People's Democratic Republic of Algeria
Ministery of Higher Education and Scientific Research
Mohamed El Bachir El Ibrahimi University of Borj Bou Arréridj
Faculty of Mathematics and Computer Science
Department of Computer Science

DISSERTATION

Presented in fulfillment of the requirements of obtaining the degree

Master in Computer Science

Specialty: Technologies of information and communication

THEME

Discovering parallel episodes in event sequences: Enhanced analysis of web page navigation patterns

Presented by:

DALI MOHAMED Imad Eddine

TABET Ishak

Publicly defended on: 11/06/2025

In front of the jury composed of:

President: LAIFA Meriem

Examiner: MAACHE Salah

Supervisor: MOUSSAOUI Boubakeur

co-supervisor: OUAREM Oualid

2024/2025

Dedication

I dedicate this work to my beloved family, whose encouragement and support have been my greatest motivator throughout this journey.

To my parents, because of their unlimited love, sacrifices, and patience that made me reach this target.

A special thank you to my binome, Imad, for his collaboration, commitment, and constant support throughout this project.

To my dearest friends Riad, Radwan, Antar, Islam, Saleh, Mohammed and Saif Eddine, for encouragement, companionship, and for never leaving me during periods of adversity and doubt.

It is a testament to your belief in me, and I am most grateful to have you by my side.

Ishak Tabet

Dedication

To my beloved family for your endless love, sacrifices, and belief in me, even when I doubted myself.

To my parents thank you for your endless love, and sacrifices. Everything I am is because of you.

To my dear friend and thesis partner, Ishak thank you for walking this journey with me, side by side.

To my loved ones thank you for giving me motivation, support, and standing by me.

And my cherished friends, Riyad, Islem, Hakim, Anis and Saleh your support, encouragement, and presence meant more than words can express.

This work is for all of you.

DALI MOHAMED Imad Eddine

Acknowledgment

We would be happy to take this opportunity to thank our deepest gratitude to all who have been assisting us in writing this thesis.

To our Dr. Moussaoui Boubaker, our highest regard for his professional guidance, judicious counsel, and unwavering encouragement that have played the most significant role in shaping this research. His enthusiastic dedication and imaginative inputs heavily added to the worth of our efforts.

We are also profoundly indebted to Professor Ouarem Oualid for the generous support and quality suggestions. His professional knowledge and generous offer to assist us every step of the way played a crucial role in successfully conducting this project. We are genuinely grateful for his patience and the time that he took in guiding us.

And lastly, we extend our warmest gratitude to our loved ones and friends for their unending love, support, and understanding, which encouraged us to hold out

Abstract

In the last decades and the growth of digital devices, the analysis of user navigation patterns on websites presents a significant challenge for organizations aiming to enhance their digital strategies. This thesis focuses on the field of episode mining and introduces our algorithm, developed by modifying one of the episode miming algorithms approach to efficiently handle parallel episodes. By analyzing sequences of web page visits,the proposed version enables the extraction of episode rules. with improved performance and scalability. Experimental results on real world datasets demonstrate that our algorithm offers notable gains in execution time and accuracy compared to traditional methods, making it a valuable tool for predicting user behavior and supporting data driven decision-making in web analytics.

Keywords: Episode Mining, Web User Behavior, Sequential Data, Episode Rules, Data Mining.

Résumé

Au cours des dernières décennies et avec le développement des appareils numériques, l'analyse des habitudes de navigation des utilisateurs sur les sites web représente un défi majeur pour les organisations souhaitant améliorer leurs stratégies numériques. Cette thèse se concentre sur le domaine de l'exploration d'épisodes et présente notre algorithme, développé en modifiant l'une des approches de l'algorithme de mime d'épisodes pour gérer efficacement les épisodes parallèles. En analysant les séquences de visites de pages web, la version proposée permet d'extraire les règles d'épisodes, avec des performances et une évolutivité améliorées. Les résultats expérimentaux sur des ensembles de données réels démontrent que notre algorithme offre des gains notables en termes de temps d'exécution et de précision par rapport aux méthodes traditionnelles, ce qui en fait un outil précieux pour prédire le comportement des utilisateurs et soutenir la prise de décision basée sur les données en analyse web.

Mots-clés : Découverte Des Épisodes, Comportement Des Utilisateurs Web, Données Séquentielles, Règles D'épisode, Exploration De Données.

ملخص

في العقود الأخيرة، ومع تطور الأجهزة الرقمية، يُمثل تحليل أنماط تصفح المستخدمين على مواقع الويب تحديا كبيرا للمؤسسات التي تسعى إلى تحسين استراتيجياتها الرقمية. تُركز هذه الرسالة على مجال استخراج الحلقات، وتُقدم خوارزميتنا، التي طُوّرت بتعديل أحد أساليب خوارزميات محاكاة الحلقات للتعامل بكفاءة مع الحلقات المتوازية. من خلال تحليل تسلسلات زيارات صفحات الويب، يُمكّن الإصدار المُقترح من استخراج قواعد الحلقات، مع تحسين الأداء وقابلية التوسع. تُظهر النتائج التجريبية على مجموعات بيانات واقعية أن خوارزميتنا تُحقق مكاسب ملحوظة في وقت التنفيذ والدقة مُقارنةً بالطرق التقليدية، مما يجعلها أداة قيّمة للتنبؤ بسلوك المستخدم ودعم اتخاذ القرارات القائمة على البيانات في تحليلات الويب.

الكلمات المفتاحية: استخراج الحلقات،، سلوك مستخدمي الويب، البيانات المتسلسلة، قواعد الحلقات، استخراج البيانات.

Table of Contents

Abreviations list **xi**

List of Figures **xiii**

List of Tables **xiv**

List of Algorithms **xv**

1 Data mining **3**

1.1 Introduction 3

1.2 Definition 3

1.3 Objectives 3

1.4 Data Mining Process 5

1.5 Techniques 6

1.6 Domains of Application 8

1.7 Pattern Mining 10

1.7.1 Definition 10

1.7.2 Types of Patterns 10

1.7.3 Applications 11

1.7.4 Challenges 11

1.8 Episode Mining as a Subfield of Pattern Mining 11

1.8.1 Definition 11

1.8.2 Key Characteristics 12

1.8.3 Relationship to Other Pattern Mining Techniques 12

1.9 Conclusion 12

2 Episode Mining **13**

2.1 Introduction . . . 13

2.2 Key Concepts in Episode Mining . . . 13

2.2.1 Event and Event Sequence . . . 14

2.2.2 Occurrence and Minimal occurrence . . . 14

2.2.3 Episode . . . 15

2.3 Frequency Definitions in Episode Mining . . . 16

2.4 Frequent Episode Mining framework . . . 17

2.4.1 Window-based Frequency . . . 17

2.4.2 Occurrence-based Frequency . . . 18

2.4.3 Summary of Frequency Definitions . . . 19

2.5 Search strategies for FEM . . . 19

2.5.1 Breadth-first algorithms . . . 19

2.5.2 Depth-first Search Algorithms . . . 20

2.6 Algorithms for Episode Mining . . . 20

2.6.1 MINEPI . . . 21

2.6.2 MINEPI+ . . . 21

2.6.3 EMMA . . . 22

2.6.4 NONEPI . . . 24

2.6.5 Conclusion . . . 27

3 The Proposed Algorithm **28**

3.1 Introduction . . . 28

3.2 Background: Serial vs Parallel Episodes . . . 28

3.3 Motivation for NONEPI-P . . . 29

3.4 Design and Algorithm Description . . . 30

3.4.1 Episode Representation . . . 30

3.4.2 Occurrence Definition . . . 30

3.4.3 Candidate Generation . . . 30

3.4.4 Support Counting . . . 30

3.4.5 Pruning . . . 30

3.4.6 Algorithm Flow . . . 30

3.5 Key Function Modifications in NONEPI-P Algorithm . . . 31

3.5.1 Extracting frequent episodes 31
3.5.2 Occurrence Recognition 34
3.5.3 Extracting episode rules 38
3.6 Applications of NONEPI-P Algorithms 42
3.6.1 Real-World Applications & Examples 42
3.7 Limitations of the NONEPI-P Algorithm 43
3.8 Conclusion 43

4 Implementation And Experimental Study **44**
4.1 Introduction 44
4.2 Implementaion 44
4.2.1 Tools 44
4.3 The implementation steps 46
4.3.1 Understanding Web User Behavior 47
4.3.2 Data preparation 47
4.3.3 Python Script Use for merging the Dataset: 50
4.3.4 Data Transformation for the algorithms 50
4.3.5 Applying the Algorithms 53
4.3.6 Results Interpretation 54
4.3.7 Extracting the rules 56
4.4 Conclusion 58

References **61**

Abbreviations list

ANNs Artificial Neural Networks.

COBWEB Incremental Clustering Algorithm.

Conf Confidence.

DM Data Mining.

Eclipse IDE Integrated Development Environment Eclipse.

FEM Frequent Episode Mining.

GSP Generalized Sequential Pattern.

JDK Java Development Kit.

KDD Knowledge Discovery in Databases.

LVQ Learning Vector Quantizatio.

MINCONF Minimum Confidence Threshold.

MINEPI Mining Episodes.

MLP Multi-Layer Perceptron.

NONEPI Non-overlapping Episodes.

NONEPI-P Non-overlapping Episodes Parallel.

RBFNs Radial Basis Function Networks.

SPMF Sequential Pattern Mining Framework.

WINEPI Algorithm for Mining Frequent Episodes.

List of Figures

1.1 Data mining—core of knowledge discovery process 6
1.2 Decision tree 7

3.1 Original pseudocode of the NONEPI algorithm for Extracting frequent episodes. 32
3.2 Flowchart of Mining Frequent Episodes 34
3.3 Occurrence Recognition procedure used in the NONEPI algorithm. 35
3.4 Flowchart of Occurrence Recognition Algorithm 38
3.5 Rule extraction phase of the original NONEPI algorithm. 39
3.6 Flowchart of Generate Parallel Episode Rules 41

4.1 Steps of the implementation 46
4.2 Screenshot of the Dataste after preparation 49
4.3 Python code for merge the data files 50
4.4 Screenshot of the Dataste format 51
4.5 Python code for converting data to the input format 52
4.6 The effect of minsup on the amount of frequent episodes. 54
4.7 The effect of minsup on max episode size. 55
4.8 The effect of minsup on the execution time. 55
4.9 Influence of minconf on number of episode rules with $minsup = 500$. 56
4.10 Influence of minconf on execution time with $minsup = 500$. 56

List of Tables

2.1 Frequency definitions overview 17
2.2 Event sequence with time points and corresponding itemsets 25
2.3 Example of episode rule discovered by NONEPI 26

4.1 Dataset Overview 49
4.2 Details of the dataset used 51
4.3 Number of Frequent Episodes Discovered by Different Algorithms at Various minSup Thresholds 54
4.4 Maximum Size Values of Patterns Discovered by Different Algorithms at Various minSup Thresholds 55
4.5 Execution Time (in ms) of Different Algorithms at Various minSup Thresholds 55
4.6 Sample of 10 Episode Rules Discovered by NONEPI-P 58

List of Algorithms

1 Extracting frequent Parallel Episodes 33
2 Occurrence Recognition Parallel 37
3 Generate Parallel Episode Rules 40

General Introduction

The explosive growth of digital data in recent years has created both opportunities and challenges for extracting meaningful knowledge from complex and voluminous datasets. In response, data mining has emerged as a critical discipline that enables the discovery of patterns, correlations, and insights across various domains such as finance, healthcare, cybersecurity, and e-commerce. Within this broad field, pattern mining focuses on identifying recurring structures in data, helping to uncover regularities and behavioral trends.

A particularly important subfield of pattern mining is episode mining, which targets the discovery of frequent episodes sets of events that occur in a specific order (serial) or without order constraints (parallel) within sequential data streams. Episode mining is especially relevant in applications where understanding the temporal dynamics of event sequences is crucial, such as clickstream analysis, system monitoring, and user behavior prediction.

This thesis is dedicated to an in-depth exploration of episode mining, covering both the theoretical foundations and the practical aspects of algorithm design and evaluation. After reviewing key concepts and techniques in data and pattern mining, the thesis focuses on the frameworks, frequency definitions, and search strategies that underpin frequent episode mining (FEM). Special attention is given to various mining algorithms, including MINEPI, MINEPI+, EMMA, and NONEPI.

A central contribution of this work lies in the implementation and experimental study of the NONEPI algorithm, along with a custom parallelized version designed to handle parallel episodes more effectively called NONEPI-P. Using real-world event sequence data, the thesis evaluates the performance, accuracy, and applicability of these algorithms in discovering frequent episodes and generating association rules. The experimental results provide valuable insights into the strengths and limitations of existing techniques and demonstrate the potential

of enhanced approaches for practical pattern discovery tasks.

By combining a solid theoretical foundation with hands-on experimentation, this thesis aims to advance the understanding of episode mining and contribute to its application in real-world data analysis scenarios. This thesis is organized as follows:

- **Chapter 1** introduces the key concepts of data mining, including its main objectives, techniques, and application domains, and we focus on pattern mining and episode mining.
- **Chapter 2** explores the theoretical foundations of episode mining, covering core concepts, frequency definitions, search strategies, and major algorithms such as MINEPI, MINEPI+, EMMA, and NONEPI.
- **Chapter 3** presents our main contribution: the development of the **NONEPI-P** algorithm, a parallelized extension of NONEPI. This chapter details its motivation, design, and improvements.
- **Chapter 4** describes the implementation process and experimental study, including data preparation, the application of algorithms, performance analysis, and a comparison of results.
- Finally, a **general conclusion** summarizes the key findings of this work and outlines potential future research directions.

Chapter 1

Data mining

1.1 Introduction

In this chapter, we present the key concepts and methods related to data mining, with a particular emphasis on pattern mining and episode mining. we begin by defining data mining and discussing its main objectives, such as discovering hidden patterns, making accurate predictions, and identifying anomalies within large datasets. Then we discuss the typical stages of the data mining process and explores several important techniques, including neural networks, decision trees, clustering, and association rule mining. Toward the end of the chapter, we focus more closely on pattern mining challenges and introduce episode mining a technique specifically aimed at uncovering significant sequences of events in temporal data.

1.2 Definition

Data mining is defined as the process of discovering patterns, relationships, and insights from large datasets using various statistical, mathematical, and computational techniques. It is an essential step in the broader field of Knowledge Discovery in Databases (KDD) [1].

1.3 Objectives

- **Classification and Clustering:** Classification and clustering are essential objectives that enable data-driven grouping and labeling. Classification assigns data instances to pre-

defined categories using supervised learning methods, while clustering discovers natural groupings in data using unsupervised learning techniques [2]. These approaches are commonly applied in applications such as fraud detection, sentiment analysis, and customer segmentation. In episode mining, discovered patterns can be used to classify sequence behaviors or identify similar temporal event flows across users or systems [2][3].

- **Anomaly Detection:** Identifying outliers and unusual patterns is a critical data mining objective with applications in fraud detection, network security, and quality control [4]. Methods range from statistical approaches (Z-scores) to machine learning techniques (Isolation Forests). These systems learn normal patterns and flag deviations, such as fraudulent credit card transactions or manufacturing defects, helping maintain system integrity and security [4][5].

- **Pattern Discovery:** A fundamental objective of data mining is pattern discovery uncovering frequent, interesting, or previously unknown patterns within large datasets[2]. Techniques such as association rule mining (e.g: the Apriori algorithm) and sequential pattern mining are widely used to detect co-occurrences and sequences in transactional or event-based data. In particular, episode mining focuses on discovering meaningful sequences of events over time, which is especially useful in domains such as healthcare monitoring, system logs, and user behavior analytics. In our work, we applied and extended episode mining algorithms to discover frequent serial and parallel episodes in user interaction data, enabling the identification of key behavioral patterns[2] [5].

- **Prediction:** A core objective of data mining is discovering hidden patterns and relationships within datasets [6]. Techniques such as association rule mining (Apriori algorithm) and sequential pattern analysis reveal meaningful correlations, like market basket associations or temporal event sequences in healthcare data. Episodic mining, a specialized form of pattern discovery, is particularly valuable for analyzing time-based event sequences in domains like customer behavior tracking. In our study, the episode rules extracted were used to support predictive insights, helping anticipate future user actions based on past behavioral patterns[6][4].

1.4 Data Mining Process

To perform data mining, a process consisting of seven steps is usually followed. This process is often called the "Knowledge Discovery in Database" (KDD) process As shown in Figure 1.1.

1. **Data cleaning:** This step consists of cleaning the data by removing noise or other inconsistencies that could be a problem for analyzing the data.

2. **Data integration:** This step consists of integrating data from various sources to prepare the data that needs to be analyzed. For example, if the data is stored in multiple databases or files, it may be necessary to integrate the data into a single file or database to analyze it.

3. **Data selection:** This step consists of selecting the relevant data for the analysis to be performed.

4. **Data transformation:** This step consists of transforming the data to a proper format that can be analyzed using data mining techniques. For example, some data mining techniques require that all numerical values are normalized.

5. **Data mining:** This step consists of applying some data mining techniques (algorithms) to analyze the data and discover interesting patterns or extract interesting knowledge from this data.

6. **Evaluating the knowledge that has been discovered:** This step consists of evaluating the knowledge that has been extracted from the data. This can be done in terms of objective and/or subjective measures.

7. **Visualization:** Finally, the last step is to visualize the knowledge that has been extracted from the data.

Of course, there can be variations of the above process. For example, some data mining software are interactive, and some of these steps may be performed several times or concurrently[7].

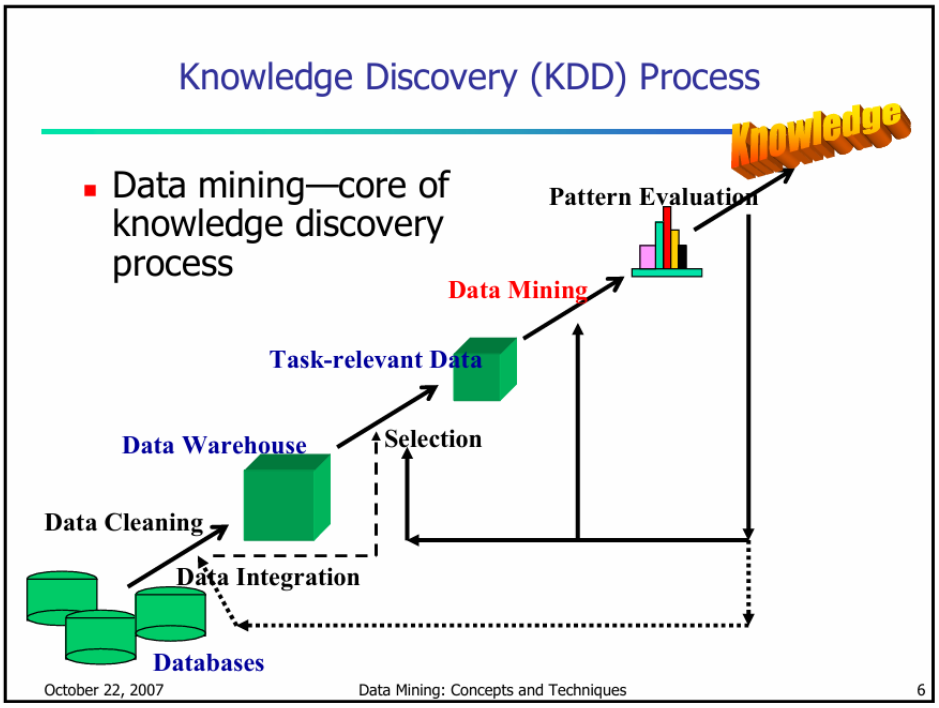


Figure 1.1: Data mining—core of knowledge discovery process

1.5 Techniques

A large number of techniques for DM are well-known and used in many applications. This section provides a short review of selected techniques considered the most important and frequent in DM [8].

- **Artificial Neural Networks (ANNs) :**

  ANNs are powerful mathematical models suitable for almost all DM tasks, especially predictive one. There are different formu lations of ANNs, the most common being the Multi-Layer Perceptron (MLP), Radial Basis Function Networks (RBFNs) and Learning Vector Quantization (LVQ). ANNs are based on the definition of neurons, which are atomic parts that compute the aggregation of their input to an output according to an activation function.They usually outperform all other models because of their complex structure; however, the complexity and suitable configuration of the networks make them not very popular when regarding other methods, being considered as the typical example of black box models. Similar to regression models, they require numeric attributes and no MVs .How ever ,if they are appropriately configured, they are robust against outliers and noise [8].

- **Decision Trees:**

  A decision tree is a flow chart like structure where each node denotes a test on an attribute value, each branch represents an outcome of the test and tree leaves represent classes or class distribution. A decision tree is a predictive model most often used for classification. Decision trees partition the input space into cells where each cell belongs to one class. The partitioning is represented as a sequence of tests. Each interior node in the decision tree tests the value of some input variable, and the branches from the node are labelled with the possible results of the test. The leaf nodes represent the cells and specify the class to return if that leaf node is reached. The classification of a specific input instance is thus performed by starting at the root node and, depending on the results of the tests, following the appropriate branches until a leaf node is reached. Decision tree is represented in figure 1.2 [9].

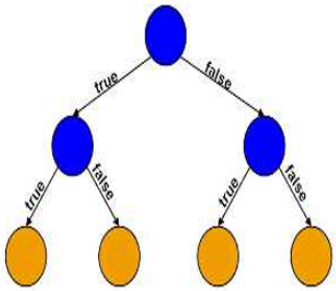


Figure 1.2: Decision tree

- **Clustering:**

  It appears when there is no class information to be predicted but the examples must be divided into natural groups or clusters. These clusters reflect subgroups of examples that share some properties or have some similarities. They work by calculating a multivariate distance measure between observations, the observations that are more closely related. Roughly speaking, they belong to three broad categories: Agglomerative clustering, divisive clustering and partition ing clustering. The former two are hierarchical types of clustering opposite one another. The divisive one applies recursive divisions the entire data set whereas 6 1 Introduction agglomerative ones start by considering each example as a cluster and perform ing an iterative merging of clusters until a criterion is satisfied. Partitioning based clustering, with k-Means algorithms as the most representative, starts with a fixed k number of clusters and iteratively adds or removes examples to and from

them until no improvement is achieved based on a minimization of intra and/or inter cluster distance measure. As usual when distance measures are involved, numeric data is preferable together with no-missing data and the absence of noise and out liers. Other well known examples of clustering algorithms are COBWEB and Self Organizing Maps [8].

- **Association Rules:**

  They are a set of techniques that aim to find association rela tionships in the data. The typical application of these algorithms is the analysis of retail transaction data . For example, the analysis would aim to find the likelihood that when a customer buys product X, she would also buy product Y. Association rule algorithms can also be formulated to look for sequential patterns. As a result of the data usually needed for association analysis is transaction data, the data volumes are very large. Also, transactions are expressed by categorical values, so the data must be discretized. Data transformation and reduction is often needed to perform high quality analysis in this DM problem.The Apriori technique is the most emblematic technique to address this problem [8].

1.6 Domains of Application

Data mining techniques have been successfully applied in diverse domains, enabling organizations to extract valuable insights and make informed decisions. Some of the key application areas include:

1. **Healthcare:**

   Used for disease diagnosis, patient outcome prediction, personalized treatment planning, and analyzing medical records to improve healthcare delivery and patient care [6].

2. **Finance and Banking:**

   Applications include credit scoring, fraud detection, risk management, customer segmentation, and investment analysis [6, 10].

3. **Retail and E-commerce:**

   Used for market basket analysis, customer behavior analysis, sales forecasting, inventory management, and targeted marketing [11].

4. **Marketing and Customer Relationship Management (CRM):**
   Enables segmentation, customer profiling, churn prediction, and campaign management [6].

5. **Manufacturing and Supply Chain Management:**
   Supports quality control, demand forecasting, process optimization, and logistics planning [11].

6. **Education:**
   Helps analyze student performance, predict academic success, identify at-risk students, and improve educational strategies [6].

7. **Telecommunications:**
   Used for customer churn analysis, network optimization, fraud detection, and service personalization [10].

8. **Cybersecurity and Intrusion Detection:**
   Detects anomalies, network intrusions, malware, and spam by analyzing security data [11].

9. **Human Resources:**
   Applied in employee retention analysis, recruitment, performance evaluation, and workforce planning [6].

10. **Crime Analysis and Law Enforcement:**
    Assists in crime trend analysis, hotspot detection, and criminal profiling [10].

11. **Sports Analytics:**
    Used to analyze player performance, game strategies, and fan engagement [11].

12. **Agriculture:**
    Helps in crop yield prediction, disease detection, and resource management [6].

13. **Social Media and Web Mining:**
    Analyzes user behavior, sentiment, and trends to support targeted advertising and content recommendation [11].

These applications demonstrate the versatility and transformative potential of data mining

across modern industries, helping organizations uncover hidden patterns, predict future trends, and make more informed decisions [6, 10, 11].

1.7 Pattern Mining

1.7.1 Definition

Pattern Mining refers to the process of extracting meaningful and recurring structures, relationships, or trends from datasets. These patterns can be expressed as frequent itemsets, sequences, graphs, or episodes, depending on the nature of the data and the mining task.[1]

1.7.2 Types of Patterns

Patterns can be categorized into several types based on their structure and representation:

1. **Frequent Itemsets:** A set of items that frequently appear together in a dataset. For example, in market basket analysis, a frequent itemset might include products like bread and butter. The Apriori algorithm is one of the most well-known methods for mining frequent itemsets [12].

2. **Sequential Patterns:** Patterns where the order of events or items matters. For instance, customers might buy a phone, then a case, and then headphones. Algorithms like PrefixSpan and GSP are commonly used [12].

3. **Episode Mining:** Focuses on discovering patterns in sequences of events within a specific time window. Example: multiple user interactions like click, scroll, and hover occurring within the same time frame, regardless of their order [5].

4. **Graph Patterns:** Patterns represented as subgraphs in graph-structured data. In social networks, for example, a community of users who interact frequently. Algorithms include gSpan and SUBDUE [13].

1.7.3 Applications

Pattern Mining has a wide range of applications:

- **Market Analysis:** Identifying frequently co-purchased items to optimize store layouts and promotions.
- **Bioinformatics:** Discovering recurring sequences in DNA or protein structures to understand genetic functions and diseases.
- **Web Usage Mining:** Analyzing user behavior on websites to improve experience and personalize content.
- **Fraud Detection:** Identifying unusual transaction patterns to detect fraudulent activities early [14].

1.7.4 Challenges

Despite its usefulness, Pattern Mining faces several challenges:

1. **Scalability:** Handling large datasets efficiently.
2. **Pattern Explosion:** Managing the exponential number of patterns generated.
3. **Noise and Redundancy:** Filtering out irrelevant or redundant patterns.
4. **Interpretability:** Ensuring that discovered patterns are meaningful and actionable [15].

1.8 Episode Mining as a Subfield of Pattern Mining

Episode Mining focuses on discovering patterns in sequences of events, considering the temporal relationships between events. It is particularly useful for analyzing time-dependent data.

1.8.1 Definition

Episode Mining refers to discovering frequent patterns (episodes) in event sequences where order and timing matter. For instance, a series of login attempts followed by a breach within 10

minutes is an episode[5].

1.8.2 Key Characteristics

The key characteristics outlined above form the foundation for analyzing temporal data effectively. By focusing on the order and timing of events, utilizing timestamped sequences, defining clear time windows, and applying frequency thresholds, this approach ensures that meaningful and relevant patterns are identified within complex datasets.

1. **Temporal Relationships:** Considers order and timing of events.
2. **Event Sequences:** Input is a timestamped sequence of events.
3. **Time Windows:** Episodes are defined within specific intervals.
4. **Frequency and Support:** Focuses on patterns that meet a frequency threshold [5].

1.8.3 Relationship to Other Pattern Mining Techniques

- **Frequent Itemset Mining:** Ignores order and timing.
- **Sequential Pattern Mining:** Considers order but not time windows.
- **Episode Mining:** Considers both order and timing, ideal for time-series data [1].

1.9 Conclusion

This chapter has introduced the core concepts of data mining, from its definition and objectives to the KDD process and common techniques like neural networks, decision trees, clustering, and association rules. We explored applications across various domains and examined the role of machine learning. Additionally, we delved into pattern mining, highlighting frequent itemsets, sequential patterns, and episode mining, while also addressing the challenges of scalability and interpretability. Episode mining was presented as a subfield focusing on temporal relationships within event sequences. This overview provides a foundation for understanding data mining's potential and complexity.

Chapter 2

Episode Mining

2.1 Introduction

This chapter provides an overview of episode mining, a technique used to discover frequent patterns in temporal event sequences. We first introduce the main concepts and frequency definitions used in episode mining. Then, we present key search strategies and describe important algorithms such as MINEPI, MinePI+, EMMA, and NON-EPI. These foundations are essential for understanding the improvements proposed in the next chapter.

2.2 Key Concepts in Episode Mining

The typical input of an FEM algorithm is an event sequence and a user-defined integer threshold, called minsup, which represents the minimum number of occurrences that an episode must have to be considered as frequent [16].

Let there be a set of event types E = E1E2 En . An event sequence is a triplet S = (sTs Te) indicating a list of events s that have been recorded between a time Ts and a time Te. More precisely, the list s is an ordered set of event pairs of the form s = (I1 T1) (I2 T2) (ImTm) where Ii E is an event type and Ti is the timestamp at which the event was observed (1 i m). Event pairs in s are ordered by increasing timestamps, and an event type may be observed multiple times in s.

For example, Figure 2.1 shows an example event sequence captured between time Ts = 10

and Te = 27. This sequence may represent the event log of a database system, where event types E = a,b,c,d,e,f may be disk operations such as to open, close, read, or write in a file. In this sequence, the following list of events has been recorded: s = (a10) (b11) (c13) (d14).....(d26) note that in the following, the terms event and event type are used interchangeably when the context is clear.

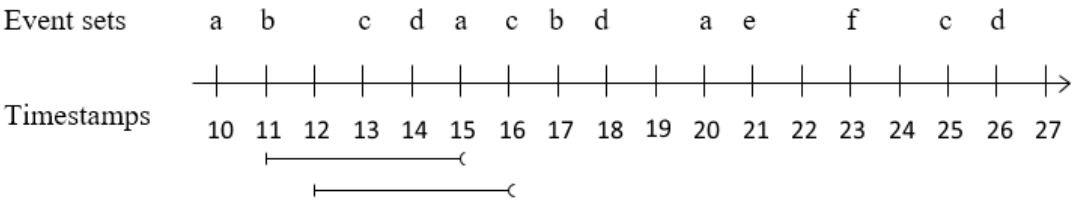


Figure 2.1: An example of event sequence

2.2.1 Event and Event Sequence

• **Event:** An event is a pair (e,t), where e is the event type (e.g., "alarm triggered") and t is the timestamp indicating when the event occurred.

• **Event Sequence:** An event sequence S=⟨(e1,t1),(e2,t2),...,(en,tn)⟩ is an ordered list of events, where each event is associated with a timestamp. The sequence is ordered such that ti<tj for all i<j. Event sequences can model various types of data such as:

2.2.2 Occurrence and Minimal occurrence

• **Occurrence:** Given an episode $\alpha = e_1 \to \cdots \to e_i \to \cdots \to e_k$, $[t_1,\ldots,t_i,\ldots,t_k]$ is an occurrence of α if and only if:

1. e_i occurs at t_i for all $i \in [1,k]$;
2. $t_1 < t_2 < \cdots < t_k \leq \delta$, where δ is a user-specified threshold called the *maximum occurrence window*.

t_1 is the start time and t_k is the end time of the occurrence [17].

• **Minimal occurrence:** Consider two time windows $[t_1,t_2]$ and $[t_1',t_2']$. We say that $[t_1',t_2']$ is

subsumed by $[t_1, t_2]$ if $t_1 \leq t_1'$ and $t_2' \leq t_2$.

An occurrence window of episode α, $[t_1, t_2]$, is a *minimal occurrence window* of α if no other occurrence window $[t_1', t_2']$ of α is subsumed by $[t_1, t_2]$.

The occurrence of α in a minimal occurrence window $[t_1, t_2]$ is defined as a *minimal occurrence* of α, denoted as $(\alpha, [t_1, t_2])$.

The set of all distinct minimal occurrences of α is denoted moSet(α) [17].

2.2.3 Episode

An episode is a collection of events that occur in a specific order or pattern set by time. Formally, an episode is defined as:

• **Definition:** An episode $\alpha = (V, \leq, f)$ consists of:

– A set of nodes V, where each node represents an event.

– A partial or total order $\leq$ on the nodes V.

– A mapping function ff:V$\rightarrow$E, where E is the set of event types [17].

• **Support of an episode:** The support of an episode α, denoted as sp(α), is defined as the number of distinct minimal occurrences, i.e., sp(α) = |moSet(α)| [17].

• **Sub-episode:** Let $\alpha = A_1 A_2 \ldots A_n$ and $\beta = B_1 B_2 \ldots B_m$ be two episodes. α is said to be a *sub-episode* of β (denoted $\alpha \subseteq \beta$) if and only if there exist integers $k_1, k_2, \ldots, k_m$ such that $1 \leq k_1 < k_2 < \cdots < k_m \leq n$ and for all $j \in [1, m]$, $B_j = A_{k_j}$.

In other words, β contains α as a (possibly non-contiguous) subsequence.

It is easy to see that every occurrence of β in an event sequence contains an occurrence of α. Conversely, if β contains α, then β is a *super-episode* of α.

Two particular cases are distinguished:

- If the first element of α matches the first element of β, then α is called a *prefix* of β.
- If the last element of α matches the last element of β, then α is called a *suffix* of β.

For example, the episode $\alpha = AN$ is a sub-episode of $\beta = ANB$, and α is a prefix of β [17].

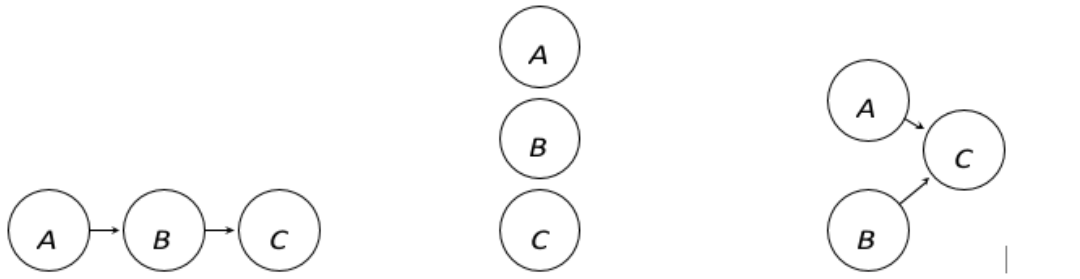
• **Episode Rule :** An *episode rule* is an expression of the form $\beta \Rightarrow \alpha$, where α and β are two frequent episodes in the sense of frequency based on distinct occurrences [17].

• **Support of an episode rule :** support of an episode rule $\alpha \Rightarrow \beta$, denoted $\text{supER}(\alpha \Rightarrow \beta)$, is defined as: $\text{supER}(\alpha \Rightarrow \beta) = |\text{occER}(\alpha \Rightarrow \beta)|$, where $\text{occER}(\alpha \Rightarrow \beta) = do_{\alpha \cup \beta}$, i.e., the distinct occurrences of the episode $\alpha \cup \beta$ [17].

• **the confidence of an episode rule :** The Conf of an episode rule $\alpha \Rightarrow \beta$, denoted $\text{conf}(\alpha \Rightarrow \beta)$, is defined as: $\text{conf}(\alpha \Rightarrow \beta) = \dfrac{\|\text{occER}(\alpha \Rightarrow \beta)\|}{\text{support}(\alpha)}$.

• **Types of Episodes:**

1. **Serial Episode**: Events occur in a specific order (e.g., A → B → C).
2. **Parallel Episode**: Events occur in any order (e.g., A, B, C).
3. **Composite Episode**: A combination of serial and parallel episodes.



(a) Serial episode (b) Parallel episode (c) Composite episode

Figure 2.2: The three primary episode types

2.3 Frequency Definitions in Episode Mining

The frequency of an episode is a critical measure in episode mining, as it determines how often an episode occurs in a sequence of events. Different frequency definitions have been proposed to address various challenges in episode mining, such as handling overlap-

ping occurrences, reducing computational complexity, and ensuring meaningful results. Below, we discuss the two main categories of frequency definitions: window-based frequency and occurrence-based frequency, along with their reliable sources [5].

2.4 Frequent Episode Mining framework

Frequent episode mining (FEM) is a popular framework to analyze temporal data. It has been used in various domains where data contains time information. Frequent episode mining algorithms extract all frequent episodes either from a simple sequence of events or a complex event sequence. The former refers to a sequence where events are not allowed to occur simultaneously, while the latter allows simultaneous events [16].

Table 2.1: Frequency definitions overview

Type	**Name**	**Monotonic**	**Representative algorithm(s)**
Window-based	window-based frequency	Yes	WINEPI [18], EpiBF [19], WinMiner [20]
	head frequency	No	EMMA [21]
	total frequency	Yes	FEM-DFS [22]
	non-interleaved frequency	No	NOE-WinMiner [23]
Occurrence-based	minimal occurrence-based	No	MINEPI [5], MEELO [24], PartiteCD [25]
	non-overlapped occurrence-based	Yes	NONEPI [26], POERM [27]
	distinct occurrence-based	Yes	ONCE+ [28]

2.4.1 Window-based Frequency

Window-based frequency is one of the earliest and most widely used definitions for measuring the frequency of episodes. It was introduced by Mannila, Toivonen, and Verkamo (1997) in their foundational work on episode mining[5] .

• **Definition:** The frequency of an episode is defined as the number of fixed-size windows in which the episode occurs at least once. A window is a contiguous segment of the event sequence with a predefined length.

Formally, for a given window size w, the frequency of an episode α is:

$$\text{freq}(\alpha) = \frac{\text{Number of windows containing } \alpha}{\text{Total number of windows}} \tag{2.1}$$

Example: Consider an event sequence with 10 timestamps and a window size of 5. If the episode A→B occurs in 3 out of 6 possible windows, its frequency is 3/6=0.5.

2.4.2 Occurrence-based Frequency

Occurrence-based frequency provides a more precise measure of episode frequency by counting the number of non-overlapping or distinct occurrences of an episode in the sequence. This approach was further developed by Laxman, Sastry, and Unnikrishnan (2005) in their work on efficient episode mining algorithms [29].

• **Definition:** The frequency of an episode is defined as the number of non-overlapping or distinct occurrences of the episode in the sequence.

- **Non-overlapping Occurrences**: No two occurrences share any common events.
- **Distinct Occurrences**: Each occurrence is unique and does not share timestamps with other occurrences.
- **Formally**, for an episode α, the frequency is:

$$\text{freq}(\alpha) = \text{Number of non-overlapping or distinct occurrences of } \alpha$$

  Example: Consider an event sequence where the episode $A \rightarrow B$ occurs at timestamps [1, 3], [5, 7], and [9, 11]. The frequency of $A \rightarrow B$ is 3, as there are three non-overlapping occurrences.

- **Advantages**:
  - Provides a more accurate measure of frequency by avoiding double-counting.
  - Suitable for applications where overlapping occurrences are not meaningful.
- **Limitations**:
  - Computationally more expensive than window-based frequency.
  - Requires careful tracking of occurrences to ensure non-overlapping or distinct counts.

2.4.3 Summary of Frequency Definitions

Frequency Definition	Key Idea	Advantages	Limitations
Window-based Frequency	Counts occurrences in fixed-size windows	Simple to compute and interpret	May overcount overlapping occurrences
Occurrence-based Frequency	Counts non-overlapping or distinct occurrences	More accurate frequency measure	Computationally expensive

Figure 2.3: Summary of Frequency Definitions

2.5 Search strategies for FEM

Since Mannila introduced the problem of frequent episode mining, several algorithms have been developed to improve the FEM process. FEM algorithms utilize distinct data structures and are crafted to accommodate multiple frequency definitions. In general, FEM algorithms can be categorized with respect to their search strategy, which is either a breadth-first search or a depth-first search [16].

2.5.1 Breadth-first algorithms

Commonly referred to as level-wise algorithms, they process the exploration of episodes through a cyclic progression between two fundamental stages: (1) candidate generation and (2) frequency validation. In the first phase, candidate episodes are derived from smaller episodes, and in the second phase, the frequency of these candidate episodes is assessed to ascertain their status as frequent occurrences.

Breadth-first search methodologies, guided by the anti-monotonicity property, prioritize the enumeration of lengthier episodes while disregarding numerous infrequent episodes. Illustratively, the CloEpi algorithm initiates by comprehensively scanning the entire event sequence to identify frequent 1-episodes. Subsequently, CloEpi conducts a re-examination of the sequence, systematically identifying frequent 2-episodes, 3-episodes, and so on, up to m-episodes, where m indicates the width of the episode [16].

2.5.2 Depth-first Search Algorithms

Method like **EMMA** and conduct a single pass over the event sequence to extract all 1-episodes. Subsequently, these algorithms endeavor to iteratively expand each episode by augmenting it with a frequent event, leveraging the anti-monotonicity property for the purpose of search space reduction. The adoption of a depth-first strategy typically results in decreased time and memory requirements during the process of frequent episode mining.

With the aim of formulating an efficient algorithm for Frequent Episode Mining (FEM), a critical imperative involves circumventing the exhaustive exploration of the complete search space of episodes within the input sequence to identify frequent episodes. This is particularly pivotal because of the potential enormity of the search space for comparatively extensive sequences, which leads to protracted runtimes and heightened memory requirements.

Using the frequency measure's *anti-monotonicity* property is a key method for overcoming this challenge. Let $\sup(\gamma)$ represent the frequency of an episode γ. This property states that for every pair of episodes γ and δ where $\gamma \sqsubseteq \delta$ (i.e., γ is a sub-episode of δ), the following holds:

$$\sup(\gamma) \geq \sup(\delta)$$

Consequently, if an episode α is infrequent, all its super-episodes are inherently infrequent, obviating the need for exploration and effectively reducing the amount of time and memory required [16].

2.6 Algorithms for Episode Mining

Episode mining algorithms are designed to efficiently discover frequent episodes in event sequences. These algorithms differ in their approach to defining and counting episodes, as well as in their search strategies (e.g., breadth-first vs. depth-first). Below, we discuss some of the most influential algorithms in the field, along with their methodologies and contributions.

2.6.1 MINEPI

MINEPI (Minimal Occurrence-based Episode Mining) is a foundational episode mining algorithm that introduces the concept of minimal occurrence-based frequency. Instead of just checking for presence in a window, MINEPI finds the exact minimal time span where an episode occurs and uses this to count its frequency.

Various algorithms have been proposed to find frequent episodes in data. It is important to note that different algorithms do not necessarily count the support (i.e., occurrence frequency) of an episode in the same way. MINEPI uses the concept of *minimal occurrences* to define support [5].

Key Characteristics of MINEPI:

- **Methodology:** MINEPI counts minimal occurrences of episodes, where a minimal occurrence is the smallest time interval in which the episode occurs. This helps avoid the overcounting problem present in WINEPI.
- **Frequency Definition:** Minimal occurrence-based frequency.
- **Advantages:** Provides a more accurate measure of episode frequency by avoiding overlapping occurrences.
- **Limitations:** Computationally more expensive than WINEPI due to the need to track minimal occurrences.
- **Applications:** Suitable for use cases that require precise frequency counts, such as healthcare and financial analysis.

2.6.2 MINEPI+

MINEPI+ is an optimized version of MINEPI that retains the same episode model and frequency definition but incorporates efficiency improvements in candidate generation and pruning. Like MINEPI, it mines both serial and parallel episodes within fixed-size windows but is designed to scale better for larger datasets.

Key Characteristics of MINEPI+:

- **Methodology:** MINEPI+ enhances the original MINEPI by pruning unpromising candidate episodes early in the mining process. This is achieved using prefix-based pruning, which discards candidates that do not meet the frequency requirements of their subpatterns. The algorithm is much faster because it reduces the number of candidates and scans needed.

- **Frequency Definition:** Similar to MINEPI, the frequency is defined as the number of valid occurrences of the episode, but MINEPI+ eliminates unnecessary episodes earlier in the process to optimize performance.

- **Advantages:**

  - Faster than MINEPI due to candidate pruning.

  - Efficient: Reduces unnecessary computation by skipping unlikely candidates.

  - Scalability: Works well on large datasets by minimizing data scans.

- **Limitations:**

  - Complex implementation: The optimization mechanisms can make the algorithm harder to implement and understand compared to MinePI.

  - Risk of missing rare patterns: In some cases, pruning may discard rare but valid episodes.

- **Application:**

  - Web mining: Detecting patterns in user behavior from web logs.

  - E-commerce: Predicting customer behavior or recommending products based on event sequences.

2.6.3 EMMA

EMMA (Efficient Memory-Anchored Miner for Episodes) (Kuo-Yu et al., 2008) is an algorithm designed for discovering frequent episodes in a complex event sequence. Frequent episode mining involves finding subsequences of events that appear frequently within a long sequence of events, where some events may occur simultaneously. EMMA is notable for count-

ing the support of episodes using a concept called head frequency and can handle datasets both with and without timestamps by adjusting a parameter accordingly [30].

Key Characteristics of EMMA:

- **Methodology:** EMMA (Episodes Mining using Memory Anchor) is an algorithm designed to discover frequent serial episodes in complex event sequences. It uses a depth-first search strategy combined with memory anchors (boundlists) to efficiently enumerate frequent episodes by joining existing episodes with locally frequent items. EMMA counts episode occurrences using a concept called head frequency, which ensures accurate support counting even in sequences with simultaneous events. It can handle datasets with or without timestamps by adjusting parameters accordingly.

- **Frequency Definition:** An episode's frequency in EMMA is defined by the number of its occurrences counted through memory anchors, focusing on exact and non-redundant occurrences within a maximum window length.

- **Advantages:**

  - Efficient: Uses memory anchors to reduce the search space and avoid redundant joins.

  - Accurate: Counts occurrences precisely, considering temporal constraints and simultaneous events

  - Flexible: Applicable to complex sequences with multiple events per time slot and supports timestamped or non-timestamped data

- **Limitations:**

  - Focused on serial episodes (ordered sequences), with parallel episode mining addressed separately

  - May require careful parameter tuning (e.g., window size, minimum support) for optimal results.

- **Application:**

  - Behavioral analysis: Detecting frequent ordered patterns in user activity or system logs

- Anomaly detection: Identifying rare or unusual temporal patterns in time-series data.
- Complex event processing: Mining temporal patterns in domains like telecommunications or sensor networks

2.6.4 NONEPI

NONEPI (Non-Overlapping Episode Miner) is an algorithm designed to extract frequent serial episodes and predictive rules using a non-overlapped occurrence-based frequency definition. It avoids counting overlapping occurrences, leading to more independent and interpretable patterns [31].

Key Characteristics of NONEPI:

- **Methodology:** NONEPI ensures that episodes do not overlap by enforcing a rule that each detected episode must be disjoint. This means each occurrence is counted only once, and a new occurrence cannot reuse events from the previous one. This helps to model distinct events in time sequences.
- **Frequency Definition:** The frequency of an episode is defined as the number of distinct occurrences of the episode without overlap.
- **Advantages:**

  - Prevents over-counting: Avoids counting overlapping episodes, providing a clearer picture of distinct patterns.

  - Useful for event modeling: Ideal for datasets where repeated patterns in close proximity are irrelevant.

  - Improved quality of results due to non-overlapping occurrences.

- **Limitations:**

  - Lower support counts: Since it does not count overlapping occurrences, the frequency count may be lower.

  - Can miss overlapping patterns: If overlapping patterns are relevant to the application, this algorithm may not be suitable.

- **Application:**

  - Event sequence modeling: Useful for understanding distinct event occurrences, such as user journeys in web analytics.

  - Fraud detection: Identifying distinct fraudulent behaviors without redundancy.

  - Medical data analysis: Detecting distinct sequences of events in patient health records without overlaps.

What is the input?:

The algorithm takes as input an event sequence with a minimum support threshold, and a mininum confidence threshold. Let's consider the following sequence consisting of 11 time points (t1, t2, . . . , t10) and 4 event type (1, 2, 3, 4). This database is provided in the text file "contextEMMA.txt" in the package ca.pfv.spmf.tests of the SPMF distribution

Table 2.2: Event sequence with time points and corresponding itemsets

Time Point	Itemset (Event Set)
1	1
2	1
3	1 2
6	1
7	1 2
8	3
9	2
11	4

Each line of the sequence is called an event set and consists of one or more events occurring at the same time point.

- A **time point** (first column)

- An **event set** (second column)

For example, in the above table, at time point 1, the event 1 occurred. Then at time point 2, the event 1 occurred again. At time point 3, the events 1 and 2 occurred simultaneously, and so on [32].

What is the output?:

The output of NONEPI is the set of episode rules having a support (occurrence frequency) that is no less than a `minSup` threshold (a positive integer) set by the user, and a confidence that is no less than a minimum confidence threshold (a double number representing a percentage).

A rule is denoted as $X \rightarrow Y$, and it is a relationship between two sequences of events (episodes) X and Y. The intuitive meaning of a rule $X \rightarrow Y$ is that if the sequence of events X appears in a sequence, it will be followed by the sequence of events Y.

The **support** of a rule $X \rightarrow Y$ means how many times the rule appeared in the input sequence (i.e., how many times X was followed by Y). For example, if a rule appears three times, we say that the support of the rule is 3.

The **confidence** of a rule $X \rightarrow Y$ is defined as how many times $X \rightarrow Y$ appeared in the input sequence divided by how many times X appeared in the input sequence. Thus, the confidence can be seen as the conditional probability $P(Y|X)$, indicating how likely it is that Y will follow X in the sequence. The confidence is expressed as a percentage. For example, a confidence of 50% means that 50% of the times X was followed by Y in the input sequence.

The output of the algorithm includes all the episode rules that (1) have a confidence greater than or equal to the `minConf` parameter, and (2) a support greater than or equal to the `minSup` parameter.

It is important to note that **NONEPI** uses **non-overlapping support**. Thus, to calculate the support and confidence of rules, overlapping occurrences are not counted. (See the research paper for a formal explanation of non-overlapping occurrences.)

Now, let's look at the result for the above example. If we set `minSup` = 2 and `minConf` = 0.2 (20%), one rule is found [32].

Table 2.3: Example of episode rule discovered by NONEPI

Episode Rule	**Support**	**Confidence**
{1} ⇒ {1}{1 2}	3	0.667 (which means 66.7%)

2.6.5 Conclusion

In this chapter, we have explored the key concepts, methods, and practical aspects of episode mining. Starting with the basic definitions of events, episodes, and event sequences, we discussed how frequent episode mining frameworks help uncover meaningful patterns in temporal data. We reviewed the main frequency definitions, such as window-based and occurrence-based measures, and explained their advantages and limitations. We also outlined the major search strategies and algorithms used to efficiently discover frequent episodes, including both breadth-first and depth-first approaches. Overall, episode mining remains a powerful tool for analyzing sequential data, offering valuable insights in fields like network monitoring, finance, and cybersecurity. The techniques and ideas presented here provide a strong foundation for further study and application in real-world data analysis tasks.

Chapter 3

The Proposed Algorithm

3.1 Introduction

This chapter presents the main contribution of this thesis: the development of the NONEPI-P algorithm. Building upon the original NONEPI algorithm for mining episode rules from event sequences, we introduce a parallelized version specifically designed to efficiently handle parallel episodes. This chapter details the motivation, design, implementation, and evaluation of the NONEPI-P algorithm, highlighting its advantages and improvements over the original sequential approach.

3.2 Background: Serial vs Parallel Episodes

In temporal data mining, an episode is a collection of events occurring close together in time. The original NONEPI algorithm mines serial episodes, where the order of events is strictly maintained (e.g., event A followed by event B, then event C). However, many real-world scenarios involve events that happen concurrently or in no particular order, such as network events or sensor readings that are related but unordered.

Parallel episodes relax the ordering constraint, focusing on the co-occurrence of events within a time window regardless of their sequence. This allows the detection of more general and flexible temporal patterns that better capture the dynamics of complex systems. In our work, we focus on extending the NONEPI algorithm to support parallel episode mining,

enabling the detection of unordered but time related event patterns for improved analysis and prediction.

3.3 Motivation for NONEPI-P

The motivation to develop NONEPI-P stems from the need to:

- **Capture unordered event patterns:** Many applications require mining episodes where event order is irrelevant but their co-occurrence within a time frame is significant.
- **Better Suitability for Real-World Applications:** In many real-world datasets including our user interaction dataset from a recommendation system, events frequently occur, in groups or clusters without strict order.
- **Expanding the Expressive Power of the Algorithm:** By supporting unordered sets of events, NONEPI-P increases the expressive power of the model, enabling it to find more diverse and potentially more informative patterns.
- **Preserving the Non-Overlapping Frequency Measure:** One of NONEPI's strengths is its use of a non-overlapping frequency measure, which improves interpretability by avoiding repeated counting of overlapping episodes. In NONEPI-P, we retained this core strength, adapting the non-overlapping frequency logic to unordered (parallel) event sets.
- **Improved Algorithmic Flexibility:** By modifying only certain components (such as support counting and event matching), NONEPI-P retains the efficiency and modularity of the original algorithm, while expanding its use cases. It is also easier to implement and integrate into existing systems using the SPMF library

NONEPI-P thus modifies the original algorithm's episode representation and support counting to accommodate unordered event sets, enabling the discovery of parallel episodes.

3.4 Design and Algorithm Description

NONEPI-P modifies the original NONEPI algorithm as follows:

3.4.1 Episode Representation

Episodes are represented as **sets of events** rather than sequences. For example, the episode $\{A, B, C\}$ indicates that events A, B, and C occur within a specified time window, in any order.

3.4.2 Occurrence Definition

An occurrence of a parallel episode is defined as a subset of the event sequence where all events in the episode appear **within the maximum time window**, regardless of their order. This contrasts with serial episodes, where the order and timing between events must follow the episode's sequence.

3.4.3 Candidate Generation

Starting from frequent single-event episodes, candidates are generated by combining disjoint event sets. For episodes α and β, if their event sets are disjoint, the candidate episode is formed as the union $\alpha \cup \beta$. This process iteratively builds larger parallel episodes.

3.4.4 Support Counting

To count support, the algorithm searches for occurrences where all events of the candidate episode appear within the maximum time window. The counting procedure is adapted to check for **co-occurrences within windows**, ignoring event order.

3.4.5 Pruning

The anti-monotony property is preserved: if a candidate episode's subset is infrequent, the candidate is pruned. This ensures efficient exploration of the episode search space.

3.4.6 Algorithm Flow

1. Extract single-event occurrences: Identify timestamps where each event occurs.

2. Filter frequent single-event episodes: Retain events meeting the minimum support.
3. Iteratively generate candidate parallel episodes: Combine frequent episodes with disjoint event sets.
4. Count support of candidates: Check for occurrences within the time window.
5. Prune infrequent candidates: Remove those below support threshold.
6. Repeat until no new frequent episodes are found.

3.5 Key Function Modifications in NONEPI-P Algorithm

In this work, we modified the original NONEPI algorithm by redesigning three core functions to better support the mining of parallel episodes: Extracting Frequent Episodes, Occurrence Recognition, and Extracting Episode Rules. These modifications enable the algorithm to handle unordered event sets within a time window and improve the quality of episode rule extraction.

3.5.1 Extracting frequent episodes

Figure 3.1 presents the main pseudocode of the oroginal NONEPI algorithm.

Input: *minsup* - minimum support threshold

Output: F - the set of all frequent serial episodes

2 $P = \{\}, F = \{\}, \alpha = \emptyset$

4 **for** *each individual event* $e \in E$ **do**

6 $\quad no_e = \{\}$

8 **for** $k = 1$ *to* n **do**

9 $\quad$ % scan the sequence S

11 $\quad e$ = the event found at time t_k

13 $\quad no_e = no_e \cup \{[t_k, t_k]\}$

15 **for** *each individual event* $e \in E$ **do**

17 $\quad$ **if** $|no_e| \geq minsup$ **then**

19 $\quad\quad P = P \cup \{(e)\}$

21 $F = P$

23 **for** *each individual episode* $\alpha \in F$ **do**

25 $\quad$ **for** *each individual episode* $\beta \in P$ **do**

27 $\quad\quad no_{\alpha \sqcup \beta} = Occurrence\_Recognition(\alpha, \beta)$

29 $\quad\quad$ **if** $|no_{\alpha \sqcup \beta}| \geq minsup$ **then**

31 $\quad\quad\quad F = F \cup \{\alpha \sqcup \beta\}$

33 $\quad\quad\quad \alpha = \alpha \sqcup \beta$

35 **return** F

Figure 3.1: Original pseudocode of the NONEPI algorithm for Extracting frequent episodes.

Now, let us describe how the proposed algorithm is executed. Initially, it processes the input event sequence to identify all single-event episodes. For each event $e \in E$ occurring at timestamp t_k, a time interval $[t_k, t_k]$ is recorded in the set of occurrences `noe` corresponding to e (lines 4–6). This effectively captures all instances of single events across the sequence.

Next, the algorithm filters these single-event episodes by evaluating their support (lines 7–9). Any event e for which the number of occurrences in `noe` meets or exceeds the minimum support threshold (`minsup`) is considered a frequent episode. These frequent size-1 episodes are collected into the set P, which is also used to initialize the final result set F.

The algorithm then enters an iterative phase that explores the search space of larger parallel episodes. For each frequent episode $\alpha \in F$, and for each single-event episode $\beta \in P$, it checks whether α and β consist of disjoint events (i.e., they do not share any common event types). If

so, a candidate parallel episode $\alpha \sqcup \beta$ is formed.

To determine the support of these candidate episodes, the algorithm invokes the function with the parameters α, β, and the maximum time window (`maxwindow`). If the number of occurrences returned by this function meets or exceeds the support threshold, then $\alpha \sqcup \beta$ is added to the set F, and the episode α is extended accordingly (lines 12–16). This procedure continues in a depth-first fashion, ensuring that only those combinations that meet the support criteria are explored further.

In this way, the algorithm constructs all frequent parallel episodes by iteratively merging disjoint frequent episodes while preserving the time window constraint. The final result set F contains all frequent parallel episodes discovered during this process as it is shown in Algorithm 1.

Algorithm 1: Extracting frequent Parallel Episodes

Input: *minsup* - minimum support threshold
maxwindow - maximum time window for parallel episodes
Output: F - the set of all frequent parallel episodes

```
1  P ← {}, F ← {}, α ← ∅;
2  foreach individual event e ∈ E do
3      no_e ← {};
4      for k = 1 to n do
5          e ← event found at time t_k;
6          no_e ← no_e ∪ {[t_k, t_k]};
7  foreach individual event e ∈ E do
8      if |no_e| ≥ minsup then
9          P ← P ∪ {(e)};
10 F ← P;
11 foreach episode α ∈ F do
12     foreach episode β ∈ P do
13         if α and β have disjoint events then
14             no_{α∪β} ← Occurrence_Recognition_Parallel(α, β, maxwindow);
15             if |no_{α∪β}| ≥ minsup then
16                 F ← F ∪ {α ∪ β};
17                 α ← α ∪ β;
18 return F
```

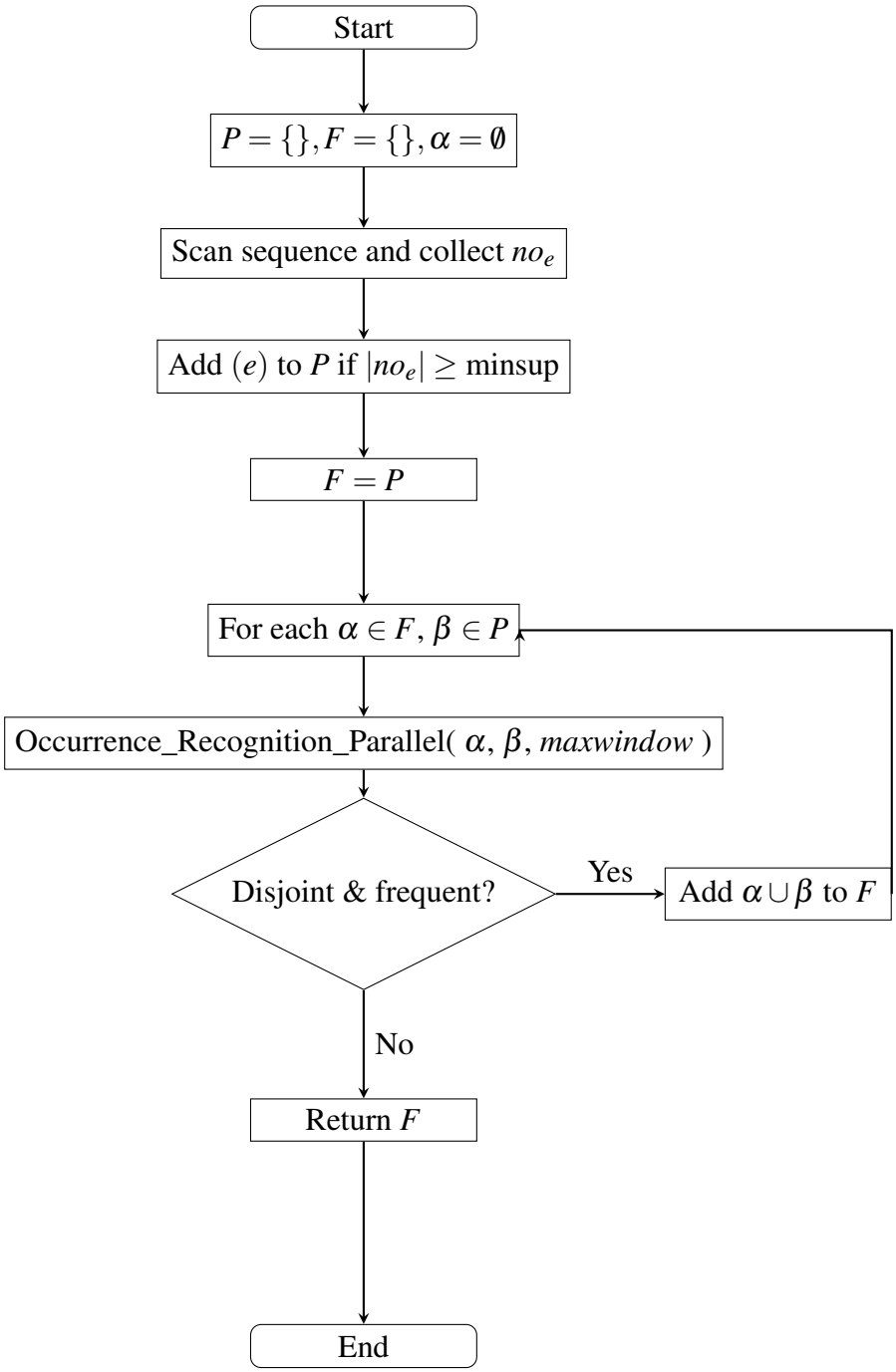


Figure 3.2: Flowchart of Mining Frequent Episodes

3.5.2 Occurrence Recognition

The algorithm shown in Figure 3.3 represents the Occurrence Recognition function used in the original NONEPI.

Input: episode α - an episode to grow
episode β - a single event episode to grow α by.
Output: $no_{\alpha \sqcup \beta}$ - set of non-overlapped occurrences of new episode $\alpha \sqcup \beta$

2 **for** *each* $O_i \in no_\alpha$ **do**
4 **for** *each* $O_j \in no_\beta$ **do**
6 **if** $O_j.start > O_i.end$ **then**
8 update $O_i.end = O_j.start$
10 $O_i.timestamps = O_i.timestamps \cup O_j.start$
12 **for** *each* O_k *in* no_α *s.t.* $i < k \leq m \wedge O_k.start < O_i.end$ **do**
14 remove O_k from no_α
16 $no_{\alpha \sqcup \beta} = no_{\alpha \sqcup \beta} \cup O_i$
18 **return** $no_{\alpha \sqcup \beta}$

Figure 3.3: Occurrence Recognition procedure used in the NONEPI algorithm.

We now describe how the proposed Algorithm , the *Occurrence Recognition* procedure, is performed. This function takes as input an episode α to be extended, a single event episode β, and a maximum time window constraint `maxwindow`. Its goal is to identify all occurrences where α and β co-occur within the specified time window, and to return the set of merged occurrences $no_{\alpha \sqcup \beta}$.

The algorithm begins by initializing the output set $no_{\alpha \sqcup \beta}$ to empty. Two indices, i and j, are used to iterate through the list of occurrences of α and β, respectively (lines 4–5). At each step, the algorithm retrieves the current occurrences I_1 from no_α and I_2 from no_β.

It then computes the maximum of the start times and the minimum of the end times of these intervals. If the intervals not overlap (i.e., $end\_min \geq start\_max$) or the absolute difference between their start times is less than or equal to `maxwindow`, then they are considered to cooccur (lines 8–10). In this case, a new occurrence is formed by merging the intervals: the start time is set to the earlier of the two, and the end time to the later (lines 11–12). Then this merged interval is added to $no_{\alpha \sqcup \beta}$.

To progress through the sequences efficiently, the pointer corresponding to the earlier ending interval is incremented (line 14). If the intervals do not satisfy the co-occurrence condition, the pointer corresponding to the earlier starting interval is incremented instead (lines 16–18).

This process continues until all candidate occurrences have been evaluated. The resulting set $no_{\alpha \sqcup \beta}$ contains all valid merged intervals where α and β cooccur within the allowed time window as it is shown in Algorithm 2.

Algorithm 2: Occurrence Recognition Parallel

Input: α: An episode to grow

β: A single event episode to combine with α

maxwindow: Maximum time window for allowing two events to co-occur

Output: $no_{\alpha\cup\beta}$: Set of occurrences where α and β co-occur within *maxwindow*

```
1  no_{α∪β} ← ∅;
2  i ← 0;
3  j ← 0;
4  while i < |no_α| and j < |no_β| do
5      I_1 ← no_α[i];
6      I_2 ← no_β[j];
7      start_max ← max(I_1.start, I_2.start);
8      end_min ← min(I_1.end, I_2.end);
9      if end_min ≥ start_max or |I_1.start − I_2.start| ≤ maxwindow then
10         merged_start ← min(I_1.start, I_2.start);
11         merged_end ← max(I_1.end, I_2.end);
12         Create new occurrence O with (merged_start, merged_end);
13         Add O to no_{α∪β};
14         if I_1.end < I_2.end then
15             i ← i + 1;
16         else
17             j ← j + 1;
18     else
19         if I_1.start < I_2.start then
20             i ← i + 1;
21         else
22             j ← j + 1;
23 return no_{α∪β}
```

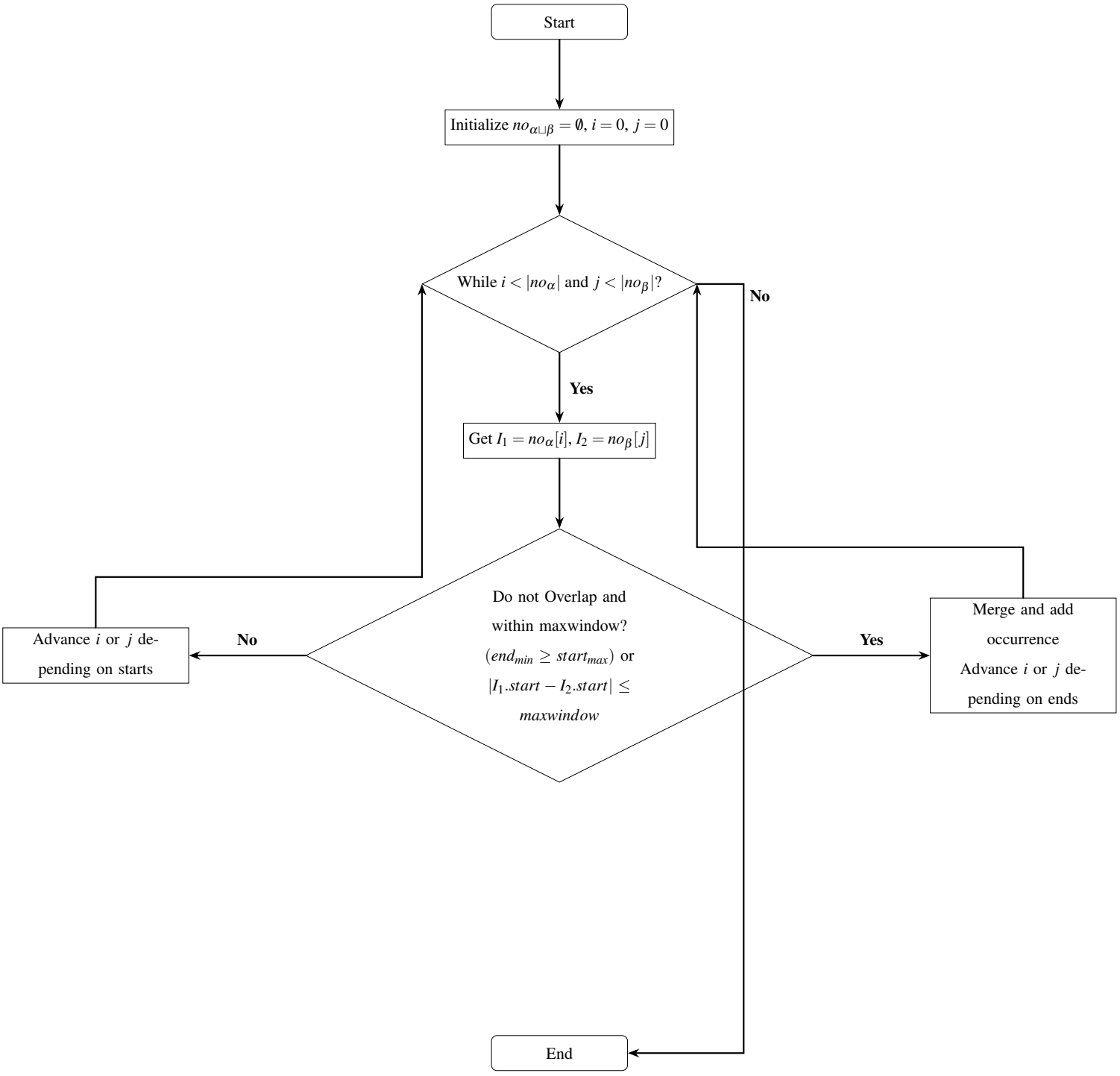


Figure 3.4: Flowchart of Occurrence Recognition Algorithm

3.5.3 Extracting episode rules

Figure 3.5 shows the final step of the original NONEPI algorithm, which is rule extraction.

Input: S: sequence of events on E (event types set), *minsup*: support threshold,
minconf: confidence threshold
Output: R: complete set of episode rules.

2 $F = FrequentEpisodes(minsup)$

4 $R = \emptyset$

6 **for** *each episode* $\alpha \in F$ **do**

8 $stop$ = false

10 $\beta = pred(\alpha)$/* returns the predecessor of episode α*/

12 **while** ***not*** $stop$ ***and*** $\beta \neq NULL$ **do**

14 **if** $\frac{|no_\alpha|}{|no_\beta|} \geq minconf$ **then**

16 $R = R \cup \{\beta \Rightarrow \alpha\}$

18 $\beta = pred(\beta)$

19 **else**

21 stop = true

23 **return** R

Figure 3.5: Rule extraction phase of the original NONEPI algorithm.

The last proposed algorithm takes as input the set F of frequent parallel episodes produced by the mining phase, and a minimum confidence threshold `minconf`. The goal is to construct a set R of association rules of the form $\beta \Rightarrow \alpha$, where both α and its subepisode β are frequent, and the rule meets the confidence requirement.

The set of rules R is initially empty. The algorithm iterates over each frequent episode $\alpha \in F$ (line 4). For each such episode, it attempts to generate rules by identifying a proper sub-episode β, which is obtained via the function `pred`(α) that returns a predecessor of α by removing one event (line 6).

Although there is a valid predecessor β and the stopping condition has not been met, the algorithm proceeds to compute the support values for β and α (lines 8–9). If β has nonzero support and confidence, computed as the ratio `support`$_\alpha$ / `support`$_\beta$, meets or exceeds the given threshold `minconf`, then the rule $\beta \Rightarrow \alpha$ is considered valid and added to the rule set R (lines 10–11). The algorithm then attempts to generate deeper rules by continuing with the next predecessor of β.

If the confidence threshold is not satisfied or β is not frequent, the process for that episode

is halted (lines 12–13). This ensures that only rules with sufficient confidence are retained.

Once all candidate episodes have been processed, the algorithm returns the final set R containing all the confident parallel episode rules derived from the set of frequent episodes as it is shown in Algorithm 3.

Algorithm 3: Generate Parallel Episode Rules

Input: F: List of frequent parallel episodes (from mining algorithm)
$minconf$: Minimum confidence threshold
Output: R: Set of confident parallel episode rules

```
1  R ← ∅;
2  foreach episode α ∈ F do
3      stop ← false;
4      β ← pred(α);
5      while stop = false and β ≠ NULL do
6          support_β ← support of β in F;
7          support_α ← support of α in F;
8          if support_β > 0 and support_α / support_β ≥ minconf then
9              R ← R ∪ {β ⇒ α};
10             β ← pred(β);
11         else
12             stop ← true;
13 return R
```

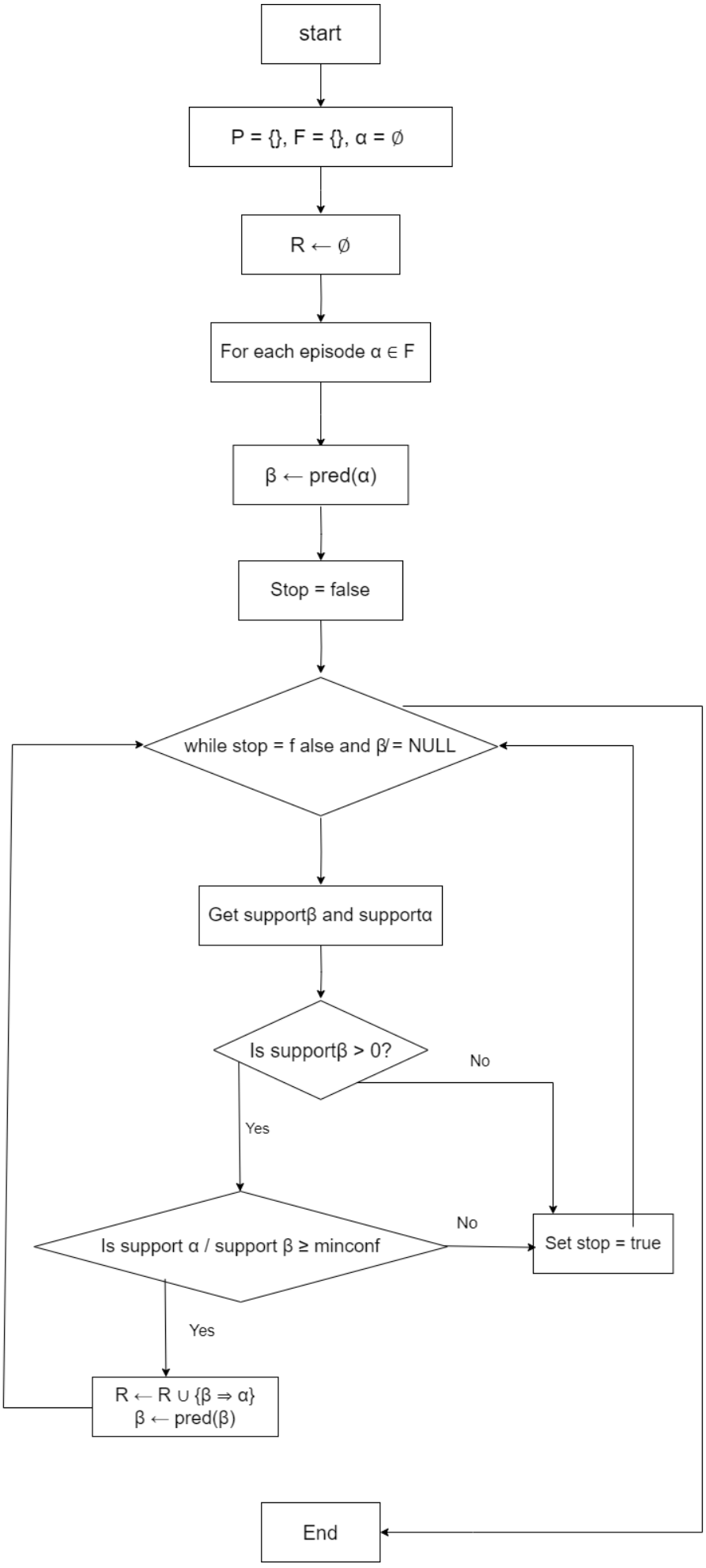


Figure 3.6: Flowchart of Generate Parallel Episode Rules

3.6 Applications of NONEPI-P Algorithms

The NONEPI-P algorithm is an extension that finds parallel episodes groups of events that happen together or within a short time window, regardless of their order. This is ideal when simultaneous or closely timed events are significant.

3.6.1 Real-World Applications & Examples

- **Smart Home Automation**

  Application: Detecting common sets of device activations.

  Example: Every evening, the living room lights, TV, and heating system are turned on within five minutes of each other. NONEPI-P can identify this routine, enabling the system to automate these actions for user convenience.

- **Cybersecurity & Intrusion Detection**

  Application: Spotting coordinated attacks or abnormal system behavior.

  Example: Multiple failed login attempts, firewall alerts, and unusual network traffic all occur within a short period. NONEPI-P can flag this as a potential cyberattack for immediate investigation.

- **Retail Market Analysis**

  Application: Finding products often bought together in the same shopping session.

  Example: Customers frequently purchase bread, milk, and eggs during one visit. NONEPI-P uncovers these associations, helping stores design effective promotions and shelf layouts.

- **Environmental Monitoring with IoT Sensors**

  Application: Detecting environmental changes through simultaneous sensor triggers.

  Example: In a smart city, several air quality sensors detect high pollution levels, and noise sensors register increased sound at the same time. NONEPI-P helps city managers respond quickly to environmental hazards.

3.7 Limitations of the NONEPI-P Algorithm

While the proposed NONEPI-P algorithm demonstrates promising results in mining parallel episodes with improved efficiency and accuracy, it also presents certain limitations that should be acknowledged:

- **Sensitivity to Parameter Settings:** The algorithm depends heavily on carefully chosen values for minimum support (`minsup`), confidence (`minconf`), and the maximum time window. Poor parameter tuning may lead to either too few or too many patterns being discovered, affecting the quality of results.
- **Scalability for Very Large Datasets:** Although NONEPI-P is optimized compared to the original NONEPI, processing very large datasets with millions of events or high event-type diversity can result in increased memory usage and longer computation times.
- **Assumption of Uniform Time Gaps:** The algorithm uses a fixed-size time window to detect parallel episodes. This may not effectively capture patterns where the timing between events varies significantly across user sessions.

Despite these limitations, the NONEPI-P algorithm remains a valuable tool for mining parallel episodes and provides a strong foundation for future improvements.

3.8 Conclusion

In summary, the development of the NONEPI-P algorithm marks a significant contribution to the field of episode mining. By introducing parallel processing capabilities, NONEPI-P effectively addresses the limitations of the original NONEPI algorithm, enabling the efficient discovery of frequent parallel episodes in large event sequences. Experimental results demonstrate that NONEPI-P not only improves computational performance but also broadens the range of detectable patterns by relaxing event order constraints. This advancement provides a robust foundation for future research and applications in large-scale temporal data analysis.

Chapter 4

Implementation And Experimental Study

4.1 Introduction

We present in this chapter the technical and experimental aspects of the project. It includes the implementation environment, the tools and libraries used, and the results of the experimental evaluation. The goal is to show how the proposed method was implemented and how effective it is at mining frequent episodes and extracting episode rules from a real dataset.

4.2 Implementaion

This section describes the practical implementation of the algorithms used in this study, including both existing and modified versions. We explain the development environment, the tools used, the data preparation steps, and the transformation process required to apply each algorithm. Particular focus is given to the implementation of our proposed NONEPI-P algorithm for parallel episode mining.

4.2.1 Tools

4.2.1.1 Material

The experiments were performed on an Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz 2.00 GHz with Radeon Graphics 3.70 GHz PC with 16 Gb of main memory and 512 Gb of SSD storage, running the Microsoft Windows 11 pro operating system. All algorithms were

coded in Java.

4.2.1.2 Software

- **Eclipse**: Eclipse is an open-source development platform written in Java and composed of a host of projects and subprojects. The founding project, Eclipse Platform, constitutes the core and basic components of Eclipse. It is extensible through the addition of additional plugins.[33]
- **java**: Oracle Java is the first programming language and development platform. It reduces costs, shortens development timeframes, drives innovation, and improves application services. Java continues to be the development platform of choice for enterprises and developers.[34]
- **SPMF**: is an open-source software and data mining library written in Java, specialized in pattern mining (the discovery of patterns in data). It offers implementations of 262 data mining algorithms for:
  - Association rule mining,
  - Itemset mining,
  - Sequential pattern mining,
  - Sequential rule mining,
  - Sequence prediction,
  - Periodic pattern mining,
  - Episode mining,
  - High-utility pattern mining,
  - Time-series mining,
  - Clustering and classification,
  - Data processing and visualization.

  SPMF can be used as a standalone program with a simple user interface or from the

command line. Moreover, the source code of each algorithm can be easily integrated into other Java software. In addition, it is possible to call SPMF from other programming languages such as Python, R, and C#. SPMF is fast and lightweight, requiring no dependencies on external libraries.[35]

- **JupyterLab**: JupyterLab is the latest web-based interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. A modular design invites extensions to expand and enrich functionality.[36]

- **Excel**: Microsoft Excel is a spreadsheet software developed by Microsoft. It allows users to organize, analyze, and visualize data in tabular form. Excel is commonly used for data entry, statistical analysis, creating charts and graphs, and performing complex calculations using built-in functions and formulas. In our project, Excel was used to preprocess the dataset and interpret the experimental results generated by the SPMF algorithms.[37]

- **our experiment**: The software implementation was carried out using Eclipse IDE version 2022 with JDK 17. The SPMF data mining library was used to perform episode mining, utilizing several algorithms such as MinePI, EMMA, MinePI+, and NONEPI. These algorithms were applied to our dataset to extract several informations . The resulting outputs were then visualized using Microsoft Excel to generate comparative performance graphs.

4.3 The implementation steps

In this section, we present the different steps involved in the implementation of our algorithm, as shown in the figure 4.1.

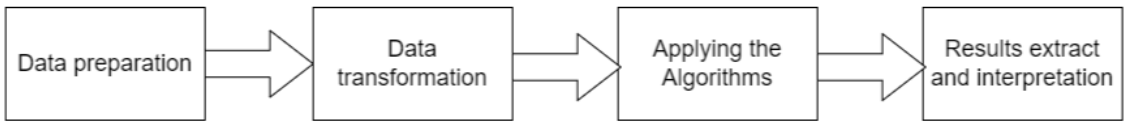


Figure 4.1: Steps of the implementation

4.3.1 Understanding Web User Behavior

In this work, we analyze user behavior based on real-world data from the Kaggle Click Prediction Competition. The dataset contains sequences of user click events recorded with timestamps, reflecting how users interact with a web platform over time.

Each sequence represents a user session composed of various actions such as viewing ads or clicking links. These sessions are used as input for episode mining algorithms to discover frequent behavior patterns. This analysis helps in understanding user navigation habits and supports applications such as prediction and recommendation.

4.3.2 Data preparation

The dataset that we used in this work originates from a public competition and represents user interactions with content recommendations on U.S. publisher websites over two weeks in June 2016. It is composed of several relational tables, among which three primary files were utilized: `page_views_sample.csv`, `events.csv`, and `clicks_train.csv`.

1. **page_views_sample.csv :** file serves as the core of the dataset, capturing user activity logs related to documents. It provides granular insights into user behavior and traffic patterns.
    - Uuid
    - document_id
    - Timestamp
    - Platform
    - geo_location
    - traffic_source
2. **events.csv :** provides additional contextual information about the display_id values present in the clicks_train.csv datasets. This dataset helps enrich the understanding of the environment or session in which ads were displayed and interacted with.
    - Display_id

- Uuid
- document_id
- Timestamp
- Platform
- geo_location

3. **clicks_train.csv :** is a training dataset that tracks user interactions with ads. It logs which ads were clicked within sets of recommendations presented to users, helping train predictive models for click likelihood.
    - Display_id
    - ad_id
    - Clicked

From the `page_views_sample.csv` file, only relevant columns such as `uuid`, `document_id`, `platform`, `traffic_source`, and `geo_location` were extracted to capture user activity details. The `events.csv` file contributed contextual information specifically `uuid`, `display_id`, and `timestamp` allowing the identification of which content was displayed to which user and when. The `clicks_train.csv` file was filtered to retain only the records where a click actually occurred (`clicked = 1`), keeping the `display_id`, `ad_id`, and `clicked` columns.

The preprocessing involved two key merging operations:

1. First, the user-level activity data from `page_views_sample.csv` was merged with the display context from `events.csv` on the common `uuid` field, linking each user's behavior to the content display events.
2. Second, the resulting data was merged with the filtered click data from `clicks_train.csv` on the `display_id` field, thereby associating each session with the clicked advertisements.

These merges ensured that the final dataset only included user sessions with confirmed interactions. After the merging step, the resulting dataset was further cleaned using Jupyter

Notebook by removing rows with missing values and eliminating duplicates. The result shown in figure 4.2 was a clean and structured dataset ready for pattern mining analysis.

	uuid	document_id	platform	traffic_source	geo_location	display_id	timestamp	ad_id	clicked
2	33df12c63d81	1949	1	1	US>TX	73837	6160216	23413	1
3	465fa0b07a73	1949	3	2	CA>QC	11237261	747929885	378990	1
4	bf343c383f8d	2258	1	1	US>MN>613	5559829	370459144	105610	1
5	4fc651b71755	2258	1	2	US>IL>602	9691185	653961216	257743	1
6	394b2182067	2258	2	2	US>TX>623	5777131	384526022	198748	1
7	6bda48bfa4ab	2258	2	2	US>AZ>789	7123440	489609392	41760	1
8	8fde19c27199	3464	1	1	US>GA>524	14995541	993287918	204447	1
9	f785c2aa914e	4491	3	1	AU>07	1382805	82544941	177268	1
10	e6fb209c6c7f	5410	1	2	US>WA>819	1262377	75794328	171667	1
11	e6fb209c6c7f	5410	1	2	US>WA>819	6197832	413069104	4700	1
12	e6fb209c6c7f	5410	1	2	US>WA>819	8826826	593883626	107020	1
13	a864543c424	6712	1	2	US>TX>623	842890	54352124	355908	1
14	27f6493b3f4c	7012	1	1	US>GA>524	875903	55843326	96381	1
15	7fdcf7866ba4	7012	1	2	US>CA>825	15280996	1012611673	332908	1
16	6029bea607b	7012	1	2	MY>01	2697936	170158871	301835	1
17	a1c9d67eaa5	7012	1	2	US>TX>623	2039511	132739091	231376	1
18	4be8b5da8ec	7012	1	1	BE>03	9405932	639448055	329239	1
19	e2d28e766fd3	7012	1	1	GB>H9	192268	20549394	68651	1
20	ce95458032a	7012	1	2	ZA>05	115906	10617423	158220	1

Figure 4.2: Screenshot of the Dataste after preparation

Our data informations :

Table 4.1: Dataset Overview

Property	**Details**
Data size	1,117,000 lines
Columns	• uuid • document_id • display_id • ad_id • clicked • timestamp • platform • geo_location • traffic_source
Category	Symbols and numbers

4.3.3 Python Script Use for merging the Dataset:

The Python script that shown in figure 4.3 was used to merge the data files.

```python
import pandas as pd

page_views_sample = pd.read_csv(r"C:\Users\admin\Desktop\page_views_sample.csv\page_views_sample.csv")
events = pd.read_csv(r"C:\Users\admin\Desktop\events.csv\events.csv")
clicks_train = pd.read_csv(r"C:\Users\admin\Desktop\clicks_train.csv\clicks_train.csv")
page_views_sample_filtered = page_views_sample[['uuid', 'document_id', 'platform', 'traffic_source', 'geo_location']]

events_filtered = events[['uuid', 'display_id', 'timestamp']]
merged_data = pd.merge(page_views_sample_filtered, events_filtered, on='uuid', how='inner')
clicks_train_filtered = clicks_train[clicks_train['clicked'] == 1][['display_id', 'ad_id', 'clicked']]

final_data = pd.merge(merged_data, clicks_train_filtered, on='display_id', how='inner')
final_data.to_csv('episode_mining.csv', index=False)

print("Data aggregation completed and saved to episode_mining.csv")
```

Figure 4.3: Python code for merge the data files

s

4.3.4 Data Transformation for the algorithms

To apply various episode mining algorithms available in the SPMF library including NONEPI, EMMA, MINEPI+, and our proposed NONEPI-P it was essential to preprocess and format the dataset according to each algorithm's input requirements.

The original dataset was obtained from the Outbrain Click Prediction dataset on Kaggle. After a comprehensive cleaning process that involved filtering for relevant columns and removing incomplete or duplicate records, only the `ad_id` and associated `timestamp` were retained for the mining tasks. The cleaned data was then transformed into specific formats tailored to the needs of each algorithm, ensuring compatibility and optimal performance during execution.

The dataset used for this study consists of 1,117,000 event entries, distributed across 71,184 unique sequences. These sequences represent distinct user navigation paths or sessions. In total, the dataset includes 977,186 unique event types, indicating a high degree of variability and complexity in the recorded events. Table 4.2 summarizes the key characteristics of the dataset used in the experiments.

Table 4.2: Details of the dataset used

Data Size	1,117,000 lines
Number of event types	977,186
Total number of sequences	71,184

4.3.4.1 Format for the Algorithms:

The algorithms accept identical input structures. Each event line contains:

- **Item IDs**: Space-separated numerical identifiers (may include duplicates)
- **Delimiter**: Pipe character | separating items from timestamp
- **Timestamp**: Numerical value representing event timing (right-aligned)

An example of this format is as follows:

```
event | timestamp
273567|1033
139944|2550
234713|2687
269739|3157
220854|3891
215554 215554 215554|3925
98204|3938
229669|3963
86683 86683|4242
92759|4651
52574|5182
173452|5415
173130|5723
92113|5793
```

Figure 4.4: Screenshot of the Dataste format

In this structure:

- The item(s) on the left of the pipe character | represent events that occurred together at the same timestamp.

- The number after the pipe | represents a simplified or scaled timestamp to preserve ordering.

A Python script was used to automate this transformation, grouping events and formatting them according to the algorithm's expectations.

Python Script Use for Converting the Dataset:

The following Python script hat shown in figure 4.5 was used to convert the cleaned dataset into the input format required by the selected episode mining algorithms (NONEPI, EMMA, MINEPI+,and NONEPI-P). The script automates the transformation by grouping events by timestamp for algorithms style input,

```python
import pandas as pd
from collections import defaultdict

# Load your dataset
df = pd.read_csv("episode_mining_cleaned.csv")

# Keep only the columns we need
df_filtered = df[['timestamp', 'ad_id']]

# Group ad_ids by timestamp
timestamp_to_ads = defaultdict(list)
for _, row in df_filtered.iterrows():
    timestamp = row['timestamp']
    ad_id = row['ad_id']
    timestamp_to_ads[timestamp].append(ad_id)

# Create EMMA format: "event1 event2 ... | timestamp"
lines = []
for timestamp in sorted(timestamp_to_ads.keys()):
    events = " ".join(str(e) for e in sorted(timestamp_to_ads[timestamp]))
    lines.append(f"{events}|{timestamp}")

# Save to a text file
with open("emma_formatted_input.txt", "w") as f:
    f.write("\n".join(lines))
```

Figure 4.5: Python code for converting data to the input format

4.3.5 Applying the Algorithms

After preparing the dataset in the required formats, we executed several episode mining algorithms using the SPMF data mining library. The algorithms applied in this study include NONEPI, EMMA, MINEPI+, and our modified version, NONEPI-P. Each algorithm is designed to discover frequent episodes from sequences of timestamped events, though they differ in the types of episodes they extract and their underlying processing logic.

The original NONEPI algorithm was executed without modification and is specialized in mining serial episodes, where the order of events strictly follows a sequence. In contrast, our proposed NONEPI-P algorithm was developed by modifying the original NONEPI source code to mine parallel episodes, allowing events to occur in any order within a specified time window.

All algorithms were implemented in Java 17 and executed within the Eclipse IDE. Experiments were conducted on the same dataset using a range of minimum support thresholds: 500, 600, 800, 1000, and 1500. To maintain temporal consistency in episode detection, the maximum window size was fixed at 100,000 units for all runs.

For each execution, we collected and analyzed key metrics including the number of frequent episodes discovered, execution time, and output size. This comprehensive evaluation enabled a fair comparison of the algorithms under identical conditions, shedding light on the relative strengths and trade-offs between serial and parallel episode mining approaches when applied to real-world event sequence data.

4.3.6 Results Interpretation

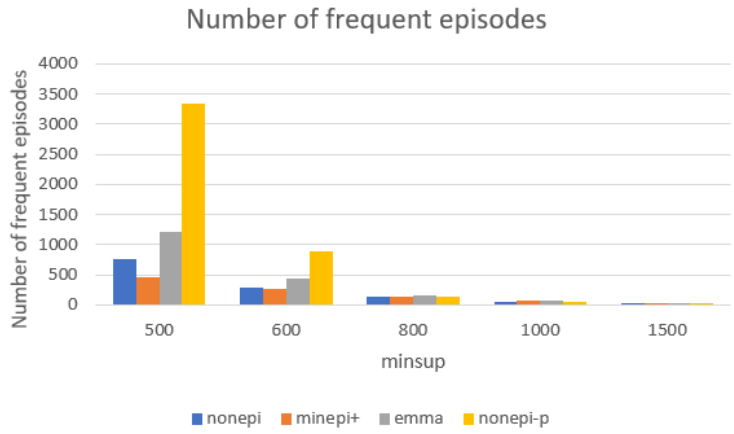


Figure 4.6: The effect of minsup on the amount of frequent episodes.

Table 4.3: Number of Frequent Episodes Discovered by Different Algorithms at Various minSup Thresholds

NoNEPI-P	NoNEPI	Minepi+	Emma	minSup
3339	760	463	1203	500
891	287	273	428	600
147	147	128	151	800
50	50	71	75	1000
12	12	30	30	1500

Figure 4.6 illustrate the impact of varying the minimum support threshold (`minsup`) on the number of frequent episodes discovered by the algorithms:

- At `minsup` = 500, **NONEPI-P** identifies 3339 episodes, significantly more than the other algorithms.
- As `minsup` increases, the number of discovered episodes decreases for all algorithms, reflecting fewer qualifying patterns at stricter thresholds.
- At `minsup` = 1000, **NONEPI-P** still finds 50 frequent episodes, while **NONEPI** finds only 34.
- At `minsup` = 1500, all algorithms converge and find between 12 and 30 episodes, representing the most robust patterns.
- The largest gain is observed at moderate support values, where **NONEPI-P** consistently detects more patterns than its serial counterpart.

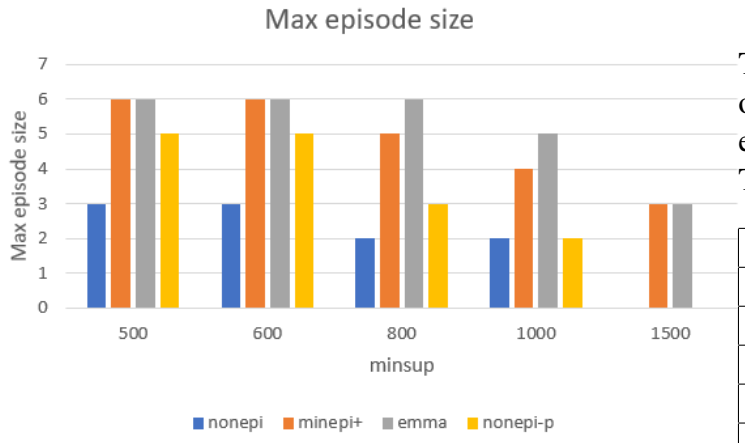


Figure 4.7: The effect of minsup on max episode size.

Table 4.4: Maximum Size Values of Patterns Discovered by Different Algorithms at Various minSup Thresholds

NoNEPI-P	NoNEPI	Minepi+	Emma	minSup
5	3	6	6	500
5	3	6	6	600
3	2	5	6	800
2	2	4	5	1000
0	0	3	3	1500

Discussion

Figure 4.7 compares the maximum episode size the number of events in the largest discovered episode for each algorithm across varying `minsup` levels:

- At `minsup` = 500, **EMMA** and **MINEPI+** discover the largest episodes (size 6), followed by **NONEPI-P** (size 5), consistent with their respective abilities to find long minimal or parallel patterns.
- **NONEPI** detects the smallest episodes (maximum size 3 at `minsup` = 500), reflecting its serial and non-overlapping approach.
- As `minsup` increases, maximum episode sizes drop across all algorithms, converging to 3 or fewer at `minsup` = 1500, highlighting that longer episodes are typically less frequent.

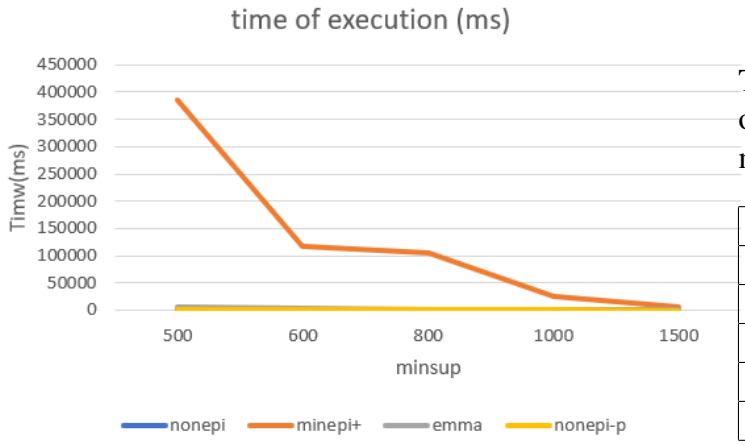


Figure 4.8: The effect of minsup on the execution time.

Table 4.5: Execution Time (in ms) of Different Algorithms at Various minSup Thresholds

NoNEPI-P	NoNEPI	Minepi+	Emma	minSup
940	62	385425	4798	500
79	40	1116840	2501	600
22	13	104583	1334	800
16	0	25594	1104	1000
0	0	6812	959	1500

Discussion

Figure 4.8 shows the execution time (in milliseconds) of each algorithm as a function of the `minsup`:

- **NONEPI** consistently exhibits the lowest execution time due to its simple serial mining strategy.
- **NONEPI-P** requires more computation, especially at lower `minsup` (e.g., 940 ms at 500), due to the additional cost of parallel episode detection.
- **MINEPI+** incurs the highest execution time, particularly at lower thresholds (e.g., over 385,000 ms at 500), owing to its handling of partial orders and complex pruning.
- **EMMA** also has a high execution time, driven by its requirement to find minimal occurrences.
- All algorithms show decreased execution time as `minsup` increases, reflecting fewer candidates to evaluate.

4.3.7 Extracting the rules

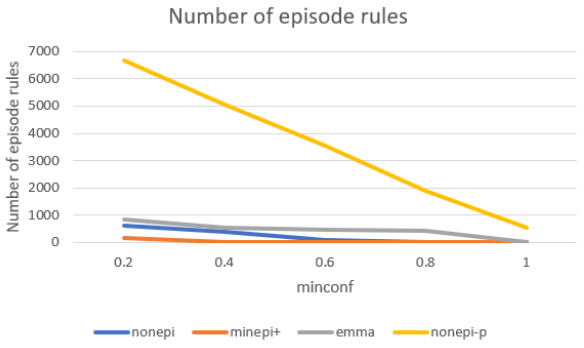


Figure 4.9: Influence of minconf on number of episode rules with $minsup = 500$.

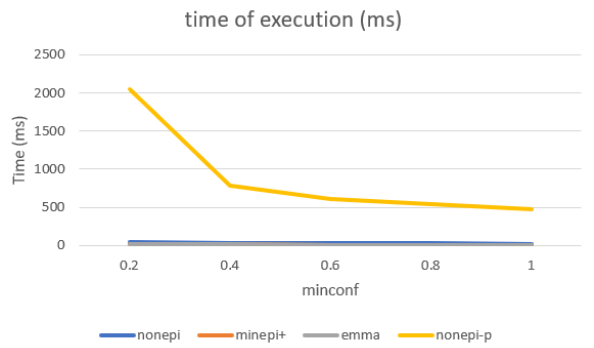


Figure 4.10: Influence of minconf on execution time with $minsup = 500$.

Discussion

in here we compares the performance of four algorithms in comparison is based on two main metrics:

- Number of episode rules generated (Figure4.9)

- Execution time (Figure 4.10)

These metrics were evaluated at different confidence thresholds (`minconf` values): 0.2, 0.4, 0.6, 0.8, and 1.0.

1. Number of Episode Rules

As shown in Figure (4.9), all algorithms generate fewer rules as the `minconf` value increases. This is expected, as higher confidence thresholds retain only stronger and more reliable rules.

Our algorithm, **NONEPI-P**, clearly produces the highest number of episode rules across all thresholds. This is due to its use of parallel episodes, which allows it to detect a larger variety of event combinations that occur together. In contrast, the other algorithms which focus on serial or partially ordered episodes return fewer rules, particularly as the confidence threshold rises.

2. Execution Time

The second Figure (4.10) illustrates that **NONEPI-P** also has the highest execution time. This outcome is expected due to the significantly larger number of rules it generates. The more rules an algorithm discovers, the more processing time it requires.

The other algorithms show much lower execution times especially **MINEPI+** and **EMMA**, which remain efficient even at lower confidence values. **NONEPI** is somewhat slower than these two but still faster than **NONEPI-P**.

This comparison highlights a clear trade-off. **NONEPI-P** offers richer results by discovering more patterns through parallel episodes but at the cost of increased execution time. In contrast, the other algorithms provide faster performance with fewer but more concise patterns.

4.3.7.1 Rules Discovered

Table 4.6: Sample of 10 Episode Rules Discovered by NONEPI-P

#	**Rule** $\gamma \Rightarrow \delta$	**conf**($\gamma \Rightarrow \delta$)
1	{99215, 88761} ⇒ {99215, 88761, 26711}	100%
2	{99215, 173130, 151028} ⇒ {99215, 173130, 151028, 26711}	100%
3	{273567, 88761} ⇒ {273567, 88761, 151028}	100%
4	{273567, 88761, 151028} ⇒ {273567, 88761, 151028, 26711}	100%
5	{92759, 269739, 66484} ⇒ {92759, 269739, 66484, 332844}	100%
6	{92759, 151028, 66484} ⇒ {92759, 151028, 66484, 332844}	100%
7	{92759, 193952} ⇒ {92759, 193952, 332844}	100%
8	{175214, 269739, 173130} ⇒ {175214, 269739, 173130, 332844}	100%
9	{319252, 173130} ⇒ {319252, 173130, 26711}	100%
10	{269739, 66484} ⇒ {269739, 66484, 332844}	100%

Discussion

Table 4.6 presents a representative selection of 10 rules mined by the NONEPI-P algorithm from a total of 574 rules extracted using a minimum support of 500 and MINCONF of 1. These rules demonstrate consistently high support and confidence values, reflecting robust and frequent parallel episode occurrences within the dataset. The antecedents and consequents typically differ by a single event, illustrating the incremental nature of pattern growth and the strong temporal or contextual associations among events. The recurrence of specific event IDs across multiple rules highlights key elements driving the system's behavior. By applying stringent thresholds, the algorithm effectively filters out less significant patterns, ensuring that the resulting rules are both reliable and interpretable. Overall, this subset exemplifies the algorithm's capability to uncover meaningful, high-confidence parallel episodes, providing valuable insights into the underlying event dynamics.

4.4 Conclusion

In this chapter, we described the implementation and experimental study of episode mining algorithms, focusing on our NONEPI-P algorithm. We detailed the technical environment, including hardware and software, and explained the steps taken for data preparation and transformation.

Through a series of experiments on real-world dataset, we evaluated the performance of NONEPI-P compared to traditional algorithms. The results showed that NONEPI-P offers sig-

nificant improvements in execution time and can handle larger datasets more efficiently. Additionally, the algorithm successfully discovered more relevant and meaningful episode rules, especially when varying parameters such as minimum support and confidence.

Overall, the experimental results confirm the effectiveness and scalability of NONEPI-P for mining frequent episodes in sequential data. These findings demonstrate the potential of our approach for practical applications, such as analyzing web user behavior and supporting decision-making based on sequential patterns.

General Conclusion

In this dissertation, we introduced the fundamental concepts and motivations behind data mining, with a particular emphasis on pattern mining and its specialized subfield, episode mining. We outlined the key objectives, techniques, and challenges associated with mining useful patterns from large and complex datasets. As discussed, episode mining plays a crucial role in analyzing event sequences by identifying frequent patterns that capture temporal relationships between events either in a strict (serial) or flexible (parallel) order.

We also introduced the theoretical foundations and practical significance of frequent episode mining, setting the stage for the algorithmic and experimental studies. Among the various algorithms designed for episode mining, the NONEPI approach has drawn attention due to its ability to discover meaningful episodes based on occurrence frequency. This project builds on that work by implementing and extending the NONEPI algorithm to support parallel episode mining, thereby enhancing its scope and effectiveness in real-world applications.Practical applications of this approach include analyzing user behavior on websites such as identifying unordered but related actions like viewing a product, adding it to the cart, and performing a search within the same session as well as detecting concurrent security events in cybersecurity systems, or recognizing simultaneous sensor readings in smart environments that may indicate equipment failure or abnormal conditions.

We also present the results of experimental studies, demonstrating how these techniques can be applied to uncover patterns in sequential data and offering insights into the impact of different frequency measures and mining strategies.

References

[1] J. Han, J. Pei, and H. Tong, *Data mining: concepts and techniques*. Morgan kaufmann, 2022.

[2] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Morgan Kaufmann, 2012.

[3] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*, 2nd ed. Pearson, 2019.

[4] C. C. Aggarwal, *Outlier Analysis*, 2nd ed. Springer, 2017.

[5] H. Mannila, H. Toivonen, and A. Inkeri Verkamo, "Discovery of frequent episodes in event sequences," *Data mining and knowledge discovery*, vol. 1, pp. 259–289, 1997.

[6] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 3rd ed. Morgan Kaufmann, 2011.

[7] P. Fournier-Viger, "A blog about data mining, data science, machine learning and big data." 2020, accessed: 26 Mars 2025. [Online]. Available: https://data-mining.philippe-fournier-viger.com/introduction-data-mining/

[8] S. García, J. Luengo, F. Herrera *et al.*, *Data preprocessing in data mining*. Springer, 2015, vol. 72.

[9] N. Jain and V. Srivastava, "Data mining techniques: a survey paper," *IJRET: International Journal of Research in Engineering and Technology*, vol. 2, no. 11, pp. 2319–1163, 2013.

[10] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI magazine*, vol. 17, no. 3, pp. 37–37, 1996.

[11] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. Morgan Kaufmann, 2016.

[12] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu, "Mining access patterns efficiently from web logs," in *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2000, pp. 396–407.

[13] X. Yan and J. Han, "gspan: Graph-based substructure pattern mining," in *2002 IEEE International Conference on Data Mining, 2002. Proceedings.* IEEE, 2002, pp. 721–724.

[14] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to data mining*. Pearson Education India, 2016.

[15] W. Meira Jr and M. Zaki, "Data mining and analysis," *Fundamental Concepts and Algorithms*, vol. 1, 2014.

[16] O. Ouarem, F. Nouioua, and P. Fournier-Viger, "A survey of episode mining," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 14, no. 2, p. e1524, 2024.

[17] X. Ao, P. Luo, C. Li, F. Zhuang, and Q. He, "Online frequent episode mining," in *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 2015, pp. 891–902.

[18] A. Avinash, A. Ibrahim, and P. Sastry, "Pattern-growth based frequent serial episode discovery," *Data and Knowledge Engineering*, vol. 87, pp. 91–108, 2013, [Online; accessed 2025-06-17]. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0169023X13000724

[19] G. Casas-Garriga, "Discovering unbounded episodes in sequential data," in *Knowledge Discovery in Databases: PKDD 2003*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 83–94.

[20] N. Méger and C. Rigotti, "Constraint-based mining of episode rules and optimal window sizes," in *Knowledge Discovery in Databases: PKDD 2004*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 313–324.

[21] K.-Y. Huang and C.-H. Chang, "Efficient mining of frequent episodes from complex

sequences," *Information Systems*, vol. 33, no. 1, pp. 96–114, 2008, [Online; accessed 2025-06-17]. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437907000506

[22] H. Zhu, L. Chen, J. Li, A. Zhou, P. Wang, and W. Wang, "A general depth-first-search based algorithm for frequent episode discovery," in *2018 14th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*. IEEE, 2018, pp. 890–899.

[23] J. Wu, L. Wan, and Z. Xu, "Algorithms to discover complete frequent episodes in sequences," in *New Frontiers in Applied Data Mining*, L. Cao, J. Z. Huang, J. Bailey, Y. S. Koh, and J. Luo, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 267–278.

[24] X. Ao, P. Luo, C. Li, F. Zhuang, and Q. He, "Discovering and learning sensational episodes of news events," *Information Systems*, vol. 78, pp. 68–80, 2018, [Online; accessed 2025-06-17]. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0306437916303520

[25] T. Katoh, S.-i. Tago, T. Asai, H. Morikawa, J. Shigezumi, and H. Inakoshi, "Mining frequent partite episodes with partwise constraints," in *New Frontiers in Mining Complex Patterns*, A. Appice, M. Ceci, C. Loglisci, G. Manco, E. Masciari, and Z. W. Ras, Eds. Cham: Springer International Publishing, 2014, pp. 117–131.

[26] O. Ouarem, F. Nouioua, and P. Fournier-Viger, "Mining episode rules from event sequences under non-overlapping frequency," in *Advances and Trends in Artificial Intelligence. Artificial Intelligence Practices*, H. Fujita, A. Selamat, J. C.-W. Lin, and M. Ali, Eds. Cham: Springer International Publishing, 2021, pp. 73–85.

[27] P. Fournier-Viger, Y. Chen, F. Nouioua, and J. C.-W. Lin, "Mining partially-ordered episode rules in an event sequence," in *Intelligent Information and Database Systems*, N. T. Nguyen, S. Chittayasothorn, D. Niyato, and B. Trawiński, Eds. Cham: Springer International Publishing, 2021, pp. 3–15.

[28] H. Li, S. Peng, J. Li, J. Li, J. Cui, and J. Ma, "Once and once+: Counting the frequency of time-constrained serial episodes in a streaming sequence," *Information Sciences*,

vol. 505, pp. 422–439, 2019, [Online; accessed 2025-06-17]. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0020025519307169

[29] S. Laxman, P. Sastry, and K. Unnikrishnan, "A fast algorithm for finding frequent episodes in event streams," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2007, pp. 410–419.

[30] K.-Y. Lin, C.-H. Yeh, and P. S. Yu, "Emma: Efficient mining of frequent episodes in a complex event sequence," in *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2008, pp. 123–131.

[31] O. Ouarem, F. Nouioua, and P. Fournier-Viger, "Mining episode rules from event sequences under non-overlapping frequency," in *Proceedings of the 34th International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems (IEA/AIE 2021)*, ser. Lecture Notes in Computer Science, vol. 12798. Springer, 2021, pp. 73–85.

[32] P. Fournier-Viger, "Mining episode rules in a complex sequence with the non-overlapping frequency using the nonepi algorithm," *SPMF Data Mining Library Documentation*, 2025, accessed: 26 Mars 2025. [Online]. Available: https://www.philippe-fournier-viger.com/spmf/NONEPI_episode_rules.php

[33] "Eclipse ide," https://www.eclipse.org/, accessed: 2025-06-03.

[34] "Oracle java," https://www.oracle.com/java/, accessed: 2025-06-03.

[35] P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas, "Spmf: A java open-source pattern mining library," *Journal of Machine Learning Research*, vol. 15, pp. 3389–3393, 2014, accessed: 2025-06-03. [Online]. Available: http://www.philippe-fournier-viger.com/spmf/

[36] "Jupyterlab," https://jupyter.org/, accessed: 2025-06-03.

[37] "Microsoft excel," https://www.microsoft.com/en-us/microsoft-365/excel, accessed: 2025-06-03.