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

Publication date:
2022

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

Citation for published version (APA):
Ho Long, V., Ho, N. T. T., & Pedersen, T. B. (2022). Efficient Temporal Pattern Mining in Big Time Series Using Mutual Information. *Proceedings of the VLDB Endowment*, *15*(3), 673-685.
https://doi.org/10.14778/3494124.3494147

Efficient Temporal Pattern Mining in Big Time Series Using Mutual Information

Van Long Ho
Aalborg University
Aalborg, Denmark
vlh@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:
Van Long Ho, Nguyen Ho, and Torben Bach Pedersen. Efficient Temporal Pattern Mining in Big Time Series Using Mutual Information. PVLDB, 15(3): 673-685, 2022.
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].

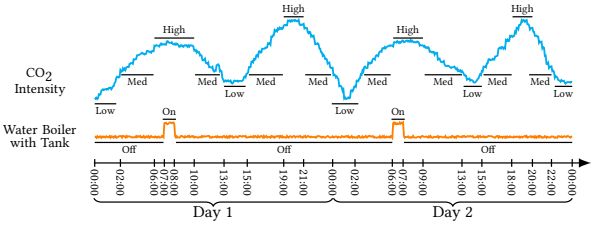


Figure 1: CO2 intensity and water boiler electricity usage

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 $\rightarrow$ [6:00 - 7:00, Day X+1] BoilerOn) (meaning BoilerOn is followed by BoilerOn), ([6:00 - 10:00, Day X] HighCO2 $\succcurlyeq$ [7:00 - 8:00, Day X] BoilerOn) (meaning HighCO2 contains BoilerOn), and ([7:00 - 8:00, Day X] BoilerOn $\rightarrow$ [0:00 - 2:00, Day X+1] LowCO2 $\rightarrow$ [6:00 - 7:00, Day X+1] BoilerOn). As the resident is very keen on reducing her CO2 footprint, we can rely on the above temporal patterns to automatically (using the smart-plug) delay turning on the boiler until the CO2 intensity is low again, saving CO2 without any loss of comfort for the resident.

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

Proceedings of the VLDB Endowment, Vol. 15, No. 3 ISSN 2150-8097.
doi:10.14778/3494124.3494147

Although temporal patterns are useful, mining them is much more expensive than sequential patterns. Not only does the temporal information add extra computation to the mining process, the complex relations between events also add an additional exponential factor $O(3^{h^2})$ to the complexity $O(m^h)$ of the search space (m is the number of events and h is the length of temporal patterns), yielding an overall complexity of $O(m^h 3^{h^2})$ (see Lemma 1 in Section 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], Iyad et al. propose a TPM framework to detect events in time series. However, their focus is to find irregularities in the data. In [38], Wang et al. propose a temporal pattern mining algorithm HUTPMiner to mine high-utility patterns. Different from our HTPGM which uses *support* and *confidence* to measure the frequency of patterns, HUTPMiner uses *utility* to measure the importance or profit of an event/ pattern, thereby addresses an orthogonal problem. In [37], Amit et al. propose STIPA which uses a Hoeppner matrix representation to compress temporal patterns for memory savings. However, STIPA does not use any pruning/ optimization strategies and thus, despite the efficient use of memory, it cannot scale to large datasets, unlike our HTPGM. Other work [4], [7] proposes TPM algorithms to classify health record data. However, these methods are very domain-specific, thus cannot generalize to other domains.

The state-of-the-art TPM methods that currently achieve the best performance are our baselines: H-DFS [35], TPMiner [8], IEMiner [36], and Z-Miner [28]. H-DFS is a hybrid algorithm that uses breadth-first and depth-first search strategies to mine frequent arrangements of temporal intervals. H-DFS uses a data structure called ID-List to transform event sequences into vertical representations, and temporal patterns are generated by merging the ID-Lists of different events. This means that H-DFS does not scale well when the number of time series increases. In [36], Patel et al. design a hierarchical lossless representation to model event relations, and propose IEMiner that uses Apriori-based optimizations to efficiently mine patterns from this new representation. In [8], Chen et al. propose TPMiner that uses endpoint and endtime representations to simplify the complex relations among events. Similar to [35], IEMiner and TPMiner do not scale to datasets with many time series. Z-Miner [28], proposed by Lee et al., is the most recent work addressing TPM. Z-Miner improves the mining efficiency over existing methods by employing two data structures: a hierarchical hash-based structure called Z-Table for time-efficient candidate generation and support count, and Z-Arrangement, a structure to efficiently store event intervals in temporal patterns for efficient memory consumption. Although using efficient data structures, Z-Miner neither employs the transitivity property of temporal relations nor mutual information for pruning. Thus, Z-Miner is less efficient than our exact and approximate HTPGM in both runtimes and memory usage, and does not scale to large datasets with many sequences and many time series (see Section 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* $X = x_1, x_2, ..., x_n$ is a sequence of data values that measure the same phenomenon during an observation time period, and are chronologically ordered.

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

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

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

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

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

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

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

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

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

3.2 Relations between Temporal Events

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

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

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

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

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

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

3.3 Temporal Pattern

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

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

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

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

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

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

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

In Fig. 1, consider the sequence S =<e_1=(HighCO2, [6:00, 10:00]), e_2=(BoilerOn, [7:00, 8:00]), e_3=(LowCO2, [13:00, 15:00])> representing the order of CO2 intensity and boiler events. Here, S supports a

Table 2: Temporal Relations between Events

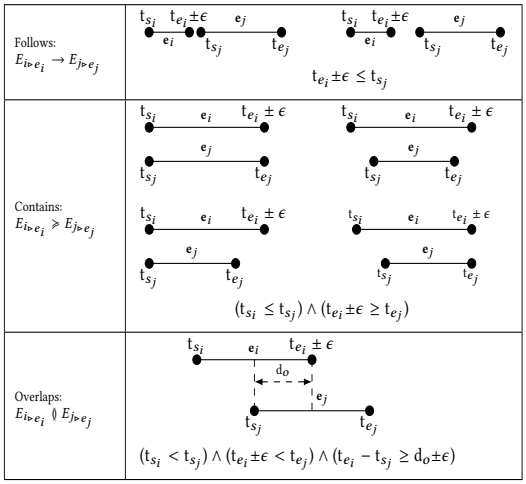


Table 3: A Temporal Sequence Database $\mathcal{D}_{\text{SEQ}}$

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

3-event pattern P=<(Contains, HighCO2$_{\triangleright} e_1$, BoilerOn$_{\triangleright} e_2$), (Follows, HighCO2$_{\triangleright} e_1$, LowCO2$_{\triangleright} e_3$), (Follows, BoilerOn$_{\triangleright} e_2$, LowCO2$_{\triangleright} e_3$)>.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

where $E_k \in P$ 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 consequents. 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.

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

4 FREQUENT TEMPORAL PATTERN MINING

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

4.1 Data Transformation

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

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

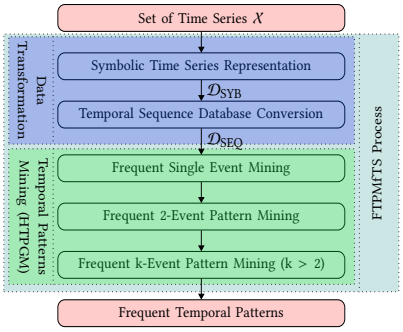
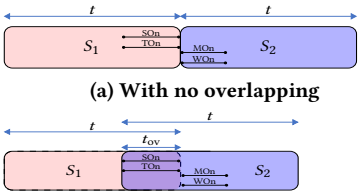


Figure 2: The FTPMfTS process



(a) With no overlapping

(b) With overlapping

Figure 3: Splitting strategy

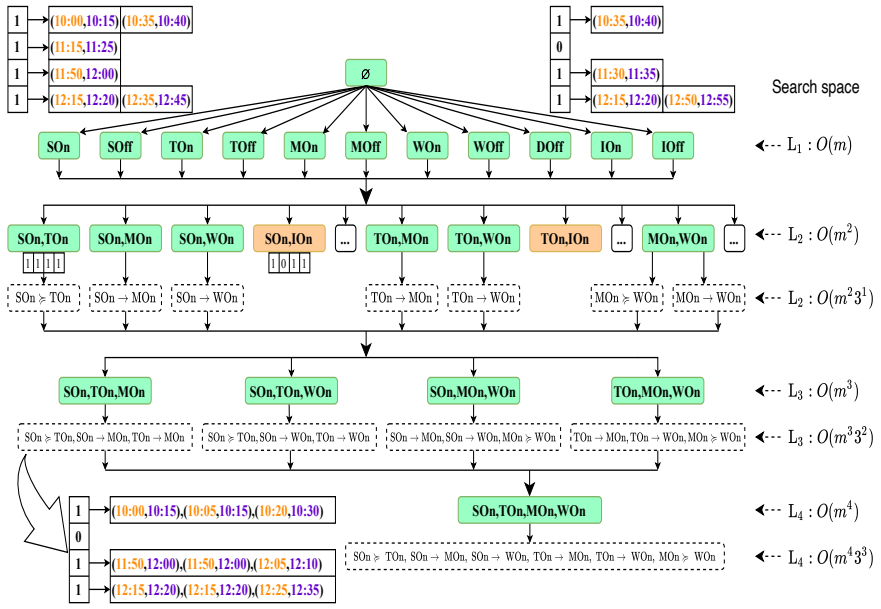


Figure 4: A Hierarchical Pattern Graph for Table 3

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

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

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

4.2 Frequent Temporal Patterns Mining

We now present our method, called Hierarchical Temporal Pattern Graph Mining (HTPGM), to mine frequent temporal patterns from $\mathcal{D}_{SEQ}$. 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 $\mathcal{D}_{SEQ}$ 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 $\mathcal{D}_{SEQ}$, enabling fast computations of support and confidence. Specifically, each event E or pattern P found in $\mathcal{D}_{SEQ}$ is associated with a *bitmap* indicating where E or P occurs. Each *bitmap* b has length $|\mathcal{D}_{SEQ}|$ (i.e., the number of sequences), and has value $b[i] = 1$ if E or P is present in sequence i of $\mathcal{D}_{SEQ}$, or $b[i] = 0$ otherwise. An example *bitmap* can be seen at L_1 in Fig. 4. The event IOn has the *bitmap* $b_{IOn} = [1,0,1,1]$, indicating that IOn occurs in all but the second sequence of $\mathcal{D}_{SEQ}$.

Constructing the *bitmap* is also done step by step. For single events in $\mathcal{D}_{SEQ}$, *bitmaps* are built by scanning $\mathcal{D}_{SEQ}$ 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 $\mathcal{D}_{SEQ}$, the support $supp(E_i)$ is computed by counting the number of set bits in *bitmap* b_{E_i}, and comparing against σ. Note that for single events, *confidence* is not considered since it is always 1.

After this step, the set *1Freq* containing frequent single events is created to build L_1 of HPG. We illustrate this process using Table 3,

Algorithm 1: Hierarchical Temporal Pattern Graph Mining

Input: Temporal sequence database $\mathcal{D}_{\text{SEQ}}$, a support threshold σ, a confidence threshold δ
Output: The set of frequent temporal patterns P
// Mining frequent single events
1: **foreach** *event* $E_i \in \mathcal{D}_{SEQ}$ **do**
2: $supp(E_i) \leftarrow$ countBitmap(b_{E_i});
3: **if** $supp(E_i) \geq \sigma$ **then**
4: Insert E_i to *1Freq*;
// Mining frequent 2-event patterns
5: EventPairs ← Cartesian(*1Freq*,*1Freq*);
6: FrequentPairs ← ∅;
7: **foreach** (E_i, E_j) *in EventPairs* **do**
8: $b_{ij} \leftarrow$ AND(b_{E_i},b_{E_j});
9: $supp(E_i, E_j) \leftarrow$ countBitmap(b_{ij});
10: **if** $supp(E_i, E_j) \geq \sigma$ **then**
11: FrequentPairs ← Apply_Lemma3(E_i, E_j);
12: **foreach** (E_i, E_j) *in FrequentPairs* **do**
13: Retrieve event instances;
14: Check frequent relations;
// Mining frequent k-event patterns
15: Filtered1Freq ← Transitivity_Filtering(1Freq); //Lemmas 4, 5
16: kEventCombinations ← Cartesian(*Filtered1Freq*,*(k-1)Freq*);
17: FrequentkEvents ← Apriori_Filtering(kEventCombinations);
18: **foreach** *kEvents in FrequentkEvents* **do**
19: Retrieve relations;
20: Iteratively check frequent relations; //Lemmas 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 $O(m{\cdot}|\mathcal{D}_{\text{SEQ}}|)$, where m is the number of distinct events.

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

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 $\mathcal{D}_{\text{SEQ}}$ to check each combination of events, the complex relations between events also add an extra exponential factor 3^{h^2} to the m^h number of possible candidates, creating a very large search space that makes the approach infeasible.

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

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

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, $supp(P) \leq supp(E_i, E_j)$.*

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

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

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

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

Step 2.2. Mining frequent 2-event patterns: This step finds frequent 2-event patterns from the nodes in L_2 (Alg. 1, lines 12-14). For each node $(E_i, E_j) \in L_2$, we use the *bitmap* b_{ij} to retrieve the set of sequences $\mathcal{S}$ where both events are present. Next, for each sequence $S \in \mathcal{S}$, the pairs of event instances (e_i, e_j) are extracted, and the relations between them are verified. The support and confidence of each relation $r(E_{i_{\triangleright e_i}}, E_{j_{\triangleright e_j}})$ are computed and compared against the thresholds, after which only frequent relations are selected and stored in the corresponding node in L_2. Examples of the relations in L_2 can be seen in Fig. 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.

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

4.5 Mining Frequent k-event Patterns

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

Step 3.1. Mining frequent k-event combinations: This step finds frequent k-event combinations in L_k (Alg. 1, lines 15-17).

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

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

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

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

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

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

Step 3.2 Mining frequent k-event patterns: This step finds frequent k-event patterns from the nodes in L_k (Alg. 1, lines 18-20). Unlike 2-event patterns, determining the relations in a k-event combination ($k \geq 3$) is much more expensive, as it requires to verify the frequency of $\frac{1}{2}k(k-1)$ triples. To reduce the cost of relation checking, we propose an iterative verification method that relies on the *transitivity property* and the Apriori principle.

Lemma 6. *Let P and P' be two temporal patterns. If $P' \subseteq P$, then conf(P') ≥ conf(P).*

Lemma 7. *Let P and P' be two temporal patterns. If $P' \subseteq P$ and $\frac{supp(P')}{\max_{1 \leq k \leq |P|}\{supp(E_k)\}} \leq \delta$, then conf($P$) ≤ δ.*

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

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

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

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

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

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

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

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

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

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

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

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

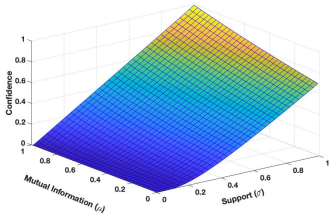


Figure 5: Shape of the lower bound

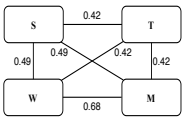


Figure 6: Corr. graph

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

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

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

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

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

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

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

Using Table 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 $\widetilde{I}(S; T) = 0.43$, which says that the uncertainty of S is reduced by 43% given T. Moreover, we also have $\widetilde{I}(T; S) = 0.42$, showing that $\widetilde{I}(S; T) \neq \widetilde{I}(T; S)$.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

It follows that:

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

□

Interpretation of the confidence lower bound: Theorem 1 says that, given an MI threshold μ, if the two symbolic series X_S and Y_S are correlated, then the confidence of a frequent event pair in (X_S, Y_S) is at least the lower bound in Eq. (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 σ and the MI μ, we analyze its shape, shown in Fig. 5 (σ and μ vary between 0 and 1). First, it can be seen that the confidence lower bound has a *direct relationship* with σ and μ (one increases if the other increases and vice versa). While the *direct relationship* between the confidence and σ can be explained using Eq. (5), it is interesting to observe the connection between μ and the confidence. As the MI represents the correlation between two symbolic series, the larger the value of μ, the more correlated the two series. Thus, when the confidence increases together with μ, it implies that patterns with high confidence are more likely to be found in highly correlated series, and vice versa.

Algorithm 2: Approximate HTPGM using MI

Input: A set of time series $\mathcal{X}$, an MI threshold μ, support threshold σ, confidence threshold δ

Output: The set of frequent temporal patterns P

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

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

5.3 Using the Bound to Approximate HTPGM

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

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

Fig. 6 shows an example of the correlation graph G_C built from $\mathcal{D}_{\text{SYB}}$ 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 $\mathcal{D}_{\text{SYB}}$, the correlation graph G_C can easily be constructed by computing the NMI for each symbolic series pair, and comparing their NMI against the threshold μ. A symbolic series pair is included in G_C if their NMI is at least μ, and vice versa.

Setting the value of μ: While NMI can easily be computed using Eq. (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* $(X_1, Y_1) \in (X_S, Y_S)$ *in* $\mathcal{D}_{SEQ}$ *is at least* δ *if* $\widetilde{I}(X_S; Y_S)$ *is at least* μ*, where:*

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

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

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

5.3.3 Complexity analysis. To compute NMI and μ, we only have to scan $\mathcal{D}_{\text{SYB}}$ once to calculate the probability for each single event and pair of events. Thus, the cost of NMI and μ computations is $|\mathcal{D}_{\text{SYB}}|$. On the other hand, the complexity of the exact HTPGM at L_1 and L_2 are $O(m^2 i^2 \ |\mathcal{D}_{\text{SEQ}}|^2) + O(m \cdot |\mathcal{D}_{\text{SEQ}}|)$ (Section 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-HTPGM, and the approximate one as A-HTPGM. We use 4 baselines (described in Section 2): Z-Miner [28], TPMiner [8], IEMiner [36], and H-DFS [35]. Since E-HTPGM and the baselines provide the same exact solutions, we use the baselines only for the quantitative evaluation, and compare only E-HTPGM and A-HTPGM qualitatively.

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

Parameters: Table 4 lists the parameters and their values used in our experiments.

Table 4: Parameters and values

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

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 μ 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., σ = 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 $\mathcal{D}_{SYB}$, and $\mathcal{D}_{SYB}$ to $\mathcal{D}_{SEQ}$. We also report the sizes of $\mathcal{D}_{SYB}$ and $\mathcal{D}_{SEQ}$ stored on disk. We see that while the storage costs for $\mathcal{D}_{SYB}$ and $\mathcal{D}_{SEQ}$ 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 μ, we add an additional bar chart for A-HTPGM. Each bar represents the runtime of A-HTPGM with two separate components: the time to compute MI and μ (top red), and the mining time (bottom blue). However, note that for each dataset, we only need to compute MI and μ once (the computed values are used across the mining process with different support and confidence thresholds). Thus, the times to compute MI and μ, for example, in Figs. 9a, 9b, and 9c, are added only for comparison and are not all actually used.

Moreover, most baselines fail for the larger configurations in the scalability study, e.g., Z-Miner on the NIST dataset when σ=δ=20% (Fig. 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 μ, and thus, more pruned time series.

Table 5: Characteristics of the Datasets

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

Table 6: The Accuracy of A-HTPGM (%)

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

Table 7: Summary of Interesting Patterns

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

Table 8: Runtime Comparison (seconds)

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

Table 9: Memory Usage Comparison (MB)

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

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

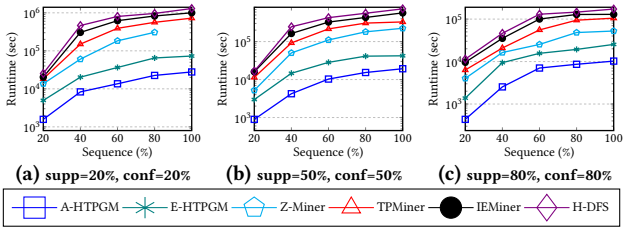
# Attr.	NIST						Smart City					
	# Pruned Time Series			# Pruned Events			# Pruned Time Series			# Pruned Events		
	20-20	20-50	20-80	20-20	20-50	20-80	20-20	20-50	20-80	20-20	20-50	20-80
200	23	55	83	46	110	166	11	27	43	27	87	135
400	37	101	157	74	202	314	17	49	81	57	197	309
600	45	141	225	90	282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

Table 11: Building $\mathcal{D}_{SYB}$ and $\mathcal{D}_{SEQ}$

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

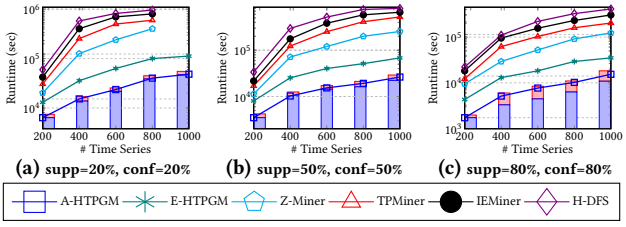
6.3.3 Evaluation of the pruning techniques in E-HTPGM. We compare different versions of E-HTPGM to understand how effective the pruning techniques are: (1) NoPrune: E-HTPGM with no pruning, (2) Apriori: E-HTPGM with Apriori-based pruning (Lemmas 2, 3), (3) Trans: E-HTPGM with transitivity-based pruning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

We use 3 different configurations that vary: the number of sequences, the confidence, and the support. Figs. 11, 12 show the



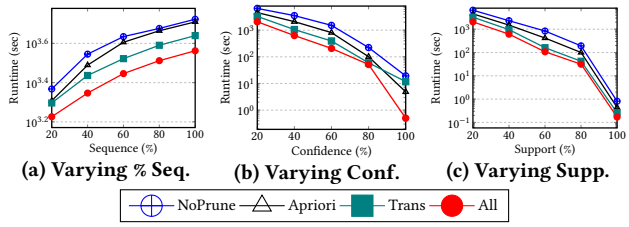
(a) supp=20%, conf=20% (b) supp=50%, conf=50% (c) supp=80%, conf=80%

Figure 7: Varying % of sequences on NIST



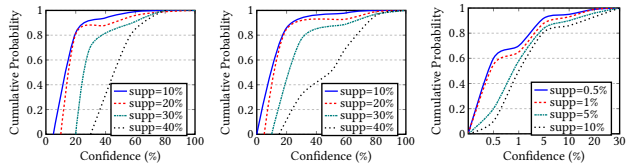
(a) supp=20%, conf=20% (b) supp=50%, conf=50% (c) supp=80%, conf=80%

Figure 9: Varying # of time series on NIST



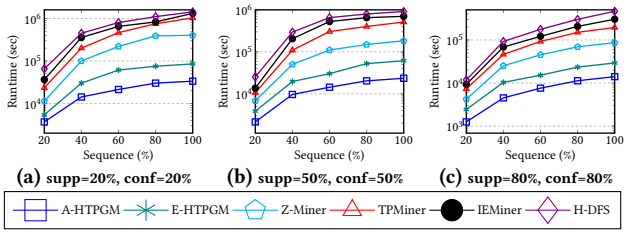
(a) Varying % Seq. (b) Varying Conf. (c) Varying Supp.

Figure 11: Runtimes of E-HTPGM on NIST



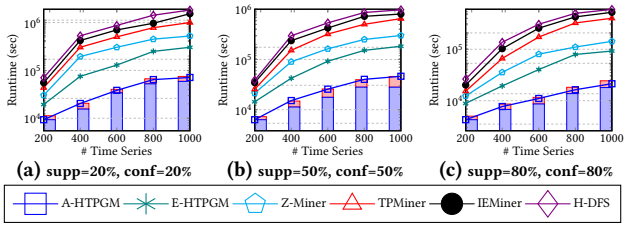
(a) NIST (b) Smart City (c) ASL

Figure 13: Cumulative probability of pruned patterns



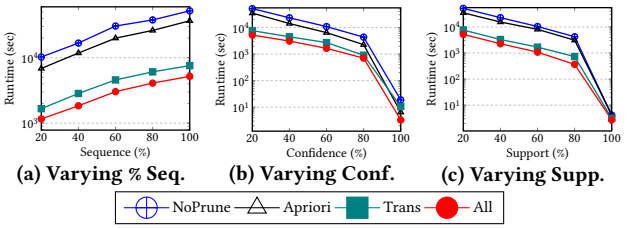
(a) supp=20%, conf=20% (b) supp=50%, conf=50% (c) supp=80%, conf=80%

Figure 8: Varying % of sequences on Smart City



(a) supp=20%, conf=20% (b) supp=50%, conf=50% (c) supp=80%, conf=80%

Figure 10: Varying # of time series on Smart City



(a) Varying % Seq. (b) Varying Conf. (c) Varying Supp.

Figure 12: Runtimes of E-HTPGM on Smart City

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 ($\geq$ 71%) when σ and δ are low, e.g., $\sigma = \delta =$ 10%, and very high accuracy ($\geq$ 95%) when σ and δ are high, e.g., $\sigma = \delta =$ 50%. Next, we analyze the quality of patterns pruned by A-HTPGM. These patterns are extracted from the uncorrelated time series. Fig. 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 buffer ϵ, and the overlapping duration t_{ov} 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 (FTPMfTS) solution that offers: (1) an end-to-end FTPMfTS process to mine frequent temporal patterns from time series, (2) an efficient and exact Hierarchical Temporal Pattern Graph Mining (E-HTPGM) algorithm that employs efficient data structures and multiple pruning techniques to achieve fast mining, and (3) an approximate A-HTPGM that uses mutual information to prune unpromising time series, allows HTPGM to scale on big datasets. Extensive experiments conducted on real world and synthetic datasets show that both A-HTPGM and E-HTPGM outperform the baselines, consume less memory, and scale well to big datasets. Compared to the baselines, the approximate A-HTPGM delivers an order of magnitude speedup on large synthetic datasets and up to 2 orders of magnitude speedup on real-world datasets. In future work, we plan to extend HTPGM to prune at the event level to further improve its performance.

ACKNOWLEDGMENTS

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

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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