Efficient Temporal Pattern Mining in Big Time Series Using Mutual Information

Ho Long, Van; Ho, Nguyen Thi Thao; Pedersen, Torben Bach

Published in:
Proceedings of the VLDB Endowment

DOI (link to publication from Publisher):
10.14778/3494124.3494147

Creative Commons License
CC BY-NC-ND 4.0

Publication date:
2022

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

Link to publication from Aalborg University

Citation for published version (APA):
Efficient Temporal Pattern Mining in Big Time Series Using Mutual Information

Van Long Ho
Aalborg University
Aalborg, Denmark
vplh@cs.aau.dk

Nguyen Ho
Aalborg University
Aalborg, Denmark
ntth@cs.aau.dk

Torben Bach Pedersen
Aalborg University
Aalborg, Denmark
tbp@cs.aau.dk

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.

PVLDB Reference Format:
doi:10.14778/3494124.3494147

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. 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 [12].

From Fig. 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 → [6:00 - 7:00, Day X+1] BoilerOn) (meaning BoilerOn is followed by BoilerOn), ([6:00 - 10:00, Day X] HighCO2 ≥ [7:00 - 8:00, Day X] BoilerOn) (meaning HighCO2 contains BoilerOn), and ([7:00 - 8:00, Day X] BoilerOn → [6:00 - 2:00, Day X+1] LowCO2 → [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 → [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 $O(3^h)$ to the complexity $O(m^3)$ of the search space ($m$ is the number of events and $h$ is the length of temporal patterns), yielding an overall complexity of $O(m^33^h)$ (see Lemma 1 in Section 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.

## 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], Ilyad 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 addressing an orthogonal problem. In [37], Amit et al. propose STIPA which uses a Hoppenr 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 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.

## 3 PRELIMINARIES

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

### 3.1 Temporal Event of Time Series

**Definition 3.1** (Time series) A *time series* $X = x_1, x_2, \ldots, x_n$ 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 $\Sigma_X$.

The symbolic time series $X_S$ is obtained using a mapping function $f: X \rightarrow \Sigma_X$ that maps each value $x_i \in X$ to a symbol $\omega_i \in \Sigma_X$. 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 $\Sigma_X = \{$On, Off$\}$, where On represents that the device is on and operating ($e_i \geq 0.5$), and Off that the device is off ($e_i < 0.5$), 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 $X = \{X_1, \ldots, X_n\}$, the set of symbolic representations of the time series in $X$ forms a symbolic database $\mathcal{DS}_S$.

An example of the symbolic database $\mathcal{DS}_S$ is shown in Table 1. There are 6 time series representing the 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 $\Sigma_X = \{$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 $T = \{[t_s, t_e]\}$ is the set of time intervals during which $X_S$ is associated with the symbol $\omega$.

Given a symbolic database $X_S$, a temporal event is created by first converting $X$ into symbolic time series $X_S$, and then combining identical consecutive symbolic time series into one single time interval. For example, consider the symbolic representation of $S$ in Table 1. By combining its consecutive On symbols, we form the temporal event ‘Stove is On’ as: (On, \{[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 $[t_s, t_e] \subseteq T$ be a time interval. The tuple $e = (\omega, [t_s, t_e])$ is called an instance of the event $E$, representing a single occurrence of $E$ during $[t_s, t_e]$. We use the notation $E_{t_s}^{e}$ to denote that event $E$ has an instance $e$.

### 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 to the relation’s endpoints. However, we change the way $\varepsilon$ 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, $e_i = (\omega_i, [t_s, t_e])$ and $e_j = (\omega_j, [t_s, t_e])$. Let $\epsilon$ 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 $e_j$.

<table>
<thead>
<tr>
<th>Time</th>
<th>On</th>
<th>On</th>
<th>On</th>
<th>Off</th>
<th>Off</th>
<th>Off</th>
<th>On</th>
<th>On</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>T</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>M</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>W</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>D</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>On</td>
</tr>
<tr>
<td>T</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>Off</td>
<td>On</td>
<td>On</td>
</tr>
</tbody>
</table>

Table 1: A Symbolic Database $\mathcal{DS}_S$
Table 2: Temporal Relations between Events

<table>
<thead>
<tr>
<th>Relation</th>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Follows</td>
<td>( E_{x}</td>
<td>E_{y} )</td>
</tr>
<tr>
<td>Contains</td>
<td>( E_{x} \subseteq E_{y} )</td>
<td>( \forall \tau_{x} \in E_{x} ), there exists ( \tau_{y} \in E_{y} ) such that ( \tau_{x} \subseteq \tau_{y} )</td>
</tr>
<tr>
<td>Overlap</td>
<td>( E_{x} \cap E_{y} )</td>
<td>( \exists \beta ) such that ( \beta \cdot \tau_{x} &lt; \beta \cdot \tau_{y} ) and ( (\beta \cdot \tau_{x}, \beta \cdot \tau_{y}) \in S_{x} \cap S_{y} )</td>
</tr>
</tbody>
</table>

3-event pattern \( P = \langle \text{Contains}, \text{HighCO2}_{e_{1}}, \text{BoilerOn}_{e_{2}} \rangle, \) \( \langle \text{Follows}, \text{HighCO2}_{e_{1}}, \text{LowCO2}_{e_{2}} \rangle, \) \( \langle \text{Follows}, \text{BoilerOn}_{e_{2}}, \text{LowCO2}_{e_{2}} \rangle \rangle \).

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 \( e_{2} \) in \( S \) must not exceed the predefined maximal time duration \( t_{\text{max}} \) (i.e., \( t_{x} - t_{y} \leq t_{\text{max}} \)).

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 \( \text{Follows} \) relation between a “Washer On” event and a “Dryer On” event in Table 3 happening one year apart should be considered invalid.

### 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 \( E \).

\[
\text{supp}(E) = |\{S \in \mathcal{D}_{\text{SEQ}} \mid S \in S : E_{e} \}| \tag{1}
\]

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

\[
\text{rel-supp}(E) = \frac{\text{supp}(E)}{|\mathcal{D}_{\text{SEQ}}|} \tag{2}
\]

Similarly, the support of a group of events \( \langle E_{1}, ..., E_{n} \rangle \), denoted as \( \text{supp}(E_{1}, ..., E_{n}) \), 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 \( P \).

\[
\text{supp}(P) = |\{S \in \mathcal{D}_{\text{SEQ}} \mid S \in S : P \}| \tag{3}
\]

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

\[
\text{rel-supp}(P) = \frac{\text{supp}(P)}{|\mathcal{D}_{\text{SEQ}}|} \tag{4}
\]

**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 \( \text{supp}(E_{i}, E_{j}) \) and the support of its most frequent event:

\[
\text{conf}(E_{i}, E_{j}) = \frac{\text{supp}(E_{i}, E_{j})}{\max \{\text{supp}(E_{i}), \text{supp}(E_{j})\}} \tag{5}
\]

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

\[
\text{conf}(P) = \frac{\text{supp}(P)}{\max \{\text{supp}(E_{i}) : E_{i} \in P\}} \tag{6}
\]

where \( E_{x} \in P \) is a temporal event. Since the denominator in Eq. (6) is the maximum support of the events in \( P \), the confidence computed in Eq. (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 consequences. Instead, they represent pairwise temporal relations between events based on their temporal occurrences. Thus, while the support and relative support of event(s)/pattern(s) defined in Eqs. (1) – (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. (5) – (6) instead represents the minimum likelihood of an event pair/pattern, knowing the likelihood of its most frequent event.

### 4 FREQUENT TEMPORAL PATTERN MINING

Fig. 2 gives an overview of the FPTMFTS process which consists of 2 phases. The first phase, **Data Transformation**, converts a set of time series \( X = \{X_{1}, ..., X_{n}\} \), let \( \mathcal{D}_{\text{SEQ}} \) be the temporal sequence database obtained from \( X \), and \( \sigma \) and \( \delta \) be the support and confidence thresholds, respectively. The FPTMFTS problem aims to find all temporal patterns \( P \) that have high enough support and confidence in \( \mathcal{D}_{\text{SEQ}} \): \( \text{supp}(P) \geq \sigma \) and \( \text{conf}(P) \geq \delta \).

#### 4.1 Data Transformation

**4.1.1 Symbolic Time Series Representation.** Given a set of time series \( X \), the symbolic representation of each time series \( X \in X \) is obtained by using a mapping function as in Def. 3.2.

**4.1.2 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
We now present our method, called Hierarchical Temporal Pattern Mining (HTPGM), to mine frequent temporal patterns from \( D_{SEQ} \). 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 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. 4 shows the HPG built from \( D_{SEQ} \) in Table 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.

Efficient bitmap indexing: We use bitmaps to index the occurrences of events and patterns in \( D_{SEQ} \), enabling fast computations of support and confidence. Specifically, each event \( E \) or pattern \( P \) found in \( D_{SEQ} \) is associated with a bitmap indicating where \( E \) or \( P \) occurs. Each bitmap \( b \) has length \( |D_{SEQ}| \) (i.e., the number of sequences), and has value \( b[i] = 1 \) if \( E \) or \( P \) is present in sequence \( i \) of \( D_{SEQ} \), or \( b[i] = 0 \) otherwise. An example bitmap can be seen at \( L_4 \) in Fig. 4. The event \( E_{On} \) has the bitmap \( b_{E_{On}} = [1,0,1,1] \), indicating that \( E_{On} \) occurs in all but the second sequence of \( D_{SEQ} \).

Constructing the bitmap is also done step by step. For single events in \( D_{SEQ} \), bitmaps are built by scanning \( D_{SEQ} \) only once. Algorithm 1 provides the pseudo-code of HTPGM. The details are explained in each mining step.

### 4.2 Frequent Temporal Patterns Mining

We present our method, called Hierarchical Temporal Pattern Graph Mining (HTPGM), to mine frequent temporal patterns from \( D_{SEQ} \). 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 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 (HPG), 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. 4 shows the HPG built from \( D_{SEQ} \) in Table 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.

Efficient bitmap indexing: We use bitmaps to index the occurrences of events and patterns in \( D_{SEQ} \), enabling fast computations of support and confidence. Specifically, each event \( E \) or pattern \( P \) found in \( D_{SEQ} \) is associated with a bitmap indicating where \( E \) or \( P \) occurs. Each bitmap \( b \) has length \( |D_{SEQ}| \) (i.e., the number of sequences), and has value \( b[i] = 1 \) if \( E \) or \( P \) is present in sequence \( i \) of \( D_{SEQ} \), or \( b[i] = 0 \) otherwise. An example bitmap can be seen at \( L_4 \) in Fig. 4. The event \( E_{On} \) has the bitmap \( b_{E_{On}} = [1,0,1,1] \), indicating that \( E_{On} \) occurs in all but the second sequence of \( D_{SEQ} \).

Constructing the bitmap is also done step by step. For single events in \( D_{SEQ} \), bitmaps are built by scanning \( D_{SEQ} \) only once. Algorithm 1 provides the pseudo-code of HTPGM. The details are explained in each mining step.

### 4.3 Mining Frequent Single Events

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

After this step, the set \( I_{freq} \) containing frequent single events is created to build \( L_1 \) of HPG. We illustrate this process using Table 3,
Algorithm 1: Hierarchical Temporal Pattern Graph Mining

Input: Temporal sequence database \( D_{SEQ} \), a support threshold \( \sigma \), a confidence threshold \( \delta \)

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

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

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 \( O(m|D_{SEQ}|) \), 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].

4.4 Mining Frequent 2-event Patterns

4.4.1 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 \( D_{SEQ} \) to check each combination of events, the complex relations between events also add an extra exponential factor \( 3^h \) 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 \( D_{SEQ} \), and \( h \) be the longest length of a temporal pattern. The total number of temporal patterns in HPG from \( L_1 \) to \( L_k \) is \( O(m^h3^h) \).

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.

4.4.2 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, \( \text{supp}(P) \leq \text{supp}(E_i, E_j) \).

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 \( \text{conf}(P) \leq \text{conf}(E_i, E_j) \).

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 \( D_{SEQ} \), using the set \( 1Freq \) found in \( L_1 \) of HPG (Alg. 1, 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 \( \text{AND} \)ing the two individual bitmaps: \( b_{ij} = \text{AND}(b_{E_i}, b_{E_j}) \). Finally, HTPGM computes the support \( \text{supp}(E_i, E_j) \) by counting the set bits in \( b_{ij} \), and comparing against \( \sigma \). If \( \text{supp}(E_i, E_j) \geq \sigma \), \( E_i, E_j \) 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 \( \delta \). 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. 1, lines 12-14). For each node \((E_i, E_j)\), we use the bitmap \( b_{ij} \) to retrieve the set of sequences \( S \) where both events are present. Next, for each sequence \( S \in S \), 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 \((E_{p_{r}}, E_{p_{r}})\) 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. 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. 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.
4.5 Mining Frequent k-event Patterns

Mining frequent k-event patterns (k ≥ 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. 1, lines 15-17).

Let (k-1)Freq be the set of frequent (k-1)-event combinations found in L_{k-1}, and IFreq 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 × IFreq. However, we observe that using IFreq to generate k-event combinations at L_k can create redundancy, since IFreq 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. 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_{r_12}, E_{r_12}), ..., (r_{(n-2)(n-1)}, E_{r_{(n-2)(n-1)}}, E_{r_{(n-2)(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 R is transitive on S′: ∀e_i ∈ S′, i < n, ∃r ∈ R s.t. r(E_{r_{i}}, E_{r_{i+1}}) 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 proved by 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} ∪ E_k can form frequent k-event temporal patterns if ∀E_i ∈ N_{k-1}, ∃r ∈ 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 IFreq 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 IFreq to remove redundant single events: FilteredIFreq = D_{k-1} ∩ IFreq. Next, the Cartesian product (k-1)Freq × FilteredIFreq 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. 1, lines 18-20). Unlike 2-event patterns, determining the relations in a k-event combination (k ≥ 3) is much more expensive, as it requires to verify the frequency of 1^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.

5 APPROXIMATE HTPGM

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

\[ H(X_S) = - \sum_{x \in \Sigma_X} p(x) \cdot \log p(x) \]  \hspace{1cm} (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 determine the outcome of X_S, given the value of Y_S, and is defined as

\[ H(X_S|Y_S) = - \sum_{x \in \Sigma_X} \sum_{y \in \Sigma_Y} p(x, y) \cdot \log \frac{p(x, y)}{p(y)} \]  \hspace{1cm} (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
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 1. We have: $p(S|O) = \frac{17}{36}$, $p(S|O) = \frac{19}{36}$, $p(T|O) = \frac{18}{36}$, and $p(T|O) = \frac{16}{36}$. We also have the joint probabilities: $p(S|T|O) = \frac{15}{36}$, $p(S|T|O) = \frac{16}{36}$, $p(S|T|O) = \frac{19}{36}$, and $p(S|T|O) = \frac{18}{36}$. Applying Eq. 9, we have $I(S; T) = 0.29$.

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 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 $\bar{I}(X_S; Y_S)$, is defined as

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

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

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

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

### 5.2 Lower Bound of the Confidence

#### 5.2.1 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 $D_{SYB}$ and $D_{SEQ}$ 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 $D_{SYB}$ and $D_{SEQ}$.

Lemma 8. Let $\text{supp}(X_1, Y_1)_{D_{SYB}}$ and $\text{supp}(X_1, Y_1)_{D_{SEQ}}$ be the support of $(X_1, Y_1)$ in $D_{SYB}$ and $D_{SEQ}$, respectively. We have the following relation: $\text{supp}(X_1, Y_1)_{D_{SYB}} \leq \text{supp}(X_1, Y_1)_{D_{SEQ}}$.

From Lemma 8, if an event pair is frequent in $D_{SYB}$, it is also frequent in $D_{SEQ}$. We now investigate the connection between $\bar{I}(X_S; Y_S)$ in $D_{SYB}$, and the confidence of $(X_1, Y_1)$ in $D_{SEQ}$.

Theorem 1. (Lower bound of the confidence) Let $\sigma$ and $\mu$ be the minimum support and mutual information thresholds, respectively.

Assume that $(X_1, Y_1)$ is frequent in $D_{SEQ}$, i.e., $\text{supp}(X_1, Y_1)_{D_{SEQ}} \geq \sigma$. If the NMI $\bar{I}(X_S; Y_S) \geq \mu$, then the confidence of $(X_1, Y_1)$ in $D_{SEQ}$ has a lower bound:

$$\text{conf}(X_1, Y_1)_{D_{SEQ}} \geq \sigma \cdot \frac{1}{\mu} \left( \frac{n_x - 1}{1 - \sigma} + \frac{n_y - 1}{1 - \sigma} \right) \tag{11}$$

where $n_x$ is the number of symbols in $X_S$, $\lambda_1$ is the minimum support of $X_1 \in X_S$, and $\lambda_2$ is the support of $(X_1, Y_1) \in (X_S, Y_S)$ such that $p(X_1|Y_1) \text{min}$, $\forall i \neq j$. Theorem 1 says that, given an MI threshold $\mu$, 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. (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 5.3).

#### 5.2.2 Shape of the confidence lower bound. To understand how the confidence changes w.r.t. the support $\sigma$ and the MI $\mu$, we analyze its shape, shown in Fig. 5 ($\sigma$ and $\mu$ vary between 0 and 1). First, it can be seen that the confidence lower bound has a direct relationship with $\sigma$ and $\mu$ (one increases if the other increases and vice versa). While the direct relationship between the confidence and $\sigma$ can be explained using Eq. (5), it is interesting to observe the connection between $\mu$ and the confidence. As the MI represents the correlation between two symbolic series, the larger the value of $\mu$, the more correlated the two series. Thus, when the confidence increases together with $\mu$, it implies that patterns with high confidence are more likely to be found in highly correlated series, and vice versa.
if 11: perform HTPGM using L

mine frequent patterns for

Definition 5.5

value. This implies that the confidence is less sensitive to

appliance. There is an edge between two nodes if their NMI is at

in Theorem 1, we can derive

frequent temporal patterns. Based on the confidence lower bound

computing the NMI for each symbolic series pair, and comparing

5.3.1 Correlation graph. Using Theorem 1, we propose to approxi-

mating HTPGM by performing the mining only on the set of corre-

symbolic series X ⊆ X. We first define the correlation graph.

Definition 5.5 (Correlation graph) A correlation graph is an un-

directed graph GC = (V, E) where V is the set of vertices, and E is the set of edges. Each vertex v ∈ V represents one symbolic series XS ∈ X. There is an edge euv between a vertex u containing XS, and a vertex v containing YS iff I(XS; YS) ≥ µ ∨ I(YS; XS) ≥ µ.

Fig. 5 shows an example of the correlation graph GC built from DSBY in Table 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 DSBY, the correlation graph GC 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 GC if their NMI is at least µ, and vice versa.

Setting the value of µ: While NMI can easily be computed using Eq. (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. (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 (X1, Y1) ∈ (XS, YS) in DSEQ at least δ if I(XS; YS) is at least µ, where:

µ ≥ 1 − σ · log x ! x − 1 ! δ ! σ

(18)

Note that µ in Eq. (18) only ensures that the event pair (X1, Y1) has a minimum confidence of δ. Thus, given (XS, YS), µ has to be computed for each event pair in (XS, YS). The final chosen µ value to be compared against I(XS; YS) is the minimum µ value among all the event pairs in (XS, YS).

5.3.2 Approximate HTPGM using the correlation graph. Using the correlation graph GC, the approximate HTPGM is described in Algorithm 2. First, DSBY 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 (XS, YS) in DSBY (lines 4-5). Then, only pairs whose I(XS; YS) or I(YS; XS) is at least µ are inserted into XC, and an edge between XS and YS is created (lines 6-8). Next, at L1 of HPG, only the correlated symbolic series in XC are used to mine frequent single events (lines 9-10). At L2, GC is used to filter 2-event combinations: for each event pair (E1, E2), we check whether there is an edge between their corresponding symbolic series in GC. If so, we proceed by verifying the support and confidence of (E1, E2) as in the exact HTPGM (lines 11-13). Otherwise, (E1, E2) is eliminated from the mining of L2. From level Lk (k ≥ 3) onwards, the exact HTPGM is used (lines 14-15).

5.3.3 Complexity analysis. To compute NMI and µ, we only have to scan DSBY once to calculate the probability for each single event and pair of events. Thus, the cost of NMI and µ computations is |DSBY|. On the other hand, the complexity of the exact HTPGM at L1 and L2 are O(2m2) + O(m |DSEQ|) (Section 4.4). Thus, the approximate HTPGM is significantly faster than HTPGM.

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].

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 5 summarizes their characteristics.

Baseline methods: Our exact method is referred to as E-HPGM, and the approximate one as A-HPGM. We use 4 baselines (described in Section 2): Z-Miner [28], TPMiner [8], IEMiner [56], and H-DFS [35]. Since E-HPGM and the baselines provide the same exact solutions, we use the baselines only for the quantitative evaluation, and compare only E-HPGM and A-HPGM 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 4 lists the parameters and their values used in our experiments.
6.2 Qualitative Evaluation

Our goal is to make sense and learn insights from extracted patterns. Table 7 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.

6.3 Quantitative Evaluation

6.3.1 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 8 and 9 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 8, 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 baselines 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 \( \mu \) for the NIST and the smart city datasets in Table 8 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 6.3.2 and 6.3.4).

In terms of memory consumption, as shown in Table 9, 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).

Finally, in Table 11, we provide the pre-processing times to convert the raw time series to \( D_{S,YB} \) and \( D_{S,YB} \) to \( D_{SEQ} \). We also report the sizes of \( D_{SYB} \) and \( D_{SEQ} \) stored on disk. We see that while the storage costs for \( D_{SYB} \) and \( D_{SEQ} \) 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.

6.3.2 Scalability evaluation on synthetic datasets. As discussed in Section 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. 7 and 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. 9 and 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), 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. 9 and 10, to illustrate the computation time of MI and \( \mu \), 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 \( \mu \) (top red), and the mining time (bottom blue). However, note that for each dataset, we only need to compute MI and \( \mu \) once (the computed values are used across the mining process with different support and confidence thresholds). Thus, the times to compute MI and \( \mu \) for the NIST and the smart city datasets in Fig. 9 are 200 and 90 seconds, respectively.

Moreover, most baselines fail for the larger configurations in the scalability study, e.g., Z-Miner on the NIST dataset when \( \sigma = 20\% \) (Fig. 7a), and Z-Miner, TPMiner, IEMiner and H-DFS when the number of time series grows to 1000 (Fig. 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.

Furthermore, the number of time series and events pruned by A-HTPGM in the scalability test are provided in Table 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 \( \mu \), and thus, more pruned time series.
6.3.3 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, 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. 11, 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, (Trans)-E-HTPGM delivers larger speedup than (Apriori)-E-HTPGM. The average speedup is from 8 to 20 for (Trans)-E-HTPGM, and from 3 to 12 for (Apriori)-E-HTPGM. However, applying both always yields better speedup than applying either of them.

6.3.4 Evaluation of A-HTPGM. 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 6 shows the accuracies of A-HTPGM for different supports and confidences. It is seen that, A-HTPGM obtains high accuracy (≥ 71%) when σ and δ are low, e.g., σ = δ = 10%, and very high accuracy (≥ 95%) when σ and δ are high, e.g., σ = δ = 50%. Next, we analyze the quality of patterns pruned by A-HTPGM. These patterns are extracted from the uncorrelated time series. Fig. 13 shows the cumulative distribution of the confidences of the pruned patterns. It is 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 parameter ϵ, and the overlapping duration tω, to the quality of extracted patterns. The analysis can be seen in the full paper [19].

7 CONCLUSION AND FUTURE WORK

This paper presents our comprehensive Frequent Temporal Pattern Mining from Time Series (FTPMTS) solution that offers: (1) an end-to-end FTPMTS 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.

ACKNOWLEDGMENTS

This work has been partially supported by the MORE project (grant agreement 957345), funded by the EU Horizon 2020 program.
REFERENCES


[32] Carol Neidle, Augustine Opolou, Gregory Dimitriadis, and Dimitris Metaxas. 2018. NEW Shared & Interconnected ASL References: 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.


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


