Mining Seasonal Temporal Patterns in Big Time Series

Van Long Ho*, Nguyen Ho*, Torben Bach Pedersen*
Department of Computer Science, Aalborg University, Denmark
*{vlh, ntth, tbp}@cs.aau.dk

Abstract—Very large time series are increasingly available from an ever wider range of IoT-enabled sensors, from which significant insights can be obtained through mining temporal patterns from them. 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. This paper presents our Frequent Seasonal Temporal Pattern Mining from Time Series (FreqSTPTTS) solution providing: (1) The first solution for seasonal temporal pattern mining (STPM) from time series that can mine STP at different data granularities. (2) The STPM algorithm that uses efficient data structures and two pruning techniques to reduce the search space and speed up the mining process. (3) An approximate version of STPM 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 STPM outperforms the baseline in runtime and memory consumption, and can scale to big datasets. The approximate STPM is up to an order of magnitude faster and less memory consuming than the baseline, while maintaining high accuracy.

I. INTRODUCTION

The widespread of IoT systems enables the collection of big time series from domains such as energy, transportation, climate, and healthcare. Mining such big 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.

Fig. 1: Weather and Influenza time series

For example, in healthcare, health experts might be interested in finding seasonal diseases in a geographical location, as shown in Fig. 1. 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, and March. 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, since temporal patterns are constructed based on temporal events, the complex relations between temporal events create an exponential and large search space of size \( O(n^h 3^h) \) (\( n \) 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 [5], [6] 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 Big Time Series (FreqSTPITS) 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 big 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 efficient data structures, i.e., 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 propose a novel approximate version of STPM to prune unpromising time series and significantly reduce the search space, while maintaining highly accurate results. The approximate STPM can scale on big datasets, i.e., 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.

II. 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 [7], [8], and later extended by [9]–[13], 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 [5], [6], [14]–[35]. 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 [14] 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 [15], [16] improves different aspects of PFP-growth, for example, Amphawan et al. [16] propose period summary to approximate the pattern periodicity to reduce the memory cost, Uday et al. [15] use the concept of item-specific support to address the rare pattern problem. Recently, Javed et al. [50] propose hashed occurrence vectors and Apriori-based approach to speed up periodic itemsets mining.

In a more recent work [5], 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 [5]. In [37], 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 [6], the authors extend PFP-growth++ to find periodic spatial patterns in spatio-temporal databases. In [35], PFP-growth++ is extended to find maximal periodic frequent patterns. In [38], 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 VI, 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.

III. PRELIMINARIES

A. Time Granularity

Definition 3.1 (Time domain) A time domain $T$ consists of an ordered set of time instants that are isomorphic to the natural numbers. The time instants in $T$ have a time unit, presenting how they are measured.

Definition 3.2 (Time granularity) Given a time domain $T$, a time granularity $G$ is a complete and non-overlapping equal partitioning of $T$, i.e., $T$ 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) $G_i$.

As an example, consider a time domain $T$ consisting of an ordered set of 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, $T$ 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 $p$(Minute$_2$) = 2.

Definition 3.3 (Finer time granularity) A time granularity $G$ is finer than a time granularity $H$ if and only if for every granule
Granularity
sequence mapping from symbolic database and temporal sequence database
the position of the granule $G_i$
granularity $G$ is $m$-Finer than granularity $H$
time series $X$ and symbolic time series $X_S$
temporal event $E$ has an event instance $e$
sequence mapping from $X_S$ to granularity $H$
temporal event $E$ and temporal sequence event $E_p$
support set of event $E$ (pattern $P$)
distance between two near support sets
number of seasons of pattern

\textbf{Definition 3.4} (Time granularity hierarchy) Given a time domain $T$, the different time granularities of $T$ form a \textit{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.

\textbf{Definition 3.5} (Period between two granules) 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.

\textbf{Definition 3.6} (Time series) A \textit{time series} $X = x_1, x_2, ..., x_n$ in the time domain $T$ is a sequence of data values that measure the same phenomenon during an observation time period in $T$, and are chronologically ordered. We say that $X$ has granularity $G$ if $X$ is sampled at every time instant $t_i$ in $T$.

\textbf{Definition 3.7} (Symbolic time series) Let $X$ be a time series of granularity $G$. A symbolic time series $X_S$ of $X$ encodes the raw values of $X$ into a sequence of symbols using a mapping function $f$: $X \rightarrow \Sigma_X$ that maps each value $x_i$ in $X$ into a symbol $\omega \in \Sigma_X$. The finite set of permitted symbols used to encode $X$ is called the \textit{symbol alphabet} of $X$, denoted as $\Sigma_X$. Since the mapping function $f$ performs the 1-to-1 mapping from $X$ to $X_S$, $X_S$ has the same granularity $G$ as $X$.

For example, let $X = 1.82, 1.25, 0.46, 0.0$ be a time series representing the energy usage of an electrical device recorded every 5 minutes. By using $\Sigma_X = \{1, 0\}$ (1: ON, 0: OFF), we obtain $X_S = 1, 1, 1, 0$. The mapping function $f$ can be defined using time series representation techniques such as SAX [39].

\textbf{Definition 3.8} (Symbolic database) Given a set of time series $X = \{X_1, ..., X_n\}$, the set of symbolic representations of the time series in $X$ forms a \textit{symbolic database} $\mathcal{D}_{SYB}$.

\textbf{Definition 3.9} (Temporal event) Consider a symbolic time series $X_S$. A \textit{temporal event} $E$ in $X_S$ is a tuple $E = (\omega, T)$ where $\omega \in \Sigma_X$ is a symbol, and $T = \{t_s, t_e\}$ is the set of time intervals during which $X_S$ has the value $\omega$. Each time interval has $t_s$ as the start time, and $t_e$ as the end time.

\textbf{Instance of a temporal event}: The tuple $e = (\omega, [t_s, t_e])$ is an instance of the temporal event $E = (\omega, T)$, representing a single occurrence of $E$ during $[t_s, t_e]$. We use the notation $E_{re}$ to denote that the event $E$ has an instance $e$.

\textbf{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 Allen’s relation model [40] to define 3 basic temporal relations: \textit{Follows}, \textit{Contains}, \textit{Overlaps} between $E_i$ and $E_j$ through $e_i$ and $e_j$. We avoid the exact time mapping problem in Allen’s relations by adding a tolerance buffer $\epsilon$ to the relation’s endpoints, while ensuring the relations are \textit{mutually exclusive} (proof in the technical report [41]). Table III illustrates the three relations and their conditions, with $\epsilon \geq 0$ being the buffer size, and $d_o$ representing the minimal overlapping duration between two event instances in an Overlaps relation.

\textbf{Definition 3.10} (Temporal pattern) Let $\mathcal{R} = \{\text{Follows, Contains, Overlaps}\}$ be the set of temporal relations. A \textit{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 $r_{ij} \in \mathcal{R}$ between two events $E_i$ and $E_j$.

Note that each relation $r_{ij}$ is formed using the specific instances of $E_i$ and $E_j$. A temporal pattern of $n$ 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
Definition 3.11 (Sequence mapping) Consider a symbolic time series $X_S$ of granularity $G$. Let $H$ be a granularity in $\mathcal{H}$ such that $G \leq m H$. 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 I. Using $G \leq 5 H$, a sequence mapping $g : C \rightarrow 3 H$ creates granularity $H$ where the granules are: $H_1 : <C, 1, C, 0, C >$, $H_2 : <C, 1, C, 0, C >$, $H_3 : <C, 1, C, 1, C >$, and so on.

Definition 3.12 (Temporal sequence of a symbolic time series) Consider a symbolic time series $X_S$ of granularity $G$. Let $\omega_1, \ldots, \omega_m$ 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 $Seq_i = (e_1, \ldots, e_n)$ is a list of $n$ event instances, each is obtained by grouping consecutive and identical symbols $\omega$ 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: $Seq_1 = <C, 1, G, G_1, G_2 >$, $(C, 0, G, G_3, G_4)$ at $H_1$, $Seq_2 = <C, 1, G, G_1, G_2 >$, $(C, 0, G, G_3, G_4)$ at $H_2$, $Seq_3 = <C, 1, G, G_1, G_2 >$, $(C, 0, G, G_3, G_4)$ at $H_3$, and so on.

Definition 3.13 (Temporal sequence database) Consider a symbolic database $\mathcal{D}_{SYB}$ of granularity $G$ (defined in Def 3.8) 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 $\mathcal{D}_{SYB}$. The temporal sequences obtained from the mapping $g$ form a temporal sequence database $\mathcal{D}_{SEQ}$ where each row $i$ is a set of sequences $\{Seq_i\}$ of the same granule $H_i \in H$. Furthermore, the temporal sequence database $\mathcal{D}_{SEQ}$ has granularity $H$.

Table IV shows an example of $\mathcal{D}_{SEQ}$, obtained from $\mathcal{D}_{SYB}$ in Table III using the mapping $g : X_S \rightarrow 3 H$ on the five symbolic time series $\{C, D, F, M, N\}$.

Given a symbolic database $\mathcal{D}_{SYB}$ of granularity $G$ and a granularity hierarchy $\mathcal{H}$, we can construct different temporal sequence databases $\mathcal{D}_{SEQ}$ 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 $\mathcal{D}_{SEQ}$ at 15-Minutes granularity. Using $g : X_S \rightarrow 12 H$, we obtain $\mathcal{D}_{SEQ}$ at 1-Hour granularity.

E. Frequent Seasonal Temporal Pattern

Definition 3.14 (Support set of a temporal event) Consider a temporal sequence database $\mathcal{D}_{SEQ}$ of granularity $H$, and a temporal event $E$. The set of granules $H_i$ in $\mathcal{D}_{SEQ}$ where $E$ occurs, arranged in an increasing order, is called the support set of event $E$ and is denoted as $\text{SUP}^E = \{H_1^E, \ldots, H_r^E\}$, where $1 \leq l \leq r \leq |\mathcal{D}_{SEQ}|$. The granule $H_i$ at which event $E$ occurs is denoted as $H_i^E$.

The support set of a group of events $(E_1, \ldots, E_k)$ is the set of granules $H_i$ in $\mathcal{D}_{SEQ}$ where $(E_1, \ldots, E_k)$ concurrently occur, and is denoted as $\text{SUP}^E_{(E_1, \ldots, E_k)}$.

The support set of a temporal pattern: Consider a temporal sequence database $\mathcal{D}_{SEQ}$ of granularity $H$, and a temporal pattern $P$. The set of granules $H_i$ in $\mathcal{D}_{SEQ}$ where $P$ occurs, arranged in an increasing order, is called the support set of pattern $P$ and is denoted as $\text{SUP}^P = \{H_1^P, \ldots, H_r^P\}$, where $1 \leq l \leq r \leq |\mathcal{D}_{SEQ}|$.

Definition 3.15 (Near support set of a temporal pattern) Consider a pattern $P$ with the support set $\text{SUP}^P = \{H_1^P, \ldots, H_r^P\}$. Let maxPeriod be the maximum period threshold, representing the predefined maximal period between any two consecutive granules in $\text{SUP}^P$. The set $\text{SUP}^P$ is called a near support set if

\[
\text{maxPeriod} \leq \frac{1}{2}(H_{i+1}^P - H_{i}^P) \quad \text{for all} \quad i = 1, \ldots, r-1.
\]
of \( P \) if \( \forall (H_o^P, H_p^P) \in \text{SUP}^P : (H_o^P \text{and } H_p^P \text{are consecutive}) \land |p(H_o^P) - p(H_p^P)| \leq \text{maxPeriod} \), 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 \( \text{NearSUP}^P \).

We note that the term near support set is inspired by the mathematical concept called near set \([42]\) which represents a set whose elements are either spatially or descriptively close. In our context, the near support set of \( P \) is a support set where \( P \)'s occurrences are close in time, i.e., any two consecutive occurrences of \( P \) have to be within the maximal period \( \text{maxPeriod} \). Moreover, \( \text{NearSUP}^P \) is a maximal near support set if \( \text{NearSUP}^P \) 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 \( P = \{\text{C:1, D:1}\} \) (or \( \text{C:1} \supseteq \text{D:1} \)) in Table \( \text{IV} \) and let \( \text{maxPeriod} = 2 \). 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: \( \text{NearSUP}^P_1 = \{H_1, H_2, H_3\} \), \( \text{NearSUP}^P_2 = \{H_7, H_8\} \), and \( \text{NearSUP}^P_3 = \{H_{11}, H_{12}, H_{14}\} \). Fig. \( \text{III} \) illustrates the three near support sets of \( P \).

**Definition 3.16 (Season of a temporal pattern)** Let \( \text{NearSUP}^P \) be a near support set of a pattern \( P \). Then \( \text{NearSUP}^P \) is called a season of \( P \) if \( \text{den}(\text{NearSUP}^P) = |\text{NearSUP}^P| \geq \text{minDensity} \), where \( \text{den}(\text{NearSUP}^P) \) counts the number of granules in \( \text{NearSUP}^P \) called the density of \( \text{NearSUP}^P \), and \( \text{minDensity} \) is a predefined minimum density threshold.

The density of \( \text{NearSUP}^P \) measures how dense a near support set of the pattern \( P \) is. The higher the density, the more occurrences of \( P \) appear in \( \text{NearSUP}^P \). For instance, in the previous example, we have \( \text{den}(\text{NearSUP}^P_1) = |\text{NearSUP}^P_1| = 3 \). Similarly, \( \text{den}(\text{NearSUP}^P_2) = 2 \), \( \text{den}(\text{NearSUP}^P_3) = 3 \). 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 \( \text{NearSUP}^P_i = \{H^P_i, ..., H^P_n\} \) and \( \text{NearSUP}^P_j = \{H^P_i, ..., H^P_n\} \) is computed as: \( \text{dist}(\text{NearSUP}^P_i, \text{NearSUP}^P_j) = |p(H^P_i) - p(H^P_j)| \).

Based on the season concept and the distance measure, we define frequent seasonal temporal patterns as follows.

**Definition 3.17 (Frequent seasonal temporal pattern)** Let \( \mathcal{P}S = \{\text{NearSUP}^P\} \) be the set of seasons of a temporal pattern \( P \), and \( \text{minSeason} \) be the minimum seasonal occurrence threshold, \( \text{distInterval} = [\text{dist}_{\text{min}}, \text{dist}_{\text{max}}] \) be the distance interval where \( \text{dist}_{\text{min}} \) is the minimum distance and \( \text{dist}_{\text{max}} \) is the maximum distance. A temporal pattern \( P \) is called a frequent seasonal temporal pattern iff \( \text{seasons}(P) = |\mathcal{P}S| \geq \text{minSeason} \land \forall (\text{NearSUP}^P_i, \text{NearSUP}^P_j) \in \mathcal{P}S: \) they are consecutive and \( \text{dist}_{\text{min}} \leq \text{dist}(\text{NearSUP}^P_i, \text{NearSUP}^P_j) \leq \text{dist}_{\text{max}} \).

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 \( \mathcal{P}S \), and is computed as \( \text{seasons}(P) = |\mathcal{P}S| \).

**Mining Frequent Seasonal Temporal Patterns from Time Series (FreqSTPfTS).** Given a set of \( n \) time series \( \mathcal{X} = \{X_1, ..., X_n\} \) of granularity \( G \), let \( \mathcal{D}_\text{Seq} \) be the temporal sequence database of granularity \( H \in \mathcal{H} \) obtained from \( \mathcal{X} \), and \( \text{maxPeriod, minDensity, distInterval, and minSeason} \) 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 \( \text{maxPeriod, minDensity, distInterval, and minSeason} \) constraints.

**IV. FREQUENT SEASONAL TEMPORAL PATTERN MINING**

**A. Overview of FreqSTPfTS Mining Process**

The FreqSTPfTS mining process consists of two phases. **Phase 1. Data Transformation**, converts a set of time series \( \mathcal{X} \) into a symbolic database \( \mathcal{D}_\text{SYB} \) by using the mapping function defined in Def. 3.7, and then converts \( \mathcal{D}_\text{SYB} \) into a temporal sequence database \( \mathcal{D}_\text{Seq} \) by applying the sequence mapping defined in Def. 3.11. **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 (\( k \geq 2 \)).

Before introducing the STPM algorithm in detail, we first present candidate seasonal pattern, a concept designed to support Apriori-like pruning in STPM.

**B. Candidate Seasonal Pattern**

Pattern mining methods often use the anti-monotonicity property of the support measure to reduce the search space \([43]\). This property ensures that an infrequent event \( E_i \) cannot form a frequent 2-event pattern, since \( \text{support}(E_i) \geq \text{support}(P) \). 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 \( \text{maxPeriod, minDensity, distInterval and minSeason} \) thresholds do not uphold this property, as illustrated below.

Consider an event \( E = \text{M:1} \) and a 2-event pattern \( P = \text{M:1} \supseteq \text{N:1} \) in Table \( \text{IV} \). Let \( \text{maxPeriod} = 2 \), \( \text{minDensity} = 3 \), \( \text{distInterval} = [4, 10] \), and \( \text{minSeason} = 2 \). From the constraints, we can identify the seasons of \( E \) and \( P \) as: \( \mathcal{P}S^E = \{\text{NearSUP}^P\} = \{H_1, H_2, H_3, H_4, H_5, H_6, H_{10}, H_{11}, H_{13}\} \), and \( \mathcal{P}S^P = \{\{\text{NearSUP}^P\} = \{H_1, H_3, H_4, H_5, H_6\}, \{\text{NearSUP}^P_2\} = \{H_{10}, H_{11}, H_{13}\}\}. \) Hence, we have: \( |\mathcal{P}S^E| = 1 \) and \( |\mathcal{P}S^P| = 2 \).

Due to the \( \text{minSeason} \) 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 \( \text{max-} \)imum 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 \( \text{SUP}^P \) of \( P \), and the \( \text{minDensity} \) threshold:

\[
\text{maxSeason}(P) = \frac{\text{|SUP}^P|}{\text{minDensity}}
\]

Eq. (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 \((E_1, ..., E_k)\), are defined in a similar way. Below, we show how \( \text{maxSeason} \) upholds the anti-monotonicity property.

**Lemma 1.** Let \( P \) and \( P' \) be two temporal patterns such that \( P' \subseteq P \). Then \( \text{maxSeason}(P') \geq \text{maxSeason}(P) \).

**Proof.** We have:

\[
\text{maxSeason}(P') = \frac{\text{|SUP}^{P'}|}{\text{minDensity}} \leq \text{maxSeason}(P) = \frac{\text{|SUP}^P|}{\text{minDensity}}
\]

Since: \( |SUP^{P'}| \geq |SUP^P| \) (Derived from Def. 3.14)

Hence: \( \text{maxSeason}(P') \geq \text{maxSeason}(P) \)

**Lemma 2.** Let \( P \) be a \( k \)-event temporal pattern formed by a \( k \)-event group \((E_1, ..., E_k)\). Then, \( \text{maxSeason}(P) \leq \text{maxSeason}(E_1, ..., E_k) \).

**Proof.** Derived directly from Def. 3.14, and Eq. (1).

From Lemmas 1 and 2, the \( \text{maxSeason} \) of a pattern \( P \) is always at most the \( \text{maxSeason} \) of its sub-pattern \( P' \), and of its events \((E_1, ..., E_k)\). Thus, \( \text{maxSeason} \) upholds the anti-monotonicity property, and can be used to reduce the STPM search space. Below, we define the candidate pattern concept that uses \( \text{maxSeason} \) as a gatekeeper to identify frequent/infrequent seasonal patterns.

**Candidate seasonal pattern:** A temporal pattern \( P \) is a candidate seasonal pattern if \( \text{maxSeason}(P) \geq \text{minSeason} \).

Similarly, a group of \( k \) events \( G_E = (E_1, ..., E_k) \) \((k \geq 1)\) is a candidate \( k \)-event group if \( \text{maxSeason}(G_E) \geq \text{minSeason} \). Intuitively, a pattern \( P \) (or \( k \)-event group \( G_E \)) is infrequent if its \( \text{maxSeason} \) is less than \( \text{minSeason} \). 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. Algorithm 1 provides the pseudo-code of STPM.

### C. Mining Seasonal Single Events

The first step in STPM is to mine frequent seasonal single events (Alg. 1, lines 1-9) that satisfy the constraints of \( \text{maxPeriod} \), \( \text{minDensity} \), \( \text{distInterval} \) and \( \text{minSeason} \). To do that, we first look for the candidate single events defined in Section V-B and then use only the found candidates to mine frequent seasonal events.

The candidate single events are found by first scanning \( D_{SEQ} \) to identify the support set \( \text{SUP}^{E_i} \) for each event \( E_i \), from which we compute the maximum seasonal occurrence \( \text{maxSeason}(E_i) \). If \( \text{maxSeason}(E_i) \geq \text{minSeason} \), then \( E_i \) is a candidate seasonal single event. Otherwise, \( E_i \) is not a candidate and is removed from the search space. Note that we only need to scan \( D_{SEQ} \) once to find all candidate events.

To mine frequent seasonal events, for each candidate event \( E_i \), we iterate through the support set \( \text{SUP}^{E_i} \), and calculate the period \( pr_{ij} \) between every two consecutive granules in \( \text{SUP}^{E_i} \), and determine the near support sets \( \text{NearSUP}^{E_i} \) that satisfy \( \text{maxPeriod} \) and \( \text{minDensity} \). Next, the set of seasons \( \text{PS}^{E_i} \) is identified by selecting the near support sets that adhere to the \( \text{distInterval} \) constraint. Finally, the frequent seasonal events are determined by comparing the number of seasons of \( E_i \) to \( \text{minSeason} \), selecting only those that have \( \text{seasons}(E_i) = |\text{PS}^{E_i}| \geq \text{minSeason} \).

We use a hierarchical lookup hash structure \( \text{HLH}_1 \) 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 \( \text{HLH}_1 \) instead of the frequent seasonal events, as the \( \text{maxSeason} \) of candidate events upholds the anti-monotonicity property, and can thus be used for pruning. We illustrate \( \text{HLH}_1 \) in Fig. 4 and describe the data structure below.

**Hierarchical lookup hash structure** \( \text{HLH}_1 \): The \( \text{HLH}_1 \) is a hierarchical data structure that consists of two hash tables: the **single event hash table** \( \text{EH} \) maintaining the candidate events, and the **event granule hash table** \( \text{GH} \) maintaining the instances of found candidates. Each hash table has a list of \(<\text{key}, \text{value}>\) pairs. In \( \text{EH} \), the key is the event symbol \( \omega \in \Sigma_X \) representing the candidate \( E_i \), and the value is the list of granules \( <H_1, ..., H_k> \) in \( \text{SUP}^{E_i} \). In \( \text{GH} \), the key is the list of granules shared in the value field of \( \text{EH} \), while the value stores event instances of \( E_i \) that appear at the corresponding
D. Mining Seasonal k-event Patterns

Search space of STPM. The next step of STPM is to 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 naive approach is very expensive as it creates a very large search space, approximately of size \(O(n^h \cdot 3^h)\), where \(n\) is the number of distinct events in \(D_{SEQ}\), and \(h\) is the maximal length of a temporal pattern, making it computationally prohibitive to mine seasonal patterns. Detailed derivation and the proof of the search space complexity are provided in [41]. The problem of a large search space is thus alleviated by using an iterative mining process that first finds seasonal candidate k-event groups, and then mines frequent seasonal k-event patterns only from the candidates. Below, we first introduce the data structure used in this mining step.

The hierarchical lookup hash structure \(HLH_k\): We use the hierarchical lookup hash structure \(HLH_k\) to maintain candidate seasonal k-event groups and patterns, as illustrated in Fig. 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 \(P = \{(r_{(1)}(E_1, E_2), ..., (r_{(k-1)}(k), E_{k-1}, E_k)\})\), created from the k-event group \((E_1, ..., E_k)\). In \(PH_k\), 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. STPM uses the anti-monotonicity property of \(maxSeason\) to find candidate seasonal k-event groups (Alg. \[1\] lines 10-12).

Let \(F_{k-1}\) be the set of candidate seasonal (k-1)-event groups found in \(HLH_{k-1}\), and \(F_1\) be the set of candidate seasonal single events in \(HLH_1\). 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 \(SUP^{(E_1, ..., E_k)}\) 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. \[1\] 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 \(S\) that contain \((E_i, E_j)\). Next, for each sequence \(S \in S\), we extract their event instances \((e_i, e_j)\), and verify the relation between them. We then compute the \(maxSeason\) of the 2-event pattern \(P\) and determine if \(P\) is a candidate pattern, i.e., \(maxSeason\) \(\geq \).
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 $SUP$ 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 $N_2 = N_{k-1} \cup N_1 = (E_1, ..., E_k)$ 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 $P_{k-1}$ and then identifying the $EH_{k-1} \cup P_{k-1}$ table. Each $P_{k-1} \in P_{k-1}$ is a list of $\frac{1}{2}(k-1)(k-2)$ triples: $\{(r_{ij2}, E_1, E_2), ..., (r_{ij(k-2)(k-1)}, E_{k-2}, E_{k-1})\}$. We iteratively verify the possibility of $P_{k-1}$ forming a k-event pattern $P_k$ with $E_k$ as follows.

We first start with the triple $(r_{ij(k-1)k}, E_{k-1}, E_k)$. If $(r_{ij(k-1)k}, E_{k-1}, E_k)$ 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 $(r_{ij(k-2)k}, E_{k-2}, E_k)$, until it reaches $(r_{1kk}, E_1, E_k)$. 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. 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, (C:1, D:1, F:0) cannot form any candidate seasonal 3-event patterns, since F:0 is not present in any candidate 2-event patterns in $HLH_2$. To reduce such redundancy and further optimize the mining, we use the transitivity property of temporal relations to identify such event groups.

Lemma 3. Let $S = \langle e_1, ..., e_{k-1} \rangle$ be a temporal sequence, $P = \langle (r_{12}, E_{k_{12}}, E_{2k_{12}}), ..., (r_{(k-2)(k-1)}, E_{k-2k_{(k-2)(k-1)}}, E_{k-1k_{(k-2)(k-1)}}) \rangle$ be a (k-1)-event pattern that occurs in $S$, $e_k$ be a new event instance added to $S$ to create the temporal sequence $S' = \langle e_1, ..., e_k \rangle$. 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. $r(E_{k_{i-1}}, E_{k_{i+1}})$ hold.

Lemma 3 states the temporal transitivity property between temporal events, and is used to prove the following lemma.

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. The group $N_k = N_{k-1} \cup E_k$ can form candidate seasonal 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 candidate seasonal temporal relation.

From Lemma 4, only single events in $HLH_1$ 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 single events in $HLH_1$, $i$ be the average number of instances of each event, $r$ be the number of (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-pattern mining is $O(n^2i^2u^2) + O(|F_1| \cdot |F_{k-1}| \cdot r \cdot k^2 \cdot u)$. While the parameters $n$, $i$, $r$ and $k$ depend on the number of time series, others such as $|F_1|$, $|F_{k-1}|$ and $u$ depend on the number of temporal sequences. Thus, STPM complexity is driven by two main factors: the number of time series and the number of temporal sequences.

V. APPROXIMATE STPM

A. Correlated Symbolic Time Series

Let $X_S$ and $Y_S$ be the symbolic series representing the time series $X$ and $Y$, and $\Sigma_X$, $\Sigma_Y$ be their symbolic alphabets.

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

$$H(X_S) = - \sum_{x \in \Sigma_X} p(x) \log p(x)$$

where $p(x)$ is the probability of $X_S$. Intuitively, the entropy measures the uncertainty of the possible outcomes 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

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

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

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

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 $Y_S$. Since $0 \leq I(X_S; Y_S) \leq \min\{H(X_S), H(Y_S)\}$, the MI value has no upper bound. To scale it into the range $[0,1]$, we normalize the MI 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 $\tilde{I}(X_S; Y_S)$, is defined as

$$\tilde{I}(X_S; Y_S) = \frac{I(X_S; Y_S)}{\min\{H(X_S), H(Y_S)\}}$$

$\tilde{I}(X_S; Y_S)$ represents the reduction (in percentage) of the uncertainty of $X_S$ due to knowing $Y_S$. Based on Eq. (5), a pair of variables $(X_S, Y_S)$ has a mutual dependency if...
\( \tilde{I}(X_S; Y_S) > 0 \). Moreover, Eq. (5) also shows that NMI is not symmetric, i.e., \( \tilde{I}(X_S; Y_S) \neq \tilde{I}(Y_S; X_S) \).

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

**B. Lower Bound of the maxSeason**

Consider two symbolic series \( X_S \) and \( Y_S \). Let \( X_1 \) be a temporal event in \( X_S \), \( Y_1 \) be a temporal event in \( Y_S \). \( D_{SYB} \) and \( D_{SEQ} \) be the symbolic and the sequence databases created from \( X_S \) and \( Y_S \), respectively. We have the following relation between \( \tilde{I}(X_S; Y_S) \) in \( D_{SYB} \), and \( maxSeason(X_1, Y_1) \) in \( D_{SEQ} \).

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

\[
maxSeason(X_1, Y_1) \geq \frac{\lambda_2 \cdot |D_{SEQ}|}{\minDensity} \cdot e^{\left( \frac{-\log (\lambda_1 \cdot p_{X_1 \cup Y_1})}{\ln 2} \right)} \tag{6}
\]

where: \( \lambda_1 = \min\{p(X_1), \forall X_1 \in X_S\} \) is the minimum probability of \( X_1 \in X_S \), and \( \lambda_2 = p(Y_1) \) is the probability of \( Y_1 \in Y_S \), and \( W \) is the Lambert function \([4,5]\).

**Proof.** (Sketch - Full proof in [11]). From Eq. (5), we have:

\[
\tilde{I}(X_S; Y_S) = 1 - \frac{H(X_S|Y_S)}{H(X_S)} \geq \mu \tag{7}
\]

\[
\Rightarrow \frac{H(X_S|Y_S)}{H(X_S)} = \frac{p(X_1, Y_1) \cdot \log p(X_1|Y_1)}{\sum_i p(X_i) \cdot \log p(X_i)} + \sum_{i \neq j \neq 1} p(X_i, Y_j) \cdot \log p(X_i, Y_j) \cdot \frac{p(X_i, Y_j)}{p(X_i)} \leq 1 - \mu \tag{8}
\]

Let: \( \lambda_1 = \min\{p(X_i), \forall i\} \), \( \lambda_2 = p(Y_1) \).

\[
\frac{H(X_S|Y_S)}{H(X_S)} \geq \frac{p(X_1, Y_1) \cdot \log \frac{p(X_1, Y_1)}{\lambda_2}}{\log \lambda_1} \tag{9}
\]

From Eqs. (8), (9), we derive:

\[
p(X_1, Y_1) \geq \lambda_2 \cdot e^{\left( \frac{-\log (\lambda_1 \cdot p_{X_1 \cup Y_1})}{\ln 2} \right)} \tag{10}
\]

Since:

\[
\frac{|SUP^p(X_1, Y_1)|}{|D_{SEQ}|} \geq p(X_1, Y_1) \tag{11}
\]

It follows that:

\[
\frac{|SUP^p(X_1, Y_1)|}{\minDensity} \geq \lambda_2 \cdot \frac{|D_{SEQ}|}{\minDensity} \cdot e^{\left( \frac{-\log (\lambda_1 \cdot p_{X_1 \cup Y_1})}{\ln 2} \right)} \tag{12}
\]

From Eq. (12), the lower bound of \( maxSeason(X_1, Y_1) \) is:

\[
maxSeason(X_1, Y_1) \geq \frac{\lambda_2 \cdot |D_{SEQ}|}{\minDensity} \cdot e^{\left( \frac{-\log (\lambda_1 \cdot p_{X_1 \cup Y_1})}{\ln 2} \right)} \tag{13}
\]

**Setting the parameters:** To compute the lower bound of \( maxSeason(X_1, Y_1) \) in Eq. (6), several parameters need to be defined: \( \lambda_1, \lambda_2, \) and \( \mu \). Given \( D_{SYB} \), \( \lambda_1 \) and \( \lambda_2 \) can easily be determined since \( \lambda_1 \) is the minimum probability among all events \( X_1 \in X_S \), and \( \lambda_2 \) is the probability of \( Y_1 \in Y_S \). To set the value of \( \mu \), we use the lower bound of \( maxSeason \) in Theorem 1 to derive \( \mu \) as follows.

---

**Algorithm 2:** Approximate STPM using MI

**Input:** A set of time series \( X \), the thresholds: \( maxPeriod, minDensity, distInterval, minSeason \)

**Output:** The set of frequent seasonal temporal patterns \( P \)

// Find the correlated symbolic series

1. **foreach** pair of symbolic time series \((X_S, Y_S) \in D_{SYB}\) **do**
2. \( \minNMI \leftarrow \min\{\tilde{I}(X_S; Y_S), \tilde{I}(Y_S; X_S)\} \);  
3. Compute \( \mu \) using Eq. (14).
4. **if** \( \minNMI \geq \mu \) **then**
5. **Insert** \( X_S \) and \( Y_S \) into \( X_C \);
6. **Mine** frequent seasonal single events from \( X_C \);
7. **foreach** \((X_S, Y_S) \in X_C\) **do**
8. **Mine** frequent seasonal 2-event patterns from \((X_S, Y_S)\);  
9. **if** \( k \geq 3 \) **then**
10. **Perform** STPM using \( HLH_1 \) and \( HLH_{k-1} \);

**Corollary 1.1.** The maximum seasonal occurrence of an event pair \((X_1, Y_1) \in (X_S, Y_S) \) in \( D_{SEQ} \) is at least \( minSeason \) if \( \tilde{I}(X_S; Y_S) \) is at least \( \mu \), where:

\[
\mu \geq \begin{cases} 
1 - \frac{\lambda_2}{e^{\ln \lambda_1 \cdot p_{X_1 \cup Y_1}}} & \text{if } 0 \leq \rho \leq \frac{1}{1 - \frac{1}{\lambda_2 \cdot |D_{SEQ}| / \minDensity}}, \\
\rho = \frac{minSeason \cdot minDensity}{\lambda_2 \cdot |D_{SEQ}|} & \text{otherwise} 
\end{cases} \tag{14}
\]

Note that \( \mu \) in Eq. (14) only ensures that the \( maxSeason \) of the pair \((X_1, Y_1)\) is at least \( minSeason \). Thus, given \((X_S, Y_S)\), \( \mu \) has to be computed for each event pair in \((X_S, Y_S)\). The final chosen \( \mu \) value to be compared against \( \tilde{I}(X_S; Y_S) \) is the minimum \( \mu \) value among all the event pairs in \((X_S, Y_S)\).

**Interpretation of the lower bound of the maximum seasonal occurrence:** Theorem 1 says that, given an MI threshold \( \mu \), if the two symbolic series \( X_S \) and \( Y_S \) are correlated, i.e., \( \tilde{I}(X_S; Y_S) \geq \mu \), then the maximum seasonal occurrence of an event pair in \((X_S, Y_S)\) is at least the lower bound in Eq. (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. (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. 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 \( \mu \) (Eq. 14) 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 \( X_C \subseteq X \). Algorithm 2 outlines the approximate STPM.

First, NMI and \( \mu \) are computed for each pair of symbolic series \((X_S, Y_S) \in D_{SYB}\) (lines 2-3). Then, only pairs whose \( \min\{\tilde{I}(X_S; Y_S), \tilde{I}(Y_S; X_S)\} \) is at least \( \mu \) are inserted into \( X_C \). Next, only the correlated symbolic series in \( X_C \) are used to mine frequent seasonal single events (line 6). For frequent seasonal 2-event patterns, we mine frequent seasonal patterns only from event pairs in \( X_C \) (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:** To compute NMI and \( \mu \), we only need to scan \( D_{SYB} \) once to calculate the probability for each single event and pair of events. Thus,
the cost of NMI and $\mu$ computations is $|D_{SYB}|$. Moreover, the complexities of the exact STPM for $HLH_1$ and $HLH_2$ are $O(n \cdot |D_{SEQ}|)$ and $O(n^2 \cdot \mu^2)$ (Sections IV-C and IV-D). Thus, the approximate STPM is significantly faster than STPM.

VI. EXPERIMENTAL EVALUATION

We evaluate STPM (exact and approximate) both qualitative and quantitative. Due to space limitations, we only present here the most important results, and discuss other findings in [41].

A. 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 [45] and weather data [47] from Spain. For smart city (SC), we use traffic and weather datasets [48] from New York City. For health, we combine the influenza (INF) and hand-foot-mouth (HFM) datasets [49] and weather data [47] 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 V summarizes the dataset characteristics.

Baseline method: Our exact method is referred to as E-STPM, and the approximate one as 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 [38] to find seasonal temporal patterns. Specifically, the adaptation is done through a 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 virtual machine with 32 AMD EPYC cores (2GHz), 512 GB RAM, and 1 TB storage.

Parameters: Table VI lists the parameters and their values used in our experiments, where maxPeriod and minDensity are expressed as the percentage of $D_{SEQ}$. While the four parameters in Table VI 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. minSeason and distInterval values often depend on the granularity of $D_{SEQ}$. For example, if $D_{SEQ}$ 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 seasonally occurred.

B. Qualitative Evaluation

Table VIII lists some interesting seasonal patterns found in 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. For example, P1 shows that wind energy generation is high since there is more wind from December to February, while P2 indicates that the electricity demand is also high during this period since the short days and low temperatures mean more lighting and heat are required. Such patterns indicate that wind power is a good energy source during the winter. Patterns P4-P7 are extracted from INF and HFM, showing the detection of seasonal diseases. For example, P4 shows that the number of influenza cases is very high in January and February when the temperature is very low, whereas the hand-foot-mouth disease cases increase during May and June when the temperature is high (P6). Knowing such patterns can support better disease preparation and prevention. Finally, how weather affects traffic is presented in patterns P8-P11 extracted from SC, showing that bad weather conditions cause congestion, lane-blockings, and flow-incidents, which often occur in July and August.

Tables IX and X 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. Quantitative Evaluation

1) Baseline comparison on real-world datasets: We compare E-STPM and A-STPM with the baseline in terms of the runtime and memory usage. Figs. 7 [8] [9] and [10] show the comparison on RE and INF datasets. The results on other datasets are reported in the technical report [41]. Note that Figs. 7 [14] use the same legend.

As shown in Figs. 7 and 8 A-STPM achieves the best runtime among all methods, and E-STPM has better runtime than the baseline. On the tested datasets, the range and average speedups of A-STPM compared to 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 compared to the baseline is [3.5-7.2] and 4.3 on average. Note that the times to compute MI and $\mu$ for RE and INF in Figs. 7 and 8 are only 2.6 and 1.4 seconds, respectively. Moreover, A-STPM is most efficient, i.e., achieves 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, using A-STPM to prune uncorrelated time series early helps save computational time and resources. However, the speedup comes at the cost of a small loss in accuracy (discussed in Section VI-C4).

In terms of memory consumption, as shown in Figs. 9 and 10 A-STPM is the most efficient method, while E-STPM is
TABLE VI: Parameters and values

| Params       | Values (User-defined) |
|--------------|-----------------------|
| maxPeriod    | 0.2, 0.4, 0.6, 0.8, 1.0 |
| minDensity   | 0.5%, 0.75%, 1%, 1.25%, 1.5% |
| minSeason    | 4, 8, 12, 16, 20 |

TABLE VII: A-STPM Accuracy

| # minSeason | re | syn |
|-------------|----|-----|
| 12          | 0.81 0.88 0.98 0.97 1.00 | 0.81 0.88 0.98 0.97 1.00 |
| 16          | 0.84 0.92 0.90 0.90 1.00 | 0.84 0.92 0.90 0.90 1.00 |
| 20          | 0.90 0.99 1.00 1.00 1.00 | 0.90 0.99 1.00 1.00 1.00 |

TABLE VIII: Summary of Interesting Seasonal Patterns

| Pattern                                                                 | 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 | 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) Very High Temperature > 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) High Temperature > Strong wind => High Congestion                  | 0.5            | 0.6           | 8           | July, August       |
| (P8) Strong wind > Unclear Visibility > High Congestion                 | 0.5            | 0.6           | 8           | July, August       |
| (P9) Heavy Rain > Unclear Visibility > High Lane-Blocked                | 0.4            | 0.8           | 8           | July, August       |
| (P10) Heavy Rain > Strong wind > High Flow-Indicat                       | 0.4            | 0.8           | 8           | July, August       |

Fig. 10: Memory Usage Comparison on INF (real-world)

Fig. 12: Scalability: Varying #Sequences on INF (synthetic)

Fig. 13: Scalability: Varying #TimeSeries on RE (synthetic)

Fig. 14: Scalability: Varying #TimeSeries on INF (synthetic)
more efficient than the baseline. The range and the average memory consumption of A-STPM 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.

2) Scalability evaluation on synthetic datasets: As discussed in Section VI the complexity of STPM is driven by two main factors: (1) the number of temporal sequences, (2) the number of time series. Thus, to further evaluate STPM scalability, we scale these two factors on synthetic datasets (reported in Table V), using two configurations: varying the number of sequences, and varying the number of time series.

Figs. 11 and 12 show the runtimes of A-STPM, E-STPM and the baseline when the number of sequences changes (y-axis is in log scale). 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. 11a) and on the synthetic INF at 100% sequences ($\approx 6 \times 10^5$) (Fig. 12a), showing that A-STPM and E-STPM can scale well on big datasets while the baseline cannot.

Figs. 13 and 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. 13a), and $\geq 8000$ on the synthetic INF (Fig. 14a).

Furthermore, we provide the computation time of MI and $\mu$ in Figs. 13 and 14 by adding an additional bar chart for A-STPM. Each bar represents the runtime of A-STPM with two separate components: the time to compute MI and $\mu$ (top red), and the mining time (bottom blue). We note that for each dataset, we only need to compute MI once (the computed MIs are used across different minSeason and minDensity thresholds), while the computation of $\mu$ is negligible (in milliseconds using Eq. (4)). Thus, the MI and $\mu$ computation times, for example, in Figs. 13a, 13b and 13c are added only for comparison and are not all actually used.

Finally, we provide the percentage of time series and events pruned by A-STPM in the scalability test in Table XI. Here, we can see that low minSeason and minDensity lead to more time series (events) to be pruned. This is because minSeason and minDensity have an inverse relationship with $\mu$, therefore, low minSeason and minDensity result in higher $\mu$, and thus, more pruned time series.

3) Evaluation of the pruning techniques in E-STPM: To understand how effective the proposed pruning techniques are, we compare different versions of E-STPM: (1) NoPrune: E-STPM with no pruning, (2) Apriori: E-STPM with Apriori-like pruning (Lemmas 1, 2), (3) Trans: E-STPM with transitivity-based pruning (Lemmas 3, 4), and (4) All: E-STPM applied both pruning techniques.

Figs. 15 and 16 show the results (the y-axis is in log scale). It can be seen that (All)-E-STPM achieves the best performance among all versions. Its speedup w.r.t. (NoPrune)-E-STPM ranges from 3 up to 6 depending on the configurations, showing that the proposed prunings are very effective in improving E-STPM performance. Furthermore, (Trans)-E-STPM delivers larger speedup than (Apriori)-E-STPM. The average speedup is from 2 to 5 for (Trans)-E-STPM, and from 1.5 to 4 for (Apriori)-E-STPM. However, applying both always yields better speedup than applying either of them.

4) Evaluation of A-STPM: We proceed to evaluate the accuracy of A-STPM by comparing the patterns extracted by A-STPM and E-STPM. Table VII 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 XII shows the accuracies of A-STPM on the synthetic datasets: very high accuracy ($\geq 96\%$) when minSeason and minDensity are high, e.g., minSeason = 16 and minDensity = 0.75%.

VI. 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 efficient and exact Seasonal Temporal Pattern Mining (E-STPM) algorithm that employs the efficient data structures and pruning techniques to achieve fast mining, and (3) the approximate A-STPM that uses mutual information to prune unpromising time series and allows STPM to scale on big datasets. Extensive experiments conducted on real-world and synthetic datasets show that both A-STPM and E-STPM outperform the baseline, consume less memory, and scale well to big datasets. Compared to the baseline, the approximate A-STPM delivers up to an order of magnitude speedup. In future work, we plan to extend STPM to prune at the event level to further improve its performance.
[43] P. Papapetrou, G. Kollios, S. Sclaroff, and D. Gunopulos, “Mining frequent arrangements of temporal intervals,” *KAIS*, vol. 21, 2009.
[44] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.
[45] 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.
[46] E.-E. T. Platform. (2019) Entso-e. [Online]. Available: https://transparency.entsoe.eu/dashboard/show
[47] O. Weather. (2021) Open weather. [Online]. Available: https://openweathermap.org/
[48] S. Moosavi, M. H. Samavatian, A. Nandi, S. Parthasarathy, and R. Ramnath, “Short and long-term pattern discovery over large-scale geo-spatiotemporal data,” in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019, pp. 2905–2913.
[49] K. city infectious disease surveillance system. (2021) Kidss. [Online]. Available: https://kidss.city.kawasaki.jp/


A. Mutual exclusive property of temporal relations

Property 1. (Mutual exclusive) Consider the set of temporal relations \( \mathcal{R} = \{ \text{Follows, Contains, Overlaps} \} \). Let \( E_i \) and \( E_j \) be two temporal events, and \( e_i \) occurring during \([t_{s_i}, t_{e_i}] \), \( e_j \) occurring during \([t_{s_j}, t_{e_j}] \), be their corresponding event instances, and \( \epsilon \) be the tolerance buffer. The relations in \( \mathcal{R} \) are mutually exclusive on \( E_i \) and \( E_j \).

Proof. * Case 1: Assume the relation \( \text{Follows}(E_{b_{e_i}}, E_{j_{e_j}}) \) holds between \( E_i \) and \( E_j \). Thus, we have:

\[
    t_{e_i} \pm \epsilon \leq t_{s_j}
\]

(15)

and:

\[
    t_{s_j} < t_{e_j} \Rightarrow t_{e_i} \pm \epsilon < t_{e_j}
\]

(16)

Hence, \( \text{Contains}(E_{b_{e_i}}, E_{j_{e_j}}) \) cannot exist between \( E_i \) and \( E_j \), since \( \text{Contains}(E_{b_{e_i}}, E_{j_{e_j}}) \) holds iff \((t_{s_j} \leq t_{s_j}) \land (t_{e_i} \pm \epsilon \geq t_{e_j}) \) (contradict Eq. (16)). Similarly, \( \text{Overlaps}(E_{b_{e_i}}, E_{j_{e_j}}) \) cannot exist between \( E_i \) and \( E_j \), since \( \text{Overlaps}(E_{b_{e_i}}, E_{j_{e_j}}) \) holds iff \((t_{s_j} < t_{s_j}) \land (t_{e_i} \pm \epsilon < t_{e_j}) \land (t_{e_i} - t_{s_j} \geq d_0 \pm \epsilon) \) (contradict Eq. (15)).

In conclusion, if \( \text{Follows}(E_{b_{e_i}}, E_{j_{e_j}}) \) holds between \( E_i \) and \( E_j \), the then two remaining relations cannot exist between \( E_i \) and \( E_j \).

* Case 2: Assume the relation \( \text{Contains}(E_{b_{e_i}}, E_{j_{e_j}}) \) holds between \( E_i \) and \( E_j \). Thus, we have:

\[
    t_{s_i} \leq t_{s_j}
\]

(17)

\[
    t_{e_i} \pm \epsilon \geq t_{e_j}
\]

(18)

Hence, \( \text{Follows}(E_{b_{e_i}}, E_{j_{e_j}}) \) cannot exist between \( E_i \) and \( E_j \), since \( \text{Follows}(E_{b_{e_i}}, E_{j_{e_j}}) \) holds iff \( t_{e_i} \pm \epsilon < t_{e_j} \) (contradict Eq. (18)).

Similarly, \( \text{Overlaps}(E_{b_{e_i}}, E_{j_{e_j}}) \) cannot exist between \( E_i \) and \( E_j \), since \( \text{Overlaps}(E_{b_{e_i}}, E_{j_{e_j}}) \) holds iff \((t_{s_j} < t_{s_j}) \land (t_{e_i} \pm \epsilon \leq t_{e_j}) \land (t_{e_i} - t_{s_j} \geq d_0 \pm \epsilon) \) (contradict Eq. (18)).

In conclusion, if \( \text{Contains}(E_{b_{e_i}}, E_{j_{e_j}}) \) holds between \( E_i \) and \( E_j \), the then two remaining relations cannot exist between \( E_i \) and \( E_j \).

* Case 3: Assume the relation \( \text{Overlaps}(E_{b_{e_i}}, E_{j_{e_j}}) \) holds between \( E_i \) and \( E_j \). Thus, we have:

\[
    t_{s_j} < t_{s_j}
\]

(19)

\[
    t_{e_i} \pm \epsilon < t_{e_j}
\]

(20)

\[
    t_{e_i} - t_{s_j} \geq d_0 \pm \epsilon \Rightarrow t_{s_j} \leq t_{e_i} - d_0 \pm \epsilon
\]

(21)

Hence, \( \text{Follows}(E_{b_{e_i}}, E_{j_{e_j}}) \) cannot exist between \( E_i \) and \( E_j \), since \( \text{Follows}(E_{b_{e_i}}, E_{j_{e_j}}) \) holds iff \( t_{e_i} \pm \epsilon < t_{s_j} \) (contradict Eq. (21)).

Similarly, \( \text{Contains}(E_{b_{e_i}}, E_{j_{e_j}}) \) cannot exist between \( E_i \) and \( E_j \), since \( \text{Contains}(E_{b_{e_i}}, E_{j_{e_j}}) \) holds iff \( t_{e_i} \pm \epsilon \geq t_{e_j} \) (contradict Eq. (20)).

In conclusion, if \( \text{Overlaps}(E_{b_{e_i}}, E_{j_{e_j}}) \) holds between \( E_i \) and \( E_j \), then the two remaining relations cannot exist between \( E_i \) and \( E_j \).

B. Lemma 1

Lemma 1. Let \( P \) and \( P' \) be two temporal patterns such that \( P' \subseteq P \). Then \( \maxSeason(P') \geq \maxSeason(P) \).

Proof. We have:

\[
    \maxSeason(P') = \frac{|SUP_{P'}|}{\minDensity}
\]

Since: \( |SUpp_{P'}| \geq |SUpp_{P'}| \) (Derived from Def. 3.14)

Hence: \( \maxSeason(P') \geq \maxSeason(P) \)

\[
\]

C. Lemma 2

Lemma 2. Let \( P \) be a k-event temporal pattern formed by a k-event group \( (E_1, ..., E_k) \). Then, \( \maxSeason(P) \leq \maxSeason(E_1, ..., E_k) \).

Proof. Derived directly from Def. 3.14, and Eq. (1).

D. Mining frequent seasonal single event

Complexity: The complexity of finding frequent seasonal single events is \( O(n \cdot |D_{SEQ}|) \), where \( n \) is the number of distinct events.

Proof. Computing \( \maxSeason \) for each event \( E_i \) takes \( O(|D_{SEQ}|) \). Thus, computing \( \maxSeason \) for \( n \) events takes \( O(n \cdot |D_{SEQ}|) \). Moreover, for each candidate event \( E_i \), identifying the set of seasons \( PS \) takes \( O(|SUP_{E_i}|) \). We have potentially \( n \) events. And thus, it takes \( O(n \cdot |SUpp_{E_i}|) \). The overall complexity is: \( O(n \cdot |D_{SEQ}| + n \cdot |SUpp_{E_i}|) \sim O(n \cdot |D_{SEQ}|) \).

E. Search space of STPM

Complexity: The search space of finding seasonal temporal patterns is \( O(n^h3^h) \), where \( n \) is the number of distinct events in \( D_{SEQ} \), and \( h \) is the maximal length of a temporal pattern.

Proof. The number of seasonal single events is: \( N_1 = n \sim O(n) \). For mining 2-event groups, the number of permutations of \( n \) distinct events taken 2 at a time is: \( P(n, 2) \). However, since the same event can form a pair of events with itself, the total number of 2-event groups is: \( N_2 = P(n, 2) + n \sim O(n^2) \). Each 2-event group in \( N_2 \) can form 3 different temporal relations, and thus, the total number of seasonal 2-event patterns is: \( N_2 \times 3^1 \sim O(n^23^1) \). Similarly, the number of 3-event groups is: \( N_3 = P(n, 3) + P(n, 2) + n \sim O(n^3) \), and the number of seasonal 3-event patterns is: \( N_3 \times 3^2 \sim O(n^33^2) \). For mining h-event groups, the number of h-event groups is \( O(n^h) \), while the number of seasonal h-event patterns is \( O(n^h3^h) \). Therefore, the total number of seasonal temporal patterns is \( O(n) + O(n^23^1) + O(n^33^2) + ... + O(n^h3^h) \sim O(n^h3^h) \).

F. Lemma 3

Lemma 3. Let \( S = \langle e_1, ..., e_{k-1} > \) be a temporal sequence, \( P = \langle (r_1, E_{b_{e_1}}, E_{b_{e_2}}), ..., (r_{(k-2)-(k-1)}, E_{b_{e_{k-2}}}, E_{b_{e_{k-1}}}) > \) be a (k-1)-event pattern that occurs in \( S \), \( e_k \) be a new event instance added to \( S \) to create the temporal sequence \( S' = \langle e_1, ..., e_k > \). The set of temporal relations \( \mathcal{R} \) is transitive on \( S' \): \( \forall e_i \in S', i < k, \exists r \in \mathcal{R} \text{ s.t. } r(E_{b_{e_i}}, E_{b_{e_k}}) \) hold.
For each candidate pattern, identifying its average number of granules of each event/temporal relation. Thus, finding seasonal (k-1)-event group, and computing the support set of frequent seasonal 2-event pattern mining is:}

\[ \text{ps} \implies \text{minDensity} \]

**Proof.** Let \( p_k \) be any k-event pattern formed by \( N_k \). Then \( p_k \) is a list of \( \frac{1}{2}(k-1) \) triples \((E_i, r_{ij}, E_j)\) where each represents a relation \( r((E_i, E_j)) \) between two events. In order for \( p_k \) to be a candidate seasonal k-event temporal pattern, each of the relations in \( p_k \) must be a candidate seasonal temporal pattern (Def. 3.10, Eq. (1), and Lemmas [1] and [3]).

**H. Mining frequent seasonal k-event pattern mining**

**Complexity:** Let \( n \) be the number of single events in \( H LH_1 \), \( i \) be the average number of event instances of each event, \( r \) be the number of (k-1)-event patterns in \( H LH_{k-1} \), 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.** The complexity of frequent seasonal 2-event pattern mining: The Cartesian product of \( n \) events in \( H LH_1 \) generates \( n^2 \) 2-event groups. Computing maxSeason of \( n^2 \) 2-event groups takes \( O(n^2 u) \). For each 2-event group, we need to compute maxSeason of their temporal relations, which takes \( O(i^2 u^2) \). We have potentially \( n^2 \) nodes. And thus, it takes \( O(n^2 i^2 u^2) \). For each candidate 2-event pattern, identifying the set of season \( PS \) takes \( O(u) \). And we have potentially \( (n^2 s) \) relations (s is the number of temporal relations). Thus, finding \( PS \) takes \( O(n^2 su) \). The complexity of frequent seasonal 2-event pattern mining is: \( O(n^2 u + n^2 i^2 u^2 + n^2 su) \) \sim O(n^2 i^2 u^2).

**The complexity of frequent seasonal k-event pattern mining** \((k > 2)\): For each (k-1)-event pattern, we need to compute the support set of \( \frac{1}{2}(k-1)(k-2) \) triples, which takes \( O\left(\frac{1}{2}(k-1)(k-2)u\right) \sim O(k^2 u) \). We have \( |F_1| \times |F_{k-1}| \cdot r \cdot k^2 \cdot u \). For each candidate pattern, identifying its \( PS \) takes \( O(u) \). We have potentially \( |F_1| \times |F_{k-1}| \times r \) patterns. Thus, it takes \( O(|F_1| \cdot |F_{k-1}| \cdot r \cdot u) \). The complexity of computing maxSeason is \( O(|F_1| \cdot |F_{k-1}| \cdot r \cdot k^2 \cdot u) \). For each candidate pattern, identifying its \( PS \) takes \( O(u) \). We have potentially \( |F_1| \times |F_{k-1}| \times r \) patterns. Thus, it takes \( O(|F_1| \cdot |F_{k-1}| \cdot r \cdot u) \). The complexity of frequent seasonal k-event pattern mining \((k > 2)\) is: \( O(|F_1| \cdot |F_{k-1}| \cdot r \cdot k^2 \cdot u + |F_1| \cdot |F_{k-1}| \cdot r \cdot u) \sim O(|F_1| \cdot |F_{k-1}| \cdot r \cdot k^2 \cdot u) \).

Thus, the total complexity is \( O(n^2 i^2 u^2) + O(|F_1| \cdot |F_{k-1}| \cdot r \cdot k^2 \cdot u) \).

**I. Theorem**

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

\[ \text{maxSeason}(X_1, Y_1) \geq \lambda_2 \cdot \frac{\text{minDensity} \cdot \sum_{i \neq 1, j \neq 1} p(X_i, Y_j) \cdot \log p(Y_j)}{\sum_{i} p(X_i) \cdot \log p(X_i)} \]

where: \( \lambda_1 = \min\{p(X_i), \forall X_i \in X_S\} \) is the minimum probability of \( X_1 \in X_S \), and \( \lambda_2 = \max\{p(Y_i) \} \) is the probability of \( Y_1 \in Y_S \). and \( W \) is the Lambert function [45].

**Proof.** From Eq. (5), we have:

\[ \tilde{I}(X_S; Y_S) = 1 - \frac{H(X_S|Y_S)}{H(X_S)} \geq \mu \]

Hence:

\[ \frac{H(X_S|Y_S)}{H(X_S)} \leq 1 - \mu \]

First, we derive a lower bound for \( \frac{H(X_S|Y_S)}{H(X_S)} \). We have:

\[ p(X_1, Y_1) \cdot \log \frac{p(X_1, Y_1)}{p(Y_1)} \leq p(X_1, Y_1) \cdot \log \frac{p(X_1, Y_1)}{p(Y_1)} \]

\[ \leq p(X_1, Y_1) \cdot \log \frac{p(X_1, Y_1)}{p(X_1)} \]

\[ \leq p(X_1, Y_1) \cdot \log \frac{p(X_1, Y_1)}{\lambda_2} \]

where \( \lambda_2 = p(Y_1) \).

Next, we consider the denominator in Eq. (25). Suppose that:

\[ p(X_k) = \min\{p(X_i)\}, \forall X_i \in X_S \]

Then we have:

\[ p(X_i) \geq p(X_k), \forall X_i \in X_S \]

\[ \Rightarrow \log p(X_i) \geq \log p(X_k) \]

\[ \Rightarrow p(X_i) \cdot \log p(X_i) \geq p(X_k) \cdot \log p(X_k) \]

\[ \Rightarrow \sum_{i} p(X_i) \cdot \log p(X_i) \geq \sum_{i} p(X_i) \cdot \log p(X_k) \]

\[ = \log p(X_k) \sum_{i} p(X_i) \]

\[ = \log p(X_k) \]

\[ = \log \lambda_1 \]

where \( \lambda_1 = p(X_k) \).

Replace Eqs. (26) and (28) into Eq. (25), we get:

\[ \frac{H(X_S|Y_S)}{H(X_S)} \geq \frac{p(X_1, Y_1) \cdot \log \frac{p(X_1, Y_1)}{\lambda_2}}{\log \lambda_1} \]
From Eqs. (24) and (29), it follows that:

\[
(1 - \mu) \geq \frac{p(X_1, Y_1) \cdot \log \frac{p(X_1, Y_1)}{\lambda_2}}{\log \lambda_1} \Rightarrow p(X_1, Y_1) \cdot \log \frac{p(X_1, Y_1)}{\lambda_2} \geq \log \lambda_1^{1-\mu}
\]

Assign \(x = p(X_1, Y_1), b = \log \lambda_1^{1-\mu}\). Replace \(x \) and \(b \) into Eq. (30), we get:

\[
x \cdot \log \frac{1}{\lambda_2} \geq b \\
\Rightarrow x \cdot \log \frac{x}{\lambda_2} \geq \frac{b}{\lambda_2} \\
\Rightarrow \frac{x}{\lambda_2} \cdot \ln \frac{x}{\lambda_2} \geq \frac{b}{\lambda_2} \\
\Rightarrow \frac{x}{\lambda_2} \cdot \ln \frac{x}{\lambda_2} \geq \frac{b \cdot \ln 2}{\lambda_2}
\]

Assign \(y = \ln \frac{x}{\lambda_2} \Rightarrow \frac{x}{\lambda_2} = e^y\). Replace \(y \) and \(e^y \) into Eq. (31), we get:

\[
y \cdot e^y \geq \frac{b \cdot \ln 2}{\lambda_2} \\
\Rightarrow y \geq W\left(\frac{b \cdot \ln 2}{\lambda_2}\right), \text{ where } W \text{ is the Lambert function [45].}
\]

\[
\Rightarrow \ln \left(\frac{x}{\lambda_2}\right) \geq W\left(\frac{\log \lambda_1^{1-\mu} \cdot \ln 2}{\lambda_2}\right) \\
\Rightarrow e^{\ln \left(\frac{x}{\lambda_2}\right)} \geq e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot \ln 2}{\lambda_2}\right)} \\
\Rightarrow \frac{x}{\lambda_2} \geq e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot \ln 2}{\lambda_2}\right)} \\
\Rightarrow \frac{x}{\lambda_2} \geq \lambda_2 \cdot e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot \ln 2}{\lambda_2}\right)} \\
\Rightarrow p(X_1, Y_1) \geq \lambda_2 \cdot e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot \ln 2}{\lambda_2}\right)}
\]

Since the relative support of \((X_1, Y_1)\) in \(D_{\text{SEQ}}\) is greater than or equal to the relative support of \((X_1, Y_1)\) in \(D_{\text{SYB}}\), hence:

\[
\frac{|SUP(X_1, Y_1)|}{|D_{\text{SEQ}}|} \geq \frac{|SUP(X_1, Y_1)|}{|D_{\text{SYB}}|} = p(X_1, Y_1)
\]

We can derive a lower bound of \(\mu\) from Eq. (34).

\[
\text{maxSeason}(X_1, Y_1) \geq \lambda_2 \cdot |D_{\text{SEQ}}| \cdot e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot \ln 2}{\lambda_2}\right)} \geq \text{minSeason}
\]

\[
\Rightarrow e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot \ln 2}{\lambda_2}\right)} \geq \frac{\text{minSeason} \cdot \text{minDensity}}{\lambda_2 \cdot |D_{\text{SEQ}}|} \\
\Rightarrow e^{W\left(\frac{\log \lambda_1^{1-\mu} \cdot \ln 2}{\lambda_2}\right)} \geq \rho
\]

where \(\rho = \frac{\text{minSeason} \cdot \text{minDensity}}{\lambda_2 \cdot |D_{\text{SEQ}}|}\).

To solve Eq. (35), we consider two cases.

* **Case 1**: \(0 \leq \rho \leq \frac{1}{e}\), we have:

\[
\Rightarrow \log \lambda_1^{1-\mu} \cdot \ln 2 \geq -\frac{1}{e} \\
\Rightarrow (1 - \mu) \cdot \log \lambda_1 \geq -\frac{\lambda_2}{e \cdot \ln 2} \\
\Rightarrow 1 - \mu \leq \frac{\lambda_2}{e \cdot \ln 2 \cdot \log \lambda_1} \quad (\log \lambda_1 < 0)
\]

\[
\Rightarrow \mu \geq 1 - \frac{\lambda_2}{e \cdot \ln 2 \cdot \log \lambda_1}
\]

* **Case 2**: \(\rho > \frac{1}{e}\), we have:

\[
\Rightarrow \log \lambda_1^{1-\mu} \cdot \ln 2 \geq \rho \cdot \log \rho \\
\Rightarrow (1 - \mu) \cdot \log \lambda_1 \geq \rho \cdot \ln 2 \\
\Rightarrow 1 - \mu \leq \frac{\rho \cdot \lambda_2}{\ln 2 \cdot \log \lambda_1} \quad (\log \lambda_1 < 0)
\]

\[
\Rightarrow \mu \geq 1 - \frac{\rho \cdot \lambda_2}{\ln 2 \cdot \log \lambda_1}
\]

\[\square\]

### J. Qualitative Evaluation

Tables [XIII] and [XIV] list the number of seasonal patterns found in the SC and HFM datasets. It can be seen that high \(\text{minSeason}\) leads to less generated patterns, as many have few seasonal occurrences. Moreover, higher \(\text{minDensity}\) also generates fewer patterns since only few patterns have high occurrence density. Finally, high \(\text{maxPeriod}\) results in more generated patterns, since high \(\text{maxPeriod}\) allows more temporal relations to be formed, thus increasing the number of patterns.

| \(\text{maxPeriod}\) (%) | 8.0% | 8.5% | 9.0% | 12.8% | 12.5% | 12.0% | 16.8% | 16.5% | 16.0% |
|---------------------|-----|-----|-----|-------|-------|-------|-------|-------|-------|
| \(0.4\) | 1274 | 1266 | 1278 | 1264 | 1278 | 1262 | 1276 | 1259 | 1283 |
| \(0.8\) | 1249 | 1260 | 1262 | 1258 | 1270 | 1254 | 1266 | 1259 | 1273 |
| \(1.6\) | 1262 | 1268 | 1270 | 1266 | 1278 | 1262 | 1274 | 1267 | 1281 |

TABLE XIII: The Number of Seasonal Patterns on SC
K. Baselines comparison

Figs. 17, 18, 19 and 20 show the experimental results on SC and HFM datasets. Note that Figs. 17, 18, 19 and 20 use the same legend.

As shown in Figs. 17 and 18, A-STPM achieves the best runtime among all methods, and E-STPM has better runtime than the baseline. The range and average speedups of A-STPM compared to other methods are: [1.4-3.1] and 2.2 (E-STPM), and [4.9-10.3] and 6.9 (APS-growth). The speedup of E-STPM compared to the baseline is [2.8-5.4] and 3.5 on average. Note that the time to compute MI and $\mu$ for SC and HFM in Figs. 17 and 18 are 0.9 and 1.2 seconds, respectively.

In terms of memory consumption, as shown in Figs. 19 and 20, A-STPM is the most efficient method, while E-STPM is more efficient than the baseline. The range and the average memory consumption of A-STPM compared to other methods are: [1.4-2.6] and 1.7 (E-STPM), and [2.5-6.3] and 3.7 (APS-growth). The memory usage of E-STPM compared to the baseline is [1.3-3.2] and 2.1 on average.

L. Scalability evaluation on synthetic datasets

Figs. 21 and 22 show the runtimes of A-STPM, E-STPM and the baseline when the number of sequences changes (y-axis is in log scale). We obtain the range and average speedups of A-STPM are: [1.4-2.7] and 2.1 (E-STPM), and [2.4-6.1] and 4.2 (APS-growth). Similarly, the range and average speedup of E-STPM compared to APS-growth is [1.6-3.5] and 2.8. We note that the baseline fails for larger configurations in this scalability study, i.e., on the synthetic SC at 60% sequences (Fig. 21a) and on the synthetic HFM at 100% sequences (Fig. 22a), showing that A-STPM and E-STPM can scale well on big datasets while the baseline cannot.

Figs. 23 and 24 compare the runtimes of A-STPM, E-STPM and APS-growth when changing the number of time series (y-axis is in log scale). We obtain the range and average speedups of A-STPM are: [1.4-2.9] and 2.2 (E-STPM), and [3.1-6.7] and 4.5 (APS-growth), and of E-STPM is [1.7-3.8] and 2.9 (APS-growth). The baseline also fails at large configurations in this study, i.e., when # Time Series $\geq$ 6000 on the synthetic SC (Fig. 23a) and the synthetic HFM (Fig. 24a).

Finally, we provide the percentage of time series and events pruned by A-STPM in the scalability test in Tables XV and XVI. Here, we can see that low $minSeason$ and $minDensity$ lead to more time series (events) to be pruned because low $minSeason$ and $minDensity$ result in higher $\mu$.

M. Evaluation of the pruning techniques in E-STPM

In this section, we report the evaluation results of the proposed pruning techniques in E-STPM on SC and HFM. We use 3 different configurations that vary: the minimum season, the minimum density, and the maximum period. Figs. 25 and 26 show the results (the y-axis is in log scale). It can be seen that All-E-STPM achieves the best performance among all versions. Its speedup w.r.t. NoPrune-E-STPM ranges from 2.5 up to 4.5 depending on the configurations, showing that the proposed prunings are very effective in improving E-STPM performance. The average speedup is from 2 to 4 for Trans-E-STPM, and from 1.5 to 3 for Apriori-E-STPM. However, applying both always yields better speedup than applying either of them.

N. Evaluation of A-STPM

Table XVII 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 ($\geq$ 80%) 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 XVIII shows the accuracies of A-STPM on the synthetic datasets: very high accuracy ($\geq$ 95%) when $minSeason$ and $minDensity$ are high, e.g., $minSeason$ = 16 and $minDensity$ = 0.75%.

O. Evaluation of the tolerance buffer $\epsilon$

We evaluate the impact of the buffer $\epsilon$ on extracted seasonal patterns. Tables XIX and XX report the number of extracted seasonal patterns for different $\epsilon$ values, and the corresponding percentages of pattern loss compared to $\epsilon$ = 0. For RE and SC datasets, there are no lost patterns with $\epsilon$ = 1 hour. For INF and HFM datasets, there are no lost patterns with $\epsilon$ = 1 day and $\epsilon$ = 2 days. And the losses among other $\epsilon$ values are very low since there is low noise level in the datasets.
TABLE XIX: Number of Extracted Patterns and Percentages of Pattern Loss on RE and SC

| $\epsilon$ value | RE | SC |
|------------------|----|----|
|                  | # Patterns | Patterns (%) | # Patterns | Patterns (%) |
| 1 hour           | 35626       | 0.00         | 17241      | 0.00         |
| 2 hours          | 35407       | 0.61         | 16921      | 1.85         |
| 3 hours          | 35192       | 1.21         | 16812      | 2.48         |

TABLE XX: Number of Extracted Patterns and Percentages of Pattern Loss on INF and HFM

| $\epsilon$ value | INF | HFM |
|------------------|-----|-----|
|                  | # Patterns | Patterns (%) | # Patterns | Patterns (%) |
| 1 day            | 7812     | 0.00         | 14763      | 0.00         |
| 2 days           | 7812     | 0.00         | 14763      | 0.00         |
| 3 days           | 7803     | 0.11         | 14750      | 0.08         |