Efficiently Mining Temporal Patterns in Time Series Using Information Theory

Ho Long, Van

DOI (link to publication from Publisher):
10.54337/aau617104352

Publication date:
2023

Document Version
Publisher's PDF, also known as Version of record

EFFICIENTLY MINING TEMPORAL PATTERNS IN TIME SERIES USING INFORMATION THEORY

BY
VAN LONG HO

DISSERTATION SUBMITTED 2023

AALBORG UNIVERSITY
DENMARK

Efficiently Mining Temporal Patterns in Time Series Using Information Theory

Ph.D. Dissertation

Van Long Ho

Dissertation submitted August 12, 2023

Dissertation submitted:	August 12, 2023
PhD supervisor::	Professor Torben Bach Pedersen Aalborg University
PhD Co-Supervisor:	Assistant Professor Nguyen Ho Aalborg University
PhD committee:	Associate Professor Alvaro Torralba (chairman) Aalborg University, Denmark Professor Themis Palpanas Université Paris Cité, France Senior LecturerQuoc Viet Hung Nguyen Griffith University, Australia
PhD Series:	Technical Faculty of IT and Design, Aalborg University
Department:	Department of Computer Science

ISSN (online): 2446-1628
ISBN (online): 978-87-7573-656-0

Published by:
Aalborg University Press
Kroghstræde 3
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Stibo Complete, 2023

Abstract

The rapid and persistent development of IoT technology has generated a massive volume of time series data. For example, sensors are deployed in smart city applications to collect time series on air quality, humidity, and temperature. In energy management, IoT-enabled smart grids and smart meters contain time series on energy consumption and distribution. In health monitoring applications, wearable devices, such as medical sensors and fitness trackers, collect time series on sleep patterns, heart rate, and physical activity. These time series contain hidden insights and patterns, and when they are discovered, they can offer valuable information to support forecasting and decision-making.

Temporal pattern mining in time series is an approach that assists in extracting valuable insights. A temporal pattern has two characteristics. First, temporal information is added to each event within a pattern. Second, the pattern is formed by the complex temporal relations between events. The characteristics make the temporal patterns more expressive and comprehensive, enabling them to provide detailed information. However, it is worth noting that these characteristics also increase the complexity of the mining process due to the search space's explosion.

In this thesis, we focus on optimization methods for temporal pattern mining to enhance the efficiency of the mining process. Moreover, we use information theory-based measures, i.e., mutual information and entropy, to estimate the correlation between time series, thereby pruning the uncorrelated time series to reduce the search space. We solve three problems: frequent temporal pattern mining, rare temporal pattern mining, and seasonal temporal pattern mining.

First, we present a comprehensive process for mining frequent temporal patterns from time series. The input of this process consists of a set of time series, while the output comprises all the frequent temporal patterns. As part of this process, we use a splitting strategy that converts time series into event sequences, while preserving the underlying temporal patterns. Our proposal includes an efficient algorithm for Frequent Temporal Pattern Mining, called FTPM, that optimizes the mining process by utilizing efficient data structures and pruning techniques. Additionally, we propose an approximate version of

FTPM that employs mutual information to eliminate unpromising time series. This approximation method proves the efficiency when working with large datasets.

Second, we propose a solution to mine rare temporal patterns from time series. The solution comprises an efficient Rare Temporal Pattern Mining (RTPM) algorithm that incorporates a support lower bound and a support upper bound. These support bounds are assigned to low values that constrain a low occurrence frequency for temporal patterns. Furthermore, we set the confidence threshold to a high value to ensure that the discovered patterns exhibit high confidence. The RTPM algorithm uses an efficient data structure, i.e., a variant of the hierarchical hash table, and applies two pruning techniques based on the Apriori principle and the transitivity property to perform the efficient mining process. Moreover, by establishing the connection between mutual information and support as well as confidence, we put forth an approximate version of RTPM that focuses exclusively on mining rare temporal patterns from the most promising time series, accelerating the mining process while maintaining high accuracy.

Third, we propose the first-ever solution for mining seasonal temporal patterns from time series. In this solution, we introduce several measures to capture the seasonality characteristics of temporal patterns. Additionally, we propose an efficient Seasonal Temporal Pattern Mining (STPM) algorithm including several novelties. The first novelty is we introduce a new measure called a *maximum season*, which adheres to the anti-monotonicity property. We then use the maximum season to define the concept of a *candidate seasonal temporal pattern* that is used to eliminate infrequent seasonal temporal patterns. The second novelty is we use hierarchical hash tables data structures, ensuring fast retrieval of candidate events and patterns, and propose two efficient pruning techniques: Apriori-like pruning and transitivity pruning. To handle large datasets more effectively, we introduce an approximate version of STPM that utilizes mutual information to perform the mining on only the promising time series, speeding up the mining process, while retaining high accuracy.

We evaluate the proposed solutions on real-world and synthetic datasets. For real-world datasets, four smart energy datasets are from Spain, the U.S.A., and the U.K.; one smart city dataset is from the U.S.A.; one American Sign Language dataset is from the U.S.A.; and two health datasets are from Japan. For synthetic datasets, we generate a large number of sequences and time series from each real-world dataset, adapting the generation process based on the problem being addressed. The experimental results show that the exact algorithms (FTPM, RTPM, and STPM) outperform the baselines in terms of runtime and memory usage and scale well on large datasets. Moreover, the approximate FTPM is up to two orders of magnitude, and the approximate RTPM and STPM are up to one order of magnitude, faster than the baselines, while maintaining a high level of accuracy.

Resumé

Den hurtige og vedvarende udvikling af IoT-teknologi har genereret en massiv volumen af tidsseriedata. For eksempel bliver der i smart city-applikationer udrullet sensorer til at indsamle tidsseriedata om luftkvalitet, luftfugtighed og temperatur. Inden for energistyring, IoT-aktiverede smart grid og smart målere indeholder tidsseriedata om energiforbrug og distribution. I sundhedsmonitoreringsapplikationer indsamler bærbare enheder som medicinske sensorer og fitness-trackere tidsseriedata om søvnmønstre, hjertefrekvens og fysisk aktivitet. Disse tidsseriedata indeholder skjulte indsigter og mønstre, som, når de bliver opdaget, kan give værdifuld information til at understøtte prognoser og beslutningstagning.

Temporal mønsterudvinding i tidsserier er en tilgang, der hjælper med at udtrække værdifuld viden. Et temporalt mønster har to karakteristika. For det første, temporal information er tilføjet til hvert event. For det andet dannes mønsteret af de komplekse tidsmæssige relationer mellem begivenhederne. Disse karakteristika gør det temporale mønster mere udtryksfuldt og omfattende, hvilket gør det i stand til at levere detaljeret information. Det er dog værd at bemærke, at disse karakteristika også øger kompleksiteten af udvindingsprocessen på grund af at søgeområdet eksploderer.

I denne afhandling fokuserer vi på optimeringsmetoder til temporal mønsterudvinding for at forbedre effektiviteten af udvindingsprocessen. Derudover anvender vi informationsbaserede mål baseret på informationsteori, f.eks. gensidig information og entropi, til at estimere korrelationen mellem tidsserier. Dette medfører en yderligere forbedring af udvindingsprocessen, da vi kun udfører udvindingsprocessen på de korrelerede tidsserier. Vi løser tre problemer, udvinding af: hyppig temporal mønstre, sjælden temporal mønstre og sæsonbetonet temporal mønstre.

Først præsenterer vi en omfattende proces til udvinding af hyppige temporale mønstre fra tidsserier. Inputtet af denne proces består af en række tidsserier, mens resultatet omfatter alle de hyppige temporale mønstre. Som en del af denne proces anvender vi en opdelingsstrategi, der konverterer tidsserier til begivenhedssekvenser, samtidig med at de underliggende temporale mønstre bevares. Vores forslag inkluderer en effektiv algoritme til Frequent Tempo-

ral Pattern Mining, forkortet FTPM, der optimerer udvindingsprocessen ved at bruge effektive datastrukturer og beskæringsteknikker. Derudover foreslår vi en tilnærmet version af FTPM, der anvender gensidig information til at eliminere ikke lovende tidsserier. Denne tilnærmelsesmetode viser sig at være effektiv, når man arbejder med store datasæt.

Dernæst foreslår vi en løsning til at udvinde sjældne temporale mønstre fra tidsserier. Løsningen omfatter en effektiv algoritme til Rare Temporal Pattern Mining (RTPM), der inkorporerer en støtte nedre grænse og en støtte øvre grænse. Disse støtte grænser er tildelt laverer værdier som begrænser de temporale mønstre der sjælendt forkommer. Derudover anvender vi en tærskel for konfidens for at sikre, at de opdagede mønstre har høj konfidens. RTPM-algoritmen bruger en effektiv datastruktur, dvs. en variant af de hierarkiske hash-tabeller, og anvender to beskæringsteknikker baseret på Apriori-princippet og transitivitetsegenskaben, for at udføre den effektive udvindingsproces. Derudover etablerer vi forbindelsen mellem gensidig information, støtte samt konfidens og præsenterer en tilnærmet version, der fokuserer udelukkende på at udvinde sjældne temporale mønstre fra de mest lovende tidsserier, hvilket fremskynder udvindingsprocessen samtidig med at der opretholdes høj nøjagtighed.

Afsluttende foreslår vi den første løsning til udvinding af sæsonbestemte temporale mønstre fra tidsserier. I denne løsning introducerer vi flere metrikker til at fange sæsonbestemte karakteristika fra temporale mønstre. Derudover foreslår vi en effektiv Seasonal Temporal Pattern Mining (STPM) algoritme, der inkluderer flere nye bidrag. Det første nye bidrag omhandler introduceringen af en ny metrikker kaldet maksimumsæson, der overholder den anti-monotoniske egenskab. Derefter bruger vi maksimumsæsonen til at definere begrebet sæsonbestemt temporal mønsterkandidat, der bruges til at eliminere sjældne sæsonbestemte temporale mønstre. Et andet nyt bidrag er, at vi bruger hierarkiske hash-tabeller som datastruktur, hvilket sikrer hurtige opslag af kandidatbegivenheder og mønstre. Vi foreslår to effektive beskæringsteknikker: Apriori-lignende beskæring og transitivitetsbeskæring. For at håndtere store datasæt mere effektivt introducerer vi en tilnærmet version af STPM, der bruger gensidig information til at udføre udvindingen kun på de lovende tidsserier, hvilket forbedre udvindingsprocessens køretid, mens nøjagtivheden vedligeholdes.

Vi evaluerer de foreslåede løsninger på virkelige og syntetiske datasæt. Blandt datasættene fra den virkelige verden, stammer fire smarte energidatasæt fra Spanien, USA og Storbritannien; ét smart city-datasæt stammer fra USA; ét amerikansk tegnsprog-datasæt stammer fra USA; og to sundhedsdatasæt stammer fra Japan. For syntetiske datasæt genererer vi et stort antal sekvenser og tidsserier fra hvert virkelig datasæt og tilpasser genereringsprocessen baseret på det specifikke problem, der bliver adresseret. De eksperimentelle resultater viser, at de nøjagtige algoritmer (FTPM, RTPM og STPM) præsterer bedre end baseline-metoderne i forhold til køretid og hukommelsesforbrug og skalerer godt på store datasæt. Derudover, opnår den tilnærmede FTPM op til 2 størrelsesordner samt den tilnærmede RTPM og STPM er op til 1 størresorden hurtigere end baselinemetoderne, samtidig med at der opretholdes en høj nøjagtighed.

Acknowledgements

I would like to thank many people who were with me to overcome the challenges of this Ph.D. journey and achieve its completion.

First, I would like to express my gratitude to my supervisor Prof. Torben Bach Pedersen. Under his guidance, I have gained a wealth of knowledge regarding scientific writing skills and research methods. His insightful feedback and valuable suggestion have played a crucial role in refining my ideas and enhancing the quality of my work. I have gained worthwhile lessons from him, particularly in his meticulousness and pursuit of perfection. I am deeply appreciative of his guidance and instructions throughout my Ph.D. study.

Second, I am especially grateful to my co-supervisor Asst. Prof. Nguyen Ho. She has put so much effort in supervising me. She hold weekly meetings with me throughout my PhD years, actively engaged in thought-provoking discussions, suggesting appropriate approaches to solve problems that guided me in the right direction. I have learned so much from her, from how to look for a promising and interesting research topic, define a research problem and find the solutions for it, to how to write a good scientific paper. I also deeply appreciate the significant amount of time she dedicated to revise my drafts. Further, I am truly grateful for her encouragement during the challenging phases of my Ph.D. study, which helped me overcome the difficulties posed by the Corona lockdown and the inherent challenges in my research.

I am thankful to Prof. Panagiotis Papapetrou for providing me with the opportunity to pursue my study at Stockholm University, which is a great benefit to my research. I am grateful for the support during this study through the funding from the Innovation Fund Denmark and the Horizon 2020 program.

Moreover, I would like to express my gratitude to all my colleagues at the Data Engineering, Science and Systems group for fostering a warm and welcoming work environment. I would like to thank my friends Rudra, Bhuvan, Suela, Carlos, Jonas, Kasper, Razvan, Tung, Fabio, Roshni, and Søren for making my Ph.D. life happier in lonely gray days. I am grateful to Kasper Fromm Pedersen and Theis Erik Jendal for helping me translate the abstract into Danish. I also thank Helle Westmark, Helle Schroll, Susanne Taunsig Larsen, Aage Sørensen, and Ulla Øland for their support of AAU administrative matters.

Lastly, I would like to express my gratitude to my family for their steady support throughout my Ph.D. journey. Their love, encouragement, and understanding have been the pillars of strength that have carried me through the challenges and accomplishments of pursuing my dreams.

Contents

Thesis Details

Thesis Title:	Efficiently Mining Temporal Patterns in Time Series Using Information Theory
PhD Student:	Van Long Ho Aalborg University
PhD Supervisor:	Prof. Torben Bach Pedersen Aalborg University
PhD Co-Supervisor:	Assistant Prof. Nguyen Ho Aalborg University

The main body of the thesis consists of the following papers.

[A] V. L. Ho, N. Ho, and T. B. Pedersen, "Efficient Temporal Pattern Mining in Big Time Series Using Mutual Information", in Proceedings of the VLDB Endowment (PVLDB), Volume 15, Number 3, Pages 673-685, 2021.

[B] V. L. Ho, N. Ho, T. B. Pedersen, and P. Papapetrou, "Efficient Generalized Temporal Pattern Mining in Big Time Series Using Mutual Information". Submitted to IEEE Transactions on Knowledge and Data Engineering (TKDE).

[C] V. L. Ho, N. Ho, and T. B. Pedersen, "Mining Seasonal Temporal Patterns in Time Series", in Proceedings of the IEEE International Conference on Data Engineering (ICDE), Pages 2240-2252, 2023.

Paper B significantly extends Paper A by generalizing the temporal pattern mining problem to mine both frequent temporal patterns (significant improvements) and rare temporal patterns (a novel proposal). In addition to the above papers, I am co-authors of the following three papers as part of my Ph.D. studies, which are not included in the thesis.

[D] N. Ho, V. L. Ho, T. B. Pedersen, and M. Vu, "Efficient and Distributed Temporal Pattern Mining", in IEEE International Conference on Big Data (Big Data), Pages 335-343, 2021.

[E] N. Ho, T. B. Pedersen, V. L. Ho, and M. Vu, "Efficient Search for Multi-Scale Time Delay Correlations in Big Time Series", in 23rd International

Conference on Extending Database Technology (EDBT), Pages 37-48, 2020.

[F] N. Ho, V. L. Ho, T. B. Pedersen, M. Vu, and C. Biscio, "A Unified Approach for Multi-Scale Synchronous Correlation Search in Big Time Series". Under submission.

This thesis has been submitted for assessment in partial fulfillment of the Ph.D. degree. The thesis is based on the submitted or published scientific papers listed above. Parts of the content of the papers in the main body of the thesis are used directly or indirectly in the extended summary part of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Technical Faculty of IT and Design at Aalborg University. The permission for using the published and accepted articles in the thesis have been obtained from the corresponding publishers with the condition that they are cited and copyrights are placed prominently in the references.

Van Long Ho
Aalborg University, August 12, 2023

Part I

Thesis Summary

Chapter 1

Introduction

The thesis focuses on extracting temporal patterns from time series. Unlike traditional sequential patterns in which occurrences of events are sequential, temporal patterns add temporal information into patterns, making them more expressive and providing more information in relations between events. This thesis will discover the different varieties of temporal patterns and propose optimization techniques for efficient mining algorithms. This section first introduces the background and motivation of the thesis, then outlines the objectives of the thesis, and finally describes the thesis structure.

1.1 Background and Motivation

1.1.1 Temporal Pattern Mining

The rapid development of IoT technology has enabled the collection of extensive volumes of time series data on unprecedented extent and acceleration. For example, smart meters and smart plugs are equipped in modern residential households, allowing for meticulous monitoring of the power consumption of electrical appliances [17], [16], [24]. Thousands of sensors are deployed in weather stations to observe numerous weather-related variables [67]. Additionally, mobile devices are supplied with various sensors to record user behaviors and track locations. These IoT-based systems generate daily terabytes of time series data that contain valuable information and, when they are explored, can provide priceless insights into specific application domains. These insights can be utilized to support evidence-based decision-making and optimization.

One of the first approaches for discovering hidden insights from time series is to find and analyze patterns from them. The traditional sequential pattern mining methods [50], [39] can detect such patterns.

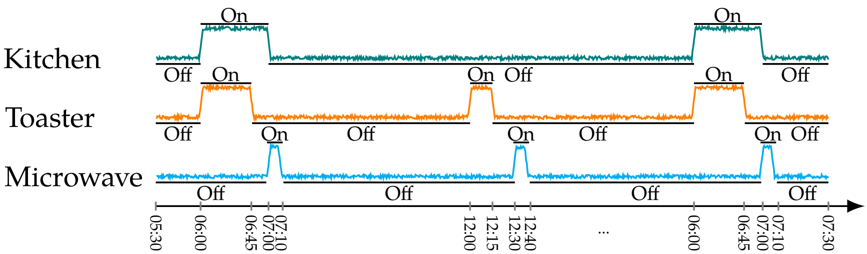


Fig. 1.1: Time series data on the energy consumption of electrical appliances

> **Example 1.1.1 (A sequential pattern)**
> Fig. 1.1 shows the energy usage of electrical devices. An interesting discovery is that the electrical devices are often used together in a specific time period of the day. A sequential pattern in Fig. 1.1 can be expressed as {Kitchen On} ⇒ {Toaster On, Microwave On}, showing that the presence of {Toaster On, Microwave On} is linked to the presence of {Kitchen On}.

However, sequential patterns only express the occurrence of events sequentially. In contrast, temporal patterns add further temporal information to events, which can express complex relations between events, such as contains and overlaps, and provide the details of when the events happen and for how long [51], [57], [58], [64], [10].

> **Example 1.1.2 (A temporal pattern)**
> The previous pattern in Example 1.1.1 would be expressed: ([06:00,07:00] Kitchen On $\succcurlyeq$ [06:00,06:45] Toaster On) (meaning Kitchen On contains Toaster On), ([06:00,07:00] Kitchen On $\rightarrow$ [07:00,07:10] Microwave On) (meaning Kitchen On is followed by Microwave On), and ([06:00,06:45] Toaster On $\rightarrow$ [07:00,07:10] Microwave On). Such insights are essential, as they can be used in facilitating the creation of smart homes, enabling the automation of electrical appliances.

There are different types of temporal patterns that can occur in a given specific time series dataset. Temporal patterns that occur frequently throughout the entire dataset are called *frequent temporal patterns*. In contrast, temporal patterns that rarely occur are called *rare temporal patterns*. While these rare temporal patterns are very important in many application domains, they can be easily missed if we do not have sufficient solutions to mine them.

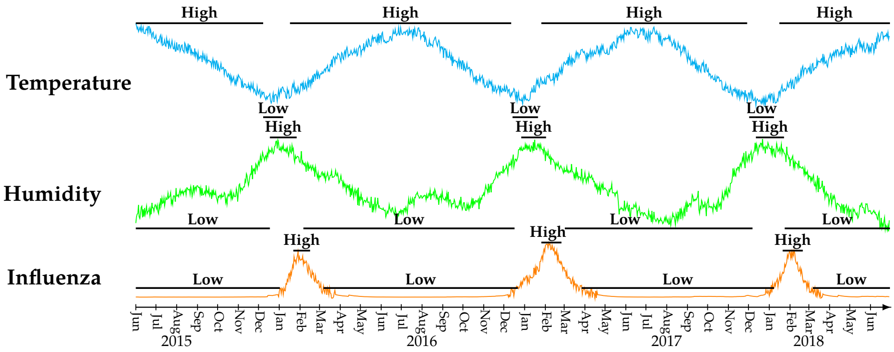


Fig. 1.2: Weather and Influenza time series [38]

> **Example 1.1.3 (A rare temporal pattern)**
> A rare temporal pattern would be found in smart city domain as: ([16:00,21:00] Snow $\succcurlyeq$ [17:30,19:30] StrongWind), ([16:00,21:00] Snow $\succcurlyeq$ [18:00,19:00] HighPedestrianInjury), and ([17:30,19:30] StrongWind $\succcurlyeq$ [18:00,19:10] HighPedestrianInjury). The pattern occurs infrequently but with high confidence, assisting in warning the citizens of severe weather conditions.

Another pattern that we also study in this thesis is called *seasonal temporal pattern*. The characteristic of this pattern is that it occurs concentrated in a particular period of time and repeats throughout the dataset periodically.

> **Example 1.1.4 (A seasonal temporal pattern)**
> Fig. 1.2 shows weather and influenza time series from Kawasaki, Japan between 2015-2018 [12], [67]. A seasonal temporal pattern would be explored as: Low Temperature *overlaps* High Humidity, Low Temperature *is followed by* High Influenza Cases, and High Humidity *is followed by* High Influenza Cases. This pattern occurs yearly in January and February. Based on the detection of such patterns, health experts could plan disease prevention and health protection.

Although temporal patterns are useful, there are existing gaps in discovering them in the current literature. Mining the three above types of temporal patterns is very expensive since each event has additional temporal information, and relations between temporal events are complex. Moreover, the current literature has several limitations when mining each type of pattern.

Specifically, for frequent temporal patterns, the existing algorithms cannot scale on big datasets, i.e., numerous time series or sequences, and only operate directly on pre-processed temporal events instead of time series data. For rare temporal patterns, using traditional pattern mining algorithms leads to an explosion of pattern candidates as the support measure has to be set to a low value. For seasonal temporal patterns, using the support measure is insufficient since the support does not reflect the seasonality characteristic. Moreover, the seasonal patterns do not uphold the anti-monotonicity property, i.e., a pattern is seasonal but its non-empty subsets may not be seasonal. Thus, we cannot apply the pruning techniques based on the anti-monotonicity property for seasonal temporal patterns. Motivated by these analyses, this thesis proposes efficient algorithms to mine these three types of temporal patterns: frequent, rare, and seasonal.

1.1.2 Information Theory

Mining temporal patterns are very expensive in real-world applications. The thesis uses information theory-based measures to reduce the mining cost, i.e., improve the speedup of the mining process but still obtain high accuracy. This section introduces fundamental concepts in information theory used in the proposed solutions of this thesis.

Entropy. The *entropy* $H(X)$ [14] of a discrete random variable X is defined as

$$H(X) = -\sum_{x \in X} p(x) \cdot \log p(x) \tag{1.1}$$

Intuitively, the entropy measures the uncertainty of a random variable X. As the value of $H(X)$ increases, the uncertainty of X also increases.

Conditional Entropy. The *conditional entropy* $H(X|Y)$ [14] of a discrete random variable X, given a discrete random variable Y, is defined as

$$H(X|Y) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \cdot \log \frac{p(x,y)}{p(y)} \tag{1.2}$$

Intuitively, the conditional entropy $H(X|Y)$ measures the uncertainty of X, given Y.

Mutual information. The *mutual information* $I(X;Y)$ [14] of two discrete random variables X and Y is defined as

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} p(x,y) \cdot \log \frac{p(x,y)}{p(x) \cdot p(y)} \tag{1.3}$$

Intuitively, the mutual information measures the reduction of uncertainty of one variable X, given another variable Y. The larger the value $I(X;Y)$, the more correlated information between X and Y.

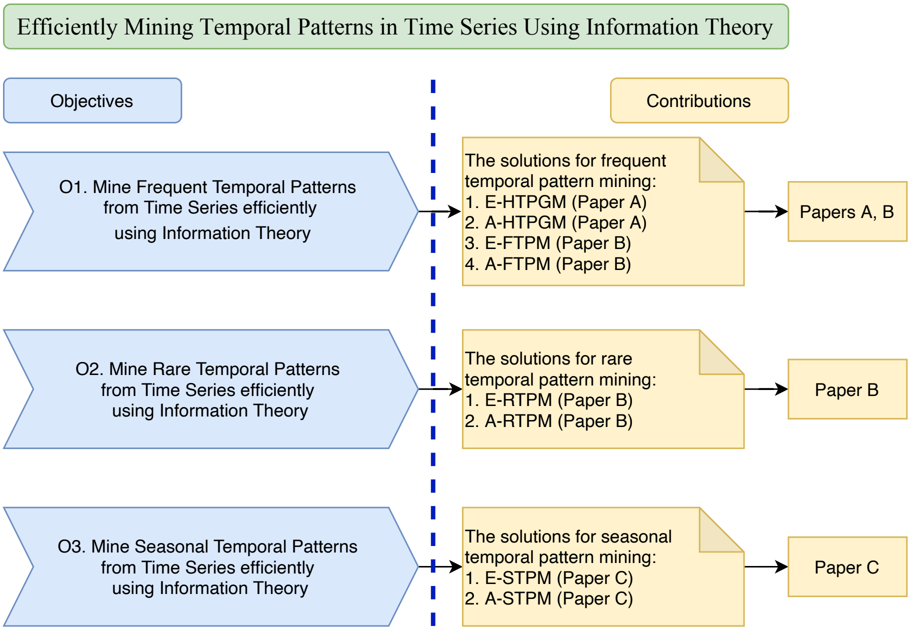


Fig. 1.3: The thesis objectives and the contributions

1.2 Objectives of the Thesis

The overall objective of the thesis is to propose efficient solutions to mine three types of temporal patterns: frequent, rare, and seasonal, directly from time series. In order to achieve this goal, three objectives need to be fulfilled:

- **O1**. We can mine frequent temporal patterns from time series efficiently and use information theory measures to further optimize the mining process.
- **O2**. We can efficiently discover rare temporal patterns from time series and enhance the mining process by incorporating information theory measures.
- **O3**. We can mine seasonal temporal patterns from time series efficiently and employ information theory measures to further optimize the mining process.

Fig. 1.3 shows the objectives of the thesis and the contributions of the three papers [36–38]. The left side of the figure lists three objectives of the thesis, and the right side is contributions to achieve the respective objectives.

For **O1**, the contributions are that we propose solutions for mining frequent temporal patterns efficiently. The solutions are proposed in two papers, A and

B [36, 37]. Two algorithms for frequent temporal pattern mining, the exact E-HTPGM and the approximate A-HTPGM, are presented in paper A [36]. E-HTPGM employs the hierarchical pattern graph structure and pruning techniques based on the Apriori principle and the transitivity property to facilitate faster mining. Moreover, based on mutual information, we derive a lower bound of the confidence of an event pair, thereby proposing the A-HTPGM that only mines temporal patterns on the promising time series, reducing the search space for the mining process. We continue working on the frequent temporal pattern mining problem on paper B [37] with several improvements. First, the exact E-FTPM algorithm improves the exact E-HTPGM by using the hierarchical hash tables instead of the hierarchical pattern graph [36] to enable faster retrieval of events and patterns. Then, the approximate A-FTPM is proposed by combining the lower bound of confidence in paper A and the new lower bound of support, further improving the speedup of the mining process.

For **O2**, the contributions are that we propose solutions for mining rare temporal patterns efficiently in paper B [37]. The solutions contain two algorithms: the exact E-RTPM and the approximate A-RTPM. The E-RTPM uses the variant of the hierarchical hash tables data structure [38] that enables fast retrieval of events and patterns, and applies the pruning techniques based on the Apriori principle and the transitivity property to optimize the search space. Moreover, the A-RTPM is proposed that uses the mutual information to prune the uncorrelated time series, thereby scaling well on large datasets.

For **O3**, the contributions are that we propose the first solution for mining seasonal temporal patterns efficiently in paper C [38]. In this solution, we first introduce several measures, including the maximum period, minimum density, distance interval, and minimum seasonal occurrence, to capture the seasonality characteristic of the seasonal temporal patterns in time series. Then, we present a new measure, called *maximum season*, and use it to define a new concept, called *candidate seasonal pattern*, used as a gatekeeper to identify frequent seasonal patterns. The exact E-STPM algorithm is proposed that uses the hierarchical hash tables structures to store and access the candidate events and patterns quickly and the pruning techniques based on the candidate seasonal pattern to reduce the search space. Finally, we propose the approximate A-STPM algorithm that uses the mutual information to eliminate the unpromising time series, helping A-STPM performs efficiently on large datasets.

1.3 Thesis Structure

The thesis focuses on achieving all the objectives mentioned in Section 1.2. The thesis is structured as follows.

Chapter 2 presents a solution for efficiently mining frequent temporal patterns in time series. The solution provides: (i) a comprehensive process that receives time series as input and produces all frequent temporal patterns as output, (ii) an efficient frequent temporal pattern mining (FTPM) algorithm that leverages the efficient data structure and the pruning techniques to optimize the mining process, (iii) an approximate version of FTPM using mutual information to help FTPM scale well in large datasets.

Frequent temporal patterns are significant; however, rare temporal patterns are still very interesting and useful in many applications because of high confidence. Based on the concepts of temporal patterns in Chapter 2, Chapter 3 proposes a solution for mining rare temporal patterns in time series. The solution includes: (i) a concept of rare temporal pattern with low support and high confidence, (ii) an efficient rare temporal pattern mining (RTPM) algorithm that utilizes the efficient data structures and the pruning techniques to optimize the mining process, (iii) an approximate version of RTPM that is based on mutual information to prune the unpromising time series to reduce the search space, thus, speed up the mining process.

Besides frequent and rare temporal patterns, seasonal temporal patterns are useful with the characteristic of periodic occurrences. Based on the concepts of temporal patterns in Chapters 2 and 3, Chapter 4 presents a solution for mining seasonal temporal patterns in time series. The key contributions are: (i) the first solution for seasonal temporal pattern mining (STPM) from time series, (ii) an efficient STPM algorithm using the concept of candidate seasonal pattern for pruning and the efficient data structures for rapid retrieval of events and patterns candidates; (iii) an approximate STPM that eliminates the redundant time series to achieve faster mining.

Chapter 5 summarizes our contributions and considers future works.

Chapter 2

Frequent Temporal Pattern Mining

This chapter summarizes Paper A [36] and a part of Paper B [37] which provide a significant improvement in frequent temporal pattern mining. Content from these papers is reused in the most effective way.

2.1 Problem Motivation and Statement

Section 1.1 shows that temporal patterns are useful in many real-world applications. However, mining frequent temporal patterns is very expensive. The temporal information and the complex relations between temporal events create an exponential search space with quadratic exponent in the pattern length h, i.e., the overall complexity $O(s^h r^{h^2})$ (s is the number of distinct events and r is the number of temporal relations). The explanation for search space complexity is as follows. The number of single events in the considered database is $M_1 \sim O(s)$. The number of event pairs is $M_2 \sim O(s^2)$. In M_2, each event pair can establish r temporal relations. Thus, the number of 2-event patterns is $M_2 \times r^1 \sim O(s^2 r^1)$. Similarly, the number of k-event patterns is $O(s^h \times r^{\frac{1}{2}h(h-1)}) \sim O(s^h r^{h^2})$. Hence, the total number of temporal patterns is $O(s) + O(s^2 r^1) + ... + O(s^h r^{h^2}) \sim O(s^h r^{h^2})$.

Several recent approaches have been proposed to mine frequent temporal patterns. TPrefix [68] is proposed by Wu et al. to mine temporal patterns. They define unambiguous temporal relations from which temporal patterns are mined. However, TPrefix repeats database scanning many times to mine patterns and does not apply pruning techniques to optimize the search space. Papapetrou et al. propose H-DFS [57] that employs different strategies, such as the breadth-first and depth-first search, to extract frequent temporal patterns.

H-DFS mines patterns in an enumeration tree of temporal arrangement and uses an IDList to store event intervals in which events/patterns occur. Thus, a dataset with many sequences or time series can deteriorate HDFS performance. Patel et al. propose IEMiner [58] in which relations between events are represented in an augmented hierarchical model and pruning techniques based on the Apriori principle to explore temporal patterns. Chen et al. propose TPMiner [10] that presents the end-point and end-time representations to handle the complex relations among events and offers several pruning techniques to decrease the search space. Recently, Lee et al. propose ZMiner [51] for a more efficient temporal pattern mining. ZMiner employs Z-Table data structure for the event's occurrence count and efficient candidate generation and Z-Arrangement data structure for fast arrangements extension. However, Z-Miner does not use any pruning techniques based on the transitivity property of temporal relations. Thus, IEMiner, TPMiner, and ZMiner cannot scale to big datasets. Moreover, the existing solutions for frequent temporal pattern mining only operate on the pre-processed temporal sequences instead of directly on time series.

In order to address the above limitations from existing literature, we focus on two main objectives for mining temporal patterns: efficiency and scalability. For the first objective, we propose an efficient frequent temporal pattern mining (FTPM) algorithm. For the second objective, we propose an approximate version of FTPM using mutual information that only mines frequent temporal patterns on the promising time series, thereby speeding up the mining process due to reducing the search space and helping the algorithm scale on big datasets. In summary, our main contributions are as follows:

- We introduce a general process for frequent temporal pattern mining from time series in which input is a set of time series, and output is all frequent temporal patterns.
- We propose an efficient frequent temporal pattern mining (FTPM) algorithm that employs efficient data structures and pruning techniques for optimization.
- We propose an approximate version of FTPM using mutual information to eliminate the unpromising time series that scale well on big datasets.
- We conduct extensive experiments on real-world and synthetic datasets to evaluate the performance of the proposed algorithms.

2.2 Preliminaries

In this section, we formally define temporal patterns and present some measures for mining frequent temporal patterns. All definitions are reproduced

Table 2.1: A Symbolic Database $\mathcal{D}_{\text{SYB}}$

Time	10:00	10:05	10:10	10:15	10:20	10:25	10:30	10:35	10:40	10:45	10:50	10:55	11:00	11:05	11:10	11:15	11:20	11:25
S	On	On	On	On	Off	Off	Off	On	On	Off	Off	Off	Off	Off	Off	On	On	On
T	Off	Off	Off	Off	Off	Off	Off	On	On	Off	Off	On	On	Off	Off	On	On	On
W	On	On	On	On	On	On	On	On	On	Off	Off	On	On	On	On	On	Off	Off
I	Off	Off	Off	Off	Off	Off	On	On	On	Off	Off	Off	On	On	Off	Off	On	On

Time	11:30	11:35	11:40	11:45	11:50	11:55	12:00	12:05	12:10	12:15	12:20	12:25	12:30	12:35	12:40	12:45	12:50	12:55
S	Off	Off	On	On	On	On	Off	Off	Off	On	On	On	Off	On	On	On	Off	Off
T	Off	Off	On	On	On	On	Off	Off	Off	On	On	Off	Off	Off	On	On	On	Off
W	On	On	On	On	On	On	On	On	On	Off	Off	On	On	On	On	On	Off	Off
I	Off	Off	Off	Off	Off	Off	Off	Off	Off	On	On	Off	Off	Off	Off	Off	On	On

from Paper A [36] and part of Paper B [37].

Definition 2.2.1 (Time series)
A *time series* $X = x_1, x_2, ..., x_n$ is a sequence of data values that measure the same phenomenon during an observation time period, and are chronologically ordered.

Definition 2.2.2 (Symbolic time series)
A *symbolic time series* X_S of a time series X encodes the raw values of X into a sequence of symbols. The finite set of permitted symbols used to encode X is called the *symbol alphabet* Σ_X of X.

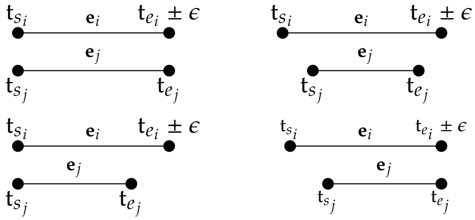
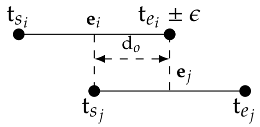
In order to obtain X_S, we use a mapping function f: $X \rightarrow \Sigma_X$ that maps $x_i \in X$ to a symbol $\omega \in \Sigma_X$.

> **Example 2.2.1 (A symbolic time series)**
> Let X = 1.8, 1.4, 1.1, 0.2, 0.0 be a time series of the energy consumption of an electrical appliance and Σ_X = {On, Off}, where On represents that the appliance is on and operating (e.g., $x_i \geq 0.5$), and Off represents that the appliance is off (e.g., $x_i < 0.5$), the symbolic time series of X is: X_S = On, On, On, Off, Off.

Definition 2.2.3 (Symbolic database)
Given a set of time series $\mathcal{X} = \{X_1, ..., X_n\}$, the set of symbolic representations of the time series in $\mathcal{X}$ forms a *symbolic database* $\mathcal{D}_{\text{SYB}}$.

> **Example 2.2.2 (A symbolic database)**
> A symbolic database $\mathcal{D}_{\text{SYB}}$ is shown in Table 2.1 . Four time series represent the energy consumption of four electrical appliances: {Stove (S), Toaster (T), Clothes Washer (W), Iron (I)}. All four appliances use the same symbol alphabets: Σ = {On, Off}.

Table 2.2: Temporal Relations between Events [36]

Follows: $E_{i \triangleright e_i} \rightarrow E_{j \triangleright e_j}$	 $\mathrm{t}_{e_i} \pm \epsilon \leq \mathrm{t}_{s_j}$
Contains: $E_{i \triangleright e_i} \succcurlyeq E_{j \triangleright e_j}$	 $(\mathrm{t}_{s_i} \leq \mathrm{t}_{s_j}) \wedge (\mathrm{t}_{e_i} \pm \epsilon \geq \mathrm{t}_{e_j})$
Overlaps: $E_{i \triangleright e_i} \between E_{j \triangleright e_j}$	 $(\mathrm{t}_{s_i} < \mathrm{t}_{s_j}) \wedge (\mathrm{t}_{e_i} \pm \epsilon < \mathrm{t}_{e_j}) \wedge (\mathrm{t}_{e_i} - \mathrm{t}_{s_j} \geq \mathrm{d}_o \pm \epsilon)$

Definition 2.2.4 (Temporal event in a symbolic time series)
A *temporal event* E in a symbolic time series X_S is a tuple $E = (\omega, T)$ where $\omega \in \Sigma_X$ is a symbol, and $T = \{[t_{s_i}, t_{e_i}]\}$ is the set of time intervals during which X_S is associated with the symbol ω.

A temporal event can be obtained by combining the identical continuous symbols in X_S into a time interval.

A single occurrence of the event E, i.e., $e = (\omega, [t_{s_i}, t_{e_i}])$, during $[t_{s_i}, t_{e_i}]$ is called an *instance* of E, denoted as $E_{\triangleright e}$.

Example 2.2.3 (A temporal event)
Let's examine the symbolic series of S as depicted in Table 2.1. The temporal event *"Stove is On"* is represented as follows: (SOn, {[10:00, 10:15], [10:35, 10:40], [11:15, 11:25], [11:40, 11:55], [12:15, 12:25], [12:35, 12:45]}). Here, (SOn, [10:00, 10:15]) is an event instance.

Relations between Temporal Events: We define three basic temporal relations between events based on Allen's relations model [2]. Moreover, we also add a tolerance *buffer* ϵ to the endpoints of the relations in order to limit the absolute time mapping problem but guarantee the temporal relations are mutually exclusive.

Table 2.2 lists the relations and their conditions, where ϵ is the buffer size ($\epsilon \geq 0$) and d_o is the minimum duration of overlap between two instances of

Table 2.3: A Temporal Sequence Database $\mathcal{D}_{\text{SEQ}}$

ID	Temporal sequences
1	(SOn, [10:00,10:15]), (TOff, [10:00,10:35]), (WOn, [10:00,10:40]), (IOff, [10:00,10:30]), (SOff, [10:15,10:35]), (IOn, [10:30,10:40]), (SOn, [10:35,10:40]), (TOn, [10:35,10:40])
2	(SOff, [10:45,11:15]), (TOff, [10:45,10:55]), (WOff, [10:45,10:55]), (IOff, [10:45,11:00]), (TOn, [10:55,11:00]), (WOn, [10:55,11:15]), (TOff, [11:00,11:15]), (IOn, [11:00,11:05]), (IOff, [11:05,11:20]), (SOn, [11:15,11:25]), (TOn, [11:15,11:25]), (WOff, [11:15,11:25]), (IOn, [11:20,11:25])
3	(SOff, [11:30,11:40]), (TOff, [11:30,11:40]), (WOn, [11:30,12:10]), (IOff, [11:30,12:10]), (SOn, [11:40,11:55]), (TOn, [11:40,11:55]), (SOff, [11:55,12:10]), (TOff, [11:55,12:10])
4	(SOn, [12:15,12:25]), (TOn, [12:15,12:20]), (WOff, [12:15,12:25]), (IOn, [12:15,12:20]), (TOff, [12:20,12:40]), (IOff, [12:20,12:50]), (SOff, [12:25,12:35]), (WOn, [12:25,12:45]), (SOn, [12:35,12:45]), (TOn, [12:40,12:50]), (SOff, [12:45,12:55]), (WOff, [12:45,12:55]), (TOff, [12:50,12:55]), (IOn, [12:50,12:55])

an event ($0 \leq \epsilon \ll d_o$).

Definition 2.2.5 (Temporal pattern)
Let $\mathfrak{R}$={Follows, Contains, Overlaps} be the set of temporal relations. A *temporal pattern* P=<(r_{12}, E_1, E_2),...,($r_{(n-1)(n)}, E_{n-1}, E_n$)> is a list of triples (r_{ij},E_i,E_j), each representing a relation $r_{ij} \in \mathfrak{R}$ between two events E_i and E_j.

We note that the relation r_{ij} is formed between the event instances of E_i and E_j. If the number of events in the temporal pattern P is n, P is called an n-event pattern. We denote $E_i \in P$ if the event E_i occurs in P, and $P_1 \subseteq P$ if a pattern P_1 is a sub-pattern of P.

Definition 2.2.6 (Temporal sequence)
A list of n event instances S=<$e_1, ..., e_i, ..., e_n$> forms a *temporal sequence* if the instances are chronologically ordered by their start times. Moreover, S has size n, denoted as $|S| = n$.

A set of temporal sequences forms a *temporal sequence database* $\mathcal{D}_{\text{SEQ}}$ where each row i contains a temporal sequence S_i.

Example 2.2.4 (A temporal sequence database)
Table 2.3 is an example of the temporal sequence database $\mathcal{D}_{\text{SEQ}}$ that is converted from the symbolic database $\mathcal{D}_{\text{SYB}}$ in Table 2.1 .

The temporal sequence S *supports* a temporal pattern P, denoted as $P \in S$, iff $|S| \geq 2 \land \forall (r_{ij}, E_i, E_j) \in P$, $\exists (e_l, e_m) \in S$ such that r_{ij} holds between $E_{i_{\triangleright e_l}}$ and $E_{j_{\triangleright e_m}}$. Otherwise, if P is supported by S, P can be written as P=<(r_{12}, $E_{1_{\triangleright e_1}}$, $E_{2_{\triangleright e_2}}$), ..., ($r_{(n-1)(n)}$,$E_{n-1_{\triangleright e_{n-1}}}$, $E_{n_{\triangleright e_n}}$)>, where the relation between two events in each triple is expressed using the event instances.

> **Example 2.2.5 (A temporal sequence supports a temporal patern)**
> Consider the sequence S =<e_1=(SOn, [12:15, 12:25]), e_2=(TOn, [12:15, 12:20]), e_3=(IOn, [12:50, 12:55])> at the fourth row of $\mathcal{D}_{\text{SEQ}}$ in Table 2.3 . We can see that S supports a 3-event pattern P=<(Contains, $\text{SOn}_{\triangleright e_1}$, $\text{TOn}_{\triangleright e_2}$), (Follows, $\text{SOn}_{\triangleright e_1}$, $\text{IOn}_{\triangleright e_3}$), (Follows, $\text{TOn}_{\triangleright e_2}$, $\text{IOn}_{\triangleright e_3}$)>.

Definition 2.2.7 (Support of a temporal pattern)
The *support* of a pattern P is the number of sequences $S \in \mathcal{D}_{\text{SEQ}}$ that support P.

$$supp(P) = |\{S \in \mathcal{D}_{\text{SEQ}} \ s.t. \ P \in S\}| \tag{2.1}$$

The support of a group of events $(E_1, ..., E_n)$, denoted as $supp(E_1, ..., E_n)$, is defined similarly to that of a temporal pattern.

Intuitively, the support of an event group/ pattern shows the frequency of occurrence of an event group/ pattern in a database.

Definition 2.2.8 (Confidence of a temporal pattern)
The *confidence* of a temporal pattern P in $\mathcal{D}_{\text{SEQ}}$ is the fraction between $supp(P)$ and the support of its most frequent event:

$$conf(P) = \frac{supp(P)}{\max_{1 \leq k \leq |P|}\{supp(E_k)\}} \tag{2.2}$$

where $E_k \in P$ is a temporal event.

The confidence of a group of events $(E_1, ..., E_n)$, denoted as $conf(E_1, ..., E_n)$, is defined similarly to that of a temporal pattern.

Intuitively, the confidence reflects the minimum probability of an event group/ pattern, given the probability of its most frequent event.

Problem Formulation: Frequent Temporal Pattern Mining from Time Series (FTPMfTS) [36]
Given a set of univariate time series $\mathcal{X} = \{X_1, ..., X_n\}$, let $\mathcal{D}_{\text{SEQ}}$ be the temporal sequence database obtained from $\mathcal{X}$, and σ and δ be the support and confidence thresholds, respectively. The FTPMfTS problem aims to find all temporal patterns P that have high enough support and confidence in $\mathcal{D}_{\text{SEQ}}$: $supp(P) \geq \sigma$ $\land$ $conf(P) \geq \delta$.

2.3 Frequent Temporal Pattern Mining from Time Series (FTPMfTS) process

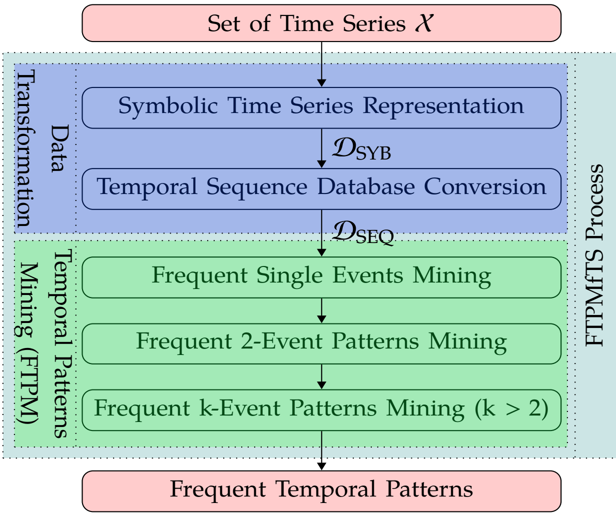


Fig. 2.1: The FTPMfTS process [36]

The FTPMfTS process consists of two phases, shown in Fig. 2.1. The first phase, *Data Transformation*, transforms the set of time series $\mathcal{X}$ into the symbolic time series database $\mathcal{D}_{\text{SYB}}$, then converts $\mathcal{D}_{\text{SYB}}$ to the sequence database $\mathcal{D}_{\text{SEQ}}$. The second phase, *Frequent Temporal Patterns Mining*, includes 3 step: (1) *Frequent Single Events Mining*, (2) *Frequent 2-Event Patterns Mining*, and (3) *Frequent k-Event Patterns Mining* (k>2). The final output is all frequent temporal patterns.

2.3.1 Data Transformation

Symbolic Time Series Representation

We use the mapping function in Def. 2.2.2 to convert each time series in $\mathcal{X}$ to the symbolic time series. This step will create the $\mathcal{D}_{\text{SYB}}$ database.

Temporal Sequence Database Conversion

To transform $\mathcal{D}_{\text{SYB}}$ into $\mathcal{D}_{\text{SEQ}}$, we divide the symbolic series in $\mathcal{D}_{\text{SYB}}$ into sequences of equal length, with each sequence corresponding to a row in $\mathcal{D}_{\text{SEQ}}$. Nevertheless, the process of splitting may cause a loss of temporal patterns since a *splitting point* can place a pattern into different sequences.

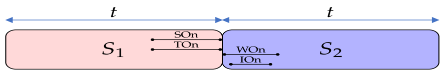


Fig. 2.2: Non-overlapped splitting strategy [36]

Example 2.3.1 (Non-overlapped splitting)
We split each symbolic series in Table 2.1 into 4 sequences and each sequence will span 40 minutes. Temporal events S, T, W, and I occurring between 10:00 and 10:40 will be in the first sequence S_1. And temporal events S, T, W, and I from 10:45 to 11:25 will be in the second sequence S_2, similarly for S_3 and S_4. Fig. 2.2 shows the loss of the straightforward non-overlapped splitting strategy. Four events, SOn, TOn, WOn, and IOn, are divided into 2 sequences. Specifically, SOn and TOn are in S_1, and WOn and IOn are in S_2. This separation leads to the loss of the 4-event pattern P=<(Contains, SOn, TOn), (Follows, SOn, WOn), (Follows, SOn, IOn), (Follows, TOn, WOn), (Follows, TOn, IOn), (Contains, WOn, IOn)>.

In order to address the loss issue, we use an overlapping sequences strategy. Let t_{ov} be a overlapped duration, where $0 \leq t_{\text{ov}} \leq t_{\text{max}}$ and t_{max} represents the maximum duration of a pattern. Two consecutive sequences are overlapped within t_{ov}.

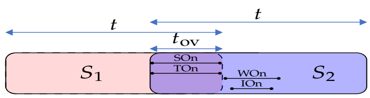


Fig. 2.3: Overlapped splitting strategy [36]

Example 2.3.2 (Overlapped splitting)
Fig. 2.3 shows a splitting strategy using overlapping sequences. The overlapping between S_1 and S_2 ensures that the four events, SOn, TOn, WOn, and IOn, remain together in S_2, thereby preserving the pattern.

2.3.2 Frequent Temporal Pattern Mining

After the data transformation phase, we proceed to the frequent temporal pattern mining phase. First, we find frequent single events, which is the fundamental step for the subsequent mining process. Next, we mine frequent 2-event patterns that use the found frequent single events. Finally, we mine frequent k-event patterns that utilize both the frequent single events and the frequent 2-event patterns to generate the k-event patterns. Details of the mining process for each step are described in Section 2.4.

2.4 Frequent Temporal Pattern Mining (Exact FTPM)

In this section, we present the frequent temporal pattern mining (FTPM) algorithm to mine frequent temporal patterns from $\mathcal{D}_{\text{SEQ}}$. The lemmas are reproduced from Paper A [36] and part of Paper B [37], and detailed proofs of the lemmas can be found in Paper A and Paper B.

2.4.1 Hierarchical lookup hash structure for FTPM

Paper A presents an algorithm for mining frequent temporal patterns, called *HTPGM*, that uses *Hierarchical Pattern Graph* data structure to maintain frequent events and patterns. Paper B improves the HTPGM algorithm by using *Hierarchical Hash Tables* instead of the Hierarchical Pattern Graph, enabling faster retrieval of events and patterns. Now, we discuss the Hierarchical Hash Tables data structure used in FTPM.

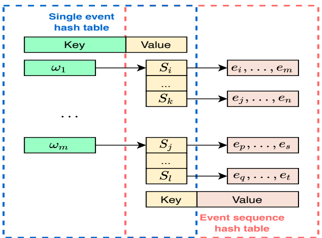


Fig. 2.4: The HLH_1 structure [37]

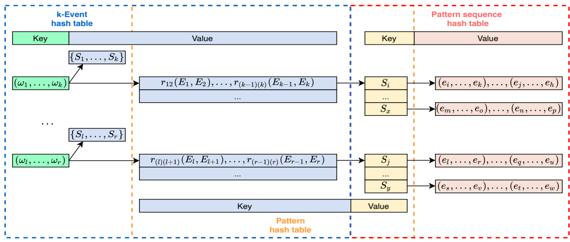


Fig. 2.5: The $HLH_k(k \geq 2)$ structure [37]

Hierarchical lookup hash structure HLH_1: The HLH_1 structure is used to store single events, illustrated in Fig. 2.4 . HLH_1 includes two hash tables. The first hash table is the *single event hash table*, denoted as EH, and the second hash table is the *event sequence hash table*, denoted as SH. Each hash table comprises a collection of <key, value> pairs. In the EH hash table, the key corresponds to the symbol $\omega \in \Sigma_X$ associated with the event E_i, and the corresponding value contains the list of sequences $< S_i, ..., S_k >$ that support E_i. In the SH hash table, the key consists of the sequence list from EH, and the value corresponds to instances of the event E_i.

Hierarchical lookup hash structure HLH_k: k-event groups and k-event patterns are stored in the HLH_k ($k \geq 2$), illustrated in Fig. 2.5 . HLH_k consists of 3 hash tables. The first hash table is the *k-event hash table*, denoted as EH_k, the second one is the *pattern hash table*, denoted as PH_k, and the third one is the *pattern sequence hash table*, denoted as SH_k. In the EH_k hash table, the *key* corresponds to the symbols list $(\omega_1..., \omega_k)$ that represents the group of k-events $g = (E_1, ..., E_k)$, and the *value* includes 2 components: the sequence list $< S_i, ..., S_k >$ where g occurs, and the list containing the k-event patterns P of g. In the PH_k hash table, the *key* corresponds to the k-event pattern P from EH_k, and the *value* contains the sequence list of P. In the SH_k hash table, the *key* is the sequence list from PH_k, and the *value* contains the list of event instances forming P.

The hierarchical lookup hash structures support quick retrieving of events and patterns during the mining process. Next, we describe the FTPM algorithm as in Algorithm 1 .

Algorithm 1: Frequent Temporal Pattern Mining [37]

Input: Temporal sequence database $\mathcal{D}_{\text{SEQ}}$, minimum support threshold σ, confidence threshold δ
Output: The set of temporal patterns P satisfying σ, δ
//Mining frequent single events
1: **foreach** *event* $E_i \in \mathcal{D}_{SEQ}$ **do**
2: Compute *supp*(E_i);
3: **if** *supp*$(E_i) \geq \sigma$ **then**
4: Insert E_i to *1Freq*;
//Mining frequent 2-event patterns
5: EventPairs $\longleftarrow$ Cartesian(*1Freq*,*1Freq*);
6: FrequentPairs $\longleftarrow \emptyset$;
7: **foreach** (E_i, E_j) *in EventPairs* **do**
8: Compute *supp*(E_i, E_j);
9: **if** *supp*$(E_i, E_j) \geq \sigma$ **then**
10: FrequentPairs $\longleftarrow$ Apply_Lemma4(E_i, E_j);
11: **foreach** (E_i, E_j) *in FrequentPairs* **do**
12: Retrieve event instances;
13: Check temporal relations against σ, δ;
//Mining frequent k-event patterns
14: Candidate1Freq $\longleftarrow$ Transitivity_Filtering(1Freq);
15: kEvents $\longleftarrow$ Cartesian(*Candidate1Freq*,*(k-1)Freq*);
16: FrequentkEvents $\longleftarrow$ Apriori_Filtering(kEvents);
17: **foreach** *kEvents in FrequentkEvents* **do**
18: Retrieve relations;
19: Iteratively check relations against σ, δ;

2.4.2 Mining Frequent Single Events

The initial step of FTPM aims to look for frequent single events (Alg. 1 , lines 1-4). We calculate the support for each event E_i and determine whether the support of E_i satisfies σ. At this step, we do not consider the confidence of single events since it is always 1.

We provide a running example in Fig. 2.6 using data in Table 2.3, with $\sigma = 0.7$ and $\delta = 0.7$. We have 7 frequent single events, including SOn, SOff, WOn, TOn, TOff, IOff, and IOn.

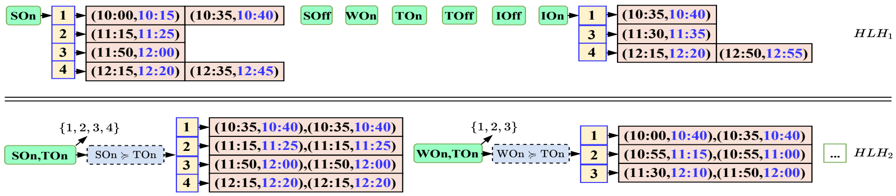


Fig. 2.6: An example for the hierarchical lookup hash tables

2.4.3 Mining Frequent 2-event Patterns

In order to reduce the cost of checking candidates of frequent patterns, we propose to divide the mining process into two steps: (1) it first finds frequent k-event groups, (2) it then determines frequent temporal patterns only from those frequent k-event groups. Two following lemmas ensure the correctness of this approach.

Lemma 1 *Let P be a 2-event pattern formed by an event pair (E_i, E_j). Then, $supp(P) \leq supp(E_i, E_j)$.*

From Lemma 1 , we can prune infrequent event pairs safely since we cannot create frequent patterns from infrequent event pairs.

Lemma 2 *Let (E_i, E_j) be a pair of events forming a 2-event pattern P. Then conf(P) $\leq$ conf(E_i, E_j).*

From Lemma 2 , we can prune low-confidence event pairs safely since they cannot create high-confidence patterns. We apply Lemmas 1 and 2 to the mining process to reduce the candidate patterns generation.

Mining frequent event pairs: Alg. 1 (lines 5-10) describes this step. First, we generate all event pairs. Next, for each pair (E_i, E_j), the set of sequences $\mathcal{S}_{ij}$ where both events occur is retrieved, and we compute the support $supp(E_i, E_j)$ using $\mathcal{S}_{ij}$. If (E_i, E_j) has high enough support and high confidence, they are stored in EH_2 of HLH_2.

Mining frequent 2-event patterns: Alg. 1 (lines 11-13) describes the mining for frequent 2-event patterns. First, for each frequent event pair (E_i, E_j), we look for the temporal relations between E_i and E_j using the set of sequences $\mathcal{S}_{ij}$. Only the relations that satisfy the two constraints σ and δ are stored in HLH_2. Several relations in HLH_2 are shown in Fig. 2.6 , e.g., event pair (SOn, TOn).

2.4.4 Mining Frequent k-event Patterns

The mining steps for frequent k-event patterns are similar to frequent 2-event patterns, consisting of finding frequent k-event combinations and then mining frequent k-event patterns. Moreover, we employ the transitivity property of temporal relations to further optimize the frequent k-event patterns mining.

Mining frequent k-event combinations: Alg. 1 (lines 14-16) describes the mining step of frequent k-event combinations. First, we calculate the Cartesian product between the frequent (k-1)-event combinations *(k-1)Freq* at HLH_{k-1} and the frequent single events *1Freq*: *(k-1)Freq* $\times$ *1Freq*, to create k-event combinations. Then, we only select the k-event combinations that satisfy the support σ and the confidence δ.

However, we observe that not all frequent single events at HLH_1 can create frequent patterns at HLH_k. For example, we consider the event IOn at HLH_1 in Fig. 2.6 . Here, we can use IOn to combine with (SOn, TOn) at HLH_2 to create a 3-event combination (SOn, TOn, IOn). However, (SOn, TOn, IOn) cannot create any frequent 3-event patterns, since IOn is not present at HLH_2. Thus, the combination (SOn, TOn, IOn) should not be created. To reduce such redundancy, we rely on the *transitivity property* as follows.

Lemma 3 *Let* $S =< e_1,..., e_{n-1} >$ *be a temporal sequence that supports an (n-1)-event pattern* $P =< (r_{12}, E_{1_{\triangleright e_1}}, E_{2_{\triangleright e_2}}),..., (r_{(n-2)(n-1)}, E_{n-2_{\triangleright e_{n-2}}}, E_{n-1_{\triangleright e_{n-1}}}) >$*. Let* e_n *be a new event instance added to* S *to create the temporal sequence* $S' =< e_1, ..., e_n >$.

The set of temporal relations $\mathfrak{R}$ *is transitive on* S': $\forall e_i \in S'$, $i < n$, $\exists r \in \mathfrak{R}$ *s.t.* $r(E_{i_{\triangleright e_i}}, E_{n_{\triangleright e_n}})$ *holds.*

From Lemma 3 , a new event instance that is added to a temporal sequence S will always form at least one temporal relation.

Lemma 4 *Let* $N_{k-1} = (E_1, ..., E_{k-1})$ *be a (k-1)-event combination and* E_k *be a single event, both satisfying the* σ *constraint. The combination* $N_k = N_{k-1} \cup E_k$ *can form k-event temporal patterns whose support is greater than* σ *if* $\forall E_i \in N_{k-1}$, $\exists r \in \mathfrak{R}$ *s.t.* $r(E_i, E_k)$ *is a frequent temporal relation.*

From Lemma 4 , only events in HLH_1 that exist in HLH_{k-1} should be employed to generate k-event combinations. Using Lemma 4 , we extract distinct single events X_{k-1} from HLH_{k-1}, and *intersect* X_{k-1} with the frequent single events *1Freq* in HLH_1 to eliminate redundant events: *Candidate1Freq* = $X_{k-1} \cap$ *1Freq*. Next, we calculate the Cartesian product *(k-1)Freq* $\times$ *Candidate1Freq* to create k-event combinations. Finally, we only select frequent k-event combinations *kFreq*.

Mining frequent k-event patterns: Alg. 1 (lines 17-19) describes this step. The cost of checking temporal relations in a k-event combination ($k \geq 3$) satisfying support and confidence constraints is very expensive. Thus, we propose a more efficient method to check these temporal relations based on the *transitivity property* and the Apriori principle.

Lemma 5 *Let P and P' be two temporal patterns. If $P' \subseteq P$, then conf(P') $\geq$ conf(P).*

Lemma 6 *Let P and P' be two temporal patterns. If $P' \subseteq P$ and $\frac{supp(P')}{\max_{1 \leq k \leq |P|}\{supp(E_k)\}}_{E_k \in P} \leq \delta$, then conf(P) $\leq \delta$.*

According to Lemma 5, the confidence of a pattern P is less than or equal to the confidence of its sub-patterns. Lemma 6 says that if any of the sub-patterns of a temporal pattern P have low confidence, then P cannot have high confidence. We use Lemmas 5 and 6 as follows.

Let $M_{k-1} = (E_1, ..., E_{k-1})$ be a frequent (k-1)-event combination, $M_1 = (E_k)$ be an single event, and $M_k = M_{k-1} \cup M_1 = (E_1, ..., E_k)$ be a k-event combination. To determine k-event patterns for M_k, we first retrieve the set P_{k-1} containing frequent (k-1)-event patterns of M_{k-1}. Each $p_{k-1} \in P_{k-1}$ is a list of $\frac{1}{2}(k-1)(k-2)$ triples: $\{(r_{12}, E_{1_{\triangleright e_1}}, E_{2_{\triangleright e_2}}),...,(r_{(k-2)(k-1)}, E_{k-2_{\triangleright e_{k-2}}}, E_{k-1_{\triangleright e_{k-1}}})\}$. We iteratively check the possibility of p_{k-1} and E_k can create a frequent k-event pattern as follows. We first check whether the triple $(r_{(k-1)k}, E_{k-1_{\triangleright e_{k-1}}}, E_{k_{\triangleright e_k}})$ has high enough support and high confidence by accessing the HLH_2 table. If the triple does not have high enough support (using Lemmas 3 and 4), or high confidence (using Lemmas 3 , 5 , and 6), the checking process stops immediately for p_{k-1}. Otherwise, it continues on the triple $(r_{(k-2)k}, E_{k-2_{\triangleright e_{k-2}}}, E_{k_{\triangleright e_k}})$, until it reaches $(r_{1k}, E_{1_{\triangleright e_1}}, E_{k_{\triangleright e_k}})$.

2.5 Approximate FTPM

This section introduces an approximate version of FTPM using mutual information to find dependent time series, and performing FTPM only on these time series. The definitions, theorems, and corollaries are reproduced from Paper A [36] and part of Paper B [37], and their proofs can be found in Paper A and Paper B.

Let X_S and Y_S be the symbolic series representing the time series X and Y, respectively, and Σ_X, Σ_Y be their alphabets.

2.5.1 Mutual Information of Symbolic Time Series

As mentioned in Section 1.1.2, mutual information measures how dependent two random variables are. For approximate FTPM, we calculate the mutual information (MI) of two symbolic time series, i.e., $I(X_S, Y_S)$, and calculate the entropy of each symbolic time series, i.e., $H(X_S)$.

However, MI has no upper bound since $0 \leq I(X_S; Y_S) \leq \min(H(X_S), H(Y_S))$ [14]. To scale the MI into the range $[0 - 1]$, we use normalized mutual information as defined below.

Definition 2.5.1 (Normalized mutual information)
The *normalized mutual information* (NMI) of two symbolic time series X_S and Y_S, denoted as $\widetilde{I}(X_S;Y_S)$, is defined as

$$\widetilde{I}(X_S;Y_S) = \frac{I(X_S;Y_S)}{H(X_S)} = 1 - \frac{H(X_S|Y_S)}{H(X_S)} \tag{2.3}$$

Based on Eq. (2.3), a pair of (X_S, Y_S) holds a mutual dependency if $\widetilde{I}(X_S;Y_S) > 0$. Moreover, NMI is not symmetric, i.e., $\widetilde{I}(X_S;Y_S) \neq \widetilde{I}(Y_S;X_S)$.

2.5.2 Relationship between the Support of an Event Pair in $\mathcal{D}_{\mathbf{SYB}}$ and $\mathcal{D}_{\mathbf{SEQ}}$

In Sections 2.5.3 and 2.5.4 , we study the relationship between mutual information of two symbolic time series, and the support and the confidence of an event pair. Since calculating mutual information of two symbolic time series uses the database $\mathcal{D}_{\text{SYB}}$, and calculating the support and the confidence of an event pair uses the database $\mathcal{D}_{\text{SEQ}}$, thus we first derive a connection between the support of an event pair in $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$, and use this connection to prove the relationships of mutual information and the support and confidence in Sections 2.5.3 and 2.5.4 .

Lemma 1 *Let* $supp(X_1, Y_1)_{\mathcal{D}_{SYB}}$ *and* $supp(X_1, Y_1)_{\mathcal{D}_{SEQ}}$ *be the support of* (X_1, Y_1) *in* $\mathcal{D}_{SYB}$ *and* $\mathcal{D}_{SEQ}$*, respectively. We have the following relation:* $supp(X_1, Y_1)_{\mathcal{D}_{SYB}} \leq supp(X_1, Y_1)_{\mathcal{D}_{SEQ}}$.

Lemma 1 shows that if an event pair is frequent in $\mathcal{D}_{\text{SYB}}$ then it is also frequent in $\mathcal{D}_{\text{SEQ}}$.

2.5.3 Lower Bound of the Support

In the approximate FTPM, we use mutual information to select the dependent time series and perform the mining only on these time series. Since the FTPMfTS problem uses the support to evaluate the occurrence frequency of events/patterns, in this section, we investigate the relationship between the mutual information of two symbolic series and the support of an event pair as in Theorem 1 , and use this relationship in the approximate FTPM to prune the unpromising time series, help reduce the search space of the mining.

Theorem 1 *(Lower bound of the support)*
Let μ *be the minimum mutual information threshold. If* $\widetilde{I}(X_S;Y_S) \geq \mu$*, then the lower bound of the support of* (X_1, Y_1) *in* $\mathcal{D}_{SEQ}$ *is:*

$$supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \lambda_2 \cdot e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot ln2}{\lambda_2}\right)} \tag{2.4}$$

where λ_1 is the minimum support of $X_i \in X_S$, λ_2 is the support of $Y_1 \in Y_S$, and W is the Lambert function [13].

Using Theorem 1 , μ is derived such that $supp(X_1, Y_1)$ is at least σ.

Corollary 1.1 *The support of an event pair $(X_1, Y_1) \in (X_S, Y_S)$ in $\mathcal{D}_{SEQ}$ is at least σ if $\widetilde{I}(X_S; Y_S)$ is at least μ, where:*

$$\mu \geq \begin{cases} 1 - \frac{\lambda_2}{e \cdot \ln 2 \cdot \log \frac{1}{\lambda_1}}, & \text{if} \quad 0 \leq \frac{\sigma}{\lambda_2} \leq \frac{1}{e} \\ 1 - \frac{\sigma \cdot \log \frac{\sigma}{\lambda_2}}{\ln 2 \cdot \log \lambda_1}, & \text{otherwise} \end{cases} \tag{2.5}$$

Interpretation: Theorem 1 states that if two series, namely X_S and Y_S, exhibit mutual dependence on the value μ, then the support of an event pair in (X_S, Y_S) is not less than the specified lower bound as presented in Eq. (2.4). By applying both Theorem 1 and Lemma 1, if the support of an event pair of (X_S, Y_S) is less than the specified bound in Eq. (2.4), any pattern formed by that event pair will also have a support value lower than the established bound.

2.5.4 Lower bound of the Confidence

The FTPMfTS problem uses the confidence to evaluate the likelihood of an events group/ pattern. Besides that, mutual information is used to select the dependent time series in the approximate FTPM. Thus, we investigate the relationship between the mutual information of two symbolic series and the confidence of an event pair as in Theorem 2 , and combine this relationship with the result of Theorem 1 to prune the unpromising time series in the approximate FTPM, reducing the search space of the mining.

Theorem 2 *(Lower bound of the confidence)*
Let σ and μ be the minimum support and minimum mutual information thresholds, respectively. Assume that $supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \sigma$. If the NMI $\widetilde{I}(X_S; Y_S) \geq \mu$, then the lower bound of the confidence of (X_1, Y_1) in $\mathcal{D}_{SEQ}$ is:

$$conf(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \sigma \cdot \lambda_1^{\frac{1-\mu}{\sigma}} \cdot \left(\frac{n_x - 1}{1 - \sigma} \right)^{\frac{\lambda_3}{\sigma}} \tag{2.6}$$

where n_x is the number of symbols in Σ_X, λ_1 is the minimum support of $X_i \in X_S$, and λ_3 is the support of $(X_i, Y_j) \in (X_S, Y_S)$ such that $p(X_i|Y_j)$ is minimal, $\forall (i \neq 1 \wedge j \neq 1)$.

From Theorem 2 , μ can be derived such that $conf(X_1, Y_1)$ is at least δ.

Corollary 2.1 *The confidence of an event pair* $(X_1, Y_1) \in (X_S, Y_S)$ *in* $\mathcal{D}_{SEQ}$ *is at least* δ *if* $\widetilde{I}(X_S; Y_S)$ *is at least* μ*, where:*

$$\mu \geq 1 - \sigma \cdot \log_{\lambda_1}\left(\frac{\delta}{\sigma} \cdot \left(\frac{1-\sigma}{n_x - 1}\right)^{\frac{\lambda_3}{\sigma}}\right) \tag{2.7}$$

Interpretation: From Theorem 2, if two symbolic time series, namely X_S and Y_S, are mutually dependent, then the confidence of an event pair in (X_S, Y_S) is not less than the confidence lower bound in Eq. (2.6). Applying Theorem 2 and Lemma 2 , if the confidence of an event pair (X_1, Y_1) of (X_S,Y_S) is less than the bound in Eq. (2.6), then any pattern created by (X_1, Y_1) also has a confidence value lower than the established bound.

2.5.5 Approximate FTPM

This section describes the approximate FTPM algorithm. We first present how to determine the value of μ for selecting the dependent time series. We then explain the approximate FTPM algorithm in detail.

Setting the value of μ: FTPM uses two pre-defined parameters, the minimum support σ and the minimum confidence δ, to extract frequent temporal patterns. To identify patterns that adhere to both the σ and δ constraints, we choose a value μ that ensures both Eqs. (2.5) and (2.7) hold.

Algorithm 2: Approximate FTPM using Mutual Information [37]

Input: A set of time series $\mathcal{X}$, a minimum support threshold σ, a minimum confidence threshold δ

Output: The set of frequent temporal patterns P

1: Convert $\mathcal{X}$ to $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$;
2: Scan $\mathcal{D}_{\text{SYB}}$ to compute the probability of each event and event pair;
3: **foreach** *pair of symbolic time series* $(X_S, Y_S) \in \mathcal{D}_{SYB}$ **do**
4:     Compute $\widetilde{I}(X_S; Y_S)$ and $\widetilde{I}(Y_S; X_S)$;
5:     Compute μ using Eqs. (2.5) and (2.7);
6:     **if** $\min\{\widetilde{I}(X_S; Y_S), \widetilde{I}(Y_S; X_S)\} \geq \mu$ **then**
7:         Insert X_S and Y_S into X_C;
8: **foreach** $X_S \in \mathsf{X}_C$ **do**
9:     Mine frequent single events from X_S ;
10: **foreach** $(X_S, Y_S) \in \mathsf{X}_C$ **do**
11:     Mine frequent 2-event patterns from (X_S, Y_S) ;
12: **if** $k \geq 3$ **then**
13:     Mine frequent k-event patterns similar to the exact FTPM ;

Approximate FTPM: The approximate FTPM is described as in Alg. 2. The approximate FTPM only mines on *the set of mutually dependent symbolic series* $\mathsf{X}_C \in \mathcal{X}$ *with the minimum threshold* μ. We first scan $\mathcal{D}_{\text{SYB}}$ to calculate the probability associated with each event and each pair of events (line 2).

Subsequently, for each pair of symbolic series, NMI and μ values are calculated (lines 3-5). The pairs of symbolic series whose $\min\{\widetilde{I}(X_S;Y_S), \widetilde{I}(Y_S;X_S)\}$ is at least μ are added to X_C (lines 6-7). Next, we proceed to iterate through each series within X_C in order to extract frequent single events (lines 8-9). Following this, each pair of events within the respective series of X_C is utilized to look for frequent 2-event patterns (lines 10-11). When dealing with frequent k-event patterns ($k \geq 3$), the mining process resembles the exact FTPM (lines 12-13).

2.6 Experimental Evaluation

We assess the performance of both the exact and approximate versions of FTPM on real-world datasets originating from diverse application domains: energy, smart city, sign language, and health. Moreover, we generate synthetic datasets with 10 times more sequences and 1000 time series from real-world datasets to assess the scalability.

2.6.1 Experimental Design

Datasets: We use six real-world datasets, i.e., NIST [24], UKDALE [44], DataPort [16], *Smart City* (SC) [11], American Sign Language (ASL) [55], and *Influenza* (INF) [12]. Three energy datasets, such as NIST, UKDALE, and DataPort, measure the energy usage of electrical applicances in households. The SC dataset is collected from NYC Open Data Portal. The ASL dataset contains annotated videos of signs and gestures in America. The INF dataset contains the influenza data from Kawasaki, Japan.
Baseline methods: The exact FTPM version is denoted as E-FTPM, and the approximate version as A-FTPM. Four baselines are used in the experiment: Z-Miner [51], TPMiner [10], IEMiner [58], and H-DFS [57].

2.6.2 Experimental Results

We only report the most important resuts here, the other results can be found in [37].
Qualitative Evaluation: Table 2.4 lists several interesting frequent patterns from the energy datasets and ASL. Patterns P1 - P7 pertain to the energy datasets, which reveal how citizens interact with electrical appliances in their homes. For instance, P3 indicates that the citizens might turn on the light at the hall entry in the late afternoon, suggesting that the citizens may have just come home. They then start preparing dinner around 18:00 by turning on the light and device plugs in the kitchen, then a few minutes later with the microwave. These patterns provide insights into citizens' living habits, enabling action for power optimization, such as pre-heating water for showers when surplus electricity from wind is redundant at night.

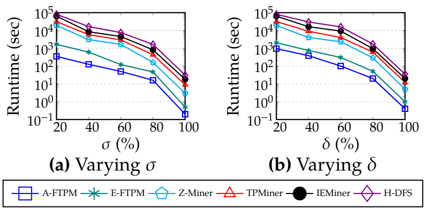
We obtain patterns P8 - P10 from the ASL dataset that depict the associations between various linguistic gestures and signs. These patterns serve a practical purpose, enabling automated translation from recorded video to text. As an example, P8 shows that a negation sign would encompass a leftward head tilt and a downward movement of the eyebrows gesture. P10 shows a Wh-question would involve a low movement of the eyebrows followed by a rapid opening and closing of the eyes.

Table 2.4: Summary of Interesting Frequent Patterns [37]

Patterns	σ (%)	δ (%)
(P1) ([05:58, 08:24] First Floor Lights) $\geqslant$ ([05:58, 06:59] Upstairs Bathroom Lights) $\geqslant$ ([05:59, 06:06] Microwave)	20	30
(P2) ([18:00, 18:30] Lights Dining Room) $\rightarrow$ ([18:31, 20:16] Children Room Plugs) $\between$ ([19:00, 22:31] Lights Living Room)	20	20
(P3) ([15:59, 16:05] Hallway Lights) $\rightarrow$ ([17:58, 18:29] Kitchen Lights $\geqslant$ ([18:00, 18:18] Plug In Kitchen) $\geqslant$ ([18:08, 18:15] Microwave)	20	25
(P4) ([06:02, 06:19] Kitchen Lights) $\rightarrow$ ([06:05, 06:12] Microwave) $\between$ ([06:09, 06:11] Kettle)	20	35
(P5) ([16:45, 17:30] Washer) $\rightarrow$ ([17:40,18:55] Dryer) $\rightarrow$ ([19:05, 20:10] Dining Room Lights) $\geqslant$ ([19:10, 19:30] Cooktop)	10	30
(P6) ([06:10, 07:00] Kitchen Lights) $\geqslant$ ([06:10, 06:15] Kettle) $\rightarrow$ ([06:30, 06:40] Toaster) $\rightarrow$ ([06:45, 06:48] Microwave)	25	40
(P7) ([18:00, 18:25] Kitchen Lights) $\geqslant$ ([18:00, 18:05] Kettle) $\rightarrow$ ([18:05, 18:10] Microwave) $\rightarrow$ ([19:35, 20:50] Washer)	20	40
(P8) [2.12 seconds] Negation $\geqslant$ [0.27 seconds] Lowered Eye-brows	10	10
(P9) [2.04 seconds] Negation $\geqslant$ [0.52 seconds] Rapid Shake-head	10	10
(P10) [1.53 seconds] Wh-question $\geqslant$ [0.36 seconds] Lowered Eye-brows $\rightarrow$ [0.05 seconds] Blinking Eye-aperture	10	15

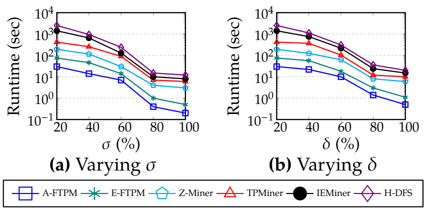
Quantitative evaluation with baselines comparison on real-world datasets: We compare our algorithms with the baselines on real-world datasets. Figs. 2.7 , 2.8 , 2.9 , and 2.10 show the experimental results on NIST and SC datasets.

In terms of runtime, Figs. 2.7 and 2.8 show that among all the methods, A-FTPM exhibits the fastest runtime, while E-FTPM shows a runtime faster than the baselines. Compared with other methods, the range and average speedups of A-FTPM are [1.5-6.1] and 2.7 (E-FTPM), [4.2-356.1] and 45.8 (all baselines). Compared with the baselines, the range and average speedup of E-FTPM are [2.6-130.4] and 24.7.



(a) Varying σ (b) Varying δ

Fig. 2.7: Runtime Comparison on NIST (real-world) [37]



(a) Varying σ (b) Varying δ

Fig. 2.8: Runtime Comparison on SC (real-world) [37]

In memory consumption, Figs. 2.9 and 2.10 show that A-FTPM consumes the least memory, while E-FTPM consumes less memory than the baselines.

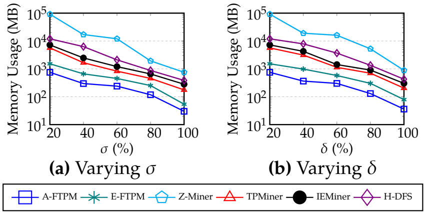


Fig. 2.9: Memory Usage Comparison on NIST (real-world) [37]

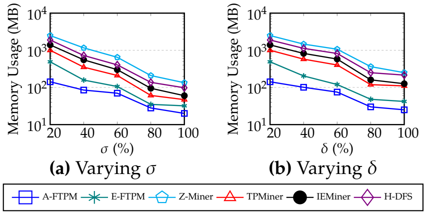


Fig. 2.10: Memory Usage Comparison on SC (real-world) [37]

In average, A-FTPM consumes 1.9 times less memory than E-FTPM, and 15.4 times less memory than the baselines. E-FTPM consumes 5.8 times less memory than the baselines in average.

Scalability evaluation on synthetic datasets: To further evaluation the performance of our algorithms, we conduct the experiment on synthetic datasets. We generated a collection of $1,000$ synthetic time series for each real-world dataset. We compare our algorithms with the baselines by increasing the number of time series, as shown Figs. 2.11 and 2.12 . As a result, A-FTPM has higher speedup when with the number of time series is large. The speedups of A-FTPM are: [2.4-6.1] and 3.2 on avg. (E-FTPM), [5.3-78.1] and 35.8 on avg. (all baselines), and of E-FTPM is: [2.6-27.4] and 14.7 on avg. (all baselines).

In Figs. 2.11 and 2.12 , an bar chart for A-FTPM is added, with the top red bar being the time to compute MI and μ. The bar is only for comparison, and is not actually used. Moreover, the baselines fail for the large configurations, e.g., Z-Miner, TPMiner, IEMiner and H-DFS when the number of time series grows up to 1000 (Fig. 2.11a). We can see that A-FTPM and E-FTPM can scale well on big datasets, unlike the baselines.

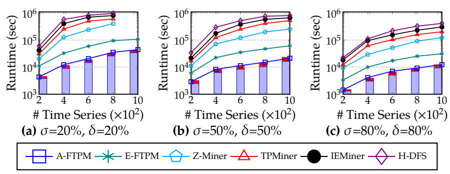


Fig. 2.11: Varying # of time series on NIST (synthetic) [37]

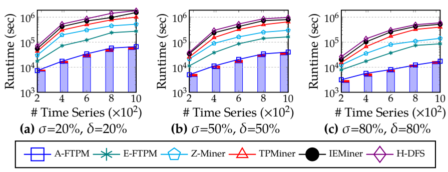


Fig. 2.12: Varying # of time series on SC (synthetic) [37]

Table 2.5: The Accuracy of A-FTPM (%) [37]

σ (%)	δ (%)							
	NIST				SC			
	10	20	50	80	10	20	50	80
10	87	89	91	94	78	83	98	100
20	96	89	91	94	83	83	98	100
50	100	100	96	94	99	99	98	100
80	100	100	100	100	100	100	100	100

Accuracy evaluation of A-FTPM: We evaluate the accuracy of A-FTPM by comparing the patterns extracted by A-FTPM and E-FTPM. Table 2.5 shows the accuracy of A-FTPM. A-FTPM achieves a high level of accuracy ($\geq$ 78%) when the values of σ and δ are low, such as $\sigma = \delta = 10\%$. It achieves a very high accuracy ($\geq$ 95%) when σ and δ are high, such as $\sigma = \delta = 50\%$.

Chapter 3

Rare Temporal Pattern Mining

This chapter gives an overall summarization of the rare temporal pattern mining problem presented in Paper B [37]. The chapter uses content from the paper in the most effective way.

3.1 Problem Motivation and Statement

Appearing with infrequent occurrence but with high confidence in a given database, rare temporal patterns still hold substantial interest and usefulness. However, there are two challenges when mining rare temporal patterns. First, it is a costly process due to the inclusion of temporal information for each event and the complex relations between events, resulting in a huge search space. Second, setting a low support threshold to identify rare patterns causes a combinatorial explosion of the search space. Consequently, the development of an efficient approach for mining rare temporal patterns becomes crucial.

The exploration of finding rare patterns has gained attention in recent years. Such techniques have been proposed in [5, 20, 68] for identifying rare motifs in time series. However, these approaches focus on repeated time series subsequences without considering temporal events; thus, they are insufficient for mining rare temporal patterns. Alternative methods such as rare association rules [1, 6–9, 15, 19, 42, 52, 59] and rare sequential patterns [40, 56, 61–63, 70] have been explored, but they do not consider temporal events and the temporal relationships between them. To the best of our knowledge, no existing research has explicitly addressed the mining of rare temporal patterns.

We focus on addressing the above problem with three proposals. First, we present an algorithm for rare temporal pattern mining efficiently (RTPM).

Second, we introduce an approximate version of RTPM that leverages mutual information to retain the most promising time series, thereby reducing the overall search space. Third, we propose a generalized algorithm that can mine both frequent and rare temporal patterns. Our main contributions include the following:

- We introduced the first solution for rare temporal pattern mining (STPM) that leverages an efficient data structure and pruning techniques to reduce the search space.
- Additionally, we developed an approximate version of STPM that utilizes mutual information to mine rare temporal patterns solely from the most promising time series, thus enhancing the scalability of the mining process.
- We proposed the efficient generalized temporal pattern mining (GTPM) to mine both frequent and rare temporal patterns.
- Extensive experiment evaluations are performed to assess the proposed algorithms.

3.2 Rare Temporal Pattern Mining Problem

The difference between *frequent temporal pattern* and *rare temporal pattern*. Both frequent temporal patterns and rare temporal patterns use the *support* and *confidence* measures to assess the frequency and the likelihood of a temporal pattern. However, the utilization of these measures varies for each pattern type. Let's consider a temporal pattern P, where the support is denoted as $\sigma = supp(P)$ and the confidence as $\delta = conf(P)$. If both σ and δ are high, indicating a large presence of P in the database, P is classified as a frequent temporal pattern. On the other hand, if σ is low and δ is high, suggesting rare occurrences but with high confidence, P is considered a rare temporal pattern. The characteristic of the rare pattern is that its support is low. Thus, we use $\sigma_{\max}$ as the upper bound for the support, and only find the rare temporal patterns such that $\sigma \leq \sigma_{\max}$. We assign $\sigma_{\max} = \infty$ if we mine frequent temporal patterns.

Problem Formulation: Rare Temporal Pattern Mining from Time Series (RTPMfTS) [37]
Given a set of univariate time series $\mathcal{X} = \{X_1, ..., X_n\}$, let $\mathcal{D}_{\text{SEQ}}$ be the temporal sequence database obtained from $\mathcal{X}$, and $\sigma_{\min}$, $\sigma_{\max}$, and δ be the minimum support, maximum support, and minimum confidence thresholds, respectively. The RTPMfTS problem aims to find all temporal patterns P that have low support and high confidence in $\mathcal{D}_{\text{SEQ}}$: $\sigma_{\min} \leq supp(P) \leq \sigma_{\max} \wedge conf(P) \geq \delta$.

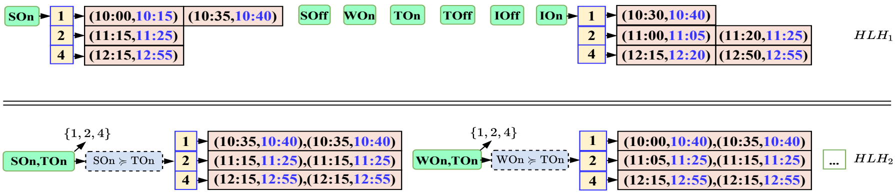


Fig. 3.1: An example of a hierarchical hash tables [37]

3.3 Rare Temporal Pattern Mining (Exact RTPM)

This section proposes a solution for mining rare temporal patterns. The lemmas are reproduced from Paper B [37], and detailed proofs of the lemmas can be found in Paper B.

There are two phases in the RTPMfTS process. The first phase is *Data Transformation* which converts the set of time series $\mathcal{X}$ to $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$. The second phase is *Rare Temporal Pattern Mining* which includes three steps: Mining Single Events, Mining Rare 2-Event Patterns, and Mining Rare k-Event Patterns ($k > 2$).

In the rare temporal pattern mining phase, the Hierarchical Lookup Hash structures with two types of tables (HLH_1 and HLH_k) mentioned in Section 2.4.1 are used to store the event/pattern candidates, i.e., the patterns satisfying the two constraints σ_{min} and δ. The details of using the data structure are explained in each mining step.

3.3.1 Mining Single Events

This step finds single events that satisfy σ_{min}. To do that, we first scan $\mathcal{D}_{\text{SEQ}}$ to compute the support of each event E_i, then compare against σ_{min}. It is important to note that two constraints, δ and σ_{max}, are not considered at this step. The confidence of an event is always 1 in this case, and we also do not utilize σ_{max} due to the reasoning presented in the following lemma.

Lemma 1 *Let P be a temporal pattern and E_i be a single event such that $E_i \in P$. Then $supp(P) \leq supp(E_i)$.*

From Lemma 1 , if $supp(E_i) > \sigma_{\text{max}}$, then E_i can still form a pattern P that has $supp(P) \leq \sigma_{\text{max}}$. In order to prevent the possibility of missing out on potential patterns, σ_{max} will not be used at this step.

We provide a running example in Fig. 3.1 using $\mathcal{D}_{\text{SEQ}}$ in [37] with $\sigma_{\text{min}} = 0.7$, $\sigma_{\text{max}} = 0.9$, and $\delta = 0.7$. We have 7 events that satisfy σ_{min}.

3.3.2 Mining Rare 2-event Patterns

We conduct two steps to mine rare 2-event patterns: (1) it first finds event pairs that satisfy $\sigma_{\min}$ and δ, (2) it then finds rare 2-event temporal patterns from the found event pairs.

Mining event pairs with the constraints $\sigma_{\min}$ and δ: First, we generate all possible pairs of events. Then, for each pair (E_i, E_j), we retrieve the set of sequences $\mathcal{S}_{ij}$ in which (E_i, E_j) occurs, and calculate the support of (E_i, E_j) base on $\mathcal{S}_{ij}$. If (E_i, E_j) satisfies both $\sigma_{\min}$ and δ, they are kept and used to mine rare 2-event patterns. Note that the constraint $\sigma_{\max}$ is not taken into account here to prevent the loss of rare 2-event temporal patterns (Lemma 1).

Mining rare 2-event temporal patterns: We iterate each event pair (E_i, E_j) in the above step to find the temporal relations between E_i and E_j. The relations R satisfying both $\sigma_{\min}$ and δ are stored in HLH_2. Next, for each relation r in R, we compare the support of r against $\sigma_{\max}$. If r satisfies $\sigma_{\max}$, r is a rare pattern. We also note that HLH_2 only stores patterns that satisfy only the two constraints, $\sigma_{\min}$ and δ. Fig. 3.1 provides some patterns, e.g., at event pair (WOn,TOn).

3.3.3 Mining Rare k-event Patterns

We mine rare k-event patterns similarly to rare 2-event patterns mining, including mining k-event combinations satisfying the constraints $\sigma_{\min}$ and δ, and mining rare k-event patterns. Moreover, we apply the transitivity property of temporal relations to further optimize the mining.

Mining k-event combinations with the constraints $\sigma_{\min}$ and δ: We first calculate the Cartesian product between the (k-1)-event combinations in HLH_{k-1} and the single events in HLH_1 to generate k-event combinations. Then, the k-event combinations that satisfy $\sigma_{\min}$ and δ are kept for mining rare k-event patterns.

However, it has been observed that single events in HLH_1 may not generate any patterns in HLH_k satisfying the $\sigma_{\min}$. For example, consider the event IOn in HLH_1 as shown in Fig. 3.1. In this case, we can combine IOn with (SOn, TOn) in HLH_2 to form a 3-event combination (SOn, TOn, IOn). However, we see that (SOn, TOn, IOn) cannot generate any frequent 3-event patterns whose support is at least $\sigma_{\min}$ because IOn does not exist in HLH_2. Therefore, it is unnecessary to create the combination (SOn, TOn, IOn). To address this, we utilize the transitivity property, explained as follows.

Lemma 2 *Let $N_{k-1} = (E_1, ..., E_{k-1})$ be a (k-1)-event combination and E_k be a single event, both satisfying the $\sigma_{\min}$ constraint. The combination $N_k = N_{k-1} \cup E_k$ can form k-event temporal patterns whose support is at least $\sigma_{\min}$ if $\forall E_i \in N_{k-1}$, $\exists r \in \mathfrak{R}$ s.t. $r(E_i, E_k)$ satisfies $\sigma_{\min}$.*

From Lemma 2, we should only consider single events in HLH_1 that also appear in HLH_{k-1} when generating combinations of k-events. Let *(k-1)Events* be the set of (k-1)-event combinations in HLH_{k-1} and *1Events* be the set of events in HLH_1. By using Lemma 2 , we retrieve distinct single events A_{k-1} from HLH_{k-1} and intersect them with the single events *1Events* to remove redundant events: *Candidate1Events* = $A_{k-1} \cap$ *1Events*. Subsequently, we generate k-event combinations by calculating the Cartesian product of *(k-1)Events* and *Candidate1Events*. Finally, we filter out only the k-event combinations which satisfy the $\sigma_{\min}$ and δ.

Mining rare k-event temporal patterns: This step mines rare k-event temporal patterns. Consider $D_{k-1} = (E_1, ..., E_{k-1})$ as a (k-1)-event combination, $D_1 = (E_k)$ as a single event, and $D_k = D_{k-1} \cup D_1 = (E_1, ..., E_k)$ as a k-event combination. To identify k-event patterns for D_k, we begin by retrieving the set P_{k-1} containing patterns for D_{k-1}. Each $p_{k-1} \in P_{k-1}$ is a list of triples: $\{(r_{12}, E_{1_{\triangleright e_1}}, E_{2_{\triangleright e_2}}),...,(r_{(k-2)(k-1)}, E_{k-2_{\triangleright e_{k-2}}}, E_{k-1_{\triangleright e_{k-1}}})\}$. We then perform an iterative check to determine if p_{k-1} and E_k can create a rare k-event pattern as follows. The process begins by examining the triple $(r_{(k-1)k}, E_{k-1_{\triangleright e_{k-1}}}, E_{k_{\triangleright e_k}})$ to verify if it meets the constraints of $\sigma_{\min}$ and δ by referring to the HLH_2. If this triple fails to satisfy these constraints, the checking process is immediately terminated for p_{k-1}. Contrarily, the process continues to the next triple $(r_{(k-2)k}, E_{k-2_{\triangleright e_{k-2}}}, E_{k_{\triangleright e_k}})$, and continues in a similar manner until it reaches the final triple $(r_{1k}, E_{1_{\triangleright e_1}}, E_{k_{\triangleright e_k}})$. Finally, we take a further step by selecting only k-event patterns in PH_k that satisfy the constraint $\sigma_{\max}$.

3.4 Approximate RTPM

In this section, we present an approximate version of RTPM, called Approximate RTPM, that mine rare temporal patterns only from the most promising time series. We use the normalized mutual information (NMI) defined in Section 2.5.1 to derive an upper bound of the support of an event pair. The theorems and corollaries are reproduced from Paper B [37], and their proofs can be found in Paper B.

3.4.1 Upper Bound of the Support

The approximate RTPM is built on the exact RTPM algorithm that uses mutual information to select dependent time series and then mines rare temporal patterns only on these dependent time series. The RTPMfTS problem uses three thresholds, minimum support, maximum support, and minimum confidence, in the mining. In this section, we derive the upper bound of the support of an event pair based on the mutual information between two symbolic series, and combine this upper bound with the support lower bound and the confidence lower bound in Sections 2.5.3 and 2.5.4 to prune the unpromising time series, improve RTPM's scalability in the approximate RTPM .

Consider two events X_1 and Y_1 from two symbolic time series X_S and Y_S, respectively. Now we derive the upper bound of the support of (X_1, Y_1) in $\mathcal{D}_{\text{SEQ}}$.

Theorem 1 *(Upper bound of the support)*
Let $\sigma_{\min}$ be the minimum support threshold, and $\mu_{\max}$ be the maximum mutual information threshold, respectively. Assume that $supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \sigma_{\min}$. If the NMI $\widetilde{I}(X_S;Y_S) \leq \mu_{\max}$, then the upper bound of the support of (X_1, Y_1) in $\mathcal{D}_{SEQ}$ is:

$$supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \leq \lambda_2 \cdot e^{W\left(\frac{\log \frac{\lambda_5^{1-\mu_{\max}}}{\lambda_4^{1-\sigma_{\min}}} \cdot \ln 2}{\lambda_2}\right)} + \vartheta \tag{3.1}$$

where: λ_2 is the support of $Y_1 \in Y_S$, λ_4 is the fraction between the support of $(X_i, Y_j) \in (X_S, Y_S)$ and the support of $Y_j \in Y_S$ such that $p(X_i|Y_j)$ is minimal, $\forall i \neq 1 \wedge j \neq 1$, λ_5 is the maximum support of $X_i \in X_S$, ϑ is the difference between the probabilities of (X_1, Y_1) in $\mathcal{D}_{SEQ}$ and $\mathcal{D}_{SYB}$, and W is the Lambert function [13].

From Theorem 1 , we can derive $\mu_{\max}$ such that *supp*(X_1, Y_1) is at most $\sigma_{\max}$.

Corollary 1.1 *The support of an event pair $(X_1, Y_1) \in (X_S, Y_S)$ in $\mathcal{D}_{SEQ}$ is at most $\sigma_{\max}$ if $\widetilde{I}(X_S; Y_S)$ is at most $\mu_{\max}$, where:*

$$\mu_{\max} \leq 1 - \frac{\frac{\sigma_{\max}-\vartheta}{\lambda_2} \cdot \log \frac{\sigma_{\max}-\vartheta}{\lambda_2} + \log \lambda_4^{1-\sigma_{\min}}}{\log \lambda_5} \tag{3.2}$$

Interpretation: Theorem 1 indicates that if $\widetilde{I}(X_S; Y_S)$ is at most $\mu_{\max}$, then the support of an event pair in (X_S, Y_S) is at most the upper bound in Eq. (3.1). Moreover, the support of a pattern is at most the support of the event pair forming that pattern. Thus, it can be inferred that if an event pair in (X_S, Y_S) has a support value lower than the upper bound, then any pattern created by that event pair also has support value lower than that upper bound.

3.4.2 Approximate RTPM

In this section, we will outline the approximate RTPM algorithm. Initially, we discuss how to determine the values of $\mu_{\min}$ and $\mu_{\max}$ for choosing the dependent time series. Subsequently, we provide a detailed explanation of the approximate RTPM algorithm.

Setting the values of $\mu_{\min}$ and $\mu_{\max}$: In RTPM, three user-defined parameters are utilized: the minimum support $\sigma_{\min}$, the maximum support $\sigma_{\max}$, and the minimum confidence δ. To satisfy both $\sigma_{\min}$ and δ constraints, we select $\mu_{\min}$ as described in Section 2.5.5 . To satisfy $\sigma_{\max}$ constraint, we determine $\mu_{\max}$ using Eq. (3.2) .

Approximate RTPM: Approximate RTPM focuses on mining patterns exclusively within the dependent symbolic series $\mathsf{X}_C \in \mathcal{X}$ with the $\mu_{\min}$ and $\mu_{\max}$ values. The process begins with a single pass scan of $\mathcal{D}_{\text{SYB}}$ to calculate the probabilities of single events, pair of events, and plus ϑ value. Subsequently, NMI, $\mu_{\min}$, and $\mu_{\max}$ are calculated for each pair of symbolic series. Series pairs that meet the condition of having $\min\{\widetilde{I}(X_S;Y_S), \widetilde{I}(Y_S;X_S)\}$ at least $\mu_{\min}$, and $\min\{\widetilde{I}(X_S;Y_S), \widetilde{I}(Y_S;X_S)\}$ at most $\mu_{\max}$ are added in X_C. Next, the single events are mined from each series in X_C. Following this, each event pair in X_C is employed to find rare 2-event patterns. For rare k-event patterns ($k \geq 3$), the mining process follows similar steps to that of RTPM.

3.5 Generalized Temporal Pattern Mining (GTPM)

Finally, we propose GTPM algorithm that combines both frequent and rare temporal patterns into one single mining process. It is important to highlight that when dealing with frequent temporal patterns, only two constraints $\sigma_{\min}$ and δ are employed.

Algorithm 3: Generalized Temporal Pattern Mining

Input: Temporal sequence database $\mathcal{D}_{\text{SEQ}}$, a minimum support threshold $\sigma_{\min}$, a maximum support threshold $\sigma_{\max}$, a minimum confidence threshold δ

Output: The set of temporal patterns P satisfying $\sigma_{\min}$, $\sigma_{\max}$, δ

//Mining single events

foreach *event* $E_i \in \mathcal{D}_{SEQ}$ **do**

    Find single events *1Events* that satisfy $\sigma_{\min}$;

//Mining 2-event patterns

EventPairs $\longleftarrow$ Cartesian(*1Events*,*1Events*);

foreach (E_i, E_j) *in EventPairs* **do**

    Find frequent event pairs *FrequentEventPairs* similarly to Sections 2.4.3 and 3.3.2 ;

foreach (E_i, E_j) *in FrequentEventPairs* **do**

    Find relations that satisfy $\sigma_{\min}$, $\sigma_{\max}$, and δ;

//Mining k-event patterns

Find frequent k-event combination *kEventCombinations* similarly to Sections 2.4.4 and 3.3.3 ;

foreach *kEvents in kEventCombinations* **do**

    Use iterative verification method to find temporal pattern against $\sigma_{\min}$, $\sigma_{\max}$, δ;

3.5.1 Exact Generalized Temporal Pattern Mining (Exact GTPM)

Algorithm 3 describes the mining process in Exact GTPM algorithm. First, we find single events that satisfy the minimum support σ_{min} (lines 1-2). We do not consider the constraints the minimum confidence δ and the maximum support σ_{max} here as in Sections 2.4.2 and 3.3.1 . Next, we mine 2-event patterns that include two steps: frequent event pairs mining (satisfying the constraints σ_{min} and δ) and 2-event temporal patterns mining (lines 3-7). For mining frequent event pairs, we proceed similarly to Sections 2.4.3 and 3.3.2 . For mining 2-event temporal patterns, we first find frequent relations R satisfying both σ_{min} and δ as in the frequent temporal patterns mining. To mine rare 2-event temporal patterns, we iterate every relation r in R and check the satisfaction of r with the constraint σ_{max}. Finally, we mine k-event patterns that consist of frequent k-event combinations mining (satisfying the constraints σ_{min} and δ) and k-event patterns mining (lines 8-10). For mining frequent k-event combinations, we also perform similarly to Sections 2.4.4 and 3.3.3 . For mining k-event patterns, we use the iterative verification method that relies on the transitivity property and the Apriori property, similarly in frequent and rare k-event pattern mining.

Algorithm 4: Approximate GTPM using Mutual Information

Input: A set of time series $\mathcal{X}$, a minimum support threshold σ_{min}, a maximum support threshold σ_{max}, a minimum confidence threshold δ

Output: The set of temporal patterns P

Convert $\mathcal{X}$ to $\mathcal{D}_{\text{SYB}}$ and convert $\mathcal{D}_{\text{SYB}}$ to $\mathcal{D}_{\text{SEQ}}$;
Scan $\mathcal{D}_{\text{SYB}}$ to compute the probability of each event, event pair, and plus ϑ value;
foreach *pair of symbolic time series* $(X_S, Y_S) \in \mathcal{D}_{SYB}$ **do**
  NMI $\leftarrow \min\{\widetilde{I}(X_S; Y_S), \widetilde{I}(Y_S; X_S)\}$;
  Calculate μ_{min}, and μ_{max};
  if $NMI \geq \mu_{\text{min}}$ **then**
    if $NMI \leq \mu_{\text{max}}$ **then**
      Add X_S and Y_S to X_C;
foreach $X_S \in \mathsf{X}_C$ **do**
  Find single events from X_S are similar to Exact GTPM;
foreach $(X_S, Y_S) \in \mathsf{X}_C$ **do**
  Find 2-event patterns from (X_S, Y_S) are similar to Exact GTPM;
if $k \geq 3$ **then**
  Find k-event patterns similar to Exact GTPM;

3.5.2 Approximate Generalized Temporal Pattern Mining (Approximate GTPM)

Similarly, we propose the approximate GTPM that integrates both frequent and rare temporal patterns into a single mining process. Algorithm 4 describes the mining steps in the approximate GTPM. First, we scan $\mathcal{D}_{\text{SYB}}$ to compute the probability of every single event, event pairs, and plus ϑ value (for mining rare temporal patterns). Next, we calculate NMI, $\mu_{\min}$, and $\mu_{\max}$ (for mining rare temporal patterns) for each symbolic series (lines 4-5). The pairs symbolic time series whose $\min\{\widetilde{I}(X_S;Y_S), \widetilde{I}(Y_S;X_S)\}$ satisfies $\mu_{\min}$ and $\min\{\widetilde{I}(X_S;Y_S), \widetilde{I}(Y_S;X_S)\}$ satisfies $\mu_{\max}$ (for mining rare temporal patterns) are added to X_C (lines 6-8). Then, we mine single events from symbolic series in X_C. Next, we traverse each pair of events in X_C to mine the 2-event patterns. Finally, we mine k-event patterns similar to the exact GTPM.

3.6 Experimental Evaluation

This section evaluates the exact and approximate RTPM algorithms in six real-world datasets from different domains: energy, smart city, sign language, and health.

3.6.1 Experimental Design

Datasets: We utilize a total of six real-world datasets for our experiment. There are 3 energy datasets that are NIST [24], UKDALE [44], and DataPort [16]. A smart city dataset, SC [11], is sourced from the NewYork city. The ASL dataset [55] comprises annotated videos in American Sign Language. Finally, the INF dataset [12] encompasses influenza-related data obtained from Kawasaki, Japan.
Baseline methods: Our exact RTPM version is denoted as E-RTPM and the approximate version is denoted as A-RTPM. As this is the first work for rare temporal pattern mining, there is no exact baseline method available for comparison with RTPM. However, we adapt an algorithm used for mining frequent temporal patterns, known as Z-Miner [51], to the task of identifying rare temporal patterns. This adapted version is named ARZ-Miner.

3.6.2 Experimental Results

In this section, we present the key results, the remaining results can be found in [37].
Qualitative Evaluation: Table 3.1 provides a collection of impressive rare temporal patterns. Patterns P1-P5 pertain to the SC dataset, while P6-P8 are derived from the INF dataset. Analyzing these patterns can uncover uncommon yet noteworthy relationships between temporal events. Specifically, P1-P5 reveal the association between extreme weather conditions and a high number of accidents. For instance, in P5, there is a notable pedestrian injury during heavy snowfall, which warrants significant attention despite its infrequent occurrence. On the other hand, P6-P8 highlight the correlation between

weather conditions and influenza cases. As an example, P6 demonstrates a rise in influenza cases when the temperature is freezing and snowfall is high, both of which are uncommon circumstances. The detection of such patterns contributes to disease prevention efforts in the initial phase.

Table 3.1: Summary of Interesting Rare Patterns [37]

Patterns	σ_{min} (%)	δ (%)	σ_{max} (%)
(P1) Heavy Rain ⩾ Unclear Visibility ⩾ Overcast Cloudiness → High Motorist Injury	5	30	9
(P2) Heavy Rain ≬ Strong Wind → High Motorist Injury	2	40	6
(P3) Very Strong Wind → High Motorist Injury	5	40	9
(P4) Strong Wind ≬ High Pedestrian Injury	4	30	8
(P5) Extremely Unclear Visibility ⩾ High Snow ⩾ High Pedestrian Injury	3	45	7
(P6) Frost Temperature ≬ High Snow ⩾ High Influenza	1	42	6
(P7) Low Temperature ⩾High Influenza	1	42	6
(P8) Heavy Rain ⩾ High Influenza	3	35	8

Quantitative evaluation with baselines comparison on real-world datasets: Figs. 3.2 , 3.3 , 3.4 , and 3.5 are the results on NIST and SC datasets. Figs. 3.2 and 3.3 show A-RTPM demonstrates the most efficient performance among all the methods, while E-RTPM is faster than the baseline. The speedup achieved by A-RTPM in comparison to other methods is: [1.9-7.2] (on average 3.4) when compared to E-RTPM, and [5.4-48.9] (on average 16.5) when compared to ARZ-Miner. Additionally, the speedup of E-RTPM is [2.9-24.7] (on average 7.4) when compared to ARZ-Miner.

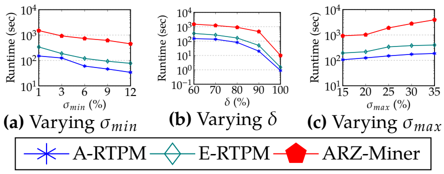


Fig. 3.2: Runtime Comparison on NIST (real-world) [37]

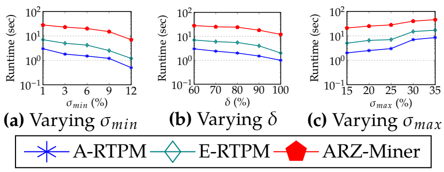


Fig. 3.3: Runtime Comparison on SC (real-world) [37]

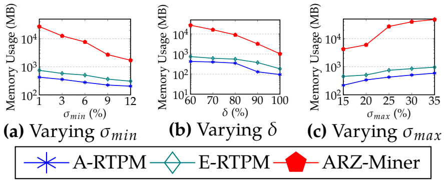


Fig. 3.4: Memory Usage Comparison on NIST (real-world) [37]

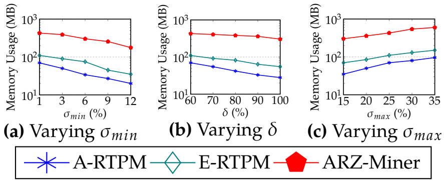


Fig. 3.5: Memory Usage Comparison on SC (real-world) [37]

Figs. 3.4 and 3.5 are the memory usage comparison among the different methods. It shows that A-RTPM has the least memory usage, followed by

E-RTPM which consumes less memory than ARZ-Miner. A-RTPM achieves a memory reduction of [1.6-3.9] times (on average 2.1) compared to E-RTPM, and [7.2-120.6] times (on average 24.1) compared to ARZ-Miner. Furthermore, E-RTPM demonstrates a memory reduction of [4.6-61.8] times (on average 14.7) compared to ARZ-Miner.

Scalability evaluation on synthetic datasets: In order to further evaluate the performance of our algorithms, we proceed with extensive experiments on synthetic datasets. From each real-world dataset, we generate a collection of $1,000$ synthetic time series for each corresponding dataset. Figures 3.6 and 3.7 show a comparison of the runtimes between the methods when varying the time series number. The speedup ranges and average speedups of A-RTPM are [3.5-7.4] and 4.6 (compared to E-RTPM), [7.2-24.8] and 15.2 (compared to ARZ-Miner), while the speedup range and average speedup of E-RTPM are [3.6-9.5] and 6.4 (compared to ARZ-Miner).

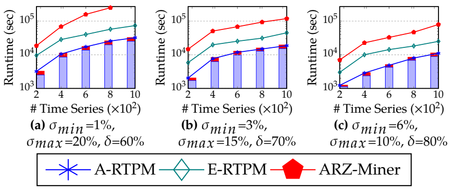


Fig. 3.6: Varying # of time series on NIST (synthetic) [37]

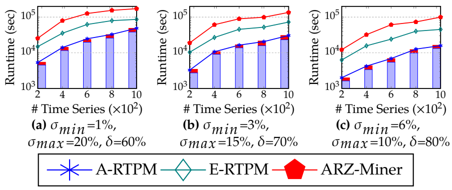


Fig. 3.7: Varying # of time series on SC (synthetic) [37]

Table 3.2: RTPM Accuracy (%) [37]

σ_{max} (%)	σ_{min} (%) - δ (%)					
	NIST			SC		
	1-60	3-70	6-80	1-60	3-70	6-80
10	93	96	100	91	93	100
15	86	92	95	86	91	100
20	84	92	92	83	87	90

Accuracy evaluation of A-STPM: In order to evaluate the accuracy of A-RTPM, we compare the extracted patterns from A-RTPM and E-RTPM. The accuracies of A-RTPM for different values of σ_{min}, δ, and σ_{max} on the real-world datasets are presented in Table 3.2. A-RTPM achieves high accuracy (at least 83%) with the lowest values of σ_{min} and δ, and the highest value of σ_{max}, e.g., $\sigma_{min} = 1\%$, $\delta = 60\%$, and $\sigma_{max} = 20\%$. A-RTPM achieves a very high accuracy (at least 93%) with higher values of σ_{min} and δ, and lower value of σ_{max}, e.g., $\sigma_{min} = 3\%$, $\delta = 70\%$, $\sigma_{max} = 10\%$.

Chapter 4

Seasonal Temporal Pattern Mining

This chapter gives a comprehensive overview of Paper C [38]. Content from the paper is reused in the most effective way.

4.1 Problem Motivation and Statement

The characteristic of a seasonal temporal pattern is that it occurs during a particular period of time and repeats periodically. This characteristic results in three challenges when mining seasonal temporal patterns. First, traditional pattern mining methods often use the *support* measure to assess the frequency of a pattern. However, the support counts the occurrence of the pattern throughout the entire dataset. Thus, it cannot be used to detect the seasonality feature of seasonal patterns. Second, mining seasonal temporal patterns from time series is very expensive since temporal relations between events create an exponential search space, i.e., $O(n^h 3^{h^2})$ (n is the number of events and h is the length of temporal patterns). Third, the anti-monotonicity property is frequently used in mining methods [57] to reduce the search space, which cannot be applied to seasonal temporal patterns since this property is not upheld in the case of seasonal patterns, i.e., subsets of a seasonal temporal pattern may not necessarily exhibit seasonality. It is necessary to have a more correct and efficient approach to mining seasonal temporal patterns.

Various techniques have been developed to identify periodic sub-sequences in time series by treating seasonality as recurring occurrences. These techniques, initially introduced by Han et al. in [22, 23], and subsequently expanded upon by [4, 43, 53, 54, 69], are commonly referred to as motif discovery techniques. However, motif discovery only focuses on identifying similar sub-sequences in time series, which means it can only uncover recurring time series sub-sequences rather than capturing periodic temporal patterns. Another approach in this field focuses on periodic association rules [3, 18, 21, 25–34, 41, 45–49, 65, 66]. However, these techniques aim to only discover seasonal associations among itemsets. To the best of our knowledge, there is currently no prior research that addresses the mining of seasonal temporal patterns, specifically targeting the identification of seasonal occurrences of temporal patterns.

Our research aims to address two key objectives. First, designing an efficient algorithm for seasonal temporal pattern mining (STPM). Second, proposing an approximate version of STPM that incorporates mutual information to improve the scalability of the mining method. This approximate version selectively focuses on mining seasonal temporal patterns from the most promising time series, resulting in a notable speedup of the mining process. Our key contributions in this chapter are:

- We propose the first solution to mine seasonal temporal patterns from time series. To achieve this, several measures, including maximum period, minimum density, distance interval, and minimum seasonal occurrence, are introduced to evaluate the seasonality characteristics of temporal patterns.

- We propose an efficient Seasonal Temporal Pattern Mining (STPM) algorithm that introduces several novel aspects. First, STPM utilizes hierarchical hash tables, enabling rapid retrieval of candidate events and patterns throughout the mining process. Second, we introduce a novel measure called *maximum season*, which adheres to the anti-monotonicity property. This measure is subsequently utilized to establish the concept of a candidate seasonal temporal pattern, effectively eliminating infrequent seasonal temporal patterns. Additionally, we design two efficient pruning techniques: Apriori-like pruning and transitivity pruning, to accelerate the mining.

- To enhance the scalability on large datasets, we introduce an approximate version of STPM that utilizes mutual information. This approach effectively eliminates unpromising time series, improving the scalability of the mining process.

- We evaluate the performance of the proposed algorithms by conducting experiments on both real-world and synthetic datasets.

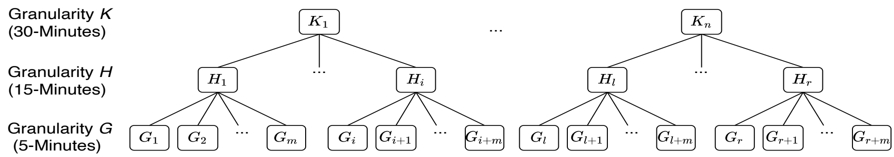


Fig. 4.1: Time granularity hierarchy $\mathcal{H}$ [38]

4.2 Preliminaries

In this section, we present several measures to capture the seasonality characteristics of seasonal temporal patterns. All definitions are reproduced from Paper C [38].

Definition 4.2.1 (Time granularity)
Given a time domain $\mathcal{T}$, a *time granularity* G is a *complete and non-overlapping equal partitioning* of $\mathcal{T}$, i.e., $\mathcal{T}$ is divided into non-overlapping equal partitions.

Each non-empty partition $G_i \in G$ is called a (time) *granule*. The position of a granule G_i in G, denoted as $p(G_i)$, is identified by counting the number of granules which appear before and up to (including) G_i. The *period* between two granules G_i and G_j in granularity G measures the time duration between G_i and G_j, and is computed as: $pr_{ij} = |p(G_i) - p(G_j)|$, where $p(G_i)$ and $p(G_j)$ are the positions of G_i and G_j, respectively.

> **Example 4.2.1 (Time granularity and period of two granules)**
> Consider a time domain $\mathcal{T}$ which is a sequential collection of minutes. Within this domain, there are various levels of time granularity, including Minute, 5-Minutes, and even larger units such as Hour, Day, and Year. When considering the Minute granularity, the period between the granules Minute_2 and Minute_5 can be determined as: $|p(\text{Minute}_5) - p(\text{Minute}_2)| = 3$.

Definition 4.2.2 (Time granularity hierarchy)
A time granularity G is *finer* than a time granularity H if and only if for every granule $H_j \in H$, there exists m adjacent granules $G_{i+1}, ..., G_{i+m} \in G$ such that $H_j = G_{i+1} \cup ... \cup G_{i+m}$ where $m \geq 1$. We call G is *m-Finer* than H, denoted as $G \trianglelefteq_m H$.

Given a time domain $\mathcal{T}$, the different time granularities of $\mathcal{T}$ form a *time granularity hierarchy* $\mathcal{H}$ where each level in $\mathcal{H}$ represents one specific granularity, with the lower levels in the hierarchy having finer granularity than the higher levels.

Table 4.1: A Symbolic Database $\mathcal{D}_{\text{SYB}}$ [38]

Granules in G		G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}	G_{13}	G_{14}	G_{15}	G_{16}	G_{17}	G_{18}	G_{19}	G_{20}	G_{21}
Position		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
Time series	**C**	1	1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1
	D	1	0	0	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	1	1	1
	F	0	0	1	0	1	1	0	0	1	0	0	1	1	1	1	0	0	0	0	0	0
	M	1	1	1	1	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0
	N	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0

Granules in G		G_{22}	G_{23}	G_{24}	G_{25}	G_{26}	G_{27}	G_{28}	G_{29}	G_{30}	G_{31}	G_{32}	G_{33}	G_{34}	G_{35}	G_{36}	G_{37}	G_{38}	G_{39}	G_{40}	G_{41}	G_{42}
Position		**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**
Time series	**C**	1	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	1	0
	D	1	1	1	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	1	1	0
	F	0	0	0	1	1	1	1	1	1	0	0	1	0	0	1	0	0	1	0	0	1
	M	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
	N	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0

> **Example 4.2.2 (A time granularity hierarchy)**
> An example of the hierarchy of time granularities is shown in Fig. 4.1. Assuming that the finest granularity G contains granules of 5 minutes, granularity H contains granules of 15 minutes and $G \trianglelefteq_3 H$.

Definition 4.2.3 (Time series)
A *time series* $X = x_1, x_2, ..., x_n$ in the time domain $\mathcal{T}$ is a chronologically ordered sequence of data values measuring the same phenomenon during an observation time period in $\mathcal{T}$. We say that X has granularity G if X is sampled at every time instant t_i in $\mathcal{T}$.

The mapping function f: $X \rightarrow \Sigma_X$ is employed to transform each value $x_i \in X$ into a symbol $\omega \in \Sigma_X$, thereby producing a sequence of symbols, called a *symbolic time series* X_S of X [35]. The symbol alphabet Σ_X is a finite collection of symbols utilized to encode X. As f establishes a one-to-one mapping between X and X_S, X_S maintains the same granularity G as X.

> **Example 4.2.3 (A symbolic time series)**
> Let $\Sigma_X = \{1, 0\}$ (1 representing ON and 0 representing OFF), the energy usage time series of an electrical appliance X, consisting of the values 1.9, 1.5, 0.2, 0.1, and 0.0. We have X_S = 1, 1, 0, 0, 0.

Definition 4.2.4 (Symbolic database)
The set of symbolic representations of a given set of time series $\mathcal{X} = \{X_1, ..., X_n\}$ forms a *symbolic database* $\mathcal{D}_{\text{SYB}}$.

> **Example 4.2.4 (A symbolic database)**
> A symbolic database $\mathcal{D}_{\text{SYB}}$ is shown in Table 4.1. 5 time series: {C, D, F, M, N} use the same symbol alphabet Σ = {0, 1}.

Definition 4.2.5 (Temporal event)
A *temporal event* E in a symbolic time series X_S is a tuple $E = (\omega, T)$. Here, $\omega \in \Sigma_X$ is a symbol, and $T = \{[t_{s_i}, t_{e_i}]\}$ is the set of time intervals during which X_S has the value ω. Each time interval has t_{s_i} as the start time, and t_{e_i} as the end time.

The tuple $e = (\omega, [t_{s_i}, t_{e_i}])$ is called an *instance* of the temporal event E, denoted as $E_{\triangleright e}$.

> **Example 4.2.5 (A temporal event and an event instance)**
> Let's examine the symbolic series D presented in Table 4.1. In this context, (D:1) is an event of D that consists of the time intervals, i.e., (D:1, $\{[G_1, G_1], [G_4, G_4], [G_7, G_8], [G_{10}, G_{11}], [G_{19}, G_{24}], [G_{31}, G_{31}], [G_{34}, G_{34}], [G_{37}, G_{38}], [G_{40}, G_{41}]\}$). And (D:1, $[G_1, G_1]$) is an event instance of (D:1). For simplicity, we utilize granules to signify the start and end times of the time intervals, as we can track the corresponding timestamps for each granule.

Relations between temporal events: Similar to frequent temporal patterns, we define 3 temporal relations among two events, including *Follows*, *Contains*, and *Overlaps*. The conditions of these 3 relations can be referred to [38].

Definition 4.2.6 (Temporal pattern)
Assume the set of temporal relations to be $\mathfrak{R}$={Follows, Contains, Overlaps}. A *temporal pattern* P = <(r_{12}, E_1, E_2),...,$(r_{(n-1)(n)}, E_{n-1}, E_n)$> contains triples (r_{ij}, E_i, E_j), each represents a temporal relation $r_{ij} \in \mathfrak{R}$ between E_i and E_j.

Definition 4.2.7 (Temporal sequence of a symbolic time series)
Let X_S be a symbolic time series at the granularity G, a granularity H belonging to $\mathcal{H}$, and $G \trianglelefteq_m H$.

We define a sequence mapping function $g : X_S \rightarrow_m H$ groups m consecutive symbols from X_S into a single granule $H_i \in H$.

Let $<\omega_1, ..., \omega_m>$ be a symbolic sequence at granule H_i in H, obtained by performing a sequence mapping g: $X_S \rightarrow_m H$. A *temporal sequence* $Seq_i =< e_1, ..., e_n >$ is a list of n event instances, each is obtained by grouping consecutive and identical symbols ω in H_i into an event instance $e = (\omega, [t_s, t_e])$.

Consider a symbolic database $\mathcal{D}_{\text{SYB}}$ including a set of symbolic time series $\{X_S\}$. We use $g : X_S \rightarrow_m H$ to map each X_S in $\mathcal{D}_{\text{SYB}}$ to sequences. The set of temporal sequences obtained from g create a *temporal sequence database* $\mathcal{D}_{\text{SEQ}}$.

Table 4.2: A Temporal Sequence Database $\mathcal{D}_{\text{SEQ}}$ [38]

Granules	Position	Temporal sequences
$\boldsymbol{H_1}$={$\boldsymbol{G_1}$,$\boldsymbol{G_2}$,$\boldsymbol{G_3}$}	1	(C:1,[G_1, G_2]), (C:0,[G_3, G_3]), (D:1,[G_1, G_1]), (D:0,[G_2, G_3]), (F:0,[G_1, G_2]), (F:1,[G_3, G_3]), (M:1,[G_1, G_3]), (N:1,[G_1, G_2]), (N:0,[G_3, G_3])
$\boldsymbol{H_2}$={$\boldsymbol{G_4}$,$\boldsymbol{G_5}$,$\boldsymbol{G_6}$}	2	(C:1,[G_4, G_4]), (C:0,[G_5, G_6]), (D:1,[G_4, G_4]), (D:0,[G_5, G_6]), (F:0,[G_4, G_4]), (F:1,[G_5, G_6]), (M:1,[G_4, G_4]), (M:0,[G_5, G_6]), (N:1,[G_4, G_6])
$\boldsymbol{H_3}$={$\boldsymbol{G_7}$,$\boldsymbol{G_8}$,$\boldsymbol{G_9}$}	3	(C:1,[G_7, G_8]), (C:0,[G_9, G_9]), (D:1,[G_7, G_8]), (D:0,[G_9, G_9]), (F:0,[G_7, G_8]), (F:1,[G_9, G_9]), (M:1,[G_7, G_9]), (N:1,[G_7, G_9])
$\boldsymbol{H_4}$={$\boldsymbol{G_{10}}$,$\boldsymbol{G_{11}}$,$\boldsymbol{G_{12}}$}	4	(C:0,[G_{10}, G_{12}]), (D:1,[G_{10}, G_{11}]), (D:0,[G_{12}, G_{12}]), (F:0,[G_{10}, G_{11}]), (F:1,[G_{12}, G_{12}]), (M:1,[G_{10}, G_{11}]), (M:0,[G_{12}, G_{12}]), (N:1,[G_{10}, G_{11}]), (N:0,[G_{12}, G_{12}])
$\boldsymbol{H_5}$={$\boldsymbol{G_{13}}$,$\boldsymbol{G_{14}}$,$\boldsymbol{G_{15}}$}	5	(C:0,[G_{13}, G_{15}]), (D:0,[G_{13}, G_{15}]), (F:1,[G_{13}, G_{15}]), (M:1,[G_{13}, G_{15}]), (N:1,[G_{13}, G_{15}])
$\boldsymbol{H_6}$={$\boldsymbol{G_{16}}$,$\boldsymbol{G_{17}}$,$\boldsymbol{G_{18}}$}	6	(C:0,[G_{16}, G_{18}]), (D:0,[G_{16}, G_{18}]), (F:0,[G_{16}, G_{18}]), (M:1,[G_{16}, G_{18}]), (N:1,[G_{16}, G_{18}])
$\boldsymbol{H_7}$={$\boldsymbol{G_{19}}$,$\boldsymbol{G_{20}}$,$\boldsymbol{G_{21}}$}	7	(C:1,[G_{19}, G_{21}]), (D:1,[G_{19}, G_{21}]), (F:0,[G_{19}, G_{21}]), (M:0,[G_{19}, G_{21}]), (N:0,[G_{19}, G_{21}])
$\boldsymbol{H_8}$={$\boldsymbol{G_{22}}$,$\boldsymbol{G_{23}}$,$\boldsymbol{G_{24}}$}	8	(C:1,[G_{22}, G_{24}]), (D:1,[G_{22}, G_{24}]), (F:0,[G_{22}, G_{24}]), (M:1,[G_{22}, G_{24}]), (N:0,[G_{22}, G_{24}])
$\boldsymbol{H_9}$={$\boldsymbol{G_{25}}$,$\boldsymbol{G_{26}}$,$\boldsymbol{G_{27}}$}	9	(C:0,[G_{25}, G_{27}]), (D:0,[G_{25}, G_{27}]), (F:1,[G_{25}, G_{27}]), (M:1,[G_{25}, G_{27}]), (N:1,[G_{25}, G_{27}])
$\boldsymbol{H_{10}}$={$\boldsymbol{G_{28}}$,$\boldsymbol{G_{29}}$,$\boldsymbol{G_{30}}$}	10	(C:0,[G_{28}, G_{30}]), (D:0,[G_{28}, G_{30}]), (F:1,[G_{28}, G_{30}]), (M:1,[G_{28}, G_{30}]), (N:1,[G_{28}, G_{30}])
$\boldsymbol{H_{11}}$={$\boldsymbol{G_{31}}$,$\boldsymbol{G_{32}}$,$\boldsymbol{G_{33}}$}	11	(C:1,[G_{31}, G_{31}]), (C:0,[G_{32}, G_{33}]), (D:1,[G_{31}, G_{31}]), (D:0,[G_{32}, G_{33}]), (F:0,[G_{31}, G_{32}]), (F:1,[G_{33}, G_{33}]), (M:1,[G_{31}, G_{33}]), (N:1,[G_{31}, G_{33}])
$\boldsymbol{H_{12}}$={$\boldsymbol{G_{34}}$,$\boldsymbol{G_{35}}$,$\boldsymbol{G_{36}}$}	12	(C:1,[G_{34}, G_{35}]), (C:0,[G_{36}, G_{36}]), (D:1,[G_{34}, G_{34}]), (D:0,[G_{35}, G_{36}]), (F:0,[G_{34}, G_{35}]), (F:1,[G_{36}, G_{36}]), (M:0,[G_{34}, G_{36}]), (N:1,[G_{34}, G_{36}])
$\boldsymbol{H_{13}}$={$\boldsymbol{G_{37}}$,$\boldsymbol{G_{38}}$,$\boldsymbol{G_{39}}$}	13	(C:0,[G_{37}, G_{39}]), (D:1,[G_{37}, G_{38}]), (D:0,[G_{39}, G_{39}]), (F:0,[G_{37}, G_{38}]), (F:1,[G_{39}, G_{39}]), (M:1,[G_{37}, G_{39}]), (N:1,[G_{37}, G_{39}])
$\boldsymbol{H_{14}}$={$\boldsymbol{G_{40}}$,$\boldsymbol{G_{41}}$,$\boldsymbol{G_{42}}$}	14	(C:1,[G_{40}, G_{41}]), (C:0,[G_{42}, G_{42}]), (D:1,[G_{40}, G_{41}]), (D:0,[G_{42}, G_{42}]), (F:0,[G_{40}, G_{41}]), (F:1,[G_{42}, G_{42}]), (M:0,[G_{40}, G_{42}]), (N:0,[G_{40}, G_{42}])

Example 4.2.6 (Temporal sequences)
Consider the symbolic series D in Table 4.1. A sequence mapping g: $D \rightarrow_3 H$ creates the granularity H containing the granules: H_1: <D:1, D:0, D:0>, H_2: <D:1, D:0, D:0>, H_3: <D:1, D:1, D:0>, and so forth. The temporal sequences at each granule in H are: Seq_1 = <(D:1, [G_1, G_1]), (D:0, [G_2, G_3])> at H_1, Seq_2 = <(D:1, [G_4, G_4]), (D:0, [G_5, G_6])> at H_2, Seq_3 = <(D:1, [G_7, G_8]), (D:0, [G_9, G_9])> at H_3, and so forth. An example of $\mathcal{D}_{\text{SEQ}}$ is shown in Table 4.2, obtained from $\mathcal{D}_{\text{SYB}}$ in Table 4.1 using the mapping $g : X_S \rightarrow_3 H$.

Definition 4.2.8 (Support set of a temporal pattern)
Let $\mathcal{D}_{\text{SEQ}}$ be a temporal sequence database with a granularity level of H, P be a temporal pattern. The set of granules H_i in $\mathcal{D}_{\text{SEQ}}$ where P occurs, arranged in an increasing order, is called the *support set* of temporal pattern P and is denoted as $\text{SUP}^P = \{H_l^P, ..., H_r^P\}$. The granule H_i at which event P occurs is denoted as H_i^P. The support set of a group of events, denoted as $\text{SUP}^{(E_i,...,E_k)}$, is defined similarly to that of a temporal pattern.

Definition 4.2.9 (Near support set of a temporal pattern)
Let $\text{SUP}^P = \{H_l^P, ..., H_r^P\}$ be the support set of a pattern P, *maxPeriod* be the *maximum period threshold* that represents the predefined maximal period between any two consecutive granules in SUP^P. The set SUP^P is called a *near support set* of P if $\forall(H_o^P, H_p^P) \in \text{SUP}^P$: ($H_o^P$and $H_p^P$are consecutive) $\wedge$ $|p(H_o^P) - p(H_p^P)| \leq$ *maxPeriod*, where $p(H_o^P)$ and $p(H_p^P)$ are the positions of H_o^P and H_p^P in granularity H. We denote the near support set of pattern P as NearSUP^P.

In an intuitive sense, the near support set of a pattern P contains a set of occurrences of P where they are closely positioned in time. Furthermore, a near support set NearSUP^P is *maximal* if it does not have any other superset besides itself that exists as a near support set. Similarly, the near support set of an event is defined in a similar manner as that of a pattern.

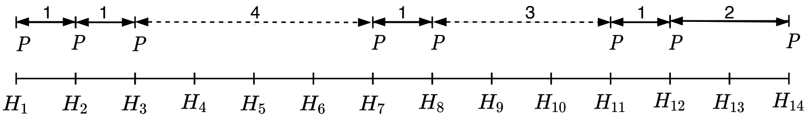


Fig. 4.2: Near support sets of pattern P = (Contains, C:1, D:1) [38]

> **Example 4.2.7 (A near support set)**
> Let's examine the pattern P = (Contains, C:1, D:1) from Table 4.2, and assume that the value of *maxPeriod* is 2. We have: SUP^P = $\{H_1, H_2, H_3, H_7, H_8, H_{11}, H_{12}, H_{14}\}$. Three maximal near support sets of P are: NearSUP$_1^P$ = $\{H_1, H_2, H_3\}$, NearSUP$_2^P$ = $\{H_7, H_8\}$, and NearSUP$_3^P$ = $\{H_{11}, H_{12}, H_{14}\}$, as illustrated in Fig. 4.2.

Definition 4.2.10 (Season of a temporal pattern))
Let NearSUP^P be a near support set of a pattern P and *minDensity* be a predefined minimum density threshold. Then NearSUP^P is called a *season* of P if *den*(NearSUP^P) $\geq$ *minDensity*, where *den*(NearSUP^P) = |NearSUP^P| is the number of granules in NearSUP^P, called the *density of* NearSUP^P.

> **Example 4.2.8 (Density of a near support set)**
> In Example 4.2.7, *den*(NearSUP$_1^P$) = 3, *den*(NearSUP$_2^P$) = 2, and *den*(NearSUP$_3^P$) = 3.

When P's occurrences are densely distributed, NearSUP^P becomes a season of P. Intuitively, a *season* of a temporal pattern represents a *concentrated period of occurrences*, followed by a *long gap period* with little to no occurrences, before the next season begins. The definition of a season for an event is defined in a similar manner as it is for a pattern.

The *distance* between two seasons NearSUP$_i^P$ = $\{H_k^P, ..., H_n^P\}$ and NearSUP$_j^P$ = $\{H_r^P, ..., H_u^P\}$ is calculated as: *dist*(NearSUP$_i^P$, NearSUP$_j^P$) = $|p(H_n^P) - p(H_r^P)|$.

Definition 4.2.11 (Frequent seasonal temporal pattern)
Let $\mathcal{PS} = \left\{\text{NearSUP}^P\right\}$ be the set of seasons of a temporal pattern P, *minSeason* be the *minimum seasonal occurrence* threshold, and *distInterval* = $[\text{dist}_{\text{min}}, \text{dist}_{\text{max}}]$ be the *distance interval* where dist_{min} is the minimum distance and dist_{max} is the maximum distance. A temporal pattern P is called a *frequent seasonal temporal pattern* iff $seasons(P) = |\mathcal{PS}| \geq minSeason \land \forall(\text{NearSUP}^P_i, \text{NearSUP}^P_j) \in \mathcal{PS}$: they are consecutive and $\text{dist}_{\text{min}} \leq dist(\text{NearSUP}^P_i, \text{NearSUP}^P_j) \leq \text{dist}_{\text{max}}$.

Intuitively, a pattern P is considered *seasonal* when the gap between two consecutive seasons falls within a predefined distance range. Furthermore, a seasonal temporal pattern is deemed *frequent* if it occurs more frequently than a specified threshold as the *minimum seasonal occurrence*. The number of seasons of a pattern P is calculated as $seasons(P) =| \mathcal{PS} |$.

Problem Formulation: Mining Frequent Seasonal Temporal Patterns from Time Series (FreqSTPfTS) [38]
Let $\mathcal{D}_{\text{SEQ}}$ be the temporal sequence database of granularity $H \in \mathcal{H}$ obtained from a given set of n time series $\mathcal{X} = \{X_1, ..., X_n\}$ of granularity $G_{\mathcal{X}}$. Let *maxPeriod*, *minDensity*, *distInterval*, and *minSeason* be the maximum period, minimum density, distance interval, and minimum seasonal occurrence thresholds, respectively. The FreqSTPfTS problem aims to find all frequent seasonal temporal patterns P in $\mathcal{D}_{\text{SEQ}}$ that satisfy the *maxPeriod*, *minDensity*, *distInterval*, and *minSeason* constraints.

4.3 Seasonal Temporal Pattern Mining (Exact STPM)

This section presents a solution for mining seasonal temporal patterns. The definitions and lemmas are reproduced from Paper C [38], and detailed proofs of the lemmas can be found in Paper C.

The FreqSTPfTS process consists of two phases: *Data Conversion* and *Seasonal Temporal Pattern Mining (STPM)*. In the Data Conversion phase, a set of time series is converted into a symbolic database $\mathcal{D}_{\text{SYB}}$ using the mapping function in Def. 4.2.3 , then $\mathcal{D}_{\text{SYB}}$ is converted into a sequence database $\mathcal{D}_{\text{SEQ}}$. In the Seasonal Temporal Pattern Mining phase, we conduct two mining steps: *Seasonal Single Event Mining* and *Seasonal k-Event Pattern Mining* ($k \geq 2$). Now, we introduce *candidate seasonal pattern* concept used for Apriori-like pruning in STPM.

4.3.1 Candidate Seasonal Pattern

In traditional mining methods, the *support* measure is used to prune the search space since it adheres the the *anti-monotonicity* property [57]. Assume that an event E_i is not frequent, then a pattern P formed by E_i is also not frequent, since support(E_i) ≥ support(P). Thus, E_i can be pruned safely along with its combinations from the search space, while still maintaining the completeness of the algorithm. However, seasonal temporal patterns do not adhere to this property.

> **Example 4.3.1 (Violation of the anti-monotonicity of seasonal patterns)**
> Let's examine an event E = M:1 and a 2-event pattern P = M:1 $\succcurlyeq$ N:1 (or (Contains,M:1,N:1)) as presented in Table 4.2 . Assume the following threshold values: *maxPeriod* = 2, *minDensity* = 3, *distInterval* = [4, 10], and *minSeason* = 2. E has the season: $\mathcal{PS}^E$ = {NearSUP$_1^E$} = $\{H_1, H_2, H_3, H_4, H_5, H_6, H_8, H_9, H_{10}, H_{11}, H_{13}\}$, and P has the seasons: $\mathcal{PS}^P$ = {{NearSUP$_1^P$} = $\{H_1, H_3, H_4, H_5, H_6\}$, {NearSUP$_2^P$} = $\{H_{10}, H_{11}, H_{13}\}$}. Hence, the number of seasons of E is: $|\mathcal{PS}^E|$=1 and of P is: $|\mathcal{PS}^P|$=2. As a result of the *minSeason* constraint, event E does not qualify as a frequent seasonal event, while pattern P does. This demonstrates that seasonal temporal patterns do not comply with the anti-monotonic property.

To enhance the performance of STPM, we introduce a novel measure, called *maxSeason*, which estimates the *maximum seasonal occurrence* of a pattern and adheres to the anti-monotonicity property.

Definition 4.3.1 (Maximum seasonal occurrence)
The maximum seasonal occurrence of a temporal pattern P is the ratio between the number of granules in the support set SUP^P of P, and the *minDensity* threshold:

$$maxSeason(P) = \frac{|SUP^P|}{minDensity} \tag{4.1}$$

If a pattern P' is a subset of the pattern P, then $|SUP^{P'}| \geq |SUP^P|$. Hence: $maxSeason(P') \geq maxSeason(P)$. Thus, *maxSeason* upholds the anti-monotonicity property. Moreover, from Eq. (4.1), *maxSeason* represents an upper limit of the number of seasons of a pattern.

The maximum seasonal occurrence of a group of events $(E_i, ..., E_k)$ is defined similarly to that of a temporal pattern. Assume that a pattern P is created by $(E_i, ..., E_k)$. Then: $maxSeason(E_i, ..., E_k) \geq maxSeason(P)$ since $|SUP^{(E_i,...,E_k)}| \geq |SUP^P|$.

Now we use *maxSeason* to define the concept *candidate pattern* that is used to prune infrequent seasonal patterns.

Definition 4.3.2 (Candidate seasonal pattern)
A temporal pattern P is a *candidate seasonal pattern* if $maxSeason(P) \geq minSeason$.

Similarly, a *candidate seasonal k-event group* $G_E = (E_1, ..., E_k)$ is defined similarly as a temporal pattern.

Intuitively, a pattern P is not frequent seasonal pattern if its *maxSeason* value is lower than *minSeason*. As a result, P can be eliminated safely.

4.3.2 Hierarchical lookup hash structure for STPM

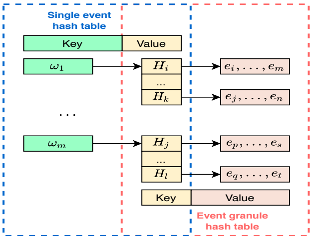


Fig. 4.3: The HLH_1 structure [38]

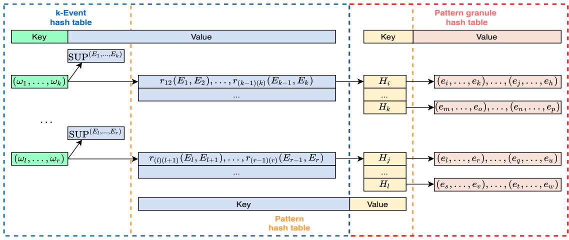


Fig. 4.4: The $HLH_k(k \geq 2)$ structure [38]

Hierarchical lookup hash structure HLH_1: For storing candidate seasonal single events, we utilize the HLH_1 structure, illustrated in Fig. 4.3. The HLH_1 contains 2 hash tables: the *single event hash table EH*, and the *event granule hash table GH*. Each hash table is a collection of <key, value> pairs. In EH, the key is $\omega \in \Sigma_X$ of the candidate E_i, while the value corresponds to the granules $< H_i, ..., H_k >$ in SUP^{E_i}. In GH, the key is taken from values in EH, and the value is instances of E_i.

Hierarchical lookup hash structure HLH_k: For storing candidate seasonal k-event groups and patterns, we use the HLH_k structure ($k \geq 2$), as illustrated in Fig. 4.4. The HLH_k comprises 3 hash tables: the *k-event hash table* EH_k, the *pattern hash table* PH_k, and the *pattern granule hash table* GH_k. In EH_k, *key* is the collection of symbols $(\omega_1..., \omega_k)$ of the candidate k-event group $(E_1, ..., E_k)$, while *value* consists of two components: (1) $SUP^{(E_1,...,E_k)}$, and (2) a collection of candidate seasonal k-event patterns of $(E_1, ..., E_k)$. In PH_k, *key* corresponds the candidate pattern P, and *value* contains the granules of P. In GH_k, *key* is the granules of P, and *value* is the collection of event instances from which P are formed.

The hierarchical lookup hash structures facilitates retrieval of candidate events and patterns quickly. Subsequently, we present the STPM algorithm, outlined in Alg. 5 .

4.3.3 Mining Seasonal Single Events

First, we mine frequent seasonal single events (Alg. 5 , lines 1-9). Specifically, we identify candidate single events, and then mine frequent seasonal events from the found candidates.

Algorithm 5: Frequent Seasonal Temporal Pattern Mining [38]

Input: Temporal sequence database $\mathcal{D}_{\text{SEQ}}$, the thresholds: *maxPeriod*, *minDensity*, *distInterval*, *minSeason*
Output: All frequent seasonal temporal patterns $\mathcal{P}$
// Step 2.1: Mine frequent seasonal single events
foreach *event* $E_i \in \mathcal{D}_{SEQ}$ **do**
  Find SUP^{E_i} and compute $maxSeason(E_i)$;
  if $maxSeason(E_i) \geq minSeason$ **then**
    Insert E_i into *Candidate1Event* ;
foreach *candidate* $E_i \in$ *Candidate1Event* **do**
  Find NearSUP^{E_i} that satisfies *maxPeriod* and *minDensity* ;
  Find PS^{E_i} that adheres *distInterval* ;
  if $|PS^{E_i}| \geq minSeason$ **then**
    Insert E_i into $\mathcal{P}$; $//E_i$ is a frequent seasonal event
// Step 2.2: Mine frequent seasonal k-event patterns, $k \geq 2$
FilteredF1 ⟵ Transitivity_Filtering(F_1);
kEventGroups ⟵ Cartesian(FilteredF1, F_{k-1});
CandidatekEvent ⟵ maxSeason_Filtering(kEventGroups);
foreach *kEvent in CandidatekEvent* **do**
  (k-1)-event_patterns ⟵ Retrieve_Relations(PH_{k-1});
  k-event_patterns ⟵ Iterative_Check((k-1)-event_patterns, E_k);
  foreach *P in k-event_patterns* **do**
    if $maxSeason(P) \geq minSeason$ **then**
      Insert P into *CandidatekPatterns*;
foreach *candidate* $P \in$ *CandidatekPatterns* **do**
  Find NearSUP^P satisfying *maxPeriod* and *minDensity*;
  Identify $\mathcal{PS}^P$ adhering to *distInterval* ;
  if $|\mathcal{PS}^P| \geq minSeason$ **then**
    Insert P into $\mathcal{P}$; $//P$ is a frequent seasonal pattern

To identify the candidate single events, we initially scan $\mathcal{D}_{\text{SEQ}}$ to determine the support set SUP^{E_i} for each event E_i. From SUP^{E_i}, we calculate the $maxSeason(E_i)$ and check whether E_i is a candidate single event.

We continue the following steps to mine frequent seasonal events. For each candidate event E_i, we iterate through the SUP^{E_i} and identify the near support sets NearSUP^{E_i} satisfying the constraints of *maxPeriod* and *minDensity*. By applying the *distInterval* constraint, we determine the set of seasons $\mathcal{PS}^{E_i}$. Next, we only select the frequent seasonal events that have $seasons(E_i) = |\mathcal{PS}^{E_i}| \geq minSeason$.

Example 4.3.2 (Candidate single events at HLH_1)
Let *maxPeriod* = 2, *minDensity* = 3, *distInterval* = [4, 10], and *minSeason* = 2. Fig. 4.5 shows HLH_1 in Table 4.2. There are 8 candidate seasonal single events at HLH_1. The event M:1 is not a frequent seasonal event (*season*(M:1) = 1

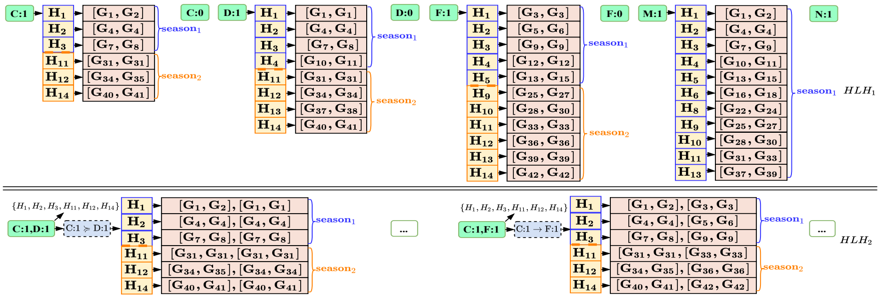


Fig. 4.5: An example of a hierarchical lookup hash tables [38]

$< minSeason$), but is kept in HLH_1 since it might contribute to the formation of frequent seasonal k-event patterns.

4.3.4 Mining Seasonal k-event Patterns

To address the issue of a large search space [38], we first identify candidate seasonal k-event groups and use these candidates to determine frequent seasonal k-event patterns.

Mining candidate seasonal k-event groups. Alg. 5 (lines 10-12) describes this step. First, to generate k-event groups, we utilize the set of candidate seasonal (k-1)-event groups F_{k-1} and the set of candidate seasonal single events F_1, and take the Cartesian product of F_{k-1} and F_1. Next, we find the support set $SUP^{(E_1,\ldots,E_k)}$ for each k-event group $(E_1, \ldots, E_k)$. We then calculate $maxSeason(E_1, \ldots, E_k)$, and check if $maxSeason(E_1, \ldots, E_k) \geq minSeason$ then $(E_1, \ldots, E_k)$ is considered as a candidate group and is stored in HLH_k.

Mining frequent seasonal k-event patterns. Alg. 5 (lines 13-23) describes this mining step. Let $F_{k-1} = (E_1, \ldots, E_{k-1})$ and $F_1 = (E_k)$ be a candidate (k-1)-event group and a candidate single event, respectively, and $F_k = F_{k-1} \cup F_1 = (E_1, \ldots, E_k)$ be a candidate k-event. We first access the EH_{k-1} table to take the set $\mathcal{P}_{k-1}$ containing the candidate (k-1)-event patterns of F_{k-1}. We verify that each $P_{k-1} = \{(r_{12}, E_1, E_2),\ldots,(r_{(k-2)(k-1)}, E_{k-2}, E_{k-1})\} \in \mathcal{P}_{k-1}$ can create a k-event pattern P_k with E_k as follows. First, we check if $(r_{(k-1)k}, E_{k-1}, E_k)$ is not exist, then the verification process terminates immediately. However, if it exists, we continue the verification in a similar manner with the triple $(r_{(k-2)k}, E_{k-2}, E_k)$, and proceed iteratively until we reach (r_{1k}, E_1, E_k). Next, we check if P_k is a candidate k-event pattern then it is stored in HLH_k. Finally, we find frequent seasonal k-event patterns from the discovered candidates.

Using transitivity property to optimize candidate k-event groups: We observe that using the candidate events in F_1 at HLH_1 to generate k-event groups can result in redundancy, as some events in F_1 combined with F_{k-1} may not generate any frequent seasonal k-event patterns.

Example 4.3.3 (The redundancy of k-event groups generation)
Let's consider the event F:0 in HLH_1 shown in Fig. 4.5 . In this case, F:0 can be used to combine with 2-event groups in HLH_2, such as (C:1, D:1), to form a 3-event group (C:1, D:1, F:0). However, F:0 does not exist in any candidate 2-event patterns in HLH_2. Thus, there do not have any candidate seasonal 3-event patterns that can be derived from (C:1, D:1, F:0).

To address this, we employ the *transitivity property* of temporal relations to minimize redundancy as follows.

Lemma 1 *Let $N_{k-1} = (E_1, ..., E_{k-1})$ be a candidate seasonal (k-1)-event group, and E_k be a candidate seasonal single event. If $\forall E_i \in N_{k-1}$, $\exists r \in \mathfrak{R}$ s.t. $r(E_i, E_k)$ is a candidate seasonal relation, then $N_k = N_{k-1} \cup E_k$ can form candidate seasonal k-event patterns.*

Based on Lemma 1, we only consider single events in HLH_1 that are present in HLH_{k-1} for creating k-event groups. To do that, we apply a filtering process to F_1 and obtain a new set called *FilteredF1*. Subsequently, we replace the Cartesian product $F_{k-1} \times F_1$ with $F_{k-1} \times$ *FilteredF1* to generate the k-event groups.

4.4 Approximate STPM

In this section, we use mutual information to measure the correlation between symbolic time series, then propose an approximate version of STPM that only mine frequent seasonal temporal patterns on the correlated time series. The definitions, theorems, and corollaries are reproduced from Paper C [38], and their proofs can be found in Paper C.

Consider two time series X and Y, and their corresponding symbolic series X_S, Y_S.

4.4.1 Correlated symbolic time series

This section introduces the concept of correlated symbolic time series used in the approximate STPM.

Definition 4.4.1 (Normalized mutual information)
The *normalized mutual information* (NMI) of X_S and Y_S, denoted as $\widetilde{I}(X_S;Y_S)$, quantifies the degree of shared information between X_S and Y_S in percentage:

$$\widetilde{I}(X_S;Y_S) = \frac{I(X_S;Y_S)}{H(X_S)} = 1 - \frac{H(X_S|Y_S)}{H(X_S)} \tag{4.2}$$

where $I(X_S;Y_S)$ is the mutual information of X_S and Y_S and $H(X_S)$ is the entropy of X_S.

$\widetilde{I}(X_S;Y_S)$ indicates the reduction (in percentage) of the uncertainty of X_S due to knowing Y_S. From Eq. 4.2 , if $I(X_S;Y_S) > 0$ then $(X_S;Y_S)$ has a certain mutual dependency. Moreover, it is important to note that NMI is not symmetric, i.e., $\widetilde{I}(X_S;Y_S) \neq \widetilde{I}(Y_S;X_S)$.

Definition 4.4.2 (Correlated symbolic time series)
Let μ where $0 < \mu \leq 1$ be the mutual information threshold. The series X_S and Y_S are *correlated* iff $\min\{\widetilde{I}(X_S;Y_S), \widetilde{I}(Y_S;X_S)\} \geq \mu$, and *uncorrelated* otherwise.

4.4.2 Lower bound of the maximum seasonal occurrence

The approximate STPM is built on the exact STPM algorithm that uses mutual information to identify the correlated time series and subsequently mine similarly as the exact STPM on these correlated time series. Besides, the exact STPM algorithm uses the maximum seasonal occurrence to prune infrequent candidate seasonal patterns. Thus, this section investigates the relationship between the mutual information of two symbolic series and the maximum seasonal occurrence of an event pair, as in Theorem 1, and then uses this relationship to prune the uncorrelated time series, reducing the search space of the mining process.

Theorem 1 *(Lower bound of the maxSeason)*
Let μ be the mutual information threshold. If the NMI $\widetilde{I}(X_S;Y_S) \geq \mu$, then the maximum seasonal occurrence of (X_1, Y_1) in $\mathcal{D}_{SEQ}$ has a lower bound:

$$maxSeason(X_1, Y_1) \geq \frac{\lambda_2 \cdot |\mathcal{D}_{SEQ}|}{minDensity} \cdot e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot ln2}{\lambda_2}\right)} \tag{4.3}$$

where: $\lambda_1 = \min\{p(X_i), \forall X_i \in X_S\}$ is the minimum probability of $X_i \in X_S$, and $\lambda_2 = p(Y_1)$ is the probability of $Y_1 \in Y_S$, and W is the Lambert function [13].

Setting the parameters: In order to calculate the lower bound of *maxSeason*(X_1, Y_1) in Equation (4.3), we need to compute several parameters: λ_1, λ_2, and μ. λ_1 and λ_2 can be computed easily from $\mathcal{D}_{\text{SYB}}$. To determine the value of μ, we derive the corollary from Theorem 1 as follows.

Corollary 1.1 *The maximum seasonal occurrence of an event pair* $(X_1, Y_1) \in (X_S, Y_S)$ *in* $\mathcal{D}_{SEQ}$ *is at least minSeason if* $\widetilde{I}(X_S; Y_S)$ *is at least* μ*, where:*

$$\mu \geq \begin{cases} 1 - \frac{\lambda_2}{e \cdot \ln 2 \cdot \log \frac{1}{\lambda_1}}, \textit{ if } 0 \leq \rho \leq \frac{1}{e} \\ 1 - \frac{\rho \cdot \lambda_2 \cdot \log \rho}{\ln 2 \cdot \log \lambda_1}, \quad \textit{otherwise} \end{cases}, \textit{where } \rho = \frac{minSeason \cdot minDensity}{\lambda_2 \cdot |\mathcal{D}_{SEQ}|} \tag{4.4}$$

Interpretation: Theorem 1 states that if the two series X_S and Y_S exhibit correlation, then the maximum seasonal occurrence of an event pair in (X_S,Y_S) has the lower bound in Equation (4.3). Moreover, we have the maximum seasonal occurrence of an event pair is always at least the maximum seasonal occurrence of the 2-event pattern formed by that event pair (proved in Def. 4.3.1). Thus, we can deduce that if the event pair (X_1, Y_1) has a maximum seasonal occurrence lower than the lower bound defined in Equation (4.3), any 2-event pattern P formed by (X_1, Y_1) will also have a maximum seasonal occurrence lower than the same lower bound. This is the basis for constructing the Approximate STPM algorithm.

4.4.3 Using the Bound to Approximate STPM

Algorithm 6: Approximate STPM using Mutual Information [38]

Input: A set of time series $\mathcal{X}$, the thresholds: $maxPeriod$, $minDensity$, $distInterval$, $minSeason$

Output: All frequent seasonal temporal patterns $\mathcal{P}$

```
1: foreach pair of series (X_S, Y_S) ∈ D_SYB do
2:     minNMI ← min{Ĩ(X_S; Y_S), Ĩ(Y_S; X_S)};
3:     Compute μ using Eq. (2.5);
4:     if minNMI ≥ μ then
5:         Insert X_S and Y_S into X_C;
6: Mine frequent seasonal single events from X_C;
7: if k ≥ 2 then
8:     Perform STPM using HLH_1 and HLH_{k−1};
```

Algorithm 6 describes the Approximate STPM. First, we compute NMI and μ for each pair (X_S, Y_S) (lines 2-3). We note that μ is calculated using Eq. 4.4. Then, we select only the correlated pairs to insert into $\mathcal{X}_C$. Next, we proceed to mine frequent seasonal single events exclusively from the series in $\mathcal{X}_C$ (line 6). For frequent seasonal k-event patterns ($k \geq 2$), we employ the exact STPM approach (lines 7-8).

4.5 Experimental Evaluation

We evaluate the exact STPM and approximate STPM algorithms using real-world datasets from three domains: renewable energy, smart city, and health. The evaluation includes both qualitative and quantitative analyses.

4.5.1 Experimental Design

Datasets: We use four real-world datasets: RE [60], SC [11], INF [12], and HFM [12]. The RE dataset is renewable energy from Spain. The SC dataset is traffic data from New York City. The INF and HMF datasets are the influenza and hand-foot-mouth data from Japan.
Baseline methods: We refer to our exact method as E-STPM and the approximate method as A-STPM. As this is the first study on frequent seasonal temporal pattern mining, there is currently no exact baseline for comparison against STPM. However, we adapt the PS-growth algorithm, originally designed for recurring itemset mining [49], to discover seasonal temporal patterns. The adapted version of PS-growth is referred to as APS-growth.

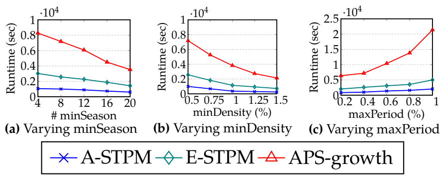
4.5.2 Experimental Results

In this summary, we present the key results, and additional results can be found in [38].
Qualitative Evaluation: Table 4.3 provides several interesting seasonal patterns discovered in the datasets. Patterns P1-P3 are found from the RE dataset, revealing the seasonal occurrence of both high renewable energy generation and electricity demand. For instance, P1 demonstrates that wind energy generation is high during the months of December to February, which coincides with increased wind availability. Similarly, P2 indicates high electricity demand during this period due to shorter daylight hours and lower temperatures, necessitating greater lighting and heating. These patterns suggest that wind power can effectively supplement the energy supply during high-demand winter periods, facilitating better supply response optimization. Patterns P4-P7, extracted from the INF and HFM datasets, capture the seasonality of specific diseases. For example, P4 reveals a substantial rise in influenza cases during January and February when the temperature is very low. On the other hand, hand-foot-mouth disease cases increase during May and June when temperatures are high (P6). Awareness of these patterns enables enhanced planning and prevention strategies for diseases. Lastly, patterns P8-P11, extracted from the SC dataset, illustrate the impact of weather conditions on traffic. Adverse weather conditions lead to congestion, lane blockages, and flow incidents, often observed in July and August.

Table 4.3: Summary of Interesting Seasonal Patterns [38]

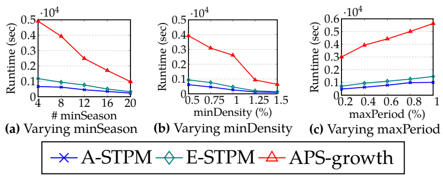
Patterns	minDensity (%)	maxPeriod (%)	# minSeason	Seasonal occurrence
(P1) Strong Wind ≽ High Wind Power Generation	0.5	0.4	12	December, January, February
(P2) Low Temperature ≽ High Energy Consumption	0.5	0.4	12	December, January, February
(P3) Very Few Clouds ≽ Very High Temperature ≬ High Solar Power Generation	0.75	0.6	8	July, August
(P4) High Humidity ≬ Very Low Temperature → Very High Influenza Cases	0.5	0.4	12	January, February
(P5) Strong Wind ≽ Heavy Rain ≽ High Influenza Cases	0.5	0.4	12	January, February
(P6) Low Humidity ≽ High Temperature ≽ Very High Hand-Foot-Mouth Disease Cases	1.0	0.6	12	May, June
(P7) Very High Temperature ≽ High Wind ≽ High Hand-Foot-Mouth Disease Cases	1.0	0.6	12	May, June
(P8) High Temperature ≽ Strong Wind → High Congestion	0.5	0.6	8	July, August
(P9) Strong Wind ≽ Unclear Visibility ≽ High Congestion	0.5	0.6	8	July, August
(P10) Heavy Rain ≽ Unclear Visibility ≽ High Lane-Blocked	0.4	0.8	8	July, August
(P11) Heavy Rain ≽ Strong Wind ≽ High Flow-Incident	0.4	0.8	8	July, August

Quantitative evaluation with baselines comparison on real-world datasets: E-STPM and A-STPM are compared with the baseline on real world datasets. Figs. 4.6 , 4.7 , 4.8 , and 4.9 show the results. Figs. 4.6 and 4.7 are the runtime comparisons between the algorithms. A-STPM is the fastest algorithm, while E-STPM has faster runtime than the baseline. The range and average speedups of A-STPM compared with other methods are: [1.5-4.7] and 2.6 (E-STPM), and [5.2-10.6] and 7.1 (APS-growth). Furthermore, E-STPM achieves a speedup over the baseline within the range of [3.5-7.2] and with a speedup average of 4.3.



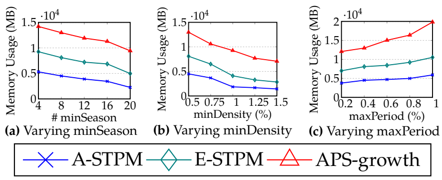
(a) Varying minSeason (b) Varying minDensity (c) Varying maxPeriod

Fig. 4.6: Runtime Comparison on RE (real-world) [38]



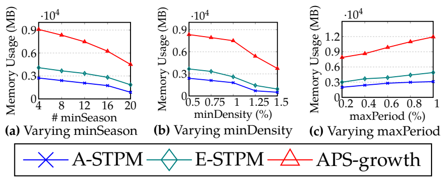
(a) Varying minSeason (b) Varying minDensity (c) Varying maxPeriod

Fig. 4.7: Runtime Comparison on INF (real-world) [38]



(a) Varying minSeason (b) Varying minDensity (c) Varying maxPeriod

Fig. 4.8: Memory Usage Comparison on RE (real-world) [38]



(a) Varying minSeason (b) Varying minDensity (c) Varying maxPeriod

Fig. 4.9: Memory Usage Comparison on INF (real-world) [38]

The memory usage comparison between the algorithms is shown in Figs. 4.8 and 4.9. Among the compared methods, A-STPM consumes the lowest memory usage, while E-STPM consumes less memory than the baseline. A-STPM consumes [1.4-2.7] (on average 1.8) times less memory than E-STPM, and [2.7-7.6] (on average 3.9) times less memory than APS-growth. E-STPM uses [1.5-4.1] (on average 2.3) times less memory than APS-growth.

Scalability evaluation on synthetic datasets: To assess the scalability of STPM, we compare performance between the algorithms on synthetic datasets. For each real-world dataset, we generated 10,000 synthetic time series. Figs. 4.10 and 4.11 shows the results of runtime comparisons when changing the number of time series. The range and average speedups of A-STPM in this scalability test are: [1.7-3.5] and 2.3 (E-STPM), [3.8-9.5] and 5.3 (APS-growth). The range and average speedups of E-STPM compared the baseline is [2.3-4.4] and 3.6. Moreover, we add a bar chart for A-STPM that has two components: the computation time (top red) for MI and μ , and the time of the mining process (bottom blue). However, they are added for only comparison and not actually used. We can see that the baseline fails at large configurations, e.g., when # Time Series ≥ 8000 on the synthetic INF (Fig. 4.11a).

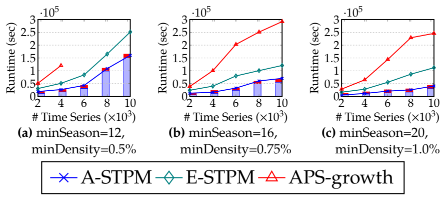


Fig. 4.10: Scalability: Varying #TimeSeries on RE (synthetic) [38]

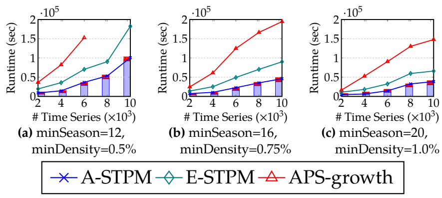


Fig. 4.11: Scalability: Varying #TimeSeries on INF (synthetic) [38]

Accuracy evaluation of A-STPM: We assess the accuracy of A-STPM by comparing the extracted patterns from A-STPM and E-STPM. Table 4.4 shows the results. We can see that A-STPM achieves high accuracy (≥ 81%) when both *minSeason* and *minDensity* are low, e.g., *minSeason* = 8 and *minDensity* = 0.5%. Additionally, A-STPM achieves very high accuracy (≥ 95%) when both *minSeason* and *minDensity* are high, such as when *minSeason* = 16 and *minDensity* = 0.75%.

Table 4.4: A-STPM Accuracy [38]

# minSeason	minDensity (%)					
	RE (real)			INF (real)		
	0.5	0.75	1	0.5	0.75	1
8	81	82	86	81	83	87
12	84	86	92	88	90	93
16	94	95	100	95	96	100
20	97	100	100	100	100	100

Chapter 5

Conclusion and Future Work

5.1 Contributions

The overall objective of this thesis is to propose efficient solutions to mine temporal patterns from time series and use information theory measures to further improve the mining process. This thesis fulfills the objective by proposing three different algorithms that can mine three types of temporal patterns: frequent temporal patterns, rare temporal patterns, and seasonal temporal patterns. In this thesis, we first addressed challenges of the frequent temporal pattern mining (FTPM) problem. Current literature for FTPM has limitations when working on big datasets, i.e., they fail on a large number of time series and temporal sequences. We solve this problem with the proposal of the exact FTPM algorithm in paper A [36] and part of paper B [37] that uses the efficient data structures and the pruning techniques, and the approximate FTPM algorithm that uses mutual information to eliminate the unpromising time series, helping FTPM scale well on big datasets. Second, we tackled challenges of the rare temporal pattern mining problem in paper B [37]. Current literature of rare temporal pattern mining only explores rare association rules, rare sequential patterns, and rare motifs that do not consider temporal relations between events. We proposed the first solution to mine rare temporal patterns (RTPM) efficiently. The exact RTPM algorithm that uses efficient data structures and different pruning techniques was proposed. Moreover, we presented the approximate version of RTPM using mutual information to perform the mining only on the promising time series, thereby reducing the search space of the mining. Also, in paper B, we proposed a generalized temporal pattern mining (GTPM) that combines both frequent and rare temporal patterns as a generalized approach for mining two types of temporal patterns. Finally, we deal with seasonal temporal pattern mining (STPM) in paper C [38]. Various techniques, such as motif discovery and periodic association rules, have treated seasonality as recurring occurrences without considering the seasonality characteristic of temporal patterns. We proposed the first-ever solution to mine seasonal temporal patterns. This solution consists of the exact STPM algorithm that uses efficient data structures and the novel concept of candidate seasonal temporal patterns for pruning infrequent seasonal temporal patterns. Moreover, we introduced the approximate version of STPM using mutual information to prune redundant time series, accelerating the mining process while maintaining highly accurate results.

The summarized results of this thesis are presented in three main papers A [36], B [37], and C [38] that have the following key contributions:

- Paper A [36] proposes a comprehensive process for frequent temporal pattern mining from time series (FTPMfTS). This process takes a collection of time series as input and produces a complete set of frequent temporal patterns as output. In the process, a splitting strategy is used to transform time series into sequences of temporal events that ensure the preservation of the patterns. FTPMfTS comprises an efficient Hierarchical Temporal Pattern Graph Mining (HTPGM) that uses efficient data structures, i.e., the Hierarchical Pattern Graph, and pruning techniques, i.e., the Apriori principle and the transitivity property of temporal relations, to enable faster mining. Furthermore, we present an approximate version of HTPGM that utilizes mutual information to remove unpromising time series, making it work well for large datasets.

- Paper B [37] extends paper A [36] with three main contributions. The first contribution is that we introduce an enhanced frequent temporal pattern mining (FTPM) algorithm that improves upon the HTPGM algorithm [36]. The key improvement involves the use of Hierarchical Hash Tables instead of the Hierarchical Pattern Graph, enabling faster retrieval of events and patterns. Additionally, we derive the lower bound of support and combine it with the lower bound of confidence to speed up the mining process in the approximate FTPM version. The second contribution is the proposal of an efficient Rare Temporal Pattern Mining (RTPM) algorithm designed specifically for searching rare temporal patterns. RTPM also employs efficient data structures and pruning techniques to optimize the mining process. In addition to the exact RTPM, we propose an approximate version that leverages mutual information to prune unpromising time series, further enhancing the speed of the mining process. The third contribution is that we presented the generalized temporal pattern mining (GTPM) that can mine both frequent and rare temporal patterns.

- Paper C [38] proposes the first solution to mine seasonal temporal patterns from time series. We first introduce several measures aimed at capturing the seasonality characteristics of temporal patterns, including the maximum period, minimum density, distance interval, and minimum seasonal occurrence. Then, an efficient algorithm called Seasonal Temporal Pattern Mining (STPM) is proposed that has several novelties. The first novelty is that a new measure, called the maximum season, is presented that upholds the anti-monotonicity property. This measure is then utilized to define the concept of a candidate seasonal temporal pattern, serving the purpose of eliminating infrequent seasonal temporal patterns. The second novelty pertains to the use of hierarchical hash tables data structures to ensure efficient retrieval of candidate temporal events and temporal patterns and the use of two pruning techniques, Apriori-like pruning and transitivity pruning, to speed up the mining process. In order to improve the scalability of STPM on large datasets, we introduce an approximate version of STPM that leverages mutual information to prune unpromising time series and reduce the search space.

5.2 Future Work

This thesis opens up multiple possibilities for future research directions. In the thesis, we use mutual information to prune unpromising time series in the approximate mining algorithms of three types of patterns. We have derived the relationships between mutual information of two symbolic series and the support, the confidence, and the maximum seasonal occurrence of an event pair in three papers, A [36], B [37], and C [38]. Based on these relationships, the unpromising time series are pruned. In future work, we can use mutual information to remove unpromising temporal events at the event level showing potential for improving the performance of approximate algorithms. To do this, we need to find the relationships between temporal events and the support, the confidence, and the maximum seasonal occurrence measures.

Additionally, the proposal of distributed solutions capable of mining various types of temporal patterns is a fascinating direction to pursue. The efficient distributed temporal pattern mining algorithm, called Distributed Hierarchical Pattern Graph Temporal Pattern Mining (DHPG-TPM), was proposed in [27]. DHPG-TPM outperforms the baselines and scales well to large datasets. DHPG-TPM uses the distributed bitmap and the distributed Hierarchical Pattern Graph to enable fast computations of support and confidence. However, we can improve DHPG-TPM by using the distributed Hierarchical Hash Table data structure that could accelerate further DHPG-TPM's performance. Furthermore, research on distributed algorithms for RTPM and STPM should be considered.

Another promising direction involves the discovery of high-utility temporal patterns within time series, which could have the potential for practical applications. A temporal pattern has high utility, implying that it is very important for users. High-utility temporal pattern mining is to find all temporal patterns whose utility is at least the minimum utility threshold. High-utility temporal pattern mining is very challenging due to two reasons. The first reason is that the complex temporal relations between events create an exponential search space. The second reason is that high-utility temporal patterns do not hold the anti-monotonicity property, i.e., a superset of a low-utility pattern may be a high-utility pattern. Thus, efficient solutions to mine high-utility temporal patterns will be promising research.

The exploration of spatial-temporal co-occurring patterns can offer significant insights with data that encompass both spatial and temporal components. The spatial-temporal co-occurring patterns represent types of events that occur in both space and time together. To mine them, we need to consider both spatial presentations and temporal relations. This is also an interesting future research problem.

Clustering and classification of temporal events/patterns can also be considered for future research. Since time information is added to each event/pattern, the selection of suitable distance functions becomes crucial in determining the outcomes of clustering and classification tasks. Moreover, pruning techniques also help further speed up the clustering and classification. The proposal of efficient methods to address these problems holds great promise for many real-world applications.

Bibliography

References

[1] E. Alipourchavary, S. M. Erfani, and C. Leckie, "Mining rare recurring events in network traffic using second order contrast patterns," in *International Joint Conference on Neural Networks*. IEEE, 2021.

[2] J. F. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, 1983.

[3] K. Amphawan, P. Lenca, and A. Surarerks, "Mining top-k periodic-frequent pattern from transactional databases without support threshold," in *International conference on advances in information technology*. Springer, 2009.

[4] J. Assfalg, T. Bernecker, H.-P. Kriegel, P. Kröger, and M. Renz, "Periodic pattern analysis in time series databases," in *International Conference on Database Systems for Advanced Applications*. Springer, 2009.

[5] N. Begum and E. Keogh, "Rare time series motif discovery from unbounded streams," *Proceedings of the VLDB Endowment*, vol. 8, no. 2, 2014.

[6] S. Biswas and K. C. Mondal, "Dynamic fp tree based rare pattern mining using multiple item supports constraints," in *Computational Intelligence, Communications, and Business Analytics (CICBA)*. Springer, 2019.

[7] A. Borah and B. Nath, "Rare association rule mining from incremental databases," *Pattern Analysis and Applications*, vol. 23, 2020.

[8] S. Bouasker, W. Inoubli, S. B. Yahia, and G. Diallo, "Pregnancy associated breast cancer gene expressions: new insights on their regulation based on rare correlated patterns," *Transactions on Computational Biology and Bioinformatics*, vol. 18, no. 3, 2020.

[9] S. Cai, J. Chen, H. Chen, C. Zhang, Q. Li, R. N. A. Sosu, and S. Yin, "An efficient anomaly detection method for uncertain data based on minimal rare patterns with the consideration of anti-monotonic constraints," *Information Sciences*, vol. 580, 2021.

[10] Y. Chen, W. Peng, and S. Lee, "Mining temporal patterns in time interval-based data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, 2015.

[11] N. Y. City. (2019) Nyc opendata. [Online]. Available: https://opendata.cityofnewyork.us/

[12] K. city infectious disease surveillance system. (2021) Kidss. [Online]. Available: https://kidss.city.kawasaki.jp/

[13] R. M. Corless, G. H. Gonnet, D. E. Hare, D. J. Jeffrey, and D. E. Knuth, "On the lambert w function," *Advances in Computational mathematics*, vol. 5, 1996.

[14] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[15] Y. Cui, W. Gan, H. Lin, and W. Zheng, "Fri-miner: fuzzy rare itemset mining," *Applied Intelligence*, 2022.

[16] P. S. Data. (2016) Pecan street dataport. [Online]. Available: https://www.pecanstreet.org/dataport/

[17] Energinet. (2021) Energi data portal. [Online]. Available: https://www.energidataservice.dk/tso-electricity/co2emis/

[18] P. Fournier-Viger, Y. Wang, P. Yang, J. C.-W. Lin, U. Yun, and R. U. Kiran, "Tspin: Mining top-k stable periodic patterns," *Applied Intelligence*, vol. 52, no. 6, 2022.

[19] P. Fournier-Viger, P. Yang, Z. Li, J. C.-W. Lin, and R. U. Kiran, "Discovering rare correlated periodic patterns in multiple sequences," *Data & Knowledge Engineering*, vol. 126, 2020.

[20] Y. Gao and J. Lin, "Efficient discovery of time series motifs with large length range in million scale time series," in *IEEE International Conference on Data Mining*. IEEE, 2017.

[21] M. Gribaudo, T. T. N. Ho, B. Pernici, and G. Serazzi, "Analysis of the influence of application deployment on energy consumption," in *International Workshop on Energy Efficient Data Centers*. Springer, 2014.

[22] J. Han, G. Dong, and Y. Yin, "Efficient mining of partial periodic patterns in time series database," in *IEEE International Conference on Data Engineering*. IEEE, 1999.

[23] J. Han, W. Gong, and Y. Yin, "Mining segment-wise periodic patterns in time-related databases," in *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, vol. 98, 1998.

[24] W. Healy, F. Omar, L. Ng, T. Ullah, W. Payne, B. Dougherty, and A. H. Fanney. (2018) Net zero energy residential test facility instrumented data. [Online]. Available: https://pages.nist.gov/netzero/index.html/

[25] N. Ho, V. L. Ho, T. B. Pedersen, M. Vu, and C. A. Biscio, "A unified approach for multi-scale synchronous correlation search in big time series–full version," *arXiv preprint arXiv:2204.09131*, 2022.

[26] N. Ho, T. B. Pedersen, M. Vu, C. A. Biscio *et al.*, "Efficient bottom-up discovery of multi-scale time series correlations using mutual information," in *IEEE International Conference on Data Engineering*. IEEE, 2019.

[27] N. Ho, T. B. Pedersen, M. Vu *et al.*, "Efficient and distributed temporal pattern mining," in *IEEE International Conference on Big Data*. IEEE, 2021.

[28] N. Ho, H. Vo, and M. Vu, "An adaptive information-theoretic approach for identifying temporal correlations in big data sets," in *IEEE International Conference on Big Data*. IEEE, 2016.

[29] N. Ho, H. Vo, M. Vu, and T. B. Pedersen, "Amic: An adaptive information theoretic method to identify multi-scale temporal correlations in big time series data," *IEEE Transactions on Big Data*, vol. 7, no. 1, 2019.

[30] N. Ho, H. Vo, M. Vu, and T. B. Pedersen, "Amic: An adaptive information theoretic method to identify multi-scale temporal correlations in big time series data – accepted version," *arXiv preprint arXiv:1906.09995*, 2019.

[31] N. T. T. Ho, T. B. Pedersen, L. Van Ho, and M. Vu, "Efficient search for multi-scale time delay correlations in big time series," in *International Conference on Extending Database Technology*. OpenProceedings. org, 2020.

[32] N. Ho, M. Gribaudo, and B. Pernici, "Improving energy efficiency for transactional workloads in cloud environments," in *Proceedings of the Eighth International Conference on Future Energy Systems*, 2017.

[33] T. T. N. Ho, M. Gribaudo, and B. Pernici, "Characterizing energy per job in cloud applications," *Electronics*, vol. 5, no. 4, 2016.

[34] T. T. N. Ho and B. Pernici, "A data-value-driven adaptation framework for energy efficiency for data intensive applications in clouds," in *IEEE conference on technologies for sustainability*. IEEE, 2015.

[35] V. L. Ho, N. Ho, and T. B. Pedersen, "Efficient temporal pattern mining in big time series using mutual information," *arXiv preprint arXiv:2010.03653*, 2020. [Online]. Available: https://arxiv.org/abs/2010.03653

[36] V. L. Ho, N. Ho, and T. B. Pedersen, "Efficient temporal pattern mining in big time series using mutual information," vol. 15, no. 3. VLDB Endowment, 2022.

[37] V. L. Ho, N. Ho, and T. B. Pedersen, "Efficient generalized temporal pattern mining in big time series using mutual information," 2023. [Online]. Available: https://arxiv.org/abs/2010.03653

[38] V. L. Ho, N. Ho, and T. B. Pedersen, "Mining seasonal temporal patterns in time series," in *IEEE International Conference on Data Engineering*. IEEE, 2023.

[39] J.-W. Huang, C.-Y. Tseng, J.-C. Ou, and M.-S. Chen, "A general model for sequential pattern mining with a progressive database," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 9, 2008.

[40] M. Iqbal, C. P. Wulandari, W. Yunanto, and G. I. P. Sari, "Mining non-zero-rare sequential patterns on activity recognition," *Jurnal Matematika MANTIK*, vol. 5, no. 1, 2019.

[41] M. F. Javed, W. Nawaz, and K. U. Khan, "Hova-fppm: flexible periodic pattern mining in time series databases using hashed occurrence vectors and apriori approach," *Scientific Programming*, vol. 2021, 2021.

[42] Y. Ji and Y. Ohsawa, "Mining frequent and rare itemsets with weighted supports using additive neural itemset embedding," in *International Joint Conference on Neural Networks*. IEEE, 2021.

[43] L. Kegel, C. Hartmann, M. Thiele, and W. Lehner, "Season-and trend-aware symbolic approximation for accurate and efficient time series matching," *Datenbank-Spektrum*, vol. 21, no. 3, 2021.

[44] J. Kelly and W. Knottenbelt, "The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes," *Scientific Data*, 2015.

[45] R. U. Kiran, M. Kitsuregawa, and P. K. Reddy, "Efficient discovery of periodic-frequent patterns in very large databases," *Journal of Systems and Software*, vol. 112, 2016.

[46] R. U. Kiran, C. Saideep, K. Zettsu, M. Toyoda, M. Kitsuregawa, and P. K. Reddy, "Discovering partial periodic spatial patterns in spatiotemporal databases," in *IEEE International Conference on Big Data*. IEEE, 2019.

[47] R. U. Kiran, H. Shang, M. Toyoda, and M. Kitsuregawa, "Discovering recurring patterns in time series," in *International Conference on Extending Database Technology*, 2015.

[48] R. U. Kiran, Y. Watanobe, B. Chaudhury, K. Zettsu, M. Toyoda, and M. Kitsuregawa, "Discovering maximal periodic-frequent patterns in very large temporal databases," in *International Conference on Data Science and Advanced Analytics*. IEEE, 2020.

[49] R. U. Kiran, A. Anirudh, C. Saideep, M. Toyoda, P. K. Reddy, and M. Kitsuregawa, "Finding periodic-frequent patterns in temporal databases using periodic summaries," *Data Science and Pattern Recognition*, vol. 3, no. 2, 2019.

[50] H. T. Lam, F. Mörchen, D. Fradkin, and T. Calders, "Mining compressing sequential patterns," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 7, no. 1, 2014.

[51] Z. Lee, T. Lindgren, and P. Papapetrou, "Z-miner: an efficient method for mining frequent arrangements of event intervals," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[52] Y. Li and S. Cai, "Detecting outliers in data streams based on minimum rare pattern mining and pattern matching," *Information Technology and Control*, vol. 51, no. 2, 2022.

[53] H. Liu, F. Han, H. Zhou, X. Yan, and K. S. Kosik, "Fast motif discovery in short sequences," in *IEEE International Conference on Data Engineering*. IEEE, 2016.

[54] Y. Mohammad and T. Nishida, "Approximately recurring motif discovery using shift density estimation," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2013.

[55] C. Neidle, A. Opoku, G. Dimitriadis, and D. Metaxas, "New shared & interconnected asl resources: Signstream® 3 software; dai 2 for web access to linguistically annotated video corpora; and a sign bank," in *Workshop on the Representation and Processing of Sign Languages: Involving the Language Community, Miyazaki, Language Resources and Evaluation Conference*, 2018.

[56] W. Ouyang, "Mining rare sequential patterns in large transaction databases," in *International Conference on Computer Science and Electronic Technology*. Atlantis Press, 2016.

[57] P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos, "Mining frequent arrangements of temporal intervals," *Knowledge and Information Systems*, vol. 21, 2009.

[58] D. Patel, W. Hsu, and M. L. Lee, "Mining relationships among interval-based events for classification," in *Proceedings of the ACM SIGMOD international conference on Management of data*, 2008.

[59] S. Piri, D. Delen, T. Liu, and W. Paiva, "Development of a new metric to identify rare patterns in association analysis: The case of analyzing diabetes complications," *Expert Systems with Applications*, vol. 94, 2018.

[60] E.-E. T. Platform. (2019) Entso-e. [Online]. Available: https://transparency.entsoe.eu/dashboard/show

[61] A. Rahman, "Rare sequential pattern mining of critical infrastructure control logs for anomaly detection," Ph.D. dissertation, Queensland University of Technology, 2019.

[62] A. Rahman, Y. Xu, K. Radke, and E. Foo, "Finding anomalies in scada logs using rare sequential pattern mining," in *Network and System Security*. Springer, 2016.

[63] A. Samet, T. Guyet, and B. Negrevergne, "Mining rare sequential patterns with asp," in *International Conference on Inductive Logic Programming*, 2017.

[64] A. K. Sharma and D. Patel, "Stipa: A memory efficient technique for interval pattern discovery," in *IEEE International Conference on Big Data*. IEEE, 2018.

[65] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "Discovering periodic-frequent patterns in transactional databases," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2009.

[66] R. Uday Kiran and P. Krishna Reddy, "Towards efficient mining of periodic-frequent patterns in transactional databases," in *International Conference on Database and Expert Systems Applications*. Springer, 2010.

[67] O. Weather. (2021) Open weather. [Online]. Available: https://openweathermap.org/

[68] S.-Y. Wu and Y.-L. Chen, "Mining nonambiguous temporal patterns for interval-based events," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, 2007.

[69] M. Zhang, P. Wang, and W. Wang, "Efficient consensus motif discovery of all lengths in multiple time series," in *International Conference on Database Systems for Advanced Applications*. Springer, 2022.

[70] J. Zhu, K. Wang, Y. Wu, Z. Hu, and H. Wang, "Mining user-aware rare sequential topic patterns in document streams," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 7, 2016.

Part II

Papers

Paper A

Efficient Temporal Pattern Mining in Big Time Series Using Mutual Information

Van Long Ho, Nguyen Ho, and Torben Bach Pedersen

The paper has been published in the
Proceedings of the VLDB Endowment (PVLDB), Volume 15, Issue 3, Pages 673–685, November, 2021, ISSN 2150-8097, DOI:10.14778/3494124.3494147.

Abstract

Very large time series are increasingly available from an ever wider range of IoT-enabled sensors deployed in different environments. Significant insights can be gained by mining temporal patterns from these time series. Unlike traditional pattern mining, temporal pattern mining (TPM) adds event time intervals into extracted patterns, making them more expressive at the expense of increased time and space complexities. Existing TPM methods either cannot scale to large datasets, or work only on pre-processed temporal events rather than on time series. This paper presents our Frequent Temporal Pattern Mining from Time Series (FTPMfTS) approach providing: (1) The end-to-end FTPMfTS process taking time series as input and producing frequent temporal patterns as output. (2) The efficient Hierarchical Temporal Pattern Graph Mining (HTPGM) algorithm that uses efficient data structures for fast support and confidence computation, and employs effective pruning techniques for significantly faster mining. (3) An approximate version of HTPGM that uses mutual information, a measure of data correlation, to prune unpromising time series from the search space. (4) An extensive experimental evaluation showing that HTPGM outperforms the baselines in runtime and memory consumption, and can scale to big datasets. The approximate HTPGM is up to two orders of magnitude faster and less memory consuming than the baselines, while retaining high accuracy.

A.1 Introduction

IoT-enabled sensors have enabled the collection of many big time series, e.g., from smart-meters, -plugs, and -appliances in households, weather stations, and GPS-enabled mobile devices. Extracting patterns from these time series can offer new domain insights for evidence-based decision making and optimization. As an example, consider Fig. A.1 that shows the electricity usage of

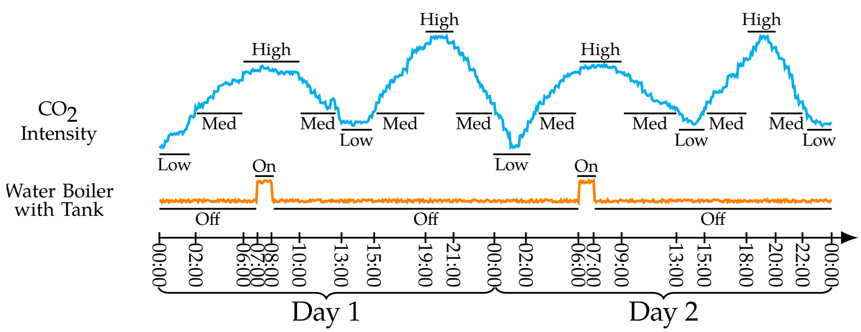


Fig. A.1: CO2 intensity and water boiler electricity usage a water boiler with a hot water tank collected by a 20 euro wifi-enabled smart-plug, and accurate CO2 intensity (g/kWh) forecasts of local electricity, e.g., as supplied by the Danish Transmission System Operator [12]. From Fig. A.1, we can identify several useful patterns. First, the water boiler switches *On* once a day, for one hour between 6 and 8AM. This indicates that the resident takes only one hot shower per day which starts between 5.30 and 6.30AM. Second, all water boiler *On* events are contained in CO2 *High* events, i.e., the periods when CO2 intensity is high. Third, between two consecutive *On* events of the boiler, there is a CO2 *Low* event lasting for one or more hours which occurs at most 4 hours before the hot shower (so water heated during that event will still be hot at 6AM). Pattern mining can be used to extract the relations between CO2 intensity and water boiler events. However, traditional sequential patterns only capture the sequential occurrence of events, e.g., that one boiler *On* event follows after another, but not that there is at least 23 hours between them; or that there is a CO2 *Low* event between the two boiler *On* events, but not when or for how long it lasts. In contrast, *temporal pattern mining* (TPM) adds temporal information into patterns, providing details on when certain relations between events happen, and for how long. For example, TPM expresses the above relations as: ([7:00 - 8:00, Day X] BoilerOn $\rightarrow$ [6:00 - 7:00, Day X+1] BoilerOn) (meaning BoilerOn is followed by BoilerOn), ([6:00 - 10:00, Day X] HighCO2 $\succcurlyeq$ [7:00 - 8:00, Day X] BoilerOn) (meaning HighCO2 contains BoilerOn), and ([7:00 - 8:00, Day X] BoilerOn $\rightarrow$ [0:00 - 2:00, Day X+1] LowCO2 $\rightarrow$ [6:00 - 7:00, Day X+1] BoilerOn). As the resident is very keen on reducing her CO2 footprint, we can rely on the above temporal patterns to automatically (using the smart-plug) delay turning on the boiler until the CO2 intensity is low again, saving CO2 without any loss of comfort for the resident.

Another example is in the smart city domain in which temporal patterns extracted from vehicle GPS data [41] can reveal spatio-temporal correlations between traffic jams. For example, if the pattern ([07:30, 08:00] SlowSpeedTunnel $\rightarrow$ [08:00, 08:30] SlowSpeedMainBoulevard) is found with high frequency and high confidence on weekdays, it can be used to advise drivers to take another route for their morning commute.

Although temporal patterns are useful, mining them is much more expensive than sequential patterns. Not only does the temporal information add extra computation to the mining process, the complex relations between events also add an additional exponential factor $O(3^{h^2})$ to the complexity $O(m^h)$ of the search space (m is the number of events and h is the length of temporal patterns), yielding an overall complexity of $O(m^h 3^{h^2})$ (see Lemma 1 in Section A.4.4). Existing TPM methods [8, 35, 36] do not scale on big datasets, i.e., many time series and many sequences, and/or do not work directly on time series but rather on pre-processed temporal events.

Contributions. In this paper, we present our comprehensive Frequent Temporal Pattern Mining from Time Series (FTPMfTS) approach which overcomes the above limitations. Our key contributions are: (1) We present the first end-to-end FTPMfTS process that receives time series as input, and produces frequent temporal patterns as output. Within this process, a splitting strategy is proposed to convert time series into event sequences while ensuring the preservation of temporal patterns. (2) We propose the efficient Hierarchical Temporal Pattern Graph Mining (HTPGM) algorithm that employs: a) efficient data structures, Hierarchical Pattern Graph and *bitmap*, to enable fast support and confidence computation; and b) pruning techniques based on the Apriori principle and the transitivity property of temporal relations to enable faster mining. (3) Based on the concept of mutual information which measures the correlation among time series, we propose a novel approximate version of HTPGM that prunes unpromising time series to significantly reduce the search space and can scale on big datasets, i.e., many time series and many sequences. (4) We perform extensive experiments on synthetic and real-world datasets which show that HTPGM outperforms the baselines in both runtime and memory usage. The approximate HTPGM is up to two orders of magnitude faster and less memory consumption than the baselines while retaining high accuracy compared to the exact HTPGM.

A.2 Related work

Temporal pattern mining: Compared to sequential pattern mining, TPM is rather a new research area. One of the first papers in this area is [20] from Kam et al. that uses a hierarchical representation to manage temporal relations, and based on that mines temporal patterns. However, the approach in [20] suffers from *ambiguity* when presenting temporal relations. In [39], Wu et al. develop TPrefix to mine temporal patterns from non-ambiguous temporal relations. However, TPrefix has several inherent limitations: it scans the database repeatedly, and the algorithm does not employ any pruning strategies to reduce the search space. In [32], Moskovitch et al. design a TPM algorithm using the transitivity property of temporal relations. They use this property to generate candidates by inferring new relations between events. In comparison, our HTPGM uses the transitivity property for effective pruning. In [3], Iyad et al. propose a TPM framework to detect events in time series. However, their focus is to find irregularities in the data. In [38], Wang et al. propose a temporal pattern mining algorithm HUTPMiner to mine high-utility patterns. Different from our HTPGM which uses *support* and *confidence* to measure the frequency of patterns, HUTPMiner uses *utility* to measure the importance or profit of an event/ pattern, thereby addresses an orthogonal problem. In [37], Amit et al. propose STIPA which uses a Hoeppner matrix representation to compress temporal patterns for memory savings. However, STIPA does not use any pruning/ optimization strategies and thus, despite the efficient use of memory, it cannot scale to large datasets, unlike our HTPGM. Other work [4], [7] proposes TPM algorithms to classify health record data. However, these methods are very domain-specific, thus cannot generalize to other domains.

The state-of-the-art TPM methods that currently achieve the best performance are our baselines: H-DFS [35], TPMiner [8], IEMiner [36], and Z-Miner [28]. H-DFS is a hybrid algorithm that uses breadth-first and depth-first search strategies to mine frequent arrangements of temporal intervals. H-DFS uses a data structure called ID-List to transform event sequences into vertical representations, and temporal patterns are generated by merging the ID-Lists of different events. This means that H-DFS does not scale well when the number of time series increases. In [36], Patel et al. design a hierarchical lossless representation to model event relations, and propose IEMiner that uses Apriori-based optimizations to efficiently mine patterns from this new representation. In [8], Chen et al. propose TPMiner that uses endpoint and endtime representations to simplify the complex relations among events. Similar to [35], IEMiner and TPMiner do not scale to datasets with many time series. Z-Miner [28], proposed by Lee et al., is the most recent work addressing TPM. Z-Miner improves the mining efficiency over existing methods by employing two data structures: a hierarchical hash-based structure called Z-Table for time-efficient candidate generation and support count, and Z-Arrangement, a structure to efficiently store event intervals in temporal patterns for efficient memory consumption. Although using efficient data structures, Z-Miner neither employs the transitivity property of temporal relations nor mutual information for pruning. Thus, Z-Miner is less efficient than our exact and approximate HTPGM in both runtimes and memory usage, and does not scale to large datasets with many sequences and many time series (see Section A.6). Our HTPGM algorithm improves on these methods by: (1) using efficient data structures and applying pruning techniques based on the Apriori principle and the transitivity property of temporal relations to enable fast mining, (2) the approximate HTPGM can handle datasets with many time series and sequences, and (3), providing an end-to-end FTPMfTS process to mine temporal patterns directly from time series, a feature that is not supported by the baselines.

Using correlations in TPM: Different correlation measures such as expected support [1], all-confidence [27], and mutual information (MI) [6, 11, 15–18, 21–25, 40] have been used to optimize the pattern mining process. However, these only support sequential patterns. To the best of our knowledge, our proposed approximate HTPGM is the first that uses MI to optimize TPM.

Table A.1: A Symbolic Database $\mathcal{D}_{\mathrm{SYB}}$

Time	10:00	10:05	10:10	10:15	10:20	10:25	10:30	10:35	10:40	10:45	10:50	10:55	11:00	11:05	11:10	11:15	11:20	11:25
S	On	On	On	On	Off	Off	Off	On	On	Off	Off	Off	Off	Off	Off	On	On	On
T	Off	On	On	On	Off	Off	Off	On	On	Off	Off	On	On	Off	Off	On	On	On
M	Off	Off	Off	Off	On	On	On	Off	Off	On	On	On	Off	On	On	Off	Off	Off
W	Off	Off	Off	Off	On	On	On	Off	Off	On	On	Off	On	On	On	Off	Off	Off
D	Off	Off	Off	Off	Off	Off	Off	Off	Off	On	On	Off	Off	Off	Off	Off	On	On
I	Off	Off	Off	Off	Off	Off	Off	On	On	Off	Off	Off	Off	Off	Off	Off	Off	Off

Time	11:30	11:35	11:40	11:45	11:50	11:55	12:00	12:05	12:10	12:15	12:20	12:25	12:30	12:35	12:40	12:45	12:50	12:55
S	Off	Off	Off	Off	On	On	On	Off	Off	On	On	Off	Off	On	On	On	Off	Off
T	Off	Off	Off	Off	On	On	On	Off	Off	On	On	Off	Off	Off	On	On	On	Off
M	On	On	Off	On	On	Off	Off	On	On	Off	Off	On	On	On	Off	Off	On	On
W	On	On	Off	On	On	Off	Off	On	On	Off	Off	On	On	On	Off	Off	On	On
D	Off	Off	Off	Off	Off	Off	Off	Off	Off	On	On	Off	Off	Off	On	On	Off	Off
I	On	On	Off	Off	Off	Off	Off	Off	Off	On	On	Off	Off	Off	Off	Off	On	On

A.3 Preliminaries

In this section, we introduce the notations and the main concepts that will be used throughout the paper.

A.3.1 Temporal Event of Time Series

Definition 3.1 (Time series) A *time series* $X = x_1, x_2, ..., x_n$ is a sequence of data values that measure the same phenomenon during an observation time period, and are chronologically ordered.

Definition 3.2 (Symbolic time series) A *symbolic time series* X_S of a time series X encodes the raw values of X into a sequence of symbols. The finite set of permitted symbols used to encode X is called the *symbol alphabet* of X, denoted as Σ_X.

The symbolic time series X_S is obtained using a mapping function $f\colon X \rightarrow \Sigma_X$ that maps each value $x_i \in X$ to a symbol $\omega \in \Sigma_X$. For example, let X = 1.61, 1.21, 0.41, 0.0 be a time series representing the energy usage of an electrical device. Using the symbol alphabet Σ_X = {On, Off}, where On represents that the device is on and operating (e.g., $x_i \geq 0.5$), and Off that the device is off ($x_i < 0.5$), the symbolic representation of X is: X_S = On, On, Off, Off. The mapping function f can be defined using existing time series representation techniques such as SAX [29] or MVQ [30].

Definition 3.3 (Symbolic database) Given a set of time series $\mathcal{X} = \{X_1, ..., X_n\}$, the set of symbolic representations of the time series in $\mathcal{X}$ forms a *symbolic database* $\mathcal{D}_{\mathrm{SYB}}$.

An example of the symbolic database $\mathcal{D}_{\mathrm{SYB}}$ is shown in Table A.1. There are 6 time series representing the energy usage of 6 electrical appliances: {Stove, Toaster, Microwave, Clothes Washer, Dryer, Iron}. For brevity, we name the appliances respectively as {S, T, M, W, D, I}. All appliances have the same alphabet Σ = {On, Off}.

Definition 3.4 (Temporal event in a symbolic time series) A *temporal event* E in a symbolic time series X_S is a tuple $E = (\omega, T)$ where $\omega \in \Sigma_X$ is a symbol, and $T = \{[t_{s_i}, t_{e_i}]\}$ is the set of time intervals during which X_S is associated with the symbol ω.

Given a time series X, a temporal event is created by first converting X into symbolic time series X_S, and then combining identical consecutive symbols in X_S into one single time interval. For example, consider the symbolic representation of S in Table A.1. By combining its consecutive On symbols, we form the temporal event *"Stove is On"* as: (SOn, {[10:00, 10:15], [10:35, 10:40], [11:15, 11:25], [11:50, 12:00], [12:15, 12:20], [12:35, 12:45]}).

Definition 3.5 (Instance of a temporal event) Let $E = (\omega, T)$ be a temporal event, and $[t_{s_i}, t_{e_i}] \in T$ be a time interval. The tuple $e = (\omega, [t_{s_i}, t_{e_i}])$ is called an *instance* of the event E, representing a single occurrence of E during $[t_{s_i}, t_{e_i}]$. We use the notation $E_{\triangleright e}$ to denote that event E has an instance e.

A.3.2 Relations between Temporal Events

We adopt the popular Allen's relations model [2] and define three basic temporal relations between events. Furthermore, to avoid the exact time mapping problem in Allen's relations, we adopt the *buffer* idea from [35], adding a tolerance *buffer* ϵ to the relation's endpoints. However, we change the way ϵ is used in [35] to ensure the relations are *mutually exclusive* (proof is in the full paper [19]).

Consider two temporal events E_i and E_j, and their corresponding instances, $e_i = (\omega_i, [t_{s_i}, t_{e_i}])$ and $e_j = (\omega_j, [t_{s_j}, t_{e_j}])$. Let ϵ be a non-negative number ($\epsilon \geq 0$) representing the buffer size. The following relations can be defined between E_i and E_j through e_i and e_j.

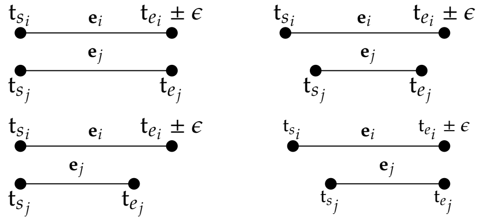
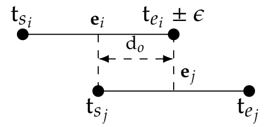
Definition 3.6 (Follows) E_i and E_j form a *Follows* relation through e_i and e_j, denoted as Follows($E_{i_{\triangleright e_i}}$,$E_{j_{\triangleright e_j}}$) or $E_{i_{\triangleright e_i}} \rightarrow E_{j_{\triangleright e_j}}$, iff $t_{e_i} \pm \epsilon \leq t_{s_j}$.

Definition 3.7 (Contains) E_i and E_j form a *Contains* relation through e_i and e_j, denoted as Contains($E_{i_{\triangleright e_i}}$, $E_{j_{\triangleright e_j}}$) or $E_{i_{\triangleright e_i}} \succcurlyeq E_{j_{\triangleright e_j}}$, iff $(t_{s_i} \leq t_{s_j}) \wedge (t_{e_i} \pm \epsilon \geq t_{e_j})$.

Definition 3.8 (Overlaps) E_i and E_j form an *Overlaps* relation through e_i and e_j, denoted as Overlaps($E_{i_{\triangleright e_i}}$, $E_{j_{\triangleright e_j}}$) or $E_{i_{\triangleright e_i}} \between E_{j_{\triangleright e_j}}$, iff $(t_{s_i} < t_{s_j}) \wedge (t_{e_i} \pm \epsilon < t_{e_j}) \wedge (t_{e_i} - t_{s_j} \geq d_o \pm \epsilon)$, where d_o is the minimal overlapping duration between two event instances, and $0 \leq \epsilon \ll d_o$.

The *Follows* relation represents sequential occurrences of one event after another. For example, $E_{i_{\triangleright e_i}}$ is followed by $E_{j_{\triangleright e_j}}$ if the end time t_{e_i} of e_i occurs before the start time t_{s_j} of e_j. Here, the buffer ϵ is used as a tolerance, i.e., the *Follows* relation between $E_{i_{\triangleright e_i}}$ and $E_{j_{\triangleright e_j}}$ holds if $(t_{e_i} + \epsilon)$ or $(t_{e_i} - \epsilon)$ occurs before t_{s_j}. On the other hand, in a *Contains* relation, one event occurs entirely within the timespan of another event. Finally, in an *Overlaps* relation, the timespans of the two occurrences overlap each other. Table A.2 illustrates the three temporal relations and their conditions.

Table A.2: Temporal Relations between Events

Follows: $E_{i \triangleright e_i} \rightarrow E_{j \triangleright e_j}$	 $t_{e_i} \pm \epsilon \leq t_{s_j}$
Contains: $E_{i \triangleright e_i} \succcurlyeq E_{j \triangleright e_j}$	 $(t_{s_i} \leq t_{s_j}) \wedge (t_{e_i} \pm \epsilon \geq t_{e_j})$
Overlaps: $E_{i \triangleright e_i} \between E_{j \triangleright e_j}$	 $(t_{s_i} < t_{s_j}) \wedge (t_{e_i} \pm \epsilon < t_{e_j}) \wedge (t_{e_i} - t_{s_j} \geq d_o \pm \epsilon)$

A.3.3 Temporal Pattern

Definition 3.9 (Temporal sequence) A list of n event instances S=<e_1, ... , e_i, ... , e_n> forms a *temporal sequence* if the instances are chronologically ordered by their start times. Moreover, S has size n, denoted as $|S| = n$.

Definition 3.10 (Temporal sequence database) A set of temporal sequences forms a *temporal sequence database* $\mathcal{D}_{\text{SEQ}}$ where each row i contains a temporal sequence S_i.

Table A.3 shows the temporal sequence database $\mathcal{D}_{\text{SEQ}}$, created from the symbolic database $\mathcal{D}_{\text{SYB}}$ in Table A.1.

Table A.3: A Temporal Sequence Database $\mathcal{D}_{\text{SEQ}}$

ID	Temporal sequences
1	(SOn,[10:00,10:15]), (TOff,[10:00,10:05]), (MOff,[10:00,10:20]), (WOff,[10:00,10:20]), (DOff,[10:00,10:40]), (IOff,[10:00,10:35]), (TOn,[10:05,10:15]), (SOff,[10:15,10:35]), (TOff,[10:15,10:35]), (MOn,[10:20,10:30]), (WOn,[10:20,10:30]), (WOff,[10:30,10:40]), (MOff,[10:30,10:40]), (SOn,[10:35,10:40]), (TOn,[10;35,10:40]), (IOn,[10:35,10:40])
2	(SOff,[10:45,11:15]), (TOff,[10:45,10:55]), (MOn,[10;45,10:55]), (WOn,[10:45,10:50]), (DOn,[10:45,10:50]), (IOff,[10:45,11:25]), (WOff,[10:50,11:00]), (DOff,[10:50,11:20]), (MOff,[10:55,11:05]), (TOn,[10:55,11:00]), (TOff,[11:00,11:15]), (WOn,[11:00,11:10]), (MOn,[11:05,11:10]), (WOff,[11:10,11:25]), (MOff,[11:10,11:25]), (SOn,[11:15,11:25]), (TOn,[11:15,11:25]), (DOn,[11:20,11:25])
3	(SOff,[11:30,11:50]), (TOff,[11:30,11:50]), (MOn,[11:30,11:35]), (WOn,[11:30,11:35]), (DOff,[11:30,12:10]), (IOn,[11:30,11:35]), (IOff,[11:35,12:10]), (MOff,[11:35,11:45]), (WOff,[11:35,11:45]), (WOn,[11:45,11:50]), (MOn,[11:45,11:50]), (SOn,[11:50,12:00]), (MOff,[11:50,12:05]), (TOn,[11:50,12:00]), (WOff,[11:50,12:05]), (SOff,[12:00,12:10]), (TOff,[12:00,12:10]), (MOn,[12:05,12:10]), (WOn,[12:05,12:10])
4	(SOn,[12:15,12:20]), (TOn,[12:15,12:20]), (MOff,[12:15,12:25]), (WOff,[12:15,12:25]), (DOn,[12:15,12:20]), (IOn,[12:15,12:20]), (IOff,[12:20,12:50]), (DOff,[12:20,12:40]), (TOff,[12:20,12:40]), (SOff,[12:20,12:35]), (WOn,[12:25,12:35]), (MOn,[12:25,12:35]), (MOff,[12:35,12:50]), (SOn,[12:35,12:45]), (WOff,[12:35,12:50]), (TOn,[12:40,12:50]), (DOn,[12:40,12:45]), (DOff,[12:45,12:55]), (SOff,[12:45,12:55]), (TOff,[12:50,12:55]), (MOn,[12:50,12:55]), (WOn,[12:50,12:55]), (IOn,[12:50,12:55])

Definition 3.11 (Temporal pattern) Let $\mathfrak{R}$={Follows, Contains, Overlaps} be the set of temporal relations. A *temporal pattern* P=<(r_{12}, E_1, E_2),...,$(r_{(n-1)(n)}$, E_{n-1} , $E_n)$> is a list of triples $(r_{ij}$,E_i,$E_j)$, each representing a relation $r_{ij} \in \mathfrak{R}$ between two events E_i and E_j.

Note that the relation r_{ij} in each triple is formed using the specific instances of E_i and E_j. A temporal pattern that has n events is called an n-event pattern. We use $E_i \in P$ to denote that the event E_i occurs in P, and $P_1 \subseteq P$ to say that a pattern P_1 is a sub-pattern of P.

Definition 3.12 (Temporal sequence supports a pattern) Let S=<e_1, ...,e_i,...,e_n> be a temporal sequence. We say that S *supports* a temporal pattern P, denoted as $P \in S$, iff $|S| \geq 2 \wedge \forall(r_{ij}, E_i, E_j) \in P, \exists(e_l, e_m) \in S$ such that r_{ij} holds between $E_{i_{\triangleright e_l}}$ and $E_{j_{\triangleright e_m}}$.

If P is supported by S, P can be written as P=<$(r_{12}, E_{1_{\triangleright e_1}}, E_{2_{\triangleright e_2}})$, ..., $(r_{(n-1)(n)}$,$E_{n-1_{\triangleright e_{n-1}}}$, $E_{n_{\triangleright e_n}})$>, where the relation between two events in each triple is expressed using the event instances.

In Fig. A.1, consider the sequence S =<e_1=(HighCO2, [6:00, 10:00]), e_2=(BoilerOn, [7:00, 8:00]), e_3=(LowCO2, [13:00, 15:00])> representing the order of CO2 intensity and boiler events. Here, S supports a 3-event pattern P=<(Contains, HighCO2$_{\triangleright e_1}$, BoilerOn$_{\triangleright e_2}$), (Follows, HighCO2$_{\triangleright e_1}$, LowCO2$_{\triangleright e_3}$), (Follows, BoilerOn$_{\triangleright e_2}$, LowCO2$_{\triangleright e_3}$)>.

Maximal duration constraint: Let $P \in S$ be a temporal pattern supported by the sequence S. The duration between the start time of the instance e_1, and the end time of the instance e_n in S must not exceed the predefined maximal time duration $t_{\max}$: $t_{e_n} - t_{s_1} \leq t_{\max}$.

The maximal duration constraint guarantees that the relation between any two events is temporally valid. This enables the pruning of invalid patterns.

For example, under this constraint, a *Follows* relation between a *"Washer On"* event and a *"Dryer On"* event in Table A.3 happening one year apart should be considered invalid.

A.3.4 Frequent Temporal Pattern

Given a temporal sequence database $\mathcal{D}_{\text{SEQ}}$, we want to find patterns that occur frequently in $\mathcal{D}_{\text{SEQ}}$. We use *support* and *confidence* [34] to measure the frequency and the likelihood of a pattern.

Definition 3.13 (Support of a temporal event) The *support* of a temporal event E in $\mathcal{D}_{\text{SEQ}}$ is the number of sequences $S \in \mathcal{D}_{\text{SEQ}}$ which contain at least one instance e of E.

$$supp(E) = |\{S \in \mathcal{D}_{\text{SEQ}} \ s.t. \ \exists e \in S : E_{\triangleright e}\}| \tag{A.1}$$

The *relative support* of E is the fraction between $supp(E)$ and the size of $\mathcal{D}_{\text{SEQ}}$:

$$rel\text{-}supp(E) = supp(E)/|\mathcal{D}_{\text{SEQ}}| \tag{A.2}$$

Similarly, the support of a group of events $(E_1, ..., E_n)$, denoted as $supp(E_1, ..., E_n)$, is the number of sequences $S \in \mathcal{D}_{\text{SEQ}}$ which contain at least one instance $(e_1, ..., e_n)$ of the event group.

Definition 3.14 (Support of a temporal pattern) The *support* of a pattern P is the number of sequences $S \in \mathcal{D}_{\text{SEQ}}$ that support P.

$$supp(P) = |\{S \in \mathcal{D}_{\text{SEQ}} \ s.t. \ P \in S\}| \tag{A.3}$$

The *relative support* of P in $\mathcal{D}_{\text{SEQ}}$ is the fraction

$$rel\text{-}supp(P) = supp(P)/|\mathcal{D}_{\text{SEQ}}| \tag{A.4}$$

Definition 3.15 (Confidence of an event pair) The *confidence* of an event pair (E_i, E_j) in $\mathcal{D}_{\text{SEQ}}$ is the fraction between $supp(E_i, E_j)$ and the support of its most frequent event:

$$conf(E_i, E_j) = \frac{supp(E_i, E_j)}{\max\{supp(E_i), supp(E_j)\}} \tag{A.5}$$

Definition 3.16 (Confidence of a temporal pattern) The *confidence* of a temporal pattern P in $\mathcal{D}_{\text{SEQ}}$ is the fraction between $supp(P)$ and the support of its most frequent event:

$$conf(P) = \frac{supp(P)}{\max_{1 \le k \le |P|}\{supp(E_k)\}} \tag{A.6}$$

where $E_k \in P$ is a temporal event. Since the denominator in Eq. (B.6) is the maximum support of the events in P, the confidence computed in Eq. (B.6) is the *minimum confidence* of a pattern P in $\mathcal{D}_{\text{SEQ}}$, which is also called the *all-confidence* as in [34].

Note that unlike association rules, temporal patterns do not have antecedents and consequents. Instead, they represent pair-wise temporal relations between events based on their temporal occurrences. Thus, while the *support* and *relative support* of event(s)/ pattern(s) defined in Eqs. (B.1) – (B.4) follow the same intuition as the traditional support concept, indicating how frequently an event/ pattern occurs in a given database, the *confidence* computed in Eqs. (B.5) – (B.6) instead represents the minimum likelihood of an event pair/ pattern, knowing the likelihood of its most frequent event.

Frequent Temporal Pattern Mining from Time Series (FTP MfTS). Given a set of univariate time series $\mathcal{X} = \{X_1, ..., X_n\}$, let $\mathcal{D}_{\text{SEQ}}$ be the temporal sequence database obtained from $\mathcal{X}$, and σ and δ be the support and confidence thresholds, respectively. The FTPMfTS problem aims to find all temporal patterns P that have high enough support and confidence in $\mathcal{D}_{\text{SEQ}}$: $supp(P) \geq \sigma \wedge conf(P) \geq \delta$.

A.4 Frequent Temporal Pattern Mining

Fig. A.2 gives an overview of the FTPMfTS process which consists of 2 phases. The first phase, *Data Transformation*, converts a set of time series $\mathcal{X}$ into a symbolic database $\mathcal{D}_{\text{SYB}}$, and then converts $\mathcal{D}_{\text{SYB}}$ into a temporal sequence database $\mathcal{D}_{\text{SEQ}}$. The second phase, *Frequent Temporal Pattern Mining*, mines frequent patterns which includes 3 steps: (1) *Frequent Single Event Mining*, (2) *Frequent 2-Event Pattern Mining*, and (3) *Frequent k-Event Pattern Mining* (k>2). The final output is a set of all frequent patterns in $\mathcal{D}_{\text{SEQ}}$.

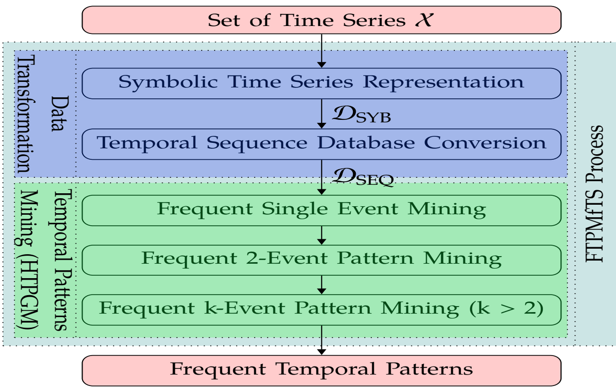


Fig. A.2: The FTPMfTS process

A.4.1 Data Transformation

Symbolic Time Series Representation

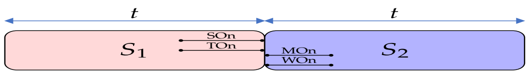
Given a set of time series $\mathcal{X}$, the symbolic representation of each time series $X \in \mathcal{X}$ is obtained by using a mapping function as in Def. 3.2.

Temporal Sequence Database Conversion

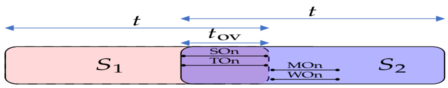
To convert $\mathcal{D}_{\text{SYB}}$ to $\mathcal{D}_{\text{SEQ}}$, a straightforward approach is to split the symbolic series in $\mathcal{D}_{\text{SYB}}$ into equal-length sequences, each belongs to a row in $\mathcal{D}_{\text{SEQ}}$. For example, if each symbolic series in Table A.1 is split into 4 sequences, then each sequence will last for 40 minutes. The first sequence S_1 of $\mathcal{D}_{\text{SEQ}}$ therefore contains temporal events of S, T, M, W, D, and I from 10:00 to 10:40. The second sequence S_2 contains events from 10:45 to 11:25, and similarly for S_3 and S_4.

However, the splitting can lead to a potential loss of temporal patterns. The loss happens when a *splitting point* accidentally divides a temporal pattern into different sub-patterns, and places these into separate sequences. We explain this situation in Fig. A.3a. Consider 2 sequences S_1 and S_2, each of length t. Here, the splitting point divides a pattern of 4 events, {SOn, TOn, MOn, WOn}, into two sub-patterns, in which SOn and TOn are placed in S_1, and MOn and WOn in S_2. This results in the loss of this 4-event pattern which can be identified only when all 4 events are in the same sequence.

To prevent such a loss, we propose a *splitting strategy* using overlapping sequences. Specifically, two consecutive sequences are overlapped by a duration t_{ov}: $0 \leq t_{\text{ov}} \leq t_{\max}$, where $t_{\max}$ is the *maximal duration* of a temporal pattern. The value of t_{ov} decides how large the overlap between S_i and S_{i+1} is: $t_{\text{ov}} = 0$ results in no overlap, i.e., no redundancy, but with a potential loss of patterns, while $t_{\text{ov}} = t_{\max}$ creates large overlaps between sequences, i.e., high redundancy, but all patterns are preserved. As illustrated in Fig. A.3b, the overlapping between S_1 and S_2 keeps the 4 events together in the same sequence S_2, and thus helps preserve the pattern.



(a) With no overlapping



(b) With overlapping

Fig. A.3: Splitting strategy

A.4.2 Frequent Temporal Patterns Mining

We now present our method, called Hierarchical Temporal Pattern Graph Mining (HTPGM), to mine frequent temporal patterns from $\mathcal{D}_{\text{SEQ}}$. The main novelties of HTPGM are: a) the use of efficient data structures, i.e., the proposed Hierarchical Pattern Graph and *bitmap indexing*, to enable fast computations of support and confidence, and b) the proposal of two groups of pruning techniques based on the Apriori principle and the temporal transitivity property of temporal events. In Section A.5, we introduce an approximate version of HTPGM based on mutual information to further optimize the mining process. We first discuss the data structures used in HTPGM.

Hierarchical Pattern Graph (HPG): We use a hierarchical graph structure, called the *Hierarchical Pattern Graph*, to keep track of the frequent events and patterns found in each mining step. The HPG allows HTPGM to mine iteratively (e.g., 2-event patterns are mined based on frequent single events, 3-event patterns are mined based on 2-event patterns, and so on) and perform effective pruning. Fig. A.4 shows the HPG built from $\mathcal{D}_{\text{SEQ}}$ in Table A.3: the root is the empty set $\emptyset$, and each level L_k maintains frequent k-event patterns. As HTPGM proceeds, HPG is constructed gradually. We explain this process for each mining step.

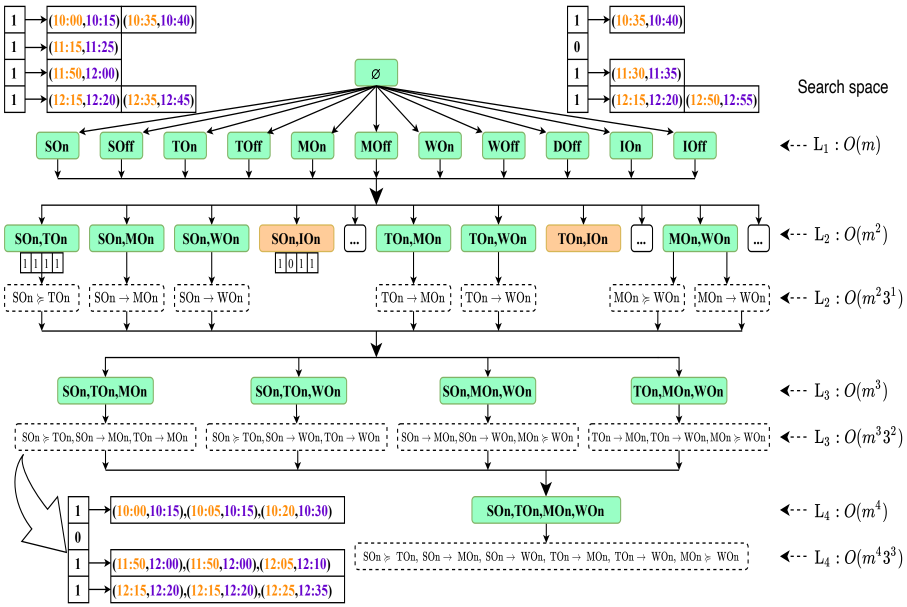


Fig. A.4: A Hierarchical Pattern Graph for Table A.3

Efficient bitmap indexing: We use *bitmaps* to index the occurrences of events and patterns in $\mathcal{D}_{\text{SEQ}}$, enabling fast computations of support and confidence. Specifically, each event E or pattern P found in $\mathcal{D}_{\text{SEQ}}$ is associated with a *bitmap* indicating where E or P occurs. Each *bitmap* b has length $|\mathcal{D}_{\text{SEQ}}|$ (i.e., the number of sequences), and has value $b[i] = 1$ if E or P is present in sequence i of $\mathcal{D}_{\text{SEQ}}$, or $b[i] = 0$ otherwise. An example *bitmap* can be seen at L_1 in Fig. A.4. The event IOn has the *bitmap* $b_{\text{IOn}} = [1,0,1,1]$, indicating that IOn occurs in all but the second sequence of $\mathcal{D}_{\text{SEQ}}$.

Constructing the *bitmap* is also done step by step. For single events in

Algorithm 7: Hierarchical Temporal Pattern Graph Mining

Input: Temporal sequence database $\mathcal{D}_{\text{SEQ}}$, a support threshold σ, a confidence threshold δ

Output: The set of frequent temporal patterns P

// Mining frequent single events
1: **foreach** *event* $E_i \in \mathcal{D}_{SEQ}$ **do**
2:     $supp(E_i) \leftarrow$ countBitmap(b_{E_i});
3:     **if** $supp(E_i) \geq \sigma$ **then**
4:         Insert E_i to *1Freq*;
// Mining frequent 2-event patterns
5: EventPairs $\leftarrow$ Cartesian(*1Freq*,*1Freq*);
6: FrequentPairs $\leftarrow \emptyset$;
7: **foreach** (E_i, E_j) *in EventPairs* **do**
8:     $b_{ij} \leftarrow$ AND(b_{E_i},b_{E_j});
9:     $supp(E_i, E_j) \leftarrow$ countBitmap(b_{ij});
10:     **if** $supp(E_i, E_j) \geq \sigma$ **then**
11:         FrequentPairs $\leftarrow$ Apply_Lemma3(E_i, E_j);
12: **foreach** (E_i, E_j) *in FrequentPairs* **do**
13:     Retrieve event instances;
14:     Check frequent relations;
// Mining frequent k-event patterns
15: Filtered1Freq $\leftarrow$ Transitivity_Filtering(1Freq); //Lemmas 4, 5
16: kEventCombinations $\leftarrow$ Cartesian(*Filtered1Freq*,*(k-1)Freq*);
17: FrequentkEvents $\leftarrow$ Apriori_Filtering(kEventCombinations);
18: **foreach** *kEvents in FrequentkEvents* **do**
19:     Retrieve relations;
20:     Iteratively check frequent relations; //Lemmas 4, 6, 7

$\mathcal{D}_{\text{SEQ}}$, *bitmaps* are built by scanning $\mathcal{D}_{\text{SEQ}}$ only once. Algorithm 7 provides the pseudo-code of HTPGM. The details are explained in each mining step.

A.4.3 Mining Frequent Single Events

The first step in HTPGM is to find frequent single events (Alg. 7, lines 1-4) which is easily done using the *bitmap*. For each event E_i in $\mathcal{D}_{\text{SEQ}}$, the support $supp(E_i)$ is computed by counting the number of set bits in *bitmap* b_{E_i}, and comparing against σ. Note that for single events, *confidence* is not considered since it is always 1.

After this step, the set *1Freq* containing frequent single events is created to build L_1 of HPG. We illustrate this process using Table A.3, with $\sigma = 0.7$ and $\delta = 0.7$. Here, *1Freq* contains 11 frequent events, each belongs to one node in L_1. The event DOn is not frequent (only appears in sequences 2 and 4), and is thus omitted. Each L_1 node has a unique event name, a *bitmap*, and a list of instances corresponding to that event (see SOn at L_1).

Complexity: The complexity of finding frequent single events is $O(m \cdot |\mathcal{D}_{\text{SEQ}}|)$, where m is the number of distinct events.

Proof. *Detailed proofs of all complexities, lemmas and theorems in this article can be found in the Appendix of the full paper [19].*

A.4.4 Mining Frequent 2-event Patterns

Search space of HTPGM

The next step in HTPGM is to mine frequent 2-event patterns. A straightforward approach would be to enumerate all possible event pairs, and check whether each pair can form frequent patterns. However, this *naive* approach is very expensive. Not only does it need to repeatedly scan $\mathcal{D}_{\text{SEQ}}$ to check each combination of events, the complex relations between events also add an extra exponential factor 3^{h^2} to the m^h number of possible candidates, creating a very large search space that makes the approach infeasible.

Lemma 1 *Let m be the number of distinct events in $\mathcal{D}_{SEQ}$, and h be the longest length of a temporal pattern. The total number of temporal patterns in HPG from L_1 to L_h is $O(m^h 3^{h^2})$.*

Lemma 1 shows the driving factors of HTPGM's exponential search space (proof in [19]): the number of events (m), the max pattern length (h), and the number of temporal relations (3). A dataset of just a few hundred events can create a search space with billions of candidate patterns. The optimizations and approximation proposed in the following sections help mitigate this problem.

Two-steps filtering approach

Given the huge set of pattern candidates, it is expensive to check their support and confidence. We propose a *filtering approach* to reduce the unnecessary candidate checking. Specifically, at any level l ($l \geq 2$) in HPG, the mining process is divided into two steps: (1) it first finds frequent nodes (i.e., remove infrequent combinations of events), (2) it then generates temporal patterns only from frequent nodes. The correctness of this filtering approach is based on the Apriori-inspired lemmas below.

Lemma 2 *Let P be a 2-event pattern formed by an event pair (E_i, E_j). Then, supp(P) $\leq$ supp(E_i, E_j).*

From Lemma 2, the support of a pattern is at most the support of its events. Thus, infrequent event pairs cannot form frequent patterns and thereby, can be safely pruned.

Lemma 3 *Let (E_i, E_j) be a pair of events occurring in a 2-event pattern P. Then conf(P) $\leq$ conf(E_i, E_j).*

From Lemma 3, the confidence of a pattern P is always at most the confidence of its events. Thus, a low-confidence event pair cannot form any high-confidence patterns and therefore, can be safely pruned. We note that the Apriori principle has already been used in other work, e.g., [8, 35], for mining optimization. However, they only apply this principle to the support (Lemma 2), while we further extend it to the confidence (Lemma 3). Applying Lemmas 2 and 3 to the first filtering step will remove infrequent or low-confidence event pairs, reducing the candidate patterns of HTPGM. We detail this filtering below.

Step 2.1. Mining frequent event pairs: This step finds frequent event pairs in $\mathcal{D}_{\text{SEQ}}$, using the set *1Freq* found in L_1 of HPG (Alg. 7, lines 5-11). First, HTPGM generates all possible event pairs by calculating the Cartesian product *1Freq* $\times$ *1Freq*. Next, for each pair (E_i, E_j), the joint *bitmap* b_{ij} (representing the set of sequences where both events occur) is computed by *ANDing* the two individual bitmaps: $b_{ij} = AND(b_{E_i}, b_{E_j})$. Finally, HTPGM computes the support $supp(E_i, E_j)$ by counting the set bits in b_{ij}, and comparing against σ. If $supp(E_i, E_j) \geq \sigma$, (E_i, E_j) has high enough support. Next, (E_i, E_j) is further filtered using Lemma 3: (E_i, E_j) is selected only if its confidence is at least δ. After this step, only frequent and high-confidence event pairs remain and form the nodes in L_2.

Step 2.2. Mining frequent 2-event patterns: This step finds frequent 2-event patterns from the nodes in L_2 (Alg. 7, lines 12-14). For each node $(E_i, E_j) \in L_2$, we use the *bitmap* b_{ij} to retrieve the set of sequences $\mathcal{S}$ where both events are present. Next, for each sequence $S \in \mathcal{S}$, the pairs of event instances (e_i, e_j) are extracted, and the relations between them are verified. The support and confidence of each relation $r(E_{i_{\triangleright e_i}}, E_{j_{\triangleright e_j}})$ are computed and compared against the thresholds, after which only frequent relations are selected and stored in the corresponding node in L_2. Examples of the relations in L_2 can be seen in Fig. A.4, e.g., node (SOn, TOn).

Step 2.2 results in two different sets of nodes in L_2. The first set contains nodes that have frequent events but do not have any frequent patterns. These nodes (colored in brown in Fig. A.4) are removed from L_2. The second set contains nodes that have both frequent events and frequent patterns (colored in green), which remain in L_2 and are used in the subsequent mining steps.

Complexity: Let m be the number of frequent single events in L_1, and i be the average number of event instances of each frequent event. The complexity of frequent 2-event pattern mining is $O(m^2 i^2 \, |\mathcal{D}_{\text{SEQ}}|^2)$.

A.4.5 Mining Frequent k-event Patterns

Mining frequent k-event patterns ($k \geq 3$) follows a similar process as 2-event patterns, with additional prunings based on the transitivity property of temporal relations.

Step 3.1. Mining frequent k-event combinations: This step finds frequent k-event combinations in L_k (Alg. 7, lines 15-17).

Let *(k-1)Freq* be the set of frequent (k-1)-event combinations found in L_{k-1}, and *1Freq* be the set of frequent single events in L_1. To generate all k-event combinations, the typical process is to compute the Cartesian product: *(k-1)Freq* $\times$ *1Freq*. However, we observe that using *1Freq* to generate k-event combinations at L_k can create redundancy, since *1Freq* might contain events that when combined with nodes in L_{k-1}, result in combinations that clearly cannot form any frequent patterns. To illustrate this observation, consider node IOn at L_1 in Fig. A.4. Here, IOn is a frequent event, and thus, can be combined with frequent nodes in L_2 such as (SOn, TOn) to create a 3-event combination (SOn, TOn, IOn). However, (SOn, TOn, IOn) cannot form any frequent 3-event patterns, since IOn is not present in any frequent 2-event patterns in L_2. To reduce the redundancy, the combination (SOn, TOn, IOn) should not be created in the first place. We rely on the *transitivity property* of temporal relations to identify such event combinations.

Lemma 4 *Let* $S =< e_1,..., e_{n-1} >$ *be a temporal sequence that supports an (n-1)-event pattern* $P =< (r_{12}, E_{1_{\triangleright e_1}}, E_{2_{\triangleright e_2}}),..., (r_{(n-2)(n-1)}, E_{n-2_{\triangleright e_{n-2}}}, E_{n-1_{\triangleright e_{n-1}}}) >$. *Let* e_n *be a new event instance added to* S *to create the temporal sequence* $S' =< e_1, ..., e_n >$.

The set of temporal relations $\mathfrak{R}$ *is transitive on* S': $\forall e_i \in S', i < n, \exists r \in \mathfrak{R}$ *s.t.* $r(E_{i_{\triangleright e_i}}, E_{n_{\triangleright e_n}})$ *holds.*

Lemma 4 says that given a temporal sequence S, a new event instance added to S will always form at least one temporal relation with existing instances in S. This is due to the temporal transitivity property, which can be used to prove the following lemma.

Lemma 5 *Let* $N_{k-1} = (E_1, ..., E_{k-1})$ *be a frequent (k-1)-event combination, and* E_k *be a frequent single event. The combination* $N_k = N_{k-1} \cup E_k$ *can form frequent k-event temporal patterns if* $\forall E_i \in N_{k-1}, \exists r \in \mathfrak{R}$ *s.t.* $r(E_i, E_k)$ *is a frequent temporal relation.*

From Lemma 5, only single events in L_1 that occur in L_{k-1} should be used to create k-event combinations. Using this result, a filtering on *1Freq* is performed before calculating the Cartesian product. Specifically, from the nodes in L_{k-1}, we extract the distinct single events D_{k-1}, and *intersect* them with *1Freq* to remove redundant single events: *Filtered1Freq* = $D_{k-1} \cap$ *1Freq*. Next, the Cartesian product *(k-1)Freq* $\times$ *Filtered1Freq* is calculated to generate k-event combinations. Finally, we apply Lemmas 2 and 3 to select frequent and high-confidence k-event combinations *kFreq* to form L_k.

Step 3.2 Mining frequent k-event patterns: This step finds frequent k-event patterns from the nodes in L_k (Alg. 7, lines 18-20). Unlike 2-event patterns, determining the relations in a k-event combination ($k \geq 3$) is much more expensive, as it requires to verify the frequency of $\frac{1}{2}k(k-1)$ triples.

To reduce the cost of relation checking, we propose an iterative verification method that relies on the *transitivity property* and the Apriori principle.

Lemma 6 *Let P and P′ be two temporal patterns. If P′ ⊆ P, then conf(P′) ≥ conf(P).*

Lemma 7 *Let P and P′ be two temporal patterns. If P′ ⊆ P and* $\frac{supp(P')}{\max_{1 \le k \le |P|}\{supp(E_k)\}}_{E_k \in P}$ *≤ δ, then conf(P) ≤ δ.*

Lemma 6 says that, the confidence of a pattern P is always at most the confidence of its sub-patterns. Consequently, from Lemma 7, a temporal pattern P cannot be high-confidence if any of its sub-patterns are low-confidence.

Let $N_{k-1} = (E_1, ..., E_{k-1})$ be a node in L_{k-1}, $N_1 = (E_k)$ be a node in L_1, and $N_k = N_{k-1} \cup N_1 = (E_1, ..., E_k)$ be a node in L_k. To find k-event patterns for N_k, we first retrieve the set P_{k-1} containing frequent (k-1)-event patterns in node N_{k-1}. Each $p_{k-1} \in P_{k-1}$ is a list of $\frac{1}{2}(k-1)(k-2)$ triples: $\{(r_{12}, E_{1_{\triangleright e_1}}, E_{2_{\triangleright e_2}}),...,(r_{(k-2)(k-1)}, E_{k-2_{\triangleright e_{k-2}}}, E_{k-1_{\triangleright e_{k-1}}})\}$. We iteratively verify the possibility of p_{k-1} forming a frequent k-event pattern with E_k as follows.

We first check whether the triple $(r_{(k-1)k}, E_{k-1_{\triangleright e_{k-1}}}, E_{k_{\triangleright e_k}})$ is frequent and high-confidence by accessing the node (E_{k-1}, E_k) in L_2. If the triple is not frequent (using Lemmas 4 and 5) or high-confidence (using Lemmas 4, 6, and 7), the verifying process stops immediately for p_{k-1}. Otherwise, it continues on the triple $(r_{(k-2)k}, E_{k-2_{\triangleright e_{k-2}}}, E_{k_{\triangleright e_k}})$, until it reaches $(r_{1k}, E_{1_{\triangleright e_1}}, E_{k_{\triangleright e_k}})$.

We note that the transitivity property of temporal relations has been exploited in [32] to generate new relations. Instead, we use this property to prune unpromising candidates (Lemmas 4, 5, 6, 7).

Complexity: Let r be the average number of frequent (k-1)-event patterns in L_{k-1}. The complexity of frequent k-event pattern mining is $O(|1Freq| \cdot |L_{k-1}| \cdot r \cdot k^2 \cdot |\mathcal{D}_{SEQ}|)$.

HTPGM overall complexity: Throughout this section, we have seen that HTPGM complexity depends on the size of the search space ($O(m^h 3^{h^2})$) and the complexity of the mining process itself, i.e., $O(m \cdot |\mathcal{D}_{SEQ}|) + O(m^2 i^2 \, |\mathcal{D}_{SEQ}|^2) + O(|1Freq| \cdot |L_{k-1}| \cdot r \cdot k^2 \cdot |\mathcal{D}_{SEQ}|)$. While the parameters m, h, i, r and k depend on the number of time series, others such as $|1Freq|$, $|L_{k-1}|$ and $|\mathcal{D}_{SEQ}|$ also depend on the number of temporal sequences. Thus, given a dataset, HTPGM complexity is driven by two main factors: the number of time series and the number of temporal sequences.

A.5 Approximate HTPGM

A.5.1 Correlated Symbolic Time Series

Let X_S and Y_S be the symbolic series representing the time series X and Y, respectively, and Σ_X, Σ_Y be their symbolic alphabets.

Definition 5.1 (Entropy) The *entropy* of X_S, denoted as $H(X_S)$, is defined as

$$H(X_S) = -\sum_{x \in \Sigma_X} p(x) \cdot \log p(x) \tag{A.7}$$

Intuitively, the entropy measures the amount of information or the inherent uncertainty in the possible outcomes of a random variable. The higher the $H(X_S)$, the more uncertain the outcome of X_S.

The conditional entropy $H(X_S|Y_S)$ quantifies the amount of information needed to describe the outcome of X_S, given the value of Y_S, and is defined as

$$H(X_S|Y_S) = -\sum_{x \in \Sigma_X} \sum_{y \in \Sigma_Y} p(x,y) \cdot \log \frac{p(x,y)}{p(y)} \tag{A.8}$$

Definition 5.2 (Mutual information) The *mutual information* of two symbolic series X_S and Y_S, denoted as $I(X_S;Y_S)$, is defined as

$$I(X_S;Y_S) = \sum_{x \in \Sigma_X} \sum_{y \in \Sigma_Y} p(x,y) \cdot \log \frac{p(x,y)}{p(x) \cdot p(y)} \tag{A.9}$$

The MI represents the reduction of uncertainty of one variable (e.g., X_S), given the knowledge of another variable (e.g., Y_S). The larger $I(X_S;Y_S)$, the more information is shared between X_S and Y_S, and thus, the less uncertainty about one variable given the other.

We demonstrate how to compute the MI between the symbolic series S and T in Table A.1. We have: p(SOn)=$\frac{17}{36}$, p(SOff)=$\frac{19}{36}$, p(TOn)=$\frac{18}{36}$, and p(TOff)=$\frac{18}{36}$. We also have the joint probabilities: p(SOn, TOn) = $\frac{15}{36}$, p(SOff, TOff) = $\frac{16}{36}$, p(SOn, TOff) = $\frac{2}{36}$, and p(SOff, TOn) = $\frac{3}{36}$. Applying Eq. A.9, we have $I(S;T) = 0.29$.

Since $0 \leq I(X_S;Y_S) \leq \min(H(X_S), H(Y_S))$ [10], the MI value has no upper bound. To scale the MI into the range $[0-1]$, we use normalized mutual information as defined below.

Definition 5.3 (Normalized mutual information) The *normalized mutual information* (NMI) of two symbolic time series X_S and Y_S, denoted as $\widetilde{I}(X_S;Y_S)$, is defined as

$$\widetilde{I}(X_S;Y_S) = \frac{I(X_S;Y_S)}{H(X_S)} = 1 - \frac{H(X_S|Y_S)}{H(X_S)} \tag{A.10}$$

$\widetilde{I}(X_S;Y_S)$ represents the reduction (in percentage) of the uncertainty of X_S due to knowing Y_S. Based on Eq. (A.10), a pair of variables (X_S, Y_S) holds a mutual dependency if $\widetilde{I}(X_S;Y_S) > 0$. Eq. (A.10) also shows that NMI is not symmetric, i.e., $\widetilde{I}(X_S;Y_S) \neq \widetilde{I}(Y_S;X_S)$.

Using Table A.1, we have $I(S;T) = 0.29$. However, we do not know what the 0.29 reduction means in practice. Applying Eq. (A.10), we can compute NMI $\widetilde{I}(S;T) = 0.43$, which says that the uncertainty of S is reduced by 43% given T. Moreover, we also have $\widetilde{I}(T;S) = 0.42$, showing that $\widetilde{I}(S;T) \neq \widetilde{I}(T;S)$.

Definition 5.4 (Correlated symbolic time series) Let μ ($0 < \mu \leq 1$) be the mutual information threshold. We say that the two symbolic series X_S and Y_S are *correlated* iff $\widetilde{I}(X_S; Y_S) \geq \mu \vee \widetilde{I}(Y_S; X_S) \geq \mu$, and *uncorrelated* otherwise.

A.5.2 Lower Bound of the Confidence

Derivation of the lower bound

Consider 2 symbolic series X_S and Y_S. Let X_1 be a temporal event in X_S, Y_1 be a temporal event in Y_S, and $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$ be the symbolic and the sequence databases created from X_S and Y_S, respectively. We first study the relationship between the support of (X_1, Y_1) in $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$.

Lemma 1 *Let $supp(X_1, Y_1)_{\mathcal{D}_{SYB}}$ and $supp(X_1, Y_1)_{\mathcal{D}_{SEQ}}$ be the support of (X_1, Y_1) in $\mathcal{D}_{SYB}$ and $\mathcal{D}_{SEQ}$, respectively. We have the following relation: $supp(X_1, Y_1)_{\mathcal{D}_{SYB}} \leq supp(X_1, Y_1)_{\mathcal{D}_{SEQ}}$.*

From Lemma 1, if an event pair is frequent in $\mathcal{D}_{\text{SYB}}$, it is also frequent in $\mathcal{D}_{\text{SEQ}}$. We now investigate the connection between $\widetilde{I}(X_S; Y_S)$ in $\mathcal{D}_{\text{SYB}}$, and the confidence of (X_1, Y_1) in $\mathcal{D}_{\text{SEQ}}$.

Theorem 1 *(Lower bound of the confidence) Let σ and μ be the minimum support and mutual information thresholds, respectively. Assume that (X_1, Y_1) is frequent in $\mathcal{D}_{SEQ}$, i.e., $supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \sigma$. If the NMI $\widetilde{I}(X_S;Y_S) \geq \mu$, then the confidence of (X_1, Y_1) in $\mathcal{D}_{SEQ}$ has a lower bound:*

$$conf(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \sigma \cdot \lambda_1^{\frac{1-\mu}{\sigma}} \cdot \left(\frac{n_x - 1}{1 - \sigma}\right)^{\frac{\lambda_2}{\sigma}} \tag{A.11}$$

where: n_x is the number of symbols in Σ_X, λ_1 is the minimum support of $X_i \in X_S$, and λ_2 is the support of $(X_i, Y_j) \in (X_S, Y_S)$ such that $p(X_i|Y_j)$ is minimal, $\forall(i \neq 1$ & $j \neq 1)$.

Proof *(Sketch - Detailed proof in [19]). From Eq.* (A.10)*, we have:*

$$\widetilde{I}(X_S; Y_S) = 1 - \frac{H(X_S|Y_S)}{H(X_S)} \geq \mu \tag{A.12}$$

$$\Rightarrow \frac{H(X_S|Y_S)}{H(X_S)} = \frac{p(X_1, Y_1) \cdot \log p(X_1|Y_1)}{\sum_i p(X_i) \cdot \log p(X_i)} + \frac{\sum_{i\neq 1 \& j \neq 1} p(X_i, Y_j) \cdot \log \frac{p(X_i, Y_j)}{p(Y_j)}}{\sum_i p(X_i) \cdot \log p(X_i)} \leq 1 - \mu \tag{A.13}$$

Let $\lambda_1 = p(X_k)$ *such that* $p(X_k) = \min\{p(X_i)\}\forall i$*, and* $\lambda_2 = p(X_m, Y_n)$ *such that* $p(X_m|Y_n) = \min\{p(X_i|Y_j)\}, \forall(i \neq 1 \& j \neq 1)$*. Then, by applying the min-max inequality theorem for the sum of ratio [5] to the numerator of Eq.* (A.13)*, we obtain:*

$$\frac{H(X_S|Y_S)}{H(X_S)} \geq \frac{p(X_1,Y_1) \cdot \log p(X_1|Y_1) + \lambda_2 \cdot \log \frac{1-p(X_1,Y_1)}{n_x - p(Y_1)}}{\log \lambda_1}$$
$$\geq \frac{\sigma \cdot \log \frac{p(X_1,Y_1)}{p(Y_1)} + \lambda_2 \cdot \log \frac{1-\sigma}{n_x - 1}}{\log \lambda_1} \tag{A.14}$$

Next, assume that $supp(Y_1)_{\mathcal{D}_{SYB}} \geq supp(X_1)_{\mathcal{D}_{SYB}}$*. From Eqs.* (A.13), (A.14)*, the confidence lower bound of* (X_1, Y_1) *in* $\mathcal{D}_{SYB}$ *is derived as:*

$$conf(X_1, Y_1)_{\mathcal{D}_{SYB}} = \frac{supp(X_1, Y_1)_{\mathcal{D}_{SYB}}}{supp(Y_1)_{\mathcal{D}_{SYB}}} \geq \lambda_1^{\frac{1-\mu}{\sigma}} \cdot \left(\frac{n_x - 1}{1-\sigma}\right)^{\frac{\lambda_2}{\sigma}} \tag{A.15}$$

Since:

$$conf(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \sigma \cdot conf(X_1, Y_1)_{\mathcal{D}_{SYB}} \tag{A.16}$$

It follows that:

$$conf(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \sigma \cdot \lambda_1^{\frac{1-\mu}{\sigma}} \cdot \left(\frac{n_x - 1}{1-\sigma}\right)^{\frac{\lambda_2}{\sigma}} \tag{A.17}$$

Interpretation of the confidence lower bound: Theorem 1 says that, given an MI threshold μ, if the two symbolic series X_S and Y_S are correlated, then the confidence of a frequent event pair in (X_S,Y_S) is at least the lower bound in Eq. (A.11). Combining Theorem 1 and Lemma 3, we can conclude that given (X_S,Y_S), if its event pair has a confidence less than the lower bound, then any pattern P formed by that event pair also has a confidence less than that lower bound. This allows to approximate HTPGM (discussed in Section A.5.3).

Shape of the confidence lower bound

To understand how the confidence changes w.r.t. the support σ and the MI μ, we analyze its shape, shown in Fig. A.5 (σ and μ vary between 0 and 1). First, it can be seen that the confidence lower bound has a *direct relationship* with σ and μ (one increases if the other increases and vice versa). While the *direct relationship* between the confidence and σ can be explained using Eq. (B.5), it is interesting to observe the connection between μ and the confidence. As the MI represents the correlation between two symbolic series, the larger the value of μ, the more correlated the two series. Thus, when the confidence increases together with μ, it implies that patterns with high confidence are more likely to be found in highly correlated series, and vice versa.

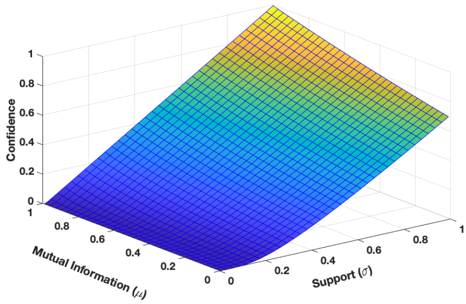


Fig. A.5: Shape of the lower bound

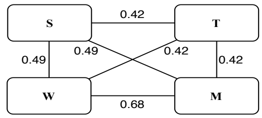


Fig. A.6: Correlation graph

Fig. A.5 also shows that, when σ is low, e.g., $\sigma < 0.1$, we obtain a very low value of the confidence lower bound regardless of μ value. This implies that the confidence is less sensitive to μ when the support is low. The opposite is obtained when the support is high, e.g., $\sigma > 0.1$, where we see a visible increase of the confidence lower bound as μ increases. This indicates that the "insensitive" area of the lower bound (when $\sigma \leq 0.1$) is less accurate than the "sensitive" area ($\sigma > 0.1$) when performing the approximate mining, as we will discuss in Section A.6.

A.5.3 Using the Bound to Approximate HTPGM

Correlation graph

Using Theorem 1, we propose to approximate HTPGM by performing the mining only on *the set of correlated symbolic series* $\mathcal{X}_C \subseteq \mathcal{X}$. We first define the correlation graph.

Definition 5.5 (Correlation graph) A *correlation* graph is an *undirected* graph $G_C = (V, E)$ where V is the set of vertices, and E is the set of edges. Each vertex $v \in V$ represents one symbolic series $X_S \in \mathcal{X}_C$. There is an edge e_{uv} between a vertex u containing X_S, and a vertex v containing Y_S iff $\widetilde{I}(X_S; Y_S) \geq \mu \vee \widetilde{I}(Y_S; X_S) \geq \mu$.

Fig. A.6 shows an example of the correlation graph G_C built from $\mathcal{D}_{\text{SYB}}$ in Table A.1. Here, each node corresponds to one electrical appliance. There is an edge between two nodes if their NMI is at least μ. The number on each edge is the NMI between two nodes.

Constructing the correlation graph: Given a symbolic database $\mathcal{D}_{\text{SYB}}$, the correlation graph G_C can easily be constructed by computing the NMI for each symbolic series pair, and comparing their NMI against the threshold μ. A symbolic series pair is included in G_C if their NMI is at least μ, and vice versa.

Setting the value of μ: While NMI can easily be computed using Eq. (A.10), it is not trivial how to set the value for μ. Here, we propose a method to determine μ using the lower bound in Eq. (A.11).

Recall that HTPGM relies on two user-defined parameters, the support threshold σ and the confidence threshold δ, to look for frequent temporal patterns. Based on the confidence lower bound in Theorem 1, we can derive μ using σ and δ as the following.

Corollary 1.1 *The confidence of an event pair $(X_1, Y_1) \in (X_S, Y_S)$ in $\mathcal{D}_{SEQ}$ is at least δ if $\widetilde{I}(X_S; Y_S)$ is at least μ, where:*

$$\mu \geq 1 - \sigma \cdot \log_{\lambda_1}\left(\frac{\delta}{\sigma} \cdot \left(\frac{1-\sigma}{n_x - 1}\right)^{\frac{\lambda_2}{\sigma}}\right) \tag{A.18}$$

Note that μ in Eq. (A.18) only ensures that the event pair (X_1, Y_1) has a minimum confidence of δ. Thus, given (X_S, Y_S), μ has to be computed for each event pair in (X_S, Y_S). The final chosen μ value to be compared against $\widetilde{I}(X_S; Y_S)$ is the minimum μ value among all the event pairs in (X_S, Y_S).

Algorithm 8: Approximate HTPGM using MI

Input: A set of time series $\mathcal{X}$, an MI threshold μ, support threshold σ, confidence threshold δ
Output: The set of frequent temporal patterns P

1: convert $\mathcal{X}$ to $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$;
2: scan $\mathcal{D}_{\text{SYB}}$ to compute the probability of each event and event pair;
3: **foreach** *pair of symbolic time series* $(X_S, Y_S) \in \mathcal{D}_{SYB}$ **do**
4:     compute $\widetilde{I}(X_S; Y_S)$ and $\widetilde{I}(Y_S; X_S)$;
5:     compute μ;
6:     **if** $\widetilde{I}(X_S;Y_S) \geq \mu \vee \widetilde{I}(Y_S; X_S) \geq \mu$ **then**
7:         insert X_S and Y_S into $\mathcal{X}_C$;
8:         create an edge between X_S and Y_S in G_C;
9: **foreach** $X_S \in \mathcal{X}_C$ **do**
10:     mine frequent single events from X_S;
11: **foreach** *event pair* (E_i, E_j) *in* L_1 **do**
12:     **if** *there is an edge between* X_S *and* Y_S *in* G_C **then**
13:         mine frequent patterns for (E_i, E_j);
14: **if** $k \geq 3$ **then**
15:     perform HTPGM using L_1 and L_2;

Approximate HTPGM using the correlation graph

Using the correlation graph G_C, the approximate HTPGM is described in Algorithm 8. First, $\mathcal{D}_{\text{SYB}}$ is scanned once to compute the probability of each single event and pair of events (line 2). Next, NMI and μ are computed for each pair of symbolic series (X_S, Y_S) in $\mathcal{D}_{\text{SYB}}$ (lines 4-5). Then, only pairs whose $\widetilde{I}(X_S; Y_S)$ or $\widetilde{I}(Y_S; X_S)$ is at least μ are inserted into $\mathcal{X}_C$, and an edge between X_S and Y_S is created (lines 6-8). Next, at L_1 of HPG, only the correlated symbolic series in $\mathcal{X}_C$ are used to mine frequent single events (lines 9-10). At L_2, G_C is used to filter 2-event combinations: for each event pair (E_i, E_j), we check whether there is an edge between their corresponding symbolic series in G_C. If so, we proceed by verifying the support and confidence of (E_i, E_j) as in the exact HTPGM (lines 11-13). Otherwise, (E_i, E_j) is eliminated from the mining of L_2. From level L_k ($k \geq 3$) onwards, the exact HTPGM is used (lines 14-15).

Complexity analysis

To compute NMI and μ, we only have to scan $\mathcal{D}_{\text{SYB}}$ once to calculate the probability for each single event and pair of events. Thus, the cost of NMI and μ computations is $|\mathcal{D}_{\text{SYB}}|$. On the other hand, the complexity of the exact HTPGM at L_1 and L_2 are $O(m^2 i^2\ |\mathcal{D}_{\text{SEQ}}|^2) + O(m \cdot |\mathcal{D}_{\text{SEQ}}|)$ (Section A.4.4). Thus, the approximate HTPGM is significantly faster than HTPGM.

Table A.4: Characteristics of the Datasets

	NIST	UKDALE	DataPort	Smart City	ASL
# sequences	1460	1520	1460	1216	1908
# variables	49	24	21	26	25
# distinct events	98	48	42	130	173
# instances/seq.	55	190	49	162	20

A.6 Experimental Evaluation

We evaluate HTPGM (both exact and approximate), using real-world datasets from three application domains: smart energy, smart city, and sign language. Due to space limitations, we only present here the most important results, and discuss other findings in [19].

A.6.1 Experimental Setup

Datasets: We use 3 *smart energy* datasets, NIST [14], UKDALE [26], and DataPort [13], all of which measure the energy/power consumption of electrical appliances in residential households. For the *smart city*, we use weather and vehicle collision data obtained from NYC Open Data Portal [9]. For *sign language*, we use the American Sign Language (ASL) datasets [33] containing annotated video sequences of different ASL signs and gestures. Table A.4 summarizes their characteristics.

Baseline methods: Our exact method is referred to as E-HTPGM, and the approximate one as A-HTPGM. We use 4 baselines (described in Section A.2): Z-Miner [28], TPMiner [8], IEMiner [36], and H-DFS [35]. Since E-HTPGM and the baselines provide the same exact solutions, we use the baselines only for the quantitative evaluation, and compare only E-HTPGM and A-HTPGM qualitatively.

Infrastructure: The experiments are run on virtual machines (VM) with AMD EPYC Processor 32 cores (2GHz) CPU, 256 GB main memory, and 1 TB storage. For scalability evaluation, we use VMs with 512 GB main memory.

Parameters: Table A.5 lists the parameters and their values used in our experiments.

Table A.5: Parameters and values

Params	Values
Support σ	User-defined: σ = 0.5%, 1%, 10%, 20%, ...
Confidence δ	User-defined: δ = 0.5%, 1%, 10%, 20%, ...
Overlapping duration t_{ov}	User-defined: t_{ov} (hours) = 0, 1, 2, 3 (NIST, UKDALE, DataPort, and Smart City) t_{ov} (frames) = 0, 150, 300, 450 (ASL)
Tolerance buffer ϵ	User-defined: ϵ (mins) = 0, 1, 2, 3 (NIST, UKDALE, DataPort) ϵ (mins) = 0, 5, 10, 15 (Smart City) ϵ (frames) = 0, 30, 45, 60 (ASL)

A.6.2 Qualitative Evaluation

Our goal is to make sense and learn insights from extracted patterns. Table A.6 lists some interesting patterns found in the datasets.

Patterns P1 - P9 are extracted from the energy datasets, showing how the residents interact with electrical devices in their houses. Patterns P10 - P15 extracted from the *smart city* datasets, while patterns P16 - P19 are from the ASL dataset.

A.6.3 Quantitative Evaluation

Baselines comparison on real world datasets

We compare E-HTPGM and A-HTPGM with the baselines in terms of the runtime and memory usage. Tables A.7 and A.8 show the experimental results on the energy and the smart city datasets. The quantitative results of other datasets are reported in the full paper [19].

As shown in Table A.7, A-HTPGM achieves the best runtime among all methods, and E-HTPGM has better runtime than the baselines. On the tested datasets, the range and average speedups of A-HTPGM compared to other methods are: [1.21-4.82] and 2.31 (E-HTPGM), [2.52-25.86] and 7.85 (Z-Miner), [7.43-69.68] and 21.65 (TPMiner), [8.61-188.16] and 40.75 (IEMiner), and [14.50-332.98] and 61.36 (H-DFS). The speedups of E-HTPGM compared to the base-

Table A.6: Summary of Interesting Patterns

Patterns	Supp. (%)	Conf. (%)
(P1) ([05:58, 08:24] First Floor Lights) ≽ ([05:58, 06:59] Upstairs Bathroom Lights) ≽ ([05:59, 06:06] Microwave)	20	30
(P2) ([06:00, 07:01] Upstairs Bathroom Lights) ≽ ([06:40, 06:46] Upstairs Bathroom Plugs)	30	55
(P3) ([18:00, 18:30] Lights Dining Room) → ([18:31, 20:16] Children Room Plugs) ≬ ([19:00, 22:31] Lights Living Room)	20	20
(P4) ([15:59, 16:05] Hallway Lights) → ([17:58, 18:29] Kitchen Lights ≽ ([18:00, 18:18] Plug In Kitchen) ≽ ([18:08, 18:15] Microwave)	20	25
(P5) ([06:02, 06:19] Kitchen Lights) → ([06:05, 06:12] Microwave) ≬ ([06:09, 06:11] Kettle)	20	35
(P6) ([18:10,18:15] Kitchen App) → ([18:15,19:00] Lights Plugs) ≽ ([18:20,18:25] Microwave) → ([18:25,18:55] Cooktop)	25	50
(P7) ([16:45, 17:30] Washer) → ([17:40,18:55] Dryer) → ([19:05, 20:10] Dining Room Lights) ≽ ([19:10, 19:30] Cooktop)	10	30
(P8) ([06:10, 07:00] Kitchen Lights) ≽ ([06:10, 06:15] Kettle) → ([06:30, 06:40] Toaster) → ([06:45, 06:48] Microwave)	25	40
(P9) ([18:00, 18:25] Kitchen Lights) ≽ ([18:00, 18:05] Kettle) → ([18:05, 18:10] Microwave) → ([19:35, 20:50] Washer)	20	40
(P10) Heavy Rain ≽ Unclear Visibility ≽ Overcast Cloudiness → High Motorist Injury	5	30
(P11) Extremely Unclear Visibility ≽ High Snow ≽ High Motorist Injury	3	45
(P12) Very Strong Wind → High Motorist Injury	5	40
(P13) Frost Temperature → Medium Cyclist Injury	5	20
(P14) Strong Wind → High Pedestrian Killed	4	30
(P15) Strong Wind → High Motorist Killed	4	10
(P16) [2.12 seconds] Negation ≽ [0.61 seconds] Left Head Tilt-side ≽ [0.27 seconds] Lowered Eye-brows	5	10
(P17) [1.53 seconds] Wh-question ≽ [0.36 seconds] Lowered Eye-brows → [0.05 seconds] Blinking Eye-aperture	10	15
(P18) [1.69 seconds] Wh-question ≽ [0.35 seconds] Right Head Tilt-side ≽ [0.27 seconds] Lowered Eye-brows	5	5
(P19) [1.92 seconds] Wh-question ≽ [0.82 seconds] Squint Eye-aperture → [0.13 seconds] Forward Body Lean	1	5

Table A.7: Runtime Comparison (seconds)

Supp. (%)	Methods	Conf. (%)					
		NIST			**Smart City**		
		20	**50**	**80**	**20**	**50**	**80**
20	H-DFS	73864.39	8967.15	1538.49	2516.64	223.47	10.27
	IEMiner	69440.62	7965.41	622.79	1419.51	130.80	8.59
	TPMiner	31445.99	7702.02	533.95	418.25	118.89	6.66
	Z-Miner	19063.24	2409.22	160.19	194.86	33.60	4.85
	E-HTPGM	3968.19	672.45	109.08	86.36	16.89	2.85
	A-HTPGM	**1174.28**	**262.56**	**55.48**	**37.54**	**8.46**	**0.70**
50	H-DFS	6268.88	5170.72	1296.01	453.47	88.32	9.82
	IEMiner	5497.78	4581.10	564.48	300.80	73.81	7.81
	TPMiner	3483.02	2976.37	512.23	118.89	37.54	6.14
	Z-Miner	2971.26	2061.75	149.81	92.22	21.05	1.70
	E-HTPGM	573.50	365.30	80.19	23.84	8.76	0.82
	A-HTPGM	**309.37**	**207.46**	**47.86**	**3.71**	**1.69**	**0.68**
80	H-DFS	1057.21	867.73	761.61	13.27	8.39	4.41
	IEMiner	954.99	460.93	355.19	9.59	5.47	4.37
	TPMiner	899.25	412.01	306.91	6.66	3.44	3.37
	Z-Miner	241.87	170.64	139.74	3.19	1.23	1.19
	E-HTPGM	143.66	93.55	63.51	1.47	0.58	0.47
	A-HTPGM	**63.71**	**51.35**	**41.26**	**0.51**	**0.35**	**0.21**

lines are: [1.47-5.64] and 3.19 on average (Z-Miner), [3.59-30.97] and 9.08 on avg. (TPMiner), [4.63-78.41] and 15.86 on avg. (IEMiner), and [5.54-118.21] and 23.37 on avg. (H-DFS). Note that the time to compute MI and μ for the

Table A.8: Memory Usage Comparison (MB)

Supp. (%)	Methods	Conf. (%)					
		NIST			Smart City		
		20	**50**	**80**	**20**	**50**	**80**
20	H-DFS	11976.25	4382.12	1143.17	1293.28	470.49	107.89
	IEMiner	7241.96	1613.96	705.51	1197.74	460.52	65.92
	TPMiner	6558.48	1216.96	700.75	1002.82	254.26	61.23
	Z-Miner	91875.84	17642.01	5241.76	1690.75	602.08	149.77
	E-HTPGM	1748.93	732.39	571.48	510.30	140.76	40.48
	A-HTPGM	**875.29**	**674.44**	**562.77**	**161.63**	**85.95**	**32.56**
50	H-DFS	3744.73	3173.70	940.48	1040.56	412.14	92.81
	IEMiner	1455.14	1155.31	663.52	870.64	353.18	60.87
	TPMiner	1109.89	909.38	600.73	660.66	150.68	58.98
	Z-Miner	16278.14	10277.83	2153.03	1195.59	505.16	117.64
	E-HTPGM	621.77	424.36	345.94	139.50	119.08	34.69
	A-HTPGM	**319.59**	**227.06**	**186.70**	**83.55**	**62.16**	**29.26**
80	H-DFS	877.13	726.56	641.43	249.78	139.59	63.65
	IEMiner	657.46	609.25	549.25	149.45	119.83	59.59
	TPMiner	575.98	512.86	475.22	119.59	69.91	58.63
	Z-Miner	1934.23	1735.01	1613.09	263.27	153.16	93.23
	E-HTPGM	313.99	261.78	153.26	52.93	36.96	29.89
	A-HTPGM	**257.32**	**187.29**	**106.87**	**35.75**	**31.74**	**25.28**

NIST and the smart city datasets in Table A.7 are 28.01 and 20.82 seconds, respectively.

Moreover, A-HTPGM is most efficient, i.e., achieves highest speedup and memory saving, when the support threshold is low, e.g., $\sigma = 20\%$. This is because typical datasets often contain many patterns with very low support and confidence. Thus, using A-HTPGM to prune uncorrelated series early helps save computational time and resources. However, the speedup comes at the cost of a small loss in accuracy (discussed in Sections A.6.3 and A.6.3).

In terms of memory consumption, as shown in Table A.8, A-HTPGM is the most efficient method, while E-HTPGM is more efficient than the baselines. The range and the average memory consumption of A-HTPGM compared to other methods are: [1.1-3.2] and 1.6 (E-HTPGM), [3.7-105.1] and 19.1 (Z-Miner), [1.3-7.9] and 3.4 (TPMiner), [1.4-10.4] and 4.5 (IEMiner), and [2.1-13.9] and 6.7 (H-DFS). The memory usage of E-HTPGM compared to the baselines are: [2.9-52.5] and 11.4 on avg. (Z-Miner), [1.2-4.7] and 2.1 on average (TPMiner), [1.3-6.2] and 2.7 on avg. (IEMiner), and [1.9-7.5] and 4.1 on avg. (H-DFS).

Table A.9: Building $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$

Dataset	$\mathcal{D}_{\text{SYB}}$		$\mathcal{D}_{\text{SEQ}}$	
	Time (sec)	**Storage (MB)**	**Time (sec)**	**Storage (MB)**
NIST	24.92	10.3	21.60	4.2
UKDALE	19.88	24.1	8.95	11.4
DataPort	11.32	17.7	20.62	2.9
Smart City	17.41	21.9	13.76	7.8
ASL	14.47	5.8	10.05	1.5

Finally, in Table A.9, we provide the pre-processing times to convert the raw time series to $\mathcal{D}_{\text{SYB}}$, and $\mathcal{D}_{\text{SYB}}$ to $\mathcal{D}_{\text{SEQ}}$. We also report the sizes of $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$ stored on disk. We see that while the storage costs for $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$ are small, the pre-processing times are 10-25 seconds. This is a one-time cost which can be reused for many mining runs, making it negligible in all non-trivial cases.

Scalability evaluation on synthetic datasets

As discussed in Section A.4, the complexity of HTPGM is driven by two main factors: (1) the number of temporal sequences, and (2) the number of time series. The evaluation on real-world datasets has shown that E-HTPGM and A-HTPGM outperform the baselines significantly in both runtimes and memory usage. However, to further assess the scalability, we scale these two factors on synthetic datasets. Specifically, starting from the real-world datasets, we generate 10 times more sequences, and create up to 1000 synthetic time series. We evaluate the scalability using two configurations: varying the number of sequences, and varying the number of time series.

Figs. A.7 and A.8 show the runtimes of A-HTPGM, E-HTPGM and the baselines when the number of sequences changes (y-axis is in log scale). The range and average speedups of A-HTPGM w.r.t. other methods are: [1.5-3.7] and 2.5 (E-HTPGM), [3.1-13.6] and 8.1 (Z-Miner), [5.1-31.2] and 16.8 (TPMiner), [6.4-45.8] and 24.9 (IEMiner), and [9.4-59.1] and 31.8 (H-DFS). In particular, A-HTPGM obtains even higher speedup for more sequences. Similarly, the range and average speedups of E-HTPGM are: [1.6-5.3] and 3.2 (Z-Miner), [2.2-12.1] and 6.7 (TPMiner), [3.5-17.4] and 10.1 (IEMiner), and [4.9-22.8] and 12.9 (H-DFS).

Figs. A.9 and A.10 compare the runtimes of A-HTPGM with other methods when changing the number of time series (y-axis is in log scale). It is seen that, A-HTPGM achieves even higher speedup with more time series. The range and average speedups of A-HTPGM are: [2.1-4.9] and 2.9 (E-HTPGM), [2.9-10.4] and 6.8 (Z-Miner), [3.6-21.5] and 12.8 (TPMiner), [4.7-30.2] and 18.1 (IEMiner),

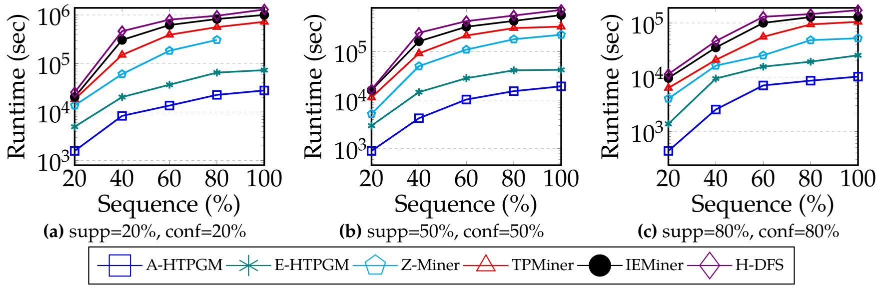


Fig. A.7: Varying % of sequences on NIST

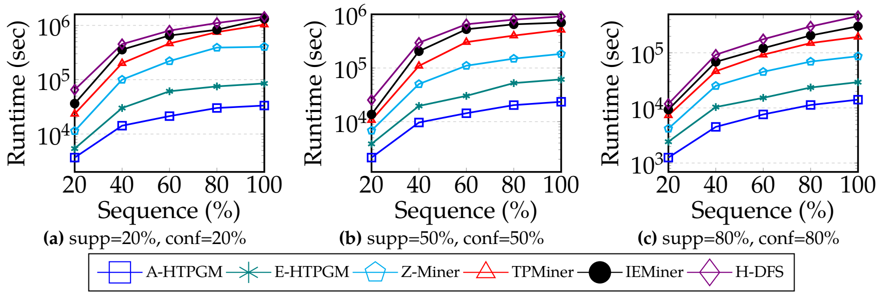


Fig. A.8: Varying % of sequences on Smart City

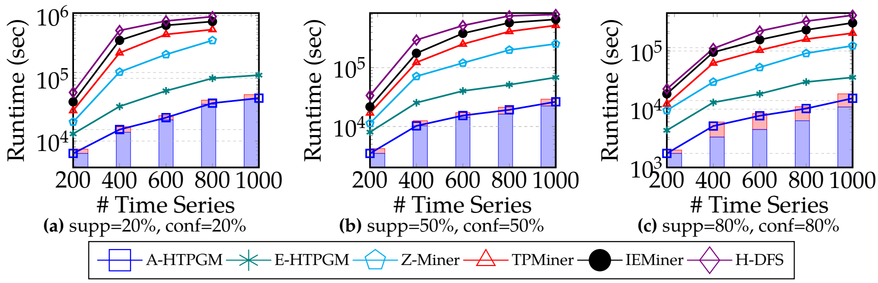


Fig. A.9: Varying # of time series on NIST

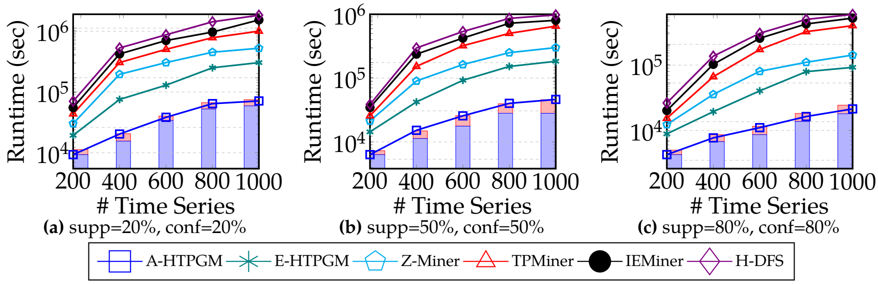


Fig. A.10: Varying # of time series on Smart City

and [6.1-39.6] and 23.2 (H-DFS), and of E-HTPGM are: [1.4-4.1] and 2.4 (Z-

Miner), [1.7-8.1] and 4.4 (TPMiner), [2.3-11.3] and 6.2 (IEMiner), and [2.7-16.3] and 8.1 (H-DFS).

In Figs. A.9 and A.10, to illustrate the computation time of MI and μ, we add an additional bar chart for A-HTPGM. Each bar represents the runtime of A-HTPGM with two separate components: the time to compute MI and μ (top red), and the mining time (bottom blue). However, note that for each dataset, we only need to compute MI and μ once (the computed values are used across the mining process with different support and confidence thresholds). Thus, the times to compute MI and μ, for example, in Figs. A.9a, A.9b, and A.9c, are added only for comparison and are not all actually used.

Moreover, most baselines fail for the larger configurations in the scalability study, e.g., Z-Miner on the NIST dataset when σ=δ=20% (Fig. A.7a), and Z-Miner, TPMiner, IEMiner and H-DFS when the number of time series grows to 1000 (Fig. A.9a). The scalability test shows that A-HTPGM and E-HTPGM can scale well on big datasets, both vertically (many sequences) and horizontally (many time series), unlike the baselines.

Table A.10: Pruned Time Series and Events from A-HTPGM

# Attr.	NIST						Smart City					
	# Pruned Time Series			# Pruned Events			# Pruned Time Series			# Pruned Events		
	20-20	20-50	20-80	20-20	20-50	20-80	20-20	20-50	20-80	20-20	20-50	20-80
200	23	55	83	46	110	166	11	27	43	27	87	135
400	37	101	157	74	202	314	17	49	81	57	197	309
600	45	141	225	90	282	450	32	80	128	96	316	492
800	54	182	294	108	364	588	41	105	169	129	429	669
1000	83	243	383	166	486	766	51	131	211	163	543	847

Furthermore, the number of time series and events pruned by A-HTPGM in the scalability test are provided in Table A.10. Here, we can see that high confidence threshold leads to more time series (events) to be pruned. This is because confidence has a direct relationship with MI, therefore, high confidence results in higher μ, and thus, more pruned time series.

Evaluation of the pruning techniques in E-HTPGM

We compare different versions of E-HTPGM to understand how effective the pruning techniques are: (1) NoPrune: E-HTPGM with no pruning, (2) Apriori: E-HTPGM with Apriori-based pruning (Lemmas 2, 3), (3) Trans: E-HTPGM with transitivity-based pruning (Lemmas 4, 5, 6, 7), and (4) All: E-HTPGM applied both pruning techniques.

We use 3 different configurations that vary: the number of sequences, the confidence, and the support. Figs. A.11, A.12 show the results (the y-axis is in log scale). It can be seen that (All)-E-HTPGM achieves the best performance among all versions. Its speedup w.r.t. (NoPrune)-E-HTPGM ranges from 5 up to 60 depending on the configurations, showing that the proposed prunings are very effective in improving E-HTPGM performance. Furthermore,

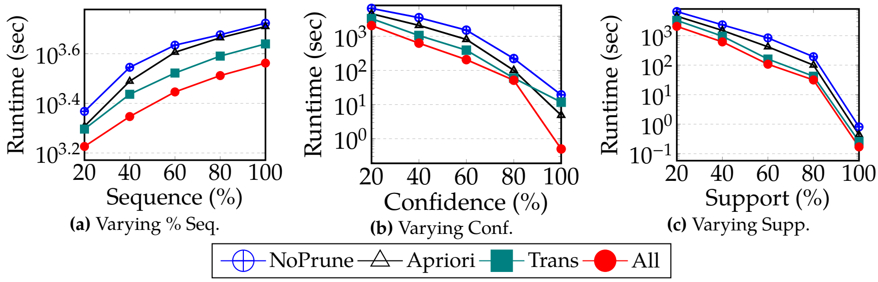


Fig. A.11: Runtimes of E-HTPGM on NIST

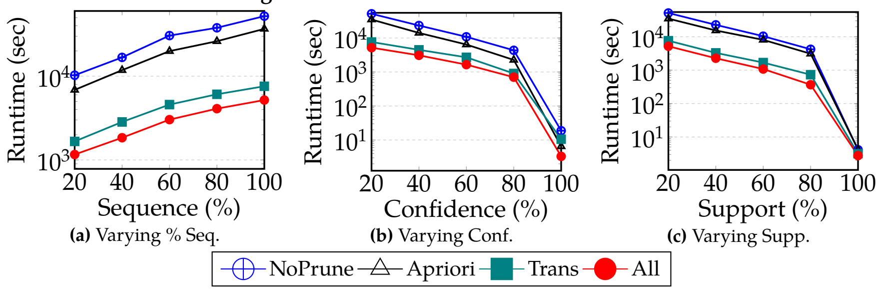


Fig. A.12: Runtimes of E-HTPGM on Smart City

(Trans)-E-HTPGM delivers larger speedup than (Apriori)-E-HTPGM. The average speedup is from 8 to 20 for (Apriori)-E-HTPGM. However, applying both always yields better speedup than applying either of them.

Evaluation of A-HTPGM

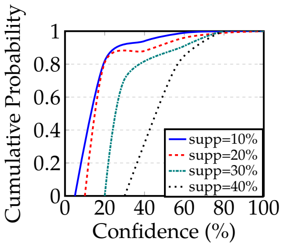
Table A.11: The Accuracy of A-HTPGM (%)

Supp. (%)	Conf. (%)							
	NIST				Smart City			
	10	20	50	80	10	20	50	80
10	87	89	91	94	78	83	98	100
20	96	89	91	94	83	83	98	100
50	100	100	96	94	99	99	98	100
80	100	100	100	100	100	100	100	100

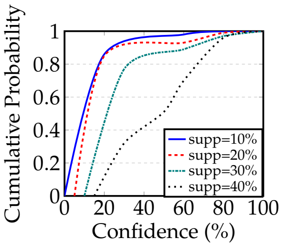
We proceed to evaluate the accuracy of A-HTPGM and the quality of patterns pruned by A-HTPGM.

To evaluate the accuracy, we compare the patterns extracted by A-HTPGM and E-HTPGM. Table A.11 shows the accuracies of A-HTPGM for different supports and confidences. It is seen that, A-HTPGM obtains high accuracy ($\geq$ 71%) when σ and δ are low, e.g., $\sigma = \delta = 10\%$, and very high accuracy ($\geq$ 95%) when σ and δ are high, e.g., $\sigma = \delta = 50\%$.

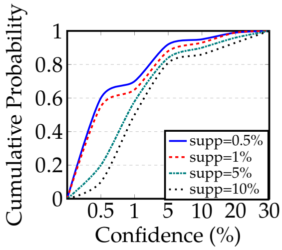
Next, we analyze the quality of patterns pruned by A-HTPGM. These patterns are extracted from the uncorrelated time series. Fig. A.13 shows the cumulative distribution of the confidences of the pruned patterns. It is



(a) NIST



(b) Smart City



(c) ASL

Fig. A.13: Cumulative probability of pruned patterns

seen that most of these patterns have low confidences, and can thus safely be pruned. For NIST and Smart City datasets, 80% of pruned patterns have confidences less than 20% when the support is 10% and 20%, and 70% of pruned patterns have confidences less than 30% when the support is 30%. For the ASL dataset, 80% of pruned patterns have confidences less than 5%.

Other experiments: We analyze the effects of the tolerance buffer ϵ, and the overlapping duration t_{ov} to the quality of extracted patterns. The analysis can be seen in the full paper [19].

A.7 Conclusion and Future Work

This paper presents our comprehensive Frequent Temporal Pattern Mining from Time Series (FTPMfTS) solution that offers: (1) an end-to-end FTPMfTS process to mine frequent temporal patterns from time series, (2) an efficient and exact Hierarchical Temporal Pattern Graph Mining (E-HTPGM) algorithm that employs efficient data structures and multiple pruning techniques to achieve fast mining, and (3) an approximate A-HTPGM that uses mutual information to prune unpromising time series, allows HTPGM to scale on big datasets. Extensive experiments conducted on real world and synthetic datasets show that both A-HTPGM and E-HTPGM outperform the baselines, consume less memory, and scale well to big datasets. Compared to the baselines, the approximate A-HTPGM delivers an order of magnitude speedup on large synthetic datasets and up to 2 orders of magnitude speedup on real-world datasets. In future work, we plan to extend HTPGM to prune at the event level to further improve its performance.

References

[1] Akiz Uddin Ahmed, Chowdhury Farhan Ahmed, Md Samiullah, Nahim Adnan, and Carson Kai-Sang Leung. 2016. Mining interesting patterns from uncertain databases. *Information Sciences* 354 (2016).

[2] James F Allen. 1983. Maintaining knowledge about temporal intervals. *Commun. ACM* 26 (1983).

[3] Iyad Batal, Dmitriy Fradkin, James Harrison, Fabian Moerchen, and Milos Hauskrecht. 2012. Mining recent temporal patterns for event detection in multivariate time series data. In *SIGKDD*.

[4] Iyad Batal, Hamed Valizadegan, Gregory F Cooper, and Milos Hauskrecht. 2013. A temporal pattern mining approach for classifying electronic health record data. *TIST* 4 (2013).

[5] Edwin F Beckenbach, Richard Bellman, and Richard Ernest Bellman. 1961. *An introduction to inequalities*. Technical Report. Mathematical Association of America Washington, DC.

[6] Julien Blanchard, Fabrice Guillet, Regis Gras, and Henri Briand. 2005. Using information-theoretic measures to assess association rule interestingness. In *ICDM'05*.

[7] Elizabeth A Campbell, Ellen J Bass, and Aaron J Masino. 2020. Temporal condition pattern mining in large, sparse electronic health record data: A case study in characterizing pediatric asthma. *JAMIA* 27 (2020).

[8] Yi-Cheng Chen, Wen-Chih Peng, and Suh-Yin Lee. 2015. Mining Temporal Patterns in Time Interval-Based Data. *TKDE* 27 (2015).

[9] New York City. 2019. NYC OpenData. https://opendata.cityofnewyork.us/

[10] Thomas M Cover and Joy A Thomas. 2012. *Elements of information theory*. John Wiley & Sons.

[11] Xue Cunjin, Song Wanjiao, Qin Lijuan, Dong Qing, and Wen Xiaoyang. 2015. A mutual-information-based mining method for marine abnormal association rules. *Computers & Geosciences* 76 (2015).

[12] Energi Data Portal. 2021. https://www.energidataservice.dk/tso-electricity/co2emis/

[13] Pecan Street Data. 2016. Pecan Street Dataport. https://www.pecanstreet.org/dataport/

[14] William Healy, Farhad Omar, Lisa Ng, Tania Ullah, William Payne, Brian Dougherty, and A Hunter Fanney. 2018. Net zero energy residential test facility instrumented data. https://pages.nist.gov/netzero/index.html/

[15] Nguyen Ho, Torben Bach Pedersen, Van Long Ho, and Mai Vu. 2020. Efficient Search for Multi-Scale Time Delay Correlations in Big Time Series Data. In 23rd International Conference on Extending Database Technology, EDBT 2020. 37–48.

[16] Nguyen Ho, Torben Bach Pedersen, Mai Vu, Christophe AN Biscio, et al. 2019. Efficient bottom-up discovery of multi-scale time series correlations using mutual information. In 2019 IEEE 35th International Conference on Data Engineering (ICDE). IEEE, 1734–1737.

[17] Nguyen Ho, Huy Vo, Mai Vu, and Torben Bach Pedersen. 2019. Amic: An adaptive information theoretic method to identify multi-scale temporal correlations in big time series data. IEEE Transactions on Big Data 7, 1 (2019), 128–146.

[18] Nguyen Ho, Huy Vo, and Mai Vu. An adaptive information-theoretic approach for identifying temporal correlations in big data sets. In 2016 IEEE International Conference on Big Data (Big Data), pp. 666-675. IEEE, 2016.

[19] Van Long Ho, Nguyen Ho, and Torben Bach Pedersen. 2021. Efficient Temporal Pattern Mining in Big Time Series Using Mutual Information. *arXiv preprint arXiv:2010.03653* (2020). https://arxiv.org/abs/2010.03653

[20] Po-shan Kam and Ada Wai-Chee Fu. 2000. Discovering temporal patterns for interval-based events. In *DaWak*.

[21] Yiping Ke, James Cheng, and Wilfred Ng. 2008. Correlated pattern mining in quantitative databases. *TODS* 33 (2008).

[22] Thi Thao Nguyen Ho, and Barbara Pernici. A data-value-driven adaptation framework for energy efficiency for data intensive applications in clouds. In 2015 IEEE conference on technologies for sustainability (SusTech), pp. 47-52. IEEE, 2015.

[23] Thi Thao Nguyen Ho, Marco Gribaudo, and Barbara Pernici. "Improving energy efficiency for transactional workloads in cloud environments." In Proceedings of the Eighth International Conference on Future Energy Systems, pp. 290-295. 2017.

[24] Thi Thao Nguyen Ho, Marco Gribaudo, and Barbara Pernici. "Characterizing energy per job in cloud applications." Electronics 5, no. 4 (2016): 90.

[25] Macro Gribaudo, Thi Thao Nguyen Ho, Barbara Pernici, and Giuseppe Serazzi. "Analysis of the influence of application deployment on energy consumption." In International Workshop on Energy Efficient Data Centers, pp. 87-101. Springer, Cham, 2014.

[26] Jack Kelly and William Knottenbelt. 2015. The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes. *Scientific Data* (2015).

[27] Young-Koo Lee, Won-Young Kim, Y Dora Cai, and Jiawei Han. 2003. CoMine: Efficient Mining of Correlated Patterns. In *ICDM*.

[28] Zed Lee, Tony Lindgren, and Panagiotis Papapetrou. 2020. Z-Miner: An Efficient Method for Mining Frequent Arrangements of Event Intervals. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 524–534.

[29] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. 2003. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*. 2–11.

[30] Vasileios Megalooikonomou, Qiang Wang, Guo Li, and Christos Faloutsos. 2005. A multiresolution symbolic representation of time series. In *21st International Conference on Data Engineering (ICDE'05)*. IEEE, 668–679.

[31] Fabian Mörchen. 2007. Unsupervised pattern mining from symbolic temporal data. *ACM SIGKDD Explorations Newsletter* 9, 1 (2007), 41–55.

[32] Robert Moskovitch and Yuval Shahar. 2015. Fast time intervals mining using the transitivity of temporal relations. *KAIS* 42 (2015).

[33] Carol Neidle, Augustine Opoku, Gregory Dimitriadis, and Dimitris Metaxas. 2018. NEW Shared & Interconnected ASL Resources: SignStream® 3 Software; DAI 2 for Web Access to Linguistically Annotated Video Corpora; and a Sign Bank. In *8th Workshop on the Representation and Processing of Sign Languages: Involving the Language Community, Miyazaki, Language Resources and Evaluation Conference 2018*.

[34] Edward R Omiecinski. 2003. Alternative interest measures for mining associations in databases. *IEEE Transactions on Knowledge and Data Engineering 15*, 1 (2003), 57–69.

[35] Panagiotis Papapetrou, George Kollios, Stan Sclaroff, and Dimitrios Gunopulos. 2009. Mining frequent arrangements of temporal intervals. *KAIS* 21 (2009).

[36] Dhaval Patel, Wynne Hsu, and Mong Li Lee. 2008. Mining relationships among interval-based events for classification. In *SIGMOD*.

[37] Amit Kumar Sharma and Dhaval Patel. 2018. Stipa: A memory efficient technique for interval pattern discovery. In *2018 IEEE International Conference on Big Data (Big Data)*. IEEE, 1767–1776.

[38] Jun-Zhe Wang, Yi-Cheng Chen, Wen-Yueh Shih, Lin Yang, Yu-Shao Liu, and Jiun-Long Huang. 2020. Mining High-utility Temporal Patterns on Time Interval–based Data. *ACM Transactions on Intelligent Systems and Technology (TIST)* 11, 4 (2020), 1–31.

[39] Shin-Yi Wu and Yen-Liang Chen. 2007. Mining nonambiguous temporal patterns for interval-based events. *TKDE* 19 (2007).

[40] YY Yao. 2003. Information-theoretic measures for knowledge discovery and data mining. In *Entropy measures, maximum entropy principle and emerging applications*. 115–136.

[41] Torp K., Andersen O., Thomsen C. (2020) Travel-Time Computation Based on GPS Data. In: Kutsche RD., Zimányi E. (eds) Big Data Management and Analytics. eBISS 2019. Lecture Notes in Business Information Processing, vol 390. Springer, Cham. https://doi.org/10.1007/978-3-030-61627-4_4.

Paper B

Efficient Generalized Temporal Pattern Mining in Big Time Series Using Mutual Information

Van Long Ho, Nguyen Ho, Torben Bach Pedersen, and
Panagiotis Papapetrou

The paper has been submitted in the
IEEE Transactions on Knowledge and Data Engineering (TKDE).

Abstract

Big time series are increasingly available from an ever wider range of IoT-enabled sensors deployed in various environments. Significant insights can be gained by mining temporal patterns from these time series. Temporal pattern mining (TPM) extends traditional pattern mining by adding event time intervals into extracted patterns, making them more expressive at the expense of increased time and space complexities. Besides frequent temporal patterns (FTPs), which occur frequently in the entire dataset, another useful type of temporal patterns are so-called rare temporal patterns (RTPs), which appear rarely but with high confidence. Mining rare temporal patterns yields additional challenges. For FTP mining, the temporal information and complex relations between events already create an exponential search space. For RTP mining, the support measure is set very low, leading to a further combinatorial explosion and potentially producing too many uninteresting patterns. Thus, there is a need for a generalized approach which can mine both frequent and rare temporal patterns. This paper presents our Generalized Temporal Pattern Mining from Time Series (GTPMfTS) approach with the following specific contributions: (1) The end-to-end GTPMfTS process taking time series as input and producing frequent/rare temporal patterns as output. (2) The efficient Generalized Temporal Pattern Mining (GTPM) algorithm mines frequent and rare temporal patterns using efficient data structures for fast retrieval of events and patterns during the mining process, and employs effective pruning techniques for significantly faster mining. (3) An approximate version of GTPM that uses mutual information, a measure of data correlation, to prune unpromising time series from the search space. (4) An extensive experimental evaluation of GTPM for rare temporal pattern mining (RTPM) and frequent temporal pattern mining (FTPM), showing that RTPM and FTPM signficantly outperform the baselines on runtime and memory consumption, and can scale to big datasets. The approximate RTPM is up to one order of magnitude, and the approximate FTPM up to two orders of magnitude, faster than the baselines, while retaining high accuracy.

B.1 Introduction

IoT-enabled sensors have enabled the collection of many big time series, e.g., from smart-meters, -plugs, and -appliances in households, weather stations, and GPS-enabled mobile devices. Extracting patterns from these time series can offer new domain insights for evidence-based decision making and optimization. As an example, consider Fig. B.1 that shows the electricity usage of a water boiler with a hot water tank collected by a 20 euro Wifi-enabled smart-plug, and accurate CO2 intensity (g/kWh) forecasts of local electricity, e.g., as supplied by the Danish Transmission System Operator [1]. From Fig. B.1, we can identify several useful patterns. First, the water boiler switches *On* once a day, for one hour between 6 and 7AM. This indicates that the resident takes

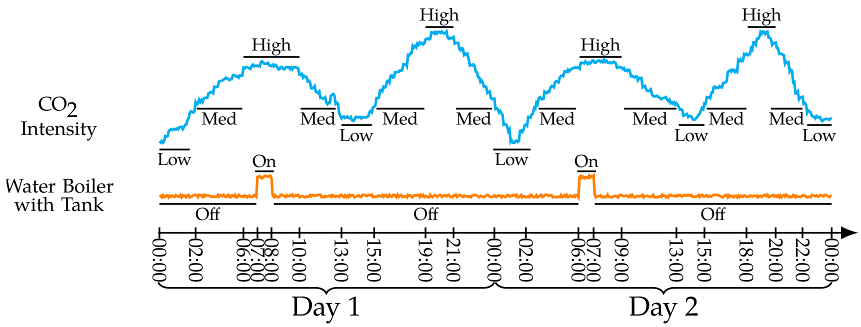


Fig. B.1: CO2 intensity and water boiler electricity usage

only one hot shower per day which starts between 5.30 and 6.30AM. Second, all water boiler *On* events are contained in CO2 *High* events, i.e., the periods when CO2 intensity is high. Third, between two consecutive *On* events of the boiler, there is a CO2 *Low* event lasting for one or more hours which occurs at most 4 hours before the hot shower (so water heated during that event will still be hot at 6AM). Pattern mining can be used to extract the relations between CO2 intensity and water boiler events. However, traditional sequential patterns only capture the sequential occurrence of events, e.g., that one boiler *On* event follows after another, but not that there is at least 23 hours between them; or that there is a CO2 *Low* event between the two boiler *On* events, but not when or for how long it lasts. In contrast, *temporal pattern mining* (TPM) adds temporal information into patterns, providing details on when certain relations between events happen, and for how long. For example, TPM expresses the above relations as: ([7:00 - 8:00, Day X] BoilerOn $\rightarrow$ [6:00 - 7:00, Day X+1] BoilerOn) (meaning BoilerOn is followed by BoilerOn the next day), ([6:00 - 10:00, Day X] HighCO2 $\succcurlyeq$ [7:00 - 8:00, Day X] BoilerOn) (meaning HighCO2 contains BoilerOn), and ([7:00 - 8:00, Day X] BoilerOn $\rightarrow$ [0:00 - 2:00, Day X+1] LowCO2 $\rightarrow$ [6:00 - 7:00, Day X+1] BoilerOn) (meaning there is a LowCO2 event between two BoilerOn events). As the resident is very keen on reducing her CO2 footprint, we can rely on the above temporal patterns to automatically (using the smart-plug) delay turning on the boiler until the CO2 intensity is low again, saving CO2 without any loss of comfort for the resident. In the smart city domain, temporal patterns extracted from vehicle GPS data [2] can reveal spatio-temporal correlations between traffic jams, advising drivers to take another route for their morning commute.

Finding frequent temporal patterns (FTPs) is useful; however, in many applications, some patterns appear rarely but are still very interesting and useful due to high confidence. We call such patterns *rare temporal patterns (RTPs)*. For example, considering smart city applications, a rare pattern could be: ([20:00, 22:00] Snow $\succcurlyeq$ [20:15, 21:15] HighWind $\rightarrow$ [21:20, 21:50] HighInjuryMotorist), which means that the coincidence of snow and strong winds leads to traffic accidents within an hour. This pattern occurs rarely but supports transportation coordinators in warning citizens about traffic accidents. In health care, identifying symptoms and relations among them supports health experts in diagnosing diseases in the early phases.

Challenges of mining frequent temporal patterns. Mining temporal patterns is much more expensive than mining sequential patterns. Not only does the temporal information add extra computation to the mining process, the complex relations between events also add an additional exponential factor $O(3^{h^2})$ to the $O(m^h)$ search space complexity (m is the number of events and h is the length of temporal patterns), yielding an overall complexity of $O(m^h 3^{h^2})$ (see Lemma 1 in Section A.4.4). Existing TPM methods [3–5] do not scale to big datasets, i.e., many time series and many sequences, and/or do not work directly on time series but only on pre-processed temporal events.

Challenges of mining rare temporal patterns. The support measure represents the frequency of a temporal pattern across the entire dataset. However, to find rare temporal patterns, the support has to be set very low, which causes a combinatorial explosion, potentially producing too many patterns that are uninteresting to the user. Existing work proposes solutions to mine rare itemsets [6–9] and rare sequential patterns [10–12]. However, they do not consider the temporal aspect of items/events. Thus, addressing the explosion of rare temporal patterns with high confidence is still an open problem.

Generalized temporal pattern mining. Since there are many joint challenges in mining frequent and rare temporal patterns, there is a need for a *generalized* approach that can mine both types of patterns efficiently.

Contributions. In this paper, we present our comprehensive *Generalized Temporal Pattern Mining from Time Series (GTPMfTS) approach* which solves the above challenges. The paper significantly extends a previous conference paper [13]. Our key contributions are: (1) We present *end-to-end GTPMfTS process* that receives time series as input, and produces frequent/rare temporal patterns as output. Within this process, a splitting strategy is proposed to convert time series into event sequences while ensuring the preservation of temporal patterns. (2) We propose the *efficient Generalized Temporal Pattern Mining (GTPM) algorithm* to mine both frequent and rare temporal patterns. The novelties of GTPM are: a) the use of an efficient data structure, Hierarchical Hash Tables, to enable fast retrieval of events and patterns during the mining process; and b) pruning techniques based on the Apriori principle and the transitivity property of temporal relations to enable faster mining. (3) Based on the information theory concept of mutual information, which measures the correlation among time series, we propose a novel *approximate version of GTPM* that prunes unpromising time series to significantly reduce the search space and can scale on big datasets, i.e., many time series and many sequences. (4) We perform extensive experiments on synthetic and real-world datasets for both rare temporal pattern mining (RTPM) and frequent temporal pattern mining (FTPM), showing that our RTPM and FTPM significantly outperform the baselines on both runtime and memory usage. Compared to the baselines, the approximate RTPM has up to one order of magnitude speedup, and the approximate FTPM up to two orders of magnitude speedup, while retaining high accuracy compared to the exact algorithms.

Compared to the the conference version [13], this paper generalizes the TPM problem, to mine both frequent and (the novel proposal of) rare temporal patterns. For FTPM, this paper uses Hierarchical Hash Tables to retrieve events and patterns quickly, a significant improvement over the Hierarchical Pattern Graph in the conference version [13]. Moreover, we now combine the lower bound of support and the lower bound of confidence from the conference version [13] for the approximate FTPM to further accelerate the mining. For RTPM, we introduce the first exact and approximate algorithms to mine rare temporal patterns. In the present paper, we further provide a set of new experiments to compare our algorithms with the baselines.

Paper Outline. The paper is structured as follows. Section 2 discusses the related work. Section 3 formulates the generalized temporal pattern mining problem. Section 4 describes the exact GTPM algorithm. Section 5 presents the approximate GTPM algorithm. Section 6 presents the experimental evaluation. Finally, Section 7 concludes and points to future work.

B.2 Related work

Temporal pattern mining: Compared to sequential pattern mining, TPM is rather a new research topic. One of the first papers in this area is of Kam et al. that uses a hierarchical representation to manage temporal relations [14], and based on that mines temporal patterns. However, the approach in [14] suffers from *ambiguity* when presenting temporal relations. For example, using the representation in [14], it is possible to have two temporal patterns that involve the same set of temporal events, for example, (((a overlaps b) before c) overlaps d), and ((a overlaps b) before (c contains d)). Thus, the same set of events can be mapped to different temporal patterns that are semantically different. Our GTPM avoids this ambiguity by defining a temporal pattern as a set of pairwise temporal relations between two events. In [15], Wu et al. develop TPrefix to mine temporal patterns from non-ambiguous temporal relations. However, TPrefix has several inherent limitations: it scans the database repeatedly, and the algorithm does not employ any pruning strategies to reduce the search space. In [16], Moskovitch et al. design a TPM algorithm using the transitivity property of temporal relations. They use this property to generate candidates by inferring new relations between events. In comparison, our GTPM uses the transitivity property for effective pruning. In [17], Iyad et al. propose a TPM framework to detect events in time series. However, their focus is to find irregularities in the data. In [18], Wang et al. propose a temporal pattern mining algorithm HUTPMiner to mine high-utility patterns. Different from our GTPM which uses *support* and *confidence* to measure the frequency of patterns, HUTPMiner uses *utility* to measure the importance or profit of an event/ pattern, thereby addresses an orthogonal problem. In [19], Amit et al. propose STIPA which uses a Hoeppner matrix representation to compress temporal patterns for memory savings. However, STIPA does not use any pruning/ optimization strategies and thus, despite the efficient use of memory, it cannot scale to large datasets, unlike our GTPM. Other work [20], [21] proposes TPM algorithms to classify health record data. However, these methods are very domain-specific, thus cannot generalize to other domains.

The state-of-the-art TPM methods that currently achieve the best performance are our baselines: H-DFS [5], TPMiner [3], IEMiner [4], and Z-Miner [22]. H-DFS is a hybrid algorithm that uses breadth-first and depth-first search strategies to mine frequent arrangements of temporal intervals. H-DFS uses a data structure called ID-List to transform event sequences into vertical representations, and temporal patterns are generated by merging the ID-Lists of different events. This means that H-DFS does not scale well when the number of time series increases. In [4], Patel et al. design a hierarchical lossless representation to model event relations, and propose IEMiner that uses Apriori-based optimizations to efficiently mine patterns from this new representation. In [3], Chen et al. propose TPMiner that uses endpoint and endtime representations to simplify the complex relations among events. Similar to [5], IEMiner and TPMiner do not scale to datasets with many time series. Z-Miner [22], proposed by Lee et al., is the most recent work addressing TPM. Z-Miner improves the mining efficiency over existing methods by employing two data structures: a hierarchical hash-based structure called Z-Table for time-efficient candidate generation and support count, and Z-Arrangement, a structure to efficiently store event intervals in temporal patterns for efficient memory consumption. Although using efficient data structures, Z-Miner neither employs the transitivity property of temporal relations nor mutual information for pruning. Thus, Z-Miner is less efficient than our exact and approximate GTPM in both runtimes and memory usage, and does not scale to large datasets with many sequences and many time series (see Section B.6). Our GTPM algorithm improves on these methods by: (1) using efficient data structures and applying pruning techniques based on the Apriori principle and the transitivity property of temporal relations to enable fast mining, (2) the approximate GTPM can handle datasets with many time series and sequences, and (3), providing an end-to-end GTPMfTS process to mine temporal patterns directly from time series, a feature that is not supported by the baselines.

Rare pattern mining: Finding rare patterns that occur infrequently in a given database has received some attention in recent years. Techniques to find rare patterns in time series, often called rare motifs, are proposed in [15, 23, 24]. However, since time series motifs are the repeated sub-sequences of the time series, rare motif discovery techniques cannot deal with temporal events, and thus, are insufficient for rare temporal pattern mining. A related approach concerns rare association rules [6–9, 25–30] that find rare associations between items in the database. However, all the mentioned work can only discover rare association rules built among itemsets, and cannot deal with temporal events and the complex temporal relations between them. Another research direction studies rare sequential patterns [10–12, 31–33]. However, rare sequential patterns only consider sequential occurrence between events, and therefore, cannot model other complex relations such as overlapping or containing between temporal events. To the best of our knowledge, there is currently no existing work that studies rare temporal pattern mining which mines rare occurrences of temporal patterns in a time series database.

Using correlations in TPM: Different correlation measures such as expected support [34], all-confidence [35], and mutual information (MI) [36–39] have been used to optimize the pattern mining process. However, these only support sequential patterns. To the best of our knowledge, our proposed approximate GTPM is the first that uses MI to optimize TPM.

B.3 Preliminaries

In this section, we introduce the notations and the main concepts that will be used throughout the paper.

B.3.1 Temporal Event of Time Series

Definition 3.1 (Time series) A *time series* $X = x_1, x_2, ..., x_n$ is a sequence of data values that measure the same phenomenon during an observation time period, and are chronologically ordered.

Definition 3.2 (Symbolic time series) A *symbolic time series* X_S of a time series X encodes the raw values of X into a sequence of symbols. The finite set of permitted symbols used to encode X is called the *symbol alphabet* Σ_X of X.

The symbolic time series X_S is obtained using a mapping function $f\colon X \rightarrow \Sigma_X$ that maps each value $x_i \in X$ to a symbol $\omega \in \Sigma_X$. For example, let X = 1.61, 1.21, 0.41, 0.0 be a time series representing the energy usage of an electrical device. Using the symbol alphabet Σ_X = {On, Off}, where On represents that the device is on and operating (e.g., $x_i \geq 0.5$), and Off that the device is off ($x_i < 0.5$), the symbolic representation of X is: X_S = On, On, Off, Off. The mapping function f can be defined using existing time series representation techniques such as SAX [40].

Table B.1: A Symbolic Database $\mathcal{D}_{\text{SYB}}$

Time	10:00	10:05	10:10	10:15	10:20	10:25	10:30	10:35	10:40	10:45	10:50	10:55	11:00	11:05	11:10	11:15	11:20	11:25
S	On	On	On	On	Off	Off	Off	On	On	Off	Off	Off	Off	Off	Off	On	On	On
T	Off	Off	Off	Off	Off	Off	Off	On	On	Off	Off	On	On	Off	Off	On	On	On
W	On	On	On	On	On	On	On	On	On	Off	Off	Off	Off	On	On	On	On	On
I	Off	Off	Off	Off	Off	Off	On	On	On	Off	Off	Off	On	On	Off	Off	On	On

Time	11:30	11:35	11:40	11:45	11:50	11:55	12:00	12:05	12:10	12:15	12:20	12:25	12:30	12:35	12:40	12:45	12:50	12:55
S	Off	Off	Off	Off	Off	Off	Off	Off	Off	On	On	On	On	On	On	On	On	On
T	Off	Off	Off	Off	Off	Off	Off	Off	Off	On	On	On	On	On	On	On	On	On
W	Off	Off	Off	Off	Off	Off	Off	Off	Off	On	On	On	On	On	On	On	On	On
I	Off	Off	Off	Off	Off	Off	Off	Off	Off	On	On	Off	Off	Off	Off	Off	On	On

Definition 3.3 (Symbolic database) Given a set of time series $\mathcal{X} = \{X_1, ..., X_n\}$, the set of symbolic representations of the time series in $\mathcal{X}$ forms a *symbolic database* $\mathcal{D}_{\text{SYB}}$.

An example of the symbolic database $\mathcal{D}_{\text{SYB}}$ is shown in Table B.1. There are 4 time series representing the energy usage of 4 electrical appliances: {Stove, Toaster, Clothes Washer, Iron}. For brevity, we name the appliances respectively as {S, T, W, I}. All appliances have the same alphabet Σ = {On, Off}.

Definition 3.4 (Temporal event in a symbolic time series) A *temporal event* E in a symbolic time series X_S is a tuple $E = (\omega, T)$ where $\omega \in \Sigma_X$ is a symbol, and $T = \{[t_{s_i}, t_{e_i}]\}$ is the set of time intervals during which X_S is associated with the symbol ω.

Given a time series X, a temporal event is created by first converting X into symbolic time series X_S, and then combining identical consecutive symbols in X_S into one single time interval. For example, consider the symbolic representation of S in Table B.1. By combining its consecutive On symbols, we form the temporal event *"Stove is On"* as: (SOn, {[10:00, 10:15], [10:35, 10:40], [11:15, 11:25], [12:15, 12:55]}).

Definition 3.5 (Instance of a temporal event) Let $E = (\omega, T)$ be a temporal event, and $[t_{s_i}, t_{e_i}] \in T$ be a time interval. The tuple $e = (\omega, [t_{s_i}, t_{e_i}])$ is called an *instance* of the event E, representing a single occurrence of E during $[t_{s_i}, t_{e_i}]$. We use the notation $E_{\triangleright e}$ to say that event E has an instance e.

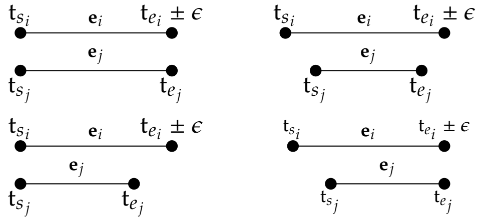
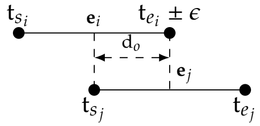
B.3.2 Relations between Temporal Events

We adopt the popular Allen's relations model [41] and define three basic temporal relations between events. Furthermore, to avoid the exact time mapping problem in Allen's relations, we adopt the *buffer* idea from [5], adding a tolerance *buffer* ϵ to the relation's endpoints. However, we change the way ϵ is used in [5] to ensure the relations are *mutually exclusive* (proof is in the electronic appendix [42]).

Consider two temporal events E_i and E_j, and their corresponding instances, $e_i = (\omega_i, [t_{s_i}, t_{e_i}])$ and $e_j = (\omega_j, [t_{s_j}, t_{e_j}])$. Let ϵ be a non-negative number ($\epsilon \geq 0$) representing the buffer size. The following relations can be defined between E_i and E_j through e_i and e_j.

Definition 3.6 (Follows) E_i and E_j form a *Follows* relation through e_i and e_j,

Table B.2: Temporal Relations between Events

Follows: $E_{i_{\triangleright e_i}} \rightarrow E_{j_{\triangleright e_j}}$	 $t_{e_i} \pm \epsilon \leq t_{s_j}$
Contains: $E_{i_{\triangleright e_i}} \succcurlyeq E_{j_{\triangleright e_j}}$	 $(t_{s_i} \leq t_{s_j}) \wedge (t_{e_i} \pm \epsilon \geq t_{e_j})$
Overlaps: $E_{i_{\triangleright e_i}} \between E_{j_{\triangleright e_j}}$	 $(t_{s_i} < t_{s_j}) \wedge (t_{e_i} \pm \epsilon < t_{e_j}) \wedge (t_{e_i} - t_{s_j} \geq d_o \pm \epsilon)$

denoted as Follows($E_{i_{\triangleright e_i}}$,$E_{j_{\triangleright e_j}}$) or $E_{i_{\triangleright e_i}} \rightarrow E_{j_{\triangleright e_j}}$, iff $t_{e_i} \pm \epsilon \leq t_{s_j}$.
Definition 3.7 (Contains) E_i and E_j form a *Contains* relation through e_i and e_j, denoted as Contains($E_{i_{\triangleright e_i}}$, $E_{j_{\triangleright e_j}}$) or $E_{i_{\triangleright e_i}} \succcurlyeq E_{j_{\triangleright e_j}}$, iff $(t_{s_i} \leq t_{s_j}) \wedge (t_{e_i} \pm \epsilon \geq t_{e_j})$.
Definition 3.8 (Overlaps) E_i and E_j form an *Overlaps* relation through e_i and e_j, denoted as Overlaps($E_{i_{\triangleright e_i}}$, $E_{j_{\triangleright e_j}}$) or $E_{i_{\triangleright e_i}} \between E_{j_{\triangleright e_j}}$, iff $(t_{s_i} < t_{s_j}) \wedge (t_{e_i} \pm \epsilon < t_{e_j}) \wedge (t_{e_i} - t_{s_j} \geq d_o \pm \epsilon)$, where d_o is the minimal overlapping duration between two event instances, and $0 \leq \epsilon \ll d_o$.

The *Follows* relation represents sequential occurrences of one event after another. For example, $E_{i_{\triangleright e_i}}$ is followed by $E_{j_{\triangleright e_j}}$ if the end time t_{e_i} of e_i occurs before the start time t_{s_j} of e_j. Here, the buffer ϵ is used as a tolerance, i.e., the *Follows* relation between $E_{i_{\triangleright e_i}}$ and $E_{j_{\triangleright e_j}}$ holds if $(t_{e_i} + \epsilon)$ or $(t_{e_i} - \epsilon)$ occurs before t_{s_j}. On the other hand, in a *Contains* relation, one event occurs entirely within the timespan of another event. Finally, in an *Overlaps* relation, the timespans of the two occurrences overlap each other. Table B.2 illustrates the three temporal relations and their conditions.

B.3.3 Temporal Pattern

Definition 3.9 (Temporal sequence) A list of n event instances S=<e_1, ..., e_i, ..., e_n> forms a *temporal sequence* if the instances are chronologically ordered by their start times. Moreover, S has size n, denoted as $|S| = n$.
Definition 3.10 (Temporal sequence database) A set of temporal sequences forms a *temporal sequence database* $\mathcal{D}_{\text{SEQ}}$ where each row i contains a temporal sequence S_i.

Table B.3: A Temporal Sequence Database $\mathcal{D}_{\text{SEQ}}$

ID	Temporal sequences
1	(SOn,[10:00,10:15]), (TOff,[10:00,10:35]), (WOn,[10:00,10:40]), (IOff,[10:00,10:30]), (SOff,[10:15,10:35]), (IOn,[10:30,10:40]), (SOn, [10:35,10:40]), (TOn,[10:35,10:40])
2	(SOff,[10:45,11:15]), (TOff,[10:45,10:55]), (WOff,[10:45,11:05]), (IOff,[10:45,11:00]), (TOn,[10:55,11:00]), (TOff,[11:00,11:15]), (IOn,[11:00,11:05]), (WOn,[11:05,11:25]), (IOff,[11:05,11:20]), (SOn,[11:15,11:25]), (TOn,[11:15,11:25]), (IOn,[11:20,11:25])
3	(SOff,[11:30,12:10]), (TOff,[11:30,12:10]), (WOff,[11:30,12:10]), (IOff,[11:30,12:10])
4	(SOn,[12:15,12:55]), (TOn,[12:15,12:55]), (WOn,[12:15,12:55]), (IOn,[12:15,12:20]), IOff,[12:20,12:50]), (IOn,[12:50,12:55])

Table B.3 shows the temporal sequence database $\mathcal{D}_{\text{SEQ}}$, created from the symbolic database $\mathcal{D}_{\text{SYB}}$ in Table B.1.

Definition 3.11 (Temporal pattern) Let $\mathfrak{R}$={Follows, Contains, Overlaps} be the set of temporal relations. A *temporal pattern* $P = <(r_{12}, E_1, E_2),...,(r_{(n-1)(n)},E_{n-1},E_n)$ $>$ is a list of triples (r_{ij},E_i,E_j), each representing a relation $r_{ij} \in \mathfrak{R}$ between two events E_i and E_j.

Note that the relation r_{ij} in each triple is formed using the specific instances of E_i and E_j. A temporal pattern that has n events is called an n-event pattern. We use $E_i \in P$ to denote that the event E_i occurs in P, and $P_1 \subseteq P$ to say that a pattern P_1 is a sub-pattern of P.

Definition 3.12 (Temporal sequence supports a pattern) Let S=<e_1,...,e_i,...,e_n> be a temporal sequence. We say that S *supports* a temporal pattern P, denoted as $P \in S$, iff $|S| \geq 2 \wedge \forall(r_{ij}, E_i, E_j) \in P, \exists(e_l, e_m) \in S$ such that r_{ij} holds between $E_{i_{\triangleright e_l}}$ and $E_{j_{\triangleright e_m}}$.

If P is supported by S, P can be written as P=<(r_{12}, $E_{1_{\triangleright e_1}}$, $E_{2_{\triangleright e_2}}$), ..., ($r_{(n-1)(n)}$,$E_{n-1_{\triangleright e_{n-1}}}$, $E_{n_{\triangleright e_n}}$)>, where the relation between two events in each triple is expressed using the event instances.

In Fig. A.1, consider the sequence S =<e_1=(HighCO2, [6:00, 10:00]), e_2=(BoilerOn, [7:00, 8:00]), e_3=(LowCO2, [13:00, 15:00])> representing the order of CO2 intensity and boiler events. Here, S supports a 3-event pattern P=<(Contains, HighCO2$_{\triangleright e_1}$, BoilerOn$_{\triangleright e_2}$), (Follows, HighCO2$_{\triangleright e_1}$, LowCO2$_{\triangleright e_3}$), (Follows, BoilerOn$_{\triangleright e_2}$, LowCO2$_{\triangleright e_3}$)>.

Maximal duration constraint: Let $P \in S$ be a temporal pattern supported by the sequence S. The duration between the start time of the instance e_1, and the end time of the instance e_n in S must not exceed the predefined maximal time duration $t_{\max}$: $t_{e_n} - t_{s_1} \leq t_{\max}$.

The maximal duration constraint guarantees that the relation between any two events is temporally valid. This enables the pruning of invalid patterns. For example, under this constraint, a *Follows* relation between a *"Washer On"* event and a *"Dryer On"* event in Table B.3 happening one year apart should be considered invalid.

B.3.4 Frequency and Likelihood Measures

Given a temporal sequence database $\mathcal{D}_{\text{SEQ}}$, we want to find patterns that occur at certain frequency in $\mathcal{D}_{\text{SEQ}}$. We use *support* and *confidence* [43] to measure the frequency and likelihood of a pattern.

Definition 3.13 (Support of a temporal event) The *support* of a temporal event E in $\mathcal{D}_{\text{SEQ}}$ is the number of sequences $S \in \mathcal{D}_{\text{SEQ}}$ containing at least one instance e of E.

$$supp(E) = |\{S \in \mathcal{D}_{\text{SEQ}} \; s.t. \; \exists e \in S : E_{\triangleright e}\}| \tag{B.1}$$

The *relative support* of E is the fraction between $supp(E)$ and the size of $\mathcal{D}_{\text{SEQ}}$:

$$rel\text{-}supp(E) = supp(E)/|\mathcal{D}_{\text{SEQ}}| \tag{B.2}$$

Similarly, the support of a group of events $(E_1, ..., E_n)$, denoted as $supp(E_1, ..., E_n)$, is the number of sequences $S \in \mathcal{D}_{\text{SEQ}}$ which contain at least one instance $(e_1, ..., e_n)$ of the event group.

Definition 3.14 (Support of a temporal pattern) The *support* of a pattern P is the number of sequences $S \in \mathcal{D}_{\text{SEQ}}$ that support P.

$$supp(P) = |\{S \in \mathcal{D}_{\text{SEQ}} \; s.t. \; P \in S\}| \tag{B.3}$$

The *relative support* of P in $\mathcal{D}_{\text{SEQ}}$ is the fraction

$$rel\text{-}supp(P) = supp(P)/|\mathcal{D}_{\text{SEQ}}| \tag{B.4}$$

Definition 3.15 (Confidence of an event pair) The *confidence* of an event pair (E_i, E_j) in $\mathcal{D}_{\text{SEQ}}$ is the fraction between $supp(E_i, E_j)$ and the support of its most frequent event:

$$conf(E_i, E_j) = \frac{supp(E_i, E_j)}{\max\{supp(E_i), supp(E_j)\}} \tag{B.5}$$

Definition 3.16 (Confidence of a temporal pattern) The *confidence* of a temporal pattern P in $\mathcal{D}_{\text{SEQ}}$ is the fraction between $supp(P)$ and the support of its most frequent event:

$$conf(P) = \frac{supp(P)}{\max_{1 \le k \le |P|}\{supp(E_k)\}} \tag{B.6}$$

where $E_k \in P$ is a temporal event. Since the denominator in Eq. (B.6) is the maximum support of the events in P, the confidence computed in Eq. (B.6) is the *minimum confidence* of a pattern P in $\mathcal{D}_{\text{SEQ}}$, which is also called the *all-confidence* as in [43]. Note that unlike association rules, temporal patterns do not have antecedents and consequents. Instead, they represent pair-wise temporal relations between events based on their temporal occurrences. Thus, while the *support* and *relative support* of event(s)/ pattern(s) defined in Eqs. (B.1) – (B.4) follow the same intuition as the traditional support concept, indicating how frequently an event/ pattern occurs in a given database, the *confidence* computed in Eqs. (B.5) – (B.6) instead represents the minimum likelihood of an event pair/ pattern, knowing the likelihood of its most frequent event.

Frequent temporal patterns vs. Rare temporal patterns: Consider a temporal pattern P in a temporal sequence database $\mathcal{D}_{\text{SEQ}}$ with the support $\sigma = supp(P)$ and the confidence $\delta = conf(P)$. Pattern P is considered to be *frequent* in $\mathcal{D}_{\text{SEQ}}$ if both support σ and confidence δ are high, representing the presence of pattern P in a large fraction of sequences in the database. In contrast, pattern P is considered to be *rare* in $\mathcal{D}_{\text{SEQ}}$ if the support σ is low and the confidence δ is high, indicating a type of pattern that occurs only in a small fraction of sequences but with high likelihood, given the occurrence evidence of the involved events.

Problem Definition: Generalized Temporal Pattern Mining. Given a set of univariate time series $\mathcal{X} = \{X_1, ..., X_n\}$, let $\mathcal{D}_{\text{SEQ}}$ be the temporal sequence database obtained from $\mathcal{X}$, and $\sigma_{\min}$, $\sigma_{\max}$ and δ be the minimum support, maximum support and minimum confidence thresholds, respectively. The Generalized Temporal Pattern Mining from Time Series (GTPMfTS) problem aims to find all temporal patterns P in $\mathcal{D}_{\text{SEQ}}$ such that P satisfies the support and confidence constraints, i.e., $\sigma_{\min} \leq supp(P) \leq \sigma_{\max} \wedge conf(P) \geq \delta$.

Using the three constraints $\sigma_{\min}$, $\sigma_{\max}$ and δ, GTPMfTS can mine frequent temporal patterns in $\mathcal{D}_{\text{SEQ}}$ by setting $\sigma_{\max} = \infty$, and assigning $\sigma_{\min}$ and δ to high threshold values. In contrast, to mine rare temporal patterns, GTPMfTS will assign low threshold values to $\sigma_{\min}$ and $\sigma_{\max}$, constraining on a low occurrence frequency, and a high value to δ, constraining on a high likelihood of the patterns.

B.4 Generalized Temporal Pattern Mining

In this section, we present the Generalized Temporal Pattern Mining (GTPM) algorithm to mine both frequent and rare temporal patterns from time series. Fig. B.2 gives an overview of the GTPMfTS process which consists of two phases. The first phase, *Data Transformation*, converts a set of time series $\mathcal{X}$ into a symbolic database $\mathcal{D}_{\text{SYB}}$, and then converts $\mathcal{D}_{\text{SYB}}$ into a temporal sequence database $\mathcal{D}_{\text{SEQ}}$. The second phase, *Generalized Temporal Pattern Mining (GTPM)*, mines both frequent and rare temporal patterns, and consists of three steps: (1) *Mining Single Events*, (2) *Mining 2-Event Patterns*, and (3) *Mining k-Event Patterns* (k>2). The final output is a set of all temporal patterns in $\mathcal{D}_{\text{SEQ}}$ that satisfy the minimum support, maximum support and minimum confidence constraints.

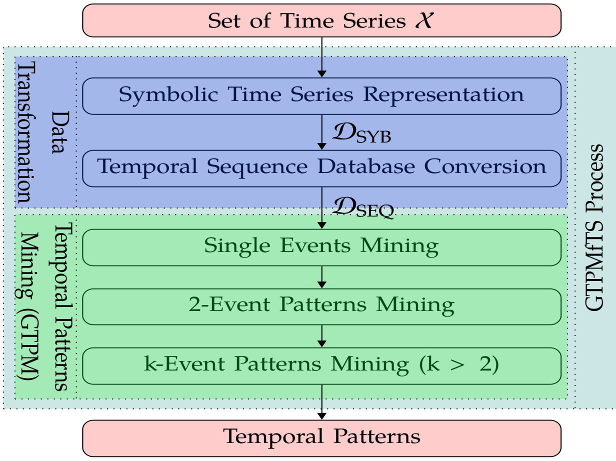
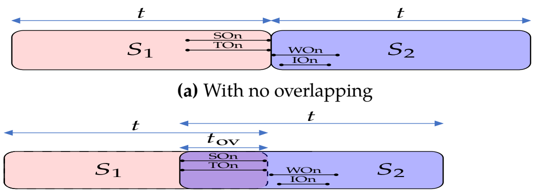


Fig. B.2: The GTPMfTS process



(a) With no overlapping

(b) With overlapping

Fig. B.3: Splitting strategy

B.4.1 Data Transformation

Symbolic Time Series Representation

Given a set of time series $\mathcal{X}$, the symbolic representation of each time series $X \in \mathcal{X}$ is obtained by using a mapping function as in Def. 3.2.

Temporal Sequence Database Conversion

To convert $\mathcal{D}_{SYB}$ to $\mathcal{D}_{SEQ}$, a straightforward approach is to split the symbolic series in $\mathcal{D}_{SYB}$ into equal-length sequences, each belongs to a row in $\mathcal{D}_{SEQ}$. For example, if each symbolic series in Table B.1 is split into 4 sequences, then each sequence will last for 40 minutes. The first sequence S_1 of $\mathcal{D}_{SEQ}$ therefore contains temporal events of S, T, W, and I from 10:00 to 10:40. The second sequence S_2 contains events from 10:45 to 11:25, and similarly for S_3 and S_4.

However, the splitting can lead to a potential loss of temporal patterns. The loss happens when a *splitting point* accidentally divides a temporal pattern into different sub-patterns, and places these into separate sequences. We explain this situation in Fig. B.3a. Consider 2 sequences S_1 and S_2, each of length t. Here, the splitting point divides a pattern of 4 events, {SOn, TOn, WOn, IOn},

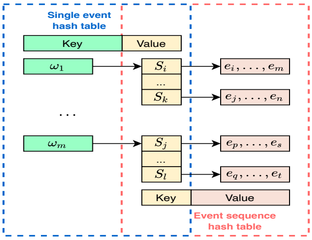


Fig. B.4: The HLH_1 structure

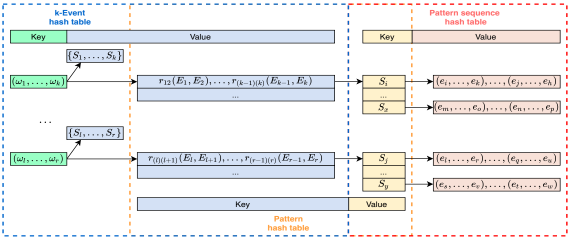


Fig. B.5: The $HLH_k (k \geq 2)$ structure

into two sub-patterns, in which SOn and TOn are placed in S_1, and WOn and IOn in S_2. This results in the loss of this 4-event pattern which can be identified only when all 4 events are in the same sequence.

To prevent such a loss, we propose a *splitting strategy* using overlapping sequences. Specifically, two consecutive sequences are overlapped by a duration t_{ov}: $0 \leq t_{\text{ov}} \leq t_{\text{max}}$, where t_{max} is the *maximal duration* of a temporal pattern. The value of t_{ov} decides how large the overlap between S_i and S_{i+1} is: $t_{\text{ov}} = 0$ results in no overlap, i.e., no redundancy, but with a potential loss of patterns, while $t_{\text{ov}} = t_{\text{max}}$ creates large overlaps between sequences, i.e., high redundancy, but all patterns are preserved. As illustrated in Fig. B.3b, the overlapping between S_1 and S_2 keeps the 4 events together in the same sequence S_2, and thus helps preserve the pattern.

B.4.2 Generalized Temporal Pattern Mining

We now present the GTPM algorithm to mine temporal patterns, both frequent and rare, from $\mathcal{D}_{\text{SEQ}}$. We note that for frequent patterns, only two constraints σ_{min} and δ are used, whereas with rare patterns, all three constraints σ_{min}, σ_{max}, and δ are used. In the following when presenting the GTPM algorithm, the discussion applies to both frequent and rare patterns, with the implication that σ_{max} is set to ∞ when mining frequent patterns.

The main novelties of GTPM are: a) the use of efficient data structures, i.e., the *Hierarchical Lookup Hash (HLH)* structure [44], and b) the proposal of two groups of pruning techniques based on the Apriori principle and the temporal transitivity property of temporal events. Particularly, instead of using the Hierarchical Pattern Graph as in [13], we use the Hierarchical Lookup Hash data structure to enable faster retrieval of events and patterns during the mining process. Algorithm 9 provides the pseudo-code of our GTPM algorithm.

B.4.3 Mining Single Events

Hierarchical lookup hash structure HLH_1: We use the hierarchical lookup hash structure HLH_1, illustrated in Fig. B.4 to store single events. HLH_1 is a

Algorithm 9: Generalized Temporal Pattern Mining

Input: Temporal sequence database $\mathcal{D}_{\text{SEQ}}$, minimum support threshold σ_{min}, maximum support threshold σ_{max}, confidence threshold δ

Output: The set of temporal patterns P satisfying σ_{min}, σ_{max}, δ

//Mining single events

1: **foreach** *event* $E_i \in \mathcal{D}_{SEQ}$ **do**
2:     Compute $supp(E_i)$;
3:     **if** $supp(E_i) \geq \sigma_{\text{min}}$ **then**
4:         Insert E_i to *1Freq*;

//Mining 2-event patterns

5: EventPairs ← Cartesian(*1Freq*,*1Freq*);
6: FrequentPairs ← $\emptyset$;
7: **foreach** (E_i, E_j) *in EventPairs* **do**
8:     Compute $supp(E_i, E_j)$;
9:     **if** $supp(E_i, E_j) \geq \sigma_{\text{min}}$ **then**
10:         FrequentPairs ← Apply_Lemma4(E_i, E_j);
11: **foreach** (E_i, E_j) *in FrequentPairs* **do**
12:     Retrieve event instances;
13:     Check temporal relations against σ_{min}, σ_{max}, δ;

//Mining k-event patterns

14: Filtered1Freq ← Transitivity_Filtering(1Freq);
15: kEvents ← Cartesian(*Filtered1Freq*,*(k-1)Freq*);
16: FrequentkEvents ← Apriori_Filtering(kEvents);
17: **foreach** *kEvents in FrequentkEvents* **do**
18:     Retrieve relations;
19:     Iteratively check relations against σ_{min}, σ_{max}, δ;

hierarchical data structure that consists of two hash tables: the *single event hash table* EH, and the *event sequence hash table* SH. Each hash table has a list of <key, value> pairs. In EH, the key is the event symbol $\omega \in \Sigma_X$ representing the event E_i, and the value is the set of sequences $< S_i, ..., S_k >$ (arranged in an increasing order) that contain E_i. In SH, the key is taken from the value component of EH, i.e., the set of sequences, while the value stores event instances of E_i that occur in the corresponding sequence in $\mathcal{D}_{\text{SEQ}}$. The HLH_1 structure enables faster retrieval of event sequences and instances when mining k-event patterns.

Mining Single Events: The first step in GTPM is to find single events that satisfy the minimum support constraint σ_{min} (Alg. 9, lines 1-4). To do that, GTPM scans $\mathcal{D}_{\text{SEQ}}$ to compute the support of each event E_i, and checks whether $supp(E_i) \geq \sigma_{\text{min}}$. Note that for single events, we do not consider the constraints on the confidence δ, since confidence of single events is always 1, and on maximum support σ_{max} because of the following lemma.

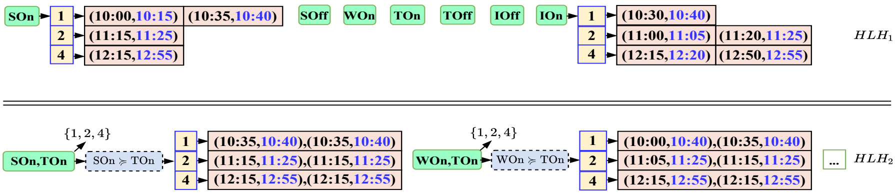


Fig. B.6: A hierarchical lookup hash tables for the running example

Lemma 1 *Let P be a temporal pattern and E_i be a single event such that $E_i \in P$. Then $supp(P) \le supp(E_i)$.*

Proof. *Detailed proofs of all lemmas, theorems, and complexities in this article can be found in the electronic appendix [42].*

From Lemma 1, a single event E_i whose support $supp(E_i) > \sigma_{\max}$ can form a pattern P that has $supp(P) \le \sigma_{\max}$. Thus, the constraint on $\sigma_{\max}$ is not considered for single events to avoid the loss of potential temporal patterns.

We provide a running example using data in Table B.3, with $\sigma_{\min} = 0.7$, $\sigma_{\max} = 0.9$, and $\delta = 0.7$. The data structure HLH_1, shown in Fig. B.6, stores 7 single events satisfying $\sigma_{\min}$ constraint. The event WOff does not satisfy $\sigma_{\min}$ (only appears in sequences 2 and 4), and is thus omitted.

Complexity: The complexity of finding single events is $O(m \cdot |\mathcal{D}_{\text{SEQ}}|)$, where m is the number of distinct events.

B.4.4 Mining 2-event Patterns

Search space of GTPM: The next step in GTPM is to mine 2-event patterns. A straightforward approach would be to enumerate all possible event pairs, and check whether each pair can form patterns that satisfy the support and confidence constraints. However, this *naive* approach is very expensive. Not only does it need to repeatedly scan $\mathcal{D}_{\text{SEQ}}$ to check each combination of events, the complex relations between events also add an extra exponential factor 3^{h^2} to the m^h number of possible candidates, creating a very large search space that makes the approach infeasible.

Lemma 2 *Let m be the number of distinct events in $\mathcal{D}_{\text{SEQ}}$, and h be the longest length of a temporal pattern. The total number of temporal patterns is $O(m^h 3^{h^2})$.*

Lemma 2 shows the driving factors of GTPM's exponential search space (proof in the electronic appendix [42]): the number of events (m), the max pattern length (h), and the number of temporal relations (3). A dataset of just a few hundred events can create a very large search space with billions of candidate patterns. The optimizations and approximation proposed in the following sections will help mitigate this problem.

Hierarchical lookup hash structure HLH_k**:** We maintain k-event groups and patterns found by GTPM using the HLH_k ($k \ge 2$) data structure, illustrated in Fig. B.5. HLH_k contains three hash tables, each has a list of <key, value> pairs: the *k-event hash table* EH_k, the *pattern hash table* PH_k, and the *pattern sequence hash table* SH_k. For each *<key, value>* pair of EH_k, *key* is the list of symbols $(\omega_1..., \omega_k)$ representing the k-event group $(E_1, ..., E_k)$, and *value* is an *object* structure which consists of two components: (1) the list of sequences $< S_i, ..., S_k >$ (arranged in increasing order) where $(E_1, ..., E_k)$ occurs, and (2), a list of k-event temporal patterns $P = \{(r_{12}, E_1, E_2), ..., (r_{(k-1)(k)}, E_{k-1}, E_k)\}$ created from the k-event group $(E_1, ..., E_k)$. In PH_k, the *key* takes the *value* component of EH_k, i.e. the k-event pattern P, while the *value* is the list of sequences that support P. In SH_k, the *key* takes the *value* component of PH_k, i.e., the list of sequences that support P, while the *value* is the list of event instances from which the temporal relations in P are formed. The HLH_k hash structure helps speed up the mining of k-event groups through the use of sequences in EH_k, and enables faster search for temporal relations between k events using the information in PH_k and SH_k.

Two-steps filtering approach to mine 2-event patterns: Given the huge set of pattern candidates stated in Lemma 1, it is expensive to check their support and confidence. We propose a *filtering approach* to reduce the unnecessary candidate checking. Specifically, the mining process is divided into two steps: (1) it first finds k-event groups that satisfy the minimum support and confidence constraints using $\sigma_{\min}$ and δ, (2) it then generates temporal patterns only from those k-event groups. The correctness of this filtering approach is based on the Apriori-inspired lemmas below.

Lemma 3 *Let P be a 2-event pattern formed by an event pair (E_i, E_j). Then, $supp(P) \leq supp(E_i, E_j)$.*

From Lemma 3, the support of a pattern is at most the support of its events. Thus, infrequent event pairs (those do not satisfy minimum support) cannot form frequent patterns and thereby, can be safely pruned.

Lemma 4 *Let (E_i, E_j) be a pair of events forming a 2-event pattern P. Then conf(P) $\leq$ conf(E_i, E_j).*

From Lemma 4, the confidence of a pattern P is always at most the confidence of its events. Thus, a low-confidence event pair cannot form any high-confidence patterns and therefore, can be safely pruned. We note that the Apriori principle has already been used in other work, e.g., [3, 5], for mining optimization. However, they only apply this principle to the support (Lemma 3), while we further extend it to the confidence (Lemma 4). Applying Lemmas 3 and 4 to the event filtering step will remove infrequent or low-confidence event pairs, reducing the candidate patterns of GTPM. Furthermore, we do not consider the constraint on $\sigma_{\max}$ in this filtering step to avoid the loss of 2-event patterns, as event pairs that do not satisfy the $\sigma_{\max}$ constraint can still form 2-event patterns satisfying $\sigma_{\max}$ (Lemma 3).

Step 2.1. Mining event pairs considering $\sigma_{\min}$ and δ: This step finds event pairs in $\mathcal{D}_{\text{SEQ}}$ satisfying $\sigma_{\min}$ and δ, using the set *1Freq* found in HLH_1 (Alg. 9, lines 5-10). First, GTPM generates all possible event pairs by calculating the Cartesian product *1Freq* $\times$ *1Freq*. Next, for each pair (E_i, E_j), the set $\mathcal{S}_{ij}$ (representing the set of sequences where both events occur) is computed by taking the *intersection* between the set of sequences $\mathcal{S}_i$ of E_i and the set of sequences $\mathcal{S}_j$ of E_j in HLH_1. Finally, we compute the support $supp(E_i, E_j)$ using $\mathcal{S}_{ij}$, and compare against $\sigma_{\min}$. If $supp(E_i, E_j) \geq \sigma_{\min}$, (E_i, E_j) has high enough support. Next, (E_i, E_j) is further filtered using Lemma 4: (E_i, E_j) is selected only if its confidence is at least δ. After this step, only event pairs satisfying $\sigma_{\min}$ and δ are kept in EH_2 of HLH_2.

Step 2.2. Mining 2-event patterns: This step mines 2-event patterns from the event pairs found in step 2.1 (Alg. 9, lines 11-13), considering three constraints $\sigma_{\min}$, $\sigma_{\max}$, and δ. For each event pair (E_i, E_j), we use the set of sequences $\mathcal{S}_{ij}$ to check the temporal relations between E_i and E_j. Specifically, for each sequence $S \in \mathcal{S}_{ij}$, the pairs of event instances (e_i, e_j) are extracted, and the relations between them are verified. The support and confidence of each relation $r(E_{i_{\triangleright e_i}}, E_{j_{\triangleright e_j}})$ are computed and compared against $\sigma_{\min}$, and δ thresholds, after which only relations satisfying the two constraints are selected and stored in PH_2, while their event instances are stored in SH_2. Examples of the relations in HLH_2 can be seen in Fig. B.6, e.g., event pair (SOn, TOn). We also emphasize that HLH_2 only stores patterns that satisfy the two constraints $\sigma_{\min}$, and δ, thus, patterns in PH_2 are frequent temporal patterns. To mine rare temporal patterns from HLH_2, we take a further step by iterating through every 2-event pattern P in PH_2, and checking the satisfaction of P against the constraint $\sigma_{\max}$.

Complexity: Let m be the number of single events in HLH_1, and i be the average number of event instances of each event. The complexity of 2-event pattern mining is $O(m^2 i^2 \, |\mathcal{D}_{\text{SEQ}}|^2)$.

B.4.5 Mining k-event Patterns

Mining k-event patterns ($k \geq 3$) follows a similar process as 2-event patterns, with additional prunings based on the transitivity property of temporal relations.

Step 3.1. Mining k-event combinations considering $\sigma_{\min}$ and δ: This step finds k-event combinations that satisfy the minimum support and confidence constraints (Alg. 9, lines 14-16).

Let *(k-1)Freq* be the set of (k-1)-event combinations found in HLH_{k-1}, and *1Freq* be the set of single events in HLH_1. To generate all k-event combinations, the typical process is to compute the Cartesian product: *(k-1)Freq* $\times$ *1Freq*. However, we observe that using *1Freq* to generate k-event combinations at HLH_k can create redundancy, since *1Freq* might contain events that when combined with *(k-1)Freq*, result in combinations that clearly cannot form any patterns satisfying the minimum support constraint. To illustrate this observation, consider the event IOn in HLH_1 in Fig. B.6. Here, IOn is a frequent event, and thus, can be combined with frequent event pairs in HLH_2 such as (SOn, TOn) to create a 3-event combination (SOn, TOn, IOn). However, (SOn, TOn, IOn) cannot form any 3-event patterns whose support is greater than $\sigma_{\min}$, since IOn is not present in any frequent 2-event patterns in HLH_2. To reduce the redundancy, the combination (SOn, TOn, IOn) should not be created in the first place. We rely on the *transitivity property* of temporal relations to identify such event combinations.

Lemma 5 *Let* $S =< e_1,..., e_{n-1} >$ *be a temporal sequence that supports an (n-1)-event pattern* $P =< (r_{12}, E_{1_{\triangleright e_1}}, E_{2_{\triangleright e_2}}),..., (r_{(n-2)(n-1)}, E_{n-2_{\triangleright e_{n-2}}}, E_{n-1_{\triangleright e_{n-1}}}) >$. *Let* e_n *be a new event instance added to* S *to create the temporal sequence* $S^{'} =< e_1, ..., e_n >$.

The set of temporal relations $\mathfrak{R}$ *is transitive on* $S^{'}$: $\forall e_i \in S^{'}, i < n, \exists r \in \mathfrak{R}$ *s.t.* $r(E_{i_{\triangleright e_i}}, E_{n_{\triangleright e_n}})$ *holds.*

Lemma 5 says that given a temporal sequence S, a new event instance added to S will always form at least one temporal relation with existing instances in S. This is due to the temporal transitivity property, which can be used to prove the following lemma.

Lemma 6 *Let* $N_{k-1} = (E_1, ..., E_{k-1})$ *be a (k-1)-event combination and* E_k *be a single event, both satisfying the* $\sigma_{\min}$ *constraint. The combination* $N_k = N_{k-1} \cup E_k$ *can form k-event temporal patterns whose support is at least* $\sigma_{\min}$ *if* $\forall E_i \in N_{k-1}, \exists r \in \mathfrak{R}$ *s.t.* $r(E_i, E_k)$ *is a frequent temporal relation.*

From Lemma 6, only single events in HLH_1 that appear in HLH_{k-1} should be used to create k-event combinations. Using this result, a filtering on *1Freq* is performed before calculating the Cartesian product. Specifically, from the events in HLH_{k-1}, we extract distinct single events D_{k-1}, and *intersect* D_{k-1} with *1Freq* to remove redundant single events: *Filtered1Freq* $= D_{k-1} \cap$ *1Freq*. Next, the Cartesian product *(k-1)Freq* $\times$ *Filtered1Freq* is calculated to generate k-event combinations. Finally, we apply Lemmas 3 and 4 to select k-event combinations *kFreq* which upheld the $\sigma_{\min}$ and δ constraints. Similar to step 2.1, we do not consider $\sigma_{\max}$ when generating the k-event combination.

Step 3.2. Mining k-event patterns: This step mines k-event patterns that satisfy the three constraints of $\sigma_{\min}$, $\sigma_{\max}$, and δ (Alg. 9, lines 17-19). Unlike 2-event patterns, verifying the relations in a k-event combination ($k \geq 3$) is much more expensive, as it requires to compute the frequency of $\frac{1}{2}k(k-1)$ triples of temporal relations. To reduce the cost of relation checking, we propose an iterative verification method that relies on the *transitivity property* and the Apriori principle.

Lemma 7 *Let* P *and* $P^{'}$ *be two temporal patterns. If* $P^{'} \subseteq P$, *then conf(*$P^{'}$*)* $\geq$ *conf(*P*).*

Lemma 8 *Let P and P' be two temporal patterns. If $P' \subseteq P$ and $\frac{supp(P')}{\max_{1\le k\le |P|}\{supp(E_k)\}}_{E_k \in P}$ $\le \delta$, then conf(P) $\le \delta$.*

Lemma 7 says that, the confidence of a pattern P is always at most the confidence of its sub-patterns. Consequently, from Lemma 8, a temporal pattern P cannot be high-confidence if any of its sub-patterns are low-confidence.

Let $N_{k-1} = (E_1, ..., E_{k-1})$ be a (k-1)-event combination in HLH_{k-1}, $N_1 = (E_k)$ be an event in HLH_1, and $N_k = N_{k-1} \cup N_1 = (E_1, ..., E_k)$ be a k-event combination in HLH_k. To find k-event patterns for N_k, we first retrieve the set P_{k-1} containing (k-1)-event patterns of N_{k-1} by accessing the EH_{k-1} table. Each $p_{k-1} \in P_{k-1}$ is a list of $\frac{1}{2}(k-1)(k-2)$ triples: $\{(r_{12}, E_{1_{\triangleright e_1}}, E_{2_{\triangleright e_2}})$,...,$(r_{(k-2)(k-1)}, E_{k-2_{\triangleright e_{k-2}}}, E_{k-1_{\triangleright e_{k-1}}})\}$. We iteratively verify the possibility of p_{k-1} forming a k-event pattern with E_k that can satisfy the $\sigma_{\min}$ constraint as follows. We first check whether the triple $(r_{(k-1)k}, E_{k-1_{\triangleright e_{k-1}}}, E_{k_{\triangleright e_k}})$ satisfies the constraints of $\sigma_{\min}$, $\sigma_{\max}$, and δ by accessing the HLH_2 table. If the triple does not satisfy the minimum and maximum support constraints (using Lemmas 5 and 6), or the confidence constraint (using Lemmas 5, 7, and 8), the verifying process stops immediately for p_{k-1}. Otherwise, it continues on the triple $(r_{(k-2)k}, E_{k-2_{\triangleright e_{k-2}}}, E_{k_{\triangleright e_k}})$, until it reaches $(r_{1k}, E_{1_{\triangleright e_1}}, E_{k_{\triangleright e_k}})$.

We note that the transitivity property of temporal relations has been exploited in [16] to generate new relations. Instead, we use this property to prune unpromising candidates (Lemmas 5, 6, 7, 8).

Complexity: Let r be the average number of (k-1)-event patterns in HLH_{k-1}. The complexity of k-event pattern mining is $O(|1Freq| \cdot |(k\text{-}1)Freq| \cdot r \cdot k^2 \cdot |\mathcal{D}_{\text{SEQ}}|)$.

GTPM overall complexity: Throughout this section, we have seen that GTPM complexity depends on the size of the search space ($O(m^h 3^{h^2})$) and the complexity of the mining process itself, i.e., $O(m \cdot |\mathcal{D}_{\text{SEQ}}|) + O(m^2 i^2\ |\mathcal{D}_{\text{SEQ}}|^2)$ + $O(|1Freq| \cdot |(k\text{-}1)Freq| \cdot r \cdot k^2 \cdot |\mathcal{D}_{\text{SEQ}}|)$. While the parameters m, h, i, r and k depend on the number of time series, others such as |*1Freq*|, |*(k-1)Freq*| and $|\mathcal{D}_{\text{SEQ}}|$ also depend on the number of temporal sequences. Thus, given a dataset, GTPM complexity is driven by two main factors: the number of time series and the number of temporal sequences.

B.5 Approximate GTPM

B.5.1 Mutual Information of Symbolic Time Series

Let X_S and Y_S be the symbolic series representing the time series X and Y, respectively, and Σ_X, Σ_Y be their alphabets.

Definition 5.1 (Entropy) The *entropy* of X_S, denoted as $H(X_S)$, is defined as

$$H(X_S) = -\sum_{x \in \Sigma_X} p(x) \cdot \log p(x) \tag{B.7}$$

Intuitively, the entropy measures the amount of information or the inherent uncertainty in the possible outcomes of a random variable. The higher the $H(X_S)$, the more uncertain the outcome of X_S.

The conditional entropy $H(X_S|Y_S)$ quantifies the amount of information needed to describe the outcome of X_S, given the value of Y_S, and is defined as

$$H(X_S|Y_S) = -\sum_{x\in\Sigma_X}\sum_{y\in\Sigma_Y} p(x,y)\cdot\log\frac{p(x,y)}{p(y)} \tag{B.8}$$

Definition 5.2 (Mutual information) The *mutual information* (MI) of two symbolic series X_S and Y_S, denoted as $I(X_S;Y_S)$, is defined as

$$I(X_S;Y_S) = \sum_{x\in\Sigma_X}\sum_{y\in\Sigma_Y} p(x,y)\cdot\log\frac{p(x,y)}{p(x)\cdot p(y)} \tag{B.9}$$

The MI represents the reduction of uncertainty of one variable (e.g., X_S), given the knowledge of another variable (e.g., Y_S). The larger $I(X_S;Y_S)$, the more information is shared between X_S and Y_S, and thus, the less uncertainty about one variable given the other.

Since $0 \leq I(X_S;Y_S) \leq \min(H(X_S), H(Y_S))$ [45], MI has no upper bound. To scale the MI into the range $[0-1]$, we use normalized mutual information as defined below.

Definition 5.3 (Normalized mutual information) The *normalized mutual information* (NMI) of two symbolic time series X_S and Y_S, denoted as $\widetilde{I}(X_S;Y_S)$, is defined as

$$\widetilde{I}(X_S;Y_S) = \frac{I(X_S;Y_S)}{H(X_S)} = 1 - \frac{H(X_S|Y_S)}{H(X_S)} \tag{B.10}$$

$\widetilde{I}(X_S;Y_S)$ represents the reduction (in percentage) of the uncertainty of X_S due to knowing Y_S. Based on Eq. (A.10), a pair of variables (X_S, Y_S) holds a mutual dependency if $\widetilde{I}(X_S;Y_S) > 0$. Eq. (A.10) also shows that NMI is not symmetric, i.e., $\widetilde{I}(X_S;Y_S) \neq \widetilde{I}(Y_S;X_S)$.

B.5.2 Lower Bound of the Support of an Event Pair

Consider two symbolic series X_S and Y_S. Let X_1 be an event in X_S, Y_1 be an event in Y_S, and $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$ be the symbolic and the sequence databases created from X_S and Y_S, respectively. We first study the relationship between the support of (X_1, Y_1) in $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$.

Lemma 1 *Let $supp(X_1, Y_1)_{\mathcal{D}_{SYB}}$ and $supp(X_1, Y_1)_{\mathcal{D}_{SEQ}}$ be the support of (X_1, Y_1) in $\mathcal{D}_{SYB}$ and $\mathcal{D}_{SEQ}$, respectively. Then $supp(X_1, Y_1)_{\mathcal{D}_{SYB}} \leq supp(X_1, Y_1)_{\mathcal{D}_{SEQ}}$ holds.*

Proof *(Sketch - Detailed proof in the electronic appendix [42]). Let n be the length of each symbolic time series in $\mathcal{D}_{SYB}$, and m be the length of each temporal sequence. The number of temporal sequences obtained in $\mathcal{D}_{SEQ}$ is: $\lceil \frac{n}{m} \rceil$.*

The support of (X_1, Y_1) *in* $\mathcal{D}_{SYB}$ *is computed as:*

$$supp(X_1, Y_1)_{\mathcal{D}_{SYB}} = \frac{\sum_{i=1}^{\lceil \frac{n}{m} \rceil} \sum_{j=1}^{m} s_{ij}}{n} \tag{B.11}$$

where

$$s_{ij} = \begin{cases} 1, & \textit{if } (X_1, Y_1) \textit{ occurs in row } j \textit{ of the sequence } s_i \textit{ in } \mathcal{D}_{SYB} \\ 0, & \textit{otherwise} \end{cases}$$

Moreover, we have:

$$supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} = \frac{\sum_{i=1}^{\lceil \frac{n}{m} \rceil} g_i}{n/m} = \frac{m \cdot \sum_{i=1}^{\lceil \frac{n}{m} \rceil} g_i}{n} \tag{B.12}$$

where

$$g_i = \begin{cases} 1, & \textit{if } (X_1, Y_1) \textit{ occurs in the sequence } g_i \textit{ in } \mathcal{D}_{SEQ} \\ 0, & \textit{otherwise} \end{cases}$$

We also get:

$$\begin{aligned} supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} &= \frac{m \cdot \sum_{i=1}^{\lceil \frac{n}{m} \rceil} g_i}{n} = \frac{\sum_{i=1}^{\lceil \frac{n}{m} \rceil} m \cdot g_i}{n} \\ &= \frac{\sum_{i=1}^{\lceil \frac{n}{m} \rceil} \left(\sum_{j=1}^{m} s_{ij} + \vartheta_i \right)}{n} \end{aligned} \tag{B.13}$$

where s_{ij} *is defined as in Eq.* (B.11)*, and*

$$\vartheta_i = \begin{cases} m - \sum_{j=1}^{m} s_{ij}, & \textit{if } \sum_{j=1}^{m} s_{ij} \neq 0 \\ 0, & \textit{otherwise} \end{cases}$$

From Eq. (B.13)*, we have:*

$$\begin{aligned} supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} &= \frac{\sum_{i=1}^{\lceil \frac{n}{m} \rceil} \sum_{j=1}^{m} s_{ij}}{n} + \frac{\sum_{i=1}^{\lceil \frac{n}{m} \rceil} \vartheta_i}{n} \\ &= supp(X_1, Y_1)_{\mathcal{D}_{SYB}} + \vartheta \end{aligned} \tag{B.14}$$

where $\vartheta = \frac{\sum_{i=1}^{\lceil \frac{n}{m} \rceil} \vartheta_i}{n}$ *is the difference between the probabilities of* (X_1, Y_1) *in* $\mathcal{D}_{SEQ}$ *and* $\mathcal{D}_{SYB}$.
From Eq. (B.14)*, we have:*

$$supp(X_1, Y_1)_{\mathcal{D}_{SYB}} \leq supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \tag{B.15}$$

From Lemma 1, a frequent event pair in $\mathcal{D}_{\text{SYB}}$ is also frequent in $\mathcal{D}_{\text{SEQ}}$. We now investigate the relation between $\widetilde{I}(X_S; Y_S)$ in $\mathcal{D}_{\text{SYB}}$ and the support of (X_1, Y_1) in $\mathcal{D}_{\text{SEQ}}$.

Theorem 1 *(Lower bound of the support) Let $\mu_{\min}$ be the minimum mutual information threshold. If the NMI $\widetilde{I}(X_S;Y_S) \geq \mu_{\min}$, then the lower bound of the support of (X_1, Y_1) in $\mathcal{D}_{SEQ}$ is:*

$$supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \lambda_2 \cdot e^{W\left(\frac{\log \lambda_1^{1-\mu_{\min}} \cdot ln2}{\lambda_2}\right)} \tag{B.16}$$

where λ_1 is the minimum support of $X_i \in X_S$, λ_2 is the support of $Y_1 \in Y_S$, and W is the Lambert function [46].

Proof *(Sketch - Detailed proof in the electronic appendix [42]). From Eq.* (B.10)*, we have:*

$$\widetilde{I}(X_S; Y_S) = 1 - \frac{H(X_S|Y_S)}{H(X_S)} \geq \mu_{\min} \tag{B.17}$$

$$\begin{aligned} \Rightarrow \frac{H(X_S|Y_S)}{H(X_S)} &= \frac{p(X_1, Y_1) \cdot \log p(X_1|Y_1)}{\sum_i p(X_i) \cdot \log p(X_i)} \\ &+ \frac{\sum_{i \neq 1 \land j \neq 1} p(X_i, Y_j) \cdot \log \frac{p(X_i, Y_j)}{p(Y_j)}}{\sum_i p(X_i) \cdot \log p(X_i)} \leq 1 - \mu_{\min} \end{aligned} \tag{B.18}$$

Let $\lambda_1 = p(X_k)$ such that $p(X_k) = \min\{p(X_i)\}, \forall i$, and $\lambda_2 = p(Y_1)$. We obtain:

$$\frac{H(X_S|Y_S)}{H(X_S)} \geq \frac{p(X_1, Y_1) \cdot \log \frac{p(X_1, Y_1)}{\lambda_2}}{\log \lambda_1} \tag{B.19}$$

From Eqs. (B.18), (B.19)*, the support lower bound of (X_1, Y_1) in $\mathcal{D}_{SYB}$ is derived as:*

$$supp(X_1, Y_1)_{\mathcal{D}_{SYB}} \geq \lambda_2 \cdot e^{W\left(\frac{\log \lambda_1^{1-\mu_{\min}} \cdot \ln 2}{\lambda_2}\right)} \tag{B.20}$$

Since:

$$supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq supp(X_1, Y_1)_{\mathcal{D}_{SYB}} \tag{B.21}$$

It follows that:

$$supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \lambda_2 \cdot e^{W\left(\frac{\log \lambda_1^{1-\mu_{\min}} \cdot \ln 2}{\lambda_2}\right)} \tag{B.22}$$

From Theorem 1, we can derive the minimum MI threshold $\mu_{\min}$ such that the support of (X_1, Y_1) is at least $\sigma_{\min}$.

Corollary 1.1 *The support of an event pair* $(X_1, Y_1) \in (X_S, Y_S)$ *in* $\mathcal{D}_{SEQ}$ *is at least* $\sigma_{\min}$ *if* $\widetilde{I}(X_S; Y_S)$ *is at least* $\mu_{\min}$*, where:*

$$\mu_{\min} \geq \begin{cases} 1 - \frac{\lambda_2}{e \cdot \ln 2 \cdot \log \frac{1}{\lambda_1}}, & \textit{if} \quad 0 \leq \frac{\sigma_{\min}}{\lambda_2} \leq \frac{1}{e} \\ 1 - \frac{\sigma_{\min} \cdot \log \frac{\sigma_{\min}}{\lambda_2}}{\ln 2 \cdot \log \lambda_1}, & \textit{otherwise} \end{cases} \tag{B.23}$$

Interpretation of the support lower bound: Given two symbolic series X_S and Y_S, and a minimum mutual information threshold $\mu_{\min}$. Theorem 1 says that, if X_S and Y_S are mutually dependent with the minimum MI value $\mu_{\min}$, then the support of an event pair in (X_S, Y_S) is at least the lower bound in Eq. (B.16). Combining Theorem 1 and Lemma 3, we can conclude that if an event pair of (X_S,Y_S) has a support less than the lower bound in Eq. (B.16), then any pattern P formed by that event pair also has support less than that lower bound. This allows us to construct an approximate version of GTPM (discussed in Section B.5.5).

B.5.3 Lower bound of the Confidence of an Event Pair

Consider two events X_1, Y_1 of two symbolic series X_S and Y_S. We derive the confidence lower bound of (X_1, Y_1) in the sequence database $\mathcal{D}_{\text{SEQ}}$ as follows.

Theorem 2 *(Lower bound of the confidence) Let* $\sigma_{\min}$ *and* $\mu_{\min}$ *be the minimum support and minimum mutual information thresholds, respectively. Assume that* $supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \sigma_{\min}$*. If the NMI* $\widetilde{I}(X_S;Y_S) \geq \mu_{\min}$*, then the lower bound of the confidence of* (X_1, Y_1) *in* $\mathcal{D}_{SEQ}$ *is:*

$$conf(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \sigma_{\min} \cdot \lambda_1^{\frac{1-\mu_{\min}}{\sigma_{\min}}} \cdot \left(\frac{n_x - 1}{1 - \sigma_{\min}}\right)^{\frac{\lambda_3}{\sigma_{\min}}} \tag{B.24}$$

where n_x *is the number of symbols in* Σ_X*,* λ_1 *is the minimum support of* $X_i \in X_S$*, and* λ_3 *is the support of* $(X_i, Y_j) \in (X_S, Y_S)$ *such that* $p(X_i|Y_j)$ *is minimal,* $\forall(i \neq 1 \wedge j \neq 1)$.

Proof *(Sketch - Detailed proof in the electronic appendix [42]). Let* $\lambda_1 = p(X_k)$ *such that* $p(X_k) = \min\{p(X_i)\}, \forall i$*, and* $\lambda_3 = p(X_m, Y_n)$ *such that* $p(X_m|Y_n) = \min\{p(X_i|Y_j)\}$ *,* $\forall(i \neq 1 \wedge j \neq 1)$*. Then, by applying the min-max inequality theorem for the sum of ratio [47] to the numerator of Eq.* (B.18)*, we obtain:*

$$\begin{aligned} \frac{H(X_S|Y_S)}{H(X_S)} &\geq \frac{p(X_1, Y_1) \cdot \log p(X_1|Y_1) + \lambda_3 \cdot \log \frac{1-p(X_1,Y_1)}{n_x - p(Y_1)}}{\log \lambda_1} \\ &\geq \frac{\sigma_{\min} \cdot \log \frac{p(X_1,Y_1)}{p(Y_1)} + \lambda_3 \cdot \log \frac{1-\sigma_{\min}}{n_x - 1}}{\log \lambda_1} \end{aligned} \tag{B.25}$$

Next, assume that $supp(Y_1)_{\mathcal{D}_{SYB}} \geq supp(X_1)_{\mathcal{D}_{SYB}}$*. From Eqs.* (B.18), (B.25)*, the confidence lower bound of* (X_1, Y_1) *in* $\mathcal{D}_{SYB}$ *is derived as:*

$$conf(X_1, Y_1)_{\mathcal{D}_{SYB}} = \frac{supp(X_1, Y_1)_{\mathcal{D}_{SYB}}}{supp(Y_1)_{\mathcal{D}_{SYB}}} \geq \lambda_1^{\frac{1-\mu_{\min}}{\sigma_{\min}}} \cdot \left(\frac{n_x - 1}{1 - \sigma_{\min}}\right)^{\frac{\lambda_3}{\sigma_{\min}}} \tag{B.26}$$

Since:

$$conf(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \sigma_{\min} \cdot conf(X_1, Y_1)_{\mathcal{D}_{SYB}} \tag{B.27}$$

It follows that:

$$conf(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \sigma_{\min} \cdot \lambda_1^{\frac{1-\mu_{\min}}{\sigma_{\min}}} \cdot \left(\frac{n_x - 1}{1 - \sigma_{\min}}\right)^{\frac{\lambda_3}{\sigma_{\min}}} \tag{B.28}$$

From Theorem 2, we can derive the minimum MI threshold $\mu_{\min}$ such that the confidence of (X_1, Y_1) is at least δ.

Corollary 2.1 *The confidence of an event pair* $(X_1, Y_1) \in (X_S, Y_S)$ *in* $\mathcal{D}_{SEQ}$ *is at least* δ *if* $\widetilde{I}(X_S; Y_S)$ *is at least* $\mu_{\min}$*, where:*

$$\mu_{\min} \geq 1 - \sigma_{\min} \cdot \log_{\lambda_1}\left(\frac{\delta}{\sigma_{\min}} \cdot \left(\frac{1 - \sigma_{\min}}{n_x - 1}\right)^{\frac{\lambda_3}{\sigma_{\min}}}\right) \tag{B.29}$$

Interpretation of the confidence lower bound: Given two symbolic series X_S and Y_S, and a minimum mutual information threshold $\mu_{\min}$. Theorem 2 says that, if X_S and Y_S are mutually dependent with the minimum MI value $\mu_{\min}$, then the confidence of an event pair in (X_S, Y_S) is at least the lower bound in Eq. (B.24). Combining Theorem 2 and Lemma 4, if an event pair of (X_S,Y_S) has a confidence less than the lower bound in Eq. (B.24), then any pattern P formed by that event pair also has a confidence less than that lower bound. This allows us to construct an approximate version of GTPM (discussed in Section B.5.5).

B.5.4 Upper Bound of the Support of an Event Pair

We derive the support upper bound of the event pair (X_1, Y_1) of X_S and Y_S in $\mathcal{D}_{\text{SEQ}}$ as follows.

Theorem 3 *(Upper bound of the support) Let* $\sigma_{\min}$ *be the minimum support threshold, and* $\mu_{\max}$ *be the maximum mutual information threshold, respectively. Assume that*

$supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \geq \sigma_{\min}$. If the NMI $\widetilde{I}(X_S;Y_S) \leq \mu_{\max}$, then the upper bound of the support of (X_1, Y_1) in $\mathcal{D}_{SEQ}$ is:

$$supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \leq \lambda_2 \cdot e^{W\left(\frac{\log \frac{\lambda_5^{1-\mu_{\max}}}{\lambda_4^{1-\sigma_{\min}}} \cdot \ln 2}{\lambda_2}\right)} + \vartheta \tag{B.30}$$

where: λ_2 is the support of $Y_1 \in Y_S$, λ_4 is the fraction between the support of $(X_i, Y_j) \in (X_S, Y_S)$ and the support of $Y_j \in Y_S$ such that $p(X_i|Y_j)$ is minimal, $\forall i \neq 1 \wedge j \neq 1$, λ_5 is the maximum support of $X_i \in X_S$, and ϑ is the difference between the probabilities of (X_1, Y_1) in $\mathcal{D}_{SEQ}$ and $\mathcal{D}_{SYB}$.

Proof *(Sketch - Detailed proof in the electronic appendix [42]). Let $\lambda_2 = p(Y_1)$, $\lambda_4 = \min\{p(X_i|Y_j)\}\ \forall(i \neq 1 \wedge j \neq 1)$, and $\lambda_5 = \max\{p(X_i)\}\ \forall i$. We obtain:*

$$\frac{H(X_S|Y_S)}{H(X_S)} \leq \frac{p(X_1, Y_1) \cdot \log \frac{p(X_1,Y_1)}{\lambda_2} + (1 - \sigma_{\min}) \cdot \log \lambda_4}{\log \lambda_5} \tag{B.31}$$

From Eqs. (B.10)*, we have:*

$$\widetilde{I}(X_S; Y_S) = 1 - \frac{H(X_S|Y_S)}{H(X_S)} \leq \mu_{\max} \Rightarrow \frac{H(X_S|Y_S)}{H(X_S)} \geq 1 - \mu_{\max} \tag{B.32}$$

From Eqs. (B.31) *and* (B.32)*, we have:*

$$\frac{p(X_1, Y_1) \cdot \log \frac{p(X_1,Y_1)}{\lambda_2} + (1 - \sigma_{\min}) \cdot \log \lambda_4}{\log \lambda_5} \geq 1 - \mu_{\max} \tag{B.33}$$

$$\Leftrightarrow p(X_1, Y_1) \leq \lambda_2 \cdot e^{W\left(\frac{\log \frac{\lambda_5^{1-\mu_{\max}}}{\lambda_4^{1-\sigma_{\min}}} \cdot \ln 2}{\lambda_2}\right)} \tag{B.34}$$

From Eq. (B.14)*, we have:*

$$p(X_1, Y_1) = supp(X_1, Y_1)_{\mathcal{D}_{SYB}} = supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} - \vartheta \tag{B.35}$$

From Eqs. (B.34) *and* (B.35)*, we have:*

$$supp(X_1, Y_1)_{\mathcal{D}_{SEQ}} \leq \lambda_2 \cdot e^{W\left(\frac{\log \frac{\lambda_5^{1-\mu_{\max}}}{\lambda_4^{1-\sigma_{\min}}} \cdot \ln 2}{\lambda_2}\right)} + \vartheta \tag{B.36}$$

From Theorem 3, we can derive the maximum MI threshold $\mu_{\max}$ such that the support of (X_1, Y_1) is at most $\sigma_{\max}$.

Corollary 3.1 *The support of an event pair* $(X_1, Y_1) \in (X_S, Y_S)$ *in* $\mathcal{D}_{SEQ}$ *is at most* $\sigma_{\max}$ *if* $\widetilde{I}(X_S; Y_S)$ *is at most* $\mu_{\max}$*, where:*

$$\mu_{\max} \leq 1 - \frac{\frac{\sigma_{\max}-\vartheta}{\lambda_2} \cdot \log \frac{\sigma_{\max}-\vartheta}{\lambda_2} + \log \lambda_4^{1-\sigma_{\min}}}{\log \lambda_5} \tag{B.37}$$

Interpretation of the support upper bound: Given a maximum MI threshold $\mu_{\max}$, let X_S and Y_S be two symbolic series. Theorem 3 says that, if the NMI of X_S and Y_S is at most $\mu_{\max}$, then the support of an event pair in (X_S, Y_S) is at most the upper bound in Eq. (B.30). Combining Theorem 3 and Lemma 3, we can conclude that if an event pair in (X_S, Y_S) has a support less than the upper bound, then any pattern P formed by that event pair also has support less than that upper bound.

Setting the values of $\mu_{\min}$ and $\mu_{\max}$: GTPM uses three user-defined parameters, the minimum support $\sigma_{\min}$, the maximum support $\sigma_{\max}$, and the minimum confidence δ to mine both frequent and rare temporal patterns (with $\sigma_{\max}$ is set to ∞ in case of frequent patterns). To mine frequent patterns that satisfy both $\sigma_{\min}$ and δ constraints, we select $\mu_{\min}$ such that both Eqs. (B.23) and (B.29) hold, i.e., the maximum value of $\mu_{\min}$ provided by the two equations. On the other hand, to mine rare patterns that also have to satisfy $\sigma_{\max}$ constraint, $\mu_{\max}$ is chosen using Eq. (B.37).

Algorithm 10: Approximate GTPM using MI

Input: A set of time series $\mathcal{X}$, a minimum support threshold $\sigma_{\min}$, a maximum support threshold $\sigma_{\max}$, a minimum confidence threshold δ

Output: The set of temporal patterns P

Convert $\mathcal{X}$ to $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$;
Scan $\mathcal{D}_{\text{SYB}}$ to compute the probability of each event, event pair, and plus ϑ value;
foreach *pair of symbolic time series* $(X_S, Y_S) \in \mathcal{D}_{SYB}$ **do**
    Compute $\widetilde{I}(X_S; Y_S)$ and $\widetilde{I}(Y_S; X_S)$;
    Compute $\mu_{\min}$ using Eqs. (B.23) and (B.29);
    Compute $\mu_{\max}$ using Eqs. (B.37);
    if $\min\{\widetilde{I}(X_S; Y_S), \widetilde{I}(Y_S; X_S)\} \geq \mu_{\min}$ **then**
        if $\min\{\widetilde{I}(X_S; Y_S), \widetilde{I}(Y_S; X_S)\} \leq \mu_{\max}$ **then**
            Insert X_S and Y_S into X_C;
foreach $X_S \in \mathsf{X}_C$ **do**
    Mine single events from X_S as in Section B.4.3;
foreach $(X_S, Y_S) \in \mathsf{X}_C$ **do**
    Mine 2-event patterns from (X_S, Y_S) as in Section B.4.4;
if $k \geq 3$ **then**
    Mine k-event patterns similar to the exact GTPM in Section B.4.5;

B.5.5 Using the Bounds for Approximate GTPM

Approximate GTPM: Approximate GTPM is based on the exact GTPM and performs the mining only on *the set of mutually dependent symbolic series* $\mathsf{X}_C \in \mathcal{X}$ *with minimum and maximum MI thresholds* $\mu_{\min}$ *and* $\mu_{\max}$. Algorithm 10 describes the approximate GTPM. First, $\mathcal{D}_{\text{SYB}}$ is scanned once to compute the probability of each single event, pair of events, and plus ϑ value (line 2). Next, NMI, $\mu_{\min}$, and $\mu_{\max}$ are computed for each symbolic series pair (lines 4-6). The pairs of symbolic series whose $\min\{\widetilde{I}(X_S;Y_S),\widetilde{I}(Y_S;X_S)\}$ is at least $\mu_{\min}$, and $\min\{\widetilde{I}(X_S;Y_S),\widetilde{I}(Y_S;X_S)\}$ is at most $\mu_{\max}$ are inserted into X_C (lines 7-9). Then, we traverse each series in X_C to mine the single events (lines 10-11). Next, each event pair in corresponding series in X_C is employed to mine the 2-event patterns (lines 12-13). For k-event pattern ($k \geq 3$), the mining process is similar to GTPM (lines 14-15).

Complexity analysis of Approximate GTPM: To compute NMI, $\mu_{\min}$, and $\mu_{\max}$, we only have to scan $\mathcal{D}_{\text{SYB}}$ once to calculate the probability for each single event, pair of events, and plus ϑ value. Thus, the cost of NMI, $\mu_{\min}$, and $\mu_{\max}$ computations is $|\mathcal{D}_{\text{SYB}}|$. On the other hand, the complexity of the exact GTPM at HLH_1 and HLH_2 are $O(m^2 i^2\ |\mathcal{D}_{\text{SEQ}}|^2) + O(m\cdot|\mathcal{D}_{\text{SEQ}}|)$ (Sections B.4.3 and B.4.4). Thus, the approximate GTPM is significantly faster than the exact GTPM.

B.6 Experimental Evaluation

We evaluate GTPM in two different settings: to mine rare temporal patterns, named as RTPM, and to mine frequent temporal patterns, named as FTPM. Note that for RTPM, all three constraints $\sigma_{\min}$, $\sigma_{\max}$ and δ are used, whereas for FTPM, only $\sigma_{\min}$ and δ are used. In each setting, the performance of both exact and approximate versions are assessed. We use real-world datasets from four application domains: smart energy, smart city, sign language, and health. Due to space limitations, we only present here the most important results, and discuss other findings in the electronic appendix [42].

B.6.1 Experimental Setup

Datasets: We use three *smart energy* (SE) datasets, NIST [48], UKDALE [49], and DataPort [50] that measure the energy consumption of electrical appliances in residential households. For the *smart city* (SC), we use weather and vehicle collision data obtained from NYC Open Data Portal [51]. For *sign language*, we use the American Sign Language (ASL) datasets [52] containing annotated video sequences of different ASL signs and gestures. For *health*, we combine the *influenza* (INF) dataset [53] and weather data [54] from Kawasaki, Japan. Table B.4 summarizes their characteristics.

Table B.4: Characteristics of the Datasets

	NIST	UKDALE	DataPort	SC	ASL	INF
# sequences	1460	1520	1460	1216	1908	608
# variables	49	24	21	26	25	25
# distinct events	98	48	42	130	173	124
# instances/seq.	55	190	49	162	20	48

Table B.5: Parameters and values

Params	Values
Minimum support σ_{min}	User-defined: σ_{min} = 0.2%, 0.4%, 0.6% 1%, 3% ...
Maximum support σ_{max}	User-defined: σ_{max} = 2%, 6%, 10%, 15%, 20%, ...
Minimum confidence δ	User-defined: δ = 40%, 50%, 60%, 70%, 80%, ...
Overlapping duration t_{ov}	User-defined: t_{ov} (hours) = 0, 1, 2, 3 (NIST, UKDALE, DataPort, SC) t_{ov} (frames) = 0, 150, 300, 450 (ASL) t_{ov} (days) = 0, 7, 10, 14 (INF)
Tolerance buffer ϵ	User-defined: ϵ (mins) = 0, 1, 2, 3 (NIST, UKDALE, DataPort) ϵ (mins) = 0, 5, 10, 15 (SC) ϵ (frames) = 0, 30, 45, 60 (ASL) ϵ (days) = 0, 1, 2, 3 (INF)

Baseline methods: Our exact RTPM version is referred to as E-RTPM, and the approximate one as A-RTPM. Since our work is the first that studies rare temporal pattern mining, there is not an exact baseline to compare against RTPM. However, we adapt the state-of-the-art method for frequent temporal pattern mining Z-Miner [22] to find rare temporal patterns. The Adapted Rare Z-Miner is referred to as ARZ-Miner. Similarly, we denote the exact FTPM version as E-FTPM, and the approximate one as A-FTPM. We use 4 baselines (detailed in Section B.2) to compare with our FTPM: Z-Miner [22], TPMiner [3], IEMiner [4], and H-DFS [5]. Since the exact versions (E-RTPM and E-FTPM) and the baselines provide the same exact solutions, we use the baselines only for quantitative evaluation.

Infrastructure: We use a VM with 32 AMD EPYC cores (2GHz), 512 GB RAM, and 1 TB storage.

Parameters: Table B.5 lists the parameters and their values used in our experiments.

B.6.2 Qualitative Evaluation

Rare temporal patterns: Table B.6 shows several interesting rare temporal patterns extracted by RTPM. Patterns P1-P5 are from SC and P6-P8 are from INF. Analyzing these patterns can reveal some rare but interesting relations between temporal events. For example, P1-P5 show there exists an association between extreme weather conditions and high accident numbers, such as high pedestrian injury during a heavy snowing day, which is very important to act on even though it occurs rarely.

Table B.6: Summary of Interesting Rare Patterns

Patterns	$\sigma_{\min}$ **(%)**	δ **(%)**	$\sigma_{\max}$ **(%)**
(P1) Heavy Rain $\succcurlyeq$ Unclear Visibility $\succcurlyeq$ Overcast Cloudiness $\rightarrow$ High Motorist Injury	5	30	9
(P2) Heavy Rain $\between$ Strong Wind $\rightarrow$ High Motorist Injury	2	40	6
(P3) Very Strong Wind $\rightarrow$ High Motorist Injury	5	40	9
(P4) Strong Wind $\between$ High Pedestrian Injury	4	30	8
(P5) Extremely Unclear Visibility $\succcurlyeq$ High Snow $\succcurlyeq$ High Pedestrian Injury	3	45	7
(P6) Frost Temperature $\between$ High Snow $\succcurlyeq$ High Influenza	1	42	6
(P7) Low Temperature $\succcurlyeq$High Influenza	1	42	6
(P8) Heavy Rain $\succcurlyeq$ High Influenza	3	35	8

Frequent temporal patterns: Table B.7 lists some interesting frequent temporal patterns extracted by FTPM. Patterns P9-P15 are from SEs and P16-P18 are from ASL. Analyzing these patterns will reveal useful information about the domains. For example, P9-P15 show how the residents interact with electrical appliances in their houses. Specifically, P9 shows that a resident turns on the light upstairs in the early morning, and goes to the bathroom. Then, within a minute later, the microwave in the kitchen is turned on. This pattern occurs with minimum support of 20%, reflecting a living habit of the residents. Moreover, P9 also implies that there might be more than one person living in the house, in which one resident is in the bathroom while the other is downstairs preparing breakfast.

B.6.3 Quantitative Evaluation of RTPM

RTPM: Baseline comparison on real world datasets

We compare E-RTPM and A-RTPM with the adapted baseline ARZ-Miner in terms of runtime and memory usage. Figs. B.7, B.8, B.9, and B.10 show the comparison results on NIST and SC.

As shown in Figs. B.7 and B.8, A-RTPM achieves the best runtime among all methods, and E-RTPM has better runtime than the baseline. The range and average speedups of A-RTPM compared to other methods are: [1.9-7.2] and 3.4 (E-RTPM), [5.4-48.9] and 16.5 (ARZ-Miner). The speedup of E-RTPM compared to the baseline is [2.9-24.7] and 7.4 on average. Note that the time to compute MI, $\mu_{\min}$, and $\mu_{\max}$ for NIST and SC in Figs. B.7 and B.8 are 35.4 and 28.7 seconds, respectively, i.e., negligible compared to the total runtime.

Table B.7: Summary of Interesting Frequent Patterns

Patterns	σ_{min} (%)	δ (%)
(P9) ([05:58, 08:24] First Floor Lights) $\succcurlyeq$ ([05:58, 06:59] Upstairs Bathroom Lights) $\succcurlyeq$ ([05:59, 06:06] Microwave)	20	30
(P10) ([18:00, 18:30] Lights Dining Room) $\rightarrow$ ([18:31, 20:16] Children Room Plugs) $\between$ ([19:00, 22:31] Lights Living Room)	20	20
(P11) ([15:59, 16:05] Hallway Lights) $\rightarrow$ ([17:58, 18:29] Kitchen Lights $\succcurlyeq$ ([18:00, 18:18] Plug In Kitchen) $\succcurlyeq$ ([18:08, 18:15] Microwave)	20	25
(P12) ([06:02, 06:19] Kitchen Lights) $\rightarrow$ ([06:05, 06:12] Microwave) $\between$ ([06:09, 06:11] Kettle)	20	35
(P13) ([16:45, 17:30] Washer) $\rightarrow$ ([17:40,18:55] Dryer) $\rightarrow$ ([19:05, 20:10] Dining Room Lights) $\succcurlyeq$ ([19:10, 19:30] Cooktop)	10	30
(P14) ([06:10, 07:00] Kitchen Lights) $\succcurlyeq$ ([06:10, 06:15] Kettle) $\rightarrow$ ([06:30, 06:40] Toaster) $\rightarrow$ ([06:45, 06:48] Microwave)	25	40
(P15) ([18:00, 18:25] Kitchen Lights) $\succcurlyeq$ ([18:00, 18:05] Kettle) $\rightarrow$ ([18:05, 18:10] Microwave) $\rightarrow$ ([19:35, 20:50] Washer)	20	40
(P16) [2.12 seconds] Negation $\succcurlyeq$ [0.27 seconds] Lowered Eye-brows	10	10
(P17) [2.04 seconds] Negation $\succcurlyeq$ [0.52 seconds] Rapid Shake-head	10	10
(P18) [1.53 seconds] Wh-question $\succcurlyeq$ [0.36 seconds] Lowered Eye-brows $\rightarrow$ [0.05 seconds] Blinking Eye-aperture	10	15

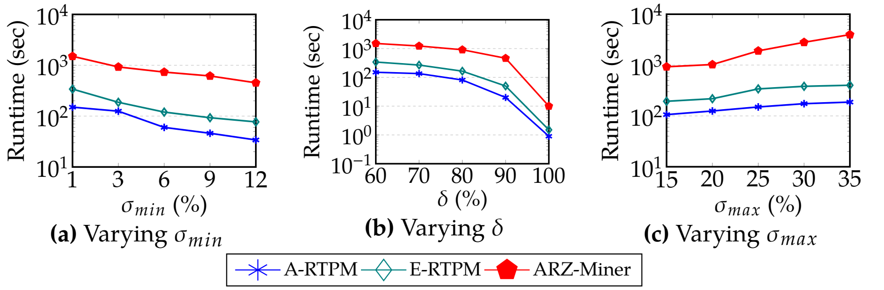


Fig. B.7: RTPM-Runtime Comparison on NIST (real-world)

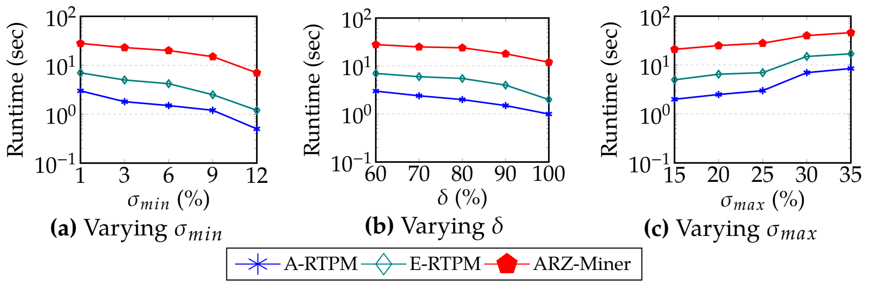


Fig. B.8: RTPM-Runtime Comparison on SC (real-world)

In terms of memory consumption, as shown in Figs. B.9 and B.10, A-RTPM uses the least memory, while E-RTPM uses less memory than the baseline. A-RTPM consumes [1.6-3.9] (on average 2.1) times less memory than E-RTPM, and [7.2-120.6] (on average 24.1) times less than ARZ-Miner. E-RTPM uses [4.6-61.8] (on average 14.7) times less memory than ARZ-Miner.

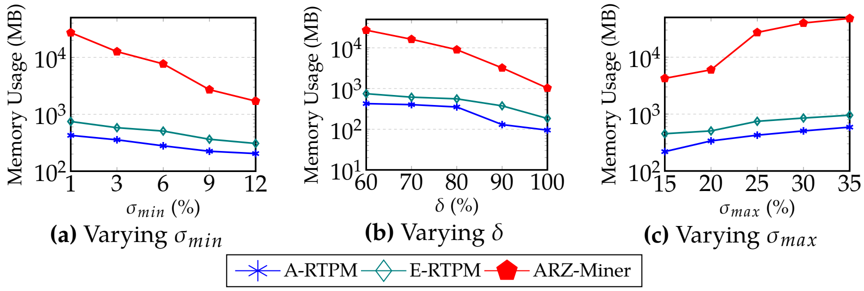


Fig. B.9: RTPM-Memory Usage Comparison on NIST (real-world)

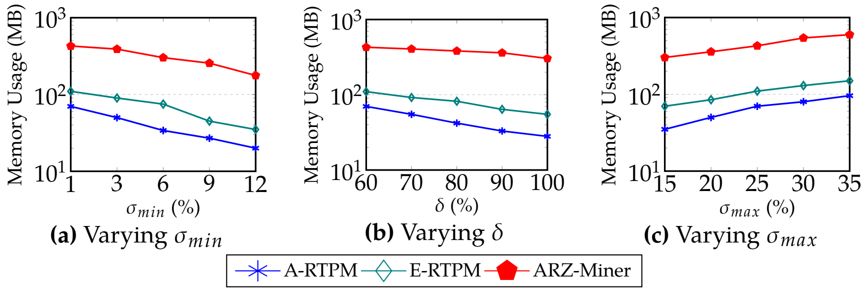


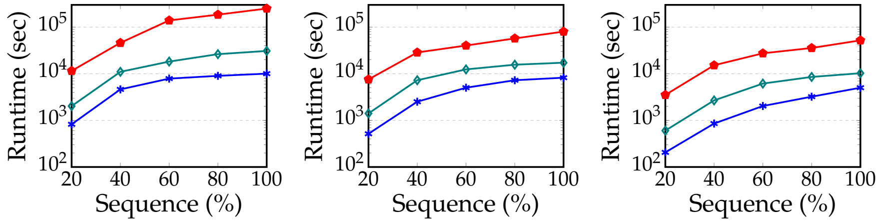
Fig. B.10: RTPM-Memory Usage Comparison on SC (real-world)

RTPM: Scalability evaluation on synthetic datasets

As discussed in Section B.4, the complexity of GTPM in general (and RTPM in particular) is driven by two main factors: (1) the number of temporal sequences, and (2) the number of time series. The evaluation on real-world datasets has shown that E-RTPM and A-RTPM outperform the baseline significantly in both runtimes and memory usage. However, to further assess the scalability of RTPM, we scale these two factors using synthetic datasets. Specifically, starting from the real-world datasets, we generate 10 times more sequences, and create up to 1000 synthetic time series. We then evaluate the scalability of RTPM in two scenarios: varying the number of sequences, and varying the number of time series.

Figs. B.11 and B.12 show the runtimes of A-RTPM, E-RTPM and the baseline when the number of sequences changes. We can see that A-RTPM and E-RTPM outperform and scale better than the baseline in this configuration. The range and average speedups of A-RTPM w.r.t. other methods are: [2.3-5.7] and 3.2 (E-RTPM), [5.1-19.8] and 12.5 (ARZ-Miner). Similarly, the range and average speedups of E-RTPM compared to ARZ-Miner are [2.7-7.6] and 5.3.

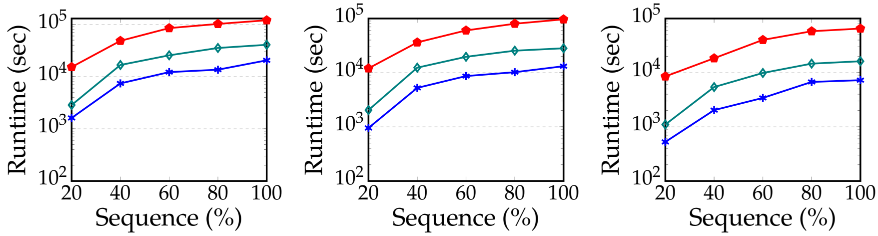
Figs. B.13 and B.14 compare the runtimes of A-RTPM with other methods when changing the number of time series. It is seen that, A-RTPM achieves highest speedup in this configuration. The range and average speedups of A-RTPM are [3.5-7.4] and 4.6 (E-RTPM), [7.2-24.8] and 15.2 (ARZ-Miner), and



(a) σ_{min}=1%, σ_{max}=20%, δ=60%
(b) σ_{min}=3%, σ_{max}=15%, δ=70%
(c) σ_{min}=6%, σ_{max}=10%, δ=80%

A-RTPM — E-RTPM — ARZ-Miner

Fig. B.11: RTPM-Varying % of sequences on NIST (synthetic)



(a) σ_{min}=1%, σ_{max}=20%, δ=60%
(b) σ_{min}=3%, σ_{max}=15%, δ=70%
(c) σ_{min}=6%, σ_{max}=10%, δ=80%

A-RTPM — E-RTPM — ARZ-Miner

Fig. B.12: RTPM-Varying % of sequences on SC (synthetic)

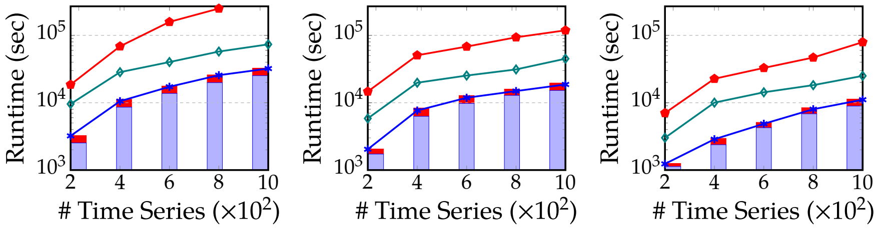
of E-RTPM is [3.6-9.5] and 6.4 (ARZ-Miner).

On average, E-RTPM consumes 17.2 times less memory than the baseline, while A-RTPM uses 20.6 times less memory than E-RTPM and the baseline in the scalability study. Furthermore, Fig. B.13a shows that A-RTPM and E-RTPM can scale well on big datasets while the baseline cannot. Specifically, the baseline fails for large configurations as it runs out of memory, e.g., when # Time Series ≥ 1000 on the synthetic NIST. We add an additional bar chart for A-RTPM, including the time to compute MI, $\mu_{\min}$, and $\mu_{\max}$ (top red) and the mining time (bottom blue) for comparison, showing that this time is negligible.

Table B.8: Pruned Time Series and Events from A-RTPM

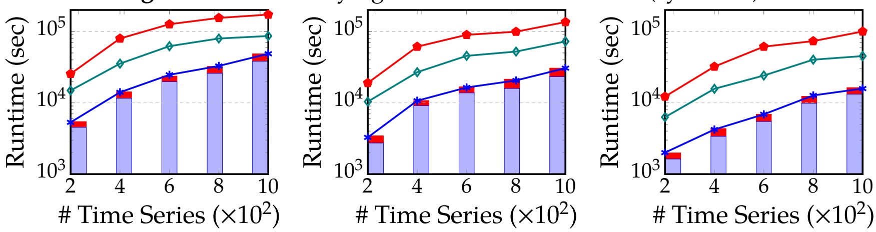
# Attr.	$\sigma_{\min}$ (%) - δ (%) - $\sigma_{\max}$ (%)								
	NIST			SC					
	Pruned Time Series / Events (%)			Pruned Time Series (%)			Pruned Events (%)		
	6-80-20	3-70-15	1-60-10	6-80-20	3-70-15	1-60-10	6-80-20	3-70-15	1-60-10
200	59.50	39.50	22.50	48.50	30.50	15.50	39.10	25.10	11.90
400	58.50	38.25	21.25	45.75	29.75	14.75	37.55	24.30	11.45
600	56.50	36.17	19.83	43.17	27.17	14.33	36.43	23.03	10.57
800	51.63	35.88	19.63	42.38	23.88	14.25	33.55	21.28	10.30
1000	49.70	34.10	19.40	41.30	22.70	13.80	32.94	20.14	9.96

Finally, the percentage of time series and events pruned by A-RTPM in the scalability test are provided in Table B.8. Note that for the NIST dataset, every



(a) σ_{min}=1%, σ_{max}=20%, δ=60%
(b) σ_{min}=3%, σ_{max}=15%, δ=70%
(c) σ_{min}=6%, σ_{max}=10%, δ=80%

Fig. B.13: RTPM-Varying # of time series on NIST (synthetic)



(a) σ_{min}=1%, σ_{max}=20%, δ=60%
(b) σ_{min}=3%, σ_{max}=15%, δ=70%
(c) σ_{min}=6%, σ_{max}=10%, δ=80%

Fig. B.14: RTPM-Varying # of time series on SC (synthetic)

time series has two events, On and Off. Thus, the percentage of pruned time series and the percentage of pruned events are the same in NIST. We can see that the higher σ_{min}, δ, and σ_{max}, the more time series (events) are pruned. This is because higher σ_{min} and δ result in higher μ_{min}, and higher σ_{max} results in lower μ_{max}, and thus, more pruned time series.

E-RTPM: Evaluation of different pruning techniques

We evaluate the following combinations of E-RTPM pruning techniques: (1) NoPrune: E-RTPM with no pruning, (2) Apriori: E-RTPM with Apriori-based pruning (Lemmas 3, 4), (3) Trans: E-RTPM with transitivity-based pruning (Lemmas 5, 6, 7, 8), and (4) All: E-RTPM applied both pruning techniques.

We use 3 different scenarios that vary: the minimum support, the minimum confidence, and the maximum support. Figs. B.15, B.16 show the results. We see that (All)-E-RTPM has the best performance of all versions, with a speedup over (NoPrune)-E-RTPM ranging from 15 up to 74, depending on the configurations. Thus, the proposed prunings are very effective in improving E-RTPM performance. Furthermore, (Trans)-E-RTPM delivers a larger speedup than (Apriori)-E-RTPM, with the average speedup between 12 and 28 for (Trans)-E-RTPM, and between 7 and 19 for (Apriori)-E-RTPM, but applying both yields

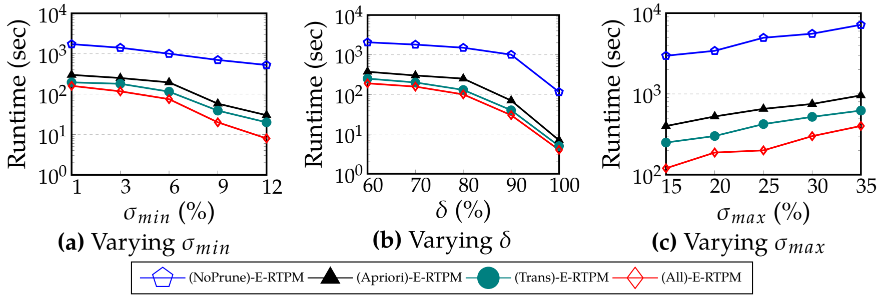


Fig. B.15: Runtimes of E-RTPM on NIST (real-world)

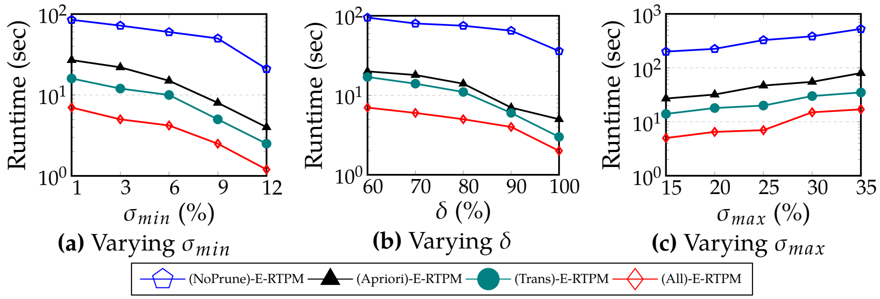


Fig. B.16: Runtimes of E-RTPM on SC (real-world)

the best speedup.

A-RTPM: Evaluation of accuracy

To evaluate A-RTPM accuracy, we compare the patterns extracted by A-RTPM and E-RTPM. Table B.9 shows the accuracies of A-RTPM for different σ_{min}, δ, and σ_{max} on the real world datasets. It is seen that A-RTPM obtains high accuracy ($\geq 83\%$) with lowest σ_{min} and δ, and highest σ_{max}, e.g., $\sigma_{\text{min}} = 1\%, \delta = 60\%$, $\sigma_{\text{max}} = 20\%$, and very high accuracy ($\geq 93\%$) with higher σ_{min} and δ, and lower σ_{max}, e.g., $\sigma_{\text{min}} = 3\%$, $\delta = 70\%$, $\sigma_{\text{max}} = 10\%$.

Table B.9: RTPM Accuracy (%)

σ_{max} (%)	**σ_{min} (%) - δ (%)**					
	NIST			**SC**		
	1-60	**3-70**	**6-80**	**1-60**	**3-70**	**6-80**
10	93	96	100	91	93	100
15	86	92	95	86	91	100
20	84	92	92	83	87	90

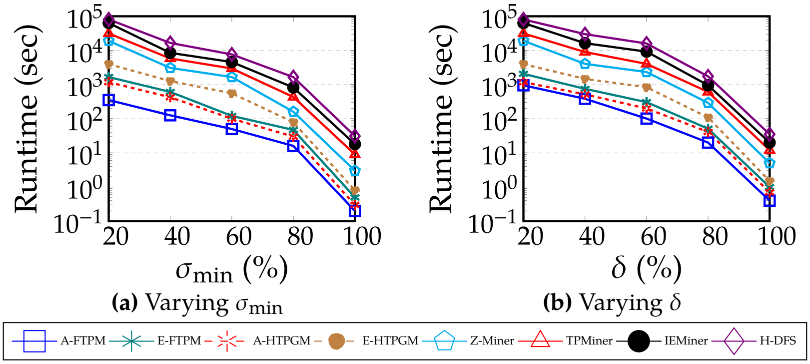


Fig. B.17: FTPM-Runtime Comparison on NIST (real-world)

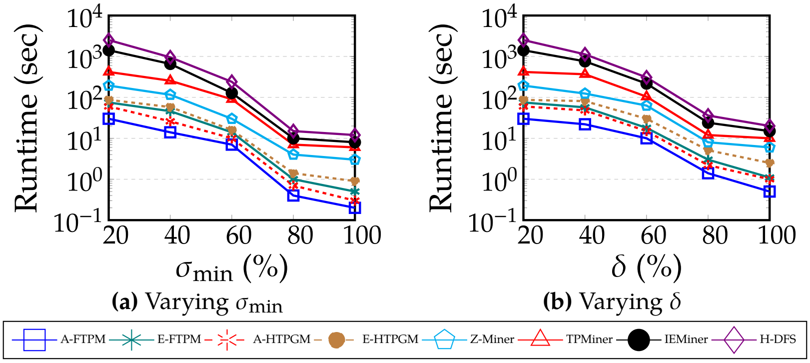


Fig. B.18: FTPM-Runtime Comparison on SC (real-world)

B.6.4 Quantitative Evaluation of FTPM

FTPM: Baselines comparison on real world datasets

We compare E-FTPM and A-FTPM against the baselines in terms of runtime and memory usage. Further, we also compare E-FTPM and A-FTPM against E-HTPGM and A-HTPGM from the conference version [13] to assess the performance improvement obtained by using the new data structure. Figs. B.17, B.18, B.19, and B.20 show the experimental results on NIST and SC.

We can see from Figs. B.17 and B.18 that A-FTPM achieves the fastest runtime among all methods, and E-FTPM has faster runtime than the baselines. On the tested datasets, the range and average speedups of A-FTPM compared to E-FTPM is [1.5-6.1] and 2.7, and compared to the baselines is [4.2-356.1] and 45.8. The range and average speedup of E-FTPM compared to the baselines is [2.6-130.4] and 24.7.

Note that the time to compute MI and $\mu_{\min}$ for NIST and SC datasets in Figs. B.17 and B.18 are 32.6 and 26.4 seconds, respectively, making it negligible in the total runtime. Moreover, by using the improved hierarchical hash table

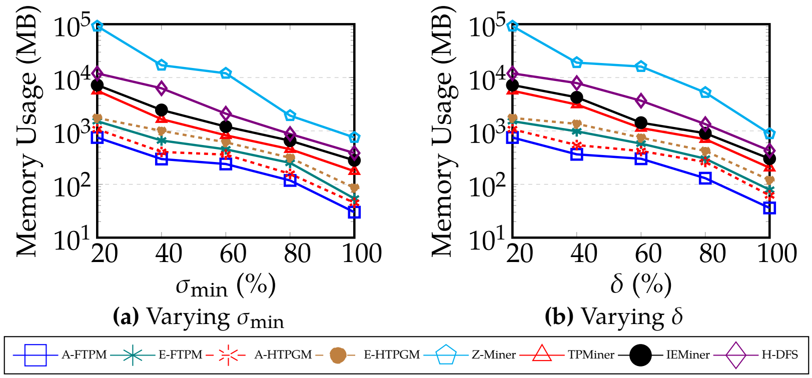


Fig. B.19: FTPM-Memory Usage Comparison on NIST (real-world)

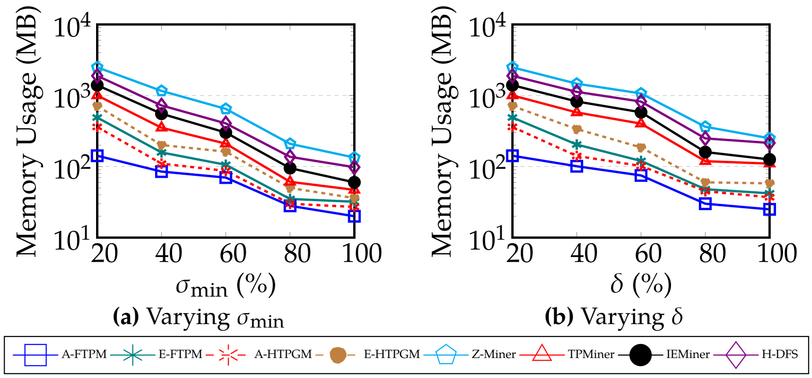


Fig. B.20: FTPM-Memory Usage Comparison on SC (real-world)

instead of the hierarchical pattern tree in [13], both E-FTPM and A-FTPM are more efficient than E-HTPGM and A-HTPGM. The speedup of E-FTPM over E-HTPGM is in the range [1.1-4.7], and A-FTPM over A-HTPGM is in the range [1.3-5.6].

Finally, A-FTPM is most efficient, i.e., achieves highest speedup and memory saving, when the support threshold is low, e.g., $\sigma_{\min} = 20\%$. This is because typical datasets often contain many patterns with very low support and confidence. Thus, using A-FTPM to prune uncorrelated series early helps save computational time and resources. However, the speedup comes at the cost of a small loss in accuracy.

In terms of memory consumption, as shown in Figs. B.19 and B.20, A-FTPM uses the least memory, while E-FTPM uses less memory than the baselines. A-FTPM consumes [1.4-3.6] (on average 1.9) times less memory than E-FTPM, and [6.8-112.6] (on average 15.4) times less than the baselines. E-FTPM uses [4.1-58.2] (on average 5.8) times less memory than the baselines. Compared to E-HTPGM and A-HTPGM [13], E-FTPM and A-FTPM are both more memory efficient. E-FTPM consumes [1.1-2.8] times less memory than E-HTPGM, while A-FTPM uses [1.2-3.1] times less memory than A-HTPGM.

We also perform other experiments on FTPM, including scalability evaluation on synthetic datasets, and evaluation of different pruning techniques on real-world datasets as in RTPM. These experiments are reported in the electronic appendix [42].

Table B.10: The Accuracy of A-FTPM (%)

σ_{min} (%)	δ (%)							
	NIST				**SC**			
	10	**20**	**50**	**80**	**10**	**20**	**50**	**80**
10	87	89	91	94	78	83	98	100
20	96	89	91	94	83	83	98	100
50	100	100	96	94	99	99	98	100
80	100	100	100	100	100	100	100	100

A-FTPM: Evaluation of the accuracy

We proceed to evaluate the accuracy of A-FTPM by comparing the patterns extracted by A-FTPM and E-FTPM. Table B.10 shows the accuracies of A-FTPM for different support and confidence thresholds on the real-world datasets. It is seen that A-FTPM obtains high accuracy (≥ 78%) when σ_{min} and δ are low, e.g., $\sigma_{\text{min}} = \delta = 10\%$, and very high accuracy (≥ 95%) when σ_{min} and δ are high, e.g., $\sigma_{\text{min}} = \delta = 50\%$.

Other experiments: We analyze the effects of the tolerance buffer ϵ, and the overlapping duration t_{ov} to the quality of extracted patterns. The analysis can be found in the electronic appendix [42].

B.7 Conclusion

This paper presents our comprehensive Generalized Frequent Temporal Pattern Mining from Time Series (GTPMfTS) solution that offers: (1) an end-to-end GTPMfTS process to mine both rare and frequent temporal patterns from time series, (2) an efficient and exact Generalized Temporal Pattern Mining (GTPM) algorithm that employs efficient data structures and multiple pruning techniques to achieve fast mining, and (3) an approximate GTPM that uses mutual information to prune unpromising time series, allows GTPM to scale on big datasets. Extensive experiments conducted on real world and synthetic datasets for rare temporal pattern mining (RTPM) and frequent temporal pattern mining (FTPM) show that both exact and approximate algorithms for RTPM and FTPM outperform the baselines, consume less memory, and scale well on big datasets. Compared to the baselines, the approximate A-RTPM is up to an order of magnitude speedup and the approximate A-FTPM delivers two orders of magnitude speedup. In future work, we plan to extend GTPM to prune at the event level to further improve their performance.

References

[1] Energinet. (2021) Energi data portal. [Online]. Available: https://www.energidataservice.dk/tso-electricity/co2emis/

[2] K. Torp, O. Andersen, and C. Thomsen, "Travel-time computation based on gps data," in *Big Data Management and Analytics: 9th European Summer School*. Springer, 2020.

[3] Y. Chen, W. Peng, and S. Lee, "Mining temporal patterns in time interval-based data," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 27, 2015.

[4] D. Patel, W. Hsu, and M. L. Lee, "Mining relationships among interval-based events for classification," in *Proceedings of the ACM SIGMOD international conference on Management of data*, 2008.

[5] P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos, "Mining frequent arrangements of temporal intervals," *Knowledge and Information Systems (KAIS)*, vol. 21, 2009.

[6] Y. Li and S. Cai, "Detecting outliers in data streams based on minimum rare pattern mining and pattern matching," *Information Technology and Control*, vol. 51, no. 2, 2022.

[7] Y. Cui, W. Gan, H. Lin, and W. Zheng, "Fri-miner: fuzzy rare itemset mining," *Applied Intelligence*, 2022.

[8] Y. Ji and Y. Ohsawa, "Mining frequent and rare itemsets with weighted supports using additive neural itemset embedding," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021.

[9] S. Cai, J. Chen, H. Chen, C. Zhang, Q. Li, R. N. A. Sosu, and S. Yin, "An efficient anomaly detection method for uncertain data based on minimal rare patterns with the consideration of anti-monotonic constraints," *Information Sciences*, vol. 580, 2021.

[10] A. Rahman, "Rare sequential pattern mining of critical infrastructure control logs for anomaly detection," Ph.D. dissertation, Queensland University of Technology, 2019.

[11] M. Iqbal, C. P. Wulandari, W. Yunanto, and G. I. P. Sari, "Mining non-zero-rare sequential patterns on activity recognition," *Jurnal Matematika MANTIK*, vol. 5, no. 1, 2019.

[12] A. Samet, T. Guyet, and B. Negrevergne, "Mining rare sequential patterns with asp," in *International Conference on Inductive Logic Programming*, 2017.

[13] V. L. Ho, N. Ho, and T. B. Pedersen, "Efficient temporal pattern mining in big time series using mutual information," vol. 15, no. 3. VLDB Endowment, 2022.

[14] P.-s. Kam and A. W.-C. Fu, "Discovering temporal patterns for interval-based events," in *Data Warehousing and Knowledge Discovery (DaWak)*, 2000.

[15] S.-Y. Wu and Y.-L. Chen, "Mining nonambiguous temporal patterns for interval-based events," *EEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 19, 2007.

[16] R. Moskovitch and Y. Shahar, "Fast time intervals mining using the transitivity of temporal relations," *Knowledge and Information Systems*, vol. 42, 2015.

[17] I. Batal, D. Fradkin, J. Harrison, F. Moerchen, and M. Hauskrecht, "Mining recent temporal patterns for event detection in multivariate time series data," in *Proceedings of ACM SIGKDD international conference on Knowledge discovery and data mining*, 2012.

[18] J.-Z. Wang, Y.-C. Chen, W.-Y. Shih, L. Yang, Y.-S. Liu, and J.-L. Huang, "Mining high-utility temporal patterns on time interval–based data," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 11, no. 4, 2020.

[19] A. K. Sharma and D. Patel, "Stipa: A memory efficient technique for interval pattern discovery," in *IEEE International Conference on Big Data (Big Data)*. IEEE, 2018.

[20] I. Batal, H. Valizadegan, G. F. Cooper, and M. Hauskrecht, "A temporal pattern mining approach for classifying electronic health record data," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 4, 2013.

[21] E. A. Campbell, E. J. Bass, and A. J. Masino, "Temporal condition pattern mining in large, sparse electronic health record data: A case study in characterizing pediatric asthma," *Journal of the American Medical Informatics Association*, vol. 27, 2020.

[22] Z. Lee, T. Lindgren, and P. Papapetrou, "Z-miner: an efficient method for mining frequent arrangements of event intervals," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.

[23] Y. Gao and J. Lin, "Efficient discovery of time series motifs with large length range in million scale time series," in *IEEE International Conference on Data Mining (ICDM)*. IEEE, 2017.

[24] N. Begum and E. Keogh, "Rare time series motif discovery from unbounded streams," *Proceedings of the VLDB Endowment*, vol. 8, no. 2, 2014.

[25] E. Alipourchavary, S. M. Erfani, and C. Leckie, "Mining rare recurring events in network traffic using second order contrast patterns," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2021.

[26] S. Bouasker, W. Inoubli, S. B. Yahia, and G. Diallo, "Pregnancy associated breast cancer gene expressions: new insights on their regulation based on rare correlated patterns," *Transactions on Computational Biology and Bioinformatics*, vol. 18, no. 3, 2020.

[27] A. Borah and B. Nath, "Rare association rule mining from incremental databases," *Pattern Analysis and Applications*, vol. 23, 2020.

[28] P. Fournier-Viger, P. Yang, Z. Li, J. C.-W. Lin, and R. U. Kiran, "Discovering rare correlated periodic patterns in multiple sequences," *Data & Knowledge Engineering*, vol. 126, 2020.

[29] S. Biswas and K. C. Mondal, "Dynamic fp tree based rare pattern mining using multiple item supports constraints," in *Computational Intelligence, Communications, and Business Analytics (CICBA)*. Springer, 2019.

[30] S. Piri, D. Delen, T. Liu, and W. Paiva, "Development of a new metric to identify rare patterns in association analysis: The case of analyzing diabetes complications," *Expert Systems with Applications*, vol. 94, 2018.

[31] A. Rahman, Y. Xu, K. Radke, and E. Foo, "Finding anomalies in scada logs using rare sequential pattern mining," in *Network and System Security*. Springer, 2016.

[32] J. Zhu, K. Wang, Y. Wu, Z. Hu, and H. Wang, "Mining user-aware rare sequential topic patterns in document streams," *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 28, no. 7, 2016.

[33] W. Ouyang, "Mining rare sequential patterns in large transaction databases," in *International Conference on Computer Science and Electronic Technology*. Atlantis Press, 2016.

[34] A. U. Ahmed, C. F. Ahmed, M. Samiullah, N. Adnan, and C. K.-S. Leung, "Mining interesting patterns from uncertain databases," *Information Sciences*, vol. 354, 2016.

[35] Y.-K. Lee, W.-Y. Kim, Y. D. Cai, and J. Han, “Comine: Efficient mining of correlated patterns,” in *IEEE International Conference on Data Mining (ICDM)*, 2003.

[36] Y. Ke, J. Cheng, and W. Ng, “Correlated pattern mining in quantitative databases,” *ACM Transactions on Database Systems (TODS)*, vol. 33, 2008.

[37] J. Blanchard, F. Guillet, R. Gras, and H. Briand, “Using information-theoretic measures to assess association rule interestingness,” in *IEEE International Conference on Data Mining (ICDM)*, 2005.

[38] X. Cunjin, S. Wanjiao, Q. Lijuan, D. Qing, and W. Xiaoyang, “A mutual-information-based mining method for marine abnormal association rules,” *Computers & Geosciences*, vol. 76, 2015.

[39] Y. Yao, “Information-theoretic measures for knowledge discovery and data mining,” in *Entropy measures, maximum entropy principle and emerging applications*, 2003.

[40] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, “A symbolic representation of time series, with implications for streaming algorithms,” in *Proceedings of the ACM SIGMOD workshop on Research issues in data mining and knowledge discovery*, 2003.

[41] J. F. Allen, “Maintaining knowledge about temporal intervals,” *Communications of the ACM*, vol. 26, 1983.

[42] V. L. Ho, N. Ho, and T. B. Pedersen, “Appendix-efficient generalized temporal pattern mining in big time series using mutual information,” 2023. [Online]. Available: https://arxiv.org/abs/2010.03653

[43] E. R. Omiecinski, “Alternative interest measures for mining associations in databases,” *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, vol. 15, no. 1, 2003.

[44] V. L. Ho, N. Ho, and T. B. Pedersen, “Mining seasonal temporal patterns in time series,” in *IEEE International Conference on Data Engineering (ICDE)*. IEEE, 2023.

[45] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[46] R. M. Corless, G. H. Gonnet, D. E. Hare, D. J. Jeffrey, and D. E. Knuth, “On the lambert w function,” *Advances in Computational mathematics*, vol. 5, 1996.

[47] E. F. Beckenbach, R. Bellman, and R. E. Bellman, “An introduction to inequalities,” Mathematical Association of America Washington, DC, Tech. Rep., 1961.

[48] W. Healy, F. Omar, L. Ng, T. Ullah, W. Payne, B. Dougherty, and A. H. Fanney. (2018) Net zero energy residential test facility instrumented data. [Online]. Available: https://pages.nist.gov/netzero/index.html/

[49] J. Kelly and W. Knottenbelt, “The UK-DALE dataset, domestic appliance-level electricity demand and whole-house demand from five UK homes,” *Scientific Data*, 2015.

[50] P. S. Data. (2016) Pecan street dataport. [Online]. Available: https://www.pecanstreet.org/dataport/

[51] N. Y. City. (2019) Nyc opendata. [Online]. Available: https://opendata.cityofnewyork.us/

[52] C. Neidle, A. Opoku, G. Dimitriadis, and D. Metaxas, “New shared & interconnected asl resources: Signstream® 3 software; dai 2 for web access to linguistically annotated video corpora; and a sign bank,” in *Workshop on the Representation and Processing of Sign Languages: Involving the Language Community, Miyazaki, Language Resources and Evaluation Conference*, 2018.

[53] K. city infectious disease surveillance system. (2021) Kidss. [Online]. Available: https://kidss.city.kawasaki.jp/

[54] O. Weather. (2021) Open weather. [Online]. Available: https://openweathermap.org/

Paper C

Mining Seasonal Temporal Patterns in Time Series

Van Long Ho, Nguyen Ho, and Torben Bach Pedersen

The paper has been published in the
Proceedings of the IEEE 39th International Conference on Data Engineering (ICDE),
Pages 2240–2252, 2023, ISBN 979-8-3503-2227-9,
DOI:10.1109/ICDE55515.2023.00174.

The layout has been revised.

Abstract

As IoT-enabled sensors become more pervasive, very large time series data are increasingly generated and made available for advanced data analytics. By mining temporal patterns from the available data, valuable insights can be extracted to support decision making. A useful type of patterns found in many real-world applications exhibits periodic occurrences, and is thus called seasonal temporal patterns (STP). Compared to regular patterns, mining seasonal temporal patterns is more challenging since traditional measures such as support and confidence do not capture the seasonality characteristics. Further, the anti-monotonicity property does not hold for STPs, and thus, resulting in an exponential search space. We propose a first solution for seasonal temporal pattern mining (STPM) from time series that can mine STP at different data granularities. We design efficient data structures and use two pruning techniques for the STPM algorithm that downsize the search space and accelerate the mining process. Further, based on the mutual information measure, we propose an approximate version of STPM that only mine seasonal patterns on the promising time series. Finally, extensive experiments with real-world and synthetic datasets show that STPM outperforms the baseline in terms of runtime and memory usage, and can scale to large datasets. The approximate STPM is up to an order of magnitude faster and less memory-consuming than the baseline, while maintaining high accuracy.

C.1 Introduction

The widespread of IoT systems enables the collection of big time series from domains such as energy, transportation, climate, and healthcare. Mining such time series can discover hidden patterns and offer new insights into the application domains to support evidence-based decision making and planning. Often, pattern mining methods such as sequential pattern mining (SPM) [1, 2] and temporal pattern mining (TPM) [3, 4] are used to extract frequent (temporal) relations between events. In SPM, events occur in sequential order, whereas in TPM, events carry additional temporal information such as occurrence time, making relations between temporal events are more expressive and comprehensive. A useful type of temporal patterns found in many real-world applications are those that exhibit periodic occurrences. Such patterns occur concentrated within a particular time period, and then repeat that concentrated occurrence periodically. They are thus called *seasonal temporal patterns*. Here, the term *seasonal* indicates the periodic re-occurrence, while the term *temporal pattern* indicates patterns that are formed by the temporal relations between events, such as follows, contains, overlaps. Seasonal temporal patterns are useful in revealing seasonal information of temporal events and their relations. For example, in healthcare, health experts might be interested in finding seasonal diseases in a geographical location, as exemplified in Fig. C.1 using

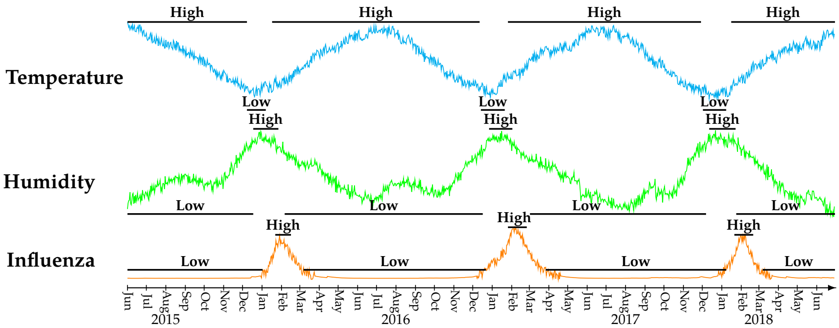


Fig. C.1: Weather and Influenza time series

the real-world data from Kawasaki, Japan between 2015 - 2018 [5], [6]. Here, a seasonal temporal pattern involving weather and epidemic events can be found: {Low Temperature *overlaps* High Humidity *followed by* High Influenza Cases}. This pattern occurs yearly and is concentrated in January, February. Detecting such seasonal diseases will support health experts in prevention and planning. In market analysis, knowing the periodic rise of certain stocks and their relations to other impact factors can be of interests for traders to plan better trading strategies. In marketing, identifying the order of search keywords that appear seasonally in the search engine can be useful to better understand customer needs and thereby improve the marketing plans.

Challenges. Although seasonal temporal patterns are useful, mining them is a challenging task for several reasons. First, the *support* measure used by TPM is not sufficient to mine seasonal patterns, since the traditional *support* represents the frequency of a pattern across the entire dataset, and thus, cannot capture the seasonality characteristic of seasonal patterns. Second, the many possible relations between temporal events create an exponential search space of size $O(n^h 3^{h^2})$ (n is the number of events and h is the length of temporal patterns). Finally, since seasonal temporal patterns do not uphold the anti-monotonicity property, i.e., the non-empty subsets of a seasonal temporal pattern may not be seasonal, mining seasonal temporal patterns is more computationally expensive as the typical pruning technique based on anti-monotonicity property cannot be applied. This raises the need for an efficient seasonal temporal pattern mining approach with effective prunings to tackle the exponential search space. Existing work such as [7, 8] proposes solutions to mine seasonal itemsets. However, they do not consider the temporal aspect of items/ events, thus, addressing the exponential search space of seasonal temporal patterns is still an open problem.

Contributions. In the present paper, we present our Frequent Seasonal Temporal Pattern Mining from Time Series (FreqSTPfTS) solution that addresses all the above challenges. Specifically, our key contributions are as follows. (1) We propose the first solution to mine *seasonal temporal patterns* from time series. Within the process, we introduce several measures to assess the seasonality characteristics, and use these to formally define the concept of *seasonal temporal patterns* in time series. The formulation allows to flexibly mine seasonal temporal patterns at different granularities. (2) Our Seasonal Temporal Pattern Mining (STPM) algorithm is efficient and has several important novelties. First, STPM employs the *hierarchical hash tables* to enable fast retrieval of candidate events and patterns during the mining process. Second, we define a new measure *maxSeason* that upholds the anti-monotonicity property, and design two efficient pruning techniques: Apriori-like pruning and transitivity pruning. (3) Based on mutual information, we approximate STPM to prune redundant time series and significantly reduce the search space, while maintaining highly accurate results. The approximate STPM can scale to many time series and many sequences. (4) We perform extensive experimental evaluation on synthetic and real-world datasets from various domains showing that STPM outperforms the baseline in both runtime and memory usage. The approximate STPM achieves up to an order of magnitude speedup w.r.t. the baseline, while obtaining high accuracy compared to the exact STPM. Artifacts are available at: https://github.com/vanholong/STPM.

C.2 Related work

Finding seasonal patterns that represent temporal periodicity in time series is an important research topic, and has received substantial attention in the last decades. By considering seasonality as periodic occurrences, different techniques have been proposed to find periodic sub-sequences in time series data. Such techniques, first introduced by Han et al. in [9, 10], and later extended by [11–15], are called *motif* discovery techniques. However, since motifs are defined as similar time series sub-sequences, motif discovery can only find recurrent sub-sequences rather than periodic temporal patterns.

Another research direction in this area concerns periodic association rules [7, 8, 16–32]. Such techniques can identify seasonal associations between itemsets, for example, market-basket analysis to reveal the seasonal occurrence of the association {Glove ⇒ Winter Hat} during the winter season. To mine such seasonal itemset patterns in transactional databases, Tanbeer et al. in [16] proposed the PFP-growth algorithm using *minSup* and *maxPer* as seasonality measures. In their method, a tree structure called PF-tree is used as a compact representation of periodic frequent itemsets, with *maxPer* imposing the periodic constraint, and *minSup* imposing the frequency constraint on the pattern occurrences. Although PFP-growth can capture seasonality characteristic through the *maxPer* measure, the use of *minSup* means that it cannot identify rare seasonal patterns. Follow-up work such as [17, 18] improves different aspects of PFP-growth, for example, Amphawan et al. [18] propose *period sum-*

mary to approximate the pattern periodicity to reduce the memory cost, Uday et al. [17] use the concept of *item-specific support* to address the rare pattern problem. Recently, Javed et al. [32] propose hashed occurrence vectors and Apriori-based approach to speed up periodic itemsets mining.

In a more recent work [7], Uday et al. propose the RP-growth algorithm to discover recurring itemset patterns in transactional databases. RP-growth uses an RP-tree to maintain frequent itemsets, and recursively mines the RP-tree to discover recurring ones. In their follow-up work, the same authors introduce several improvements of [7]. In [33], they propose the Periodic-Frequent Pattern-growth++ (PFP-growth++) algorithm that employs two new concepts, *local-periodicity* and *periodicity*, to capture locally optimal and globally optimal solutions of recurring patterns. This enables 2-phase pruning to improve the runtime efficiency. In [8], the authors extend PFP-growth++ to find periodic spatial patterns in spatio-temporal databases. In [31], PFP-growth++ is extended to find maximal periodic frequent patterns. In [34], they further improve PFP-growth++ to be memory efficient by proposing a concept called *period summary* to effectively summarize the temporal occurrence information of an itemset in a Periodic Summary-tree (PS-tree), and designing Periodic Summary Pattern Growth algorithm (PS-growth) to find all periodic-frequent itemset patterns from PS-tree. Nevertheless, all the mentioned work can only discover seasonal patterns between itemsets. To the best of our knowledge, no existing work addresses the seasonal temporal pattern mining that finds seasonal occurrences of temporal patterns. In Section C.6, we adapt the state-of-the-art method for periodic itemset mining *PS-growth* to mine seasonal temporal patterns, and use it as an experimental baseline.

C.3 Preliminaries

C.3.1 Time Granularity

Definition 3.1 (Time domain) A time domain $\mathcal{T}$ consists of an ordered set of time instants that are isomorphic to the natural numbers. The time instants in $\mathcal{T}$ have a time unit, presenting how they are measured.

Definition 3.2 (Time granularity) Given a time domain $\mathcal{T}$, a *time granularity* G is a *complete and non-overlapping equal partitioning* of $\mathcal{T}$, i.e., $\mathcal{T}$ is divided into non-overlapping equal partitions. Each non-empty partition $G_i \in G$ is called a (time) *granule*. The position of a granule G_i in G, denoted as $p(G_i)$, is identified by counting the number of granules which appear before and up to (including) G_i. The *period* between two granules G_i and G_j in granularity G measures the time duration between G_i and G_j, and is computed as: $pr_{ij} = |p(G_i) - p(G_j)|$, where $p(G_i)$ and $p(G_j)$ are the positions of G_i and G_j, respectively.

As an example, consider a time domain $\mathcal{T}$ consisting of an ordered set of

Table C.1: Frequently Used Notations

Notation	Description
$\mathcal{T}, \mathcal{H}$	time domain $\mathcal{T}$ and time granularity hierarchy $\mathcal{H}$
$p(G_i)$	the position of the granule G_i
$G \trianglelefteq_m H$	granularity G is *m-Finer* than granularity H
X, X_S	time series X and symbolic time series X_S
$E_{\triangleright e}$	temporal event E has an event instance e
g: $X_S \rightarrow_m H$	sequence mapping from X_S to granularity H
$Seq_i = <e_1,...,e_n>$	a temporal sequence of n event instances
$\mathcal{D}_{\text{SYB}}, \mathcal{D}_{\text{SEQ}}$	symbolic database and temporal sequence database
H_i^E, H_i^P	event E (pattern P) occurs at granularity H_i
$\text{SUP}^E, \text{SUP}^P$	support set of event E (pattern P)
NearSUP_i^P	near support set i of pattern P
$den(\text{NearSUP}_i^P)$	density of the near support set
$dist(\text{NearSUP}_i^P, \text{NearSUP}_j^P)$	distance between two near support sets
$seasons(P)$	number of seasons of pattern P

minutes. The time instants minute$_1$, minute$_2$, etc. are isomorphically mapped to the natural numbers, and are measured in the *Minute* time unit. Here, $\mathcal{T}$ can have different time granularities such as Minute, 5-Minutes, or even Hour, Day, Year. The position of granule Minute$_2$ in the Minute granularity is $p(\text{Minute}_2) = 2$. The period between the Minute$_1$ and Minute$_6$ granules is: $|p(\text{Minute}_6) - p(\text{Minute}_1)| = 5$, indicating that the time duration between them is 5 minutes. We note that the period is only defined between granules of the same granularity.

Definition 3.3 (Finer time granularity) A time granularity G is *finer* than a time granularity H if and only if for every granule $H_j \in H$, there exists m adjacent granules $G_{i+1}, ..., G_{i+m} \in G$ such that $H_j = G_{i+1} \cup ... \cup G_{i+m}$ where $m \geq 1$. We call G is *m-Finer* than H, denoted as $G \trianglelefteq_m H$.

In the previous example, we have the Minute granularity is 60-Finer than the Hour granularity.

Definition 3.4 (Time granularity hierarchy) Given a time domain $\mathcal{T}$, the different time granularities of $\mathcal{T}$ form a *time granularity hierarchy* $\mathcal{H}$ where each level in $\mathcal{H}$ represents one specific granularity, with the lower levels in the hierarchy having finer granularity than the higher levels.

Fig. C.2 shows an example of the time granularity hierarchy. Here, to be consistent with examples in the following sections, we assume granularity G is 5-Minutes and is the finest, whereas granularity H is 15-Minutes and $G \trianglelefteq_3 H$.

C.3.2 Symbolic Representation of Time Series

Consider the time domain $\mathcal{T}$. Let $\mathcal{H}$ be the time granularity hierarchy of $\mathcal{T}$, and G be the finest granularity in $\mathcal{H}$.

Definition 3.5 (Time series [3]) A *time series* $X = x_1, x_2, ..., x_n$ in the time

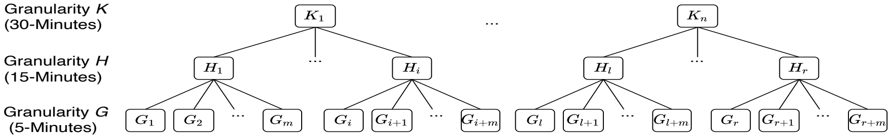


Fig. C.2: Time granularity hierarchy $\mathcal{H}$

domain $\mathcal{T}$ is a chronologically ordered sequence of data values measuring the same phenomenon during an observation time period in $\mathcal{T}$. We say that X has granularity G if X is sampled at every time instant t_i in $\mathcal{T}$.

A *symbolic time series* X_S of X uses a mapping function f: $X \rightarrow \Sigma_X$ that maps each value $x_i \in X$ into a symbol $\omega \in \Sigma_X$, results in a sequence of symbols [3]. The *symbol alphabet* Σ_X is the finite set of symbols used for encoding X. Since the mapping function f performs the 1-to-1 mapping from X to X_S, X_S has the same granularity G as X.

As an example, a time series of the energy usage (recorded every 5 minutes) of an electrical device X = 1.82, 1.25, 0.46, 0.0 can be encoded as X_S = 1, 1, 1, 0 by using Σ_X = {1, 0} (1: ON, 0: OFF).

Definition 3.6 (Symbolic database [3]) The set of symbolic representations of a given set of time series $\mathcal{X} = \{X_1, ..., X_n\}$ forms a *symbolic database* $\mathcal{D}_{\text{SYB}}$.

Table C.2 shows an example symbolic database, $\mathcal{D}_{\text{SYB}}$ using Σ = {0, 1}. There are 5 time series: {C, D, F, M, N} (C: Cooker, D: Dish Washer, F: Food Processor, M: Microwave, N: Nespresso Coffee) representing the energy usage of electrical devices at 5-Minutes granularity.

Table C.2: A Symbolic Database $\mathcal{D}_{\text{SYB}}$ (G: 5-Minutes granularity)

Granules in G		G_1	G_2	G_3	G_4	G_5	G_6	G_7	G_8	G_9	G_{10}	G_{11}	G_{12}	G_{13}	G_{14}	G_{15}	G_{16}	G_{17}	G_{18}	G_{19}	G_{20}	G_{21}
Position		**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**	**18**	**19**	**20**	**21**
Time series	**C**	1	1	0	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1
	D	1	0	0	1	0	0	1	1	0	1	1	0	0	0	0	0	0	0	1	1	1
	F	0	0	1	0	1	1	0	0	1	0	0	1	1	1	1	0	0	0	0	0	0
	M	1	1	1	1	0	0	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0
	N	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	0	0	0

Granules in G		G_{22}	G_{23}	G_{24}	G_{25}	G_{26}	G_{27}	G_{28}	G_{29}	G_{30}	G_{31}	G_{32}	G_{33}	G_{34}	G_{35}	G_{36}	G_{37}	G_{38}	G_{39}	G_{40}	G_{41}	G_{42}
Position		**22**	**23**	**24**	**25**	**26**	**27**	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**	**40**	**41**	**42**
Time series	**C**	1	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	1	0
	D	1	1	1	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	1	1	0
	F	0	0	0	1	1	1	1	1	1	0	0	1	0	0	1	0	0	1	0	0	1
	M	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
	N	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0

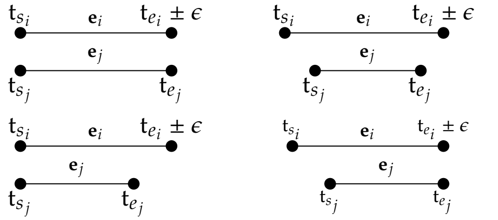
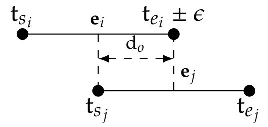
C.3.3 Temporal Event and Temporal Relation

Definition 3.7 (Temporal event [3]) A *temporal event* E in a symbolic time series X_S is a tuple $E = (\omega, T)$. Here, $\omega \in \Sigma_X$ is a symbol, and $T = \{[t_{s_i}, t_{e_i}]\}$ is the set of time intervals during which X_S has the value ω. Each time interval has t_{s_i} as the start time, and t_{e_i} as the end time.

Instance of a temporal event: An *instance* of the temporal event $E = (\omega, T)$ is a tuple $e = (\omega, [t_{s_i}, t_{e_i}])$. e represents a single occurrence of E during $[t_{s_i}, t_{e_i}]$. We use the notation $E_{\triangleright e}$ to denote that the event E has an instance e.

Consider the symbolic time series C in Table C.2. Then E = (C:1, $\{[G_1, G_2]$, $[G_4, G_4]$, $[G_7, G_8]$, $[G_{19}, G_{24}]$, $[G_{31}, G_{31}]$, $[G_{34}, G_{35}]$, $[G_{40}, G_{41}]\}$) is an event of C,

Table C.3: Temporal Relations between Events

Follows: $E_{i \triangleright e_i} \rightarrow E_{j \triangleright e_j}$	 $t_{e_i} \pm \epsilon \leq t_{s_j}$
Contains: $E_{i \triangleright e_i} \succcurlyeq E_{j \triangleright e_j}$	 $(t_{s_i} \leq t_{s_j}) \wedge (t_{e_i} \pm \epsilon \geq t_{e_j})$
Overlaps: $E_{i \triangleright e_i} \between E_{j \triangleright e_j}$	 $(t_{s_i} < t_{s_j}) \wedge (t_{e_i} \pm \epsilon < t_{e_j}) \wedge (t_{e_i} - t_{s_j} \geq d_o \pm \epsilon)$

representing the time intervals during which C is associated with the symbol 1. The tuple (C:1, $[G_1, G_2]$) is an instance of E. Note that for simplicity, we use the granules to represent the start and end times of the time intervals, as we can trace back the timestamp associated to each granule.

Relations between temporal events: Let E_i and E_j be two temporal events, and $e_i = (\omega_i, [t_{s_i}, t_{e_i}])$, $e_j = (\omega_j, [t_{s_j}, t_{e_j}])$ be their corresponding instances. We rely on the popular relation model of Allen [35] to define 3 basic temporal relations between E_i and E_j: *Follows*, *Contains*, and *Overlaps*. Furthermore, we add a tolerance *buffer* ϵ to the relation's endpoints for flexibility, and ensure that the relations are *mutually exclusive* (proof in the technical report [36]). Table C.3 illustrates the three relations and their conditions, with $\epsilon \geq 0$ being the buffer size, and d_o being the minimal overlapping duration between two instances in an Overlaps relation.

Definition 3.8 (Temporal pattern [3]) Assume the set of temporal relations to be $\mathfrak{R}$={Follows, Contains, Overlaps}. A *temporal pattern* P = <(r_{12}, E_1, E_2),...,$(r_{(n-1)(n)}, E_{n-1}, E_n)$> contains triples (r_{ij}, E_i, E_j), each represents a temporal relation $r_{ij} \in \mathfrak{R}$ between E_i and E_j.

An example of temporal pattern is shown in Fig. C.1: P = <(Overlaps, Low Temperature, High Humidity), (Follows, Low Temperature, High Influenza Cases), (Follows, High Humidity, High Influenza Cases)>. Here, P is a 3-event pattern, containing pairwise temporal relations between Low Temperature, High Humidity, and High Influenza Cases.

Table C.4: A Temporal Sequence Database $\mathcal{D}_{\text{SEQ}}$ (H: 15-Minutes granularity)

Granules	Position	Temporal sequences
$\boldsymbol{H_1}$={$\boldsymbol{G_1}$,$\boldsymbol{G_2}$,$\boldsymbol{G_3}$}	1	(C:1,[G_1, G_2]), (C:0,[G_3, G_3]), (D:1,[G_1, G_1]), (D:0,[G_2, G_3]), (F:0,[G_1, G_2]), (F:1,[G_3, G_3]), (M:1,[G_1, G_3]), (N:1,[G_1, G_2]), (N:0,[G_3, G_3])
$\boldsymbol{H_2}$={$\boldsymbol{G_4}$,$\boldsymbol{G_5}$,$\boldsymbol{G_6}$}	2	(C:1,[G_4, G_4]), (C:0,[G_5, G_6]), (D:1,[G_4, G_4]), (D:0,[G_5, G_6]), (F:0,[G_4, G_4]), (F:1,[G_5, G_6]), (M:1,[G_4, G_4]), (M:0,[G_5, G_6]), (N:1,[G_4, G_6])
$\boldsymbol{H_3}$={$\boldsymbol{G_7}$,$\boldsymbol{G_8}$,$\boldsymbol{G_9}$}	3	(C:1,[G_7, G_8]), (C:0,[G_9, G_9]), (D:1,[G_7, G_8]), (D:0,[G_9, G_9]), (F:0,[G_7, G_8]), (F:1,[G_9, G_9]), (M:1,[G_7, G_9]), (N:1,[G_7, G_9])
$\boldsymbol{H_4}$={$\boldsymbol{G_{10}}$,$\boldsymbol{G_{11}}$,$\boldsymbol{G_{12}}$}	4	(C:0,[G_{10}, G_{12}]), (D:1,[G_{10}, G_{11}]), (D:0,[G_{12}, G_{12}]), (F:0,[G_{10}, G_{11}]), (F:1,[G_{12}, G_{12}]), (M:1,[G_{10}, G_{11}]), (M:0,[G_{12}, G_{12}]), (N:1,[G_{10}, G_{11}]), (N:0,[G_{12}, G_{12}])
$\boldsymbol{H_5}$={$\boldsymbol{G_{13}}$,$\boldsymbol{G_{14}}$,$\boldsymbol{G_{15}}$}	5	(C:0,[G_{13}, G_{15}]), (D:0,[G_{13}, G_{15}]), (F:1,[G_{13}, G_{15}]), (M:1,[G_{13}, G_{15}]), (N:1,[G_{13}, G_{15}])
$\boldsymbol{H_6}$={$\boldsymbol{G_{16}}$,$\boldsymbol{G_{17}}$,$\boldsymbol{G_{18}}$}	6	(C:0,[G_{16}, G_{18}]), (D:0,[G_{16}, G_{18}]), (F:0,[G_{16}, G_{18}]), (M:1,[G_{16}, G_{18}]), (N:1,[G_{16}, G_{18}])
$\boldsymbol{H_7}$={$\boldsymbol{G_{19}}$,$\boldsymbol{G_{20}}$,$\boldsymbol{G_{21}}$}	7	(C:1,[G_{19}, G_{21}]), (D:1,[G_{19}, G_{21}]), (F:0,[G_{19}, G_{21}]), (M:0,[G_{19}, G_{21}]), (N:0,[G_{19}, G_{21}])
$\boldsymbol{H_8}$={$\boldsymbol{G_{22}}$,$\boldsymbol{G_{23}}$,$\boldsymbol{G_{24}}$}	8	(C:1,[G_{22}, G_{24}]), (D:1,[G_{22}, G_{24}]), (F:0,[G_{22}, G_{24}]), (M:1,[G_{22}, G_{24}]), (N:0,[G_{22}, G_{24}])
$\boldsymbol{H_9}$={$\boldsymbol{G_{25}}$,$\boldsymbol{G_{26}}$,$\boldsymbol{G_{27}}$}	9	(C:0,[G_{25}, G_{27}]), (D:0,[G_{25}, G_{27}]), (F:1,[G_{25}, G_{27}]), (M:1,[G_{25}, G_{27}]), (N:1,[G_{25}, G_{27}])
$\boldsymbol{H_{10}}$={$\boldsymbol{G_{28}}$,$\boldsymbol{G_{29}}$,$\boldsymbol{G_{30}}$}	10	(C:0,[G_{28}, G_{30}]), (D:0,[G_{28}, G_{30}]), (F:1,[G_{28}, G_{30}]), (M:1,[G_{28}, G_{30}]), (N:1,[G_{28}, G_{30}])
$\boldsymbol{H_{11}}$={$\boldsymbol{G_{31}}$,$\boldsymbol{G_{32}}$,$\boldsymbol{G_{33}}$}	11	(C:1,[G_{31}, G_{31}]), (C:0,[G_{32}, G_{33}]), (D:1,[G_{31}, G_{31}]), (D:0,[G_{32}, G_{33}]), (F:0,[G_{31}, G_{32}]), (F:1,[G_{33}, G_{33}]), (M:1,[G_{31}, G_{33}]), (N:1,[G_{31}, G_{33}])
$\boldsymbol{H_{12}}$={$\boldsymbol{G_{34}}$,$\boldsymbol{G_{35}}$,$\boldsymbol{G_{36}}$}	12	(C:1,[G_{34}, G_{35}]), (C:0,[G_{36}, G_{36}]), (D:1,[G_{34}, G_{34}]), (D:0,[G_{35}, G_{36}]), (F:0,[G_{34}, G_{35}]), (F:1,[G_{36}, G_{36}]), (M:0,[G_{34}, G_{36}]), (N:1,[G_{34}, G_{36}])
$\boldsymbol{H_{13}}$={$\boldsymbol{G_{37}}$,$\boldsymbol{G_{38}}$,$\boldsymbol{G_{39}}$}	13	(C:0,[G_{37}, G_{39}]), (D:1,[G_{37}, G_{38}]), (D:0,[G_{39}, G_{39}]), (F:0,[G_{37}, G_{38}]), (F:1,[G_{39}, G_{39}]), (M:1,[G_{37}, G_{39}]), (N:1,[G_{37}, G_{39}])
$\boldsymbol{H_{14}}$={$\boldsymbol{G_{40}}$,$\boldsymbol{G_{41}}$,$\boldsymbol{G_{42}}$}	14	(C:1,[G_{40}, G_{41}]), (C:0,[G_{42}, G_{42}]), (D:1,[G_{40}, G_{41}]), (D:0,[G_{42}, G_{42}]), (F:0,[G_{40}, G_{41}]), (F:1,[G_{42}, G_{42}]), (M:0,[G_{40}, G_{42}]), (N:0,[G_{40}, G_{42}])

C.3.4 Temporal Sequence Database

Definition 3.9 (Sequence mapping) Consider a symbolic time series X_S of granularity G. Let H be a granularity in $\mathcal{H}$ such that $G \trianglelefteq_m H$. A sequence mapping g: $X_S \rightarrow_m H$ maps m adjacent symbols in X_S into a single granule $H_i \in H$.

For example, consider the symbolic time series C in Table C.2. Using $G \trianglelefteq_3 H$, a sequence mapping g: $C \rightarrow_3 H$ creates granularity H where the granules are: H_1: <C:1, C:1, C:0>, H_2: <C:1, C:0, C:0>, H_3: <C:1, C:1, C:0>, and so on.

Definition 3.10 (Temporal sequence of a symbolic time series) Consider a symbolic time series X_S of granularity G. Let $<\omega_1, ..., \omega_m>$ be a symbolic sequence at granule H_i in H, obtained by performing a sequence mapping g: $X_S \rightarrow_m H$. A *temporal sequence* $Seq_i =< e_1, ..., e_n >$ is a list of n event instances, each is obtained by grouping consecutive and identical symbols ω in H_i into an event instance $e = (\omega, [t_s, t_e])$.

In the previous example, the temporal sequences of the granules in H are: Seq_1 = <(C:1, [G_1, G_2]), (C:0, [G_3, G_3])> at H_1, Seq_2 = <(C:1, [G_4, G_4]), (C:0, [G_5, G_6])> at H_2, Seq_3 = <(C:1, [G_7, G_8]), (C:0, [G_9, G_9])> at H_3, and so on.

Definition 3.11 (Temporal sequence database) Consider a symbolic database $\mathcal{D}_{\text{SYB}}$ of granularity G (defined in Def 3.6) which contains a collection of symbolic time series $\{X_S\}$, and a granularity $H \in \mathcal{H}$. Let $g : X_S \rightarrow_m H$ be a sequence mapping applied to each symbolic time series X_S in $\mathcal{D}_{\text{SYB}}$. The temporal sequences obtained from the mapping g form a *temporal sequence database* $\mathcal{D}_{\text{SEQ}}$. Each row i in $\mathcal{D}_{\text{SEQ}}$ is a set of sequences $\{Seq_i\}$ of the same granule $H_i \in H$. Furthermore, the temporal sequence database $\mathcal{D}_{\text{SEQ}}$ has granularity H.

Table C.4 shows an example of $\mathcal{D}_{\text{SEQ}}$, obtained from $\mathcal{D}_{\text{SYB}}$ in Table C.2

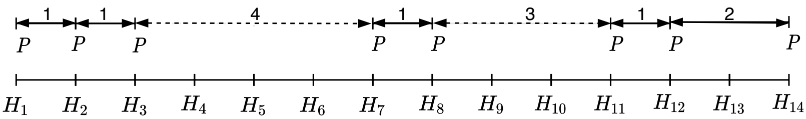


Fig. C.3: Near support sets of pattern P = (C:1 $\succcurlyeq$ D:1)

using the mapping $g : X_S \rightarrow_3 H$ on the five symbolic time series {C, D, F, M, N}.

Given a symbolic database $\mathcal{D}_{\text{SYB}}$ of granularity G and a granularity hierarchy $\mathcal{H}$, we can construct different temporal sequence databases $\mathcal{D}_{\text{SEQ}}$ of different granularities $H \in \mathcal{H}$ by using different sequence mappings $g : X_S \rightarrow_m H$. For instance, in the previous example, using $g : X_S \rightarrow_3 H$, we obtain $\mathcal{D}_{\text{SEQ}}$ at 15-Minutes granularity. Using $g : X_S \rightarrow_{12} H$, we obtain $\mathcal{D}_{\text{SEQ}}$ at 1-Hour granularity.

C.3.5 Frequent Seasonal Temporal Pattern

Definition 3.12 (Support set of a temporal event) Consider a temporal sequence database $\mathcal{D}_{\text{SEQ}}$ of granularity H, and a temporal event E. The set of granules H_i in $\mathcal{D}_{\text{SEQ}}$ where E occurs, arranged in an increasing order, is called the *support set* of event E and is denoted as $\text{SUP}^E = \{H_l^E, ..., H_r^E\}$, where $1 \leq l \leq r \leq |\mathcal{D}_{\text{SEQ}}|$. The granule H_i at which event E occurs is denoted as H_i^E. The support set of a group of events, denoted as $\text{SUP}^{(E_i,...,E_k)}$, and the support set of a temporal pattern, denoted as $\text{SUP}^P = \{H_l^P, ..., H_r^P\}$, are defined similarly to that of a temporal event.

Definition 3.13 (Near support set of a temporal pattern) Consider a pattern P with the support set $\text{SUP}^P = \{H_l^P, ..., H_r^P\}$. Let *maxPeriod* be the *maximum period threshold*, representing the predefined maximal period between any two consecutive granules in SUP^P. The set SUP^P is called a *near support set* of P if $\forall (H_o^P, H_p^P) \in \text{SUP}^P$: ($H_o^P$ and H_p^P are consecutive) $\wedge$ $|p(H_o^P) - p(H_p^P)| \leq$ *maxPeriod*, where $p(H_o^P)$ and $p(H_p^P)$ are the positions of H_o^P and H_p^P in granularity H. We denote the near support set of pattern P as NearSUP^P.

Intuitively, the near support set of P is a support set where P's occurrences are close in time. Moreover, NearSUP^P is called a *maximal near support set* if NearSUP^P has no other superset beside itself which is also a near support set. The near support set of an event is defined similarly to that of a pattern.

As an example, consider the pattern P = (Contains, C:1, D:1) (or C:1 $\succcurlyeq$ D:1) in Table C.4, and let *maxPeriod* = 2. Here, the support set of P is $\text{SUP}^P = \{H_1, H_2, H_3, H_7, H_8, H_{11}, H_{12}, H_{14}\}$. Hence, P has three maximal near support sets: $\text{NearSUP}_1^P = \{H_1, H_2, H_3\}$, $\text{NearSUP}_2^P = \{H_7, H_8\}$, and $\text{NearSUP}_3^P = \{H_{11}, H_{12}, H_{14}\}$. Fig. C.3 illustrates the three near support sets of P.

Definition 3.14 (Season of a temporal pattern) Let NearSUP^P be a near support set of a pattern P. Then NearSUP^P is called a *season* of P if $den(\text{NearSUP}^P) = |\text{NearSUP}^P| \geq minDensity$, where $den(\text{NearSUP}^P)$ counts the number of granules in NearSUP^P called the *density of* NearSUP^P, and *minDensity* is a predefined minimum density threshold.

For instance, in the previous example, we have $den(\text{NearSUP}_1^P) = |\text{NearSUP}_1^P| = 3$. Similarly, $den(\text{NearSUP}_2^P) = 2$, $den(\text{NearSUP}_3^P) = 3$. If the occurrences of a pattern P are dense enough, the near support set becomes a season of P. Intuitively, a *season* of a temporal pattern is a *concentrated occurrence period*, separated by a long *gap period* of no/few occurrences, before the next season starts. The season of an event is defined similarly as for a pattern.

The *distance* between two seasons $\text{NearSUP}_i^P = \{H_k^P, ..., H_n^P\}$ and $\text{NearSUP}_j^P = \{H_r^P, ..., H_u^P\}$ is computed as: $dist(\text{NearSUP}_i^P, \text{NearSUP}_j^P) = |p(H_n^P) - p(H_r^P)|$.

Based on the season concept and the distance measure, we define frequent seasonal temporal patterns as follows.

Definition 3.15 (Frequent seasonal temporal pattern) Let $\mathcal{PS} = \{\text{NearSUP}^P\}$ be the set of seasons of a temporal pattern P, and *minSeason* be the *minimum seasonal occurrence* threshold, *distInterval* $= [\text{dist}_{\text{min}}, \text{dist}_{\text{max}}]$ be the *distance interval* where dist_{min} is the minimum distance and dist_{max} is the maximum distance. A temporal pattern P is called a *frequent seasonal temporal pattern* iff $seasons(P) = |\mathcal{PS}| \geq minSeason \wedge \forall(\text{NearSUP}_i^P, \text{NearSUP}_j^P) \in \mathcal{PS}$: they are consecutive and $\text{dist}_{\text{min}} \leq dist(\text{NearSUP}_i^P, \text{NearSUP}_j^P) \leq \text{dist}_{\text{max}}$.

Intuitively, a pattern P is *seasonal* if the distance between two consecutive seasons is within the predefined distance interval. Moreover, a seasonal temporal pattern is *frequent* if it occurs more often than a predefined *minimum seasonal occurrence* threshold. The number of seasons of a pattern P is the size of $\mathcal{PS}$, and is computed as $seasons(P) = |\mathcal{PS}|$.

Mining Frequent Seasonal Temporal Patterns from Time Series (FreqSTPfTS). Let $\mathcal{D}_{\text{SEQ}}$ be the temporal sequence database of granularity $H \in \mathcal{H}$ obtained from a given set of n time series $\mathcal{X} = \{X_1, ..., X_n\}$ of granularity $G_{\mathcal{X}}$. Let *maxPeriod*, *minDensity*, *distInterval*, and *minSeason* be the maximum period, minimum density, distance interval, and minimum seasonal occurrence thresholds, respectively. The FreqSTPfTS problem aims to find all frequent seasonal temporal patterns P in $\mathcal{D}_{\text{SEQ}}$ that satisfy the *maxPeriod*, *minDensity*, *distInterval*, and *minSeason* constraints.

In Section C.6.1, we provide the guidelines on how to set the values of the four constraints in real-life settings.

C.4 Frequent Seasonal Temporal Pattern Mining

C.4.1 Overview of FreqSTPfTS Mining Process

The FreqSTPfTS mining process consists of two phases. **Phase 1**, *Data Conversion*, transforms a set of time series $\mathcal{X}$ into a symbolic database $\mathcal{D}_{\text{SYB}}$ by using the mapping function defined in Def. 3.5, and then transforms $\mathcal{D}_{\text{SYB}}$ into a temporal sequence database $\mathcal{D}_{\text{SEQ}}$ by applying the sequence mapping defined in Def. 3.9. **Phase 2**, *Seasonal Temporal Pattern Mining (STPM)*, consists of two steps to mine frequent seasonal temporal patterns: *Seasonal Single Event Mining* and *Seasonal k-Event Pattern Mining* (k ≥ 2).

Before introducing the STPM algorithm in detail, we first present *candidate seasonal pattern*, a concept designed to support Apriori-like pruning in STPM.

C.4.2 Candidate Seasonal Pattern

Pattern mining methods often use the *anti-monotonicity* property of the *support* measure to reduce the search space [37]. This property ensures that an infrequent event E_i cannot form a frequent 2-event pattern P, since support(E_i) ≥ support(P). Hence, if E_i is infrequent, we can safely remove E_i and any of its combinations from the search space, and still guarantee the algorithm completeness. However, seasonal temporal patterns constrained by the *maxPeriod*, *minDensity*, *distInterval* and *minSeason* thresholds do *not* uphold this property, as illustrated below.

Consider an event E = M:1 and a 2-event pattern P = M:1 ≽ N:1 in Table C.4. Let *maxPeriod* = 2, *minDensity* = 3, *distInterval* = [4, 10], and *minSeason* = 2. From the constraints, we can identify the seasons of E and P as: $\mathcal{PS}^E = \{\text{NearSUP}_1^E\}$ $= \{H_1, H_2, H_3, H_4, H_5, H_6, H_8, H_9, H_{10}, H_{11}, H_{13}\}$, and $\mathcal{PS}^P = \{\{\text{NearSUP}_1^P\} = \{H_1, H_3, H_4, H_5, H_6\}, \{\text{NearSUP}_2^P\} = \{H_{10}, H_{11}, H_{13}\}\}$. Here, for the pattern P, H_2 is not present in $\{\text{NearSUP}_1^P\}$ since P does not occur in H_2, and H_9 is not present in $\{\text{NearSUP}_2^P\}$ because of the constraint $\text{dist}_{\text{min}} = 4$. Hence, we have: $|\mathcal{PS}^E|$=1 and $|\mathcal{PS}^P|$=2. Due to the *minSeason* constraint, E is not a frequent seasonal event, whereas P is. This shows that seasonal temporal patterns do not adhere to the anti-monotonic property.

To improve STPM performance, we propose the novel *maximum seasonal occurrence* measure, called *maxSeason*, that upholds the anti-monotonicity property to prune infrequent patterns and reduce STPM search space. Indeed, *maxSeason* is an upper bound on the number of seasons of a pattern.

Maximum seasonal occurrence of a temporal pattern P: is the ratio between the number of granules in the support set SUP^P of P, and the *minDensity* threshold:

$$maxSeason(P) = \frac{|SUP^P|}{minDensity} \tag{C.1}$$

Eq. (C.1) divides the number of granules containing P by the minimum density of a season. Thus, it computes the maximum seasons a pattern P can have. The maximum seasonal occurrence of a single event E, and of a group of events $(E_i, ..., E_k)$, are defined in a similar way. Below, we show how *maxSeason* upholds the anti-monotonicity property.

Lemma 1 *Consider two patterns P and P′ such that* $P' \subseteq P$. *Then* $maxSeason(P') \geq maxSeason(P)$.

Proof *We have:*

$maxSeason(P') = \frac{|SUP^{P'}|}{minDensity}$, $maxSeason(P) = \frac{|SUP^{P}|}{minDensity}$

Since: $|SUP^{P'}| \geq |SUP^{P}|$ *(Derived from Def. 3.12)*

Hence: $maxSeason(P') \geq maxSeason(P)$

Lemma 2 *Consider a k-event pattern P created by a k-event group* $(E_1, ..., E_k)$. *Then,* $maxSeason(P) \leq maxSeason(E_1, ..., E_k)$.

Proof *Derived directly from Def. 3.12, and Eq.* (C.1).

From Lemmas 1 and 2, the *maxSeason* of a pattern P is always at most the *maxSeason* of its sub-pattern P', and of its events $(E_1, ..., E_k)$. Thus, *maxSeason* upholds the anti-monotonicity property, and can be used to reduce the STPM search space. Below, we define the *candidate pattern* concept that uses *maxSeason* as a gatekeeper to identify frequent/ infrequent seasonal patterns.

Candidate seasonal pattern: A temporal pattern P is a *candidate seasonal pattern* if $maxSeason(P) \geq minSeason$.

Similarly, a group of k events $G_E = (E_1, ..., E_k)$ $(k \geq 1)$ is a *candidate seasonal k-event group* if $maxSeason(G_E) \geq minSeason$. Intuitively, a pattern P (or k-event group G_E) is infrequent if its *maxSeason* is less than *minSeason*. Hence, P (or G_E) can be safely removed from the search space.

Next, we present our STPM algorithm and detail the two mining steps, shown in Algorithm 11.

C.4.3 Mining Seasonal Single Events

The first step in STPM is to mine frequent seasonal single events (Alg. 11, lines 1-9) that satisfy the constraints of *maxPeriod*, *minDensity*, *distInterval* and *minSeason*. To do that, we first look for the candidate single events defined in Section C.4.2, and then use only the found candidates to mine frequent seasonal events.

The candidate single events are found by first scanning $\mathcal{D}_{\text{SEQ}}$ to identify the support set SUP^{E_i} for each event E_i, from which we compute the maximum seasonal occurrence $maxSeason(E_i)$. If $maxSeason(E_i) \geq minSeason$, then E_i is a candidate seasonal single event. Otherwise, E_i is not a candidate and is

Algorithm 11: Frequent Seasonal Temporal Pattern Mining

Input: Temporal sequence database $\mathcal{D}_{\text{SEQ}}$, the thresholds: *maxPeriod*, *minDensity*, *distInterval*, *minSeason*

Output: All frequent seasonal temporal patterns $\mathcal{P}$

// Step 2.1: Mine frequent seasonal single events

foreach *event* $E_i \in \mathcal{D}_{SEQ}$ **do**
    Find SUP^{E_i} and compute *maxSeason*(E_i) ;
    if *maxSeason*$(E_i) \geq$ *minSeason* **then**
        Insert E_i into *Candidate1Event* ;
foreach *candidate* $E_i \in$ *Candidate1Event* **do**
    Find NearSUP^{E_i} that satisfies *maxPeriod* and *minDensity* ;
    Find PS^{E_i} that adheres *distInterval* ;
    if $|PS^{E_i}| \geq$ *minSeason* **then**
        Insert E_i into $\mathcal{P}$; //E_i is a frequent seasonal event

// Step 2.2: Mine frequent seasonal k-event patterns, $k \geq 2$

FilteredF1 ⟵ Transitivity_Filtering(F_1);
kEventGroups ⟵ Cartesian(FilteredF1, F_{k-1});
CandidatekEvent ⟵ maxSeason_Filtering(kEventGroups);
foreach *kEvent in CandidatekEvent* **do**
    (k-1)-event_patterns ⟵ Retrieve_Relations(PH_{k-1});
    k-event_patterns ⟵ Iterative_Check((k-1)-event_patterns, E_k);
    foreach *P in k-event_patterns* **do**
        if *maxSeason*$(P) \geq minSeason$ **then**
            Insert P into *CandidatekPatterns*;
foreach *candidate* $P \in$ *CandidatekPatterns* **do**
    Find NearSUP^P satisfying *maxPeriod* and *minDensity*;
    Identify $\mathcal{PS}^P$ adhering to *distInterval* ;
    if $|\mathcal{PS}^P| \geq$ *minSeason* **then**
        Insert P into $\mathcal{P}$; //P is a frequent seasonal pattern

removed from the search space. Note that we only need to scan $\mathcal{D}_{\text{SEQ}}$ once to find all candidate events.

To mine frequent seasonal events, for each candidate event E_i, we iterate through the support set SUP^{E_i}, and calculate the period pr_{ij} between every two consecutive granules in SUP^{E_i}, and determine the near support sets NearSUP^{E_i} that satisfy *maxPeriod* and *minDensity*. Next, the set of seasons $\mathcal{PS}^{E_i}$ is identified by selecting the near support sets that adhere to the *distInterval* constraint. Finally, the frequent seasonal events are determined by comparing the number of seasons of E_i to *minSeason*, selecting only those that have *seasons*$(E_i) =\mid \mathcal{PS}^{E_i} \mid \geq$ *minSeason*.

We use a *hierarchical lookup hash structure* HLH_1 to store the candidate seasonal single events. This data structure enables fast search when mining seasonal k-events patterns ($k \geq 2$). Note that we maintain the candidate events in HLH_1 instead of the frequent seasonal events, as the *maxSeason* of candi-

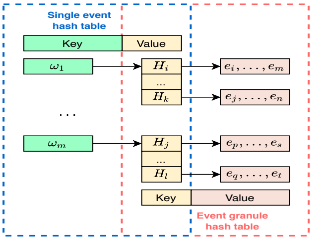


Fig. C.4: The HLH_1 structure

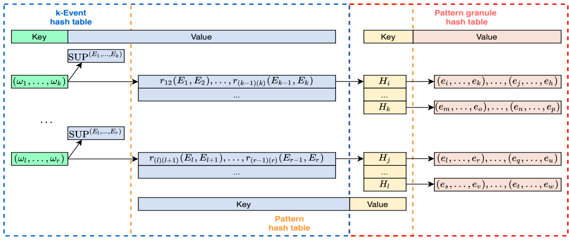


Fig. C.5: The $HLH_k (k \geq 2)$ structure

date events upholds the anti-monotonicity property, and can thus be used for pruning. We illustrate HLH_1 in Fig. C.4, and describe the data structure below.

Hierarchical lookup hash structure HLH_1**:** The HLH_1 is a hierarchical data structure that consists of two hash tables: the *single event hash table* EH, and the *event granule hash table* GH. Each hash table has a list of <key, value> pairs. In EH, the key is the event symbol $\omega \in \Sigma_X$ representing the candidate E_i, and the value is the list of granules $< H_i, ..., H_k >$ in SUP^{E_i}. In GH, the key is the list of granules shared in the value field of EH, while the value stores event instances of E_i that appear at the corresponding granule in $\mathcal{D}_{\text{SEQ}}$. The HLH_1 structure enables fast retrieval of event granules and instances when mining candidate seasonal k-event patterns in the next step of STPM.

We provide an example of HLH_1 in Fig. C.6 using data in Table C.4 with $maxPeriod = 2$, $minDensity = 3$, $distInterval = [4, 10]$, and $minSeason = 2$. Here, out of 10 events in $\mathcal{D}_{\text{SEQ}}$, we have eight candidate seasonal single events stored in HLH_1: C:1, C:0, D:1, D:0, F:1, F:0, M:1, and N:1. Due to space limitations, we only provide the detailed internal structure of four candidate events. Among the eight candidates, the event M:1 does not satisfy the *minSeason* threshold since *season*(M:1) = 1, and thus, is not a frequent seasonal event. However, M:1 is still present in HLH_1 as M:1 might create frequent seasonal k-event patterns. In contrast, N:0 and M:0 are not the candidate seasonal events because they do not satisfy the *maxSeason* constraint, and are omitted from HLH_1.

Complexity: The complexity of finding frequent seasonal events is $O(n \cdot |\mathcal{D}_{\text{SEQ}}|)$, where n is the number of events.

Proof *(Sketch - Full proof in [36]). Computing maxSeason for n events takes* $O(n \cdot |\mathcal{D}_{SEQ}|)$. *Identifying the set of seasons* $\mathcal{PS}$ *of all candidate events* E_i *takes* $O(n \cdot |SUP^{E_i}|)$. *The overall complexity is thus:* $O(n \cdot |\mathcal{D}_{SEQ}| + n \cdot |SUP^{E_i}|) \sim O(n \cdot |\mathcal{D}_{SEQ}|)$.

C.4.4 Mining Seasonal k-event Patterns

STPM's search space. Next, we mine frequent seasonal k-event patterns ($k \geq 2$). A straightforward approach is to enumerate all possible k-event combinations, and check whether each combination can form frequent seasonal patterns. However, this approach is extremely expensive because of the

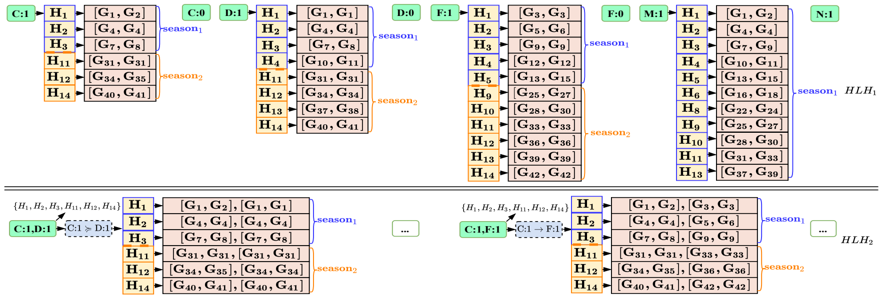


Fig. C.6: A hierarchical lookup hash tables for the running example

very large search space, approximately of size $O(n^h 3^{h^2})$, where n is the number of distinct events in $\mathcal{D}_{\text{SEQ}}$, and h is the maximal length of a pattern.

Proof *(Sketch - Full proof in [36]). The number of seasonal single events is:* $N_1 = n \sim O(n)$. *The number of 2-event groups is:* $N_2 \sim O(n^2)$. *The number of seasonal 2-event patterns is:* $N_2 \times 3^1 \sim O(n^2 3^1)$ *(3 temporal relations for each pair of events). Similarly, the number of seasonal h-event patterns is* $O(n^h 3^{h^2})$. *Hence, the total number of seasonal temporal patterns is* $O(n) + O(n^2 3^1) + ... + O(n^h 3^{h^2}) \sim O(n^h 3^{h^2})$.

To mitigate the large search space, we use an iterative mining approach that first finds candidate seasonal k-event groups, and then mines frequent seasonal k-event patterns only from the candidates.

The hierarchical lookup hash structure HLH_k**:** We use the *hierarchical lookup hash structure* HLH_k ($k \geq 2$) to maintain candidate seasonal k-event groups and patterns, as illustrated in Fig. C.5. The HLH_k contains three hash tables: the *k-event hash table* EH_k, the *pattern hash table* PH_k, and the *pattern granule hash table* GH_k. For each <*key, value*> pair of EH_k, *key* is the list of symbols $(\omega_1 ..., \omega_k)$ representing the candidate k-event group $(E_1, ..., E_k)$, and *value* is an *object* which consists of two components: (1) the support set $SUP^{(E_1,...,E_k)}$, and (2) a list of candidate seasonal k-event temporal patterns. In PH_k, *key* is the candidate pattern P which indeed takes the *value* component of EH_k, while *value* is the list of granules that contain P. In GH_k, *key* is the list of granules containing P which indeed takes the *value* component of PH_k, while *value* is the list of event instances from which the temporal relations in P are formed. The HLH_k hash structure helps speed up the candidate seasonal k-event group mining through the use of the support set in EH_k, and enables fast search for temporal relations between k events using the information in PH_k and GH_k.

4.1 Mining candidate seasonal k-event groups. We first find candidate seasonal k-event groups (Alg. 11, lines 10-12).

Let F_{k-1} and F_1 be the set of candidate seasonal (k-1)-event groups and candidate seasonal single events found in HLH_{k-1} and HLH_1, respectively. We first generate all possible k-event groups by computing the Cartesian product $F_{k-1} \times F_1$. Next, for each k-event group $(E_1, ..., E_k)$, we compute the support set $SUP^{(E_1,...,E_k)}$ by taking the *intersection* between $SUP^{(E_1,...,E_{k-1})}$ in EH_{k-1} and SUP^{E_k} in EH. We then compute $maxSeason(E_1, ..., E_k)$, and evaluate whether $(E_1, ..., E_k)$ is a candidate k-event group, i.e., $maxSeason(E_1, ..., E_k) \geq minSeason$. If $(E_1, ..., E_k)$ is a candidate, it is kept in EH_k of HLH_k.

4.2 Mining frequent seasonal k-event patterns. We use the found candidate k-event groups to mine frequent seasonal k-event patterns (Alg. 11, lines 13-23). We first discuss the case of 2-event patterns, and then generalize to k-event patterns.

4.2.1 Mining frequent seasonal 2-event patterns: For each candidate 2-event group (E_i, E_j), we use the support set $SUP^{(E_i,E_j)}$ to retrieve the temporal sequences $\mathcal{S}$ that contain (E_i, E_j). Next, for each temporal sequence $S \in \mathcal{S}$, we use the instances (e_i, e_j) to verify the temporal relation between them. We then compute the *maxSeason* of the 2-event pattern P and determine if P is a candidate pattern, i.e., $maxSeason(P) \geq minSeason$. Finally, the candidate seasonal 2-event patterns are stored in PH_2, while their event instances are stored in GH_2.

Based on the set of candidate seasonal 2-event patterns P, we determine whether P is a frequent seasonal 2-event pattern by checking the constraints of *maxPeriod*, *minDensity*, *distInterval* and *minSeason* as in the case of single events, using the support set SUP^P retrieved from the value of PH_2.

4.2.2 Mining frequent seasonal k-event patterns: Let $N_{k-1} = (E_1, ..., E_{k-1})$ be a candidate (k-1)-event group in HLH_{k-1}, $N_1 = (E_k)$ be a candidate single event in HLH_1, and $N_k = N_{k-1} \cup N_1 = (E_1, ..., E_k)$ be a candidate k-event in HLH_k. To find k-event patterns for N_k, we first retrieve the set of candidate (k-1)-event patterns $\mathcal{P}_{k-1}$ by accessing the EH_{k-1} table. We check whether P_{k-1} and E_k can form a k-event pattern P_k as follows.

We have $P_{k-1} = \{(r_{12}, E_1, E_2),...,(r_{(k-2)(k-1)}, E_{k-2}, E_{k-1})\}$. First, we start with the triple $(r_{(k-1)k}, E_{k-1}, E_k)$. If $(r_{(k-1)k}, E_{k-1}, E_k)$ does not exist in HLH_2, then P_k is not a candidate k-event pattern, and the verification stops immediately. Otherwise, we continue the similar verification on the triple $(r_{(k-2)k}, E_{k-2}, E_k)$, until it reaches (r_{1k}, E_1, E_k). Next, we compute $maxSeason(P_k)$ to determine whether P_k is a candidate k-event pattern, i.e., $maxSeason(P_k) \geq minSeason$. The candidate k-event patterns are maintained in PH_k and GH_k. Finally, we mine frequent seasonal k-event patterns from the found candidates, similar to 2-event patterns.

Using transitivity property to optimize candidate k-event groups: In Section 4.1, when mining candidate k-event groups, we perform the Cartesian product between F_{k-1} and F_1. However, using the candidate single events in F_1 to generate k-event groups can create redundancy, since events in F_1 when combined with F_{k-1} might not form any frequent seasonal k-event patterns. For example, consider the event F:0 in HLH_1 in Fig. C.6. Here, F:0 is a candidate single event, and thus, can be combined with 2-event groups in HLH_2 such as (C:1, D:1) to create a 3-event group (C:1, D:1, F:0). However, no candidate seasonal 3-event patterns can be formed by (C:1, D:1, F:0) since F:0 does not exist in any candidate 2-event patterns in HLH_2. We use the *transitivity property* of temporal relations to reduce the redundancy as below.

Lemma 3 *Let $S =< e_1,..., e_{k-1} >$ be a temporal sequence, $P =< (r_{12}, E_{1_{\triangleright e_1}}, E_{2_{\triangleright e_2}})$ $,..., (r_{(k-2)(k-1)}, E_{k-2_{\triangleright e_{k-2}}}, E_{k-1_{\triangleright e_{k-1}}}) >$ be a (k-1)-event pattern that occurs in S, e_k be a new event instance, $S^{'} =< e_1,..., e_k >$ be a new temporal sequence created by adding e_k to S, and $\mathfrak{R}$ be the set of temporal relations. $\mathfrak{R}$ is transitive on $S^{'}$: $\forall e_i \in S^{'}, i < k$, $\exists r \in \mathfrak{R}$ s.t. $r(E_{i_{\triangleright e_i}}, E_{k_{\triangleright e_k}})$ hold.*

Lemma 3 states the temporal transitivity property between temporal events, and is used to prove lemma 4.

Lemma 4 *Let $N_{k-1} = (E_1, ..., E_{k-1})$ be a candidate seasonal (k-1)-event group, and E_k be a candidate seasonal single event. If $\forall E_i \in N_{k-1}$, $\exists r \in \mathfrak{R}$ s.t. $r(E_i, E_k)$ is a candidate seasonal relation, then $N_k = N_{k-1} \cup E_k$ can form candidate seasonal k-event patterns.*

From Lemma 4, only single events in HLH_1 that occur in HLH_{k-1} should be used to create k-event groups. We identify these single events by filtering F_1, and creating the set *FilteredF1*. Then, the Cartesian product $F_{k-1} \times F_1$ is replaced by $F_{k-1} \times$ *FilteredF1* to generate k-event groups.

Complexity: Let n be the number of seasonal events in HLH_1, i be the average number of instances of each seasonal event, r be the number of seasonal (k-1)-event patterns in HLH_{k-1}, and u be the average number of granules of each event/ temporal relation. The complexity of frequent seasonal k-event pattern mining is $O(n^2 i^2 u^2) + O(|F_1| \cdot |F_{k-1}| \cdot r \cdot k^2 \cdot u)$.

Proof *(Sketch - Full proof in [36]). Computing maxSeason of 2-event patterns takes $O(n^2 i^2 u^2)$. Identifying the set of seasons $\mathcal{PS}$ of candidate 2-event patterns takes $O(n^2 u)$. The frequent seasonal 2-event pattern mining has the complexity: $O(n^2 i^2 u^2 + n^2 u) \sim O(n^2 i^2 u^2)$. Computing maxSeason of k-event patterns $(k > 2)$ takes $O(|F_1| \cdot |F_{k-1}| \cdot r \cdot k^2 \cdot u)$. Identifying the set of seasons $\mathcal{PS}$ of candidate k-event patterns takes $O(|F_1| \cdot |F_{k-1}| \cdot r \cdot u)$. The frequent seasonal k-event pattern mining has the complexity: $O(|F_1| \cdot |F_{k-1}| \cdot r \cdot k^2 \cdot u + |F_1| \cdot |F_{k-1}| \cdot r \cdot u) \sim O(|F_1| \cdot |F_{k-1}| \cdot r \cdot k^2 \cdot u)$. Thus, the total time complexity is $O(n^2 i^2 u^2) + O(|F_1| \cdot |F_{k-1}| \cdot r \cdot k^2 \cdot u)$.*

STPM overall complexity: The space complexity of STPM is $O(n^h 3^{h^2})$. The time complexity of STPM depends on the size of the search space $O(n^h 3^{h^2})$, i.e., STPM scales exponentially with quadratic exponent in the pattern length h, and on the complexity of the mining process itself, i.e., $O(n \cdot |\mathcal{D}_{\text{SEQ}}|) + O(n^2 i^2 u^2) + O(|F_1| \cdot |F_{k-1}| \cdot r \cdot k^2 \cdot u)$. While the parameters $|F_1|$, $|F_{k-1}|$ and u depend on the number of temporal sequences, others such as n, h, i, r and k

depend on the number of time series. Thus, STPM space and time complexities are driven by the sizes of $\mathcal{D}_{\text{SEQ}}$ and $\mathcal{D}_{\text{SYB}}$.

C.5 Approximate STPM

C.5.1 Correlated Symbolic Time Series

Consider two time series X and Y, and their corresponding symbolic series X_S, Y_S, and symbolic alphabets Σ_X and Σ_Y.

Definition 5.1 (Entropy [3]) The *entropy* $H(X_S)$ of X_S measures the uncertain degree of the outcomes of X_S [38], and is computed as

$$H(X_S) = -\sum_{x \in \Sigma_X} p(x) \cdot \log p(x) \tag{C.2}$$

where $p(x)$ is the probability of X_S.

The *conditional entropy* $H(X_S|Y_S)$ is defined as

$$H(X_S|Y_S) = -\sum_{x \in \Sigma_X} \sum_{y \in \Sigma_Y} p(x,y) \cdot \log \frac{p(x,y)}{p(y)} \tag{C.3}$$

where $p(x,y)$ is the joint probability of (X_S, Y_S), and $p(y)$ is the probability of Y_S.

Definition 5.2 (Mutual information [3]) The *mutual information* (MI) $I(X_S;Y_S)$ of X_S and Y_S represents the amount of shared information between X_S and Y_S, and is defined as

$$I(X_S;Y_S) = \sum_{x \in \Sigma_X} \sum_{y \in \Sigma_Y} p(x,y) \cdot \log \frac{p(x,y)}{p(x) \cdot p(y)} \tag{C.4}$$

Since $0 \leq I(X_S;Y_S) \leq \min\{H(X_S), H(Y_S)\}$ [38], the MI upper bound does not exist. We normalize MI to scale it into the range $[0,1]$.

Definition 5.3 (Normalized mutual information [3]) The *normalized mutual information* (NMI) $\widetilde{I}(X_S;Y_S)$ of X_S and Y_S represents the percentage of shared information between X_S and Y_S, and is defined as

$$\widetilde{I}(X_S;Y_S) = \frac{I(X_S;Y_S)}{H(X_S)} = 1 - \frac{H(X_S|Y_S)}{H(X_S)} \tag{C.5}$$

Based on Eq. (C.5), X_S and Y_S are dependent if $\widetilde{I}(X_S;Y_S) > 0$. Moreover, Eq. (C.5) also shows that NMI is not symmetric, i.e., $\widetilde{I}(X_S;Y_S) \neq \widetilde{I}(Y_S;X_S)$.

Definition 5.4 (Correlated symbolic time series [3]) Let μ where $0 < \mu \leq 1$ be the MI threshold. The series X_S and Y_S are *correlated* iff $\min\{\widetilde{I}(X_S;Y_S), \widetilde{I}(Y_S;X_S)\} \geq \mu$, and *uncorrelated* otherwise.

C.5.2 Lower Bound of the maxSeason

Let X_S and Y_S be two symbolic series, X_1 be a temporal event in X_S, Y_1 be a temporal event in Y_S, $\mathcal{D}_{\text{SYB}}$ and $\mathcal{D}_{\text{SEQ}}$ be the symbolic and the sequence databases created from X_S and Y_S, respectively. The relation between $\widetilde{I}(X_S;Y_S)$ in $\mathcal{D}_{\text{SYB}}$, and $maxSeason(X_1, Y_1)$ in $\mathcal{D}_{\text{SEQ}}$ is established as follows.

Theorem 1 *(Lower bound of the maximum seasonal occurrence) Let μ be the mutual information threshold. If the NMI $\widetilde{I}(X_S;Y_S) \geq \mu$, then the maximum seasonal occurrence of (X_1, Y_1) in $\mathcal{D}_{SEQ}$ has a lower bound:*

$$maxSeason(X_1, Y_1) \geq \frac{\lambda_2 \cdot |\mathcal{D}_{SEQ}|}{minDensity} \cdot e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot ln2}{\lambda_2}\right)} \tag{C.6}$$

where: $\lambda_1 = \min\{p(X_i), \forall X_i \in X_S\}$ is the minimum probability of $X_i \in X_S$, and $\lambda_2 = p(Y_1)$ is the probability of $Y_1 \in Y_S$, and W is the Lambert function [39].

Proof *(Sketch - Full proof in [36]). From Eq.* (C.5), *we have:*

$$\widetilde{I}(X_S;Y_S) = 1 - \frac{H(X_S|Y_S)}{H(X_S)} \geq \mu \tag{C.7}$$

$$\Rightarrow \frac{H(X_S|Y_S)}{H(X_S)} = \frac{p(X_1, Y_1) \cdot \log p(X_1|Y_1)}{\sum_i p(X_i) \cdot \log p(X_i)} + \frac{\sum_{i\neq 1 \& j\neq 1} p(X_i, Y_j) \cdot \log \frac{p(X_i,Y_j)}{p(Y_j)}}{\sum_i p(X_i) \cdot \log p(X_i)} \leq 1 - \mu \tag{C.8}$$

Let: $\lambda_1 = \min\{p(X_i), \forall i\}$, $\lambda_2 = p(Y_1)$.

$$\frac{H(X_S|Y_S)}{H(X_S)} \geq \frac{p(X_1, Y_1) \cdot \log \frac{p(X_1,Y_1)}{\lambda_2}}{\log \lambda_1} \tag{C.9}$$

From Eqs. (C.8), (C.9), *we derive:* $p(X_1, Y_1) \geq \lambda_2 \cdot e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot \ln 2}{\lambda_2}\right)}$

Since:

$$\frac{\left|SUP^{(X_1,Y_1)}\right|}{|\mathcal{D}_{SEQ}|} \geq p(X_1, Y_1) \geq \lambda_2 \cdot e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot \ln 2}{\lambda_2}\right)}$$

Thus:

$$maxSeason(X_1, Y_1) \geq \frac{\lambda_2 \cdot |\mathcal{D}_{SEQ}|}{minDensity} \cdot e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot \ln 2}{\lambda_2}\right)} \tag{C.10}$$

Algorithm 12: Approximate STPM using MI

Input: A set of time series $\mathcal{X}$, the thresholds: $maxPeriod$, $minDensity$, $distInterval$, $minSeason$

Output: All frequent seasonal temporal patterns $\mathcal{P}$

foreach *pair of series* $(X_S, Y_S) \in \mathcal{D}_{SYB}$ **do**
- $minNMI \leftarrow \min\{\widetilde{I}(X_S; Y_S), \widetilde{I}(Y_S; X_S)\}$;
- Compute μ using Eq. (C.11);
- **if** $minNMI \geq \mu$ **then**
  - Insert X_S and Y_S into $\mathcal{X}_C$;

Mine frequent seasonal single events from $\mathcal{X}_C$;

foreach $(X_S, Y_S) \in \mathcal{X}_C$ **do**
- Mine frequent seasonal 2-event patterns from (X_S, Y_S);

if $k \geq 3$ **then**
- Perform STPM using HLH_1 and HLH_{k-1};

Setting the parameters: To compute the lower bound of *maxSeason*(X_1, Y_1) in Eq. (C.6), several parameters need to be defined: λ_1, λ_2, and μ. Given $\mathcal{D}_{\text{SYB}}$, λ_1 and λ_2 can easily be determined since λ_1 is the minimum probability among all events $X_i \in X_S$, and λ_2 is the probability of $Y_1 \in Y_S$. To set the value of μ, we use the lower bound of *maxSeason* in Theorem 1 to derive μ as follows.

Corollary 1.1 *The maximum seasonal occurrence of an event pair* $(X_1, Y_1) \in (X_S, Y_S)$ *in* $\mathcal{D}_{SEQ}$ *is at least minSeason if* $\widetilde{I}(X_S; Y_S)$ *is at least* μ*, where:*

$$\mu \geq \begin{cases} 1 - \frac{\lambda_2}{e \cdot \ln 2 \cdot \log \frac{1}{\lambda_1}}, & \text{if } 0 \leq \rho \leq \frac{1}{e} \\ 1 - \frac{\rho \cdot \lambda_2 \cdot \log \rho}{\ln 2 \cdot \log \lambda_1}, & \text{otherwise} \end{cases}, \text{where } \rho = \frac{minSeason \cdot minDensity}{\lambda_2 \cdot |\mathcal{D}_{SEQ}|} \tag{C.11}$$

Interpretation of the lower bound of the maximum seasonal occurrence: Theorem 1 says that, given a mutual information threshold μ, if the two series X_S and Y_S are correlated, i.e., $\widetilde{I}(X_S; Y_S) \geq \mu$, then the maximum seasonal occurrence of an event pair in (X_S,Y_S) is at least the lower bound in Eq. (C.6). Combining Theorem 1 and Lemma 2, we can conclude that given a pair of symbolic series (X_S,Y_S), if its event pair (X_1, Y_1) has a maximum seasonal occurrence less than the lower bound in Eq. (C.6), then any 2-event pattern P formed by that event pair also has a maximum seasonal occurrence less than that lower bound. This allows us to construct the approximate STPM algorithm, discussed in the next section.

C.5.3 Using the Bound to Approximate STPM

Approximate STPM: We construct an approximate version of STPM using Theorem 1. Specifically, using the STPM thresholds *minSeason* and *minDensity*, we derive μ (Eq. C.11) and use it to identify *correlated symbolic series* (defined in Def. 5.4). Next, the approximate STPM performs the mining only on *the* *set of correlated symbolic series* $\mathcal{X}_C \subseteq \mathcal{X}$. Algorithm 12 outlines the approximate STPM.

First, NMI and μ for each pair (X_S, Y_S) in $\mathcal{D}_{\text{SYB}}$ are computed (lines 2-3). Then, we filter and select only the pairs that have $\min\{\widetilde{I}(X_S; Y_S), \widetilde{I}(Y_S; X_S)\} \geq \mu$. The selected pairs are inserted into $\mathcal{X}_C$. Next, we use only the series in $\mathcal{X}_C$ to mine frequent seasonal single events (line 6). For frequent seasonal 2-event patterns, we mine frequent seasonal patterns only from event pairs in $\mathcal{X}_C$ (lines 7-8). For frequent seasonal k-event patterns ($k \geq 3$), the exact STPM is used (lines 9-10).

Complexity analysis of approximate STPM: The approximate STPM differs from STPM in two mining steps, the seasonal single events at HLH_1 and the seasonal 2-event patterns at HLH_2 by mining those only from correlated time series. The approximate STPM only scan $\mathcal{D}_{\text{SYB}}$ once to calculate the probability for single events and event pairs and compute NMI and μ. Hence, the cost of computing NMI and μ is $O(|\mathcal{D}_{\text{SYB}}|)$. In contrast, the complexities of the exact STPM at HLH_1 and HLH_2 are $O(n \cdot |\mathcal{D}_{\text{SEQ}}|) + O(n^2 i^2 u^2)$ (Sections C.4.3 and C.4.4). Thus, the more time series are pruned, the faster and less memory usage of the approximate STPM. However, overall, the approximate STPM still scales exponentially with quadratic exponent in the pattern length h as in STPM.

Table C.5: Characteristics of the Datasets

Datasets	#seq.	#time series	#events	#ins./seq.
RE (real)	1,460	21	102	93
SC (real)	1,249	14	56	55
INF (real)	608	25	124	48
HFM (real)	730	24	115	40
RE (syn.)	1,460 $\times 10^3$	10^4	48,500	38,012
SC (syn.)	1,249 $\times 10^3$	10^4	40,020	37,106
INF (syn.)	608 $\times 10^3$	10^4	49,600	40,623
HFM (syn.)	730 $\times 10^3$	10^4	47,825	41,241

C.6 Experimental Evaluation

C.6.1 Experimental Setup

Datasets: We use three real-world datasets from three application domains: renewable energy, smart city, and health. For *renewable energy* (RE), we use energy data [40] and weather data [6] from Spain. For *smart city* (SC), we use traffic and weather datasets [41] from New York City. For *health*, we combine the *influenza* (INF) and *hand-foot-mouth* (HFM) datasets [5] and weather data [6]

Table C.6: Parameters and values

Params	Values (User-defined)
maxPeriod	0.2%, 0.4%, 0.6%, 0.8%, 1.0%
minDensity	0.5%, 0.75%, 1.0%, 1.25%, 1.5%
minSeason	4, 8, 12, 16, 20
distInterval	[90, 270] (RE, SC), [30, 90] (INF, HFM)

from Kawasaki, Japan. Besides real-world datasets, we also generate synthetic data for the scalability evaluation. Specifically, starting from each real-world dataset, we generate $1,000$ *times more sequences* and $10,000$ *synthetic time series* for each of them. Table C.5 summarizes the dataset characteristics.

Baseline method: Our exact method is denoted E-STPM, and the approximate one is denoted A-STPM. Since our work is the first that studies frequent seasonal temporal pattern mining, there does not exist an exact baseline to compare against STPM. However, we adapt the state-of-the-art method for recurring itemset mining PS-growth [34] to find seasonal temporal patterns. Specifically, the adaptation is done through 2-phase process: (1) PS-growth is applied to find frequent recurring events, and (2), mine temporal patterns from extracted events. The adapted PS-growth is referred to as APS-growth.

Infrastructure: We use a VM with 32 AMD EPYC cores (2GHz), 512 GB RAM, and 1 TB storage.

Parameters: Table C.6 shows the parameters and their values used in our experiments, where *maxPeriod* and *minDensity* are expressed as the percentage of $\mathcal{D}_{\text{SEQ}}$. While the four parameters in Table C.6 are user-defined, we also provide the intuition of how to set them. *maxPeriod* determines how close the patterns should occur within the same season. The smaller the *maxPeriod*, the closer the occurred patterns should be and vice versa. *minDensity* decides how dense a season should be. Combining these two, a small *maxPeriod* and a large *minDensity* will find dense seasons with close-by pattern occurrences. In contrast, a large *maxPeriod* and a small *minDensity* will find sparse seasons. On the other hand, *minSeason* and *distInterval* values often depend on the granularity of $\mathcal{D}_{\text{SEQ}}$. For example, if $\mathcal{D}_{\text{SEQ}}$ has month granularity, we then can look for patterns with yearly seasonality. Thus, *distInterval* is often between 3 and 9 months, and *minSeason* is the minimum number of years the patterns should have occurred seasonally.

C.6.2 Qualitative Evaluation

Table C.7 shows some seasonal patterns mined from the datasets. Patterns P1-P3 are extracted from RE, showing that high renewable energy generation and high electricity demand occur seasonally and often at specific *season* throughout the year. Patterns P4-P7 are extracted from INF and HFM, showing the detection of seasonal diseases. Finally, how weather affects traffic is shown in patterns P8-P11 extracted from SC.

Table C.7: Summary of Interesting Seasonal Patterns

Patterns	minDensity (%)	maxPeriod (%)	# minSeason	Seasonal occurrence
(P1) Strong Wind ⩾ High Wind Power Generation	0.5	0.4	12	December, January, February
(P2) Low Temperature ⩾ High Energy Consumption	0.5	0.4	12	December, January, February
(P3) Very Few Clouds ⩾ Very High Temperature ≬ High Solar Power Generation	0.75	0.6	8	July, August
(P4) High Humidity ≬ Very Low Temperature → Very High Influenza Cases	0.5	0.4	12	January, February
(P5) Strong Wind ⩾ Heavy Rain ⩾ High Influenza Cases	0.5	0.4	12	January, February
(P6) Low Humidity ⩾ High Temperature ⩾ Very High Hand-Foot-Mouth Disease Cases	1.0	0.6	12	May, June
(P7) Very High Temperature ⩾ High Wind ⩾ High Hand-Foot-Mouth Disease Cases	1.0	0.6	12	May, June
(P8) High Temperature ⩾ Strong Wind → High Congestion	0.5	0.6	8	July, August
(P9) Strong Wind ⩾ Unclear Visibility ⩾ High Congestion	0.5	0.6	8	July, August
(P10) Heavy Rain ⩾ Unclear Visibility ⩾ High Lane-Blocked	0.4	0.8	8	July, August
(P11) Heavy Rain ⩾ Strong Wind ⩾ High Flow-Incident	0.4	0.8	8	July, August

Table C.8: The Number of Seasonal Patterns on RE

maxPeriod (%)	minSeason (#) - minDensity (%)								
	8-0.5	**8-0.75**	**8-1.0**	**12-0.5**	**12-0.75**	**12-1.0**	**16-0.5**	**16-0.75**	**16-1.0**
0.2	35626	20427	11339	21309	12941	6935	8045	4218	3018
0.4	41462	29729	14281	25207	17381	7294	10261	7480	5483
0.6	48651	35018	16247	31860	24627	9826	14061	9738	7409

Table C.9: The Number of Seasonal Patterns on INF

maxPeriod (%)	minSeason (#) - minDensity (%)								
	8-0.5	**8-0.75**	**8-1.0**	**12-0.5**	**12-0.75**	**12-1.0**	**16-0.5**	**16-0.75**	**16-1.0**
0.2	7812	5704	4285	5159	3163	2157	3521	2105	1284
0.4	10581	8294	6535	7952	5863	4068	5293	4618	2690
0.6	12084	9618	8260	11850	8591	6028	6809	5073	3529

Tables C.8 and C.9 list the number of seasonal patterns found in the RE and INF datasets. It can be seen that high *minSeason* leads to less generated patterns, as many have few seasonal occurrences. Moreover, high *minDensity* also generates fewer patterns since only few patterns have high occurrence density. Finally, high *maxPeriod* results in more generated patterns, since high *maxPeriod* allows more temporal relations to be formed, thus increasing the number of patterns.

C.6.3 Quantitative Evaluation

Comparison with baseline on real-world datasets

We compare the runtime and memory usage of E-STPM and A-STPM with the baseline. Figs. C.7, C.8, C.9 and C.10 show the comparison on RE and INF datasets.

As shown in Figs. C.7 and C.8, A-STPM is the fastest of all methods, and E-STPM is faster than the baseline. The range and average speedups of A-STPM over the other methods are: [1.5-4.7] and 2.6 (E-STPM), and [5.2-10.6] and 7.1 (APS-growth). The speedup of E-STPM over the baseline is [3.5-7.2]

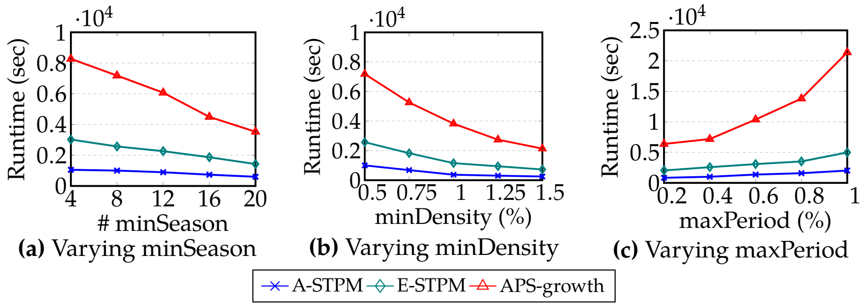


Fig. C.7: Runtime Comparison on RE (real-world)

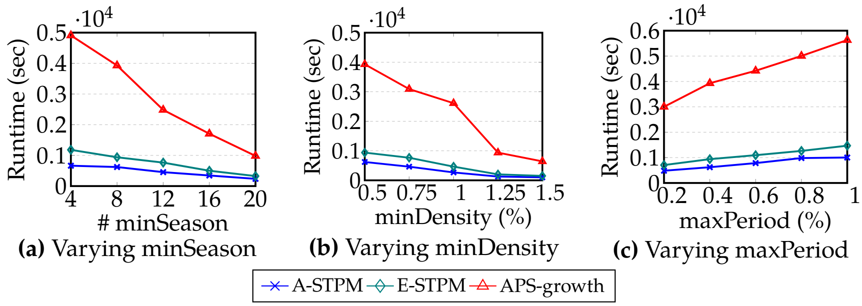


Fig. C.8: Runtime Comparison on INF (real-world)

and 4.3 on average. Note that the times to compute MI and μ for RE and INF in Figs. C.7 and C.8 are only 2.6 and 1.4 seconds, respectively. Moreover, A-STPM is the most efficient, with the highest speedup and memory saving, when the *minSeason* threshold is low, e.g., *minSeason* = 4. This is because there are typically many patterns with few seasonal occurrences. Thus, A-STPM's early pruning of uncorrelated time series saves both memory and runtime. This speedup however incurs a slightly lower accuracy (discussed in Section "Evaluation of A-STPM").

As shown in Figs. C.9 and C.10, A-STPM uses the least memory, while E-STPM uses less memory than the baseline. The range and average of A-STPM's memory consumption compared to other methods are: [1.4-2.7] and 1.8 (E-STPM), and [2.7-7.6] and 3.9 (APS-growth). The memory usage of E-STPM compared to the baseline is [1.5-4.1] and 2.3 on average.

Scalability on synthetic datasets

As discussed in Section C.4, STPM complexity is driven by two main factors, namely the number of temporal sequences and time series, respectively. Thus, to further evaluate STPM scalability, we scale these two factors on synthetic datasets (reported in Table C.5). Specifically, we vary the number of sequences and time series separately.

Figs. C.11 and C.12 show the runtimes of A-STPM, E-STPM and the baseline

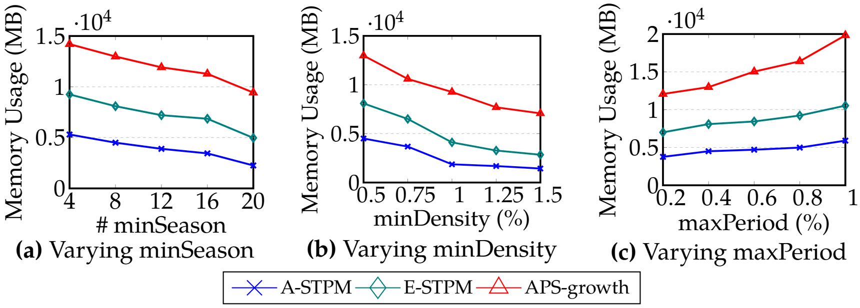


Fig. C.9: Memory Usage Comparison on RE (real-world)

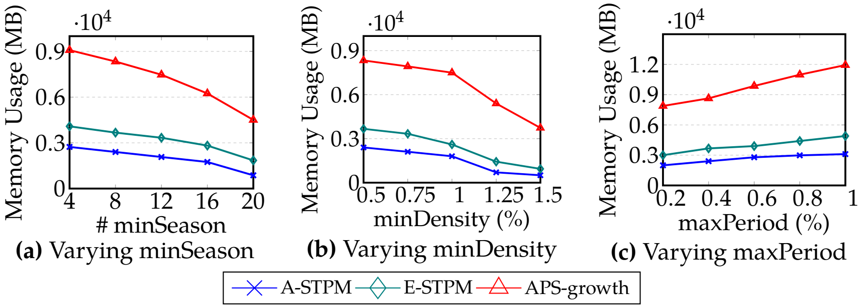
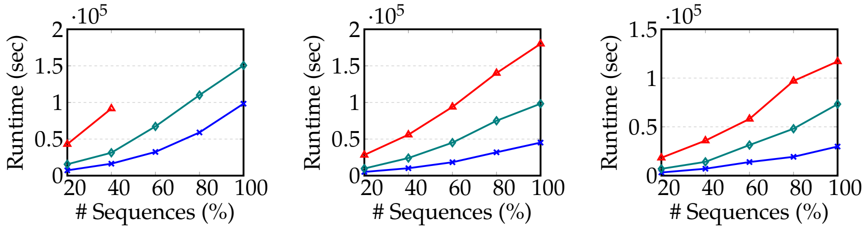


Fig. C.10: Memory Usage Comparison on INF (real-world)

when the number of sequences changes. We obtain the range and average speedups of A-STPM are: [1.6-3.2] and 2.2 (E-STPM), and [3.1-6.4] and 4.6 (APS-growth). Similarly, the range and average speedup of E-STPM compared to APS-growth is [1.9-4.3] and 3.2. We note that the baseline fails for larger configurations because of memory in this scalability study, i.e., on the synthetic RE at 60% sequences ($\approx 8 \times 10^5$) (Fig. C.11a) and on the synthetic INF at 100% sequences ($\approx 6 \times 10^5$) (Fig. C.12a), showing that A-STPM and E-STPM can scale well on big datasets while the baseline cannot.

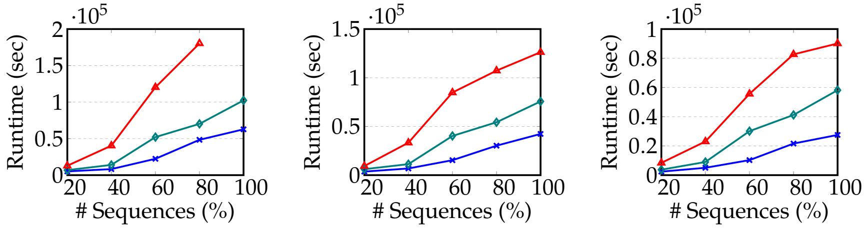
Figs. C.13 and C.14 compare the runtimes of A-STPM, E-STPM and APS-growth when changing the number of time series. We obtain the range and average speedups of A-STPM are: [1.7-3.5] and 2.3 (E-STPM), and [3.8-9.5] and 5.3 (APS-growth), and of E-STPM is [2.3-4.4] and 3.6 (APS-growth). The baseline also fails at large configurations in this study, i.e., when # Time Series $\geq$ 6000 on the synthetic RE (Fig. C.13a), and $\geq$ 8000 on the synthetic INF (Fig. C.14a).

Furthermore, we provide the computation time of MI and μ in Figs. C.13 and C.14 by adding an additional bar chart for A-STPM. Each bar has two separate components: the MI and μ computation time (top red), and the mining time (bottom blue). We only need to compute MI once for each dataset, (the computed MIs are used across different *minSeason* and *minDensity* thresholds), while the computation of μ is negligible (in milliseconds using Eq. (C.11)).



(a) minSeason=12, minDensity=0.5% (b) minSeason=16, minDensity=0.75% (c) minSeason=20, minDensity=1.0%

Fig. C.11: Scalability: Varying #Sequences on RE (synthetic)



(a) minSeason=12, minDensity=0.5% (b) minSeason=16, minDensity=0.75% (c) minSeason=20, minDensity=1.0%

Fig. C.12: Scalability: Varying #Sequences on INF (synthetic)

Thus, the MI and μ computation times, for example, in Figs. C.13a, C.13b, and C.13c, are not all actually used and added for comparison only.

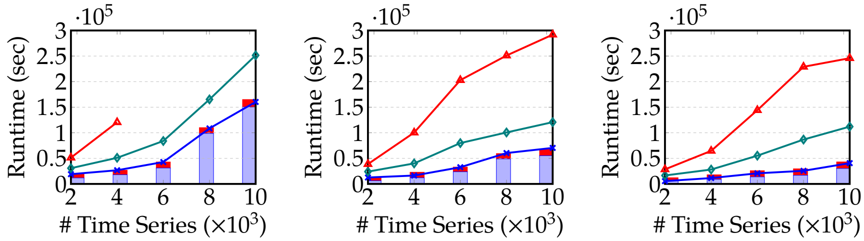
Table C.10: Pruned Time Series and Events from A-STPM

# Attr.	RE						INF					
	Pruned Time Series (%)			Pruned Events (%)			Pruned Time Series (%)			Pruned Events (%)		
	12-0.5%	16-0.75%	20-1.0%	12-0.5%	16-0.75%	20-1.0%	12-0.5%	16-0.75%	20-1.0%	12-0.5%	16-0.75%	20-1.0%
2000	35.20	32.10	26.80	27.22	23.53	19.03	42.60	36.75	29.70	28.63	26.12	22.10
4000	33.05	29.15	22.05	25.24	22.41	17.95	35.70	31.03	24.80	27.35	25.77	22.01
6000	30.25	26.32	19.55	24.75	21.60	17.28	33.22	28.78	22.13	26.98	25.29	20.81
8000	29.48	25.38	19.15	24.70	21.12	16.96	31.75	28.51	21.58	26.74	24.52	20.74
10000	28.59	24.87	18.91	24.50	21.07	16.69	31.06	26.48	21.15	26.61	24.36	20.27

Finally, we provide the percentage of events and time series pruned by A-STPM in the scalability test in Table C.10. We see that more time series (events) are pruned for low *minSeason* and *minDensity*, since *minSeason* and *minDensity* have an inverse relationship with μ, therefore, low *minSeason* and *minDensity* result in higher μ, and thus, more pruned time series.

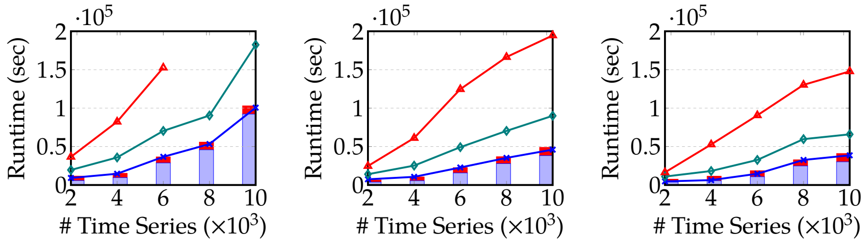
Evaluation of the pruning techniques in E-STPM

We now compare different E-STPM versions to understand how effective each of the proposed pruning techniques are: (1) NoPrune: E-STPM with no pruning, (2) Apriori: E-STPM with Apriori-liked pruning (Lemmas 1, 2), (3) Trans:



(a) minSeason=12, minDensity=0.5%
(b) minSeason=16, minDensity=0.75%
(c) minSeason=20, minDensity=1.0%

Fig. C.13: Scalability: Varying #TimeSeries on RE (synthetic)



(a) minSeason=12, minDensity=0.5%
(b) minSeason=16, minDensity=0.75%
(c) minSeason=20, minDensity=1.0%

Fig. C.14: Scalability: Varying #TimeSeries on INF (synthetic)

E-STPM with transitivity-based pruning (Lemmas 3, 4), and (4) All: E-STPM applied both pruning techniques.

Figs. C.15, C.16 show the results. We see that (All)-E-STPM has the best performance of all versions, with a speedup over (NoPrune)-E-STPM ranging from 3 up to 6, depending on the exact configuration. Thus, the proposed prunings improve E-STPM performance significantly. Specifically, (Trans)-E-STPM yields larger speedup than (Apriori)-E-STPM, with average speedups between 2 to 5 for (Trans)-E-STPM, and between 1.5 to 4 for (Apriori)-E-STPM, but applying both yields the best speedup.

Evaluation of A-STPM

We now evaluate the accuracy of A-STPM by comparing the patterns extracted by A-STPM and E-STPM. Table C.11 shows the accuracies of A-STPM for different *minSeason* and *minDensity* on the real-world datasets. It is seen that, A-STPM obtains high accuracy ($\geq$ 81%) when *minSeason* and *minDensity* are low, e.g., $minSeason = 8$ and $minDensity = 0.5\%$, and very high accuracy ($\geq$ 95%) when *minSeason* and *minDensity* are high, e.g., $minSeason = 16$ and $minDensity = 0.75\%$.

Similarly, Table C.12 shows the accuracies of A-STPM on the synthetic datasets: very high accuracy ($\geq$ 96%) when *minSeason* and *minDensity* are

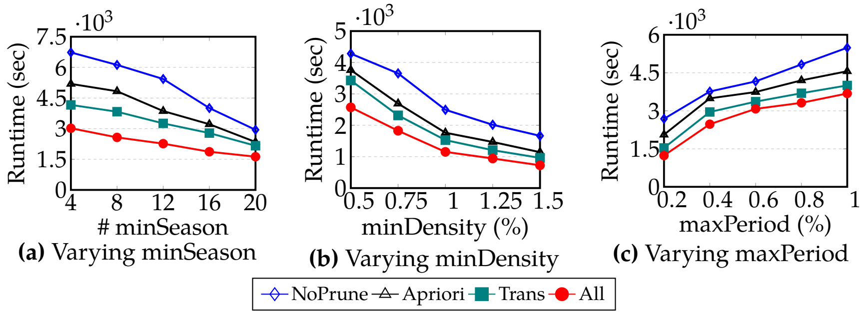


Fig. C.15: Pruning Techniques of E-STPM on RE (real-world)

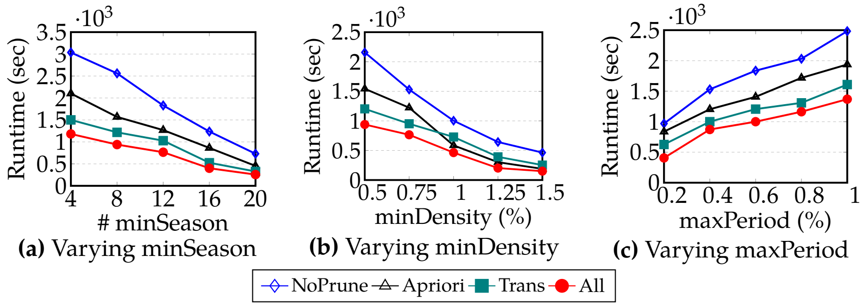


Fig. C.16: Pruning Techniques of E-STPM on INF (real-world)

Table C.11: A-STPM Accuracy

# minSeason	**minDensity (%)**					
	RE (real)			**INF (real)**		
	0.5	**0.75**	**1**	**0.5**	**0.75**	**1**
8	81	82	86	81	83	87
12	84	86	92	88	90	93
16	94	95	100	95	96	100
20	97	100	100	100	100	100

Table C.12: The Accuracy of A-STPM on Syn. Data

# Attr.	**RE**			**INF**		
	Accuracy (%)			**Accuracy (%)**		
	12-0.5%	**16-0.75%**	**20-1.0%**	**12-0.5%**	**16-0.75%**	**20-1.0%**
2000	85	96	100	89	96	100
4000	86	96	100	90	98	100
6000	86	96	100	91	98	100
8000	88	97	100	93	98	100
10000	89	98	100	93	98	100

high, e.g., *minSeason* = 16 and *minDensity* = 0.75%.

C.7 Conclusion and Future Work

This paper presents our efficient Frequent Seasonal Temporal Pattern Mining from Time Series (FreqSTPfTS) approach that offers: (1) the first solution for Seasonal Temporal Pattern Mining (STPM), (2) the exact Seasonal Temporal Pattern Mining (E-STPM) algorithm which employs efficient pruning techniques and data structures, and (3) the approximate A-STPM which prunes unpromising time series using mutual information, making STPM scale on big datasets. Our comprehensive experimental evaluation on real-world and synthetic datasets shows that A-STPM and E-STPM outperform the baseline, consuming less memory and scaling well. The approximate A-STPM delivers up to an order of magnitude speedup over the baseline. In future work, STPM will be extended to do event-level pruning.

References

[1] H. T. Lam, F. Mörchen, D. Fradkin, and T. Calders, "Mining compressing sequential patterns," *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 7, no. 1, pp. 34–52, 2014.

[2] J.-W. Huang, C.-Y. Tseng, J.-C. Ou, and M.-S. Chen, "A general model for sequential pattern mining with a progressive database," *TKDE*, vol. 20, no. 9, pp. 1153–1167, 2008.

[3] V. L. Ho, N. Ho, and T. B. Pedersen, "Efficient temporal pattern mining in big time series using mutual information," in *PVLDB*, vol. 15, no. 3, 2021.

[4] Z. Lee, T. Lindgren, and P. Papapetrou, "Z-miner: an efficient method for mining frequent arrangements of event intervals," in *Proceedings of the 26th ACM SIGKDD*, 2020, pp. 524–534.

[5] K. city infectious disease surveillance system. (2021) Kidss. [Online]. Available: https://kidss.city.kawasaki.jp/

[6] O. Weather. (2021) Open weather. [Online]. Available: https://openweathermap.org/

[7] R. U. Kiran, H. Shang, M. Toyoda, and M. Kitsuregawa, "Discovering recurring patterns in time series," in *EDBT*, 2015, pp. 97–108.

[8] R. U. Kiran, C. Saideep, K. Zettsu, M. Toyoda, M. Kitsuregawa, and P. K. Reddy, "Discovering partial periodic spatial patterns in spatiotemporal databases," in *Big Data*. IEEE, 2019, pp. 233–238.

[9] J. Han, W. Gong, and Y. Yin, "Mining segment-wise periodic patterns in time-related databases." in *KDD*, vol. 98, 1998, pp. 214–218.

[10] J. Han, G. Dong, and Y. Yin, "Efficient mining of partial periodic patterns in time series database," in *ICDE*. IEEE, 1999, pp. 106–115.

[11] J. Assfalg, T. Bernecker, H.-P. Kriegel, P. Kröger, and M. Renz, "Periodic pattern analysis in time series databases," in *DASFAA*. Springer, 2009, pp. 354–368.

[12] M. Zhang, P. Wang, and W. Wang, "Efficient consensus motif discovery of all lengths in multiple time series," in *DASFAA*. Springer, 2022, pp. 540–555.

[13] H. Liu, F. Han, H. Zhou, X. Yan, and K. S. Kosik, "Fast motif discovery in short sequences," in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 2016, pp. 1158–1169.

[14] Y. Mohammad and T. Nishida, "Approximately recurring motif discovery using shift density estimation," in *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*. Springer, 2013, pp. 141–150.

[15] L. Kegel, C. Hartmann, M. Thiele, and W. Lehner, "Season-and trend-aware symbolic approximation for accurate and efficient time series matching," *Datenbank-Spektrum*, vol. 21, no. 3, pp. 225–236, 2021.

[16] S. K. Tanbeer, C. F. Ahmed, B.-S. Jeong, and Y.-K. Lee, "Discovering periodic-frequent patterns in transactional databases," in *PAKDD*. Springer, 2009, pp. 242–253.

[17] R. Uday Kiran and P. Krishna Reddy, "Towards efficient mining of periodic-frequent patterns in transactional databases," in *International Conference on Database and Expert Systems Applications*. Springer, 2010, pp. 194–208.

[18] K. Amphawan, P. Lenca, and A. Surarerks, "Mining top-k periodic-frequent pattern from transactional databases without support threshold," in *International conference on advances in information technology*. Springer, 2009, pp. 18–29.

[19] T. T. N. Ho and B. Pernici, "A data-value-driven adaptation framework for energy efficiency for data intensive applications in clouds," in *2015 IEEE conference on technologies for sustainability (SusTech)*. IEEE, 2015, pp. 47–52.

[20] N. Ho, H. Vo, M. Vu, and T. B. Pedersen, "Amic: An adaptive information theoretic method to identify multi-scale temporal correlations in big time series data," *IEEE Transactions on Big Data*, vol. 7, no. 1, pp. 128–146, 2019.

[21] N. Ho, H. Vo, and M. Vu, "An adaptive information-theoretic approach for identifying temporal correlations in big data sets," in *2016 IEEE International Conference on Big Data*. IEEE, 2016, pp. 666–675.

[22] T. T. N. Ho, M. Gribaudo, and B. Pernici, "Characterizing energy per job in cloud applications," *Electronics*, vol. 5, no. 4, p. 90, 2016.

[23] M. Gribaudo, T. T. N. Ho, B. Pernici, and G. Serazzi, "Analysis of the influence of application deployment on energy consumption," in *International Workshop on Energy Efficient Data Centers*. Springer, 2014, pp. 87–101.

[24] N. Ho, T. B. Pedersen, M. Vu, C. A. Biscio *et al.*, "Efficient bottom-up discovery of multi-scale time series correlations using mutual information," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1734–1737.

[25] N. Ho, M. Gribaudo, and B. Pernici, "Improving energy efficiency for transactional workloads in cloud environments," in *Proceedings of the Eighth International Conference on Future Energy Systems*, 2017, pp. 290–295.

[26] N. Ho, T. B. Pedersen, M. Vu *et al.*, "Efficient and distributed temporal pattern mining," in *2021 IEEE International Conference on Big Data (Big Data)*. IEEE, 2021, pp. 335–343.

[27] N. Ho, V. L. Ho, T. B. Pedersen, M. Vu, and C. A. Biscio, "A unified approach for multi-scale synchronous correlation search in big time series–full version," *arXiv preprint arXiv:2204.09131*, 2022.

[28] N. Ho, H. Vo, M. Vu, and T. B. Pedersen, "Amic: An adaptive information theoretic method to identify multi-scale temporal correlations in big time series data – accepted version," *arXiv preprint arXiv:1906.09995*, 2019.

[29] N. T. T. Ho, T. B. Pedersen, L. Van Ho, and M. Vu, "Efficient search for multi-scale time delay correlations in big time series," in *23rd International Conference on Extending Database Technology, EDBT 2020*. OpenProceedings. org, 2020, pp. 37–48.

[30] P. Fournier-Viger, Y. Wang, P. Yang, J. C.-W. Lin, U. Yun, and R. U. Kiran, "Tspin: Mining top-k stable periodic patterns," *Applied Intelligence*, vol. 52, no. 6, pp. 6917–6938, 2022.

[31] R. U. Kiran, Y. Watanobe, B. Chaudhury, K. Zettsu, M. Toyoda, and M. Kitsuregawa, "Discovering maximal periodic-frequent patterns in very large temporal databases," in *2020 7th International Conference on Data Science and Advanced Analytics*. IEEE, 2020, pp. 11–20.

[32] M. F. Javed, W. Nawaz, and K. U. Khan, "Hova-fppm: flexible periodic pattern mining in time series databases using hashed occurrence vectors and apriori approach," *Scientific Programming*, vol. 2021, 2021.

[33] R. U. Kiran, M. Kitsuregawa, and P. K. Reddy, "Efficient discovery of periodic-frequent patterns in very large databases," *Journal of Systems and Software*, vol. 112, pp. 110–121, 2016.

[34] R. U. Kiran, A. Anirudh, C. Saideep, M. Toyoda, P. K. Reddy, and M. Kitsuregawa, "Finding periodic-frequent patterns in temporal databases using periodic summaries," *Data Science and Pattern Recognition*, vol. 3, no. 2, pp. 24–46, 2019.

[35] J. F. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, 1983.

[36] V. L. Ho, N. Ho, and T. B. Pedersen, "Mining seasonal temporal patterns in time series," *arXiv preprint arXiv:2206.14604*, 2022. [Online]. Available: https://arxiv.org/abs/2206.14604

[37] P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos, "Mining frequent arrangements of temporal intervals," *KAIS*, vol. 21, 2009.

[38] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[39] R. M. Corless, G. H. Gonnet, D. E. Hare, D. J. Jeffrey, and D. E. Knuth, "On the lambertw function," *Advances in Computational mathematics*, vol. 5, no. 1, pp. 329–359, 1996.

[40] E.-E. T. Platform. (2019) Entso-e. [Online]. Available: https://transparency.entsoe.eu/dashboard/show

[41] S. Moosavi, M. H. Samavatian, A. Nandi, S. Parthasarathy, and R. Ramnath, "Short and long-term pattern discovery over large-scale geospatiotemporal data," in *ACM SIGKDD*, 2019, pp. 2905–2913.

ISSN (online): 2446-1628
ISBN (online): 978-87-7573-656-0

AALBORG UNIVERSITY PRESS