Frequent Pattern Mining from Multivariate Time Series Data

Meserret Karaca[a,*], Michelle M. Alvarado[a], Mostafa Reisi Gahrooei[a], Azra Bihorac[b], Panos M. Pardalos[a]

[a] *Department of Industrial and Systems Engineering, University of Florida, FL, USA*
[b] *Division of Nephrology, Hypertension, and Renal Transplantation, University of Florida, FL, USA*

Abstract

Discrete sequential data is the collection of an ordered series of discrete events or abstractions from a set of data points collected at different time periods. These processes are one of the most common and important process types encountered in various domains. It is customary to discover similarities and to detect the indicators of anomalies in multivariate form in a supervised setting. In this paper, we first use an effective data transformation technique that transforms multivariate time series into multivariate sequences and use a tree-based method to mine frequent patterns from multivariate time series. However, this problem is costly in terms of solution time and memory consumption. Specifically, this study aims to improve computational efficiency for memory with reasonable solution time. We demonstrate the efficiency of this algorithm on standard datasets and then apply the method to a real healthcare problem.

Keywords: frequent pattern mining, knowledge discovery, multivariate time series, discrete sequential data, electronic healthcare data

*Corresponding author.
Email addresses: mkaraca@ufl.edu (Meserret Karaca), alvarado.m@ise.ufl.edu (Michelle M. Alvarado), mreisigahrooei@ufl.edu (Mostafa Reisi Gahrooei), ABihorac@anest.ufl.edu (Azra Bihorac), pardalos@ise.ufl.edu (Panos M. Pardalos)

1. Introduction

Accelerated improvements in computing and storage technologies facilitated data-driven approaches for pattern and feature extraction from massive datasets for systems analysis and predictions. In the literature, analytical approaches have been applied to many areas of healthcare, including accurate diagnosis, prediction of diseases, and drug development (Koh et al., 2011; Raghupathi & Raghupathi, 2014; Bates et al., 2014). Nevertheless, there are still many healthcare questions waiting to be answered with better solutions. A few of these common questions include, what is the likelihood of postoperative complications?; how do postoperative complications develop?; and are there any observable signs (or factors) prior to an operation that serve as predictors of the operation outcome? In this context, a rich amount of studies examine how age, comorbidities, physical examination findings, and serum laboratory values affect postoperative complications in order to improve risk stratification prior to surgery and to allow timely use of preventive therapies during surgery and anesthesia (Merath et al., 2020). However, there is a lack of research regarding the role of preoperative quality of life (Saxton & Velanovich, 2011). Moreover, current preoperative risk stratification is limited to a physician's subjective risk assessment (Thottakkara et al., 2016). An example of preoperative risk algorithms is MySurgeryRisk (Bihorac et al., 2019), which was developed by a diverse research group called PRISMA^p at the University of Florida. This algorithm uses existing clinical data in electronic health records to forecast patient-level probabilistic risk scores for eight major postoperative complications, including Acute Kidney Injury (AKI) and sepsis. MySurgeryRisk algorithm provides analytical guidance to surgeons in risk assessment before surgery. This study aims to discover common preoperative behaviors seen in electronic medical/health records of the patients who developed AKI. These records in the database are the collection of data points during the inpatient time. Each record consists of a value of a variable at discrete time points for a patient. To set an example, the dataset in this study consists of patient IDs and patients' health records

such as heart rate and blood pressure during their preoperative inpatient term in the hospital. In terms of the data structure, existing records for a patient are in the form of multivariable time series, and the entire study dataset consists of 30 million rows (approximately 44,000 patients). Data points are recorded at irregular but frequent time intervals.

In this study, we develop an efficient (in terms of computational and space complexity) algorithm that extracts all possible common patterns among patients who developed postoperative AKI. In the literature, this problem falls into the category of Temporal Pattern Mining (TPM), which finds statistically relevant patterns in temporal data for which the instances are represented as sequences of events (Batal & San Ramon, 2015). Data mining and machine learning techniques are the most popular approaches in recent studies that consider postoperative AKI prediction. In the literature, we observe different approaches used in disease prediction such as machine learning techniques, time series analysis, or pattern mining algorithms (Karabatak & Ince, 2009; Herland et al., 2014; Siuly & Zhang, 2016; Bakator & Radosav, 2018). However, they are designed to predict an outcome and cannot extract exact patterns from a dataset. Pattern extraction approaches have not received attention in these studies mainly due to the high computational and space complexity of these approaches and the massive size and complexity of the medical datasets. (Batal & San Ramon, 2015).

Another factor that makes this problem challenging besides the dataset size is the multivariable setting of the dataset. Current methodologies achieve the benefit of dimension reduction by converting time series to time intervals with abstraction. In recent studies, frequent temporal pattern mining (FTPM) is a valuable approach for multivariate time series mining in terms of discovering a pattern of a subgroup (Zhu et al., 2011). Searching a pattern that appears in every dateset may cause other essential criteria to be overlooked. Hence, searching a pattern with a supported probability may lead to more accurate findings, especially in healthcare, which is vulnerable to exceptions and uncertainties. Although the FTPM field has gained attention over the past 10 years due to

increased data storage and improvements in computer performance (Anastasiu et al., 2014), mining patterns from multivariate time series data sets has only been around for a few years. FTPM is a broad topic in data mining literature; however, there are few studies for multi-variate time series datasets, and those are dated in 2010s (Chee et al., 2019).

FTPM from multivariate time series datasets, also known as knowledge discovery for multivariate time series, is extremely costly in terms of memory and time due to complex relations among variable levels in different time intervals. Creating a chronological order for multivariate time intervals is not possible, so a wise approach to the problem is to create temporal abstractions after the reduction in dimensionality. In data mining, temporal abstractions are used to extract frequent patterns. In order to represent patterns consisting of time intervals, Allen's thirteen temporal relations (Allen, 1990) are often used. One of the first attempts to solve this problem came from Papapetrou et al. (2005). The solution strategy used in the study considers data transformation by reducing the dimensionality where the values are converted to abstractions. The abstractions are used to describe the level of the variables as a label. Papapetrou et al. (2005) applies vertical list representation and generates possible candidates to check whether the database includes them as frequently as required. Recently published methodologies similar to Papapetrou et al. (2005) such as Batal et al. (2016) and Kocheturov et al. (2019) propose Apriori based algorithms. Apriori based algorithms use a candidate generation-and-test approach to reduce the number of candidates to examine. Unlike Papapetrou et al. (2005), they apply pruning rules to avoid redundant candidate generation. However, scanning the database multiple times to search the candidate pattern increases memory usage. Moskovitch & Shahar (2009) also uses candidate generation and test strategy with an algorithm named KarmaLego.

One of the main contributions of this paper is showing that multivariate time series can be transformed to sequential datasets where frequent sequential pattern mining techniques can be applied. This transformation also provides an opportunity to mine multivariate time series in reasonable time and with less

memory. This paper proposes a tree-based algorithm to mine frequently appearing items in a big database after transforming multivariate time sequences into sequences.

One of the most efficient sequential pattern mining algorithms, *PrefixSpan* was developed by Han et al. (2001) and is used in many areas both commercially and academically (Chen et al., 2003; Han et al., 2007; Yu & Zhou, 2010). It uses pattern growth approach by using suffix trees as opposed to Apriori based algorithms. The Apriori approach may not be efficient in mining large sequence databases having numerous patterns and/or long patterns (Han et al., 2007). In *PrefixSpan*, a sequence database is recursively projected into a set of smaller databases, and sequential patterns are grown in each projected database by exploring only locally frequent fragments. Hence, it is a useful algorithm in large sequence databases. In most cases, *PrefixSpan* outperforms the Apriori based algorithms (Pei et al., 2004). Modifications also have been applied to a variety of data mining problems (Fournier-Viger et al., 2017).

The Apriori based algorithms (Batal et al., 2016; Kocheturov et al., 2019) fall short in terms of memory usage and cannot provide a solution when minimum support is lowered for a dataset with large number of records. However, a pattern growth approach fits our problem best since the data has many data points for a patient considering recorded vitals at every minute. Hence, we present our algorithm, Multivariate Time Series Frequent Pattern Miner (MTS-FPM), which is a *PrefixSpan* based algorithm. Although multivariate time series structure is not suitable for the suffix tree-based algorithms we transform the healthcare records to a sequential database with some modifications and show that it is possible to mine multivariate time series with a sequential pattern mining approach.

Studies closely related to ours include (Mörchen & Ultsch, 2007; Chen et al., 2010, 2015b), and in terms of incision strategy and the coincidence representation they use in the preprocessing step. While Mörchen & Ultsch (2007) use CHARM by Zaki & Hsiao (2002), which is an efficient algorithm for closed itemset mining, after the preprocessing step, Chen et al. (2010) and Chen et al.

(2015b) apply a sequential pattern mining algorithm to time interval-based event data (e.g., electricity usage of household appliances) and create algorithms, as a modification of *PrefixSpan*. Results also prove that *PrefixSpan* based approaches outperform Apriori based algorithms. Different from our study, these studies do not address multivariability in full measure. In the studies above, abstractions are created assuming that there are only two levels to consider. However, our study considers different levels of a variable rather than 0-1 (on-off) levels. Also, after the modification, the studies (Mörchen & Ultsch, 2007; Chen et al., 2010, 2015b) still face the redundant candidate generation case. This paper shows that this can be avoided using a small modification in the pattern counting step.

In the rest of this paper, we discuss the background in Section 2. We then present our methodology, Multivariate Time Series - Frequent Pattern Mining (MTS-FPM), in Section 3 and show computational results in Section 4. Finally, in Chapter 5, we conclude the paper. The Appendix section covers the pseudo-codes of the algorithms mentioned throughout the paper.

2. Related Background

In this paper, we study the frequent temporal pattern mining problem for multivariate time series database and propose a modified algorithm, MTS-FPM, based on the *PrefixSpan* algorithm. In this section, we provide the necessary background for our methodology. First, Section 2.1 demonstrates how to generate abstractions from a time series and a multivariate state sequence, including abstractions that belong to different variables. Also, we explain the *PrefixSpan* algorithm in Section 2.2 since we build our methodology on the same concept.

2.1. Dimension Reduction for Multivariate Time Series

Multivariate time series include multiple variables whose values change throughout the given time horizon. Relevantly, a multivariate time series database consists of records with multivariate time series. In the AKI dataset, a data point is

Figure 1: (a) Time series for blood pressure. (b) Multivariate time series for blood pressure and heart rate. The figures shows blood pressure and heart rate values for a patient. The abstractions and the corresponding ranges defined for blood pressure variable are"Low" :[0,90], "Normal":(90,140], and "High":(140,200]; for heart rate variable are "Low" :[0,90], "Normal":(60,100], and "High":(100,200].

composed of a tuple that includes a variable, a time point, and the value of this variable at the time point. A single time series is the collection of these data points for one variable (e.g., blood pressure in Figure 1(a)). In this manner, a record is the collection of multiple time series that belong to a patient (e.g., blood pressure and heart rate in Figure 1(b)).

To represent this setting with mathematical notations, we define a database D of n records d_i $(i = 1, ..., n)$, where each record is composed of m time series $X_j^i (j \in L, L = \{l_1, ..., l_m\})$, where L is the set of variables. Mining patterns considering all of the data points in a time series is computationally costly. Therefore, first we convert continuous values in each data points to categorical values. Moreover, we also construct abstractions merging collateral data points if they belong to same level. This process yields an ordered sequence of labeled levels and the resulting reduction in each time series also depends on the smoothness level of the time series. While a massive amount of reduction can be observed in a flat time series, the reduction is not as large for highly fluctuating time series.

To mine frequent patterns in database D that contains multivariate time series, we define time-interval abstractions such as value or trend abstractions

(Shahar, 1997). For example, a **value abstraction**, in this setting, is a level estimated by given ranges of a variable in a time period while **trend abstractions** are estimated based on increments and decrements of a variable throughout a given time period. Let us consider a patient whose heart rate is recorded regularly each minute for 5 minutes and a normal heart rate falls in the interval of $[60, 100)$. Also, assume that the values measured for each minute are 80, 85, 90, 100, and 110. The abstractions for this patient are 'normal' heart rate during the first 3 minutes and a 'high' heart rate for the last 2 minutes. This is an abstraction that is adjusted to the values so it can be called a value abstraction. In Figure 1 (a), value abstractions of the blood pressure variable over time is given by the set Σ_{BP}= {"Low", "Normal", "High" }. In Figure 1 (b), an example of multivariate value abstraction is given where the set of value abstractions is $\Sigma_{HR} = \Sigma_{BP}$= {"Low", "Normal", "High"}. Next, to represent this idea symbolically, we represent required terminology and their definitions.

Definitions:

1. $\boldsymbol{E = (F, V, s, e)}$ is a **state interval** where (F, V) is a state and s and e are the start and end times of the state interval.
2. $\boldsymbol{F}$ is the variable label of state interval E where $F \in L$.
3. $\boldsymbol{\Sigma_{l_j}}$ is the set of possible abstractions of variable l_j.
4. $\boldsymbol{V}$ is the value abstraction of state interval E where $V \in \Sigma_F$.
5. $\boldsymbol{Z}$, Multivariate State Sequence (MSS), is the sequence of state intervals where the state intervals are ordered by their starting times s, $Z = \langle E_1, E_2, ..., E_k \rangle$ where $E_t = (F_t, V_t, s_t, e_t)$, $s_{t-1} \leq s_t \leq s_{t+1}$, $1 \leq t \leq k$.

The first step is to convert each record $d_i \in D$ into a set of temporal abstractions of the form of $\langle (F_1, V_1, s_1, e_1), ..., (F_t, V_t, s_t, e_t) \rangle$ where $F_t = l_j$ $(l_j \in L)$ and $V_t \in \Sigma_{l_j}$ is an abstraction from start time s_t up to end time e_t. For example, the state interval (HR," Low", 5, 12) means that the heart rate was low from

time 5 until time 12.

To represent time-interval datasets accurately, we benefit from Allen's logic (Allen, 1984) for maintaining the relationships between different abstractions. For this purpose, we use two different relations, **"before"** and **"co-occur"**. For two state intervals E_t and $E_{t'}$, $t < t'$, we say that E_t finishes **before** $E_{t'}$ if $e_t \leq s_{t'}$. Otherwise, we say that E_t **co-occurs** with $E_{t'}$.

Table 1 Notations

Parameters	
n	Number of records (Number of patients)
m	Number of time series for each record
L	Set of variables
	(Number of variables per patient)
X_j^i	Time series j for patient i
Σ_{F_t}	Set of possible abstractions for F_t
Z	Multivariate state sequence
E_t	t^{th} state interval in a MSS
F_t	Variable of t^{th} state interval in a MSS
V_t	Value of the variable in t^{th} state interval in a MSS
s_t	Starting time of t^{th} state interval in a MSS
e_t	Ending time of t^{th} state interval in a MSS

In this study each record d_i is transformed into a MSS where start and end times are removed. This approach is different from other mutivariate time series frequent pattern mining methods (Kocheturov et al., 2019; Batal et al., 2016). After defining the relationship between any two abstractions, the abstractions are sequentially represented instead of using an upper triangular matrix to define the relationship between each state interval. In this study, each E_t is demonstrated by (F_t, V_t). With this representation, the state intervals with the same variable and a value abstraction are considered as the same state interval occur-

ring multiple times in different time periods. In this sequential representation structure, while state intervals are sorted based on their starting times, the ones that co-occur are shown in a parenthesis. However, this representation is not enough to apply a tree-based algorithm. For this purpose, we need some modifications to the MSS. Next, we show how to convert a MSS into a sequence without any loss of relations.

2.2. PrefixSpan Algorithm

PrefixSpan by Han et al. (2001) is prefix-projected sequential pattern mining approach that is used to extract frequently appearing patterns in a sequential database. Let $I = \{i_1, i_2, ..., i_N\}$ be a set of all items. An **itemset** is defined as a subset of items; it may have a single item or multiple items as a subset of I. An itemset s_j is denoted as $\{x_1, x_2, ..., x_K\}$, where x_K is an **item**. Also, a **sequence** is defined as an ordered list of those itemsets. A sequence s is denoted by $\langle s_1 s_2 ... s_p \rangle$. Moreover, an item can occur at most once in an itemset of a sequence, but can occur multiple times in different itemsets of a sequence.

Definitions:

6. A sequence $\alpha = \langle s_1 s_2 ... s_p \rangle$ is called a **subsequence** of another sequence $\beta = \langle s'_1 s'_2 ... s'_q \rangle$ and β is a **supersequence** of α denoted as $\alpha \sqsubseteq \beta$ if there exist integers $1 \leq j_1 < j_2 < ... < j_p \leq q$ such that $s_1 \subseteq s'_{j_1}, s_2 \subseteq s'_{j_2}, ..., s_p \subseteq s'_{j_p}$.

7. The **support** of subsequence α in a sequence database S is the number of sequences in the database containing α. In this paper, the support of sequence α will be denoted as $support_\alpha$.

8. A subsequence α is a **frequent pattern** if $support_\alpha \geq min\_support$ where $min\_support$ is a user defined threshold to demonstrate the least frequency of a sequence.

PrefixSpan uses a pattern growth approach. First, the database is scanned once in a *counting process* to find length-1 patterns. Then, the non-frequent items

are dropped from the database and the database becomes smaller. Later, the search space is divided by creating projected databases consisting of suffixes of length-1 patterns. Next, mining each projected database, the algorithm detects frequent items in those databases separately and appends them to the prefixes found in the first step. The process repeats itself by dividing the search space based on suffixes of length-k patterns until there are no frequent items found in all projected databases. The set of sequential patterns is the collection of patterns found in the above recursive mining process. Pseudocode for *PrefixSpan* can be found in appendix as Algorithm 1. Next, we introduce our methodology, Multivariate Time Series Frequent Pattern Miner (MTS-FPM), which is a *PrefixSpan* based algorithm.

3. Methodology

In this section, we introduce our representation technique and demonstrate the transformation procedure from a MSS to a sequential database. We also present our solution algorithm, Multivariate Time Series Frequent Pattern Miner (MTS-FPM), which can be used after the data transformation.

3.1. Transforming a MSS to a Sequence

In this study, a sequence is referred to as a list of chronologically ordered items. For instance, a list of transactions made by a customer can be considered as a list of ordered items. Each transaction includes at least one item. If a, b, c are store items and $a(bc)$ is a sequence of transactions made by a customer, then we interpret this as follows: item a occurs in the first transaction, and item b and c co-occur in the second transaction.

After dimension reduction, the MSS can also be represented as a sequence. This procedure can simply be done if one state interval is not co-occuring with more than one state interval as in synchronous multivariate time series. However, for asynchronous ones where one state interval can co-occur with more than one

state intervals in different time periods, we need a smarter way to represent a multivariate time series as a sequence. First, we assign an index to each unique value abstraction showing their order among themselves (e.g., Figure 2). Thus, a state interval becomes $E_t = (F_t, V_t, k, s_t, e_t)$ where k is the order of the abstraction among each unique (F_t, V_t) pair. Then, beginning with the earliest starting time, we compare each state interval with other state intervals that begin between the starting and ending times of this state interval. If there is a co-occurrence, we represent this by placing the state intervals in parenthesis (itemset). We append this itemset as an element to the sequence. Otherwise, one occurs after another and we simply append the state interval to the sequence as an element. We continue to apply the same procedure until no abstraction is left in this sequence. However, for the next iterations, we do insert elements after checking if this element is a sub-sequence of any element in the sequence or vice versa. In the case of a 'yes' answer, we keep/insert the element with high cardinality into the sequence and do not insert/remove the smaller size element. Finally, starting and end times are removed from each state interval, and the remaining tuple F_t, V_t, k is demonstrated with $F_{t_k}^{V_t}$ for the rest of the paper. Example 3.1 demonstrates how $E_t = (F_t, V_t, s_t, e_i)$ is transformed to (F_t, V_t, k).

Example 3.1. *We would like to represent MSS Z in Figure 2 as a multivariate sequence Z^*.*

$Z = \langle E_1 = (HR, -, 0, 5),\ E_2 = (BP, ++, 1, 2),\ E_3 = (BP, -, 2, 5),$
$E_4 = (HR, \sim, 5, 6),\ E_5 = (BP, \sim, 5, 6),\ E_6 = (HR, -, 6, 7),\ E_7 = (BP, +, 7, 9)$
$E_8 = (HR, \sim, 7, 8),\ E_9 = (HR, -, 8, 19),\ E_{10} = (BP, ++, 9, 10),\ E_{11} = (BP, +, 10, 11),$
$E_{12} = (BP, ++, 11, 12),\ E_{13} = (BP, +, 12, 14),\ E_{14} = (BP, \sim, 14, 15),$
$E_{15} = (BP, ++, 15, 16),\ E_{16} = (BP, +, 16, 18),\ E_{17} = (BP, \sim, 18, 19)\rangle$
where Σ_{BP} = { *"Low", "Normal", "High", "Very High"*} = { $-$, $\sim$, $+$, $++$}
and Σ_{HR} = { *"Low", "Normal", "High"* } = { $-$, $\sim$, $+$}.

E_1 is the first state interval in Z and we compare it with E_2 and E_3 since

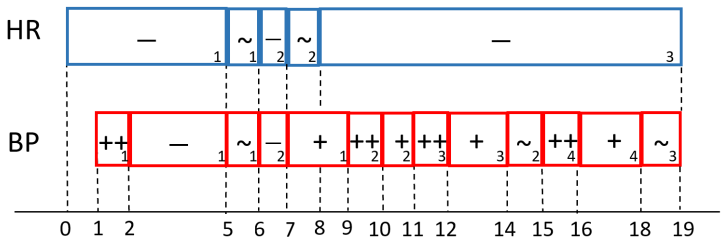


Figure 2: Enumeration of abstractions

$s_1 \leq s_2 \leq s_3 \leq e_1$. E_1 and E_2 co-occurs because $s_1 \leq s_2$ and $e_1 \geq e_2$. We add this co-occurrence to Z^* in the form of an itemset, $Z^* = \langle (E_1\, E_2) \rangle$. Next, we compare E_1 and E_3 and find another co-occurrence. Also, $(E_1\, E_3) \not\sqsubseteq (E_1\, E_2)$ or $(E_1\, E_2) \not\sqsubseteq (E_1\, E_3)$, hence we can add the itemset $(E_1\, E_3)$ to Z^* as another element. Now $Z^* = \langle (E_1,\, E_2)\, (E_1\, E_3) \rangle$. In the next iteration, we will apply same procedure for E_2. Since there is not any state interval starting between s_2 and e_2, there is not any more co-occurrence and we will check if the sub-sequence condition holds. Since, $E_2 \sqsubseteq (E_1\, E_2)$ we keep $(E_1\, E_2)$ in Z^* and do not append E_2. We continue to apply the same procedure until no state interval is left in Z and reach the following representation. Note that the MMS to sequence transformation algorithm can be found in Appendices as Algorithm 2.

$$Z^* = \langle (E_1\, E_2)\, (E_1\, E_3)\, (E_4\, E_5)\, (E_6\, E_7)\, (E_7\, E_8)\, (E_7\, E_9)\, (E_9\, E_{10})\, (E_9\, E_{11})$$
$$(E_9\, E_{12})\, (E_9\, E_{13})\, (E_9\, E_{14})\, (E_9\, E_{15})\, (E_9\, E_{16})\, (E_9\, E_{17}) \rangle$$

After deleting time information and demonstrating every state interval with variable name and the value abstraction the multivariate sequence Z^* becomes:

$$Z^* = \langle (HR_1^- \; BP_1^{++})\, (HR_1^- \; BP_1^-)\, (HR_1^\sim \; BP_1^\sim)\, (HR_2^- \; BP_2^-)\, (HR_2^\sim \; BP_1^+)$$
$$(HR_3^- \; BP_1^+)\, (HR_3^- \; BP_2^{++})\, (HR_3^- \; BP_2^+)\, (HR_3^- \; BP_3^{++})\, (HR_3^- \; BP_3^+)$$
$$(HR_3^- \; BP_2^\sim)\, (HR_3^- \; BP_4^{++})\, (HR_3^- \; BP_4^+)\, (HR_3^- \; BP_3^\sim) \rangle.$$

End **Example 3.1.**

For a database with more than two variables, the same steps can be applied with one difference: Whenever a state interval is found co-occurring with another state interval, as an extra step, we also check if there are more state intervals that co-occur with those. We only check the state intervals whose starting time is between the start and end time of the first state interval.

Example 3.2. *Consider the MSS* $Z = E_1 = (A, ++, 0, 10)$, $E_2 = (C, +, 1, 8)$, $E_3 = (B, -, 3, 6)$ $E_4 = (D, -, 7, 8)$, $E_5 = (B, \sim, 6, 12)$, $E_6 = (B, -, 12, 13)$. *Figure 3 demonstrates the example to this transformation process and Table 2 explains the progress in the sequence in each iteration.*

The final representation of this 4-variable time series as a sequential list is

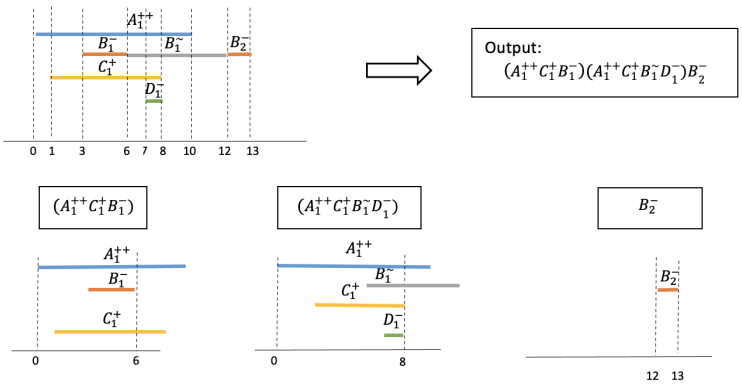


Figure 3: Example 3.2: An example of converting multivariate time series to sequential data sets

Table 2 Example 3.2: Step-by-step conversion of multivariate time series to sequential datasets where t is the index of state intervals $E_t \in Z$

$i = 1$	Step 1	$(A_1^{++}\, C_1^{+})$ found
	Step 2	$(A_1^{++}\, C_1^{+}\, B_1^{-})$ found
		$Z^* = \langle (A_1^{++}\, C_1^{+}\, B_1^{-}) \rangle$
	Step 3	$(A_1^{++}\, C_1^{+}\, B_1^{\sim})$ found
	Step 4	$(A_1^{++}\, C_1^{+}\, B_1^{\sim}\, D_1^{-})$ found
		$Z^* = \langle (A_1^{++}\, C_1^{+}\, B_1^{-})\, (A_1^{++}\, C_1^{+}\, B_1^{\sim}\, D_1^{-}) \rangle$
$i = 2$	Step 1	$(C_1^{+}\, B_1^{-})$ found but $(C_1^{+}\, B_1^{-}) \sqsubseteq (A_1^{++}\, C_1^{+}\, B_1^{-})$
	Step 2	$(C_1^{+}\, D_1^{-})$ found but $(C_1^{+}\, D_1^{-}) \sqsubseteq (A_1^{++}\, C_1^{+}\, B_1^{\sim}\, D_1^{-})$
		$Z^* = \langle (A_1^{++}\, C_1^{+}\, B_1^{-})\, (A_1^{++}\, C_1^{+}\, B_1^{\sim}\, D_1^{-}) \rangle$
$i = 3$	Step 1	no co-occurrence found and $B_1^{-} \sqsubseteq (A_1^{++}\, C_1^{+}\, B_1^{-})$
		$Z^* = \langle (A_1^{++}\, C_1^{+}\, B_1^{-})\, (A_1^{++}\, C_1^{+}\, B_1^{\sim}\, D_1^{-}) \rangle$
$i = 4$	Step 1	$(B_1^{\sim}\, D_1^{-})$ found but $(B_1^{\sim}\, D_1^{-}) \sqsubseteq (A_1^{++}\, C_1^{+}\, B_1^{\sim}\, D_1^{-})$
		$Z^* = \langle (A_1^{++}\, C_1^{+}\, B_1^{-})\, (A_1^{++}\, C_1^{+}\, B_1^{\sim}\, D_1^{-}) \rangle$
$i = 5$	Step 1	no co-occurrence found and $D_1^{-} \sqsubseteq (A_1^{++}\, C_1^{+}\, B_1^{\sim}\, D_1^{-})$
		$Z^* = \langle (A_1^{++}\, C_1^{+}\, B_1^{-})\, (A_1^{++}\, C_1^{+}\, B_1^{\sim}\, D_1^{-}) \rangle$
$i = 6$	Step 1	no co-occurrence found and B_2^{-} can be added to Z^*
		$Z^* = \langle (A_1^{++}\, C_1^{+}\, B_1^{-})\, (A_1^{++}\, C_1^{+}\, B_1^{\sim}\, D_1^{-}) B_2^{-} \rangle$

$$Z^* = \langle (A_1^{++}\, C_1^{+}\, B_1^{-})\, (A_1^{++}\, C_1^{+}\, B_1^{\sim}\, D_1^{-})\, B_2^{-} \rangle.$$

End **Example 3.2**.

This representation enables us to use a sequential frequent pattern mining technique. For this purpose, we use a *PrefixSpan* based algorithm since *PrefixSpan* is one of the efficient sequential frequent pattern mining algorithms (Han et al., 2001; Maylawati et al., 2017). So, next we revisit the *PrefixSpan* algorithm and

provide brief background information.

3.2. Multivariate Time Series Frequent Pattern Miner (MTS-FPM)

As we discussed earlier, *PrefixSpan* is a useful approach to determine sequential frequent patterns. While Apriori approaches create candidates and search the database for these candidates, it counts the existing patterns and eliminates redundant pattern search. However, it does not have any adjustments for multivariate sequences yet. We show that with only a few modifications in the format of pruning rules, it becomes eligible to mine multivariate sequences. We need the following definitions to explain MTS-FPM and the pruning rules.

Definitions:

9. The $\boldsymbol{\alpha}$ **projected database**, denoted as $\boldsymbol{S|_{\alpha}}$, is the database consisting of suffixes of α in each sequence of S and demonstrated with $\boldsymbol{S|_{\alpha}}$.

10. The $\boldsymbol{\alpha_i}$ **projected multivariate sequence**, denoted as $\boldsymbol{Z_i^*|_{\alpha_i}}$, is the suffix of α_i in sequence i where α_i is the sequence of elements in sequence i that forms current pattern α. The order of state interval k in α_i does not necessarily hold the same order in α. Therefore, we need to keep a record of the prefix sequence in each sequence as α_i.

Pruning Rules:

The MTS-FPM pruning rules are used to support the counting process (the first step of *PrefixSpan* algorithm). Frequent patterns are found based on number of sequences that include them. Recall that a subsequence is a frequent pattern if the number of sequences that includes it is equal to or larger than the minimum support. We apply these rules at the first step of MTS-FPM which can be found in the Appendix as Algorithm 3. To explain these rules, we define some variables. Let us assume that we have a current growing pattern $\alpha = \langle s_1 s_2 ... s_p \rangle$ and we are counting state interval b in $S|_{\alpha}$ for "α **is before** b" and "α **co-occurs with** b", separately. Also, assume that s_p and s_{i_p} are the last element of α and α_i, respectively. Moreover, unlike *PrefixSpan*, we consider that an item is in the format of F_k^V (e.g. BP_1^{++}). These rules are also valid for multivariable

setting with any number of variables. For the sake of simplicity and a better understanding, we give the representations and examples for a 2-variable case.

MTS-FPM Pruning Rules:

Rule 1: If b and any $c \in s_p$ have the same variable label then counting for co-occurrence is skipped.

Rule 2: If (bc) is found in $S|_{\alpha_i}$ and $c \in s_p$ then sequence i cannot be counted in support of αb (i.e. "α comes before b").

Rule 3: If $b \in s_p$, sequence i cannot be counted in support of any relationship.

Rule 4: Assume that all frequent patterns that start with item b is found. If there is a subsequence of α, α^S ($\alpha^S \sqsubseteq \alpha$) and starts with b, we can check if $\alpha^S c$ is a frequent pattern found earlier. If there is no such pattern found, then it cannot also be found in the α projected database. Therefore, under these circumstances we can skip counting αc.

These MTS-FPM pruning rules enable the algorithm to disregard redundant pattern searching. The rules can be applied in a counting process which is the first line of the *PrefixSpan* algorithm. For example, we already know that the pattern $(HR_1^{++}\ HR_1^{+})$ does not exist because heart rate cannot be 'Very High' and 'High' at the same time. So, we do not attempt to count how many times HR^{++} and HR^{+} co-occur because we already know that the answer is zero. Thus, we adapt a pruning rule (rule 1) such as eliminating the search for co-occurrence of state intervals which belong to the same variable. Another rule (rule 2) is also required if two state intervals are co-occuring but one of them is written more than once because two other state intervals are co-occuring with this state interval. As an example, consider HR_1^- in $(HR_1^-\ BP_1^{++})\,(HR_1^-\ BP_1^-)$. As a subsequent pattern, we cannot accept HR_1^- is before BP_1^- because HR_1^- and BP_1^- are co-occurring. The second rule also states that repetition of the same state interval is allowed if and only if this state interval is co-occuring in both representations, such as $(HR_1^-\ BP_1^{++})\,(HR_1^-\ BP_1^-)$. This reads as very

high and low blood pressure occur sequentially while heart rate is low. The third rule is very similar to the second rule. With a given representation, *PrefixSpan* may give a result such as $HR_1^- \, HR_1^-$ i.e., HR_1^- is before HR_1^-, or $(HR_1^- \, BP_1^{++}) \, HR_1^-$, i.e., $HR_1^- \, BP_1^{++}$ is before HR_1^-. Recall that in this study, we only define two relations ('before' and 'co-occur'), therefore, these examples are redundant because HR_1^- and HR_1^- are equivalent and $(HR_1^- \, BP_1^{++})$ already explains $(HR_1^- \, BP_1^{++}) \, HR_1^-$. On the other hand, $HR_1^- \, HR_2^-$ and $(HR_1^- \, BP_1^{++}) \, HR_2^-$ are valid frequent patterns because state intervals HR_1^- and HR_2^- belong to different time periods.

The fourth rule is developed to make MTS-FPM computationally faster than *PrefixSpan* as an independent rule from the multivariable format of the problem. MTS-FPM, like *PrefixSpan*, is a tree-based method and adapts a depth-first approach; thus, it is not fully convenient to apply the Apriori rule. While this saves us from extreme memory usage, it blocks one way to gain speed. However, we have a chance to utilize the revealed information in subtrees of previous branches. The fourth rule gives efficiency in computational time when the number of patterns to be obtained is less than the number of sequences in the database. For example, once we found all frequent patterns starting with A_1^+, we start searching for frequent patterns in the next length-1 pattern's (B_1^+) projected database. Assume that we have found a length-2 frequent pattern $B_1^+ \, A_1^+$ and now need to look for the next frequent state interval that will come after this pattern. Since patterns that start with A_1^+ are already revealed, we do not have to count the support for the state intervals that do not appear after A_1^+. Assume that we previously did not obtain a pattern such as $A_1^+ \, A_2^+$, then we can simply skip counting A^+s in the $B_1^+ \, A_1^+$ database.

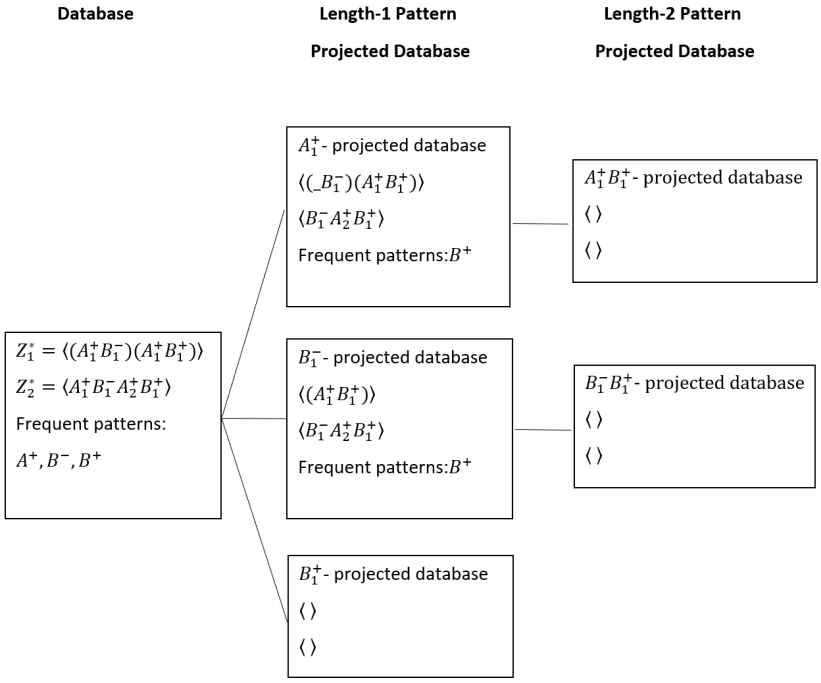
MTS-FPM (Algorithm 3 in Appendices) uses current pattern (α), current pattern length (p), projected pattern database, $(S|_\alpha)$, and set of multivariate sequences $\{\alpha_i, i = 1, ..., support_\alpha\}$. As a first step, the database is scanned and frequent items are found taking pruning rules into consideration (line 1). These items form the length-1 frequent patterns and they acquire an index $k = 1$ (line 3). First length-1 patterns are appended to α (line 7) and for each, a projected

database is created (line 8). Additionally, we keep track of the frequent items with their orders in the sequence inserting the items to α_i for each sequence $i \in \{1, ..., support_\alpha\}$ (line 10). MTS-FPM is called again for the α projected database which is the projected database generated for this length-1 frequent pattern (line 11). In the second iteration, the same search procedure is followed by applying pruning rules and α is updated appending the newly found length-1 frequent patterns to it one by one. The new length-2 patterns create child nodes on the tree. If a length-1 pattern is found as a co-occurrence with the last element in α, then it is added in a paranthesis with the last element in α; if a length-1 pattern is found in a sequential order with the last element of α, then it is added after the last element of α (line 7). Unlike *PrefixSpan*, while appending these new length-1 patterns to α, an index k is given depending on the label, abstraction value, their sequential order, and maximum index of this state interval in α (line 7). MTS-FPM is recursively called starting from the first child node until no frequent item is found. The same procedure is now applied to the next node on the tree. When all frequent patterns in all nodes are found, the algorithm terminates.

To fully understand the algorithm given above and how pruning rules can be applied, let us look at a different example where we have two sequences derived from two different multivariate time series.

Example 3.3. *Let Z_1^* and Z_2^* be two multivariate sequences where $Z_1^* = \langle (A_1^+ \, B_1^-) \, (A_1^+ \, B_1^+) \rangle$ and $Z_2^* = \langle A_1^+ \, B_1^- \, A_2^+ \, B_1^+ \rangle$. When we look at length-1 frequent patterns with min_support = 2, i.e., the length-1 subsequences that can be found in both sequences, we see A^+, B^-, and B^+ regardless of their orders. We start building α with A^+ by adding an order 1, A_1^+, as a representation of order of this element in α. It means this is the first A^+ in α. For each length-1 frequent pattern we create a projected database. These are $S|_{A_1^+}$, $S|_{B_1^-}$, and $S|_{B_1^+}$ for A_1^+, B_1^-, and B_1^+, respectively, where $S|_{A_1^+} : Z_1^*|_{A_1^+} = \langle \_B_1^-) \, (A_1^+ \, B_1^+) \rangle$, $Z_2^*|_{A_1^+} = \langle B_1^- \, A_2^+ \, B_1^+ \rangle$, $S|_{B_1^-} : Z_1^*|_{B_1^-} = \langle (A_1^+ \, B_1^+) \rangle$, $Z_2^*|_{B_1^-} = \langle A_2^+ B_1^+ \rangle$, and $S|_{B^+} : Z_1^*|_{B_1^+} =$*

Figure 4: Suffix tree representation to frequent patterns in Example 3. The frequent patterns found in the database with $min\_support$=2 : length-1 $\rightarrow$ A_1^+, B_1^-, B_1^+ , length-2 $\rightarrow$ $A_1^+ B_1^+, B_1^- B_1^+$.



$\langle\rangle$, $Z_2^*|_{B_1^+} = \langle\rangle$. *Again length-1 frequent patterns with* $min\_support = 2$ *are discovered in these projected databases and form a length-2 frequent pattern combining with the prefix. These are* A^+,B^-, *and* B^+ *for* A^+*-projected database,* $S|_{A_1^+}$. *According to PrefixSpan, now appending these new length-1 patterns found in* $S|_{A_1^+}$ *to* A_1^+ *we can form length-2 frequent patterns. However, according to rule 2, since* B_1^- *co-occurs with* A_1^+ *in* Z_1^* *and* B_1^- *comes after* A_1^+ *in* Z_2^*, *i.e.,* $support_{(A^+B^-)} = 1$ *and* $support_{A^+B^-} = 1$, *none of them can be accepted as length-2 frequent patterns. We cannot form length-2 frequent patterns by appending* A^+ *next to the previously found* A_1^+ *either because the* A_1^+ *in* $Z_1^*|_{A_1^+}$ *has the same order 1 value, meaning that they belong to the same state interval.*

However, A_2^+ in $Z_2^*|_{A_1^+}$ has order 2. In other words, there is only one occurrence of A^+ in Z_1^* while A^+ occurs twice in different time intervals in Z_2^*. Finally, We can only add B^+ to α with order 1 since this is the first B^+ in α and the length-2 frequent pattern becomes $\alpha = A_1^+ B_1^+$. The next step is to build the new $S|_\alpha$ which is $S|_{A_1^+ B_1^+} : Z_1^*|_{A_1^+ B_1^+} = \langle\rangle$, $Z_2^*|_{A_1^+ B_1^+} = \langle\rangle$. Since there is no frequent item left, α is set to $\langle\rangle$ and the same procedure is applied for other length-1 frequent patterns until no frequent pattern is found. The complete process is given in Figure 4. The frequent patterns found in the database with min_support=2 are: length-1 $\rightarrow A_1^+, B_1^-, B_1^+$, length-2 $\rightarrow A_1^+ B_1^+, B_1^- B_1^+$. End **Example 3.3**.

---

4. Experimental Results

All computations were carried out on a virtual Linux server machine, called HiperGator 2.0, with 70 GB of memory and 20 virtual cores with processor speed equivalent to 2.2 GHz each. Only one core was utilized and C++11 was used as a programming language. Solution time was limited to 24 hours.

4.1. UCR Time Series Classification Archive

MTS-FPM was also implemented on standard datasets from the University of California Riverside (UCR) Time Series Classification Archive (Chen et al., 2015a). Out of the 85 datasets available, only those that have two classes were picked which resulted in 31 datasets. In this archive, each record has only one time series which 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 datasets, where 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 non-increasing, otherwise. The minimum support and maximum size k varied in

ranges [0.2: 0.8] and [5:∞], respectfully. Table 3 compares results of MTS-FPM, FTPM by Batal et al. (2016), and FTPMwEVL by Kocheturov et al. (2019) on the cases found in (Kocheturov et al., 2019) in terms of memory usage. The settings presented in the table are adapted from (Kocheturov et al., 2019). We show that these cases are solved by MTS-FPM in reasonable time by not exhausting the memory.

Our results indicate that the memory MTS-FPM uses is significantly less than

Table 3 Computational time and memory comparison of FTPM, FTPMwEVL, and MTS-FPM on UCR datasets. k:Largest pattern obtained. max k: Maximum pattern length allowed.

dataset	min_support	max k	FTPM			FTPMwEVL			MTS-FPM		
			k	sec	MB	k	sec	MB	k	sec	MB
BeetleFly	0.8	8	8	15666	20547	8	6148	67666	8	80860	1761
BirdChicken**	0.7	Inf	18	7410	1936	18	2692	6031	18	12368	2683
Coffee	0.8	10	10	6378	15591	10	3228	49736	10	24301	1393
Computers**	0.8	14	NA	>86400	NA	NA	>25776	>70000	14	>86400	154
DistalPhalanxOutlineCorrect**	0.7	Inf	16	1667	4093	16	1178	15350	16	6594	167
Earthquakes	0.8	7	7	14266	1525	7	3468	23675	7	9751	34
FordA**	0.8	5	5	2420	2173	5	1277	16445	5	21192	30
FordB	0.8	5	5	1459	1389	5	694	10076	5	8216	21
Ham	0.8	7	7	8189	10437	7	3244	46602	7	35451	201
HandOutlines	0.8	12	12	897	20098	12	>1386	>70000	12	>86400	948
Herring	0.8	10	10	2770	6438	10	1047	20675	10	5980	258
ItalyPowerDemand	0.2	Inf	14	80	1.25	14	93	229	14	117	27
Lighting2	0.8	8	8	>86400	>11965	8	>3594	>70000	8	>86400	382
MoteStrain	0.2	Inf	20	8561	4429	20	8095	9247	20	8478	11602
ProximalPhalanxOutlineCorrect	0.4	Inf	19	4974	9358	19	2601	33700	19	11108	325

the memory usage of other methodologies. The largest memory size MTS-FPM uses is less than 11 GB while other techniques cannot solve the problem on given datasets due to the memory limitation. When we take the datasets' structure into consideration, we see that MTS-FPM performs the best compared to others in terms of memory usage for datasets where average sequence length (average number of state intervals per MSS) is larger. For datasets where average sequence length is larger than 30 (datasets highlighed with light gray), MTS-FPM spends more than 35 times less memory compared to FTPMwEVL. As an example, the datasets 'Computers' and 'Earthquakes' have average sequence

length 321 and 305. MTS-FPM's memory usage on the these datasets is more than 469 and 687 times less, respectively, compared to FTPMwEVL. Only for 'BirdChicken', where average sequence length is 38, the memory usage of MTS-FPM is 2.4 times less than FTPMwEVL. Interestingly, MTS-FPM performs very close to FTPMwEVL in terms of speed whenever the average sequence length is shorter than 30 items. As an example, the datasets 'ItalyPowerDemand' and 'MoteStrain' have an average sequence length 18 and 27, respectively. MTS-FPM performs almost the same on these datasets with speed-down coefficients 1.26 and 1.04, respectively. For those with an average sequence length less than 30, MTS-FPM spends at most 90 times less memory than FTPMwEVL.

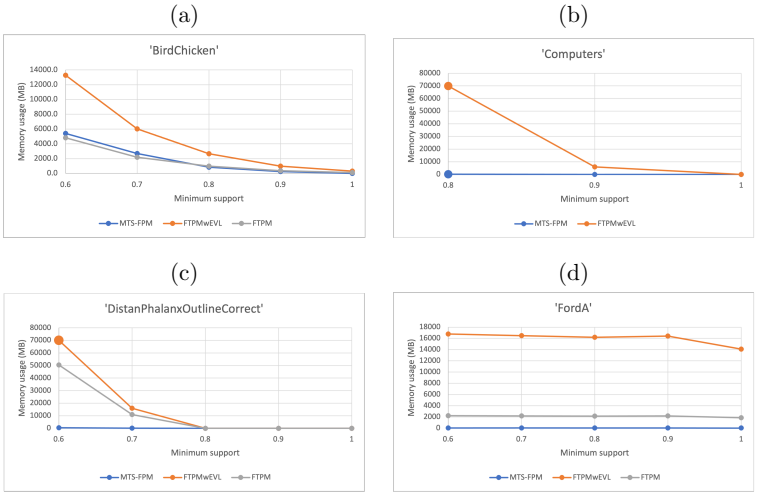


Figure 5: Memory usage comparison of MTS-FPM with existing methods FTPM and FTPMwEVL on datasets.

Figures 5 and 6 provide a visual comparison of memory usage and computational time for varying levels of minimum support for a select number of datasets. The datasets selected for the comparison are 'BirdChicken', DistalPhalanxOutlineCorrect', 'Computers', and 'FordA'. While 'BirdChicken' and

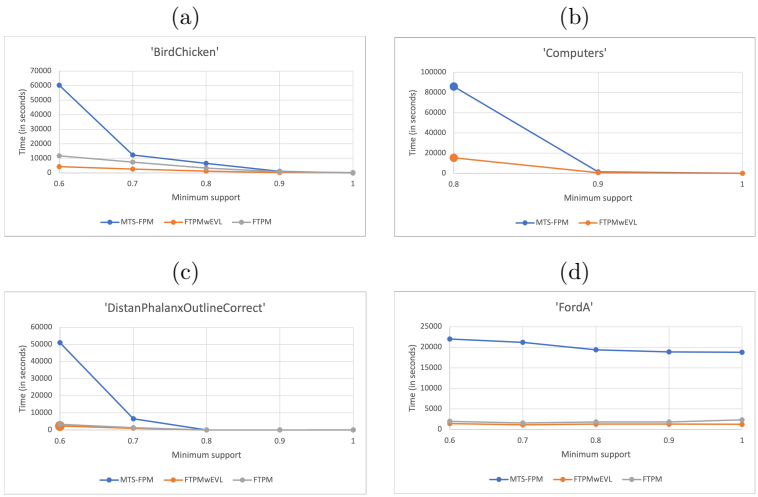


Figure 6: Computational time comparison of MTS-FPM with existing methods FTPM and FTPMwEVL on datasets.

'DistalPhalanxOutlineCorrect' have short average sequence length as 38 and 29, the sequences in 'Computers' and 'FordA' are longer with average lengths 321 and 166, repectively. Moreover, maximum pattern length allowed, $max\,k$, for the experimentation with these datasets are infinity for 'BirdChicken' and 'DistalPhalanxOutlineCorrect' due to the short average sequence length. However, $max\,k$ is selected as 10 and 5 for 'Computers' and 'FordA', respectively, since finding frequent patterns in these datasets with infinite $max\,k$ bursts the memory with FTPMwEVL and FTPM, and comparison cannot be demonstrated.

Figure 5 shows that, as minimum support decreases, memory usage in other methods is exponentially growing while there is a linear growth for MTS-FPM. In the cases where max k is infinity, other methods fail to mine frequent patterns because of excessive memory consumption, however, MTS-FPM's memory usage still follows a linear pattern. In these cases, memory usage exceeds the memory limits and the cases where memory or time usage exceeds the given limit are represented with big markers on the data points in the graphs (e.g., 5 (c),

FTPMwEVL with minimum support 0.6). For 'Computers' dataset, FTPM cannot perform under the memory limits with a minimum support less than 90%, hence, results of FTPM is discarded.

In terms of computational speed, Figure 6 shows the weakness of MTS-FPM. MTS-FPM is much slower compared to FTPM, and the disparity in computational time is magnified as minimum support decreases. For example, FTPM is more than 8.5x times faster in the 'FordA' example when minimum support is 0.8. Additionally, FTPMwEVL is 22.3x times faster than MTS-FPM in the 'DistalPhalanxOutlineCorrect' example when minimum support is 0.6. Thus, we observe the well-known trade-off between memory usage and computational speed. However, can better compare them via the speed-down coefficient and the memory efficiency coefficient. The speed-down coefficient is the ratio of how many times slower MTS-FPM is, and the memory efficiency coefficient represents how many times more efficient the memory usage is. In extreme cases the memory efficiency coefficient is always higher than the speed-down coefficient. Our computational study showed that the best memory efficiency by MTS-FPM is more than 690 times efficient than FTPMwEVL on 'Earthquake' and the worst speed-down obtained by MTS-FPM is 17 times slower than FTPMwEVL on 'FordA'. The frequent pattern mining problem does not necessarily require a fast solution technique but one that can overcome the excessive size of data. Considering the fact that even most of these datasets consist of 10-900 records, we observe that FTPM and FTPMwEVL consume allocated memory quickly. Therefore, they also fail to find frequent patterns in the AKI dataset, which has more than 25,000 patients. In the next section, we present the frequent patterns and computational metrics of MTS-FPM for the AKI dataset.

4.2. Acute Kidney Injury Dataset

This retrospective study was approved by the University of Florida (UF) Institutional Review Board and Privacy Office as exempt study with waiver for informed consent. Using the UF Health Integrated Data Repository (IDR) as Honest Broker for data de-identification, we have created a data set that

integrated multiple databases within the health system for patients admitted between January 2011 and January 2019 in University of Florida Health Medical clinics. We identified all patients who developed AKI between January 2011 and January 2019.

We identified 44,073 patients between June 1st, 2014 and March 1st 2019. Among these patients, 21.7% (9,545) were diagnosed with AKI and 78.3% (34,528) were healthy after their surgery. We used training and test cohorts given by the UF Nephrology Department. The training cohort includes patients accepted between June 1, 2014 and February 28, 2018 while the test cohort includes the patients admitted starting March 1, 2018 to March 1, 2019. A summary can be found in Table 4.

Table 4 Demographics of training and test cohorts

	AKI	healthy	total
train	7,140	26,574	33,714
test	2,405	7,954	10,359
total	9,545	34,528	44,073

For this computational study, the training cohort was used. For the heart rate variable, the range for low (HR^{-}), normal ($HR^{\sim}$), and high (HR^{+}) levels are $(0, 60)$, $[60, 100)$, and $[100, \infty)$, respectively. For systolic blood pressure variable, the range for low (BP^{-}), normal ($BP^{\sim}$), and high (BP^{+}) levels are $(0, 90)$, $[90, 140)$, and $[140, \infty)$, respectively. As an example, patterns with 70% min_support among people diagnosed with AKI is given in Table 5. Table 6 summarizes the experimental results for AKI patients in the training cohort with different min_support ranging from 40% to 100% where the longest pattern is estimated by the number of unique elements, $F_k^{V_j}$.

According to our results, the patterns seen in AKI and healthy patients' datasets show a very small difference. Only three patterns seen in AKI patients are different compared to the frequent patterns found in healthy patients and

Table 5 Frequent Patterns seen in AKI Patients with min_support 75% and the Differences with Healthy Patients

Length	Frequent patterns which exist in AKI patients' dataset
length-1	$HR_1^{\sim}$, BP_1^{-}, $BP_1^{\sim}$, BP_1^{+}
length-2	$HR_1^{\sim}\, HR_2^{\sim}$, $HR_1^{\sim}\, BP_1^{\sim}$, $(HR_1^{\sim}\, BP_1^{\sim})$, $(HR_1^{\sim}\, BP_1^{+})$, $(BP_1^{-}\, BP_1^{\sim})$, $BP_1^{\sim}\, HR_1^{\sim}$, $BP_1^{\sim}\, BP_1^{-}$, $BP_1^{\sim}\, BP_2^{\sim}$, $BP_1^{\sim}\, BP_1^{+}$, $BP_1^{+}\, BP_1^{\sim}$,
length-3	$HR_1^{\sim}\, (HR_2^{\sim}\, BP_1^{\sim})$, $BP_1^{-}\, BP_1^{\sim}\, BP_2^{\sim}$, $BP_1^{\sim}(HR_1^{\sim}\, BP_2^{\sim})$, $BP_1^{\sim}\, BP_1^{-}\, BP_2^{\sim}$, $BP_1^{\sim}\, BP_2^{\sim}\, BP_3^{\sim}$, $BP_1^{\sim}\, BP_1^{+}\, BP_2^{\sim}$, $BP_1^{+}\, BP_1^{\sim}\, BP_2^{\sim}$
length-4	$BP_1^{\sim}\, BP_1^{-}\, BP_2^{\sim}\, BP_3^{\sim}$, $BP_1^{\sim}\, BP_2^{\sim}\, BP_3^{\sim}\, BP_4^{\sim}$
	Frequent patterns which exist in AKI but not in healthy patients' dataset
length-3	$BP_1^{-}\, BP_1^{\sim}\, BP_2^{\sim}$, $BP_1^{+}\, BP_1^{\sim}\, BP_2^{\sim}$
length-4	$BP_1^{\sim}\, BP_1^{-}\, BP_2^{\sim}\, BP_3^{\sim}$
	Frequent patterns which exist in healthy but not in AKI patients' dataset
length-3	$HR_1^{\sim}\, HR_2^{\sim}\, HR_3^{\sim}$, $(HR_1^{\sim}\, BP_1^{\sim})\, HR_2^{\sim}$, $HR_1^{\sim}\, BP_1^{\sim}\, BP_2^{\sim}$, $(HR_1^{\sim}\, BP_1^{\sim})\, BP_2^{\sim}$,

Table 6 MTS-FPM results on AKI dataset for min_support varying 50% to 100%

min_support	AKI Patients					Healthy Patients				
	Number of patients	Number of Patterns	Longest Pattern	Solution Time (sec)	Memory Usage (MB)	Number of patients	Number of Patterns	Longest Pattern	Solution Time (sec)	Memory Usage (MB)
100%	7140	0	0	4	1.08	26574	1	1	25	1.08
95%	6783	2	1	12	1.08	25246	3	2	35	1.08
90%	6426	4	2	12	1.08	23917	4	2	38	1.08
85%	6069	6	3	16	1.08	22588	7	3	65	79.68
80%	5712	16	4	38	1.50	21260	11	3	93	121.01
75%	5355	23	4	51	1.50	19931	24	4	147	131.46
70%	4998	41	5	85	45.22	18602	32	4	198	136.88
65%	4641	74	6	143	63.08	17274	53	5	336	160.68
60%	4284	139	6	252	65.05	15945	92	5	468	173.06
55%	3927	265	7	456	70.04	14616	160	6	742	183.88
50%	3570	524	8	865	77.82	13287	282	6	1190	191.32

those include abstractions with low and high blood pressure. Whereas, frequent patterns such as repeatedly healthy rate ($HR_1^{\sim}\, HR_2^{\sim}\, HR_3^{\sim}$) cannot be seen in 75% of AKI patients. For a better estimation, a comprehensive experimental study including laboratory/test results and more than two vitals should be conducted. For the purpose of this paper, we only demonstrate the results in terms of computational metrics by using only two variables.

5. Conclusions

The focus of this paper is mining frequent patterns from big time series databases. While existing methodologies improved the speed factor, the memory consumption factor was neglected. Mining frequent patterns requires usage of a significant amount of memory. In this paper, we showed that it is possible to mine frequent patterns from a big time series database by using a smaller amount of memory in an acceptable amount of time. By experimenting on databases with varying sequence lengths, we revealed that, for short sequence databases (shorter than 30 items as an average approximately), a tree-based method can mine frequent patterns in a shorter time than Apriori based methods. Moreover, a tree-based algorithm always uses the least amount of memory regardless of the length of sequences.

Frequent pattern mining has broad applications such as clustering and classifications. Frequent patterns are tools to provide pattern-centered insights for a variety of problems. In this study, we considered a healthcare case for classification purpose and extracted frequently appearing patterns considering the vital records of the postoperational AKI patients. In this case, frequent patterns also can be used to detect an indicator that notifies of a coming unwanted outcome earlier and helps classify patients with a predicted outcome.

This study picked two relations from Allen's temporal relations to conduct a comparison analysis with existing methodologies. This research can be extended considering more than two relations which requires more memory consumption than the two relation case. In this manner, MTS-FPM will perform better while other methods struggle with high memory consumption rates. We believe that MTS-FPM builds a solid foundation for future studies where more complex relations are discussed.

Acknowledgements

This work was supported by the National Institute of General Medical Sciences under R01 GM110240. The content is solely the responsibility of the authors and does not necessarily represent the official views of the National Institutes of Health. The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript. We also wish to thank Anton Kocheturov for sharing his research resources and materials for the comparative study.

References

Allen, J. F. (1984). Towards a general theory of action and time. *Artificial Intelligence*, *23*, 123–154.

Allen, J. F. (1990). Maintaining knowledge about temporal intervals. In *Readings in Qualitative Reasoning about Physical Systems* (pp. 361–372). Elsevier.

Anastasiu, D. C., Iverson, J., Smith, S., & Karypis, G. (2014). Big data frequent pattern mining. In *Frequent Pattern Mining* (pp. 225–259). Springer.

Bakator, M., & Radosav, D. (2018). Deep learning and medical diagnosis: A review of literature. *Multimodal Technologies and Interaction*, *2*, 47.

Batal, I., Cooper, G. F., Fradkin, D., Harrison, J., Moerchen, F., & Hauskrecht, M. (2016). An efficient pattern mining approach for event detection in multivariate temporal data. *Knowledge and Information Systems*, *46*, 115–150.

Batal, I., & San Ramon, C. (2015). Temporal data mining for healthcare data.

Bates, D. W., Saria, S., Ohno-Machado, L., Shah, A., & Escobar, G. (2014). Big data in health care: using analytics to identify and manage high-risk and high-cost patients. *Health Affairs*, *33*, 1123–1131.

Bihorac, A., Ozrazgat-Baslanti, T., Ebadi, A., Motaei, A., Madkour, M., Pardalos, P. M., Lipori, G., Hogan, W. R., Efron, P. A., Moore, F. et al. (2019). Mysurgeryrisk: development and validation of a machine-learning risk algorithm for major complications and death after surgery. *Annals of surgery*, *269*, 652.

Chee, C.-H., Jaafar, J., Aziz, I. A., Hasan, M. H., & Yeoh, W. (2019). Algorithms for frequent itemset mining: a literature review. *Artificial Intelligence Review*, *52*, 2603–2621.

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

Chen, Y.-C., Jiang, J.-C., Peng, W.-C., & Lee, S.-Y. (2010). An efficient algorithm for mining time interval-based patterns in large database. In *Proceedings of the 19th ACM International Conference on Information and Knowledge Mnagement* (pp. 49–58). ACM.

Chen, Y.-C., Peng, W.-C., & Lee, S.-Y. (2015b). Mining temporal patterns in time interval-based data. *IEEE Transactions on Knowledge and Data Engineering*, *27*, 3318–3331.

Chen, Y.-L., Chiang, M.-C., & Ko, M.-T. (2003). Discovering time-interval sequential patterns in sequence databases. *Expert Systems with Applications*, *25*, 343–354.

Fournier-Viger, P., Lin, J. C.-W., Kiran, R. U., Koh, Y. S., & Tfhomas, R. (2017). A survey of sequential pattern mining. *Data Science and Pattern Recognition*, *1*, 54–77.

Han, J., Cheng, H., Xin, D., & Yan, X. (2007). Frequent pattern mining: current status and future directions. *Data Mining and Kknowledge Discovery*, *15*, 55–86.

Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., & Hsu, M. (2001). Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 17th International Conference on Data Engineering* (pp. 215–224). Citeseer.

Herland, M., Khoshgoftaar, T. M., & Wald, R. (2014). A review of data mining using big data in health informatics. *Journal of Big data*, *1*, 1–35.

Karabatak, M., & Ince, M. C. (2009). An expert system for detection of breast cancer based on association rules and neural network. *Expert systems with Applications*, *36*, 3465–3469.

Kocheturov, A., Momcilovic, P., Bihorac, A., & Pardalos, P. M. (2019). Extended vertical lists for temporal pattern mining from multivariate time series. *Expert Systems*, *36*, e12448.

Koh, H. C., Tan, G. et al. (2011). Data mining applications in healthcare. *Journal of Healthcare Information Management*, *19*, 65.

Maylawati, D. S., Irfan, M., & Zulfikar, W. B. (2017). Comparison between bide, prefixspan, and trulegrowth for mining of indonesian text. In *Journal of Physics: Conference Series* (p. 012067). IOP Publishing volume 801.

Merath, K., Hyer, J. M., Mehta, R., Farooq, A., Bagante, F., Sahara, K., Tsilimigras, D. I., Beal, E., Paredes, A. Z., Wu, L. et al. (2020). Use of machine learning for prediction of patient risk of postoperative complications after liver, pancreatic, and colorectal surgery. *Journal of Gastrointestinal Surgery*, *24*, 1843–1851.

Mörchen, F., & Ultsch, A. (2007). Efficient mining of understandable patterns from multivariate interval time series. *Data Mining and Knowledge Discovery*, *15*, 181–215.

Moskovitch, R., & Shahar, Y. (2009). Medical temporal-knowledge discovery via temporal abstraction. In *AMIA Annual Symposium Proceedings* (p. 452). American Medical Informatics Association volume 2009.

Papapetrou, P., Kollios, G., Sclaroff, S., & Gunopulos, D. (2005). Discovering frequent arrangements of temporal intervals. In *Fifth IEEE International Conference on Data Mining (ICDM'05)* (pp. 8–pp). IEEE.

Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., & Hsu, M.-C. (2004). Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, *16*, 1424–1440.

Raghupathi, W., & Raghupathi, V. (2014). Big data analytics in healthcare: promise and potential. *Health Information Science and Systems*, *2*, 3.

Saxton, A., & Velanovich, V. (2011). Preoperative frailty and quality of life as predictors of postoperative complications. *Annals of Surgery*, *253*, 1223–1229.

Shahar, Y. (1997). A framework for knowledge-based temporal abstraction. *Artificial intelligence*, *90*, 79–133.

Siuly, S., & Zhang, Y. (2016). Medical big data: neurological diseases diagnosis through medical data analysis. *Data Science and Engineering*, *1*, 54–64.

Thottakkara, P., Ozrazgat-Baslanti, T., Hupf, B. B., 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*.

Yu, K.-M., & Zhou, J. (2010). Parallel tid-based frequent pattern mining algorithm on a pc cluster and grid computing system. *Expert Systems with Applications*, *37*, 2486–2494.

Zaki, M. J., & Hsiao, C.-J. (2002). Charm: An efficient algorithm for closed itemset mining. In *Proceedings of the 2002 SIAM International Conference on Data Mining* (pp. 457–473). SIAM.

Zhu, X., Deng, H., & Chen, Z. (2011). A brief review on frequent pattern mining. In *2011 3rd International Workshop on Intelligent Systems and Applications* (pp. 1–4). IEEE.

Appendices

Algorithm 1 - *PrefixSpan*

Input: A sequence database S, and the minimum support threshold $min\_support$.

Output: The complete set of sequential patterns.

Method: Call $PrefixSpan(\langle\rangle, 0, S)$

Subroutine: $PrefixSpan(\alpha, p, S|_\alpha)$

Parameters: 1) α is a sequential pattern; 2) p is the length of α; and 3) $S|_\alpha$ is the α-projected database if $\alpha \neq \langle\rangle$, otherwise, it is the sequence database S.

Method:

1 Counting Process: Scan $S|_\alpha$ once, find each frequent item, b, such that b can be assembled to the last element of α to form a sequential pattern; or (b) can be appended to α to form a sequential pattern.

2 For each frequent item b, append it to α to form a sequential pattern α', and output α'.

3 For each α', construct α'-projected database $S|_{\alpha'}$, call $PrefixSpan(\alpha', p + 1, S|_{\alpha'})$.

Algorithm 2 - Transforming a MSS to a sequence

Input: MSS Z and size of the MSS, $I = |Z|$.

Output: Multivariate sequence

Method: Call $f(1, \langle\rangle)$

Subroutine: $f(i, Z^*)$

Parameters: 1) i is the index of state intervals in Z; 2) Z^* is multivariate sequence.

Method:

$k \leftarrow i$, $\hat{E} \leftarrow \{E_i\}$
while $s_k \leq s_{k+1} < e_k$**:**
    $\hat{E} \leftarrow \hat{E} \cup E_{k+1}$, $k \leftarrow k + 1$
if $k = i$**:**
    $Z^* \leftarrow Z^* \sqcup \hat{E}$
else:
    if $\hat{E} \sqsubseteq z$ *or* $\hat{E} \sqsupseteq z$, $\exists z \in Z^*$ *is False*:
        $Z^* \leftarrow Z^* \sqcup (\hat{E})$
    else:
        if $|\hat{E}| > |z|$**:**
            $Z^* \leftarrow (Z^* \setminus z) \sqcup (\hat{E})$
$i \leftarrow i + 1$
if $i < I$**:**
    Call $f(i, Z^*)$
else:
    Terminate

Algorithm 3 - MTS-FPM

Input: A sequence database S, the minimum support threshold $min\_support$, and the number of records n.

Parameters: 1) α is a sequential pattern; 2) p is the length of α; 3) $S|\alpha$ is the α projected database if $\alpha \neq \langle\rangle$, otherwise, it is the sequence database S 4) α_i elements of α with their indices in sequence i

Other Variables: 1) K is the largest indice of $F_k^{V_j}$ in α where $F = l_j \in L$, $V_j \in \Sigma_{l_j}$;

Method: Call MTS-FPM($\langle\rangle, 0, S, \{\}$)

Subroutine: MTS-FPM($\alpha, p, S|_\alpha, \{\alpha_i, i = 1, .., support_\alpha\}$)

Method:

Counting Process: Scan $S|_\alpha$ once, find each frequent item F^{V_j} by applying pruning rules (see Part I Section 2.4).

for *each* F^{V_j} **:**

  if F^{V_j} *is not in* α **:**

    $K = 0$

  else:

    $K = k$, where k for F^{V_j} ($F_k^{V_j}$) is largest in α.

  F^{V_j} can be assembled to the last element of α in the form of $F_{K+1}^{V_j}$ to form a sequential pattern α'; or ($F_{K+1}^{V_j}$) can be appended to α to form a sequential pattern α'.

For each α', construct α'-projected database $S|_{\alpha'}$.

for $i \in \{1, ..., support_{\alpha'}\}$**:**

  Update α_i appending F^{V_j} found in i^{th} sequence, output α_i' .

Call MTS-FPM($\alpha', p + 1, S|_{\alpha'}, \{\alpha_i', i = 1, .., support_{\alpha'}\}$).

Meserret Karaca

https://orcid.org/0000-0002-2851-907X

No public information available.

Record last modified Jan 12, 2021 9:14:54 PM

Highlights

- Mining frequent patterns from multivariate time series requires high memory usage
- Multivariate time series can be converted to sequences reducing the dimensionality
- Thus, sequential methods can be performed on transformed multivariate time series
- The modified *PrefixSpan* method always outperforms Apriori based methods

Meserret Karaca: Conceptualization, Methodology, Coding, Experimentation, Writing- Original draft preparation.

Michelle M. Alvarado: Supervision, Investigation, Validation, Writing- Reviewing and Editing.

Mostafa Reisi Gahrooei: Supervision, Investigation, Validation.

Azra Bihorac: Conceptualization.

Panos M. Pardalos*:* Supervision.

Declaration of interests

☒ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: