HHS Public Access

Author manuscript

Expert Syst. Author manuscript; available in PMC 2020 November 06.

Published in final edited form as:

Expert Syst. 2019 October ; 36(5): . doi:10.1111/exsy.12448.

Extended vertical lists for temporal pattern mining from multivariate time series

Anton Kocheturov[1], **Petar Momcilovic**[2], **Azra Bihorac**[3], **Panos M. Pardalos**[1]

[1]Center for Applied Optimization, Industrial and Systems Engineering, University of Florida, Gainesville, Florida

[2]Industrial and Systems Engineering, University of Florida, Gainesville, Florida

[3]Division of Nephrology, Hypertension, and Renal Transplantation, University of Florida, Gainesville, Florida

Abstract

In this paper, the problem of mining complex temporal patterns in the context of multivariate time series is considered. A new method called the Fast Temporal Pattern Mining with Extended Vertical Lists is introduced. The method is based on an extension of the level-wise property, which requires a more complex pattern to start at positions within a record where all of the subpatterns of the pattern start. The approach is built around a novel data structure called the Extended Vertical List that tracks positions of the first state of the pattern inside records and links them to appropriate positions of a specific subpattern of the pattern called the prefix. Extensive computational results indicate that the new method performs significantly faster than the previous version of the algorithm for Temporal Pattern Mining; however, the increase in speed comes at the expense of increased memory usage.

Keywords

frequent pattern mining; level-wise property; temporal patterns; time-interval patterns; vertical data format

1 | INTRODUCTION

Continuously expanding resources for computing, data storage, and transmission have enabled pattern mining in complex data sets emerging from various domains such as transaction databases (Agrawal, Imielinski, & Swami, 1993a; Agrawal, Imielinski, & Swami, 1993b), web mining (Srivastava, Cooley, Deshpande, & Tan, 2000), Internet of

Correspondence: Anton Kocheturov, Corporate Technology, Siemens Corporation, Princeton, NJ, USA. antrubler@gmail.com.
Present Address
Anton Kocheturov, Corporate Technology, Siemens Corporation Princeton, NJ, USA.
Petar Momcilov, Department of Industrial and Systems Engineering, Texas A&M University, College Station, Texas

AUTHOR CONTRIBUTION

The authors contributed equally to the manuscript.

CONFLICT OF INTEREST

The authors declare no potential conflict of interests.

Things (Chen et al., 2015; Tsai, Lai, Chiang, & Yang, 2014), medicine (Batal et al., 2016; Hauskrecht et al., 2013), fraud detection (Seeja & Zareapoor, 2014), finance (Tiple, Cavique, & Cavalheiro Marques, 2017), and so on. This paper demonstrates the problem of extracting temporal patterns (TPs) from multivariate time series records. The primary contribution of the paper is a faster algorithm for mining class-specific patterns that have temporal relations between their states. The motivation behind this contribution was to develop an algorithm that could be built into real-time analytical engines.

Frequent pattern mining is the problem of finding all TPs that appear frequently in a database (Agrawal et al., 1993a; Agrawal et al., 1993b); (Yun, Lee, & Lee, 2016). In the simplest case, such patterns are frequent itemsets or subsets of items that appear in a significant proportion of transactions of the database (Agrawal et al., 1993b). Sequential pattern mining is an extension of the frequent itemset mining where the order of items or subsets of items is available (Zaki, 2001; Ayres, Flannick, Gehrke, & Yiu, 2002). TPs arise in a natural way where additional temporal information (e.g., start and end times of the events) is available (Moskovitch & Shahar, 2015). This case is the primary focus of this paper. Frequent graph mining (Kuramochi & Karypis, 2001; Zaki, 2005) is another direction of frequent pattern mining with such applications as clustering of XML documents (Aggarwal, Ta, Wang, Feng, & Zaki, 2007), chemical compound classification (Deshpande, Kuramochi, Wale, & Karypis, 2005), and so on. Uncertain pattern mining is a relatively new research direction where each item is present in a database with a certain probability (Tong, Chen, Cheng, & Yu, 2012; Lee & Yun, 2017).

In this paper, a new algorithm called the Fast Temporal Pattern Mining with Extended Vertical Lists (FTPMwEVLs) for mining frequent TPs (FTPs) is introduced. The idea is to utilize the level-wise (Aggarwal & Han, 2014) property on the level of pattern positions inside records. The level-wise property states that a TP may appear only in the records where all of its subpatterns appear. For example, if pattern “heart rate (HR) is very high *before* blood pressure (BP) is low” is found in record *i*, then both its subpatterns “HR is very high” and “BP is low” must appear in record *i*. The level-wise property was used to reduce the search space for mining TPs (Batal et al., 2016; Moskovitch & Shahar, 2015) and similar notions as itemsets (Zaki, 2000) and sequential patterns (Ayres et al., 2002; Zaki, 2001) via the vertical data format (Zaki, 2000) that tracks the occurrences of the pattern inside records. We suggest a new data structure called the Extended Vertical List (EVL) that keeps track of positions of the first state of the TP inside the records and links them to the positions of a prefix of the TP (a subpattern obtained by removing the first state of the TP) inside the records. This idea allows to reduce the computational time of FTPMwEVL by a factor of several hundreds on several data sets (Section 5). The increase in speed comes at the cost of increased memory usage that is a common trade-off in such algorithms.

This paper continuous our previous work (Kocheturov & Pardalos, 2018) in the following ways: introduction of the EVL data structure, introduction of the smallest chain for faster pattern verification, and extensive computational results to demonstrate the effectiveness of the suggested approach. The algorithm is an improved version of the method for Frequent Pattern Mining by Batal, Valizadegan, Cooper, and Hauskrecht (2011), hereinafter referred to as the Fast Temporal Pattern Mining (FTPM), where the TP is defined with no additional constraints. The output of the new algorithm coincides with the output of FTPM because it finds all FTPs. Therefore, the increased speed of mining is the main focus and the main result of this paper. The applicability of the mined patterns for a consecutive classification of the records was not analyzed.

The rest of the paper is organized as follows. We review related work and recent developments in Section 2. Section 3 provides a formal statement of the problem and all supporting definitions. We present the FTPMwEVL algorithm in Section 4 and provide the computational results in Section 5. Section 6 concludes the paper.

2 | RELATED WORK

The problem of mining FTPs considered in this paper deals with a special case of TPs called the time-interval relationship patterns that are referred to hereafter as TPs for simplicity. Each TP is a sequence of states, or time-interval events, with temporal relationships defined for each pair of the states (Definitions 1 and 2). Many multivariate time series classification methods are not applicable when records are composed of multivariate time series sampled unevenly in time. The FTP mining approach is a perfect candidate in this situation.

To define temporal relationships between states in a pattern, Allen's 13 temporal relations are usually used (Allen, 1984): *before, equal, meets, overlaps, during, starts, finishes*, and the other six are obtained by inverting. The first seven relations are enough if the states are ordered appropriately. In this paper, only two temporal relations *before* and *co-occurs* (the later combines *equal, meets, overlaps, during, starts*, and *finishes)* are used because the initial seven relations are ambiguous in the presence of noise and temporal data with a high sampling frequency (Batal, Fradkin, Harrison, Moerchen, & Hauskrecht, 2012), which leads to the problem of pattern segmentation (Moerchen, 2006).

After converting multivariate time series records to multivariate state sequences (MSSs; Definition 1), the number of state intervals per sequence was at the level of several hundreds for the data sets studied in this paper. In this situation, the breadth-first search algorithms are more efficient than those using depth-first search. Several depth-first algorithms were introduced over the years for mining of FTPs (Moskovitch & Shahar, 2015; Patel, Hsu, & Lee, 2008; Papapetrou, Kollios, Sclaroff, & Gunopulos, 2009; Winarko & Roddick, 2007; Wu & Chen, 2007). The approach by Moskovitch et al. named KarmaLego was reported to outperform other depth-first search methods (Moskovitch & Shahar, 2015). KarmaLego failed to mine all TPs for a majority of the data sets considered in Section 5; therefore, the computational section presents results for FTPMwEVL and FTPM only. Both methods are breadth-first search. The efficiency of bread-first search comes from the fact that careful pattern elimination is paramount when MSSs consist of a large number of state intervals.

3 | PROBLEM DEFINITION

We follow the definitions given in Batal et al. (2011) with slight modifications for the presentation to be self-contained.

Assume data set D of n records $d_i = (x_1^i, x_2^i, \ldots, x_m^i, y_i)$, $i = 1, \ldots, n$, where each record is composed of m time series $x_j^i \in X_j$ and an outcome, or class label, $y_i \in Y$. We start with reducing dimensionality by converting each time series into a set of temporal abstractions in the form

$$\langle (V_1, s_1, e_1), \ldots, (V_k, s_k, e_k) \rangle,$$

where $V_i \in \Sigma$ is a temporal abstraction that is in effect from start time s_i till end time e_i, for example, temporal abstraction ("low", 5, 12) means that the time variable was low from time moment 5 till time moment 12; Σ is the alphabet or set of possible abstractions (e.g., Σ = {"low," "normal," and "high"}). For a given set of temporal abstractions, we also require $s_1 \le e_1 \le s_2 \ldots \le s_k \le e_k$, meaning that no abstraction can start earlier than any previous one finishes.

The alphabet Σ can be defined in several ways. In this paper, we focus on value and trend abstractions. Value abstraction can be defined in the following ways: $\Sigma = \{\downarrow\downarrow, \downarrow, -, \uparrow, \uparrow\uparrow\}$, where $\downarrow\downarrow$, $\downarrow$, $-$, $\uparrow$, and $\uparrow\uparrow$ stand for "very low," "low," "normal," "high," and "very high," respectively. Exact ranges for transformation may be defined by a field expert. Trend abstractions may include $\Sigma = \{\rightarrow, \nearrow, \downarrow\}$, where $\rightarrow$, $\nearrow$, and $\downarrow$ stand for "steady," "increasing," and "decreasing," respectively. For this transformation, we used the approach by Keogh, Chu, Hart, and Pazzani (2004). If one decides to combine several ways and let the time abstractions overlap, copying the time series and applying one way per copy solve the issue.

Definition 1.

- $S = (F, V)$ is a state, where F is a variable label and $V \in \Sigma$ is an abstraction value.
- $E = (F, V, s, e)$ is a state interval, where pair (F, V) forms a state and s and e are the start and end times of the state interval.
- $Z = \langle E_1, \ldots, E_l \rangle$ is a MSS with the states sorted according to the nondecreasing order of their start times: $E_i.s \le E_{i+1}.s$, $1 \le i \le l - 1$.[1]

Example 1. S = (HR, $\downarrow$) is a *state* that indicates that temporal variable HR is at the low level. State interval E = (HR, $\downarrow$, 12, 15) extends the state by including information about its start and end time moments. Finally, an MSS combines several state intervals coming from different time series as in MSS $Z = \langle E_1$ = (HR, –, 0, 3), E_2 = (BP, $\downarrow$, 1, 9), E_3 = (HR, $\downarrow$, 4, 7), E_4 = (HR, –, 8, 11), E_5 = (BP, –, 10, 17), E_6 = (HR, $\downarrow$, 12, 14), E_7 = (HR, $\downarrow\downarrow$, 15, 19), E_8 = (BP, $\downarrow$, 18, 22), E_9 = (HR, $\downarrow$, 20, 29), E_{10} = (BP, $\uparrow\uparrow$, 23, 26), E_{11} = (BP, $\downarrow$, 27, 31), E_{12} = (HR, –, 30, 38), and E_{13} = (BP, –, 32, 36)$\rangle$ (Figure 1).

For two state intervals E_i and E_j with $E_i.s \le E_j.s$, we say that E_i finishes before E_j if $E_i.e < E_j.s$ and denote it as $R(E_i, E_j) = b$. Otherwise, we say that E_i co-occurs with E_j and denote it as $R(E_i, E_j) = c$.

[1]If $E_i.s = E_{i+1}.s$, an order over the time variables is assumed to resolve the ambiguity.

TP is the next level of abstraction, which allows removing exact values of start and end times and focuses on temporal relationships of the state intervals.

Definition 2.

$P = (\langle S_1, \ldots, S_k \rangle, R)$ is a TP of size k ($|P| = k$) with states $S_1, \ldots, S_k$, where R is a (upper triangular) matrix describing pairwise temporal relationships between the states: $R_{i,j} \in \{b, c\}$, $1 \le i < j \le k$.[2]

Definition 3.

Given MSS $Z = \langle E_1, E_2, \ldots, E_l \rangle$ and TP $P = (\langle S_1, \ldots, S_k \rangle, R)$ ($k \le l$), we say that Z contains P, denoted as $P \in Z$, if there is a mapping $\pi : \{1, \ldots, k\} \rightarrow \{1, \ldots, l\}$ that matches every state S_i in P to a state interval $E_{\pi(i)}$ in Z such that:

1. $S_i.F = E_{\pi(i)}.F$ and $S_i.V = E_{\pi(i)}.V$, $1 \le i \le k$,
2. $\pi(i) < \pi(j)$, $i < j$,
3. $R(E_{\pi(i)}, E_{\pi(j)}) = R_{ij}$, $i < j$.

The first requirement guarantees that each state of P matches an appropriate state interval in Z, whereas the rest of the constraints enforce that the temporal relations in P correspond to a correct overlapping of the state intervals in Z.

Example 2. $P = (\langle S_1, S_2, S_3 \rangle, R)$ is a TP of Size 3 (Figure 2a) with states S_1 = (HR, –), S_2 = (BP, –), S_3 = (HR, ↓) and relationships matrix $R = (R_{1,2}, R_{1,3}, R_{2,3})$, where $R_{1,2} = c$, $R_{1,3} = b$, and $R_{2,3} = c$. The MSS in Figure 1 contains this TP because state intervals E_4 = (HR, –, 8, 11), E_5 = (BP, –, 10, 17), and E_6 = (HR, ↓, 12, 14) match the states of P and the time relationships are satisfied. For example, $E_4.e = 11 > 10 = E_5.s$, and therefore, state intervals 4 and 5 co-occur ($R(E_4, E_5) = c$).

Definition 4.

- $\tilde{P}$ is a subpattern of P, denoted as $\tilde{P} \subset P$, where $|\tilde{P}| = \tilde{k}$, $|P| = k$, $|P| = k$, and $\tilde{k} < k$,, if there is mapping $\pi: \{1, \ldots, \tilde{k}\} \rightarrow \{1, \ldots k\}$ such that:

  $\tilde{S}_i = S_{\pi(i)},\ 1 \le i \le \tilde{k}$, where $\tilde{S}_i$ and $S_{\pi(i)}$ are states in $\tilde{P}$ and P, respectively,

  $\pi(i) < \pi(i)$, $i < j$,

  $\tilde{R}_{i,j} = R_{\pi(i), \pi(i)},\ 1 \le i \le \tilde{k}$

- $\tilde{P}$ is a prefix of P, denoted as $\tilde{P} = \mathrm{prefix(P)}$, if

  $\tilde{P} \subset P$

  $\tilde{k} = k - 1$

  $\pi(i) = i + 1,\ i = 1, \ldots, \tilde{k}$

[2] $R_{i,j}$ is defined for states i and j of the pattern. $R(E_i, E_j)$ is computed for *state intervals* i and j of the MSS.

In other words, prefix(P) is a subpattern of P obtained by removing the first state of P.

Example 3. TP P in Figure 2a has three subpatterns: the subpattern without the last state (Figure 2b), the subpattern without the middle state (Figure 2c), and its prefix, the subpattern without the first state (Figure 2d).

From the previous definitions, the following corollary easily follows:

Corollary 1. *If* $\tilde{P} \subset P$ *and* $P \in Z$, *then* $\tilde{P} \in Z$.

The goal is to mine class-specific TPs that appear in a majority of MSSs belonging to a particular class. For this purpose, we use the threshold $\theta \in [0, 1]$ and define the minimum support. Assume that $D = \{Z_1 \ldots, Z_n\}$ is a data set of n MSSs and $Y = \{y_1, \ldots, y_c\}$ is a set of possible classes. Let D_i denote a set of records from D, which belong to class y_i (each record belongs to exactly one class). $Z_j \in D_i$ denotes that record j is in class y_i.

Definition 5.

- For a given TP P and class y, the support of P in class y, denoted as support(P, D_y), is defined as a number of MSSs from D_y that contain P:

$$\text{support}(P, D_y) = \left|\left\{Z \in D_y:\ P \in \mathrm{Z}\right\}\right| .$$

- P is a FTP in D if for some class y,

$$\text{support}(P, D_y) \geq \theta \times \left|D_y\right| .$$

In other words, P is an FTP in D if the proportion of MSSs containing P is not smaller than threshold θ for at least one class.

Corollary 2. *If* $\tilde{P} \subset P$ *and* $\tilde{P}$ *is* ***not*** *FTP in D, then P is* ***not*** *FTP in D.*

Corollary 2 is a straightforward consequence of Corollary 1 and Definition 5. It is referred to as the level-wise property (Aggarwal & Han, 2014). FTPM as it was given in (Batal et al., 2011) is a breadth-search procedure for finding all FTPs. First, all FTPs of Size 1 are found. Then, a list of candidate TPs of Size 2 is generated. After that, each candidate TP is validated for being an FTP and a list of FTPs of Size 2 is formed. The procedure is repeated until all FTPs are found or some stopping criteria are met; for example, size is no more than a predefined value k_{max} (see Algorithm 1). Other schemes like depth-first search are possible, but the breadth-search paradigm is important for eliminating incoherent candidate TPs as it can be seen later.

Algorithm 1

The high-level description of FTPM algorithm.

Require: D, FTPs of size 1 **return** FTPs

FTPs ← FTPs of size 1

new-FTPs ← FTPs of size 1

while |new-FTPs| > 0 and no other criteria are met **do**

candidates ← CreateCandidates(new-FTPs, FTPs of size 1)

new-FTPs ← ∅

for all $P \in$ candidates **do**

if P is FTP in D **then**

new-FTPs ← new-FTPs u{P}

end if

end for

FTPs ← FTPs ∪ new-FTPs

end while

The most computationally expensive part of this framework is validating if a candidate TP is frequent. Thus, further careful elimination of infrequent TPs at the step of creating candidates is important. Based on Corollary 2, a TP is frequent only if all of its subpatterns are frequent. For a pattern of size k, we need to verify only if subpatterns of size $k - 1$ are frequent due to transitivity.

An idea of assigning to each FTP a list of record identifiers that contain it: $P.ids = \{i : P \in Z_i\}$, reduces the search space drastically (Batal et al., 2016). It is based on the vertical data format (Zaki, 2000, 2001). Due to Corollary 1, a candidate TP of size $k + 1$ will appear only in records where all its subpatterns appear as well. Therefore, we need to check only its subpatterns of size k because record id lists of the subpatterns of smaller sizes include the list for at least one subpattern of size k. Such a list is called the *list of potential records*. For a given TP P, it is denoted as $P.p\_ids$ and can be computed as follows:

$$P.p\_ids = \bigcap_{\widetilde{P} \in \mathrm{sub}(P)} \widetilde{P}.ids = \bigcap_{\widetilde{P} \in \mathrm{sub}_k(P)} \widetilde{P}.ids,$$

where $\mathrm{sub}(P) = \left\{\widetilde{P} : \widetilde{P} \subset P\right\}$ and $\mathrm{sub}_k(P) = \left\{\widetilde{P} : \widetilde{P} \subset P \text{ and } |\widetilde{p}| = k\right\}$.

If for all classes the number of the potential records is smaller than the corresponding minimal support values, then this pattern is not frequent, and it can be discarded.

4 | FREQUENT TEMPORAL PATTERN MINING WITH EXTENDED VERTICAL LISTS

In this section, we present our approach for FTP mining. The main idea is that, for given MSS and FTP, we keep track of positions (indices of the state intervals in the MSS) where the first state of the pattern appears inside the record. We say that the pattern starts at those positions.

Assume that FTPs of all sizes 1, …, k have been found. A coherent candidate TP P ($|P| = k + 1$) constructed from FTP P_0 ($|P_0| = k$) and state S (see Batal et al, 2016 for relevant discussion) has exactly $k + 1$ subpatterns of size k ($|\text{sub}_k(P)| = k + 1$). Some subpatterns may be identical: for example, all subpatterns of Size 2 are the same for the pattern with three identical sates (HR, –), (HR, –), and (HR, –). From $\text{sub}_k(P)$, no more than k patterns (some may be identical) start with state S, and all of them are in

$$\text{sub}_k(P) \backslash \text{prefix}(P) .$$

It is straightforward to see that P cannot start at a position i inside Z if at least one of the subpatterns from $\text{sub}_k(P) \setminus \text{prefix}(P)$ does not start at the same position.

Example 4. Assume that we want to find if MSS Z (Figure 1) contains TP P (Figure 2a):

$$p = (\langle(\text{HR}, -), (\text{BP}, -), (\text{HR}, \downarrow)\rangle, R)$$

with $R_{1,2} = c$, $R_{1,3} = b$, $R_{2,3} = c$. Pattern P has two subpatterns P_1 (Figure 2b) and P_2 (Figure 2c) that have the same first state (HR, –):

$$P_1 = \left(\langle(\text{HR}, -), (\text{BP}, -)\rangle, R_{1,2} = c\right),$$

$$P_2 = \left(\langle(\text{HR}, -), (\text{HR}, \downarrow), \rangle, R_{1,2} = b\right),$$

P_1 starts at Positions 4 and 12 in Z, while P_2 starts at Positions 1 and 4. Thus, P *may potentially* start only at Position 4 where both the subpatterns start. Those positions are potential because there are also time relationships between the states that were not checked yet.

Remark 1. P_2 appears 5 times in MSS Z because states (HR, –) and (HR, ↓) of P_2 match the following pairs of the state intervals of Z: (E_1, E_3), (E_1, E_6), (E_1, E_9), (E_4, E_6), and (E_4, E_9) (in all the cases, time relationship $R_{1,2} = b$ is satisfied). However, we require the positions of the first state to be relevant only; therefore, Positions 1 and 4 are used.

One may want to store all possible appearances of P in MSS Z. However, the number of such appearances may grow rapidly (Remark 1). This requires significant memory storage. In turn, storing only starting positions of P in the MSS requires significantly less memory because the starting positions are always inside the intersection of the starting positions of the subpatterns form $\text{sub}_k(P)\backslash\text{prefix}(P)$. Therefore, the number of starting positions is a nonincreasing function of pattern size. Such a trade-off gives the desired speed-up under a reasonable memory consumption increase (Section 5).

In general, for each TP, we assign an EVL, a structure containing information on which MSSs contain the TP, starting positions of the TP inside the MSSs, and the indices of (or links to) the starting positions of the *prefix* of the TP inside the MSSs.

Definition 6.

- Let $P.EVL$ denote EVL associated with P.
- Let $P.EVL[Z].pos$ denote starting positions of P (positions of the first state of P) inside MSS Z.
- Let $P.EVL[Z].ind$ denote indices of specific starting positions of prefix(P) (positions of the first state of prefix(P)) inside MSS Z. For position $i \in P.EVL[Z].pos$, a corresponding specific index of the prefix position is the index of the smallest prefix position in Z, which is larger than i:

$$P.EVL[Z].ind[i] = \min\left\{j: \widetilde{P}.EVL[Z].pos[j] > p\right\},$$

where $\widetilde{P} = \mathrm{prefix}(P)$ and $p = P.EVL[Z].pos[i]$.

Example 5. TP $P = (\langle S_1, S_2, S_3\rangle, R)$ (Figure 2a) has states S_1 = (HR, –), S_2 = (BP, –), S_3 = (HR, ↓) and relationships matrix $R = (R_{1,2}, R_{1,3}, R_{2,3})$, where $R_{1,2} = c$, $R_{1,3} = b$, and $R_{2,3} = c$. Its prefix is P_0 = prefix(P) = (⟨(BP, –), (HR, ↓)), ($R_{1,2} = c$)) (Figure 2d). In turn, the prefix of P_0 consists of a single state: P_{00} = prefix(P_0) = prefix(prefix(P)) = (⟨(HR, ↓)⟩, ∅).

Now, for MSS Z in Figure 1, $P_{00}.EVL[Z].pos$ = {3, 6, 9} because state (HR, ↓) corresponds to the state intervals E_3, E_6, and E_9 of Z. $P_{00}EVL[Z].ind = \varnothing$ because P_{00} does not have a prefix. $P_0.EVL[Z].pos$ = {5} because state (BP, –) corresponds to the state interval E_5 of Z (Remark 1). $P_0.EVL[Z].ind$ = {2}. Finally, $P.EVL[Z].pos$ = {4}, and $P.EVL[Z].ind$ = {1}.

Now, we are ready to present the pseudo-code of the FTPMwEVL algorithm (Algorithm 2).

The EVL data structure allows to achieve three main results. First, it reduces the number of potential starting positions of a candidate TP P by intersecting the starting positions of its subpatterns from $\mathrm{sub}_k(P)$\prefix(P) as in Example 4 and later linking the potential positions to the smallest starting positions of prefix(P). Therefore, EVL reduces the number of candidate TPs to check in general (see Algorithm 3 for the pseudo-code). For some MSSs from the data set, the set of potential starting positions may be empty after the intersection meaning that these MSSs will never contain P and they can be skipped.

Second, to verify that a candidate TP P is indeed inside FTP Z, we need to check that the the states of P match the state intervals of Z and the temporal relationships are satisfied according to Definition 2. However, instead of looking through all possible combinations of appropriate state intervals, EVL allows to check a significantly smaller amount of state intervals combinations: We need to check only the possible starting locations of P, from which we can navigate directly to the appropriate state intervals matching the first state of prefix(P). But these are the already found starting positions of prefix(P); therefore, we may skip some state intervals matching the first state of prefix(P). Then, we navigate directly to prefix(prefix(P)), and so on (see Algorithm 4).

Algorithm 2

The FTPMwEVL algorithm.

```
Require: D, FTPs of size 1 return FTPs
  FTPs ← FTPs of size 1
  new-FTPs ← FTPs of size 1
  while |new-FTPs| > 0 and no other criteria are met do
    candidates ← CreateCandidates(new-FTPs, FTPs of size 1)
    new-FTPs ← ∅
    for all P ∈ candidates do
      exposure ← exposure(P)
      if not FindPotentialPositionsAndIndices(D,P) then
        continue
      end if
      for all id ∈ P.p_ids do
        new-positions ← ∅
        new_indices ← ∅
        i ← 1
        while i ≤ |P.EVL[id].pos| do
          pos ← P.EVL[id].pos[i]
          ind ← P.EVL[id].ind[i]
          index ← Search(prefix(P), D, id, ind, {pos}, exposure)
          if index > −1 then
            new_positions ← new_positions ∪ {pos}
            new_indices ← new_indices ∪ {index}
          end if
          i ← i + 1
        end while
        P.EVL[id].pos ← new_positions
        P.EVL[id].ind ← new_indices
        if P.EVL[id].pos = ∅ then
          P.p_ids ← P.p_ids \ id
        end if
      end for
      if P is FTP in D then
        new-FTPs ← new-FTPs ∪{P}
      end if
    end for
    FTPs ← FTPs ∪ new-FTPs
  end while
```

Third, EVL allows to check only a portion of the states of *P*. For this purpose, the find the smallest starting chain of *P*:

Definition 7.

- $\widetilde{P}$ is a smallest starting chain of P, denoted $\widetilde{P} = \text{chain(P)}$, if

  1. $\widetilde{k} = \min\left\{l : \nexists j > l \text{ and } i \leq l \text{ such that } R_{i,j} = c\right\}$

  2. $\widetilde{P} \subset P, |\widetilde{P}| = \widetilde{k}$

  3. $\pi(i) = i, i = 1, \ldots, \widetilde{k}$

- Exposure of P, denoted as *exposure*(P), is computed as follows:

$$\text{exposure}(P) = \begin{cases} |\text{chain}(P)| + 1 & \text{if } \text{chain}(P) \neq P, \\ |P|, & \text{otherwise}. \end{cases}$$

Algorithm 3

Function *FindPotentialPositionsandIndices*(D, P).

Require: D, P **return** Boolean

subpatterns ← $\text{sub}_k(P) \setminus \text{prefix}(P)$

if subpatterns = ∅ **then return** False

end if

$P.p\_ids = \cap_{\widetilde{P} \in sub_k(P)} \widetilde{P}.ids$

if not PotentiallyFrequent(P) **then return** False

end if

for id ∈ $P.p\_ids$ **do**

  $P.EVL[id].pos \rightarrow \cap_{\widetilde{P} \in \text{subpatterns}} \widetilde{P}.EVL[id].pos$

  $i = 1$

  while $i \leq |P.EVL[id].pos|$ **do**

    $pos = P.EVL[id].pos[i]$

    if $\{j : \text{prefix}(P).EVL[id].pos[j] > pos\} = \varnothing$ **then**

      $P.EVL[id].pos \leftarrow P.EVL[id].pos \setminus pos$

    else

      $P.EVL[id].ind[i] = \min\{j : \text{prefix}(P).EVL[id].pos[j] > pos\}$

      $i \leftarrow i + 1$

    end if

  end while

  if $P.EVL[id].pos = \varnothing$ **then**

    $P.p\_ids \leftarrow P.p\_ids \setminus id$

  end if

end for

if not PotentiallyFrequent(P) **then return** False

end if

return True

Algorithm 4

Function *Search*(*P, D, id, ind, positions, exposure*).

Require: R D, id, i, positions, exposure **return** Integer
  $i = 1$
  while $i \le |P.EVL[id].pos|$ **do**
    if Check accumulated time relationships in D[id] **then**
      if $exposure = 0$ **then return** i
      end if
      $pos \leftarrow P.EVL[id].pos[i]$
      $ind \leftarrow P.EVL[id].ind[i]$
      return $Search(\text{prefix}(P), D, id, ind, positions \cap \{pos\}, exposure - 1)$
    end if
    $i \leftarrow i + 1$
  end while
  return -1

In other words, by the smallest starting chain, we mean the smallest nonempty subpattern at the beginning of P, such that all states of it are strictly *before* the remaining states of P (see Figure 3 for an example). For many long patterns, the corresponding smallest starting chain may be relatively small.

When we check if P is inside an MSS, we need to traverse only the states of chain(P) and the first state of P_{end} if any is present because chain(P) may be pattern P itself. It is easy to see since after we have arrived at the last state of chain(P) by recursive search function (Algorithm 4) and checked that all time relationship between the states of chain(P) are satisfied, we need to verify only that all the states of P_{chain} are *before* the next state of P after chain(P) because all the states of chain(P) will be before all the rest of the states of P by transitivity. Thus, we need to check |chain(P)| + 1 first states of P, if $P \neq$ chain(P), and all $|P|$ states of P, otherwise.

5 | COMPUTATIONAL RESULTS

To evaluate the performance of the FTPMwEVL algorithm, we tested it against FTPM on real-life data sets. The TP was defined as in Definition 2 for both algorithms.

All computations were carried out on a virtual server machine with 100 GB of memory and 20 virtual cores with processor speed equivalent to 2.5 GHz each. Only one core was utilized for single-thread computations. C++11 was used as a programming language. All computation times show actual pattern mining time taken by the algorithms after any preprocessing steps such as loading data and converting it into the abstraction domain.

It is important to state that the returned TPs were entirely identical for both algorithms. It leaves computational time and memory usage as the only criteria for algorithm comparison.

For both Tables 1 and 2, the following notation was used. In column "max k," Inf stands for no upper limit on the size of TPs. In column "mem. ratio," *NA* means that the memory ratio is not available due to the fact that FTPM algorithm was not able to find all FTPs in time limit.

5.1 | Acute kidney injury data set

The acute kidney injury (AKI) data set consists of n = 5,202 medical records composed of time series taken during surgical procedures (Korenkevych et al., 2016; Thottakkara et al., 2016). Each record has an outcome associated with it: 1 if AKI was diagnosed after the surgery (2,769 records), and 0, otherwise (2,433 records).

Using the University of Florida Integrated Data Repository, we have previously assembled a single center cohort of perioperative patients by integrating multiple existing clinical and administrative databases at University of Florida Health (Korenkevych et al., 2016; Thottakkara et al., 2016). We included all patients admitted to the hospital for longer than 24 hr following any operative procedure between January 1, 2000, and November 30, 2010. This data set was integrated with the laboratory, the pharmacy, and the blood bank databases and intraoperative database (Centricity Perioperative Management and Anesthesia, General Electric Healthcare, Inc.) to create a comprehensive intraoperative database for this cohort. The study was designed and approved by the Institutional Review Board of the University of Florida and the University of Florida Privacy Office.

Two time variables were chosen for examination: mean arterial BP and HR. The value abstractions were used to convert time series from time domain to abstraction domain with percentile values [0.1, 0.25, 0.75, 0.9] and support threshold θ (see Definition 5) ranging from 0.5 to 0.9. The comparative data (see Table 1) indicates the superior performance of FTPMwEVL from the computational time point of view. For θ = 0.7, FTPMwEVL found all FTPs (there were no FTPs of size more than 18) in 39.58 s using 3,134.2 MB of memory, whereas FTPM spent 34,280.4 s and 402.31 MB to achieve the same result. Therefore, the speed-up was of magnitude **866**, whereas the new algorithm used only 7.79 times more memory. We set the computational time limit to 24 h (86,400 s). In this time frame, FTPM was able to mine all FTPs only for $\theta \geq 0.7$. For θ = 0.6, FTPMwEVL found all FTPs (no FTPS of size more than 22), yet FTPM managed to mine FTPs of Size 10 or lower and some of Size 11. In this case, FTPMwEVL took 50.25 s (not shown in the table) to mine all FTPs of Size 11 or lower, and the speed-up column reflects ratio 8,6400s/50.25s = 1,719.3. For θ = 0.5, we limited the maximum FTP size to 12 due to FTPMwEVL memory consumption considerations. Still, FTPM mined only FTPs of Size 7 or lower and some of Size 8 in 86,400 s.

As can be seen in Figure 4, the EVLs start working significantly better than the regular vertical lists with increasing FTP size, which happens due to the better indexing strategy that allows eliminating more candidate TPs and validating that a TP is not an FTP faster. Table 1 demonstrates a phenomenon of exponential growth of computational time and memory usage with decreasing threshold level that is the main limitation of this pattern mining paradigm.

5.2 | UCR time series classification archive

The remaining data sets were taken from UCR Time Series Classification Archive (Chen et al., 2015). Out of 85 data sets available, only those that have two classes were picked what resulted in 31 data sets. In this archive, each record has only one time series that was converted into two series of time-interval states using both trend and value abstractions. Percentiles [0.1, 0.25, 0.75, 0.9] for value abstractions were used to mine patterns in the UCR data sets, where, for example, all values falling between percentiles 0.1 and 0.25 were considered as low. For trend abstractions, a segment was considered increasing if the slope was positive, and nonincreasing, otherwise. The support threshold θ and maximum size k varied in ranges [0.2, 0.8] and [5, ∞], respectively, depending on the data set complexity: We pushed the memory consumption of FTPMwEVL to the limit. Therefore, Table 2 reflects the most difficult cases from FTPMwEVL memory usage point of view.

FTPMwEVL was slower only on three data sets. The most significant speed-up of magnitude 3,685.85 was achieved on data set “computers.” For this case, FTPMwEVL found all FTPs up to the predefined maximum size $k = 14$ (it was set on this level due to memory considerations) in 446.08 s having used 26,100.1 MB of memory. FTPM mined all FTPs of Size 8 or lower and some of Size 9 in the time frame of 86,400 s. Similar to the AKI data set, the speed-up column reflects ratio 86,400s/23.44s = 3,685.85, where 23.44 s is the running time of FTPMwEVL to find all FTPs of Size 9 or lower. Speed-up of 30 times or more was achieved on four other data sets: the values in bold font. However, after removing these outliers, the speed-up was on the level of 2.34 on average for the remaining data sets. The memory consumption was 4.15 time higher for FTPMwEVL on average. In the worst case, 35,566.5 MB of memory was allocated to store all FTPs, which is not a concern for modern computational clusters.

6 | CONCLUDING REMARKS AND FUTURE RESEARCH DIRECTIONS

In this paper, a new algorithm for mining FTPs was presented where the concept of EVL was utilized. It outperformed the existing approach on many real-life data sets in terms of computational time with minor exceptions. EVL requires more memory to be stored, which is a typical trade-off in such a type of algorithms. The proof of concept is that server clusters and personal computers have enough memory nowadays. Moreover, memory is becoming cheaper significantly faster than CPU, as well as the memory size becomes five times of its previous size every 2 years (see http://www.jcmit.com/), whereas CPU resources only double during the same time frame (Moore et al., 1975). Thus, the problem of using large amounts of memory is becoming less and less critical.

The speed-up was achieved due to EVL that works in two main directions: elimination of more candidate TPs and faster verification of whether a candidate TP is an FTP or not. The candidate elimination by EVL works under the assumption that if a pattern is an FTP than all, its subpatterns are FTPs as well.

For other concepts of TP like recent TP (RTP) in Batal et al. (2016), this assumption does not hold. Thus, the candidate elimination phase will not work here, and only less efficient techniques like the vertical data format should be utilized instead. Still, the concept of positions and indices will work for RTPs because the prefix of an RTP is an RTP itself. Therefore, EVLs can give a partial speed-up for Frequent RTP Mining too. The approach can be generalized and applied to other domains where the notion of pattern is defined in other ways.

Testing the applicability of the mined FTPs for classification purposes was out of the scope of this paper. Therefore, full scale evaluation of the quality of the found FTPs will be addressed in our future research. Despite the fact that other multivariate time series classification algorithms are not directly applicable here, there are several feature engineering strategies, including different distance measures to capture associations in time series, to compare against the methodology of TP ining.

Presence of noise and high-frequency sampling leads to a large number of state intervals per MSS and, therefore, to a large number of FTPs. Therefore, the effect of smoothing and noise reduction strategies is expected to effect the classification performance.

ACKNOWLEDGEMENTS

AB, PP, and PM were supported by grant R01 GM-110240 by the National Institute of General Medical Sciences – National Institutes of Health. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health.

Funding information

National Institute of General Medical Sciences, Grant/Award Number: R01 GM-110240

AUTHOR BIOGRAPHIES

Anton Kocheturov was born in Nizhny Novgorod, Russia. He received his bachelor’s degree in business informatics from the National Research University Higher School of Economics in Moscow, Russia, in 2012. He obtained his master’s degree in applied mathematics and informatics from the same university in 2014. In August 2014, he joined the PhD program at the Department of Industrial and Systems Engineering at the University of Florida. He joined Siemens Corporation Corporate Technology in Princeton, NJ, USA, as a data scientist in September 2018.

Petar Momcilovic is an Associate Professor in the Department of Industrial and Systems Engineering at Texas A&M University. His research interests are in the domain of stochastic modeling and applied probability. He received the PhD degree in Electrical Engineering from Columbia University. His research has been supported by NSF, NIH, and IBM Research.

Azra Bihorac is a Professor of medicine, surgery, and anesthesiology with career-long clinical and research interest in critical care medicine, surgery, and nephrology. She has an abiding interest in the use of rapid analytic techniques and artificial intelligence to optimize the care of the unstable patient in real-time and advance translational studies linking basic aspects of critical illness and surgical injury to clinical outcomes. She currently leads Precision and Intelligence in Medicine (PrismaP), a multidisciplinary research partnership group of experts in the field of data science, clinical informatics, and precision medicine in the College of Engineering and Medicine. The strength of this team stems from the unique proximity between the researchers in engineering, computer science, medicine, and basic science to implement advances in data science in perioperative and critical care medicine and nephrology. With the support of NIH and NSG funding, the team has developed and implemented real-time intelligent systems for predictive analytics and prognostic enrichment in perioperative medicine. The team is developing machine learning and informatics tool for real-time risk stratification and annotation of hospital-acquired complications and kidney disease. Azra Bihorac has led several large clinical trials in AKI biomarkers including the multicenter FDA validation study for the urinary AKI test and has published extensively about the epidemiology and outcomes of perioperative AKI.

Panos Pardalos is a Distinguished Professor at the University of Florida and a world renowned leader in global optimization, mathematical modeling, and data sciences. He is a Fellow of AAAS, AIMBE, and INFORMS and was awarded the 2013 Constantin Caratheodory Prize of the International Society of Global Optimization. In addition, Dr Pardalos has been awarded the 2013 EURO Gold Medal prize bestowed by the Association for European Operational Research Societies. This medal is the preeminent European award given to Operations Research (OR) professionals for “scientific contributions that stand the test of time”. Dr Pardalos has been awarded a prestigious Humboldt Research Award (2018-2019). The Humboldt Research Award is granted in recognition of a researcher’s entire achievements to date–fundamental discoveries, new theories, and insights that have had significant impact on their discipline. Dr Pardalos is also a Member of the Lithuanian Academy of Sciences, the Royal Academy of Spain, and the National Academy of Sciences of Ukraine. He is the Founding Editor of Optimization Letters and Energy Systems and Co-Founder of the International Journal of Global Optimization and Computational Management Science. He has published over 500 papers, edited/authored over 200 books, and organized over 80 conferences. He has a google h-index of 97 and has graduated 63 PhD students so far. Details can be found in www.ise.ufl.edu/pardalos.

REFERENCES

Aggarwal CC, & Han J (2014). Frequent pattern mining: Springer International Publishing, 471.

Aggarwal CC, Ta N, Wang J, Feng J, & Zaki M (2007). Xproj: A framework for projected structural clustering of xml documents. In Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, pp. 46–55.

Agrawal R, Imielinski T, & Swami A (1993a). Database mining: A performance perspective. IEEE Transactions on Knowledge and Data Engineering, 5(6), 914–925.

Agrawal R, Imieliński T, & Swami A (1993b). Mining association rules between sets of items in large databases. In ACM SIGMOD Record, 22, pp. 207–216.

Allen JF (1984). Towards a general theory of action and time. Artificial intelligence, 23(2), 123–154.

Ayres J, Flannick J, Gehrke J, & Yiu T (2002). Sequential pattern mining using a bitmap representation. In Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, pp. 429–435.

Batal I, Cooper GF, Fradkin D, Harrison J Jr., Moerchen F, & Hauskrecht M (2016). An efficient pattern mining approach for event detection in multivariate temporal data. Knowledge and Information Systems, 46(1), 115–150. [PubMed: 26752800]

Batal I, Fradkin D, Harrison J, Moerchen F, & Hauskrecht M (2012). Mining recent temporal patterns for event detection in multivariate time series data. In Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Beijing, China, pp. 280–288.

Batal I, Valizadegan H, Cooper GF, & Hauskrecht M (2011). A pattern mining approach for classifying multivariate temporal data. In 2011 IEEE International Conference on Bioinformatics and Biomedicine (BIBM), Atlanta, Georgia, pp. 358–365.

Chen F, Deng P, Wan J, Zhang D, Vasilakos AV, & Rong X (2015). Data mining for the internet of things: Literature review and challenges. International Journal of Distributed Sensor Networks, 11(8), 431047.

Chen Y, Keogh E, Hu B, Begum N, Bagnall A, Mueen A, & Batista G (2015). The UCR time series classification archive. www.cs.ucr.edu/~eamonn/time_series_data/

Deshpande M, Kuramochi M, Wale N, & Karypis G (2005). Frequent substructure-based approaches for classifying chemical compounds. IEEE Transactions on Knowledge and Data Engineering, 17(8), 1036–1050.

Hauskrecht M, Visweswaran S, Cooper GF, & Clermont G (2013). Data-driven identification of unusual clinical actions in the ICU. In Proceedings of the Annual Symposium of the American Medical Informatics Association, Washington, DC.

Keogh E, Chu S, Hart D, & Pazzani M (2004). Segmenting time series: A survey and novel approach. Data mining in time series databases, 57, 1–22.

Kocheturov A, & Pardalos P (2018). Frequent temporal pattern mining with extended lists. In Trends in Biomathematics: Modeling, Optimization and Computational Problems, Cham: Springer, pp. 233–244.

Korenkevych D, Ozrazgat-Baslanti T, Thottakkara P, Hobson CE, Pardalos P, Momcilovic P, & Bihorac A (2016). The pattern of longitudinal change in serum creatinine and 90-day mortality after major surgery. Annals of surgery, 263(6), 1219–1227. [PubMed: 26181482]

Kuramochi M, & Karypis G (2001). Frequent subgraph discovery. In ICDM 2001, Proceedings IEEE International Conference on Data Mining, 2001, IEEE, San Jose, CA, USA, USA, pp. 313–320.

Lee G, & Yun U (2017). A new efficient approach for mining uncertain frequent patterns using minimum data structure without false positives. Future Generation Computer Systems, 68, 89–110.

Moerchen F (2006). Algorithms for time series knowledge mining. In Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA: ACM, pp. 668–673.

Moore GE (1975). Progress in digital integrated electronics. Electron Devices Meeting, 21, 11–13.

Moskovitch R, & Shahar Y (2015). Classification of multivariate time series via temporal abstraction and time intervals mining. Knowledge and Information Systems, 45(1), 35–74.

Papapetrou P, Kollios G, Sclaroff S, & Gunopulos D (2009). Mining frequent arrangements of temporal intervals. Knowledge and Information Systems, 21(2), 133.

Patel D, Hsu W, & Lee ML (2008). Mining relationships among interval-based events for classification. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, New York, NY, USA: ACM, pp. 393–404.

Seeja K, & Zareapoor M (2014). Fraudminer: A novel credit card fraud detection model based on frequent itemset mining. The Scientific World Journal, 2014, Article ID 252797, 10 pages.

Srivastava J, Cooley R, Deshpande M, &Tan P-N. (2000). Web usage mining: Discovery and applications of usage patterns from web data. Acm SIGKDD Explorations Newsletter, 1(2), 12–23.

Thottakkara P, Ozrazgat-Baslanti T, Hupf BB, Rashidi P, Pardalos P, Momcilovic P, & Bihorac A (2016). Application of machine learning techniques to high-dimensional clinical data to forecast postoperative complications. PloS one, 11(5), e0155705. [PubMed: 27232332]

Tiple P, Cavique L, &Cavalheiro Marques, N. (2017). Ramex-forum: A tool for displaying and analyzing complex sequential patterns of financial products. Expert Systems, 34(1), e12174.

Tong Y, Chen L, Cheng Y, & Yu PS (2012). Mining frequent itemsets over uncertain databases. Proceedings of the VLDB Endowment, 5(11), 1650–1661.

Tsai C-W, Lai C-F, Chiang M-C, & Yang LT (2014). Data mining for internet of things: A survey. IEEE Communications Surveys and Tutorials, 16(1), 77–97.

Winarko E, & Roddick JF (2007). Armada–An algorithm for discovering richer relative temporal association rules from interval-based data. Data & Knowledge Engineering, 63(1), 76–90.

Wu S-Y, & Chen Y-L (2007). Mining nonambiguous temporal patterns for interval-based events. IEEE Transactions on Knowledge & Data Engineering, 6, 742–758.

Yun U, Lee G, & Lee K-M (2016). Efficient representative pattern mining based on weight and maximality conditions. Expert Systems, 33(5), 439–462.

Zaki MJ (2000). Scalable algorithms for association mining. IEEE Transactions on Knowledge and Data Engineering, 12(3), 372–390.

Zaki MJ (2001). Spade: An efficient algorithm for mining frequent sequences. Machine learning, 42(1), 31–60.

Zaki MJ (2005). Efficiently mining frequent trees in a forest: Algorithms and applications. IEEE Transactions on Knowledge and data Engineering, 17(8), 1021–1035.

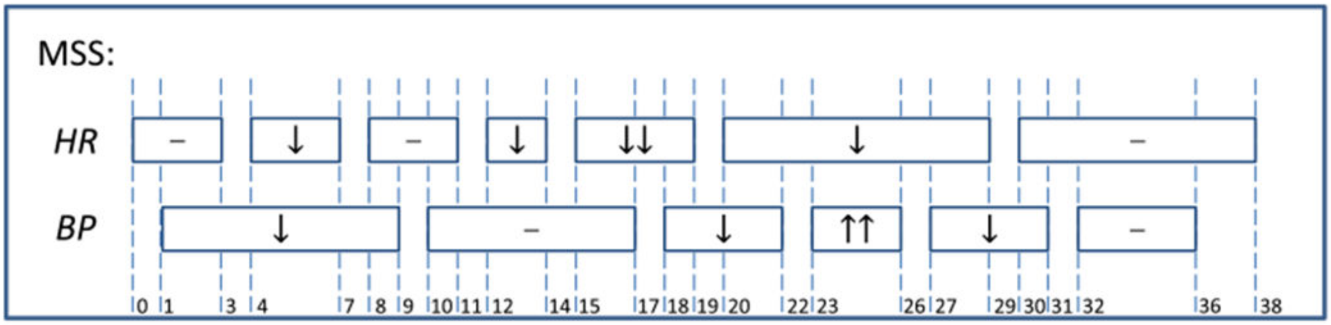

FIGURE 1.

An example of a multivariate state sequence (MSS) with time variables heart rate (HR) and blood pressure (BP)

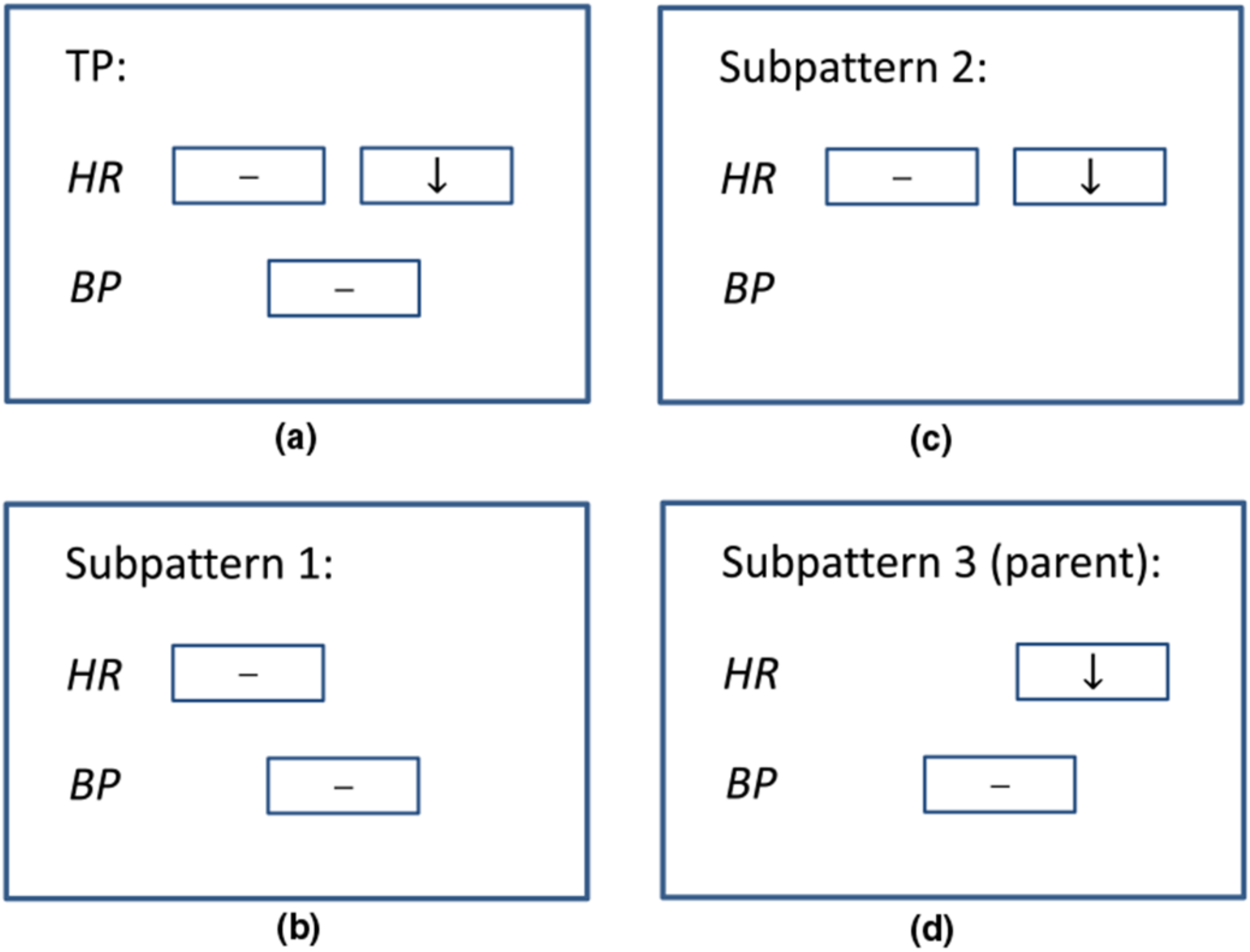

FIGURE 2.
Temporal pattern (TP) and its subpatterns: (a) an example of a TP, (b) the subpattern without the last state, (c) the subpattern without the middle state, (d) the prefix (or parent), the subpattern without the first state

Author Manuscript
Author Manuscript
Author Manuscript

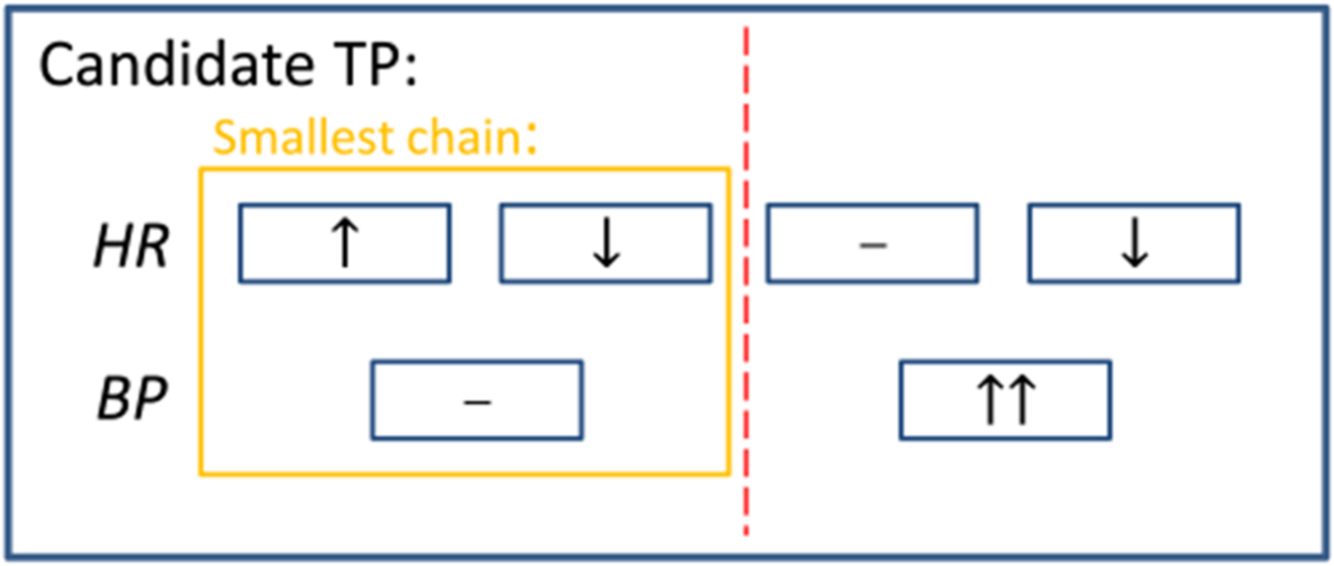

FIGURE 3.
An example of a temporal pattern (TP) and its smallest starting chain

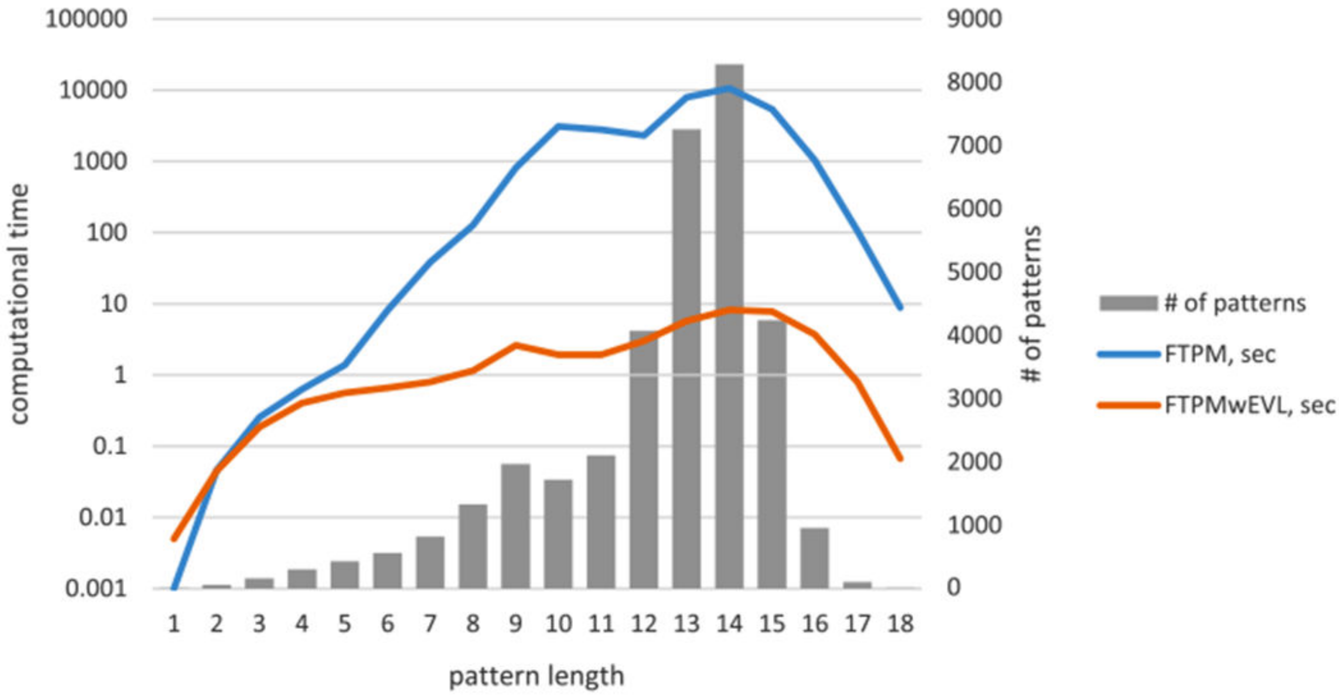

FIGURE 4.

Computational time in seconds for Fast Temporal Pattern Mining with Extended Vertical List (FTPMwEVL) and FTPM on acute kidney injury data set for mining Frequent Temporal Pattern of sizes from 1 to 18 (there were no FTPs of a size larger than 19), given that all FTPs of smaller sizes were already found. Threshold θ was set at 0.7. Total computational time was 3,4280.4 and 39.58 s for FTPM and FTPMwEVL, respectively. Memory usage was 402.31 and 3,134.2 MB. Thus, FTPMwEVL achieved a significant speed-up of magnitude **866** while consuming 7.79 times more memory

Author Manuscript
Author Manuscript
Author Manuscript

TABLE 1

Computational time comparison of FTPMwEVL and FTPM on AKI data sets

		FTPM			FTPMwEVL				
θ	**Max *k***	**k**	**Second**	**MB**	**k**	**Second**	**MB**	**Speed-up**	**Mem. ratio**
0.9	Inf	7	0.16	1.86	7	0.1	15.8	1.06	8.49
0.8	Inf	12	465.5	24.92	12	2.3	198.5	**204.27**	7.96
0.7	Inf	18	34,280.4	402.31	18	39.6	3,134.2	**866**	7.79
0.6	Inf	10	**>86400**	NA	22	621.1	35,566.1	**>1,719.3**	NA
0.5	12	7	**>86400**	NA	12	467.8	28,950.4	**>998.93**	NA

Abbreviations: AKI, acute kidney injury; FTPMwEVL, Fast Temporal Pattern Mining with Extended Vertical List; NA, not applicable.

TABLE 2

Computational time comparison of FTPMwEVL and FTPM on UCR data sets

			FTPM			FTPMwEVL				
Data set	θ	**Max** ***k***	**k**	**Second**	**MB**	**k**	**Second**	**MB**	**Speed-up**	**Mem. ratio**
BeetleFly	0.8	8	8	702.0	5,577.2	8	275.5	1,8367.7	2.55	3.29
BirdChicken	0.7	Inf	18	742.9	1,564.4	18	269.9	4,873.5	2.75	3.12
Coffee	0.8	10	10	560.1	4,108.8	10	283.5	13,107.4	1.98	3.19
Computers	0.8	14	8	**>86,400**	NA	14	446.1	26,100.1	**>3,685.8**	NA
DistalPhalanx OutlineCorrect	0.7	Inf	16	160.3	1,435.1	16	113.3	5,382.6	1.41	3.75
Earthquakes	0.8	7	7	905.4	923.7	7	220.1	14,340.2	4.11	15.52
ECG200	0.6	Inf	18	2,349.7	3,225.3	18	300.5	11,863.1	7.82	3.68
ECGFiveDays	0.5	Inf	19	336.3	1,181.4	19	226.1	3,509.4	1.49	2.97
FordA	0.8	5	5	214.9	1,462.2	5	113.4	11,068.7	1.90	7.57
FordB	0.8	5	5	125.9	891.0	5	59.9	6,461.7	2.10	7.25
Gun_Point	0.2	Inf	18	35.3	129.6	18	27.0	377.7	1.31	2.91
Ham	0.8	7	7	783.8	5,894.6	7	310.5	26,318.9	2.52	4.46
HandOutlines	0.8	12	12	87.3	2,847.4	12	134.9	9,917.5	0.65	3.48
Herring	0.8	10	10	470.7	3,186.7	10	177.9	10,216.7	2.65	3.21
ItalyPowerDemand	0.2	Inf	13	1.2	16.8	13	1.4	46.4	0.86	2.75
Lighting2	0.8	8	8	50,929.9	6,080.6	8	495.1	35,555.4	**102.87**	5.85
MiddlePhalanx OutlineCorrect	0.7	Inf	17	301.6	2,689.2	17	184.3	9,633.6	1.64	3.58
MoteStrain	0.2	Inf	20	402.5	1,109.5	20	380.6	2,316.5	1.06	2.09
PhalangesOutlines Correct	0.5	Inf	14	96.6	1,115.3	14	71.9	4,035.1	1.34	3.62
ProximalPhalanx OutlineCorrect	0.4	Inf	19	588.0	4,271.8	19	307.5	15,383.3	1.91	3.60
ShapeletSim	0.8	7	6	**>86,400**	NA	7	378.1	25,489.8	**>228.52**	NA
SonyAIBORobot Surface	0.8	10	10	1,387.0	5,255.8	10	474.6	14,650.9	2.92	2.79
SonyAIBORobot SurfaceII	0.8	10	10	2,222.3	6,619.7	10	701.0	21,730.3	3.17	3.28
Strawberry	0.7	Inf	18	200.4	1,701.7	18	105.9	6,042.2	1.89	3.55
ToeSegmenta-tion1	0.8	8	8	754.7	2,904.8	8	181.5	11,128.0	4.16	3.83
ToeSegmenta-tion2	0.8	8	8	845.9	3,076.6	8	188.2	10,987.0	4.49	3.57
TwoLeadECG	0.2	Inf	17	15.1	93.5	17	15.2	2,43.1	0.99	2.60
wafer	0.7	Inf	11	**>86,400**	NA	29	275.6	10,952.4	**>660.73**	NA
Wine	0.7	Inf	22	481.8	4,288.6	22	354.8	14,043.6	1.36	3.27
WormsTwoClass	0.8	7	7	5,459.6	2,183.7	7	148.7	11,036.3	**36.75**	5.05
yoga	0.4	Inf	16	410.3	3,593.7	16	215.6	8,825.3	1.90	2.46

Abbreviations: FTPMwEVL, Fast Temporal Pattern Mining with Extended Vertical List; NA, not applicable.