% faster than Post-DBV-ISP. ISP-PI uses 81% less memory than Post-EISPMiner and 77% less than Post-DBV-ISP.

For the Kosarak database $(minsupport = 0.6\%, maxspan = 5)$, the ISP-PI algorithm is 81% faster than Post-EISPMiner and 58% faster than Post-DBV-ISP. ISP-PI uses 82% less memory than Post-EISPMiner and 81% less than Post-DBV-ISP.

For the MSNBC database $(minsupport = 0.2\%, maxspan = 5)$, the ISP-PI algorithm is 85% faster than Post-EISPMiner and 79% faster than Post-DBV-ISP. ISP-PI uses 56% less memory than Post-EISPMiner and 54% less than Post-DBV-ISP.

1.7 Structure of the Thesis

The remaining chapters of the thesis are organized as follows:

Chapter 2: Literature Overview. This chapter provides definitions, examples, and methods related to sequential pattern mining, including algorithms such as AprioriAll, FreeSpan, SPADE, PrefixSpan, PRISM, and CM-SPADE. Additionally, definitions, examples, and methods related to sequential pattern mining on item and itemset constraints are presented, including algorithms such as MSRIC-R, MSRIC-P, MSPIC-DBV, MWAPC, and EMWAPC. The chapter also covers definitions, examples, and methods related to clickstream data mining, with algorithms like CUP, CM-WSPADE, Compact-SPADE, and SUI. Lastly, it presents definitions, examples, and methods related to inter-sequence pattern mining, featuring algorithms such as EISP-Miner, DBV-ISP, ISP-IC, *i*ISP-IC, and *pi*ISP-IC.

Chapter 3: Presentation of the inter-sequence data mining problem using the pseudo-IDList data structure. This chapter introduces the ISP-PI (Inter-Sequence Pattern

mining based on Pseudo-Index) algorithm and a pruning method named ISP-IC (Inter-Sequence Pattern mining with Index Intersection Checking).

Chapter 4: Presentation of the inter-sequence pattern mining problem with itemset constraints and the dynamic bit vector data structure. This chapter presents the DBV-ISPMIC algorithm, an improved version called DBV-ISPMIC-IMPROVING, and a parallel mining algorithm named *p*DBV-ISPMIC.

Chapter 5: Conclusion and limitations of the proposed algorithms. This section discusses research approaches or improved techniques for the proposed algorithms that could be explored in the future. Additionally, a list of our publications is provided at the end of the chapter.

CHAPTER 2: LITERATURE REVIEWS

In this chapter we present an introduction to sequential pattern mining and inter-sequential pattern mining problems, including fundamental concepts and the descriptions of commonly used methods. Additionally, we examine the strengths and weaknesses of existing mining algorithms for both problems. In order to address these issues, we put fourth efficient algorithms for inter-sequence pattern mining that take into account considerations such as memory usage and running time. The findings of our investigation will be presented in the forthcoming Chapters 3 and 4.

2.1 Methods for Mining Sequence Patterns

2.1.1 Basic Concepts

Definition 2.1 (*Customer transaction*): Database D is a representation of customer transactions, wherein each transaction includes the following information: customer ID, transaction date, transaction time, and transaction details. Transaction details encompass a collection of items that were acquired during the transaction. No customer can have more than one transaction sharing the same combination of transaction date and time.

An illustration of a customer transaction and a customer sequence is provided in Table 2.1 and Table 2.2, respectively.

Table 2.1. An example of customer transaction.

Customer Id	Transaction Date	Transaction Time	Transaction Details
1	12.12.1998	9:00	C
1	19.12.1998	10:00	A, B
2	13.12.1998	9:00	C
2	21.12.1998	14:00	A, B, C
2	24.12.1998	15:00	A
3	14.12.1998	10:00	A
3	24.12.1998	11:00	D
4	15.12.1998	15:00	A
4	25.12.1998	16:00	D

Table 2.2. An example of customer sequence from Table 2.1.

Customer Id	Customer Sequence
1	$\langle(C)(AB)\rangle$
2	$\langle(C)(ABC)(A)\rangle$
3	$\langle(A)(D)\rangle$
4	$\langle(A)(D)\rangle$

Definition 2.2 (*Items, Itemsets, Sequences*)**:** Let $I = \{i_1, i_2, i_3, \dots, i_n\}$ be a set of n distinct products, also called items. A sequence $s = \langle t_1, t_2, t_3, \dots, t_m\rangle$ is an ordered list of itemsets where $t_i \subseteq I (1 \leq i \leq m)$ is an itemset. The number of items contained in the itemset is called its size. The size of a sequence is the number of itemsets in a sequence, denoted as *k*-sequence $(1 \leq k \leq m)$. An itemset begins with symbol "(" and ends with symbol ")", if an itemset contains a single item then the brackets can be omitted. A sequence begins and ends with the symbols "⟨", "⟩", respectively.

For example, based on Table 2.2, let $I = \{A, B, C, D\}$, two possible itemsets of size two (2-itemsets) are (AB), (AC). An example of 2-sequence is $\langle C(AB)\rangle$ or $\langle AD\rangle$ and 3-sequence is $\langle C(ABC)A\rangle$.

Definition 2.3 (*Subsequences, Supersequences*)**:** Let $S_\sigma = \langle x_1, x_2, x_3, \dots, x_n\rangle$ and $S_\beta = \langle y_1, y_2, y_3, \dots, y_m\rangle$ be two sequences $(n \leq m)$. S_σ is called a subsequence of S_β or S_β is called the supersequence of S_σ, if there exists integers $1 \leq j_1 \leq j_2 \leq j_3 \leq \dots \leq j_n$ such that $x_1 \subseteq y_{j1}, x_2 \subseteq y_{j2}, x_3 \subseteq y_{j3}, \dots, x_n \subseteq y_{jn}$, denoted as $S_\sigma \sqsubseteq S_\beta$.

For example, the sequence $\langle C(AB)\rangle$ is a subsequence of $\langle C(ABC)A\rangle$, since C $\subseteq$ C and AB $\subseteq$ ABC, and the order of itemset is preserved. However, the sequence $\langle C(AB)\rangle$ is not a subsequence of $\langle CAB\rangle$ and vice versa.

Definition 2.4 (S*upport threshold*)**:** The support value of a sequence, denoted $minsupport\ (S, D)$, is calculated by the total number of occurrences of the sequence in the database D, given as $support(S, D) = |\{S_i \in D | S \sqsubseteq S_i\}|$.

For instance, based on Table 2.1, $support(\langle A\rangle, D) = 4$, $support(\langle(AB)\rangle, D) = 2$.

2.1.2 Sequence Pattern Mining

2.1.2.1 AprioriAll (Agrawal & Srikant, 1995)

The AprioriAll algorithm (Agrawal & Srikant, 1995) solves the problem of mining sequential pattern over a large database of customer transactions. The pseudo-code is presented in Algorithm 2.1. The AprioriAll algorithm is described through the following main steps:

Step 1: Finding the set of frequent 1-itemsets. (Algorithm 2.1, line 1).

Step 2: Traversing the frequent set 1-itemsets $(k = 1)$ in step 1 (Algorithm 2.1, line 2), the algorithm generates $(k + 1)$-itemset candidates (Algorithm 2.1, line 4). The algorithm scans the database again to calculate the support value for each candidate. If its $minsupport$ value is greater than or equal to the minimum support threshold, the candidate is a frequent pattern and stores it in the L_k list (Algorithm 2.1, line 5 to 7).

Step 3: Repeating step 2 until no more new candidates are created, the algorithm stops here.

Algorithm 2.1. AprioriAll Algorithm (Agrawal & Srikant, 1995)

1. L_1 = {large 1-sequence}
2. **for** (k=2; $L_{k-1} \neq 0$; k++) **do**
3. **begin**
4. C_k = New candidates generated from L_{k-1} (see Algorithm 2.2).
5. **foreach** customer-sequence c in the database **do**
6. Increment the count of all candidates in C_k that are contained in c.
7. L_k = Candidates in C_k satisfy minimum support.
8. **end**
9. Answer = Maximal Sequences in $U_k L_k$;

Algorithm 2.2. Apriori-generate function (Agrawal & Srikant, 1995)

1. **insert into** C_k
2. **select** $p.litemset_1$, …, $p.litemset_{k-1}$,$q.litemset_{k-1}$
   from $L_{k-1}p$, $L_{k-1}q$
   where $p.litemset_1 = q.litemset_1$,…, $p.litemset_{k-2} = q.litemset_{k-2}$;

Large 2-sequences	Candidate 3-sequences (after joining)	Candidate 3-sequences (after pruning)
⟨1 2⟩	⟨1 2 4⟩	⟨1 2 4⟩
⟨1 4⟩	⟨1 2 5⟩	⟨1 2 5⟩
⟨1 5⟩	⟨1 4 2⟩	⟨1 4 5⟩
⟨2 4⟩	⟨1 5 2⟩	⟨2 4 5⟩
⟨2 5⟩	⟨1 5 4⟩	
⟨4 5⟩	⟨2 4 5⟩	
	⟨2 5 4⟩	

Figure 2.1. Candidate Generation.

The apriori-generate function is described in Algorithm 2.2. It generates the set of k-sequences from the set of all large $(k-1)$-sequences. If the new candidate is not in the L_{k-1} set, then this candidate will be discarded. With this improvement, the Apriori algorithm does not calculate the support for these candidates, which increases the processing speed of the algorithm. Consider the example in Figure 2.1, the first column is the set of all large 2-sequences, after running the Apriori-generate function we get the set of 3-sequences shown in the second column. After pruning out sequences whose subsequences are not in L_2, the sequences shown in the third column. For instance, ⟨1 4 2⟩ is removed because the subsequence ⟨1 4 2⟩ is not in L_2.

The disadvantage of ApirioriAll algorithm is that it has to rescan the database many times to calculate the support (in Algorithm 2.1, line from 5 to 7).

⟨(1 3)(2)(4)(5)⟩
⟨(1)(2)(4)(3 5)⟩
⟨(1)(2)(4)(5)⟩
⟨(1)(2)(5)⟩
⟨(3)(5)⟩

Figure 2.2: Customer Sequences.

L_2	
1-Sequences	Support
⟨1⟩	4
⟨2⟩	4
⟨3⟩	3
⟨4⟩	3
⟨5⟩	5

L_2	
2-Sequences	Support
⟨1 2⟩	4
⟨1 4⟩	3
⟨1 5⟩	4
⟨2 4⟩	3
⟨2 5⟩	4
⟨4 5⟩	3

L_3	
3-Sequencess	Support
⟨1 2 4⟩	3
⟨1 2 5⟩	4
⟨1 4 5⟩	3
⟨2 4 5⟩	3

L_4	
4-Sequences	Support
⟨1 2 4 5⟩	3

Figure 2.3: Large Sequences.

Consider a customer sequence dataset in Figure 2.2, the minimum support has been specified to be 60% ($minsupport = 3$). The set of large k-sequences are presented in Figure 2.3.

The Apriori algorithm has several disadvantages, primarily related to processing time and storage space. These limitations arise due to the following factors:

- Generation of a large set of candidate sequences in a large sequence database.
- Multiple scans of the database during mining.
- Difficulty in mining long sequential patterns using the Apriori-based method.

Consequently, the AprioriAll algorithm proves to be inefficient when dealing with large volumes of data sets. For instance, if we assume a frequent 1-pattern with a count of 10^4 in the dataset, the Apriori algorithm needs to generate more than 10^7 candidates of 2-patterns. These candidates are then tested and collected cumulatively. It becomes evident that the number of candidates generated using the Apriori-like algorithms can be exponential in the worst case. For example, if there is a frequent sequence of 100 elements, generating such a sequence would require generating 2^{100} candidates. This serves as an illustration of the application of the Apriori algorithm.

Therefore, the candidate generation phase, which contributes to the time complexity of the Apriori algorithm, incurs significant costs and time consumption. Additionally, during the execution, the algorithm recalculates candidate support by rescanning the original database multiple times. This further impacts the algorithm's efficiency, particularly when the system memory is insufficient, and there is a large number of frequent transactions. Consequently, the Apriori algorithm becomes inefficient and slow when working with large databases.

2.1.2.2 **FreeSpan** (Han et al., 2000)

The FreeSpan (for **Fre**qu**e**nt pattern-projected **S**equential **pa**ttern **n** mining) algorithm (Han et al., 2000) aims to integrate the mining of frequent sequences with that of frequent patterns and uses projected sequence databases to confine the search and growth of the subsequence fragments.

Let $S = \langle x_1, x_2, x_3, \dots, x_n \rangle$ be a sequence, the itemset $x_1 \cup x_2 \cup x_3 \cup \dots \cup x_n$ is called *S*'s projected itemset. FreeSpan is based on the following property: If an itemset X is infrequent, any sequence whose projected itemset is a superset of X cannot be a sequential pattern.(Pei et al., 2001)

By using projected sequence databases, FreeSpan greatly reduces the generation of candidate sub-sequences. FreeSpan finds the frequent 1-itemsets, termed an f_list, by scanning the sequence database, and then sorts them into support descending order. Let $f\_list = \langle y_1, y_2, y_3, \dots, y_m \rangle$ be a list of all frequence 1-itemsets. This set can be divided into m subjects: those having item y_1, those having item y_2 but no item in $\{y_3, \dots, y_m\}$, those having item y_3 but no in $\{y_4, \dots, y_m\}$, and so on. In general, the j_{th} subset ($1 \leq j \leq m$) is the set of sequential patterns having item y_j but no item in $\{y_{j+1}, y_{j+2}, \dots, y_m\}$. The FreeSpan algorithms is presented as follows:

Algorithm 2.3. The FreeSpan Algorithm (Han et al., 2000)

1. **Input:** A sequence database D, the support threshold ($minsupport$)
2. **Output:** The set of frequent patterns
3. Scan D, find the set of frequent 1-sequence in D, and (infrequency descending order) sort them into f_list
4. Perform alternative-level projection mining which consists of the following steps:
5. Construct a frequent item matrix by scanning the database once.
6. Generate length-2 sequential patterns and the annotations on item-repeating patterns and projected database.
7. Scan database to generate item-repeating patterns and projected databases
8. Do matrix projection mining on projected database recursively, if there are still longer candite pattern to be mined.

Based on the analysis above, we can identify the following strengths and weaknesses of the FreeSpan algorithm:

Advantages of FreeSpan:

- A significant strength of FreeSpan is its capability to search smaller projected databases in each subsequent projection. This is achieved through recursive projection of a large sequence database into smaller ones, guided by current and future mining of frequent item patterns. Consequently, each expected database

is constrained to a reduced candidate pool. Furthermore, FreeSpan only necessitates three scans of the original database, irrespective of the maximum sequence length.

Disadvantages of FreeSpan:

- The primary overhead of FreeSpan is the necessity to generate numerous nontrivial projected databases. In cases where a pattern occurs in every sequence within a database, its projected database does not significantly reduce in size, except for the removal of some infrequent items. Furthermore, because a length-k subsequence can expand at any position, the search for length-(k+1) candidate sequences require the examination of all possible combinations, incurring considerable computational costs.

2.1.2.3 **SPADE** (Zaki, 2001)

The SPADE (**S**equential **PA**ttern **D**iscovery using **E**quivalence classes) algorithm was introduced by Zaki (Zaki, 2001) for fast mining of sequential patterns in large databases. Previous approaches (Agrawal & Srikant, 1995; Han et al., 2000) had to scan the database many times or use a complex hash-tree structure to store frequent pattern information. The SPADE algorithm uses an equivalence classes approach to store frequent patterns. The maximum number of database scans of the SPADE algorithm is only 3 times, one for frequent 1-sequences, another for frequent 2-sequences, and lastly for generating all other frequent sequences. The SPADE algorithm is presented as following:

Algorithm 2.4. The SPADE Algorithm (Zaki, 2001)

1. **SPADE** ($minsupport$, D)
2. F_1 = {frequent items or 1-sequences};
3. F_2 = {frequent 2-sequences};
4. ε = {equivalence classes [X]θ1}
5. for all [X] ∈ ε do **Enumerate-Frequent-Seq**([X]); //**Algorithm 2.5**

Algorithm 2.5. Pseudo-code for breadth-first and depth-first search (Zaki, 2001)

Enumerate-Frequent-Seq(S)

1. **for** all atoms A_i ∈ S **do**
2. $T_i = \emptyset$;
3. **for** all atoms $A_j \in S$, with $j \geq i$ **do**
4. $R = A_i \vee A_j$;
5. **if** (**Prunce**(R) == FALSE) **then //Algorithm 2.6**

```
6.          L(R) = L(A_i) ∩ L(A_j);
7.          if σ(R) ≥ minsupport then
8.              T_i = T_i ∪ {R}; F_|R| = F_|R| ∪ {R};
9.      end
10.     if(Depth-First-Search) then Enumerate-Frequent-Seq(T_i);
11. end
12. if (Breadth-First-Search) then
13.     for all T_i ≠ ∅ do Enumerate-Frequent-Seq(T_i);
```

Algorithm 2.6. Sequence pruning (Zaki, 2001)

```
Prune(β)
1.  for all (k − 1)-subsequences, α ≺ β do
2.      if ([α_1] has been proceesed, and α ∉ F_{k−1}) then
4.          return TRUE;
5.  return FLASE;
```

The SPADE algorithm introduces a new approach compared to previous algorithms, offering notable advantages and limitations:

Advantages of SPADE:

- Unlike previous approaches that involve multiple database scans and complex hash tree structures with suboptimal locations, SPADE decomposes the original problem into smaller subproblems using equivalence classes based on frequent patterns Each equivalence class can be solved independently and is likely to be processed efficiently in main memory. As a result, SPADE typically performs only three database scans: one for frequent 1-patterns, another for common 2-patterns, and an additional scan to generate all other frequent patterns.
- Additionally, the SPADE algorithm utilizes a Depth-first search method, which allows for the application of parallel computation techniques.

Disadvantages of SPADE:

- A limitation of the SPADE algorithm lies in the storage of pattern locations as integers, which requires a significant amount of storage space for large databases.
- After a frequent pattern is generated, the algorithm stores all the information about the pattern's position. This leads to redundancy and data duplication, necessitating the use of more storage space than required. Recent studies have

demonstrated that it is possible to retrieve the location information of a frequent pattern from the frequent patterns that generate it (Huynh et al., 2020b, 2022).

2.1.2.4 **PrefixSpan** (Pei et al., 2001)

The PrefixSpan algorithm (Pei et al., 2001) was based on the concept of FreeSpan (Han et al., 2000), but instead of projection sequence database it investigates the prefix subsequences (Definition 2.5) and projects only their corresponding suffix subsequences (Definition 2.6) into projected databases. The advantage of algorithm is designed to only consider patterns that exist in the database.

Definition 2.5 (*Prefix*)(Pei et al., 2001): Suppose all the items within an element are listed alphabetically. Given a sequence $\alpha = \langle x_1, x_2, x_3, \dots, x_n \rangle$ (where each x_i corresponds to a frequent element in S), a sequence $\beta = \langle x'_1, x'_2, x'_3, \dots, x'_m \rangle (m \leq n)$ is call a prefix of α if and only if 1) $x'_i = x_i$ for $(i \leq m-1)$; 2) $x'_m \subseteq x_m$; and 3) all the frequent items in $(x_m = x'_m)$ are alphabetically after those in x'_m.

For instance, $\langle A \rangle, \langle AA \rangle, \langle A(AB) \rangle,$ and $\langle A(ABC) \rangle$ are prefixes o squence $S = \langle A(ABC)(AC)D(CF) \rangle$.

Definition 2.6 (*Suffix*)(Pei et al., 2001): Given a sequence $\alpha = \langle x_1, x_2, x_3, \dots, x_n \rangle$ (where each x_i corresponds to a frequent element in S). Let $\beta = \langle x_1, x_2, x_3, \dots, x_{m-1} x'_m \rangle (m \leq n)$ be the prefix of α. Sequence $\gamma = \langle x''_m, x_{m+1}, x_{m+2}, \dots, x_n \rangle$ is called the suffix of α with regards to prefix β, denoted as $\gamma = \alpha/\beta$, where $x''_m = (x_m - x'_m)$. We also denote $\alpha = \gamma.\beta$. Note, if β is not a subsequence of α, the suffix of α with regards to β is empty.

For example, for the sequence $S = \langle A(ABC)(AC)D(CF) \rangle$, $\langle (ABC)(AC)D(CF) \rangle$ is the suffix with regards to the prefix $\langle A \rangle$, $\langle (\_BC)(AC)D(CF) \rangle$ is the suffi with regards to the prefix $\langle AA \rangle$, and $\langle (\_C)(AC)D(CF) \rangle$ is the suffix with regards to the prefix $\langle A(AB) \rangle$.

The PrefixSpan algorithm is presented as follows:

Algorithm 2.7. The PrefixSpan Algorithm (Pei et al., 2001)

Input: A sequence database S, and the minimum support threshold $minsupport$.

Output: The set of frequent patterns

Method: Call $PrefixSpan(\langle\rangle, 0, S)$

Subroutine $PrefixSpan(\alpha, l, S|_\alpha)$

1. The parameters are 1) α is a sequential pattern; 2) l is the length of α; and 3) $S|_\alpha$ is the α-projected database if $\alpha \neq \langle\rangle$, otherwise, it is the sequence database S.

   Method:

2. 1.Scan $S|_\alpha$ once, find each frequent item, b, such that

3. (a) b can be assembled to the last element of α to form a sequential pattern;

or

4. (b) $\langle b \rangle$ can be appended to α to form a sequential pattern

5. 2.For each frequent item b, append it to α to form a sequential pattern α', and output α'.

6. 3.For each α', construct α'-projected database $S|_{\alpha'}$, and call $PrefixSpan(\alpha', l+1, S|_{\alpha'})$

After analyzing the PrefixSpan algorithm, we can identify its strengths compared to previous algorithms, as well as the limitations that need to be addressed:

Advantages of PrefixSpan:

- The main advantage of the PrefixSpan algorithm is that it eliminates the need to create or test candidate sequences that do not exist in the expected database. The sample growth method employed by PrefixSpan ensures that only patterns occurring in the database are discovered. In other words, PrefixSpan extends shorter sequential patterns to generate longer ones, reducing the search space. This approach significantly reduces the cost of constructing the expected databases.
- Two optimization techniques can further enhance the efficiency of PrefixSpan. Firstly, using a two-level projection can decrease the size and number of projected databases. Secondly, employing pseudo-projection can reduce overhead by storing the projected databases entirely in main memory. Moreover, PrefixSpan proves to be efficient as it leverages the complete set of patterns and runs even faster than the FreeSpan (Han et al., 2000) algorithm.

Disadvantages of PrefixSpan:

- One drawback of the PrefixSpan algorithm is the potential runtime cost associated with scanning the database multiple times and creating database projections.
- Additionally, creating database projections can consume a considerable amount of memory if not implemented efficiently. In the worst case, it may require nearly duplicating the entire database for each database projection, leading to significant memory usage.

2.1.2.5 **PRISM** (Gouda et al., 2007, 2010)

Introduced by Gouda et al. in 2007, the PRISM (**PRI**me-Encoding Based **S**equence **M**ining) algorithm is designed for mining frequent sequences (Gouda et al., 2007). The algorithm employs a vertical approach to enumeration and support counting, which relies on the innovative concept of prime block encoding, a method grounded in prime

factorization theory. Gouda et al. further elaborated and enhanced the PRISM algorithm in 2010 (Gouda et al., 2010). When compared with earlier algorithms for sequential pattern mining, such as SPAM (Ayres et al., 2002), PrefixSpan (Pei et al., 2001), and SPADE (Zaki, 2001), the PRISM algorithm demonstrates its superiority in terms of both time efficiency and memory consumption.

The assessment of the PRISM algorithm is exemplified through the ensuing advantages:

- PRISM employs a vertical methodology for enumeration and support quantification, grounded in the innovative concept of prime block encoding. This concept is further anchored in the theory of prime factorization.
- The PRISM algorithm represents an augmentation of the SPADE algorithm, facilitated by the use of a bit data structure. In contrast to preceding algorithms which stored candidate information as integers, PRISM enhances optimization by employing bit data structures. This reduction in memory requirements is a notable advantage of the algorithm.
- In the realm of prime block encoding, the support of a candidate can be ascertained directly from its associated chain blocks.

2.1.2.6 **CM-SPADE** (Fournier-Viger et al., 2014)

The CM-SPADE algorithm proposed by Fournier-Viger et al. (Fournier-Viger et al., 2014) which is considered an improvement of the SPADE algorithm (Zaki, 2001). The SPADE algorithm has the disadvantage that the number of candidates generated is very large, even though they are infrequent pattterns. Therefore, the CM-SPADE algorithm has overcome the above disadvantage by using a new data structure named CMAP (Co-occurence MAP) (in Definition 2.9) for storing co-occurrence information. In addition, by using the CMAP structure and prefix-based pruning strategy, the CM-SPADE algorithm improve the mining time of mining sequential patterns. A example about the CMAP structure are shown in Table 2.3.

Definition 2.7 (*i-extension*): An item *k* is said to *succeed by i-extension* to an item *j* in a sequence $< I_1, I_2, I_3, \dots, I_n >$ iff $j, k \in I_x$ for an integer *x* such that $1 \leq x \leq n$ and $k \succ_{lex} j$.

Definition 2.8 (*s-extension*): An item *k* is said to *succeed by s-extension* to an item *j* in a sequence $< I_1, I_2, I_3, \dots, I_n >$ iff $j \in I_v$ and $k \in I_w$ for some integers *v* and *w* such that $1 \leq v < w \leq n$.

Definition 2.9 (*Co-occurrence MAP*): A *Co-occurrence MAP* (CMAP) is a structure mapping each item $k \in I$ to a set of items succeeding it. We define two CMAPs named CMAPi and CMAP*s*. CMAP*i* maps each item k to the set *cmi*(*k*) of all items $j \in I$ succeeding *k* by *i-extension* (Definition 2.7) in no less than *minsupport* sequences of SDB. CMAP*s* maps each item *k* to the set $cm_s(k)$ of all items $j \in I$ succeedings *k* by *s-extension* (Definition 2.8) in no less than $minsupport$ sequences of sequence database.

SID	Sequences
1	$\langle\{a, b\}, \{c\}, \{f, g\}, \{g\}, \{e\}\rangle$
2	$\langle\{a, d\}, \{c\}, \{b\}, \{a, b, e, f\}\rangle$
3	$\langle\{a\}, \{b\}, \{f\}, \{e\}\rangle$
4	$\langle\{b\}, \{f, g\}\rangle$

Figure 2.4: A sequence database. (Fournier-Viger et al., 2014)

Table 2.3: CMAPi and CMAPs for the database of Figure 2.4 and $minsupport$ = 2. (Fournier-Viger et al., 2014)

$CMAP_i$	
item	is succeeded by (i-extension)
a	{b}
b	∅
c	∅
e	∅
f	{g}
g	∅

$CMAP_s$	
item	is succeeded by (s-extension)
a	{b, c, e, f}
b	{e, f, g}
c	{e, f}
e	∅
f	{e, g}
g	∅

The advantages of the CM-SPADE algorithm, in comparison to previous algorithms, are highlighted as follows:

- The CM-SPADE algorithm is an enhancement of the SPADE (Zaki, 2001) algorithm. It utilizes the vertical format efficiently to calculate the support of candidate patterns, thereby avoiding costly repeated database scans.
- However, a primary limitation of vertical mining algorithms is their tendency to spend significant time evaluating candidates that either do not exist in the input database or occur infrequently. To address this issue, CM-SPADE introduces a novel data structure called CMAP, which stores co-occurrence information. By utilizing the CMAP structure, CM-SPADE can early prune candidates, improving efficiency and reducing unnecessary evaluations.

2.1.3 Sequence Pattern Mining with Constraints

2.1.3.1 MSPIC-DBV (Van et al., 2018a)

The MSPIC-DBV algorithm was proposed by Van et al. (Van et al., 2018a) for effective mining sequential patterns with itemset constraints (Definition 2.10). The MSPIC-DBV algorithm uses a dynamic bit vector data structure and a DBVS prefix-tree tree structure to reduce the time to calculate the support of generated patterns, and to optimize the running time and storage space of the algorithm. The problem of sequential pattern mining with item constraint is given as follows.

Definition 2.10 (*Problem statement of sequence pattern mining with itemset constraints*): Given a sequence database *D*, a set of constraint itemsets $C = \{c_1, c_2, c_3, \ldots, c_n\}$ and the minimum support $(minsupport)$ is specified by the user. The problem of mining sequential patterns with an itemset constraint is to find all frequent subsequences in the database which contain any itemsets in set *C* (Definition 2.11).

Definition 2.11: (*Itemset constraint satisfying*) A pattern $\beta = \langle b_1, b_2, b_3, \ldots, b_m \rangle$ is considered to contain an itemset c if $\exists i \in [1, m]$ such that $c \subseteq b_i$. Given a constraint itemset c, if pattern β contains the constraint itemset c, β is called a *c-satisfied pattern*.

2.1.3.2 MWAPC and EMWAPC (Van et al., 2018b)

The MWAPC (Mining Web Access Patterns based on super-pattern Constraint) algorithm was proposed by Van et al. [10] to solve the problem of mining web access patterns with super-pattern constraint (Definition 2.12). To avoid checking all candidate patterns that satisfy the conditions, Van also proposed an algorithm named EMWAPC.

Definition 2.12 (*Problem statement of sequence pattern mining with super-pattern constraints*): Given a web access sequence database WD, a set of constraint patterns $U = \{u_1, u_2, u_3, \ldots, u_n\}$ and the $minsupport$ is specified by the user. The problem of mining web access patterns with super-pattern constraint is to find all frequent patterns in the database which contain any pattern in U as subsequence (in Definition 2.13).

Definition 2.13: (*Constraint satisfied pattern*) Given a constraint pattern u, pattern p is called a *u-satisfied pattern* if $p \supseteq u$.

2.1.3.3 MSRIC-R and MSRIC-P (Van & Le, 2021)

In 2020, Van et al. proposed MSRIC-R and MSRIC-P algorithms, to solve the effective methods for integrating itemset constraints into the actual mining process, in which MSRIC-R pushed the constraints into the rule generating phase, and MSRIC-P pushes into the pattern mining phase.

The problem of sequential pattern mining or sequential rules mining with a set itemset constraint was proposed by Van et al. The following are the key benefits of the approach:

- The proposed algorithms are pattern-growth algorithms that utilize prefixes and dynamic bit vectors.
- The dynamic bit vector data structure optimizes the storage of frequent patterns. Candidate positions are represented by bits 1 and 0. A bit value of 1 indicates the positions where the candidate appears in the database, while a bit value of 0 marks positions where the candidate does not. This structure stores the position of the first occurrence of a 0 bit, representing the candidate's first non-occurrence. This removes the necessity to store all previous 0 bits, thereby saving memory space.
- These algorithms prune the search space both at the start and during the mining process, reducing the number of candidates that need to be checked. The pruning method is based on the prefix condition. If a candidate does not meet the itemset constraint, there's no need to extend from this candidate.

The limitation of the research direction applies exclusively to the domain of pattern mining. It is essential to diversify our research efforts to encompass other data mining challenges. The advantages mentioned above have motivated us to apply these principles to the task of mining inter-sequence patterns with itemset constraints, a topic that will be explored in greater depth in Chapter 4.

2.1.4 Clickstream Pattern Mining

2.1.4.1 **CUP algorithm** (Huynh et al., 2020)

The CUP (**C**lickstream pattern mining **U**sing **P**seudo-IDList) algorithm was presented by Huynh et al. (Huynh et al., 2020). The algorithm uses a vertical data structure called pseudo-IDList, and a heuristic pruning method named DUB (**D**ynamic intersection **U**pper **B**ound) to help optimize the algorithm. The pseudo-IDList data structure is organized to store 3 information: 1) P: information of a pattern; 2) DIP (data IDList pointer) a link to the IDList of the last item in P.

For example, if the P is $\langle A, B, C \rangle$ then DIP would point to the data IDList of 1-pattern $\langle C \rangle$; 3) M: a two-dimensional matrix contains the positions of IDList including three columns $\{Local\ id, Data\ id, Start\ index\}$.

An example of a clickstream database is shown in Figure 2.5. The first column is the UCID (user clickstream id) used to identify the user, the second column describes the

user's click patterns. Based on the example database in Figure 2.5-Figure **2.8** shows the Data IDList and pseudo-IDLists of pattern $\langle A \rangle$, $\langle B \rangle$ and $\langle C \rangle$.

UCID	User clickstream
100	$\langle A, B, C, A, A, B, C, D, B, C, B \rangle$
200	$\langle B, B, E, F, F, C, F \rangle$
300	$\langle B, B, A, D, A \rangle$
400	$\langle B, A, E, F, C, B, C \rangle$
500	$\langle D, D, A, A, B \rangle$

Figure 2.5: An example of a horizontal clickstream database.

Pattern$\langle A \rangle$		
Data ID	UCID	Position list
1	100	1, 4, 5,
2	300	3, 5
3	400	2
4	500	3, 4

(a)

Pattern$\langle A \rangle$		
DIP$\langle A \rangle$		
Local ID	Data ID	Position list
1	1	1
2	2	1
3	3	1
4	4	1

(b)

Figure 2.6: Data IDLists (a) and pseudo-IDLists of pattern $\langle A \rangle$.

Pattern$\langle B \rangle$		
Data ID	UCID	Position list
1	100	2, 6, 9, 11
2	200	1, 2
3	300	1, 2
4	400	1, 6
5	500	5

(a)

Pattern$\langle B \rangle$		
DIP$\langle B \rangle$		
Local ID	Data ID	Position list
1	1	1
2	2	1
3	3	1
4	4	1
5	5	1

(b)

Figure 2.7: Data IDLists (a) and pseudo-IDLists of pattern $\langle B \rangle$.

Pattern$\langle C \rangle$		
Data ID	UCID	Position list

Pattern$\langle C \rangle$		
DIP$\langle C \rangle$		
Local ID	Data ID	Position list

1	100	3, 7, 10
2	200	6
3	400	5, 7

(a)

1	1	1
2	2	1
3	3	1

(b)

Figure 2.8: Data IDLists (a) and pseudo-IDLists of pattern $\langle C \rangle$.

The CUP algorithm is described through 3 main steps as shown below:

- Creating IDLists for the frequent k-patterns ($\mathrm{k} = 1$).
- Generating $(k + 1)$-patterns from k-patterns.
- Calculating the pseudo-IDList for the $(k + 1)$-patterns candidates, using pseudo-IDList calculates the support. Candidates are discarded if its minimum support is less than the minimum support threshold. The process then loops back at the generating candidate step (step 2) until no candidates can be found.

2.1.4.2 CM-WSPADE and Compact-SPADE (Huynh et al., 2020)

The CM-WSPADE algorithm (Huynh et al., 2020) was proposed to solve the mining weighted clickstream patterns problem. And the improvement of the CM-WSPADE algorithm is called Compact-SPADE to improve the running time and memory usage. The CM-WSPADE algorithm is an extension of the CM-SPADE algorithm (Fournier-Viger et al., 2014), which uses the following depth-first-search strategy for mining weighted clickstream patterns. The CM-WSPADE algorithm starts by looking for 1-clickstream candidate patterns and calculates their weighted support. Any 1-clickstream candidate samples whose weighted support is lower than a certain weighted threshold is discarded. The remaining 1-clickstream patterns are frequent weighted patterns and are used to generate the 2-clickstream patterns, The algorithm discards any 2-click stream candidate samples which its weighted support is less then weight threshold. The process repeats until no new candidate sample is generated.

2.1.4.3 SUI (Sequential pattern mining Using Indices) (Huynh et al., 2022)

The pseudo-IDLists data structure (Huynh et al., 2020) has proposed to improve running time and memory usage for mining clickstream pattern, but it only uses for clickstreams data mining. Therefore, Huy et al. proposed an algorithm called SUI (Sequential pattern mining Using Indices) (Huynh et al., 2022), which have changes and improvements to be able to use the pseudo-IDLists data structure for sequence data mining.

Because the clickstream pattern has only one item per itemset, when mining clickstream pattern only has the s-extension. But by definition of mining sequential pattern

must have both s-extension and i-extension. To solve the above problem, Huynh et al. used both data-IDList and pseudo-IDLis data structures for the SUI algorithm.

The SUI mining process involves a series of steps to identify frequent patterns in a horizontal database.

- The first step of the process entails scanning the entire database horizontally and gathering all the frequent 1-patterns along with their associated data-ID lists.
- Subsequently, candidate patterns are generated in step two, which can have i-extensions, s-extensions, or both. This is done by combining two k-patterns that have the same $(k-1)$-prefix, where frequent 1-patterns share an empty prefix.
- In the third step, pseudo-IDLists and data-IDLists are generated for the candidates produced in the previous steps, and their support is checked against the minimum support requirement. Candidates whose support is found to be less than the $minsupport$ are discarded. The candidate support can be obtained from their ID-Lists instead of scanning the entire database, which is performed in step one. However, producing pseudo-IDLists and data-IDLists is known to consume a significant portion of the algorithm's runtime. The process then returns to step two and repeats until no further candidates can be found.

Through the examination of clickstream pattern mining algorithms proposed by Huy et al., the advantages can be observed as follows:

- The algorithms utilize the pseudo-IDLists data structure, an extension of the IDList data structure. Pseudo-IDLists inherits the exceptional attributes of IDList, thereby assisting in the avoidance of duplicate data storage. Pseudo-IDLists does not necessitate the storage of complete candidate position information in the database. Instead, it can be accessed through the frequent patterns that generated it. This feature significantly reduces storage space requirements.
- The pseudo-IDLists data structure stores a matrix of candidate positions, enabling expedient calculation of candidate support by examining the number of rows in the position matrix.

Building upon the exploration of the pseudo-IDLists data structure's characteristics, we have successfully applied this data structure to address the problem of inter-sequence

pattern mining. This application has resulted in improvements and optimizations to the approach for inter-sequence pattern mining, detailed in Chapter 3.

2.2 Inter-sequence Pattern Mining Algorithms

2.2.1 Basic Concepts and Definitions

Definition 2.14 (*Sequence database*): A sequence database is a collection of sequences (Definition 2.2), denoted as $D = \{s_1, s_2, s_3, \dots, s_n\}$, where each $s_i (1 \leq i \leq n)$ is a tuple of values $\langle Dat, Sequence \rangle$: (1) Dat, which represents contextual information related to the time of the transaction; and (2) Sequence, which is an ordered list of itemset as described above.

For example, a sequential database could contain multiple sequences of customer purchases over time, with each sequence representing the items purchased in a particular transaction and the associated time of that transaction (in Table 2.4a).

Table 2.4. Customer transactions (a) and customer sequences (b).

Transaction time	Customer	Itemsets
01.02.2023 12: 00	11	BC
01.02.2023 13: 00	22	A
01.02.2023 14: 00	33	AC
01.02.2023 15: 00	44	C
03.02.2023 10: 00	33	AC
03.02.2023 16: 00	11	BC
03.02.2023 17: 00	44	A
04.02.2023 08: 00	66	A
04.02.2023 12: 00	44	D
05.02.2023 14: 00	55	A
05.02.2023 19: 00	33	C

(a)

DAT	Sequences
1	$\langle (BC)A(AC)C \rangle$
2	$\langle (AC)(BC)A \rangle$
3	$\langle AD \rangle$
4	$\langle AC \rangle$

(b)

For example, the sequential database as shown in Table 2.4b and $DAT = 1$, the provided sequence data $\langle (BC)A(AC)C \rangle$ can be examined to identify the order in which items were purchased. This sequence comprises four distinct itemset that are delimited by parentheses, with each itemset containing one or more items. It should be noted that if an itemset contains only one item, parentheses are not required. For example, item A in the sequence represents an itemset with a single item and does not require parentheses. The first itemset, (BC), corresponds to the purchase of item B followed by item C. The second

itemset, A, represents the acquisition of item A. The third itemset, (AC), indicates the purchase of item A followed by item C. Finally, the fourth and last item is C, signifying the purchase of item C for the three times.

Definition 2.15 (*Span value*): Assuming the sequences s_1 and s_2 have domain attributes d_1 and d_2, respectively, denoted as (DAT). The span from s_1 to s_2 is calculated as $[d_2\text{-}d_1]$, if d_1 is taken as the reference point. The sequence s_2 is an extended sequence and denoted as $s_2\ [d_2\text{-}d_1]$.

For example, considering the sequence database given in Table 2.4b, the 1st transaction is used as the reference point then the extended sequence of the 2nd transaction is $\langle(AC)(BC)A\rangle[1]$.

Definition 2.16 (*Extended sequence, item and itemset*): An extended sequence $s[d] = \langle t_1, t_2, t_3, \dots, t_n\rangle[d]$ consists of itemset t_i for $1 \leq i \leq n$, where $[d]$ denotes the span of s. The itemset t_i associated with $[d]$ is 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 u_k is an item for $1 \leq k \leq m$, we define u_k associated with $[d]$ as an extended item (e-item), denoted by $(u_k)[d]$.

For instance, the extended sequence $\langle(BC)A(AC)C\rangle[1]$ contains four e-itemset, $\langle(BC)\rangle[1]$, $\langle A\rangle[1]$, $\langle(AC)\rangle[1]$ and $\langle C\rangle[1]$ which can be decomposed into three e-items, $(A)[1]$, $(B)[1]$ and $(C)[1]$.

Definition 2.17 (*Megasequences*) : In a sequential database consisting of k sequences $\langle d_1,\ s_1\rangle, \langle d_2,\ s_2\rangle, \dots, \langle d_k,\ s_k\rangle$, a megasequence with $k > 0$ is represented by the union of its subsequences, denoted as $\Psi = s_1[0] \cup s_2[d_2 - d_1] \cup \dots \cup s_k[d_k - d_1]$. The reference point for Ψ is d_1, indicating that Ψ starts from this domain attribute. The $maxspan$ refers to a threshold set by the user which specifies the maximum span. To satisfy the condition of $maxspan$, the $|d_k - d_1|$ in the sequential database must be less than or equal to $maxspan$ ($d_k - d_1 \leq maxspan$).

An example of the megasequence list is shown in Table 2.5b, generated from the database presented in Table 2.4b, with $maxspan = 1$ and $DAT = 1$ serving as the reference point.

Table 2.5. Converting a sequential database of Table 2.4 (a) to megasequences (b).

DAT	Sequences
1	$\langle(BC)A(AC)C\rangle$
2	$\langle(AC)(BC)A\rangle$

DAT	Megasequences
1	$\langle(BC)A(AC)C\rangle[0]\langle(AC)(BC)A\rangle[1]$
2	$\langle(AC)(BC)A\rangle[0]\langle AD\rangle[1]$

3	$\langle AD \rangle$
4	$\langle AC \rangle$

(a)

3	$\langle AD \rangle[0]\langle AC \rangle[1]$
4	$\langle AC \rangle[0]$

(b)

Definition 2.18 (*E-item comparing*) (T. Le et al., 2018)**:** Consider two e-items $\alpha = (x)[d_1]$ and $\beta = (y)[d_2]$. The two e-items are equal, $\alpha = \beta$, if and only if they have the same span and the same content, $(d_1 = d_2) \wedge (x = y)$. The e-item α is less than β, $\alpha < \beta$, if either the span of α is less than the span of β, $d_1 < d_2$, or if the indices are equal, but the content of α is less than the content of β, $(d_1 = d_2) \wedge (x < y)$.

For instance, $(B)[0] = (B)[0]$, $(B)[2] < (B)[3]$ and $(B)[2] < (C)[2]$.

Definition 2.19 (*Subset e-item*) (T. Le et al., 2018): The function $sub_{k,l}(\alpha)$ is defined as the set of e-items of pattern α from position k to l, where the number of e-items is equal to $(l - k + 1)$.

For instance, $sub_{1,6}(\langle (AC)(BC)A \rangle[0]\langle AD \rangle[2]) = \langle (AC)(BC)A \rangle[0]\langle A \rangle[2]$ and $sub_{6,6}(\langle (AC)(BC)A \rangle[0]\langle AD \rangle[2]) = \langle A \rangle[2]$.

Definition 2.20 (*Inter-sequence 1-patterns joining*) (T. Le et al., 2018)**:** Given two frequent inter-sequence 1-patterns $x = \langle k \rangle[0]$ and $y = \langle m \rangle[0]$, three types of join extension can be performed on them. Firstly, an i-extension can be performed where $x \cup_{i-extension} y = \{\langle (km) \rangle[0]\}$. Secondly, a s-extension can be performed where $x \cup_{s-extension} y = \{\langle km \rangle[0]\}$. Lastly, a t-extension can be performed where $x \cup_{t-extension} y = \{\langle k \rangle[0]\langle m \rangle[d] | 1 \leq d \leq maxspan\}$. It should be noted that x and y are joinable in any instance.

For instance, given $maxspan = 2$:

- $\langle A \rangle[0] \cup_{i-extension} \langle B \rangle[0] = \langle (AB) \rangle[0]$
- $\langle A \rangle[0] \cup_{s-extension} \langle B \rangle[0] = \langle AB \rangle[0]$
- $\langle A \rangle[0] \cup_{t-extension} \langle B \rangle[0] = \{\langle A \rangle[0]\langle B \rangle[1], \langle A \rangle[0]\langle B \rangle[2]\}$

Definition 2.21 (*Inter-sequence k-patterns joining*) (T. Le et al., 2018)**:** Given two frequent inter-sequence k-patterns x and y, where $k > 1$, their subpatterns of length k, denoted as $sub_{k,k}(x) = (m)[d_1]$ and $sub_{k,k}(y) = (n)[d_2]$, respectively. If $sub_{1,k-1}(x) = sub_{1,k-1}(y)$ and $d_1 \leq d_2$, then x is joinable to y, resulting in three types of join extension:

- Itemset extension: $x \cup_{i-extension} y = \{x +_{i-extension} (n)[d_2] | (d_1 = d_2) \wedge (m < n)\}$.
- Sequence extension: $x \cup_{s-extension} y = \{x +_{s-extension} (n)[d_2] | d_1 = d_2\}$.

– Inter extension:$x \cup_{t-extension} y = \{x +_{t-extension}(n)[d_2] | d_1 < d_2\}$.

For instance:

– $\langle AB\rangle[0] \cup_{i-extension} \langle AD\rangle[0] = \langle A(BD)\rangle[0]$

– $\langle AB\rangle[0] \cup_{s-extension} \langle AD\rangle[0] = \langle ABD\rangle[0]$

– $\langle AB\rangle[0] \cup_{t-extension} \langle A\rangle[0]\langle D\rangle[2] = \langle AB\rangle[0]\langle D\rangle[2]$

Definition 2.22 (*Problem statement of inter-sequence pattern mining*): Given a sequential database D and a minimum support value, the task of inter-sequence pattern mining is to discover all frequent inter-sequence patterns.

Definition 2.23 (*Problem statement of inter-sequence pattern mining with itemset constraints*): Given a sequence database D, the minimum support ($minsupport$), 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 $\exists \alpha_i[w_i] \in \alpha, \exists b_j \in IC: b_j \subseteq \alpha_i$.

2.2.2 Algorithms for Mining Inter-sequence Patterns

2.2.2.1 EISP-Miner (C. S. Wang & Lee, 2009)

The EISP-Miner algorithm (C. S. Wang & Lee, 2009) proposed by Wang et al. mines inter-sequence pattern such that a pattern can be used to describe associations across many different sequences by the *maxspan* value (specified by the user). The algorithm uses a PatternList data structure to store information about a frequent pattern. The EISP-Miner algorithm goes through the following main steps:

- First, iterate through the original database and then use a PatternList to store frequent 1-pattern patterns.
- The second step stores all frequent 1-pattern patterns into a tree structure named ISP-Tree.
- Then, EISP-Miner algorithm uses depth-first search to find all frequent patterns. By using PatternList data structure and ISP-Tree tree structure, the EISP-Miner algorithm only needs to scan the original database once. This is the optimal point of EISP-Miner algorithm compared to previous algorithms such as Apriori.

The EISP-Miner algorithm is presented as following:

Algorithm 2.8. The EISP-Miner algorithm (C. S. Wang & Lee, 2009)

Input: A sequence database D, minimum support (*minsupport)*, and maximum span *maxspan*

Output: A complete of frequent inter-sequence patterns FP

1. Scan D to generate a set of all frequent 1-patternlists, T|NULL, as the extended group of the root node of an ISP-tree T;
2. For each frequent 1-patternlist αlist in T|NULL do
3. Call ISP-Join1(α_{list}, T|NULL, FP) to get T|α_{list};
4. Call ISP-JoinK(T|α_{list}, FP);
5. End for
6. Ouput FP;

Algorithm 2.9. The ISP-Join1 function (C. S. Wang & Lee, 2009)

Function: *ISP-Join1*(α_{list}, T|NULL, FP)

1. **for each** frequent 1-pattern γ in T|NULL, where α=<u>[0] and γ=<v>[0] **do**
2. **for** x = 0 to *maxspan* **do**
3. **if** (x=0) and (u<v) **then** $\theta_{list} = \alpha_{list} \cup_i \gamma_{list}$;
4. **if** (support(θ_{list}) $\geq$ *minsupport* **then** add θ_{list} to T|α_{list} and θ to FP;
5. **if** (x=0) **then** $\rho_{list} = \alpha_{list} \cup_s \gamma_{list}$;
6. **if** support(ρ_{list}) $\geq$ *minsupport* **then** add ρ_{list} to T|α_{list} and ρ to FP;
7. **If** (x>0) **then** $\sigma_{list} = \alpha_{list} \cup_t \gamma_{list}$;
8. **if** support(σ_{list}) $\geq$ *minsupport* **then** add σ_{list} to T|α_{list} and σ to FP;
9. **end for**
10. **end for**

Algorithm 2.10. The ISP-JoinK function (C. S. Wang & Lee, 2009)

Function: *ISP-JoinK*(T|α_{list}, FP)

1. **for each** frequent k-pattern β_{list} in T|α_{list} where $sub_{k,k}(\beta)$=<u>[i] **do**
2. **for each** frequent k-pattern γ_{list} in T|α_{list} where $sub_{k,k}(\gamma)$=<v>[j] **do**
3. **if** (i=j) and (u<v) **then** $\theta_{list} = \beta_{list} \cup_i \gamma_{list}$;
4. **if** support(θ_{list}) $\geq$ *minsupport* **then** add θ_{list} to T| β_{list} and θ tp FP;
5. **if** (i=j) then $\rho_{list} = \beta_{list} \cup_s \gamma_{list}$;
6. **if** support(ρ_{list}) $\geq$ *minsupport* **then** add ρ_{list} to T| β_{list} and ρ tp FP;
7. **If** (i<j) then $\sigma_{list} = \beta_{list} \cup_t \gamma_{list}$;
8. **if** support(σ_{list}) $\geq$ *minsupport* **then** add σ_{list} to T| β_{list} and σ tp FP;
9. **end for**
10. *Call ISP-JoinK*(T|β_{list}, FP);
11. **end for**

The EISP-Miner algorithm offers several advantages in terms of efficiency and storage utilization:

- By performing only one scan of the database, EISP-Miner is able to calculate candidate support and candidate extension simultaneously. This eliminates the need for costly matching of candidate subsets, resulting in significant time and storage space savings. The algorithm leverages the PatternList data structure, which stores only the location information of candidates. This approach avoids the necessity of rescanning the original database during candidate expansion.
- However, it is important to note that the PatternList data structure has some limitations. It utilizes integers to store candidate location information, which can lead to increased memory requirements, particularly when working with large databases. Additionally, calculating the support level of candidates may be more time-consuming compared to other methods.

2.2.2.2 **DBV-ISP** (Vo et al., 2012)

Mining inter-sequence patterns was proposed by Wang et al. via the EISP-Miner algorithm (C. S. Wang & Lee, 2009), but this algorithm must use a set of integers to store the position of a pattern in the database. This leads to a lot of memory usage during the running of the algorithm. To overcome the above inefficient, Vo et al. used an alternative structure called DBV-PatternList and proposed an algorithm named DBV-ISP (Vo et al., 2012). The algorithm uses a DBV-PatternList data structure to store information about a frequent pattern. Because DBV-ISP algorithm is based on the EISP-Miner algorithm, it also has the following main steps: First, iterate through the original database and then use a DBV-PatternList to store frequent 1-pattern patterns. The second step stores all frequent 1-pattern pattern into a tree structure named DBV-tree. Then, the DBV-ISP algorithm uses depth-first search to find all frequent patterns. By using DBV-PatternList data structure and IDBV-tree structure, the DBV-ISP algorithm only needs to scan the original database once and is more optimized than the previous algorithm EISSP-Miner.

The evaluation of the DBV-ISP algorithm provides insights into its running time and storage space. The algorithm demonstrates several strengths, outlined below:

- The DBV-ISP algorithm represents an improved and extended version of the EISP-Miner algorithm (C. S. Wang & Lee, 2009). It introduces the utilization of a bit data structure for storing candidate location information, leading to reduced storage space, and processing time compared to the EISP-Miner algorithm. By employing bit intersection operations for candidate support

calculation, the DBV-ISP algorithm achieves faster execution in comparison to working with integers.

- Furthermore, the algorithm leverages a dynamic bit vector data structure to optimize the storage space for candidate locations. Specifically, for candidate patterns that occur infrequently in the database, only the first occurrence and subsequent occurrence positions need to be stored. This approach effectively reduces the storage of zero bits in candidate location information, resulting in more efficient memory utilization. These strengths make the DBV-ISP algorithm an efficient and effective solution for inter-sequence mining tasks.
- However, the dynamic bit vector data structure has limitations when it comes to storing information. Newly created candidate location information must be stored, resulting in overlapping positions with previous frequent patterns. This duplication of data poses a challenge and requires additional storage memory. To address this issue, in Chapter 3, we propose the implementation of a data structure called pseudo-IDList as a solution.

2.2.2.3 **ISP-IC, *i*ISP-IC, *pi*ISP-IC** (T. Le et al., 2018)

Mining inter-sequence pattern was proposed by Wang et al. (C. S. Wang & Lee, 2009), and later improved by Vo et al. (Vo et al., 2012). But the algorithm generates a lot of frequent patterns during mining process. To improve this problem, Le et al. proposed an algorithm called ISP-IC (**I**nter-**S**equence **P**attern with **I**tem **C**onstraint mining) (T. Le et al., 2018), which uses item constraints in the process of inter-sequence mining. Based on Lemma 2.1, Le et al. presented an improved algorithm of the ISP-IC algorithm, named *i*ISP-IC. The *i*ISP-IC algorithm reduces the number of conditional checks on item constraints for newly generated patterns, helping to optimize the running time of the algorithm. Finally, Le et al. presented a parallel version of *i*ISP-IC named *pi*ISP-IC to improve the performance.

Lemma 2.1: Let α satisfy constraint χ then $\forall\beta$, following sequences $\gamma_I = \alpha \cup_I \beta$, $\gamma_S = \alpha \cup_S \beta$, and $\gamma_T = \alpha \cup_T \beta$ also satisfy constraint χ.

The advantages and disadvantages of the ISP-IC algorithm are evaluated as follows:

- The ISP-IC algorithm extends the functionality of the DBV-ISP algorithm by addressing the challenge of mining inter-sequence patterns with item condition constraints. It inherits the dynamic bit vector data structure from DBV-ISP, leading to optimized processing time and storage space utilization. The use of

the bit structure enables efficient calculation of candidate support through bit assignment operations.

- However, the algorithm does not consider other constraints, such as itemset conditions. In Chapter 4, the problem of mining inter-sequence patterns with itemset condition constraints is introduced separately.
- Additionally, the dynamic bit vector structure does not yet resolve data redundancy. The information regarding the location of newly generated candidates still overlaps with previous frequent patterns.

2.3 Summary

In this chapter, we introduced the fundamental concepts, definitions, and examples for frequent pattern mining problems, including mining sequence patterns, sequence pattern mining with constraints, and clickstream pattern mining. We then presented the basic concepts, definitions, and examples for the inter-sequence pattern mining problem. Based on this background information, we made improvements to the inter-sequence pattern mining problem, which are presented in Chapter 3. Furthermore, we proposed problems for itemset-constrained inter-sequence pattern mining, which are presented in Chapter 4.

CHAPTER 3: MINING INTER-SEQUENCE PATTERNS BASED ON PSEUDO-IDLIST AND EARLY PRUNING TECHNIQUES

In this chapter, we introduce the problem of mining inter-sequence patterns and subsequently address the limitation of the currently employed data structure in the presence of duplicated data during the mining process. To overcome this limitation, we propose the utilization of a data structure known as pseudo-IDList. Furthermore, we present an algorithm named ISP-PI (Inter-Sequence Pattern mining based on Pseudo-Index) specifically designed for the inter-sequence pattern mining problem. The algorithm incorporates the ISP-IC (Inter-Sequence Pattern mining with Index Intersection Checking) method to optimize the mining time. To assess the effectiveness of the proposed algorithm in comparison to previous algorithms employed in the field of inter-sequence mining, we employ six test databases to evaluate the algorithm's performance in terms of running time and storage space utilization.

3.1 Introduction

In 2009, Wang and Lee (C. S. Wang & Lee, 2009) introduced a novel approach for inter-sequence pattern mining based on a vertical database format. The authors suggested utilizing the ISP-Tree structure to generate potential candidates that meet the minimum support threshold. This approach allows for mining patterns across transactions in the sequence database, while still being able to exploit traditional sequences like GSP (Srikant & Agrawal, 1996), SPAM(Ayres et al., 2002), SPADE (Zaki, 2001), CM-SPADE (Fournier-Viger et al., 2014), and PRISM (Gouda et al., 2007, 2010). The method stores sequence identifiers to calculate the support of patterns, as shown in Figure 3.2. However, this approach requires a significant amount of memory to store sequence identifiers and time to compute the intersection of sequence identifiers. To overcome these limitations, Vo et al. (Vo et al., 2012) introduced an efficient data structure called the DBV-PatternList, which replaces the PatternList structure used by the EISP-Miner approach. The DBV-PatternList data structure is illustrated in Figure 3.3 and Figure **3.6**. This approach significantly reduces the storage space and time required for MISP, as well as for mining closed inter-sequence patterns (B. Le et al., 2015). More recently, Nguyen et al. (2023) (Nguyen et al., 2023) proposed an algorithm which uses a DBV-PatternList based structure for MISP with itemset constraints, named DBV-ISPMIC. The proposed algorithm utilizes the DBV-PatternList to expediently compute the support of patterns. Moreover, they developed an improved algorithm based on a property to reduce checking candidates.

Additionally, a parallel method called *p*DBV-ISPMIC was also developed. Empirical evaluations showed that DBV-ISPMIC outperformed previous algorithms, and *p*DBV-ISPMIC outperformed DBV-ISPMIC in terms of runtime.

For instance, using the customer database given in Table 2.4, with $minsupport = 50\%$ and $maxspan = 1$. The complete set of frequent inter-sequence patterns with their supports are shown in Table 3.1 (based on Definition 2.22).

Table 3.1. Frequent inter-sequence patterns mined from the database shown in Table 2.4.

Level 1	Level 2	Level 3	Level 4
$\langle A\rangle[0]: 4$	$\langle CC\rangle[0]: 2$	$\langle CA\rangle[0]\langle A\rangle[1]$:2	$\langle (BC)A\rangle[0]\langle A\rangle[1]$:2
$\langle B\rangle[0]: 2$	$\langle (BC)\rangle[0]: 2$	$\langle CC\rangle[0]\langle A\rangle[1]$:2	$\langle (AC)C\rangle[0]\langle A\rangle[1]$:2
$\langle C\rangle[0]: 3$	$\langle (AC)\rangle[0]: 2$	$\langle BA\rangle[0]\langle A\rangle[1]$:2	
	$\langle AC\rangle[0]: 3$	$\langle (BC)\rangle[0]\langle A\rangle[1]$:2	
	$\langle CA\rangle[0]: 2$	$\langle (BC)A\rangle[0]$:2	
	$\langle C\rangle[0]\langle A\rangle[1]: 2$	$\langle AA\rangle[0]\langle A\rangle[1]$:2	
	$\langle A\rangle[0]\langle C\rangle[1]: 2$	$\langle AC\rangle[0]\langle A\rangle[1]$:2	
	$\langle BA\rangle[0]: 2$	$\langle (AC)\rangle[0]\langle A\rangle[1]$:2	
	$\langle B\rangle[0]\langle A\rangle[1]: 2$	$\langle (AC)C\rangle[0]$:2	
	$\langle AA\rangle[0]: 2$		
	$\langle A\rangle[0]\langle A\rangle[1]: 3$		

Previous research on mining inter-sequence patterns is still limited due to the duplication of data and the large number of candidates generated, which requires significant processing time and storage space. The contributions of this chapter are presented as follows:

1. Demonstrating the limitations of the PatternList and dynamic bit vector data structures in terms of data duplication in previous inter-sequence pattern mining algorithms.
2. Introducing the pseudo-IDList data structure for pattern extension by sequence and itemset and extending its use for the inter-sequence mining problem with inter-extension. The effectiveness of this data structure for avoiding data duplication in mining inter-sequence patterns is proven, and the ISP-PI algorithm is proposed for this problem.
3. Proposing and applying the ISP-IC pruning method to the ISP-PI algorithm to reduce the number of generated patterns, given the large number of candidates generated in

the inter-sequence mining problem. The pruning method helps reduce the time required for support calculations by reducing the number of candidates.

4. Evaluating the performance of the ISP-PI algorithm using the pseudo-IDList data structure and the applied pruning method. Six test databases were used for evaluation, including large databases with nearly a million rows of data.

$\langle A \rangle$	
DAT	*Position List*
1	2, 3
2	1, 3
3	1
4	1

$\langle B \rangle$	
DAT	*Position List*
1	1
2	2

$\langle C \rangle$	
DAT	*Position List*
1	1, 3, 4
2	1, 2
4	2

$\langle D \rangle$	
DAT	*Position List*
3	2

Figure 3.1. The value of DAT and the position of each item in a transaction are extracted from Table 2.4.

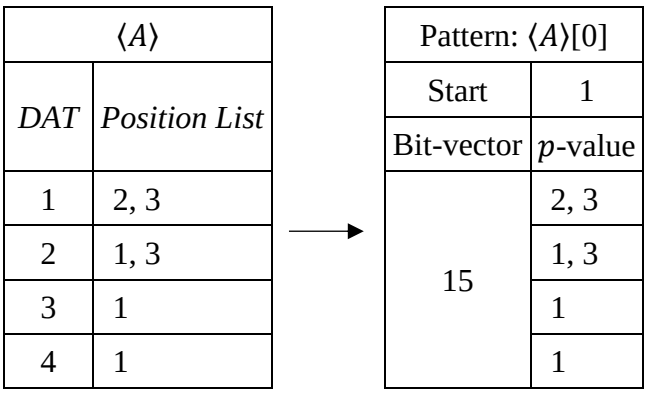
$\langle A \rangle$	
DAT	*Position List*
1	2, 3
2	1, 3
3	1
4	1

→

Pattern: $\langle A \rangle$[0]	
t-value	p-value
1	2, 3
2	1, 3
3	1
4	1

Figure 3.2. A PatternList data structure for pattern ⟨A⟩ from Table 2.4.

Figure 3.2 presents a PatternList data structure for pattern $\langle A \rangle$ from Table 2.4, which involves creating two columns. The first column represents the t-value and corresponds to the DAT column in Table 2.4. The second column represents the p-value and corresponds to the Position List column illustrated in Figure 3.1.

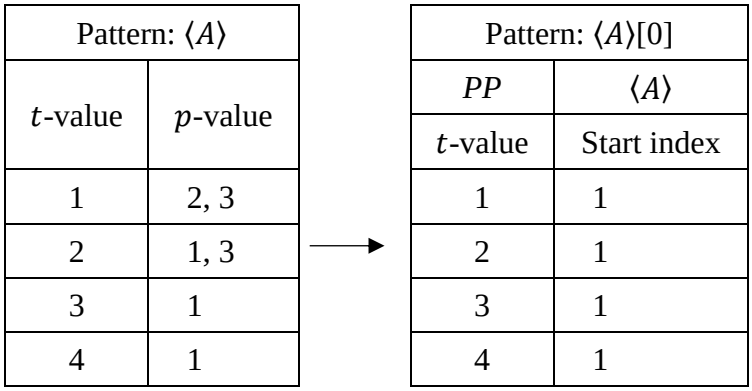


⟨A⟩	
DAT	*Position List*
1	2, 3
2	1, 3
3	1
4	1

Pattern: ⟨A⟩[0]	
Start	1
Bit-vector	p-value
15	2, 3
	1, 3
	1
	1

Figure 3.3. DBV-PatternList data structure for pattern ⟨A⟩ is constructed from Table 2.4.

Figure 3.3 presents the DBV-PatternList data structure for pattern $\langle A \rangle$ is constructed from Table 2.4. As the pattern appears in the first transaction with a DAT value of 1, the Start value is initialized to 1. The pattern $\langle A \rangle$ appears in transactions with DAT values of 1, 2, 3, and 4, and its bit value is either 1 (if it appears) or 0 (if it does not appear). Consequently, the list of bits for pattern $\langle A \rangle$ is (1111) (in binary), and the corresponding Bit-vector value is 15 (in decimal) for the first column of $\langle A \rangle$ pattern. The second column, p-value, corresponds to the PositionList column.



Pattern: $\langle A \rangle$	
t-value	p-value
1	2, 3
2	1, 3
3	1
4	1

Pattern: $\langle A \rangle$[0]	
PP	$\langle A \rangle$
t-value	Start index
1	1
2	1
3	1
4	1

Figure 3.4. A pseudo-IDList data structure is constructed for the ⟨A⟩ pattern based on its PatternList data structure.

Figure 3.4 shows a pseudo-IDList data structure constructed for the $\langle A \rangle$ pattern based on its PatternList data structure. The PP value, which is the last item in the pattern, is set to $\langle A \rangle$. The t-value column of the pseudo-IDList data structure corresponds with the t-value column of the PatternList data structure, and the start index value is initialized to 1. This is because the $\langle A \rangle$ pattern has the same p-value column value as that of the PP pattern.

$\langle A \rangle$[0]	
t-value	p-value
1	2, 3

$\langle B \rangle$[0]	
t-value	p-value
1	1

$\langle C \rangle$[0]	
t-value	p-value
1	1, 3, 4

2	1, 3
3	1
4	1

2	2

2	1, 2
4	2

Figure 3.5. The list of PatternList of frequent inter-sequence 1-patterns generated from Table 2.4.

$\langle A \rangle[0]$	
Start	1
Bit-vector	p-value
15	2
	1, 3
	1
	1

$\langle B \rangle[0]$	
Start	1
Bit-vector	p-value
12	1
	2

$\langle C \rangle[0]$	
Start	1
Bit-vector	p-value
13	1, 3, 4
	1, 2
	2

Figure 3.6. The list of dynamic bit vector of frequent inter-sequence 1-patterns generated from Table 2.4.

$\langle A \rangle[0]$	
t-value	p-value
1	2, 3
2	1, 3
3	1
4	1

$\langle C \rangle[0]$	
t-value	p-value
1	1, <u>3, 4</u>
2	1, <u>2</u>
4	<u>2</u>

→

$\langle AC \rangle[0]$	
t-value	p-value
1	<u>3, 4</u>
2	<u>2</u>
4	<u>2</u>

(a)

$\langle A \rangle[0]$	
t-value	p-value
1	2, 3
2	1, 3
3	1
4	1

$\langle C \rangle[0]$	
t-value	p-value
1	1, 3, 4
2	<u>1, 2</u>
4	<u>2</u>

→

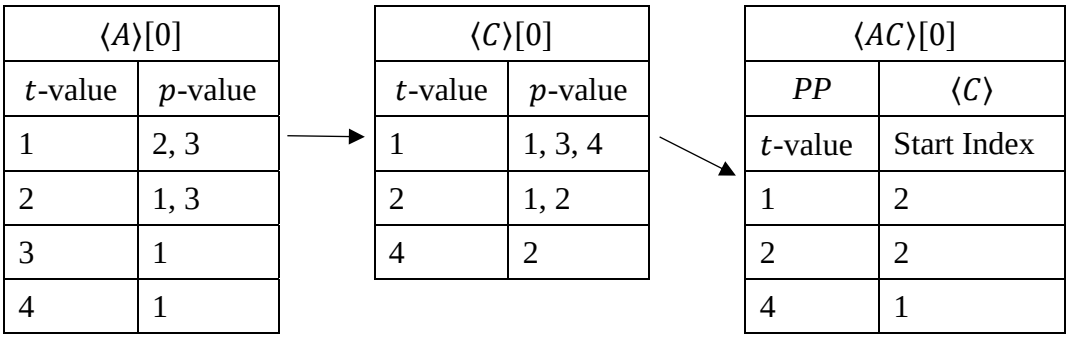
$\langle A \rangle[0]\langle C \rangle[1]$	
t-value	p-value
2	<u>1, 2</u>
4	<u>2</u>

(b)

Figure 3.7. s-extension (a) and t-extension (b) of the ⟨A⟩[0] and ⟨C⟩[0] patterns.

Figure 3.7 shows the s-extension (a) and t-extension (b) of the $\langle A \rangle[0]$ and $\langle C \rangle[0]$ patterns resulted in the $\langle AC \rangle[0]$ and $\langle A \rangle[0]\langle C \rangle[1]$ patterns, respectively. When comparing

the p-value column of these patterns with that of the $\langle C \rangle[0]$ pattern, it was found that duplicate values can occupy a considerable amount of memory storage.



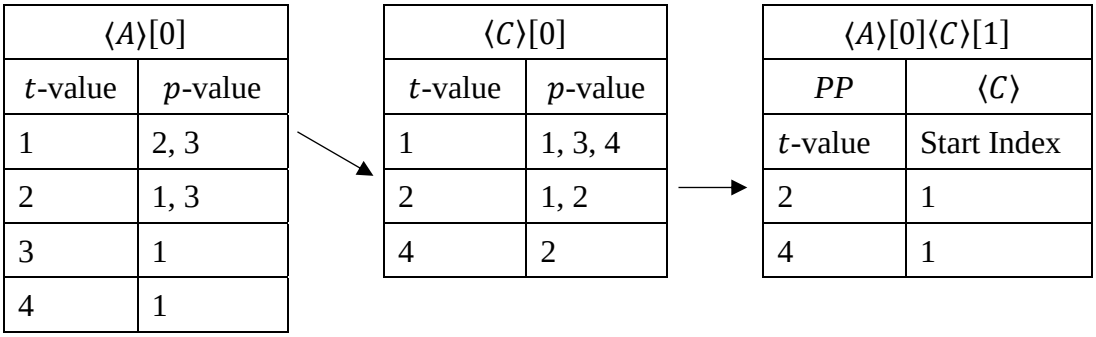
$\langle A \rangle[0]$	
t-value	p-value
1	2, 3
2	1, 3
3	1
4	1

$\langle C \rangle[0]$	
t-value	p-value
1	1, 3, 4
2	1, 2
4	2

$\langle AC \rangle[0]$	
PP	$\langle C \rangle$
t-value	Start Index
1	2
2	2
4	1

Figure 3.8. A pseudo-IDList structure is constructed for the pattern ⟨AC⟩[0].

A pseudo-IDList structure constructed for the pattern $\langle AC \rangle[0]$ by using two frequent inter-sequence 1-patterns, namely the PatternList $\langle A \rangle[0]$ and $\langle C \rangle[0]$, within an s-extension is shown in Figure 3.8. The PP variable is set to $\langle C \rangle$, which is the last item in the form $\langle AC \rangle[0]$. The first line of the pattern $\langle A \rangle[0]$ is linked to the first line of the pattern $\langle C \rangle[0]$ since they both have a t-value of 1. Following Definition 2.20, we obtain the set of p-values, which is $\{3, 4\}$ because the condition is p-value($\langle A \rangle[0]$) $<$ p-value($\langle C \rangle[0]$). Using the PP variable and t-value of 1, we identify a set of p-values $\{3, 4\}$ that occur at the second position, resulting in a StartIndex value of 2. Similarly, for the t-values that correspond to $\langle A \rangle[0]$ and $\langle C \rangle[0]$, we find the {t-value, StartIndex} pairs $\{2, 2\}$ and $\{4, 1\}$.



$\langle A \rangle[0]$	
t-value	p-value
1	2, 3
2	1, 3
3	1
4	1

$\langle C \rangle[0]$	
t-value	p-value
1	1, 3, 4
2	1, 2
4	2

$\langle A \rangle[0]\langle C \rangle[1]$	
PP	$\langle C \rangle$
t-value	Start Index
2	1
4	1

Figure 3.9. A pseudo-IDList structure is constructed for the pattern ⟨A⟩[0]⟨C⟩[1].

Figure 3.9 shows the pseudo-IDList structure constructed for the pattern $\langle A \rangle[0]\langle C \rangle[1]$ using two frequent inter-sequence 1-patterns, namely the PatternList $\langle A \rangle[0]$ and $\langle C \rangle[0]$, within a t-extension and $maxspan = 1$. The variable PP was set to $\langle C \rangle$, which is the final item in the $\langle A \rangle[0]\langle C \rangle[1]$ pattern. The first row of the pattern $\langle A \rangle[0]$ was linked to the second line of the pattern $\langle C \rangle[0]$, since t-value($\langle C \rangle[0]$) - t-value($\langle A \rangle[0]$) = $maxspan$. Following Definition 2.20, we obtained the set of p-values, which was {1,2}. Using the PP variable and a t-value of 2, we identified a set of p-values $\{1, 2\}$ that

occurred at the first position, resulting in a StartIndex value of 1. Similarly, for the t-values corresponding to $\langle A\rangle[0]$ and $\langle C\rangle[0]$, we found the {t-value, StartIndex} pairs $\{4, 1\}$.

$\langle(BC)\rangle[0]$	
t-value	p-value
1	1
2	2

$\langle BA\rangle[0]$	
t-value	p-value
1	2,3
2	3

$\langle(BC)A\rangle[0]$	
PP	$\langle A\rangle$
t-value	Start Index
1	1
2	1

Figure 3.10. A pseudo-IDList structure is constructed for the pattern $\langle(BC)A\rangle[0]$ by using two frequent inter-sequence k-patterns ($k > 1$), namely the PatternList $\langle(BC)\rangle[0]$ and $\langle BA\rangle[0]$, within a s-extension.

$\langle(AC)\rangle[0]$	
t-value	p-value
1	3
2	1

$\langle A\rangle[0]\langle A\rangle[1]$	
t-value	p-value
2	1,3
3	1
4	1

$\langle(AC)\rangle[0]\langle A\rangle[1]$	
PP	$\langle A\rangle$
t-value	Start Index
2	1
3	1

Figure 3.11. A pseudo-IDList structure is constructed for the pattern $\langle(AC)\rangle[0]\langle A\rangle[1]$ using two frequent inter-sequence k-patterns ($k > 1$), namely the PatternList $\langle(AC)\rangle[0]$ and $\langle A\rangle[0]\langle A\rangle[1]$, within an t-extension and $maxspan = 1$.

$\langle AC\rangle[0]$	
PP	$\langle C\rangle$
t-value	p-value
1	2
2	2
4	1

$\langle C\rangle[0]$	
DAT	*Position List*
1	1, 3, 4
2	1, 2
4	2

$\langle AC\rangle[0]$	
DAT	*Position List*
1	3,4
2	2
4	2

Figure 3.12. The process of data retrieval of a pseudo-IDList for pattern ⟨AC⟩[0].

Figure 3.12 shows the process of data retrieval of a pseudo-IDList for pattern $\langle AC\rangle[0]$. The first row in the pseudo-IDList of $\langle AC\rangle[0]$ indicates the first row in the PatternList of the $\langle C\rangle[0]$ pattern because the t-value and the DAT are both 1. The p-value is 2, which means that it will retrieve data starting from the second position to the end of the PositionList of $\langle C\rangle[0]$ pattern, and the result is {3,4}. Similarly, for the second and

third rows of the pseudo-IDList $\langle AC\rangle[0]$ pattern, the resulting PositionList for the second row is $\{2\}$ and for the third row is $\{3\}$ in PatternList $\langle AC\rangle[0]$.

3.2 Data Structure

3.2.1 PatternList

Definition 3.1 (C. S. Wang & Lee, 2009): Given a pattern a, we define a patternlist $a_{list} = a\{t_1.p_{11}p_{12}\dots p_{1m1}, t_1.p_{21}p_{22}\dots p_{2m2}, t_n.p_{n1}p_{n2}\dots p_{nmn}\}$, where $\{t_1.p_{11}\ p_{12}\dots p_{1m1}, t_2.p_{21}\ p_{22}\dots p_{2m2},\ t_n.p_{n1}\ p_{n2}\dots p_{nmn}\}$ is called the list; t_i is the dat (t-value); and p_{ij} is the position (p-value) that α's last e-item appears at in the database $1 \le i \le n$ and$1 \le j \le mi$. We also define $support(a_{list})$ as the number of $t\text{-}values$ contained in a_{list}. If α is a *k*-pattern and $support\ (a_{list}) \ge minsup$, we say that a_{list} is a frequent *k*-patternlist.

Based on Definition 3.1, Figure 3.1 depicts the original information construction of patterns from Table 2.4. Each pattern has DAT (column DAT) and position (column PositionList) information pertaining to the corresponding pattern as it appears in the sequence database.

Taking the $\langle A\rangle$ pattern as an example, it comprises four DATs $(1, 2, 3, 4)$ and four sets of positions $(\{2, 3\}, \{1, 3\}, \{1\}, \{1\})$, where position set $\{2, 3\}$ is linked with DAT 1, position set $\{1, 3\}$ is linked with DAT 2, position set $\{1\}$ is linked with DAT 3, and position set $\{1\}$ is linked with DAT 4. The numbers in the $\langle A\rangle$ pattern indicate that the pattern appears at 1st, 2nd, 3rd, 4th in the sequence database. Specifically, it appears in the 2nd and 3rd itemset of the sequence of DAT 1, in the 1st and 3rd itemset of the sequence of DAT 2, in the 1st itemset of the sequence of DAT 3, and in the 1st itemset of the sequence of DAT 4.

Figure 3.2 demonstrates the construction of the $\langle A\rangle$ PatternList based on the $\langle A\rangle$ pattern. A PatternList (C. S. Wang & Lee, 2009) is defined as a collection of three elements: $Pattern$, $Position\ Matrix$, and $Support$. $Pattern$ refers to the pattern itself, while $Position\ Matrix$ is a collection of 2-tuples $\{t$-value, p-value$\}$. The t-value is the DAT value, and the p-value keeps the set of positions of the pattern the transaction sequence. Finally, $Support$ refers to the support of the pattern, that is simply $Position\ Matrix$ size.

According to the above definition, the set of frequent inter-sequence PatternList shown in Figure 3.5 can be expressed as follows: $\langle A\rangle[0]\{1.23, 2.13, 3.1, 4.1\}$, $\langle B\rangle[0]\{1.1, 2.2\}$, and $\langle C\rangle[0]\{1.134, 2.12, 4.2\}$. The support values for each of these frequent inter-sequence 1-PatternLists are $Support(\langle A\rangle[0]) = 4$, $Support(\langle B\rangle[0]) = 2$, and $Support(\langle C\rangle[0]) = 3$, respectively.

3.2.2 Pseudo-IDList

In the problem of IPM, data duplication can occur when new candidates are generated through extension types such as s-extension and t-extension, as shown in Figure 3.7. This duplication arises when the position value of items in a transaction must be maintained, leading to increased memory usage during algorithm execution. To address this issue, we have utilized and expanded the pseudo-IDList data structure proposed by Huynh et al. (Huynh et al., 2022). Figure 3.4 depicts how to construct a pseudo-IDList for a PatternList. The pseudo-IDList contains the following information: $Pattern$, which is the pattern; PP, a PatternList pointer that points to the PatternList of frequent inter-sequence 1-pattern, which is the last item in $Pattern$; $Position\ Maxtrix$, a collection of 2-tuples {t-value, p-value} that are indices, with the t-value corresponding to a t-value in the PatternList and the p-value indicating the starting position of pattern $Pattern$ in PatternList PP that matches each transaction line (t-value); and $Support$, the $Pattern$'s support, which we can compute by the $Position\ Maxtrix$ size.

3.3 Algorithms

3.3.1 Candidate Generation

The EISP-Miner algorithm (C. S. Wang & Lee, 2009) generates candidates from two frequent inter-sequence patterns based on two cases: frequent inter-sequence 1-patterns (as defined in Figure 3.4) and frequent inter-sequence k-patterns $(k > 1)$ (as defined in Figure 3.5). In this paper, we use the candidate generation method of the SPADE algorithm (Zaki, 2001) that incorporates the t-extension. Given the two frequent inter-sequence patterns α and β, the generated candidates are as follows:

- If both patterns α and β have only one megasequence, the algorithm expands them by two cases of sequence and itemset. The resulting candidates are as follows:
  - If both patterns have s-extensions, three candidates are generated that are a mix of sequence and itemset extensions. However, if both patterns are the same pattern $(\alpha = \beta)$, then only one sequence extension candidate is produced.
  - If both patterns have i-extensions, only one itemset extension candidate is generated.
  - If α has an i-extension and β has an s-extension, only one sequence extension candidate is generated.
- If the two frequent inter-sequence patterns α and β are expanded according to a t-extension, there are two cases when the condition is extended to 1-patterns (as defined in Figure 3.4) or expands to k-patterns $(k > 1)$ (as defined in Figure 3.5):

+ If both frequent inter-sequence patterns satisfy the expansion condition for 1-patterns, the set of candidates is output as $\{\langle\alpha\rangle[0]\langle\beta\rangle[x]|0 < x \leq maxspan\}$.
+ In contrast, only one candidate is generated according to the k-patterns condition $(k > 1)$.

As an example, consider two frequent inter-sequence patterns: $\alpha = \langle A\rangle[0]$ and $\beta = \langle B\rangle[0]$. The candidate generation rules for sequence, itemset, and inter-extensions generate the following set of candidates: $\{\langle AB\rangle[0], \langle BA\rangle[0], \langle(AB)\rangle[0], \langle A\rangle[0]\langle B\rangle[1]\}$.

Another example: let $\alpha = \langle C\rangle[0]\langle A\rangle[1]$ and $\beta = \langle CA\rangle[0]$ be two frequent inter-sequence patterns. In this case, there is only one candidate generated for inter-extensions, which is $\langle CA\rangle[0]\langle A\rangle[1]$.

3.3.2 ISP-IC Method

Let $S_1 = \langle X, Last_{S1}\rangle\{t_1, t_2, t_3, \dots, t_n\}$ and $S_2 = \langle X, Last_{S2}\rangle\{t_1, t_2, t_3, \dots, t_m\}$ be two frequent inter-sequence patterns, where X is a prefix of S_1 and S_2, and $Last_{S1}$, $Last_{S2}$ are the last items of S_1 and S_2, respectively and t_i $(1 \leq i \leq n)$, t_j $(1 \leq j \leq m)$ is the position of $S1$, $S2$ on the sequential database, respectively.

Lemma 3.1: Then $S_1 \cup_{s-extension} S_2 = \{\langle XX_nX_m\rangle, \langle XX_mX_n\rangle\}$, if $\langle XX_nX_m\rangle$ is not frequent, then $\langle XX_mX_n\rangle$ is also not frequent, or vice versa.

Proof: Based on Definition 2.20 and Definition 2.21, the number of transactions containing $\langle XX_nX_m\rangle$ is $t_{S1} = |\{t_1, t_2, t_3, \dots, t_n\} \cup \{t_1, t_2, t_3, \dots, t_m\}|$, and the number of transactions containing $\langle XX_mX_n\rangle$ is also $t_{S2} = |\{t_1, t_2, t_3, \dots, t_n\} \cup \{t_1, t_2, t_3, \dots, t_m\}|$. Therefore, $t_{S1} = t_{S2}$.

If $\langle XX_nX_m\rangle$ is not frequent, then $t_{S1} < minsupport$. Hence, $t_{S2} < minsupport$, which implies that $\langle XX_mX_n\rangle$ is not frequent. Conversely, if $\langle XX_mX_n\rangle$ is not frequent, then $t_{S2} < minsup$, which implies that $t_{S1} < minsupport$ and $\langle XX_nX_m\rangle$ is also not frequent. Thus, the lemma is proven.

For example, let $S_1 = \langle ABC\rangle\{1, 3, 5, 7, 8\}$ and $S_2 = \langle ABD\rangle\{1, 5, 6, 9\}$ be two frequent inter-sequence patterns with $support(\langle ABC\rangle) = 5$, $support(\langle ABD\rangle) = 4$, and $minsup = 3$. Pattern $\langle ABC\rangle$ appears at positions 1st, 3rd, 5th, 7th, and 8th in the sequential database, while pattern $\langle ABD\rangle$ appears at positions 1st, 5th, 6th, and 9th. By applying the s-extension, we have $S_1 \cup_{s-extension} S_2 = \{\langle ABCD\rangle, \langle ABDC\rangle\}$. The positions where pattern $\langle ABCD\rangle$ appears are $\{1, 3, 5, 7, 8\} \cup \{1, 5, 6, 9\} = \{1, 5\}$. Since pattern $\langle ABCD\rangle$ appears only twice in the sequential database and $minsup = 3$, it is not frequent. Therefore, pattern $\langle ABDC\rangle$ is also not frequent, and there is no need to calculate its support.

Lemma 3.2: The candidates generated from S_1 and S_2 by sequence and itemset extension are:

- $S_1 \cup_{s-extension} S_2 = \{\langle XX_nX_m\rangle, \langle XX_mX_n\rangle\}$
- $S_1 \cup_{i-extension} S_2 = \{\langle X(X_nX_m)\rangle | X_n < X_m\}$

Proof: Based on Definition 2.20 and Definition 2.21, the position of each pattern in the set of generated candidates can be calculated by $t_S = |\{t_1, t_2, t_3, \dots, t_n\} \cup \{t_1, t_2, t_3, \dots, t_m\}|$. We have $support(\langle XX_nX_m\rangle) \leq t_S$, $support(\langle XX_mX_n\rangle) \leq t_S$, $support(\langle X(X_nX_m)\rangle) \leq t_S$, if $t_S < minsupport$ then $support(\langle XX_nX_m\rangle) < minsupport$, $support(\langle XX_mX_n\rangle) < minsupport$, $support(\langle X(X_nX_m)\rangle) < minsupport$ or all of the generated candidates are not frequent.

Instead of generating all possible candidates and calculating the support of each one, Lemma 3.2 suggests a more efficient approach that eliminates candidates whose occurrence is less than the support value. This can be achieved by generating candidates through two expansions, namely sequence and itemset expansions.

For example, given two frequent inter-sequence patterns $S_1 = \langle ABC\rangle\{1, 3, 5, 7, 8\}$ and $S_2 = \langle ABD\rangle\{1, 5, 6\}$ with $minsup = 3$. In the sequential database, pattern $\langle ABC\rangle$ occurs at the 1st, 3rd, 5th, 7th, and 8th positions while pattern $\langle ABD\rangle$ occurs at the 1st, 5th, and 6th positions. The position of the patterns is shown in Figure 3.13, where a bit value of 1 indicates that the pattern is present, and a bit value of 0 represents a pattern that does not occur. To calculate the number of occurrences of a candidate in the database, we can count the number of 1s using the bit AND operation. If the calculated value is less than the support value, based on Lemma 3.2, we can discard candidates without generating them, which can save time and resources. This process is possible by applying two expansions in sequence and itemset.

Bit-index	1	2	3	4	5	6	7	8
$\langle ABC\rangle$	1	0	1	0	1	0	1	0
$\langle ABD\rangle$	1	0	0	0	1	1	0	0
$\langle ABCD\rangle$, $\langle ABDC\rangle$, $\langle AB(CD)\rangle$	1	0	0	0	1	0	0	0

Figure 3.13. Example of using a bit string to calculate the number of occurrences of a candidate inter-sequence pattern in a sequential database, based on Lemma 3.2.

3.3.3 ISP-PI Algorithm

In this section, we present an extended version of the EISP-Miner (C. S. Wang & Lee, 2009) algorithm that utilizes the pseudo-IDList data structure (Section 3.3.2) to store

frequent inter-sequence patterns generated from sequence and inter-extensions and the PatternList data structure (Section 3.3.1) to store frequent inter-sequence patterns generated from the itemset extension. The algorithm comprises the following steps:

Step 1: Scan the database to find all frequent inter-sequence 1-patterns and use the PatternList data structure to store information about each pattern in the set. Figure 3.2 illustrates how to create the PatternList's information from a frequent inter-sequence pattern.

Step 2: Utilizing the methodology described in Section 3.3.1, combine one of the frequent inter-sequence patterns discovered in Step 1 (i.e., $\langle A \rangle$ pattern) with the remaining frequent inter-sequence patterns in the set to generate candidates based on the sequence, itemset, and inter-extension. Applying Lemma 3.1 to quickly eliminate the candidates generated. Store the information of candidates created under these conditions as follows:

- If the candidate is generated in an extended itemset type, its information is stored in the PatternList.
- If the candidate is generated in an extended sequence or inter, its information is stored in the pseudo-IDList. Algorithm 3.1 demonstrates how to create a pseudo-IDList from two frequent inter-sequence patterns with sequence extension, while Algorithm 3.2 illustrates how to create a pseudo-IDList from two frequent inter-sequence patterns with inter-extension.
- Figure 3.8 and Figure 3.10 show how to generate a pseudo-IDList from two frequent inter-sequence patterns based on the extended sequence condition, while Figure 3.9 and Figure 3.11 illustrate how to generate a pseudo-IDList from two frequent inter-sequence patterns based on the extended inter condition. From a pseudo-IDList, we aim to retrieve the information about the PatternList described in Figure 3.12.

Step 3: If the candidate generated in Step 2 is frequent (for instance, its support $\geq minsup$), store the candidate. Use the depth-first traversal algorithm to further generate candidates. Traverse to the end of the branch (for instance, $\langle A \rangle$ pattern) when there are no more candidates, then repeat Step 2. The algorithm stops when there are no more new candidates being generated.

Algorithm 3.1. Generating a pseudo-IDList from two frequent inter-sequence PatternList for s-extension.

Input: PatternList of α pattern and β pattern

Output: Pseudo-IDList of γ pattern

1. **let** M_α be α's PatternList
2. **let** M_β be β's PatternList

3. r_α {t-value, p-value} ← first row in M_α
4. r_β {t-value, p-value} ← first row in M_β
5. **while** $r_\alpha.t$-value ≤ M_α.row and $r_\beta.t$-value ≤ M_β.row **do**
6. **if** $r_\alpha.t$-value < $r_\beta.t$-value **then**
7. $r_\alpha.t$-value ← $r_\alpha.t$-value + 1 // move r_α to next row in M_α
8. **else if** $r_\alpha.t$-value > $r_\beta.t$-value **then**
9. $r_\beta.t$-value ← $r_\beta.t$-value + 1 //move r_β to next row in M_β
10. **else if** $r_\alpha.t$-value = $r_\beta.t$-value **then**
11. p_α ← the first element in $r_\alpha.p$-value
12. $r_\gamma.t$-value ← $r_\alpha.t$-value
13. **for each** p_β in $r_\beta.p$-value **do**
14. **if** $p_\beta > p_\alpha$ **then**
15. M_γ {t-value, p-value} ← {$r_\gamma.t$-value, p_β}//add a new row to M_γ
16. **break**;
17. $r_\alpha.t$-value ← $r_\alpha.t$-value + 1 // move r_α to next row in M_α
18. $r_\beta.t$-value ← $r_\beta.t$-value + 1 //move r_β to next row in M_β
19. γ's PP ← β's PP// PP of γ is a β pattern
20. **return** M_γ

Algorithm 3.2. Generating a pseudo-IDList from two frequent inter-sequence PatternList for t-extension.

Input: PatternList of α pattern, β pattern and $maxspan$

Output: Pseudo-IDList of γ pattern

1. **let** M_α be α's PatternList
2. **let** M_β be β's PatternList
3. r_α {t-value, p-value} ← first row in M_α
4. r_β {t-value, p-value} ← first row in M_β
5. **while** $r_\alpha.t$-value + $maxspan$ ≤ M_β.row **do**
6. **if** $r_\alpha.t$-value + $maxspan$ < $r_\beta.t$-value **then**
7. $r_\alpha.t$-value ← $r_\alpha.t$-value + 1 // move r_α to next row in M_α
8. **else if** $r_\alpha.t$-value + $maxspan$ > $r_\beta.t$-value **then**
9. $r_\beta.t$-value ← $r_\beta.t$-value + 1 //move r_β to next row in M_β
10. **else if** $r_\alpha.t$-value + $maxspan$ = $r_\beta.t$-value **then**

11. $r_\gamma.t$-value ← $r_\alpha.t$-value
12. $r_\gamma.p$-value ← first position of $r_\beta.t$-value
15. M_γ {t-value, p-value} ← {$r_\gamma.t$-value, $r_\gamma.p$-value}//add a new row to M_γ
17. $r_\alpha.t$-value ← $r_\alpha.t$-value + 1 // move r_α to next row in M_α
18. $r_\beta.t$-value ← $r_\beta.t$-value + 1 //move r_β to next row in M_β
19. γ's PP ← β's PP// PP of γ is a β pattern
20. **return** M_γ

The ISP-PI algorithm is applied to the sequential database in Table 2.4 with input values of $maxspan = 1$, $minsupport = 2$. The frequent inter-sequence patterns generated is shown in Figure 3.14 during the mining process.

Step 1: The support of each pattern in the database is calculated. The patterns $\langle A\rangle$, $\langle B\rangle$, $\langle C\rangle$, and $\langle D\rangle$ have support values of 4, 2, 3, and 1, respectively. Since $minsup$ is equal to 2, a pattern is considered frequent if it has a support value greater than or equal to 2. The set of frequent inter-sequence patterns found are $\{\langle A\rangle, \langle B\rangle, \langle C\rangle\}$, while pattern $\langle D\rangle$ is discarded due to its support value being less than 2. A PatternList structure is generated for the set of frequent inter-sequence patterns, as illustrated in Figure 3.5.

Step 2: Frequent inter-sequence pattern $\langle A\rangle$ is combined with the set of frequent inter-sequence patterns $\{\langle A\rangle, \langle B\rangle, \langle C\rangle\}$ to generate 2-pattern candidates. The set of candidates includes

$$\begin{Bmatrix} \langle AA\rangle[0], \langle AB\rangle[0], \langle AC\rangle[0], \langle (AB)\rangle[0], \langle (AC)\rangle[0], \\ \langle A\rangle[0]\langle A\rangle[1], \langle A\rangle[0]\langle B\rangle[1], \langle A\rangle[0]\langle C\rangle[1] \end{Bmatrix}$$

Candidates $\langle (AB)\rangle[0]$ and $\langle (AC)\rangle[0]$ are i-extension candidates, so a PatternList structure is generated for both candidates. The support values for these candidates are calculated as 0 and 2, respectively. Candidate $\langle (AB)\rangle[0]$ is removed due to having a support value less than the $minsup$ value of 2, i.e., $support(\langle (AB)\rangle[0]) = 0 < 2$.

- Candidates $\langle AA\rangle[0]$, $\langle AB\rangle[0]$, $\langle AC\rangle[0]$, $\langle A\rangle[0]\langle A\rangle[1]$, $\langle A\rangle[0]\langle B\rangle[1]$, and $\langle A\rangle[0]\langle C\rangle[1]$ are s-extension and t-extension candidates, so a pseudo-IDList structure is generated for each of them. The support values for these candidates are calculated as 2, 1, 3, 3, 1, and 2, respectively. Candidates $\langle AB\rangle[0]$ and $\langle A\rangle[0]\langle B\rangle[1]$ are removed due to having a support value less than the $minsup$ value of 2, i.e., $support(\langle AB\rangle[0]) = 1 < 2$ and $support(\langle A\rangle[0]\langle B\rangle[1]) = 1 < 2$.
- Since candidate $\langle AB\rangle[0]$ is discarded, candidate $\langle BA\rangle[0]$ is also discarded after applying Lemma 3.1.

Step 3: The set of frequent inter-sequence 2-patterns obtained from pattern $\langle A \rangle$ are $\{\langle (AC) \rangle [0], \langle AC \rangle [0], \langle A \rangle [0] \langle A \rangle [1], \langle AA \rangle [0], \langle A \rangle [0] \langle C \rangle [1]\}$, as shown at level 2 of pattern $\langle A \rangle$ in Figure 3.12. The algorithm continues to expand sub-patterns of pattern $\langle A \rangle$, and when no more candidates are generated, the algorithm expands patterns $\langle B \rangle$ and $\langle C \rangle$. The algorithm stops when all sub-patterns of the $\langle C \rangle$ pattern are generated.

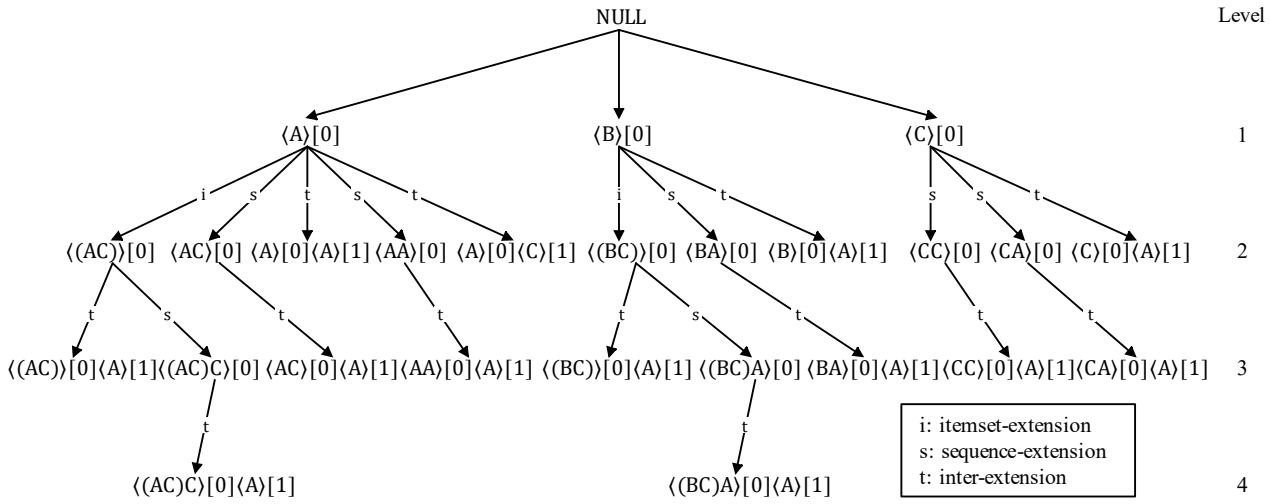


Figure 3.14. A set of frequent inter-sequence patterns implements from the the example database Table 2.4.

3.3.4 Computational Complexity Analysis

Calculating the complexity of the ISP-PI algorithm can be a complex task. Let m represent the number of transactions and n represent the number of distinct items. The ISP-PI algorithm is executed through two primary steps: firstly, reading data from the original database and constructing an ISP-tree of single-element common patterns; secondly, conducting mining on the tree based on a depth-first traversal approach.

The database scan and ISP tree construction are performed in linear time, as each transaction is processed individually, and each item is added to the tree. In the worst-case scenario, where each transaction contains all n items, the ISP-tree will consist of n frequent 1-patterns at level 1.

Next, we analyze the complexity of Algorithm 3.1 and Algorithm **3.2** in the subsequent mining process. Let X and Y represent two patterns, with X having k elements and Y having z elements. The worst-case running time for Algorithm 3.1 occurs when $X \subseteq Y$, as it requires simultaneous examination of both samples X and Y. Therefore, the time complexity to find the new pattern is $O(k + z)$. Similarly, for Algorithm 3.2, if $k + maxspan \leq z$, the algorithm needs to traverse all lines of the two samples X and Y, resulting in a time complexity of $O(k + z - maxspan)$.

3.4 Experimental Results

In this section, we evaluate the runtime and memory usage performance of the ISP-PI algorithm. All experiments were performed on a PC equipped with an Intel® Core ™ i7 10th generation processor (10510U) running at a speed of 1.8 - 2.0 GHz, and 20 GB of RAM. The operating system used was Windows 11 64-bit. The algorithms were implemented in the Java programming language using JDK 19.

3.4.1 Experimental Databases

To evaluate the proposed algorithm, we conducted experiments to compare its performance with two already established algorithms, namely Post-EISPMiner (C. S. Wang & Lee, 2009) and Post-DBV-ISP (Vo et al., 2012). We tested our algorithm on a total of six databases, namely C150S40T2, C200S12T5, BMSWebView2, FIFA, Kosarak, and MSNBC. Four of these databases are real-life databases, with MSNBC, Kosarak considered as big databases, and FIFA, BMSWebView2 as medium-sized ones. These databases are publicly available via link https://www.philippe-fournier-viger.com/spmf/index.php?link=datasets.php. Additionally, we used two synthetic test databases, C150S40T2 and C200S12T5, generated using the standard generator in (Agrawal & Srikant, 1995), which can be accessed at the following link:

https://www.mediafire.com/folder/pn3myfebx4t0e/PseudoIDList.

The characteristics of each of the six test databases are presented in Table 3.2, while Table 3.3 provides the values of *maxspan* and *minsupport*.

Table 3.2. Test database information

Database name	Database size	Distinct items	Average sequence length
C150S40T2	150,000	954	76.64
C200S12T5	183,950	1,922	51.57
FIFA	20,450	2,990	36.24
BMSWebView2	77,512	3,340	4.62
Kosarak	990,002	41,270	8.1
MSNBC	989,818	17	4.75

Table 3.3. The number of frequent inter-sequence patterns of the six test databases with *maxspan* is given from 0 to 5.

Database	*minsupport*(%) / maxspan	Number of frequent inter-sequence patterns					
		0	1	2	3	4	5
C150S40T2	6	790	3,053	5,315	7,573	9,828	12,109

C200S12T5	3	624	628	633	638	645	650
BMSWebView2	0.02	1,316,614	1,323,073	1,329,145	1,337,433	1,344,028	1,351,129
FIFA	9	68,465	79,962	93,061	105,133	117,405	130,283
Kosarak	0.6	1,135	2,221	3,311	4,402	5,491	6,587
MSNBC	0.2	4,244	6,392	8,511	10,653	12,770	14,887

Table 3.4. Comparison table of the percentage of candidates generated when ISP-IC pruning is applied and when it is not applied for six databases listed in Table 3.2.

	C150S40T2 ($minsupport = 6\%$)			C200S12T5 ($minsupport = 3\%$)			FIFA ($minsupport = 9\%$)		
maxspan	using ISP-IC	without using ISP-IC	Candidate reduction (%)	using ISP-IC	without using ISP-IC	Candidate reduction (%)	using ISP-IC	without using ISP-IC	Candidate reduction (%)
1	195,938	227,812	14.0	389,376	390,096	0.18	394,373	929,721	57.58
2	391,302	448,249	12.7	778,752	779,472	0.09	416,476	1,151,995	63.85
3	586,666	668,707	12.3	1,168,128	1,168,848	0.06	437,554	1,361,019	67.85
4	782,030	888,993	12.0	1,557,504	1,558,224	0.05	458,833	1,574,049	70.85
5	977,394	1,109,586	11.9	1,946,880	1,947,600	0.04	480,717	1,790,539	73.15

	BMSWebView2 ($minsupport = 0.02\%$)			Kosarak ($minsupport = 0.6\%$)			MSNBC ($minsupport = 0.2\%$)		
maxspan	using ISP-IC	without using ISP-IC	Candidate reduction (%)	using ISP-IC	without using ISP-IC	Candidate reduction (%)	using ISP-IC	without using ISP-IC	Candidate reduction (%)
1	8,050,516	12,610,808	36.16	15,270	32,173	52.54	10,238	24,314	57.89
2	14786,831	19,903,854	25.71	29,526	60,088	50.86	12,430	38,623	67.82
3	21,525,247	27,304,621	21.17	43,784	87,852	50.16	14,644	53,027	72.38
4	28,261,937	34,644,592	18.42	58,038	115,830	49.89	16,835	67,299	74.98
5	34,999,009	42,024,651	16.72	72,300	143,821	49.73	19,024	81,614	76.69

3.4.2 ISP-IC Evaluation

The experimental results of applying the early candidate pruning model are presented in Table 3.4. The results show that the ISP-IC model is effective in all six test databases used. Specifically, the ISP-IC model consistently prunes a significant number of candidates, thereby optimizing the algorithm processing speed and storage space.

For example, in the case of the large Kosarak and MSNBC databases, more than 50% of candidates were pruned. The best-case scenario for the Kosarak database ($minsupport = 0.6\%$) resulted in 52.54% of the candidates being discarded when

$maxspan = 1$, while for the MSNBC database ($minsupport = 0.2\%$), 76.69% of candidates were discarded when $maxspan = 5$. However, the results also indicate that the number of candidates to be eliminated depends on the database and the randomness of items in the transactions belonging to the sequential database. In the case of the C200S12T5 database, only 0.18% of the candidates were removed in the best case.

3.4.3 Runtime

The results presented in Figure 3.15 compare the running time of the proposed algorithm, ISP-PI, with that of two state-of-the-art algorithms, EISP-Miner and DBV-ISP, for MISP. Across various database evaluation scenarios, the ISP-PI algorithm demonstrated consistently faster performance compared to the other two algorithms. As the *maxspan* value increases in IPM, the number of candidate patterns generated also increases, as the algorithm must consider the relationship between transactions across the sequential database. Table 3.3 indicates that using a *maxspan* value of five results in a greater number of frequent inter-sequence patterns compared to smaller values. When comparing the performance of the three algorithms using two large databases, the ISP-PI algorithm runs significantly faster than the EISP-Miner and DBV-ISP algorithms, with improvements of 81.21% and 57.53% for the Kosarak test database, and 85.00% and 79.07% for the MSNBC test database, respectively. The superior performance of the ISP-PI algorithm can be attributed to its data structure and candidate pruning method. The proposed pseudo-IDList approach eliminates the need to replicate duplicate data multiple times during the mining process. As all IPM algorithms employ the depth-first-search traversal method, the proposed data structure ensures that the algorithm does not accumulate a large amount of data during each backtracking operation. Instead, it only needs to quickly retrieve data when necessary. In addition, the pruning method helps to reduce the number of candidates generated, because every time a candidate is generated, we have to calculate its support, which reduces the running time of the algorithm.

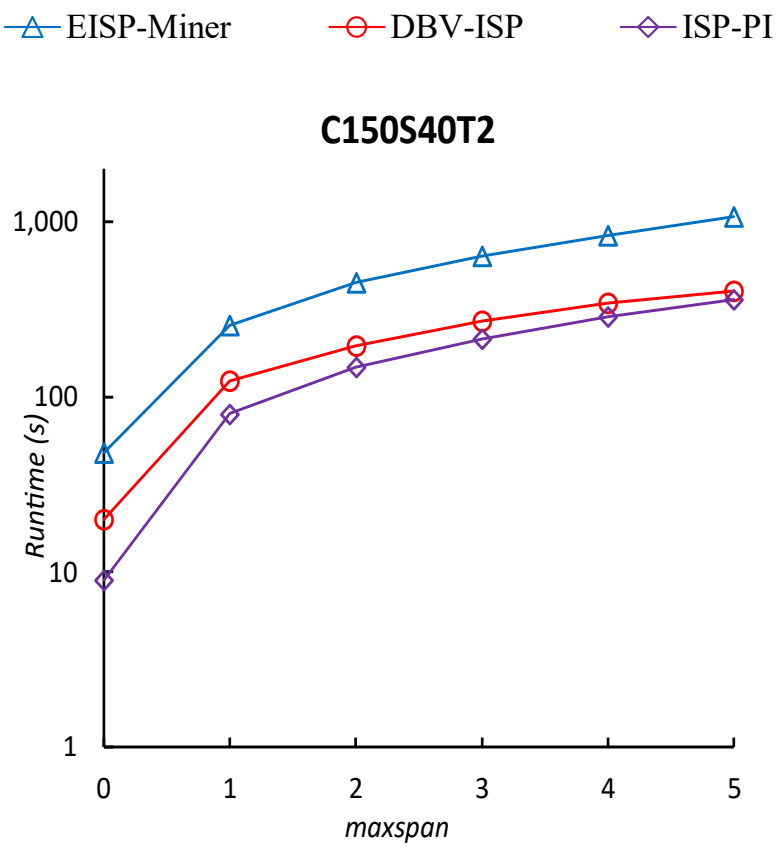


Figure 3.15. Runtime on C150S40T2 database.

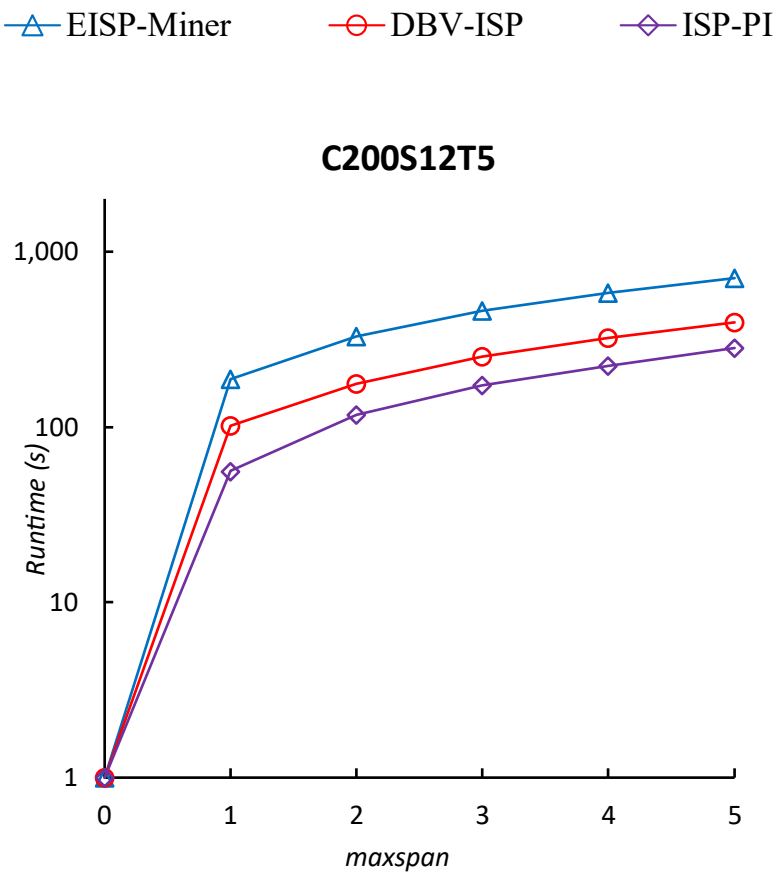


Figure 3.16. Runtime on C200S12T5 database.

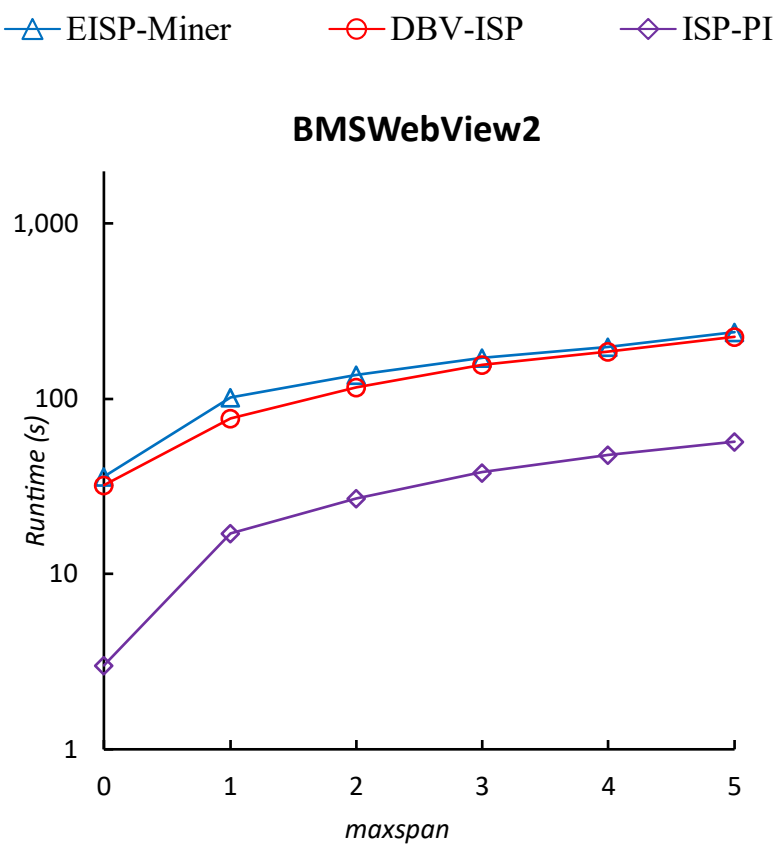


Figure 3.17. Runtime on BMSWebView2 database.

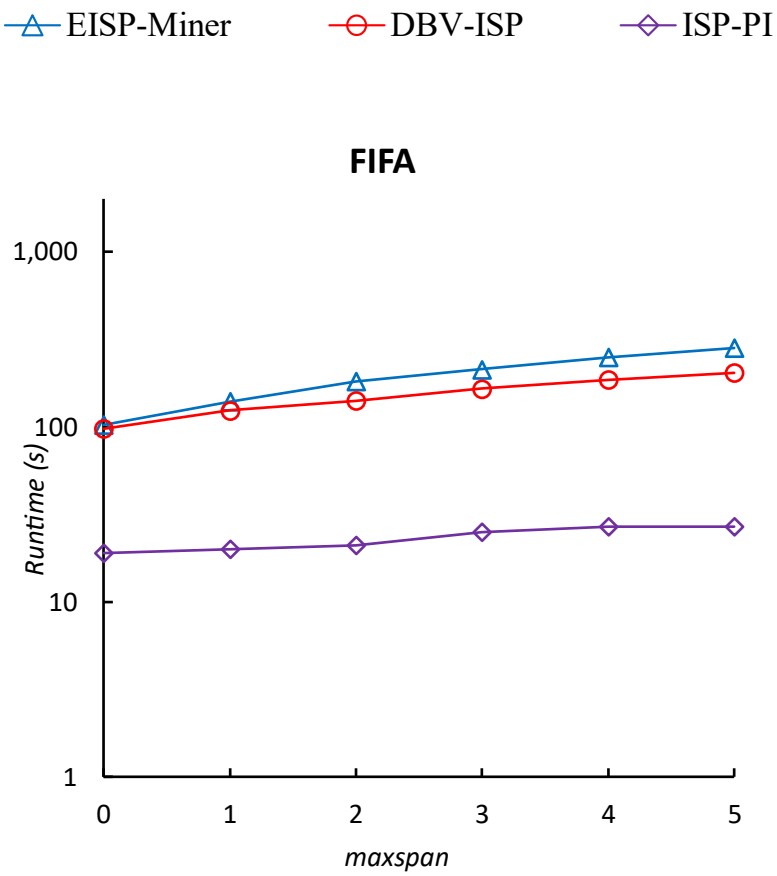


Figure 3.18. Runtime on FIFA database.

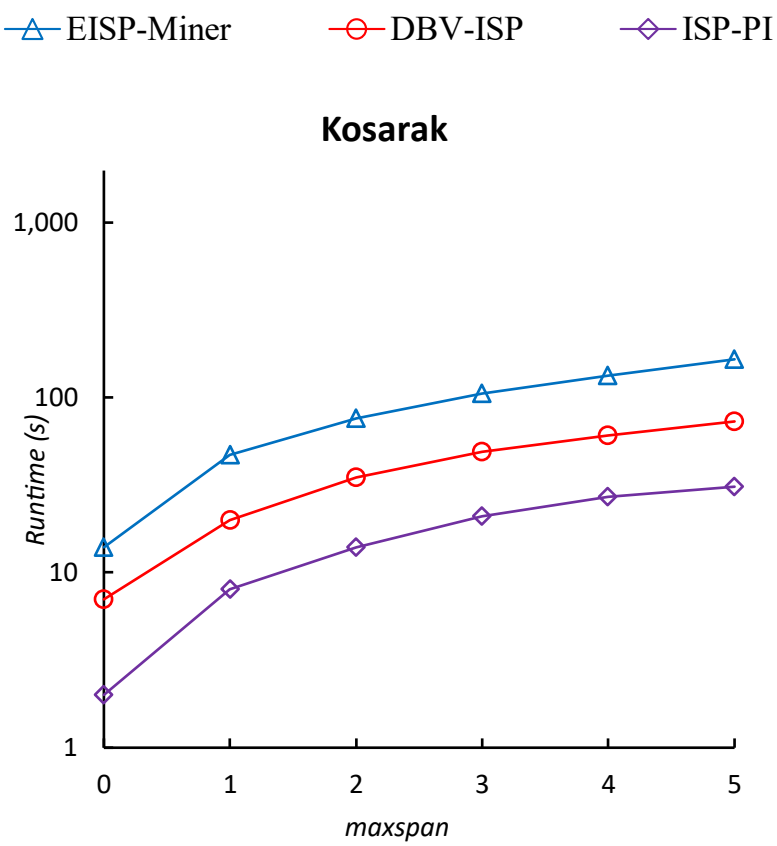


Figure 3.19. Runtime on Kosarak database.

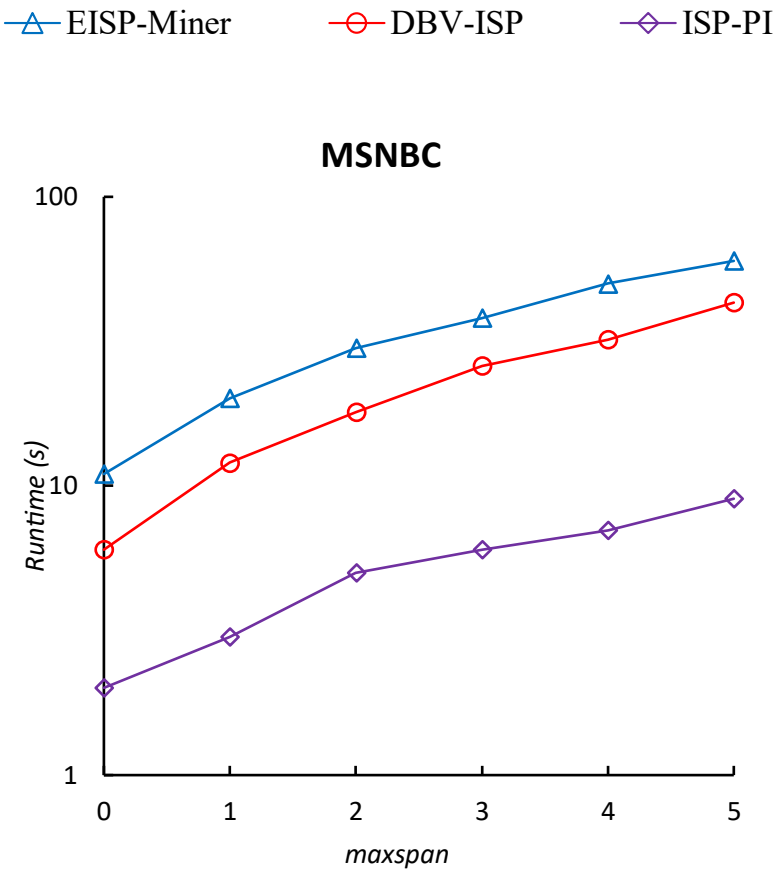


Figure 3.20. Runtime on MSNBC database.

3.4.4 Memory Usage

Figure 3.15 demonstrates that the ISP-PI algorithm uses memory more efficiently than its predecessors, EISP-Miner and DBV-ISP, across the databases used for evaluation. Figure 3.7 illustrates that the EISP-Miner and DBV-ISP inter-sequence mining algorithms generate and store all the information about a candidate, leading to increased memory usage. In contrast, the ISP-PI algorithm mitigates this issue by utilizing the pseudo-IDList data structure, which requires less memory. For instance, in the Kosarak database the ISP-PI algorithm uses 82.40% and 81.37% less memory than EISP-Miner and DBV-ISP, respectively. Similarly, in the MSNBC database the ISP-PI algorithm uses 56.29% and 53.65% less memory than EISP-Miner and DBV-ISP, respectively. However, the ISP-PI algorithm only applies the pseudo-IDList structure to the sequence and inter-extensions, while it still needs to store all candidate information for the itemset extension method.

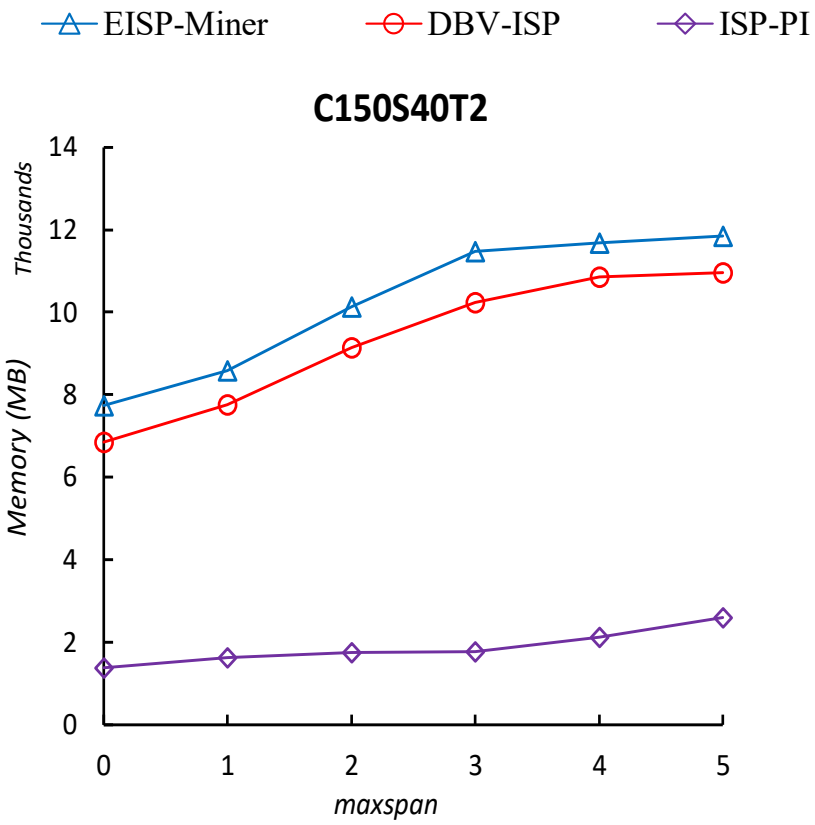


Figure 3.21. Memory usage on C150S4T2 database.

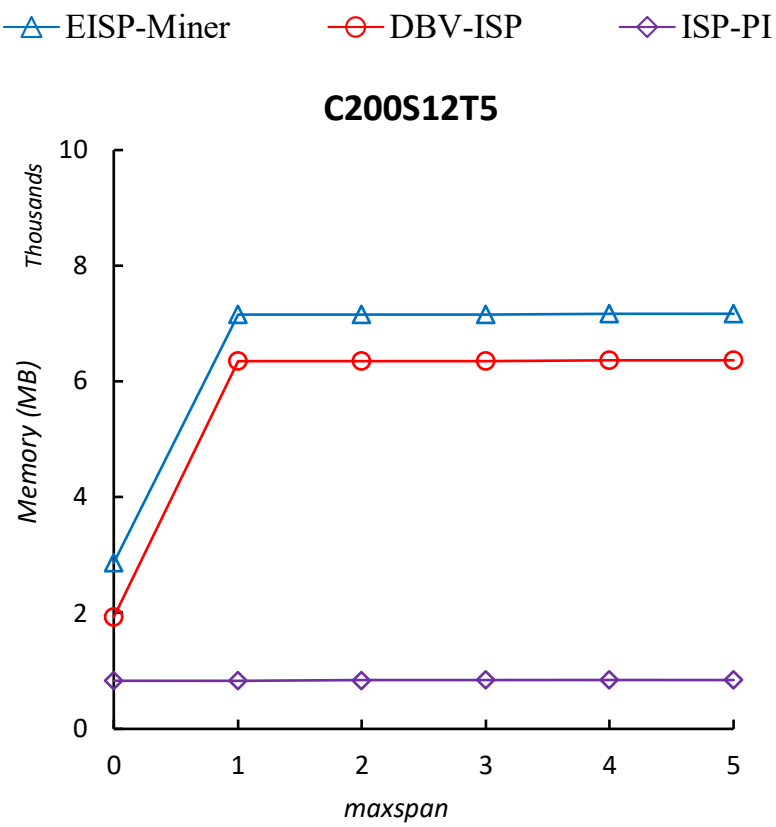


Figure 3.22. Memory usage on C200S12T5 database.

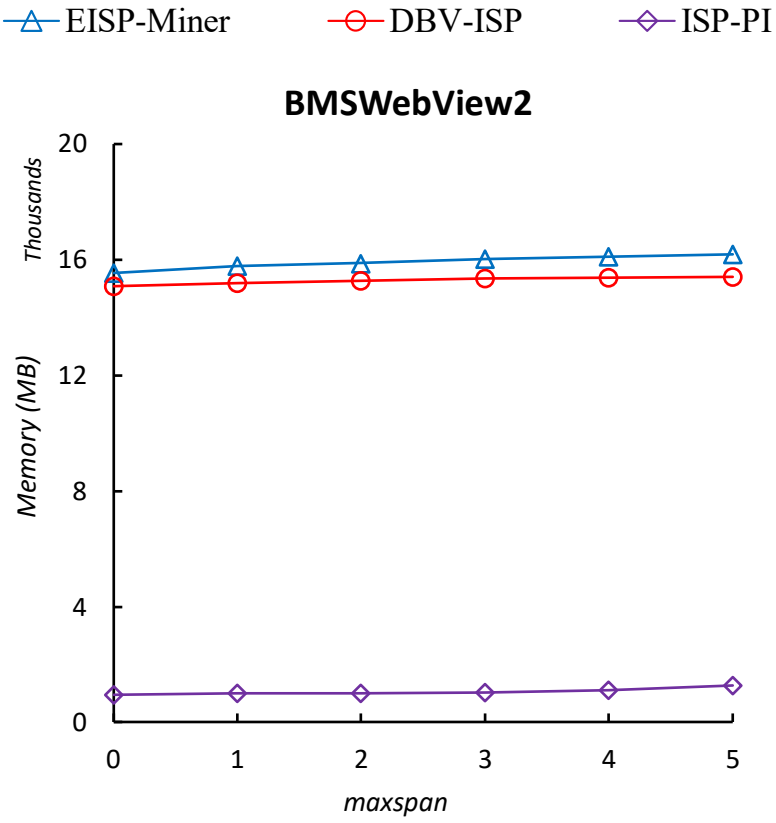


Figure 3.23. Memory usage on BMSWebView2 database.

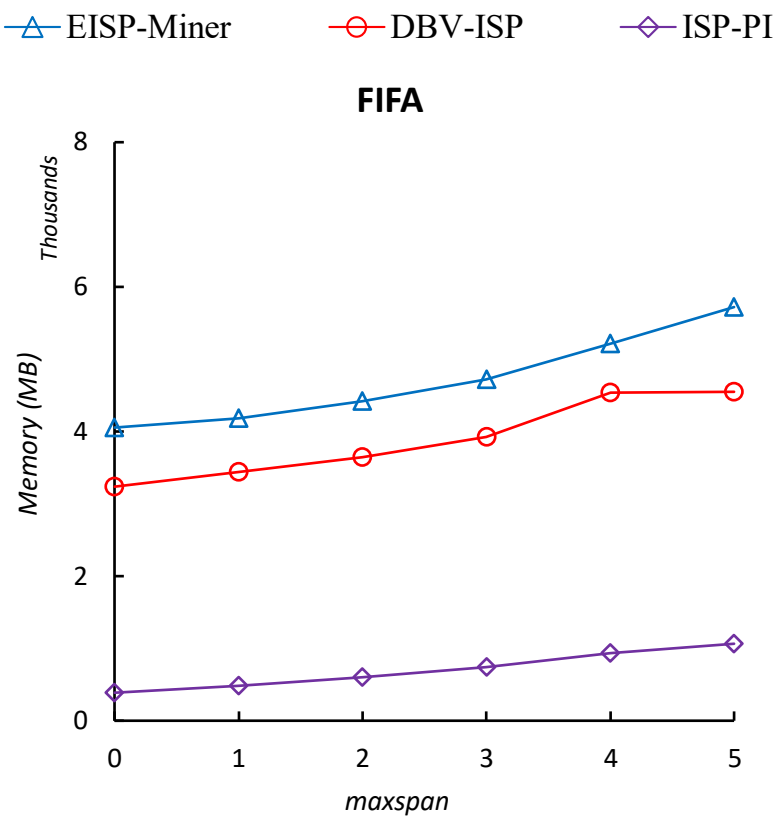


Figure 3.24. Memory usage on FIFA database.

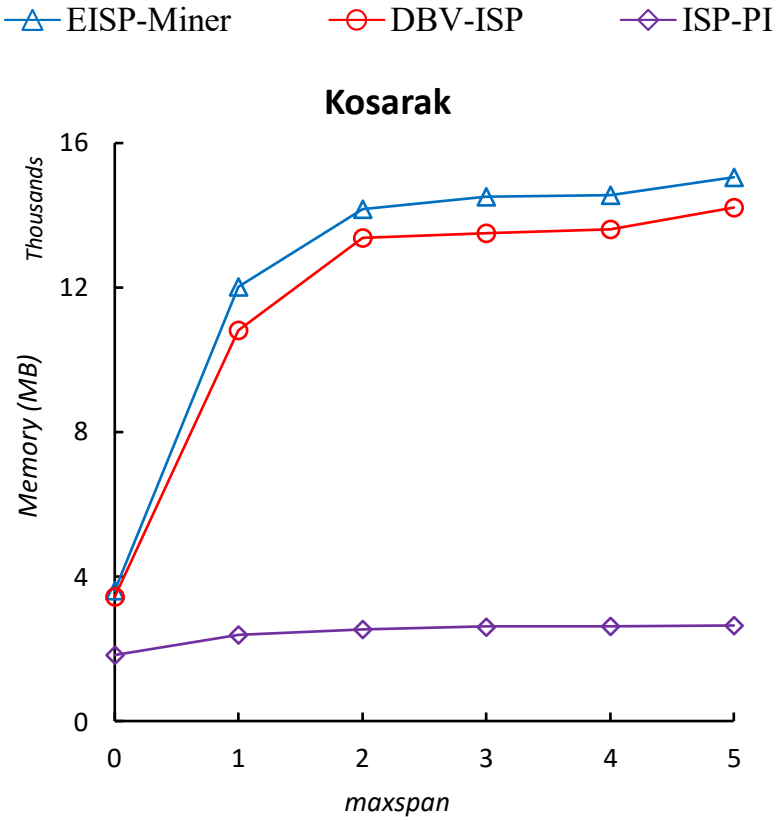


Figure 3.25. Memory usage on Kosarak database.

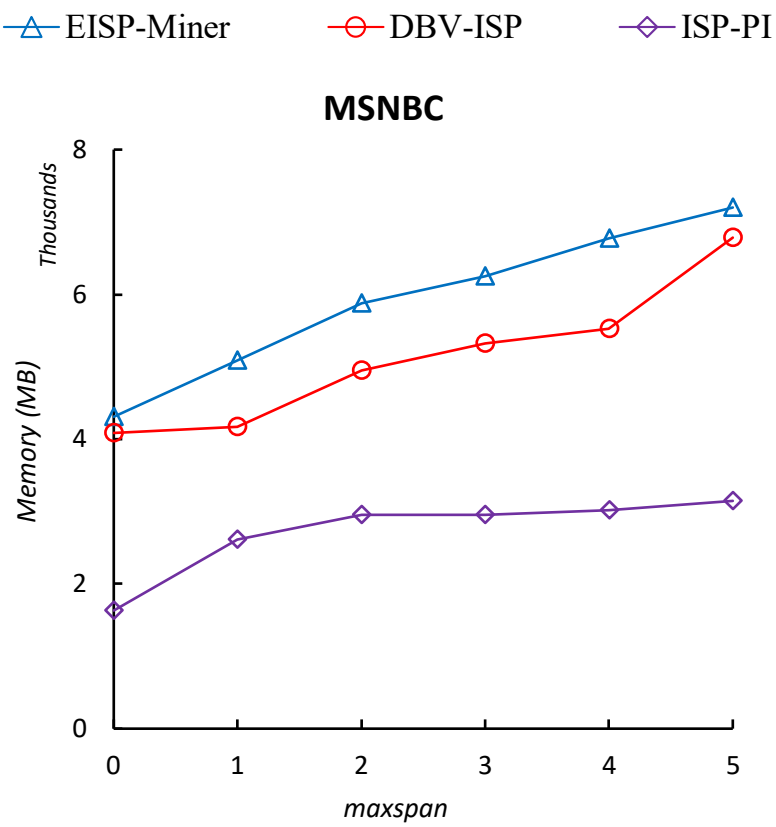


Figure 3.26. Memory usage on MSNBC database.

3.5 Summary

In this chapter, we discuss the limitations of algorithms for inter-sequence pattern mining, particularly the issue of excessive memory usage caused by data duplication. To address this problem, we propose the ISP-PI (Inter-Sequence Pattern mining based on Pseudo-Index) algorithm, which employs the pseudo-IDList data structure. This data structure enables the ISP-PI algorithm to access the frequent pattern information from its 1-pattern frequent pattern, significantly reducing memory consumption during the algorithm's execution. Furthermore, we optimize the ISP-PI algorithm and introduce two lemmas that facilitate swift support condition checks for candidates, thereby improving the algorithm's runtime.

Our proposed algorithm uses depth-first traversal, and thus future work will investigate parallel processing or distributed processing to further enhance its speed. Additionally, since heuristic pruning only considers items in the same itemset or transaction, a follow-up approach is necessary to pre-check items across transactions to reduce the number of candidates generated.

CHAPTER 4: METHODS FOR MINING INTER-SEQUENCE PATTERNS WITH CONSTRAINTS

In this chapter, we address the problem of mining inter-sequence patterns with itemset constraints. We propose an algorithm called DBV-ISPMIC to tackle this problem. Furthermore, we enhance the efficiency of the DBV-ISPMIC algorithm by introducing an improved version named DBV-ISPMIC-IMPROVING, which incorporates efficient pruning methods. Additionally, we present a parallel algorithm called *p*DBV-ISPMIC-IMPROVING that leverages parallel computing techniques. To assess the performance of the proposed algorithms compared to previous approaches such as EISP-Miner (C. S. Wang & Lee, 2009) and DBV-ISP (Vo et al., 2012), we evaluate their running time and space storage on five test databases. The experimental results demonstrate the efficiency of the proposed algorithms.

4.1 Introduction

In 2018, Van et al. proposed the MSPIC-DBV algorithm to mine sequential patterns with itemset constraints (Van et al., 2018a). 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 (Van & Le, 2021). 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. (T. Le et al., 2018) for constraints-based inter-sequence pattern mining, as well as its improvements, *i*ISP-IC and *p*iISP-IC. The authors also applied a parallel processing method to speed up the runtime. As stated in the introduction, ISP-IC, *i*ISP-IC, and *pi*ISP-IC focus on the condition of items in the sequence, and we cannot apply this approach to the itemset problem.

In this chapter, 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 (C. S. Wang & Lee, 2009) and a using dynamic bit vector data structure (Vo et al., 2012), 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 parellel version of DBV-ISPMIC algorithm, named *p*DBV-ISPMIC algorithm, was presented.
4. Finally, we conduct experiments with various databases to evaluate the proposed method.

For instance, based on Definition 2.23 and 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.

4.2 DBV-PatternList Structure

The DBV-PatternList data structure as proposed by Vo et al. in 2012 (Vo et al., 2012). 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 4.1 presents the use of PatternList and DBV-PatternList structures.

e-item	<A>[0]					
t-value	1	2	3	4	5	7
	↓	↓	↓	↓	↓	↓
p-value	2	2 3	1	1	1	1

(a) PatternList structure

e-item	<A>[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

Figure 4.1. Structures of (a) PatternList and (b) DBV-PatternList.

As in Figure 4.1a, 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 (Figure 4.1b) 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 2.5), storage space is reduced.

4.3 Algorithms

4.3.1 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 Algorithm 4.1.

In this algorithm, there are five functions – ISP-Join1, ISP-Joink, ISP-Join1-Extension, ISP-Joink-Extension and Check – as shown in Algorithm 4.2-Algorithm **4.6**, respectively. The algorithm computes on a given sequential database with *minsupport*, *maxspan* and a set of itemset constraints defined by the user. The result is a set of frequent sequential patterns that satisfy the conditions of *minsupport* 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 *minsupport* (Algorithm 4.1, 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 (Algorithm 4.1, line 2).

Step 2: The algorithm calls the ISP-Join1 function (Algorithm 4.2) for extending an α_{list} node with the remaining children of the $T|NULL$ tree (Definition 2.20). 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 *minsupport* condition and itemset constraints. The Check function in Algorithm 4.6 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 $T|\alpha_{list}$ (Algorithm 4.4, lines 1-9).

Step 3: The algorithm calls the ISP-Joink function (Algorithm 4.3) for extending the child node of the α_{list} node with the remaining children according to the *k*-pattern (Definition 2.21). 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 *minsupport* condition. The Check function is used to check if a new DBV-PatternList satisfies the constraint. If it is satisfied, we will add this DBV-PatternList as a child node of $T|\alpha_{list}$ (Algorithm 4.5, lines 1-9).

Algorithm 4.1. The DBV-ISPMIC algorithm

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

Output: A complete set of frequent inter-sequence patterns FP satisfying *minsupport* 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 α_{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;

Algorithm 4.2. The ISP-Join1 function.

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

1. **for each** frequent 1-DBV-PatternList γ_{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**

Algorithm 4.3. The ISP-Joink function.

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

1. **for each** frequent *k*-DBV-PatternList β_{list} in $T|\alpha_{list}$, where $sub_{k,k}(\beta) = \langle u \rangle[i]$ **do**
2. | **for each** frequent *k*-DBV-PatternList γ_{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;

Algorithm 4.4. 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** $support(\theta_{list}) \geq minsupport$ and $Check(\theta_{list}, IC)$ **then** add θ_{list} to $T|\alpha_{list}$ and θ to FP;
4. **if** $(x = 0)$ **then**
5. | $\rho_{list} = \alpha_{list} \cup_s \gamma_{list}$;
6. | **if** $support(\rho_{list}) \geq minsupport$ and $Check(\rho_{list}, IC)$ **then** add ρ_{list} to $T|\alpha_{list}$ and ρ to FP;
7. **if** $(x > 0)$ **then**
8. | $\sigma_{list} = \alpha_{list} \cup_t \gamma_{list}$;
9. | **if** $support(\sigma_{list}) \geq minsupport$ and $Check(\sigma_{list}, IC)$ **then** add σ_{list} to $T|\alpha_{list}$ and σ to FP;

Algorithm 4.5. The ISP-Joink-Extension function.

Function 4: *ISP-Joink-Extension*(T, FP, IC, β_{list}, γ_{list}, u, v, i, j)

1. **if** (i=j) and (u<v) **then**
2. $\theta_{list} = \beta_{list} \cup_i \gamma_{list}$;
3. **if** $support(\theta_{list}) \geq minsupport$ and Check(θ_{list}, IC) **then** add θ_{list} to $T|\beta_{list}$ and θ to FP;
4. **if** (i=j) **then**
5. $\rho_{list} = \beta_{list} \cup_s \gamma_{list}$;
6. **if** $support(\rho_{list}) \geq minsupport$ and Check(ρ_{list}, IC) **then** add ρ_{list} to $T|\beta_{list}$ and ρ to FP;
7. **if** (i<j) **then**
8. $\sigma_{list} = \beta_{list} \cup_t \gamma_{list}$;
9. **if** $support(\sigma_{list}) \geq minsupport$ and Check(σ_{list}, IC) **then** add σ_{list} to $T|\beta_{list}$ and σ to FP;

Algorithm 4.6. 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;

4.3.2 Computational Complexity Analysis

Exploring the precise complexity of the DBV-ISPMIC algorithm (Algorithm 4.1) is a highly challenging task. The algorithm consists of two primary steps: reading data from the original database and performing the mining process. Let n denote the number of distinct items in the database.

The initial database scan is conducted in a linear manner, where each line of the database is scanned. In the worst-case scenario, every row of the database contains all n items. Consequently, the number of frequent 1-patterns is also equal to n.

Subsequently, the mining time of the algorithm is calculated by $\sum_1^n time\big(branch\_of\_tree(i)\big)$. Two subfunctions, ISP-Join1-Extension (Algorithm 4.4) and ISP-Joink-Extension (Algorithm 4.5), are utilized for the prototyping process. Thus,

the mining time is determined based on these two functions. Let X represent the number of frequent patterns from the ISP-Join1-Extension function. If a frequent pattern is discovered for each $x_i \in X | 1 \leq i \leq |X|$, the number of iterations for the ISP-Joink-Extension function is calculated accordingly $\sum_1^X time(\text{ISP} - \text{Joink} - \text{Extension})$. This scenario represents the worst-case execution of the algorithm $\sum_1^n time(\text{ISP} - \text{Join1} - \text{Extension}) \sum_1^X time(\text{ISP} - \text{Joink} - \text{Extension})$.

Regarding the Check function (Algorithm 4.6), its worst performance occurs when $\alpha \subseteq IC$.

4.3.3 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 4.1 (Checking itemset constraints) Given an inter-sequence α and a set of itemset constraints *IC*, if α satisfies *IC*, then the sequence β, generated from α, 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 α_i, β_i corresponds to an itemset. Because α satisfies *IC*, based on the problem statement, this condition holds: $\exists \alpha_i[w_i] \in \alpha, \exists b_j \in IC: b_j \subseteq \alpha_i$.

Based on Definition 2.21, there are three cases to consider:

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

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

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

Algorithm 4.7. The function ISP-Join1-improving.

Function 6: ISP-Join1-Improving(T, FP, IC, α_{list})

1. **For** each frequent 1-DBV-PatternList γ_{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. **If** Check(α_{list}, IC) then
4. **If** (x=0) and (u<v) **then**
5. $\theta_{list} = \alpha_{list}$ U$_i$ γ_{list};
6. **If** *support*(θ_{list}) ≥ *minsupport* **then** add θ_{list} to $T|\alpha_{list}$ and θ to FP;
7. **If** (x=0) **then**
8. $\rho_{list} = \alpha_{list}$ U$_s$ γ_{list};
9. **If** *support*(ρ_{list}) ≥ *minsupport* **then** add ρ_{list} to $T|\alpha_{list}$ and ρ to FP;
10. **If** (x>0) **then**
11. $\sigma_{list} = \alpha_{list}$ U$_t$ γ_{list};
12. **If** *support*(σ_{list}) ≥ *minsupport* **then** add σ_{list} to $T|\alpha_{list}$ and σ to FP;
13. **Else**
14. ISP-Join1-Extension($T|\alpha_{list}$, FP, IC, α_{list}, γ_{list}, u, v, x);
15. **End for**
16. **End for**

Algorithm 4.8. The function ISP-Joink-improving.

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

1. **For** each frequent k-DBV-PatternList β_{list} in $T|\alpha_{list}$, where $sub_{k,k}(\beta)=\langle u\rangle[i]$, **do**
2. **If** Check(β_{list}, IC) then
3. **For** each frequent k-DBV-PatternList γ_{list} in $T|\alpha_{list}$, where $sub_{k,k}(\gamma)=\langle v\rangle[j]$, **do**
4. **If** (i=j) and (u<v) **then**
5. $\theta_{list} = \beta_{list}$ U$_i$ γ_{list};
6. **If** $support(\theta_{list}) \geq minsupport$ **then** add θ_{list} to $T|\beta_{list}$ and θ to FP;
7. **If** (i=j) **then**
8. $\rho_{list} = \beta_{list}$ U$_s$ γ_{list};
9. **If** $support(\rho_{list}) \geq minsupport$ **then** add ρ_{list} to $T|\beta_{list}$ and ρ to FP;
10. **If** (i<j) **then**
11. $\sigma_{list} = \beta_{list}$ U$_t$ γ_{list};
12. **If** $support(\sigma_{list}) \geq minsupport$ **then** add σ_{list} to $T|\beta_{list}$ and σ to FP;
13. **End for**
14. **Else**
15. **For** each frequent k-DBV-PatternList γ_{list} in $T|\alpha_{list}$, where $sub_{k,k}(\gamma)=\langle v\rangle[j]$, **do**
16. ISP-Joink-Extension($T|\beta_{list}$, FP, IC, β_{list}, γ_{list}, u, v, i, j);
17. **End for**
18. **Call** ISP-Joink-Improving($T|\beta_{list}$, FP, IC);
19. **End for**
20. Delete $T|\alpha_{list}$ from T;

In the ISP-Join1-improving function, we first check the α_{list} pattern with the itemset constraints. If it is true, all frequent DBV-PatternList which are extended from α_{list} also satisfy the itemset constraints (Algorithm 4.7, lines 3-12). This is the same with the ISP-Joink-improving function (Algorithm 4.8, lines 2-13).

Consider the example shown in Table 2.1, where $minsupport = 2$, $maxspan = 1$ and itemset constraints $IC = \{(AB), CA, AD\}$. The frequent patterns generated in the mining phase are presented in Figure 4.2.

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 (Figure 4.2, 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 (Algorithm

4.7). Other generated candidates ⟨*B*⟩[0]⟨*A*⟩[1], ⟨*CA*⟩[0], ⟨*C*⟩[0]⟨*A*⟩[1] and ⟨*CB*⟩[0] that share 1-prefix ⟨*B*⟩, ⟨*C*⟩ are also created at the same time during the combination (Figure 4.2, 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, ⟨(*AB*)⟩[0] satisfies the itemset constraints, so we do not need to check the itemset constraints for ⟨(*AB*)⟩[0]⟨*A*⟩[1]. Due to this checking case, we reduce the checking time of the algorithm. Otherwise, ⟨*A*⟩[0]⟨*A*⟩[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 ⟨*B*⟩, ⟨*C*⟩ then ⟨*D*⟩. As no more candidates can be found in branch ⟨*D*⟩, the algorithm terminates. We have *FP* = {⟨*AA*⟩[0]⟨*AD*⟩[1]: *support* = 2, ⟨(*AB*)⟩[0]: *support* = 2, ⟨(*AB*)⟩[0]⟨*A*⟩[1]: *support* = 2, ⟨*AD*⟩[0]: *support* = 2, ⟨*CA*⟩[0]: *support* = 2, ⟨*CA*⟩[0]⟨*A*⟩[1]: *support* = 2, ⟨*C*(*AB*)⟩[0]: *support* = 2, ⟨*C*(*AB*)⟩[0]⟨*A*⟩[1]: *support* = 2}.

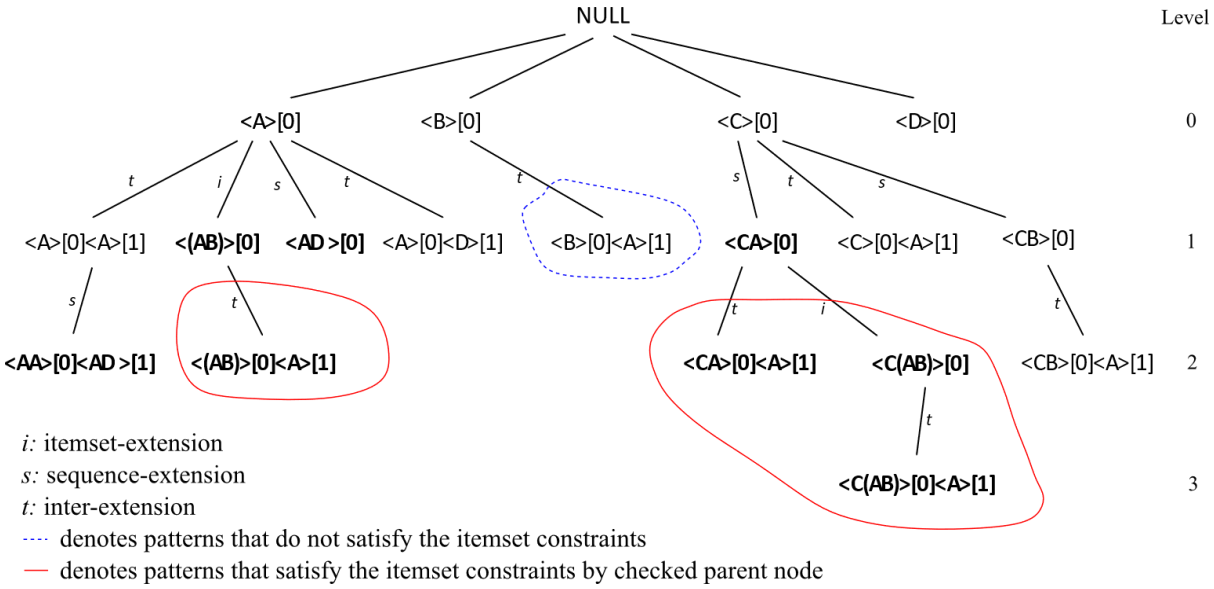


Figure 4.2. The extended tree of patterns corresponding to the example database.

4.3.4 Parallel DBV-ISPMIC Algorithm

As shown in Algorithm 4.1, the DBV-ISPMIC algorithm is a sequential algorithm. The complexity time of the DBV-ISPMIC algorithm is calculated by $t_{node\_1} + t_{node\_2} + t_{node\_3} + \ldots + t_{node\_n}$, whereas the set {*node_1*, *node_2*, *node_3*, …, *node_n*} contains child nodes of an ISP-tree *T*, t_{nodei} 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*, …, *node_n*}.

An example for the above proposal is given in Figure 4.3. Based on the database in Table 2 with *minsupport* = 2, the frequent patterns that were generated at the first level by 1-pattern extension included ⟨*A*⟩, ⟨*B*⟩, ⟨*C*⟩ and ⟨*D*⟩. 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 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-ISPMIC algorithm is based on the DBV-ISPMIC with the algorithm parallelized. Figure 4.4 shows with a flowchart the main steps of the sequential DBV-ISPMIC algorithm and its parallel (*p*DBV-ISPMIC) counterpart. The *p*DBV-ISPMIC algorithm has two main steps:

Step 1: Loading the database and finding the frequent 1-pattern sets that satisfy the *minsupport* and the itemset constraints.

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.

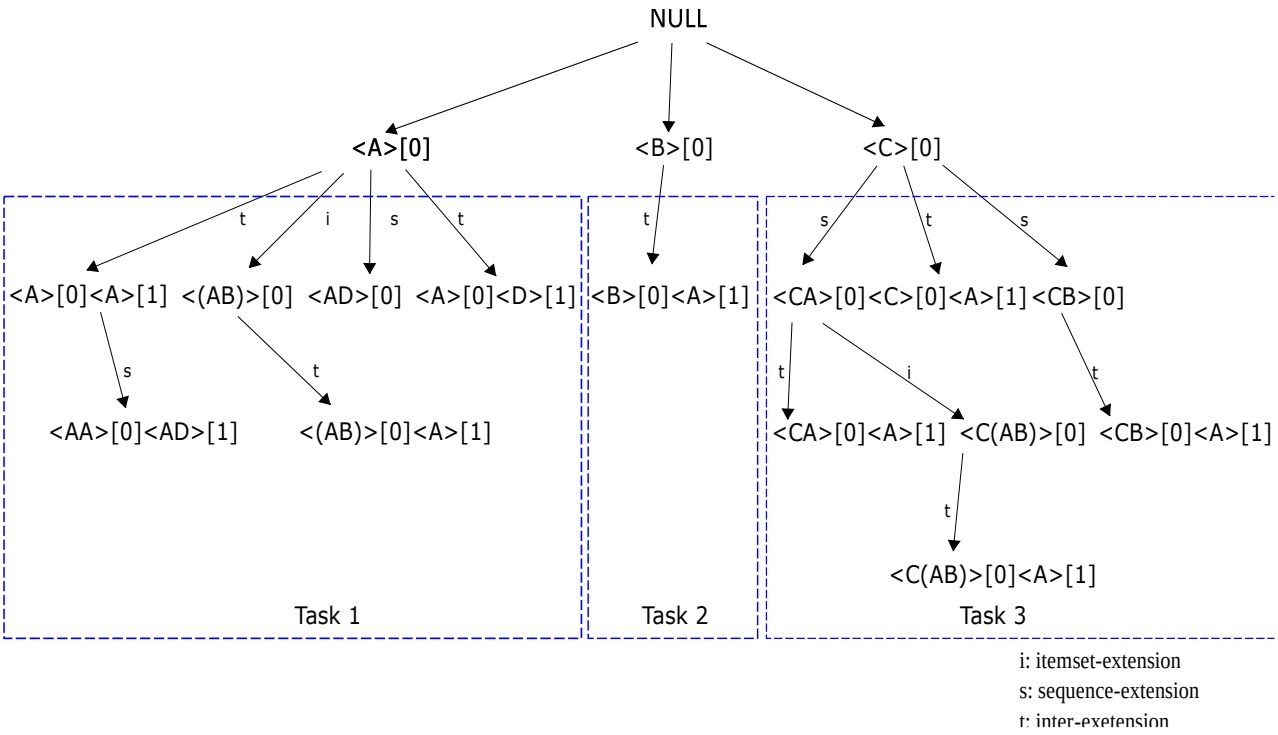


Figure 4.3. Example of using parallel processing for ISP-tree extension.

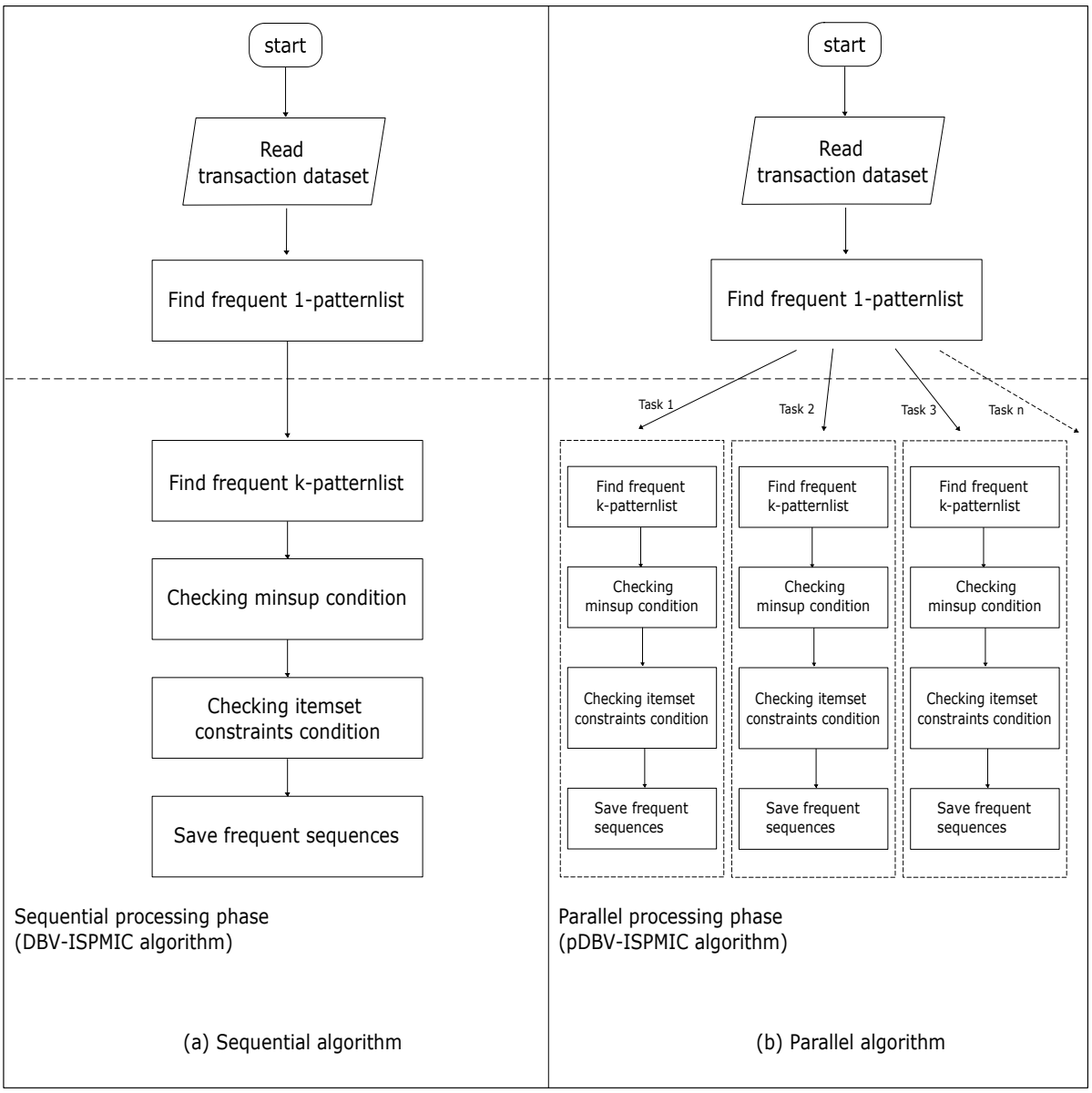


Figure 4.4. The figure shows the difference between sequential and parallel flow chart. (a) The main steps of a sequential algorithm and (b) the main steps of a parallel processing algorithm.

4.4 Experimental Evaluation

In evaluating the performance of the DBV-ISPMIC algorithm and its improvement in runtime, all the experiments were carried out on a PC with an Intel® Core ™ i7 10th gen processor (10510U) @ 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#.

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 4.1.

Table 4.1. Test database characteristics

Database	Sequence	Item	Type of data
C6T5S4I4N1kD1k	1000	1000	Synthetic databases
C6T5S4I4N1kD10k	10000	1000	Synthetic databases
Gazelle	59602	497	Clickstream data
BIKE	21078	67	Bike Share data from LA Metro
BMSWebView1	59601	497	Clickstream data

4.4.1 Runtime

For the C6T5S4I4N1kD1k database, we evaluated the algorithms with $minsupport = 0.5\%$ and $maxspan = \{1,2,3,4,5\}$, the C6T5S4I4N1kD10k database with $minsupport = 5\%$ and $maxspan = \{1, 2, 3, 4, 5\}$, the Gazelle database with $minsupport = 1\%$ and $maxspan = \{1,2,3,4,5\}$, the BIKE database with $minsupport = 0.5\%$ and $maxspan = \{1, 2, 3, 4, 5\}$, and the BMSWebView1 database with $minsupport = 0.5\%$ and $maxspan = \{1, 2, 3, 4, 5\}$. We use an EISP-Miner algorithm to evaluate all the proposed algorithms (C. S. Wang & Lee, 2009), and add a check constraint to it, with this approach called the Post-EISPMiner algorithm.

Based on the experimental results in Figure 4.5-Figure **4.9**, we can see that the DBV-ISPMIC-IMPROVING algorithm runs faster than the other two algorithms, Post-EISPMiner and DBV-ISPMIC. In Figure 4.5, we compare the runtime of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the Gazelle dataset. When the value of *maxspan* is increasing, the running time of all three algorithms increases relatively evenly.

Figure 4.6-Figure **4.9** show the results for the C6T5S4I4N1kD1k, C6T5S4I4N1kD10k, BIKE and BMSWebView1 datasets. It is clear that the runtimes for DBV-ISPMIC-IMPROVING and DBV-ISPMIC are much better than that of Post-EISPMiner.

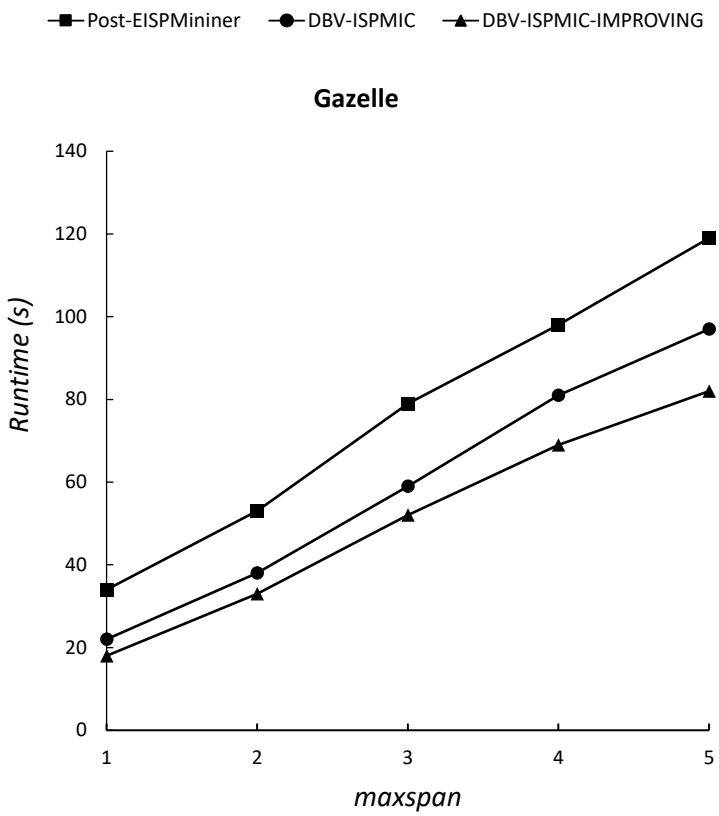


Figure 4.5. Execution times of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the Gazelle dataset.

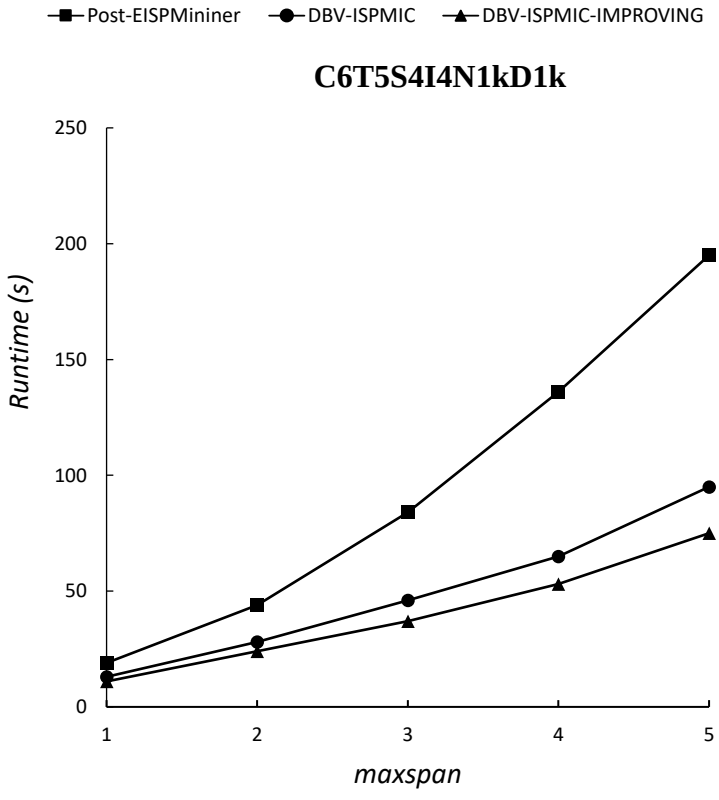


Figure 4.6. Execution times of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the C6T5S4I4N1kD1k dataset.

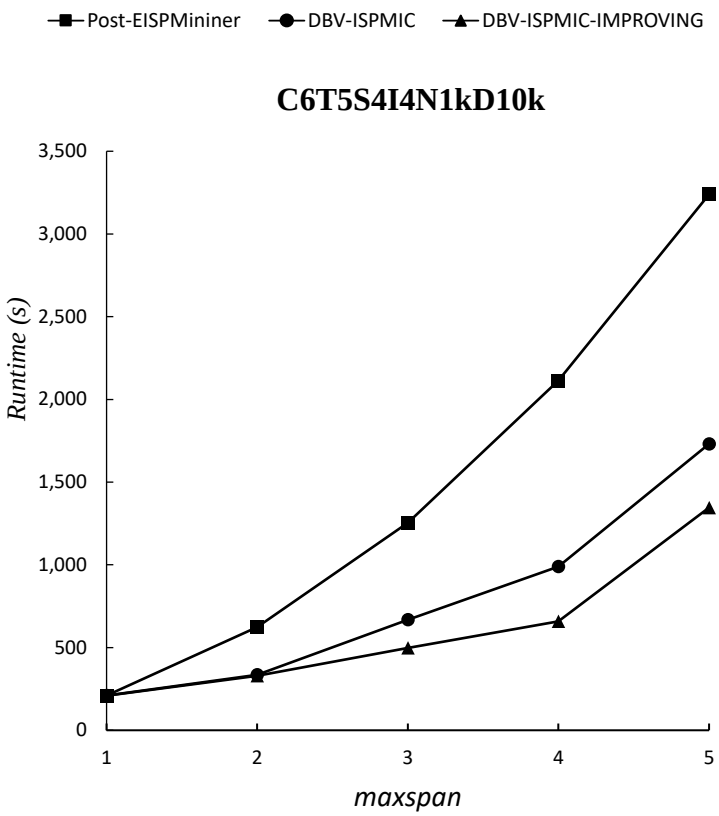


Figure 4.7. Execution times of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the C6T5S4I4N1kD10k dataset.

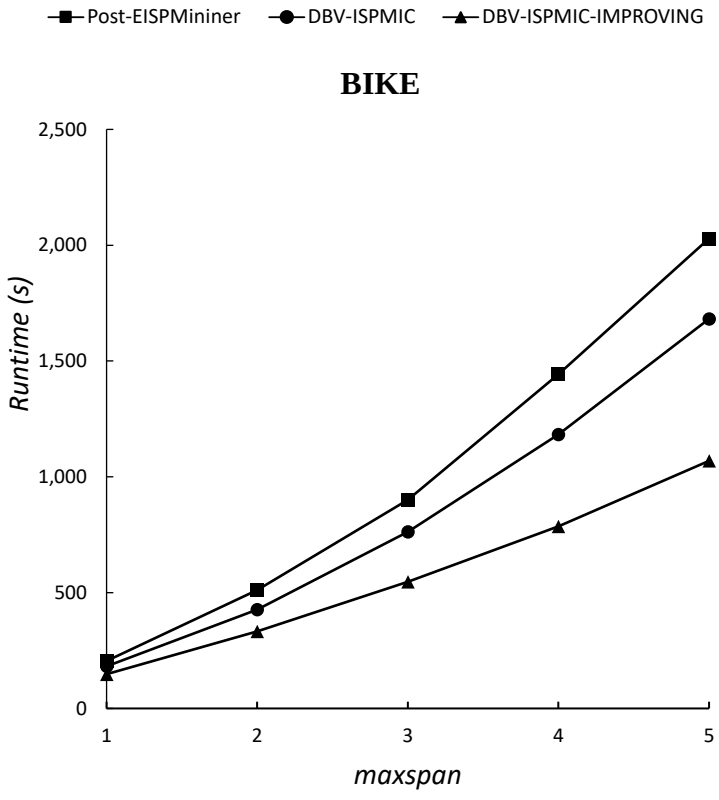


Figure 4.8. Execution times of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the BIKE dataset.

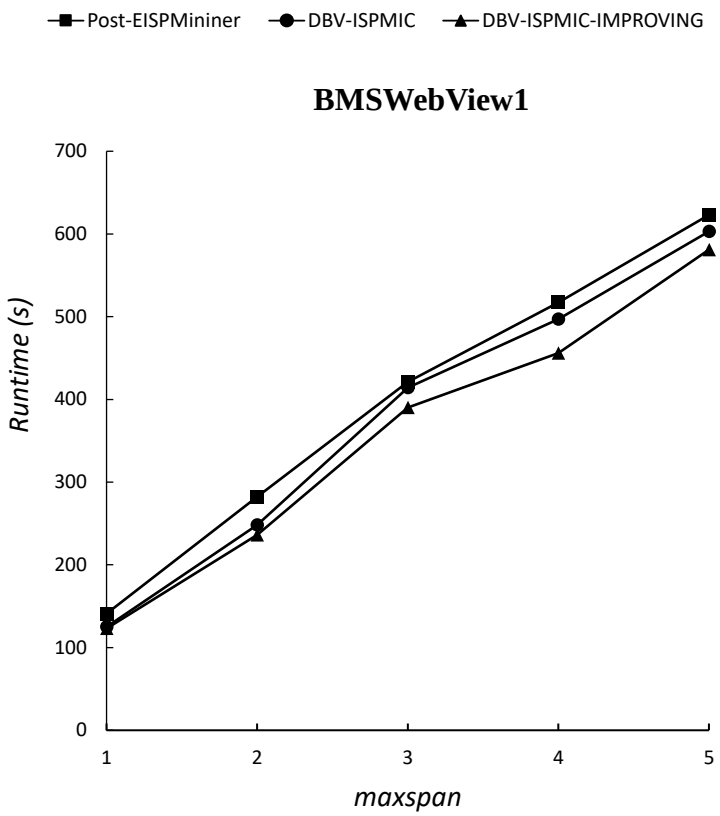


Figure 4.9. Execution times of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the BMSWebView1 dataset.

4.4.2 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 Figure 4.10-Figure **4.13**, 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.

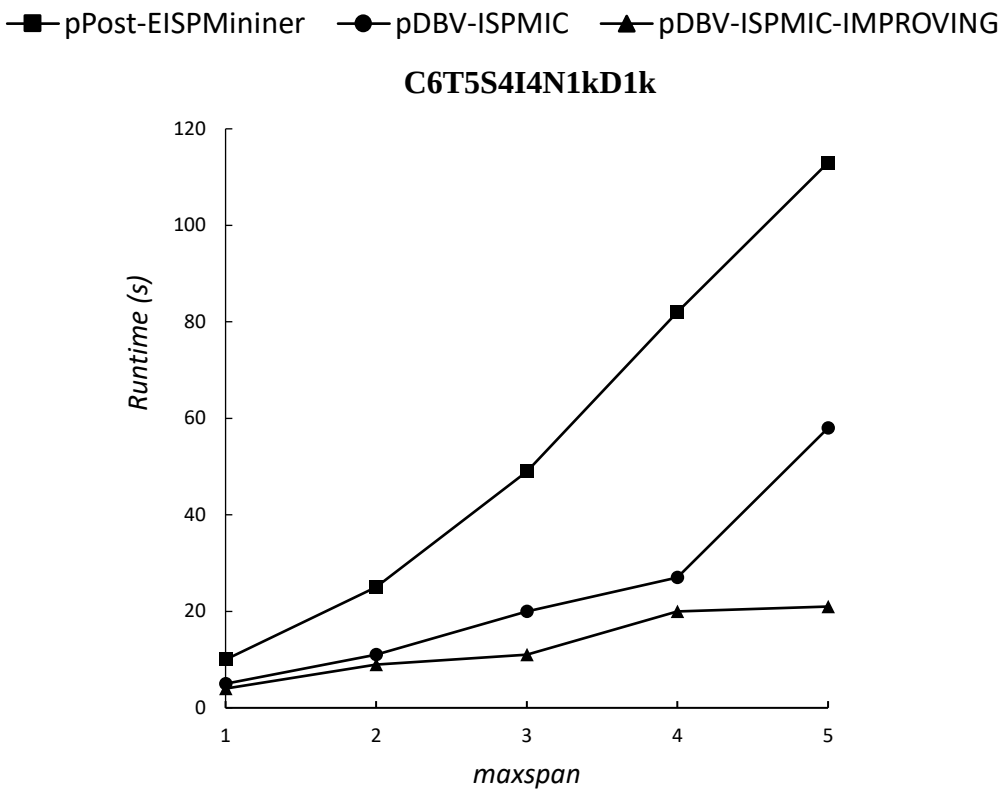


Figure 4.10. Execution times in a parallel evaluation of pPost-EISPMiner, *p*DBV-ISPMIC and *p*DBV-ISPMIC-IMPROVING for the C6T5S4I4N1kD1k dataset.

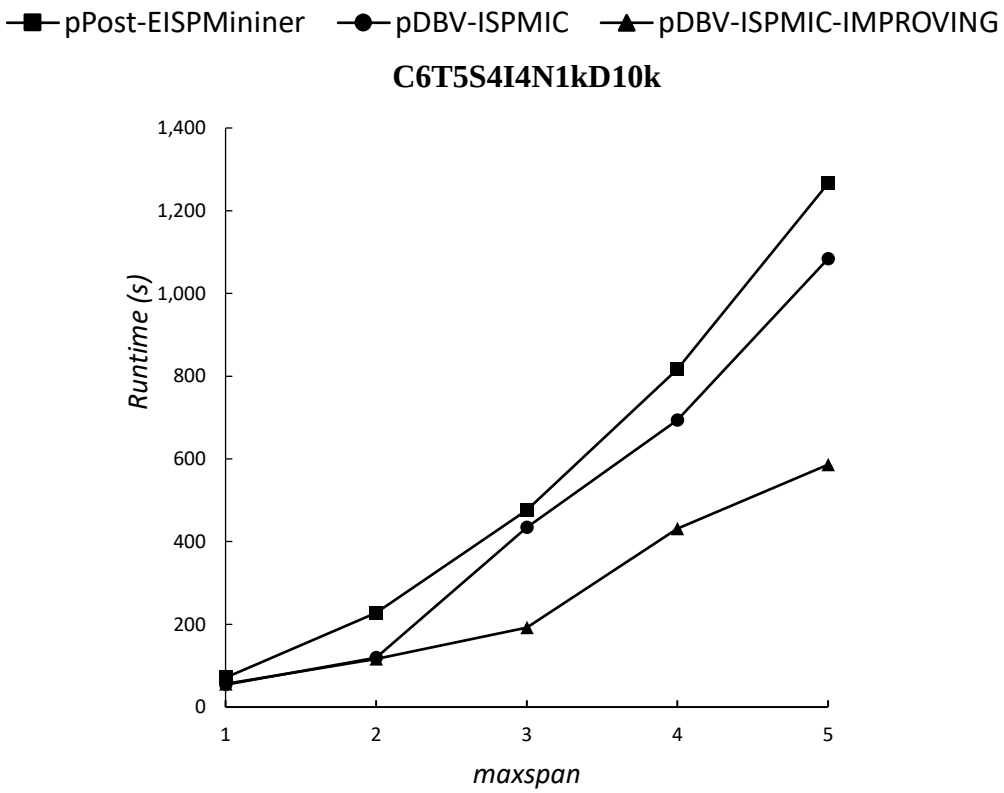


Figure 4.11. Execution time in a parallel evaluation of pPost-EISPMiner, *p*DBV-ISPMIC and *p*DBV-ISPMIC-IMPROVING for the C6T5S4I4N1kD10k dataset.

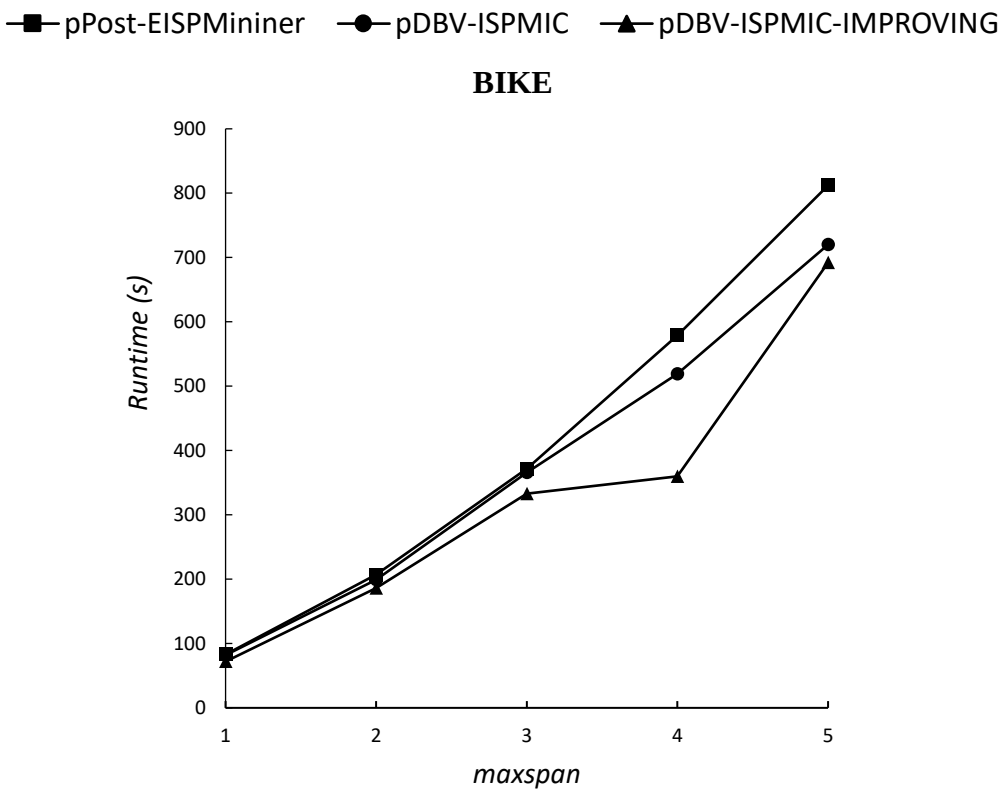


Figure 4.12. Execution times in a parallel evaluation of pPost-EISPMiner, *p*DBV-ISPMIC and *p*DBV-ISPMIC-IMPROVING for the BIKE dataset.

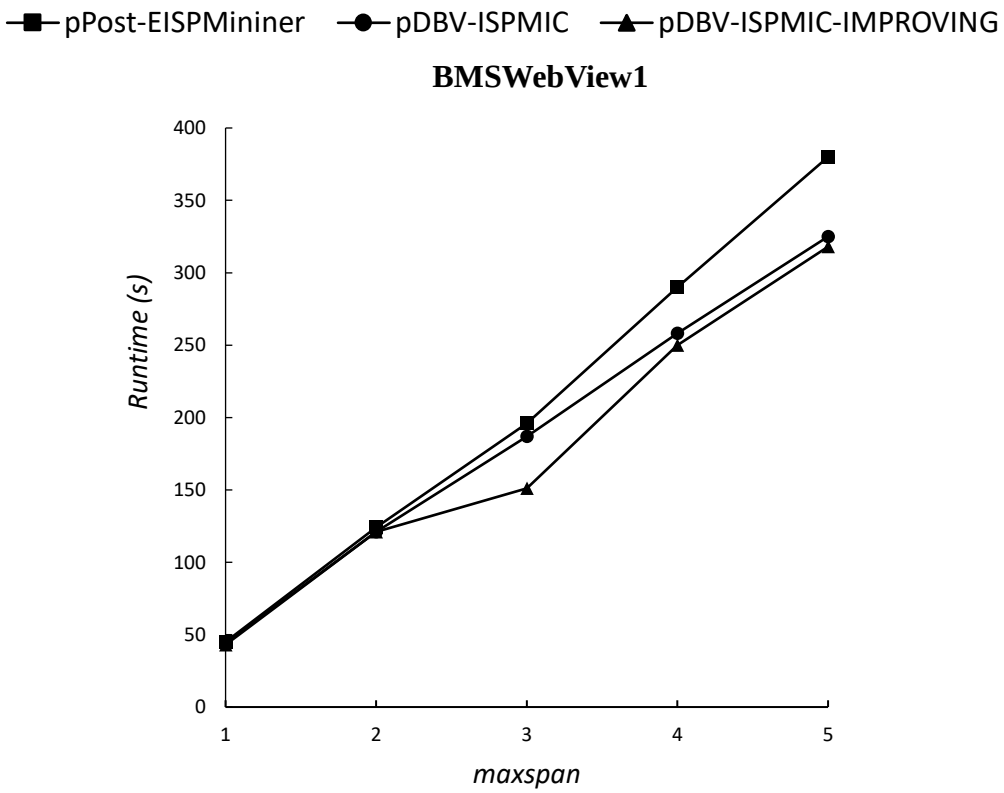


Figure 4.13. Execution times in a parallel evaluation of pPost-EISPMiner, *p*DBV-ISPMIC and *p*DBV-ISPMIC-IMPROVING for the BMSWebView1 dataset.

4.4.3 Memory Usage

Figure 4.14-Figure **4.18** 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.

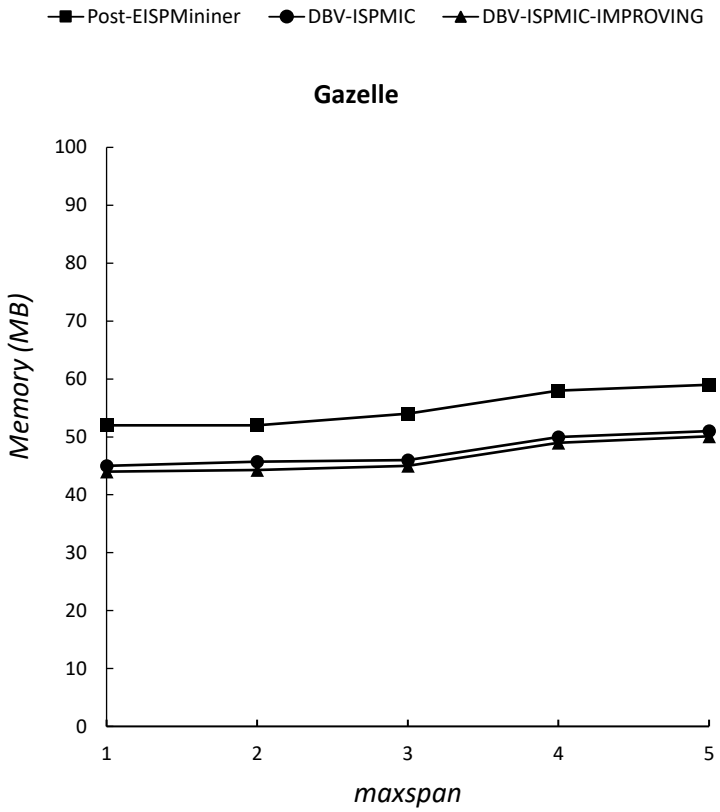


Figure 4.14. Memory usage of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the Gazelle dataset.

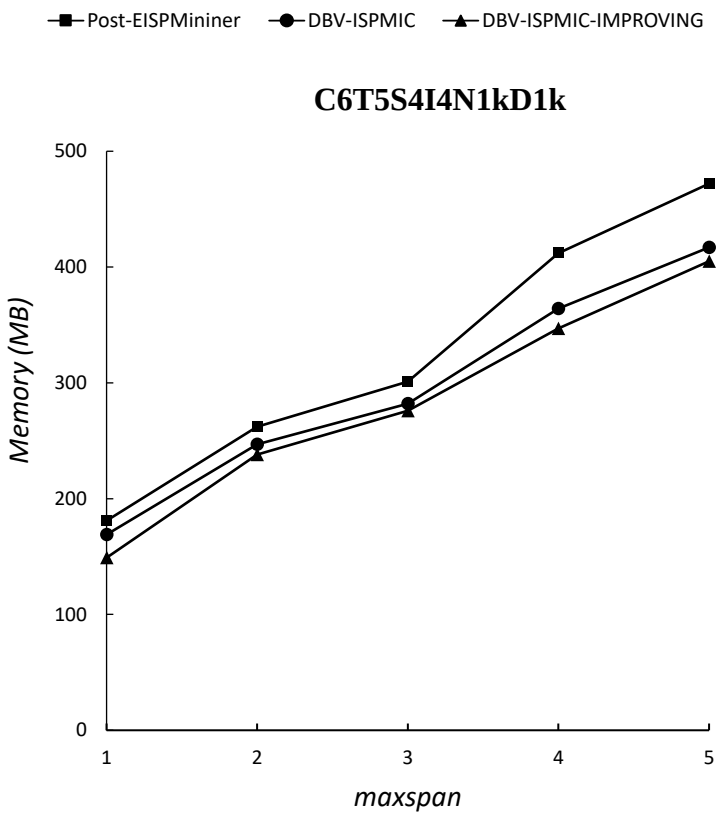


Figure 4.15. Memory usage of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the C6T5S4I4N1kD1k dataset.

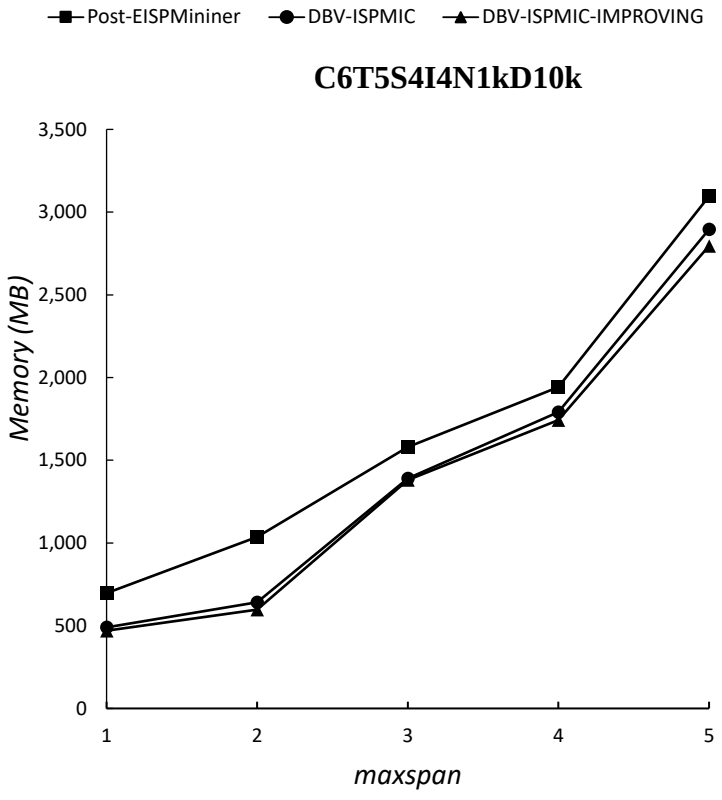


Figure 4.16. Memory usage of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the C6T5S4I4N1kD10k dataset.

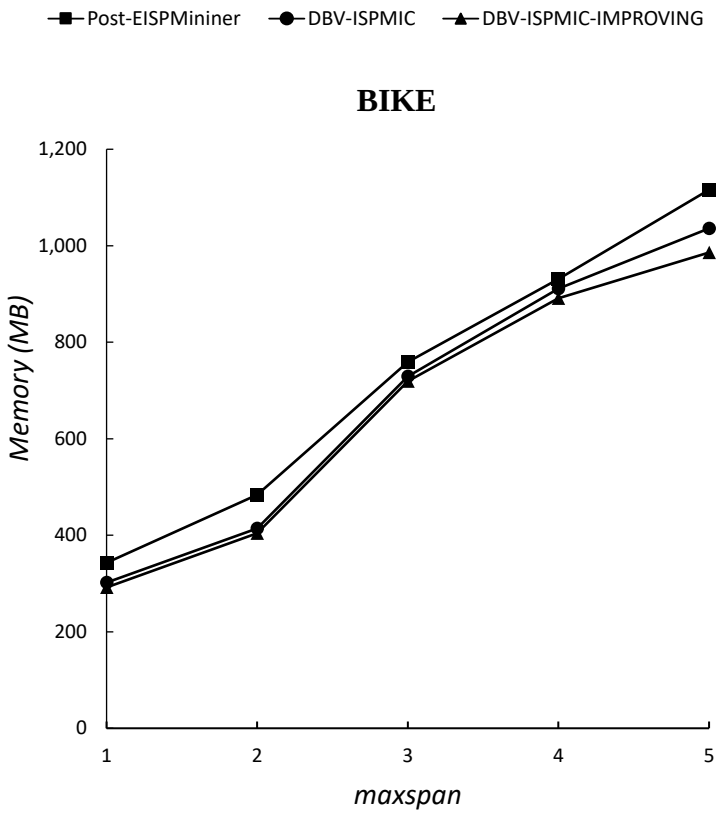


Figure 4.17. Memory usage of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the BIKE dataset.

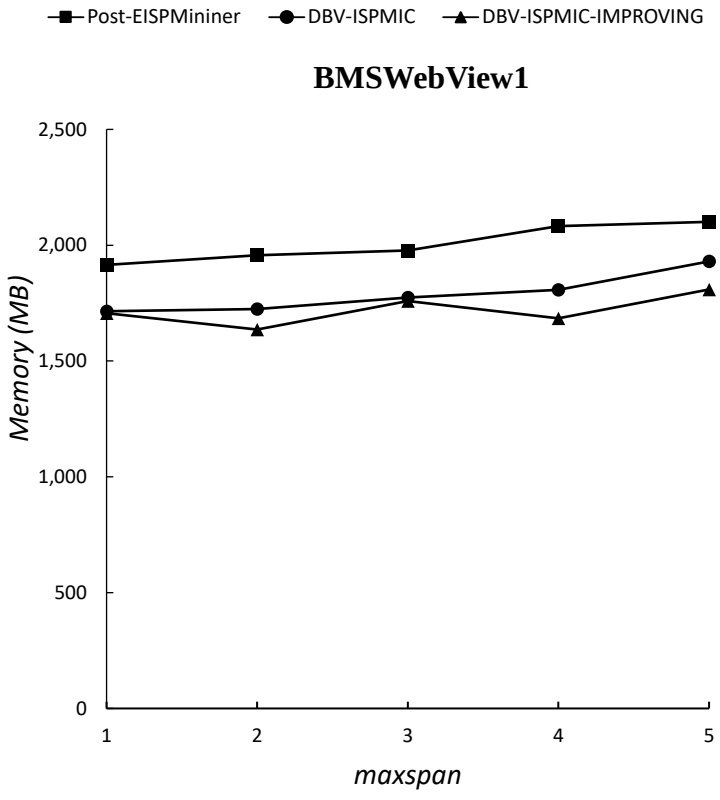


Figure 4.18. Memory usage of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the BMSWebView1 dataset.

4.4.4 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 ($minsupport = 3\%$) and C6T5S4I4N1kD1k database ($minsupport = 0.8\%$) 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 (Figure 4.19-Figure **4.20**).

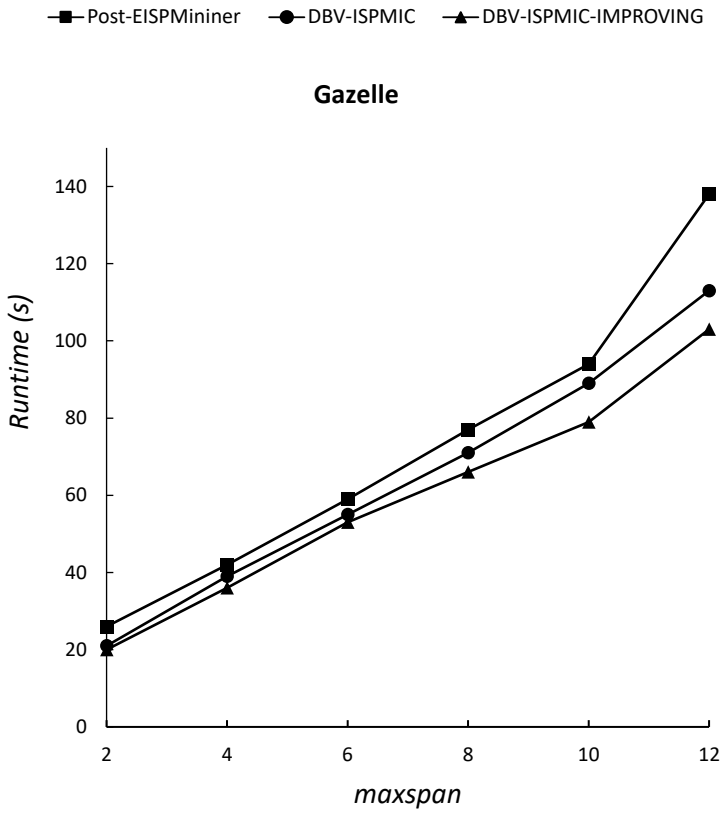


Figure 4.19. Execution times of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the Gazelle dataset, with *maxspan* from 2 to 12.

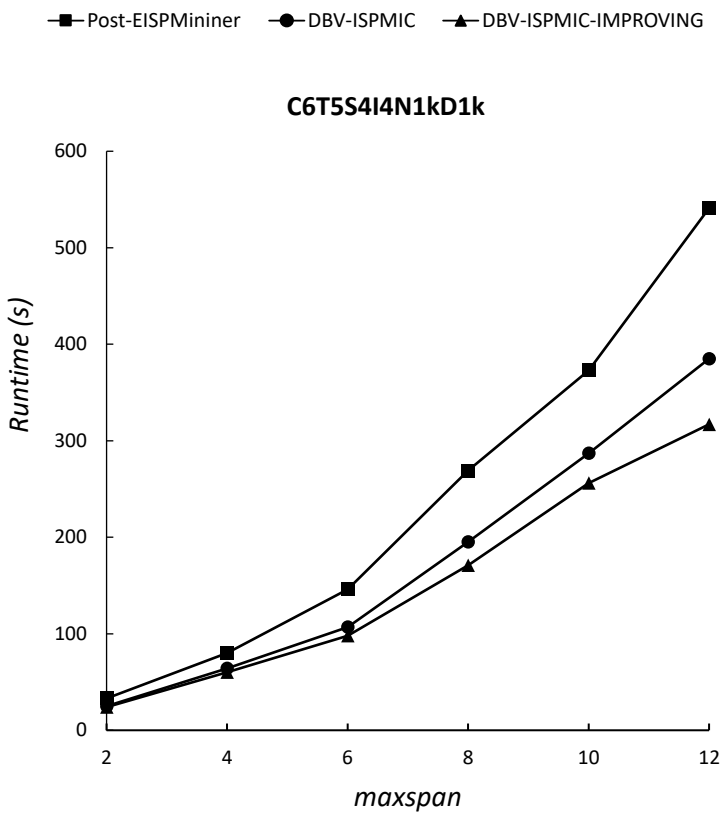


Figure 4.20. Execution times of Post-EISPMiner, DBV-ISPMIC and DBV-ISPMIC-IMPROVING for the C6T5S4I4N1kD1k dataset, with *maxspan* from 2 to 12.

4.5 Summary

In this chapter, we introduced an algorithm, named DBV-ISPMIC, to solve the problem of mining inter-sequence patterns with itemset constraints. This algorithm 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 (Lin et al., 2020b; Gan et al., 2020, 2021a, 2021b; Truong et al., 2021; Wu et al., 2021; Chun-wei Lin et al., 2021), and we will study how to mine high utility inter-sequences and high utility inter-sequences with constraints.

CHAPTER 5: CONCLUSIONS AND FUTURE WORK

5.1 Conclusion

The thesis addresses the challenge of mining inter-sequence patterns in extensive sequential databases, a topic that has gained attention in recent data mining research. The vast search space and enormous data volume have posed difficulties for serial algorithms to extract frequent patterns. Addressing these challenges is crucial for the inter-sequence pattern mining problem.

The thesis contributions can be summarized as follows:

Firstly, we propose the DBV-ISPMIC algorithm to tackle the inter-sequence pattern mining problem with itemset constraints. This algorithm employs the DBV-PatternList data structure and the ISP-Tree tree structure. The DBV-PatternList uses the bit-vector data type to store item or pattern information. The stored information for each candidate includes its index in a transaction and the index of transactions containing the candidate in the sequential database. If the primitive bit-vector data type were used, the candidate's information would have many bits with a value of 0, indicating the item's absence in the transaction. To address this, the DBV-PatternList data type identifies the first transaction index containing the pattern and stores it, allowing the DBV-PatternList data structure to eliminate the 0 bits. This process not only aids data compression but also reduces memory usage for storing candidate information.

Moreover, the DBV-PatternList data structure enables quick candidate support calculation. The candidate support value is determined by counting the bits with a value of 1, which is easily computed using the bit-vector function due to the data structure's bit-vector nature. As the inter-sequence pattern mining problem generates a large number of candidates, we propose a clause to expedite the itemset condition check for candidates, reducing the algorithm's time consumption.

Secondly, we introduce an optimal processing method for the ISP-Tree. In the DBV-ISPMIC algorithm, each tree branch is processed independently. As a result, the DBV-ISPMIC algorithm's processing time is the cumulative processing time across all branches. This insight led us to develop the *p*DBV-ISPMIC algorithm, an extension of the DBV-ISPMIC algorithm. The *p*DBV-ISPMIC algorithm leverages parallel processing to simultaneously process ISP-Tree branches, enabling concurrent processing and thus optimizing and accelerating the *p*DBV-ISPMIC algorithm's runtime.

Thirdly, we propose an algorithm called ISP-PI (Inter-Sequence Pattern mining based on Pseudo-Index), which applies the pseudo-IDList data structure to inter-sequence

pattern mining. Prior inter-sequence pattern mining algorithms were limited by the need to store all candidate information generated from sequence, itemset, and inter extensions. The ISP-PI algorithm identifies and addresses these limitations. By using the pseudo-IDList data structure, the candidate's information can be retrieved through its 1-pattern. Consequently, instead of storing all pattern information, the pseudo-IDList data structure only needs to store the indexes representing the candidate's position relative to the 1-pattern. This approach enables the pseudo-IDList data structure to store less data while still ensuring complete access to the candidate's information, resulting in lower memory usage for the ISP-PI algorithm compared to previous intersequence pattern mining algorithms.

Next, we enhance the ISP-PI algorithm by introducing a candidate pruning method called ISP-IC (Inter-Sequence Pattern mining with Index intersection Checking). The ISP-PI algorithm employs a candidate generation method based on the SPADE algorithm. Thus, the pruning method operates as follows: when candidates are generated through the sequence expansion method, if any candidate fails to meet the support condition, the remaining candidates will also be unsatisfactory. If a candidate does not fulfill the above lemma, the ISP-PI algorithm will discard the candidate without calculating its support, optimizing the algorithm's runtime.

To further refine the ISP-IC algorithm, we propose a lemma that eliminates candidates not meeting the support condition based on information from the frequent patterns that generate them. Since the pseudo-IDList data structure utilizes bit-vectors to store location information of transactions containing common patterns, we employ the bit-intersection method to pre-check the two frequent patterns. If the result is unsatisfactory, the ISP-IC algorithm will not generate candidates from the two frequent patterns.

Lastly, we verify the correctness and effectiveness of the proposed algorithms using test sets of databases. The evaluation compares the algorithm's runtime and memory usage.

5.2 Limitations

Although the proposed algorithms demonstrate improved results in terms of mining time and memory usage compared to existing methods for intersequence pattern mining, this thesis still reveals several limitations:

1. Limited scalability: The proposed algorithms have not been tested on extremely large datasets, only being evaluated on datasets ranging from tens of thousands to one million data points. This limitation prevents the assessment of algorithm performance on large-scale real-world databases.

2. Limited applicability to sequential databases: A significant limitation of the proposed algorithms is that they were designed and tested specifically for sequential databases. This constraint could potentially hinder the algorithm's ability to handle other types of databases, such as graph-based, time-series, or multimedia databases.
3. Limited maxspan evaluation: The algorithms have only been tested within a specific maxspan range $(0 < maxspan < 13)$, leaving their performance for larger maxspan values unexplored. A larger maxspan value may generate more candidates but could also yield more useful knowledge from the databases.
4. Lack of testing on advanced computing platforms: The proposed algorithms have only been executed on personal computers and not on more advanced systems, such as supercomputers, parallel computing systems, or cloud computing platforms. Testing on these platforms would provide a more accurate evaluation of the proposed algorithms' improvements.
5. Limited scope of constraints: The thesis focuses on intersequence pattern mining with itemset constraints but does not explore other constraints. Investigating and applying other types of constraints could help optimize the extraction of knowledge from the data.
6. Updated databases: The scope of the thesis solely encompasses statistical databases with fixed sizes and a static number of transactions. However, in reality, databases are regularly updated over time, which poses a significant challenge for the effective application of the proposed algorithms in practical scenarios.

5.3 Future Works

Future research will concentrate on addressing the identified limitations and developing new approaches for the inter-sequence pattern mining problem as well as the sequential pattern mining problem. Some potential directions for future research include:

1. Scalability: Perform tests on larger datasets to assess the performance of the proposed algorithms on large-scale, real-world databases, while also adjusting the maxspan value.
2. Applicability: Investigate inter-sequence pattern mining and sequential pattern mining for various types of databases, such as graph-based, time-series, multimedia databases, and data growth over time. Recent studies on this topic include:(Jaber Al, 2021; Motallebi Shabestari & Ahmadi, 2021; Hu et al., 2021; Yang et al., 2021; Wu et al., 2023b, 2023a)
3. Computing platforms: Evaluate the proposed algorithms on more advanced systems, like supercomputers, parallel computing systems, or cloud platforms. Recent results in

this area include: (Huang et al., 2020; Yin et al., 2020a; Chung & Yoo, 2020; Yoo et al., 2020; Z. Liu et al., 2020; Farag et al., 2022; Jayasri & Aruna, 2022)

4. Distributed processing and parallel processing: Implement distributed or parallel processing techniques for the proposed algorithms and apply them to the problems of inter-sequence pattern mining and sequential pattern mining. Recent studies using such methods include: (Yin et al., 2020b; Lekshmy & Rahiman, 2020; Qasem et al., 2021; X. Zhang et al., 2021; C. Zhang et al., 2022)
5. Constraints: Explore the addition of various constraints to the proposed algorithms. Other types of constraints have been examined in studies such as: (De Smedt et al., 2020; Zhou et al., 2021; Truong et al., 2021; Xia et al., 2022; Neykov, 2023)
6. Expansion of research: Beyond the issues of inter-sequence pattern mining and sequential pattern mining, consider other data mining challenges like weighted inter-sequence mining or candidate generation for inter-sequence mining problems. The most recent research on these topics is illustrated by the following results: (Chen et al., 2020; Leon-Alcaide et al., 2020; Lin et al., 2020a; M. Liu et al., 2022; Jazayeri & Yang, 2022; Li et al., 2023)
7. Inter-Sequence pattern mining on updated databases: Enhancing and applying algorithms to databases that undergo updates over time. The algorithms should be capable of accommodating new data additions without necessitating a complete rerun of the entire dataset. (Ren & Zhou, 2006; Price et al., 2022; X. Wang et al., 2022; Huynh et al., 2023; Siddiqui et al., 2023) have conducted prior research on this subject.

By pursuing these future research directions, the proposed algorithms can be enhanced and expanded to better tackle the challenges of alternating pattern mining and offer valuable insights across diverse application areas.

5.4 Publications

[R1]. **Nguyen, A.**, Nguyen, N. T., Nguyen, L. T. T., & Vo, B. (2024). An efficient pruning method for mining inter-sequence patterns based on pseudo-IDList. *Expert Systems with Applications, 238,* 121738. https://doi.org/10.1016/J.ESWA.2023.121738 (Impact Factor: 8.5, Category Quartile: Q1)

[R2]. **Nguyen, A.**, Nguyen, N. T., Nguyen, L. T. T., & Vo, B. (2023). Mining inter-sequence patterns with Itemset constraints. *Applied Intelligence*, *53*(17), 19827–19842. https://doi.org/10.1007/S10489-023-04514-7 (Impact Factor: 5.3, Category Quartile: Q2)

[R3]. Nguyen, T. T. D., Nguyen, L. T. T., **Nguyen, A.**, Yun, U., & Vo, B. (2021). A method for efficient clustering of spatial data in network space. *Journal of Intelligent & Fuzzy Systems*, *40*(6), 11653–11670. https://doi.org/10.3233/JIFS-202806 (Impact Factor: 1.737, Category Quartile: Q4)

[R4]. Huynh, H. M., Nguyen, L. T. T., Vo, B., **Nguyen, A.**, & Tseng, V. S. (2020). Efficient methods for mining weighted clickstream patterns. *Expert Systems with Applications*, *142*, 112993. https://doi.org/10.1016/j.eswa.2019.112993 (Impact Factor: 6.954, Category Quartile: Q1)

[R5]. Nguyen, L. T. T., Nguyen, T. D. D., **Nguyen, A.**, Tran, P.-N., Trinh, C., Huynh, B., & Vo, B. (2020). Efficient Method for Mining High-Utility Itemsets Using High-Average Utility Measure. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *12496 LNAI*, 305–315. https://doi.org/10.1007/978-3-030-63007-2_24 (Core ranking: C)

[R6]. Nguyen, L. T. T., Vo, B., Nguyen, T. N., & **Nguyen, A.** (2019). Mining class association rules on imbalanced class datasets. *Journal of Intelligent and Fuzzy Systems*, *37*(6), 7131–7139. https://doi.org/10.3233/JIFS-179326 (Impact Factor: 1.851, Category Quartile: Q3)

[R7]. Le, T., **Nguyen, A.**, Huynh, B., Vo, B., & Pedrycz, W. (2018). Mining constrained inter-sequence patterns: a novel approach to cope with item constraints. *Applied Intelligence*, *48*(5), 1327–1343. https://doi.org/10.1007/s10489-017-1123-9 (Impact Factor: 2.882, Category Quartile: Q2)

REFERENCES

Agrawal, R., & Srikant, R. (1995). Mining sequential patterns. *Proceedings - International Conference on Data Engineering*, 3–14. https://doi.org/10.1109/ICDE.1995.380415

Ayres, J., Flannick, J., Gehrke, J., & Yiu, T. (2002). Sequential PAttern mining using a bitmap representation. *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining -KDD '02*, 429. https://doi.org/10.1145/775047.775109

Chen, S., Nie, L., Tao, X., Li, Z., & Zhao, L. (2020). Approximation of Probabilistic Maximal Frequent Itemset Mining over Uncertain Sensed Data. *IEEE Access*, *8*, 97529–97539. https://doi.org/10.1109/ACCESS.2020.2997409

Chung, K., & Yoo, H. (2020). Edge computing health model using P2P-based deep neural networks. *Peer-to-Peer Networking and Applications*, *13*(2), 694–703. https://doi.org/10.1007/S12083-019-00738-Y

Chun-wei Lin, J., Yu, P. S., Chun-Wei Lin, J., Djenouri, Y., Srivastava, G., Li, Y., & Yu, P. S. (2021). Scalable Mining of High-Utility Sequential Patterns With Three-Tier MapReduce Model. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *16*(3), 1–26. https://doi.org/10.1145/3487046

De Smedt, J., Deeva, G., & De Weerdt, J. (2020). Mining Behavioral Sequence Constraints for Classification. *IEEE Transactions on Knowledge and Data Engineering*, *32*(6), 1130–1142. https://doi.org/10.1109/TKDE.2019.2897311

Farag, A., Abdelkader, H., & Salem, R. (2022). Parallel graph-based anomaly detection technique for sequential data. *Journal of King Saud University - Computer and Information Sciences*, *34*(1), 1446–1454. https://doi.org/10.1016/J.JKSUCI.2019.09.009

Fournier-Viger, P., Gomariz, A., Campos, M., & Thomas, R. (2014). Fast vertical mining of sequential patterns using co-occurrence information. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8443 LNAI*(PART 1), 40–52. https://doi.org/10.1007/978-3-319-06608-0_4

Fournier-Viger, P., Lin, J. C., Kiran, R. U., Koh, Y. S., & Thomas, R. (2017). *A Survey of Sequential Pattern Mining*. *Data Science and Pattern Recognition* 1(1), 54-77

Gan, W., Lin, J. C. W., Zhang, J., Chao, H. C., Fujita, H., & Yu, P. S. (2020). ProUM: Projection-based utility mining on sequence data. *Information Sciences*, *513*, 222–240. https://doi.org/10.1016/J.INS.2019.10.033

Gan, W., Lin, J. C. W., Zhang, J., Fournier-Viger, P., Chao, H. C., & Yu, P. S. (2021a). Fast Utility Mining on Sequence Data. *IEEE Transactions on Cybernetics*, *51*(2), 487–500. https://doi.org/10.1109/TCYB.2020.2970176

Gan, W., Lin, J. C. W., Zhang, J., Yin, H., Fournier-Viger, P., Chao, H. C., & Yu, P. S. (2021b). Utility Mining Across Multi-Dimensional Sequences. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, *15*(5). https://doi.org/10.1145/3446938

Gouda, K., Hassaan, M., & Zaki, M. J. (2007). PRISM: A prime-encoding approach for frequent sequence mining. *Proceedings - IEEE International Conference on Data Mining, ICDM*, 487–492. https://doi.org/10.1109/ICDM.2007.33

Gouda, K., Hassaan, M., & Zaki, M. J. (2010). Prism: An effective approach for frequent sequence mining via prime-block encoding. *Journal of Computer and System Sciences*, *76*(1), 88–102. https://doi.org/10.1016/J.JCSS.2009.05.008

Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., & Hsu, M.-C. (2000). FreeSpan: frequent pattern-projected sequential pattern mining. *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '00*, 355–359. https://doi.org/10.1145/347090.347167

Hu, Y., Zhan, P., Xu, Y., Zhao, J., Li, Y., & Li, X. (2021). Temporal representation learning for time series classification. *Neural Computing and Applications*, *33*(8), 3169–3182. https://doi.org/10.1007/S00521-020-05179-W

Huang, Y., Cheng, Z., Zhou, Q., Xiang, Y., & Zhao, R. (2020). Data mining algorithm for cloud network information based on artificial intelligence decision mechanism. *IEEE Access*, *8*, 53394–53407. https://doi.org/10.1109/ACCESS.2020.2981632

Huynh, H. M., Nguyen, L. T. T., Pham, N. N., Oplatková, Z. K., Yun, U., & Vo, B. (2022). An efficient method for mining sequential patterns with indices. *Knowledge-Based Systems*, *239*. https://doi.org/10.1016/J.KNOSYS.2021.107946

Huynh, H. M., Nguyen, L. T. T., Vo, B., Nguyen, A., & Tseng, V. S. (2020a). Efficient methods for mining weighted clickstream patterns. *Expert Systems with Applications*, *142*, 112993. https://doi.org/10.1016/j.eswa.2019.112993

Huynh, H. M., Nguyen, L. T. T., Vo, B., Yun, U., Oplatková, Z. K., & Hong, T. P. (2020b). Efficient algorithms for mining clickstream patterns using pseudo-IDLists.

Future Generation Computer Systems, *107*, 18–30. https://doi.org/10.1016/j.future.2020.01.034

Huynh, H. M., Vo, B., Oplatková, Z. K., & Pedrycz, W. (2023). An Approach for Incremental Mining of Clickstream Patterns as a Service Application. *IEEE Transactions on Services Computing*, 1–14. https://doi.org/10.1109/TSC.2023.3294945

Jaber Al, A. N. (2021). Efficient Visualization Framework for Real-Time Monitoring Network Traffic of High-Speed Networks. *Proceedings - 2021 IEEE International Conference on Big Data, Big Data 2021*, 5839–5842. https://doi.org/10.1109/BIGDATA52589.2021.9671915

Jayasri, N. P., & Aruna, R. (2022). Big data analytics in health care by data mining and classification techniques. *ICT Express*, *8*(2), 250–257. https://doi.org/10.1016/J.ICTE.2021.07.001

Jazayeri, A., & Yang, C. C. (2022). Frequent Subgraph Mining Algorithms in Static and Temporal Graph-Transaction Settings: A Survey. *IEEE Transactions on Big Data*, *8*(6), 1443–1462. https://doi.org/10.1109/TBDATA.2021.3072001

Le, B., Tran, M. T., & Vo, B. (2015). Mining frequent closed inter-sequence patterns efficiently using dynamic bit vectors. *Applied Intelligence*, *43*(1), 74–84. https://doi.org/10.1007/s10489-014-0630-1

Le, T., Nguyen, A., Huynh, B., Vo, B., & Pedrycz, W. (2018). Mining constrained inter-sequence patterns: a novel approach to cope with item constraints. *Applied Intelligence*, *48*(5), 1327–1343. https://doi.org/10.1007/s10489-017-1123-9

Lekshmy, P. L., & Rahiman, M. A. (2020). A sanitization approach for privacy preserving data mining on social distributed environment. *Journal of Ambient Intelligence and Humanized Computing*, *11*(7), 2761–2777. https://doi.org/10.1007/S12652-019-01335-W

Leon-Alcaide, P., Rodriguez-Benitez, L., Castillo-Herrera, E., Moreno-Garcia, J., & Jimenez-Linares, L. (2020). An evolutionary approach for efficient prototyping of large time series datasets. *Information Sciences*, *511*, 74–93. https://doi.org/10.1016/J.INS.2019.09.044

Li, Y., Zhang, C., Li, J., Song, W., Qi, Z., Wu, Y., & Wu, X. (2023). MCoR-Miner: Maximal Co-Occurrence Nonoverlapping Sequential Rule Mining. *IEEE*

Transactions on Knowledge and Data Engineering. https://doi.org/10.1109/TKDE.2023.3241213

Lin, J. C. W., Li, T., Pirouz, M., Zhang, J., & Fournier-Viger, P. (2020a). High average-utility sequential pattern mining based on uncertain databases. *Knowledge and Information Systems*, *62*(3), 1199–1228. https://doi.org/10.1007/S10115-019-01385-8

Lin, J. C. W., Li, Y., Fournier-Viger, P., Djenouri, Y., & Zhang, J. (2020b). Efficient Chain Structure for High-Utility Sequential Pattern Mining. *IEEE Access*, *8*, 40714–40722. https://doi.org/10.1109/ACCESS.2020.2976662

Liu, M., Yang, Z., Guo, Y., Jiang, J., & Yang, K. (2022). MICAR: nonlinear association rule mining based on maximal information coefficient. *Knowledge and Information Systems*, *64*(11), 3017–3042. https://doi.org/10.1007/S10115-022-01730-4

Liu, Z., Shi, X., He, L., Yu, D., Jin, H., Yu, C., Dai, H., & Feng, Z. (2020). A parameter-level parallel optimization algorithm for large-scale spatio-temporal data mining. *Distributed and Parallel Databases*, *38*(3), 739–765. https://doi.org/10.1007/S10619-020-07287-X

Motallebi Shabestari, M., & Ahmadi, A. (2021). Identifying the relationship between human self-esteem and general health using data mining. *26th International Computer Conference, Computer Society of Iran, CSICC 2021*. https://doi.org/10.1109/CSICC52343.2021.9420612

Neykov, M. (2023). On the Minimax Rate of the Gaussian Sequence Model Under Bounded Convex Constraints. *IEEE Transactions on Information Theory*, *69*(2), 1244–1260. https://doi.org/10.1109/TIT.2022.3213141

Nguyen, A., Nguyen, N. T., Nguyen, L. T. T., & Vo, B. (2023). Mining inter-sequence patterns with Itemset constraints. *Applied Intelligence*, 1–16. https://doi.org/10.1007/S10489-023-04514-7/METRICS

Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., & Hsu, M. C. (2001). PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. *Proceedings - International Conference on Data Engineering*, 215–224. https://doi.org/10.1109/icde.2001.914830

Price, S., Tombeur, S. D. S., Hudson, A., Sathiyamoorthy, N. K., Smyth, P., Singh, A., Peccianti, M., Baroncelli, E., Essaghir, A., Ferlenghi, I., Phogat, S. K., & Singh, G. (2022). TMQuery: A database of precomputed template modeling scores for

assessment of protein structural similarity. *Bioinformatics*, *38*(7), 2062–2063. https://doi.org/10.1093/BIOINFORMATICS/BTAC044

Qasem, M. H., Obeid, N., Hudaib, A., Almaiah, M. A., Al-Zahrani, A., & Al-Khasawneh, A. (2021). Multi-Agent System Combined with Distributed Data Mining for Mutual Collaboration Classification. *IEEE Access*, *9*, 70531–70547. https://doi.org/10.1109/ACCESS.2021.3074125

R. Agrawal and R. Srikant. (1994). Fast Algorithms for Mining Association Rules in Large Databases. *Proceedings of the 20th International Conference on Very Large Data Bases*, 487–499. https://dl.acm.org/doi/10.5555/645920.672836

Ren, J. D., & Zhou, X. L. (2006). An efficient algorithm for incremental mining of sequential patterns. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3930 LNAI*, 179–188. https://doi.org/10.1007/11739685_19/COVER

Siddiqui, S. A., Ahmad, A., & Fatima, N. (2023). IoT-based disease prediction using machine learning. *Computers and Electrical Engineering*, *108*. https://doi.org/10.1016/J.COMPELECENG.2023.108675

Srikant, R., & Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *1057 LNCS*, 3–17. https://doi.org/10.1007/BFB0014140/COVER

Truong, T., Duong, H., Le, B., Fournier-Viger, P., Yun, U., & Fujita, H. (2021). Efficient algorithms for mining frequent high utility sequences with constraints. *Information Sciences*, *568*, 239–264. https://doi.org/10.1016/J.INS.2021.01.060

Van, T., & Le, B. (2021). Mining sequential rules with itemset constraints. *Applied Intelligence*, 1–13. https://doi.org/10.1007/s10489-020-02153-w

Van, T., Vo, B., & Le, B. (2018a). Mining sequential patterns with itemset constraints. *Knowledge and Information Systems*, *57*(2), 311–330. https://doi.org/10.1007/s10115-018-1161-6

Van, T., Yoshitaka, A., & Le, B. (2018b). Mining web access patterns with super-pattern constraint. *Applied Intelligence*, *48*(11), 3902–3914. https://doi.org/10.1007/S10489-018-1182-6/METRICS

Vo, B., Tran, M. T., Hong, T. P., Nguyen, H., & Le, B. (2012). A dynamic bit-vector approach for efficiently mining inter-sequence patterns. *Proceedings - 3rd*

International Conference on Innovations in Bio-Inspired Computing and Applications, IBICA 2012, 51–56. https://doi.org/10.1109/IBICA.2012.31

Wang, C. S., & Lee, A. J. T. (2009). Mining inter-sequence patterns. *Expert Systems with Applications*, *36*(4), 8649–8658. https://doi.org/10.1016/j.eswa.2008.10.008

Wang, X., Zhou, X., Yan, Q., Liao, S., Tang, W., Xu, P., Gao, Y., Li, Q., Dou, Z., Yang, W., Huang, B., Li, J., & Zhang, Z. (2022). LLPSDB v2.0: An updated database of proteins undergoing liquid-liquid phase separation in vitro. *Bioinformatics*, *38*(7), 2010–2014. https://doi.org/10.1093/BIOINFORMATICS/BTAC026

Wu, Y., Geng, M., Li, Y., Guo, L., Li, Z., Fournier-Viger, P., Zhu, X., & Wu, X. (2021). HANP-Miner: High average utility nonoverlapping sequential pattern mining. *Knowledge-Based Systems*, *229*, 107361. https://doi.org/10.1016/J.KNOSYS.2021.107361

Wu, Y., Hu, Q., Li, Y., Guo, L., Zhu, X., & Wu, X. (2023a). OPP-Miner: Order-Preserving Sequential Pattern Mining for Time Series. *IEEE Transactions on Cybernetics*, *53*(5), 3288–3300. https://doi.org/10.1109/TCYB.2022.3169327

Wu, Y., Zhao, X., Li, Y., Guo, L., Zhu, X., Fournier-Viger, P., & Wu, X. (2023b). OPR-Miner: Order-preserving rule mining for time series. *IEEE Transactions on Knowledge and Data Engineering*, 1–15. https://doi.org/10.1109/TKDE.2022.3224963

Xia, M., Yang, H., Huang, Y., Qu, Y., Guo, Y., Zhou, G., Zhang, F., & Wang, Y. (2022). AwCPM-Net: A Collaborative Constraint GAN for 3D Coronary Artery Reconstruction in Intravascular Ultrasound Sequences. *IEEE Journal of Biomedical and Health Informatics*, *26*(7), 3047–3058. https://doi.org/10.1109/JBHI.2022.3147888

Yang, M., Qu, Q., Shen, Y., Zhao, Z., Chen, X., & Li, C. (2021). An Effective Hybrid Learning Model for Real-Time Event Summarization. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(10), 4419–4431. https://doi.org/10.1109/TNNLS.2020.3017747

Yin, C., Pan, C., & Zhang, P. (2020a). Deep neural network combined with MapReduce for abnormal data mining and detection in cloud storage. *Journal of Ambient Intelligence and Humanized Computing*. https://doi.org/10.1007/S12652-020-01996-Y

Yin, C., Pan, C., & Zhang, P. (2020b). Deep neural network combined with MapReduce for abnormal data mining and detection in cloud storage. *Journal of Ambient Intelligence and Humanized Computing*. https://doi.org/10.1007/S12652-020-01996-Y

Yoo, J. S., Boulware, D., & Kimmey, D. (2020). Parallel co-location mining with MapReduce and NoSQL systems. *Knowledge and Information Systems*, *62*(4), 1433–1463. https://doi.org/10.1007/S10115-019-01381-Y

Zaki, M. J. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, *42*(1–2), 31–60. https://doi.org/10.1023/A:1007652502315

Zhang, C., Yang, Y., Zhou, W., & Zhang, S. (2022). Distributed Bayesian Matrix Decomposition for Big Data Mining and Clustering. *IEEE Transactions on Knowledge and Data Engineering*, *34*(8), 3701–3713. https://doi.org/10.1109/TKDE.2020.3029582

Zhang, X., Zhu, X., Bao, W., Yang, L. T., Wang, J., Yan, H., & Chen, H. (2021). Distributed Learning on Mobile Devices: A New Approach to Data Mining in the Internet of Things. *IEEE Internet of Things Journal*, *8*(13), 10264–10279. https://doi.org/10.1109/JIOT.2020.3030783

Zhou, P., Zhou, G., Wu, D., & Fei, M. (2021). Detecting multi-stage attacks using sequence-to-sequence model. *Computers and Security*, *105*. https://doi.org/10.1016/J.COSE.2021.102203