IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 4 November 2022, accepted 2 December 2022, date of publication 12 December 2022,
date of current version 20 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3228962

RESEARCH ARTICLE

A Scalable Analytical Framework for Complex Event Episode Mining With Various Domains Applications

JERRY C. C. TSENG[1], SUN-YUAN HSIEH[1], (Fellow, IEEE), AND VINCENT S. TSENG[2], (Fellow, IEEE)

[1]Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan City 701, Taiwan
[2]Department of Computer Science, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan

Corresponding author: Vincent S. Tseng (vtseng@cs.nycu.edu.tw)

This work was supported in part by the National Science and Technology Council, Taiwan, under Grant 110-2221-E-A49-078-MY3 and Grant 111-2221-E-A49-124-MY3.

ABSTRACT With the ubiquity of sensor networks and smart devices that continuously collect data, we face the challenge of analyzing the growing stream of data in real time. In recent years, there has been a huge need to gain useful knowledge by incrementally analyzing event sequence data. Although episode pattern mining techniques have existed for years, people have recently become more aware of their practical value in solving real-life domain problems such as manufacturing records, stock markets, and weather forecasts. The effective and efficient application of episode pattern mining techniques to analyze complex event data is becoming increasingly important for solving real-life problems in wide domains. However, few studies have focused on developing a scalable framework based on episode pattern mining of complex event sequences for applications in various domains. In this work, we propose a novel framework named *SAAF (Scalable Analytical Application Framework)* based on complex event episode mining techniques, including batch episode mining, delta episode mining, incremental episode mining, and pattern merging, to consider both efficiency and accuracy. Moreover, to enhance scalability, we adopt the lambda architecture with *Apache Spark* and *Apache Spark Streaming* as the system development framework. Finally, the experimental results on three real datasets of different domains and two benchmark datasets showed that the proposed *SAAF* framework exhibits excellent performance in terms of efficiency, accuracy, and scalability.

INDEX TERMS Complex event sequence, data stream, episode pattern mining, incremental mining, lambda architecture.

I. INTRODUCTION

Owing to the ubiquity of sensor networks and smart devices, various types of streaming data are continuously collected, and there are growing real-life requirements to gain valuable information by analyzing these data streams. It is very common for different sources of data to be collected simultaneously as multiple time-series events, which we refer to as complex event sequences [1]. Among data mining techniques, episode mining [2], [3], [4] is a very powerful tool that can be applied to discover useful patterns from past events to foresee the possible events that will occur in the future. There are many real-life cases around us utilizing this technique, such as manufacturing improvement [5], [6], network attack detection [7], biomedical data analysis [8], high-utility pattern mining [9], news events [10], and stock trend analysis [11], [12], [13].

Mostly, people want to have the most current information from the episode patterns mined from the new growing data instead of the past information learned from historical data only. To have the most current and valuable information, full mining of the entire data is the most intuitive and easiest

The associate editor coordinating the review of this manuscript and approving it for publication was Vlad Diaconita.

way. However, it is not cost-effective; thus, a number of incremental mining approaches have arisen to address this subject. A significant portion of existing incremental mining research arches focused on the mining of association rules [14], [15], [16] or sequential patterns [17], [18], [19] in static incremental data such as traditional transaction databases. According to our surveys, there exist still very few studies have focused on developing a scalable analytical framework for complex event episode mining for cross-disciplinary applications.

The above-mentioned situations motivated us to develop an analytical application framework to address incremental episode-mining problems for complex event sequences. Efficiency and accuracy are both crucial factors of an analytical system designed for incremental mining; they need to be considered simultaneously. Regarding the scalability of the system, we adopt the lambda architecture [20] as the system architecture, and *Apache Spark* [21], [22], [23] and *Apache Spark Streaming* [22], [24], [25], [26] as the development tools of the batch layer and the speed layer, respectively. *Apache Spark* is a unified analytics tools for large-scale data processing, and *Spark Streaming* brings *Apache Spark*'s language-integrated API to stream processing, letting us write streaming jobs the same way we write batch jobs. Otherwise, it is easier to get started and better cost-efficient comparing with other similar tools. The above features are important for developing the proposed framework, so we chose them as our development tools.

In this study, we develop a framework named *SAAF* (Scalable Analytical Application Framework) by extending our previous works [27], [28]. Compared to previous studies, *SAAF* is the first framework specially designed for various domain applications with intelligence and scalability. We designed various new modules to enhance the processing capacity of streaming complex event sequences and to improve the system architecture. We also developed a user interface that allows people to use this framework more easily.

Specifically, the main improvements and differences with the previous works are described as the following:

First, we design a new mechanism of the data pipeline controller module to intelligently balance the data flow for the speed and batch layers of the lambda architecture. This design leads to a better processing performance than the previous designs.

Second, we design a new module, namely, rule access module, such that application developers can easily obtain the most updated analytic results to help domain experts retrieve valuable information in a timely manner.

Third, we design an user interface to make it easier to accelerate the tasks of episode mining process for solving real-world problems.

Another contribution of this paper is that we conducted extended experiments on three datasets of different domains and two benchmark datasets to validate the proposed framework *SAAF*. The results showed that it delivers excellent performance in terms of efficiency, accuracy, and scalability.

The remainder of this paper is organized as follows. We provide a brief description of the problem definitions in Section II and review related works in Section III. The design of the proposed system is described in Section IV, and experimental validation and results are presented in Section V. The conclusions are presented in Section VI, and future work is introduced in this section as well.

II. PROBLEM DEFINITIONS

In this subsection, we adopt the notations used in [29] for problem definitions and properties related to this study. For more details about episode mining, readers can refer to [2], [3], [4], [29], [30], [31], [32], and [33].

Definition 1(Simple Event Sequence): This is a sequence of events in which every event occurs at its respective time point. Given a set E of *event* types, a pair (e, t) represents an *event* where $e \in E$ is an event type and t is the time point at which e occurs. An event sequence S on E is a triple (s, T_s, T_e), where $s = < (e_1, t_1), (e_2, t_2), \ldots, (e_n, t_n) >$ is an ordered sequence of events such that $A_i \in E$ for all $i = 1, \ldots, n$ and $t_i \leq t_j$ for all $1 \leq i \leq j \leq n$. Furthermore, T_s and T_e are the starting and ending times of S respectively, and $T_s \leq t_i \leq T_e$ for all $i = 1 \ldots n$.

Definition 2 (Simple Episode): A simple episode α is a non-empty totally ordered set of events of the form $< (E_1), (E_2), \ldots, (E_k) >$, where event E_i appears before event E_j for all $1 \leq i < j \leq k$.

Definition 3 (Simultaneous Event Set A): simultaneous event set $SE = (E_1, E_2, \ldots, E_m)$ is composed of a set of events, where each event $E_i \in \varepsilon$ in SE occurs at the same time point t for all $1 \leq i \leq m$. The length of the SE is denoted by $|SE|$ and is equal to the number of events in SE. Given two simultaneous event sets $SE_1 = (E_1, E_2, \ldots, E_n)$ and $SE_2 = (E'_1, E'_2, \ldots, E'_m)$, where m ≤ n, SE_2 is the subset of SE_1 and SE_1 is the superset of SE_2 iff $SE_2 \subseteq SE_1$.

Definition 4. (Complex Event Sequence): A complex event sequence $CS = < (SE_1, T_1), (SE_2, T_2), \ldots, (SE_n, T_n) >$ is an ordered sequence of simultaneous event sets, where each simultaneous event set SE_i is linked to a time point $T_i \in N^+$ and $T_i < T_j$ for all $1 \leq i < j \leq n$.

Definition 5 (Episode with simultaneous events): An episode *with simultaneous events* is an ordered tuple of simultaneous event sets with the form $< SE_1, SE_2, \ldots, SE_k >$; SE_i appears before SE_j for all i, j $(1 \leq i < j \leq k)$. Based on this definition, there are two different relationships among episodes: simultaneous and serial relationships. The length of α is denoted by $|f|$, and is equal to the number of events of f.

Definition 6 (Prefix of Episode): Given an episode $f = < SE_1, SE_2, \ldots, SE_k >$, its sub-episode $< SE_1, SE_2, \ldots, SE_{m-1}, SE'_m >$ $(m \leq k)$ is called *the prefix* of f if SE'_m is a sub-simultaneous event set of SE_m and the events in $(SE_m - SE'_m)$ are alphabetically after those in SE'_m.

Definition 7 (Support of an Episode): Given an event sequence S, *the support of an episode* f is denoted by $sup(f)$,

and it is formally defined as $sup(f) = |sc(f)|/N$, where N is the number of sliding windows of S and $sc(f)$ is the support count of f.

Definition 8 (Frequent Episode): Given a user-specified minimum support threshold *minsup*, episode f is said to be frequent if $sup(f)$ is no less than *minsup*. *Minimum support count* is given as the product of *minsup* and N, where N is the size of S.

The aim of this work is to develop a scalable framework that mine frequent episode patterns for various domain applications considering both accuracy and efficiency.

III. RELATED WORK

Frequent episode mining has been a widely used technique in data mining and knowledge discovery since it was first introduced by Mannila et al. [4]. It was used to mine frequent subsequences as episodes from time-series data, which is referred to as the event sequence. Each event was linked to the time point that it occurred, and they can be presented as a pair (e, t), where stands for that event e occurred at time t. Fig. 1 is an illustration of a single-event sequence. e_1, e_2, e_3 and e_4 are different events that occur at a specific time. For example, pair (e_4, t_4) indicates that event e_4 occurred at time point t_4. Frequent episode mining helps us identify patterns known as episodes that are partially ordered sets of events.

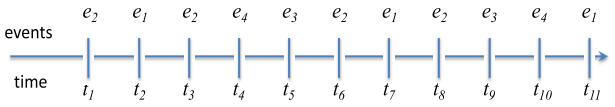

FIGURE 1. An illustration of a single-event sequence.

Frequent episode mining has been applied in different kinds of domain applications and to analyze various forms of data like time-stamped fault reports in sales transactions [9], financial stock data [12], [34], power usage [27], power plant facility maintenance [28] telecommunication network alarms [35], healthcare [36], sports data [37] and user location prediction [38], to name a few. In recent years, episode mining technique has been utilized in more domain applications, such like recommender systems [39], workload prediction [40], [41], high utility mining [29], [42].

Cheung et al. proposed the algorithm *FUP* [14] and extended *FUP* to FUP_2 [15]. *FUP* and FUP_2 are both renowned incremental mining algorithms based on the Apriori mechanism to update the association rules when new transaction records are inserted or removed from the database. Ayan et al. proposed an algorithm *UWEP* [43] with the concept of negative borders to enhance the efficiency of *FUP*-based algorithms. The above algorithms are not practical to be used in the environment of continuous streaming data, because of the following problems, namely, a). high possibility of large set of candidate itemsets; b). the need for multiple scans in a database.

Hung and Chang proposed the algorithm *EMMA* [31] for mining frequent episodes from complex sequences, but it only works for static datasets, not for streaming data. Wu et al. proposed *UP-Span* [9] for mining high utility episodes in complex event sequences. The restriction of *UP-Span* is that it targeted for mining high utility episodes, so that might lose other important information in sequences.

The pattern mining technique of data streams has become more and more welcome in recent years. Manku proposed a lossy counting approach for approximate frequency counting over streams with no assumptions regarding the stream [44]. Patnaik et al. presented an approach for mining frequent episodes over a window of recent events in a stream [45]. These approaches may work properly for a single data stream, but they are not appropriate and applicable for the complex event sequence analytic process; therefore, they are restricted in solving real-world problems around us.

Recently Ao et al. proposed a big data algorithm for episode mining called *LA-FRMH* [46], but this algorithm did not handle the case of simultaneous events. Ouarem et al. proposed an algorithm called *NONEPI* [47] for episode rule mining using the concept of non-overlapping frequency. The primary goal of *NONEPI* is to find rules that are easier to interpret as occurrences must be non-overlapping, but this algorithm did not consider complex event sequences that are very common in real-world problems.

Lin et al. proposed the algorithm *POERM* [48], and then extended it as *POERMH* [49]. Those two methods focused on finding partially-ordered episode patterns, which mean a small set of rules instead of numerous episode rules. However, it is not practical for various kinds of real-world applications without finding all rules for users.

Guyet et al. proposed the algorithm *Re-DPFE* [50], which focused on privately frequent episode mining over event streams. However, *Re-DPFE* was not designed for solving public problems of various domains and it is not scalable for big data problems. Guyet et al. presented the incremental mining algorithm IncSeq [51], which was based on counting the minimal occurrences of the sequential patterns over the course of itemsets stream. However, it cannot be scalable for big data applications either.

However, most existing studies on episode mining have focused on mining a single event sequence and are not applicable to mining complex sequences. In our previous work [27], we proposed an analytical system named *CEAS (Complex Event Analytical System)* to aid people to accelerate the process of episode pattern mining for complex event sequences, and designed an *EM-CES (Episode Mining over Complex Event Sequence)* for episode pattern mining over complex event sequences. In *CEAS*, we took *SAX* [52] as the symbolic transformation method and extended the episode mining algorithms of *WINEPI* [4] and *MINEPI* [2] to be integrated into an analytical system for multivariate complex event sequences. In another previous work [28], we proposed a new analytical system named *SICEM (Scalable Incremental Complex Event Mining)*, that adapted the lambda architecture [53] designed to process massive data by taking advantage of both batch-processing and streaming-processing methods.

In *SICEM* we developed a series of modules within the four components and designed three algorithms: batch episode mining, delta episode mining, and pattern merging. Table 1 is a comparison table of related methods in terms of pros. and cons.

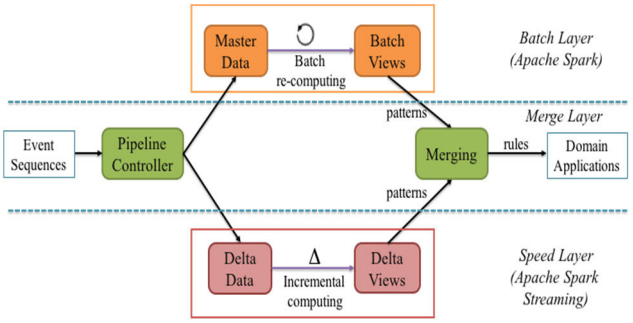

FIGURE 2. The conceptual model of proposed analytical framework.

IV. THE PROPOSED FRAMEWORK

Fig. 2 shows the conceptual model of our newly proposed analytical application framework, denoted as *SAAF*, based on complex event episode mining. The input data here are complex event sequences, and most of them are continuous streaming data from various types of data sources, for example, digital sensors or intelligent meters. The raw streaming data collected from data sources must be appropriately pre-processed before being dispatched to the batch and speed layer processes. The batch layer has a mechanism for periodic re-computing, which is a type of re-mining for all data, to maintain the accuracy of the mining result.

Between every two time points of full-mining, stream processing and incremental computing of the speed layer can help speed up mining efficiency. The merge layer works for consolidating the results of the batch views and incremental views to have the most current episode patterns with good accuracy. The final results of episode patterns or rules are output in JSON format to be accessed and applicable by other applications.

In our proposed framework, the source data of complex event sequence are injected through *Apache Kafka* [54]. *Kafka* is one of the most widely adopted tool to accomplish the tasks of capturing, retaining, and processing overwhelmingly rapid flow of information. Then, the input streaming data is kept in the *HDFS* [55], which is well integrated in *Apache Spark* environment and designed to store large datasets reliably to ensure the overall data integrity of completeness and consistency. As discussed in [20], our design of Lambda architecture provides advantages in data consistency and fault tolerance to guarantee the data safety and information trust.

The limitation of the proposed framework lies in that it is based on *Apache Spark* and *Apache Streaming* environments, thus it will take some efforts for the administrator to set and tune the run-time environment for complying with the requirements of *Apache Spark* and *Apache Streaming*.

TABLE 1. Comparison of related methods.

Algorithms	Pros. And cons.
FUP /FUP$_2$	The algorithms are based on the Apriori mechanism to update the rules when new transaction records are inserted or removed from the database, but they are not applicable to the continuous streaming data.
UWEP	It is designed with the concept of negative borders to enhance the efficiency of FUP-based algorithms, but it is not suitable for the continuous streaming data.
EMMA	It is designed for mining frequent episodes from complex sequences, but it only works for static dataset rather than streaming data.
U-Span	It is a high utility mining episode algorithm from complex event sequences, but it may lose other important information that is valuable for users.
LA-FRMH	It is a big data analytical algorithm for episode mining, but it did not handle the case of simultaneous events.
NONEPI	It is designed to find rules that are easier to interpret considering that occurrences must be non-overlapping. However, it does not work for complex event sequences in real-world applications.
POERM/ POERMH	It is designed to find partially-ordered episode patterns, which mean a small set of rules instead of numerous episode rules. However, it is not practical for various kinds of real-world application without finding all rules for users.
Re-DPFE	This algorithm focuses on privately frequent episode mining over event streams, but it is not designed for solving public problems of various domains.
IncSeq	It is based on counting the minimal occurrences of the sequential patterns over the course of itemsets stream, but it cannot be scalable for big data applications.
EM-CES	It is an episode mining algorithm for complex event sequence, but it does not consider the need for incremental mining as the data grows.
TEM-SES	This algorithm focuses on mining frequent target episode from complex event sequences, but it is does not work for the mining within streaming data.
SAAF	This is the algorithm we proposed in this work, which is based on complex event episode mining techniques, including batch and delta episode mining, to meet both of efficiency and accuracy.

TABLE 2. Segment set of the complex event sequence.

SID	Segment
1	<(A,1), (E,1), (D,2), (B,3), (F,3), (A,4), (C,4)>
2	<(B,5), (F,5), (C,6), (A,7), (E,7), (B,8), (F,8)>
3	<(B,9), <D,9), (A,10), (C,10), (D,11), (G,11), (A,12), (C,12)>
4	<(C,13), (E,13), (F,14), (B,15), (C,15), (C,16)>

A. MODULES

In this study, we developed a framework named *SAAF* by extending our previous works [27], [28]. Fig. 3 shows the system architecture of the proposed framework *SAAF*. It shows that there are 16 modules in the *SAAF*, respectively, eight modules in the pre-processing layer, two modules in the batch layer, two modules in the speed layer, two modules in the merge layer, and two independent modules. The following are the descriptions of these modules.

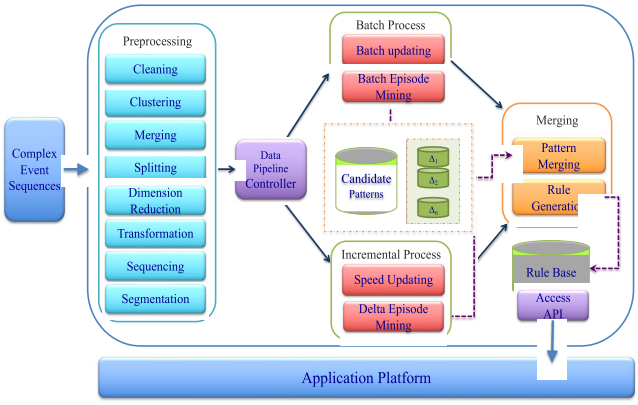

FIGURE 3. System modules of proposed analytical framework SAAF.

1) MODULES IN PRE-PROCESSING LAYER

a. Cleaning: The main goal of this module is to clean the noise in the data and convert the data into a format adequate for further mining processes. Different devices often generate various data in diverse data format of different frequencies, so we must preprocess the new input data properly and then save them as the transformed database for the next step of the pattern mining process. As for numeric data, they need to be transformed to a symbolic presentation, or we cannot pass the data to the mining phase.

b. Clustering: Event attributes are labeled according to event groups that user specified. The data input to this module consists of a table with categorical event attributes and user-specified event groups. The output is a recoded table.

c. Merging: Merge the input data and sort the records in ascending order. The input of this module consists of several tables with the same attributes, including a timestamp attribute. The output is a merged table sorted by its timestamp.

d. Splitting: Splitting the table into several subtables based on event attributes. Because events in different groups may be irrelevant in the intended applications, we must separate the records belong to different event groups. The input of this module is a table with event attributes and the output includes several tables, each representing an event group.

e. Dimension reduction: Reduce the dimensions of the table. Similar to table splitting, some specific attributes that are determined irrelevant to the event groups are removed from the table of a specified event group. The input of this module is a table that stands for an event group and the attribute sets that will be retained for the event group. The output was a reduced table.

f. Transformation: Discretize numeric variables into categorical ones using *SAX* [52]. It first calculates the average value and the variance value for each numeric variable, and then divides the scope of the variable into several lumps with the same probability, according to the properties of the normal distribution. The input of this module is a table with numeric variables, and the output is a transformed table. The parameters for each numeric variable consist of a number N, which is the number of output symbols after the transformation, and a valid range, out of which the values are not considered and are transformed into other predefined symbols, for example, outliers.

g. Sequencing: Transform the table into sequence form. Each record in the input table was transformed into an itemset. The input of this module is a table with a timestamp, and the output is the transformed complex event sequence.

h. Segmentation: Set segmentation flags into a complex event sequence for episode mining in the batch and speed layers. The user provides criteria for the segmentation, for example, 15 min after a specified event occurs. The segmentation symbols are then inserted into the input long sequence. The input of this module is a complex event sequence, and the output is a sequence with segmentation symbols inserted.

Because most of the real-world datasets are highly susceptible to be missing, inconsistent and noisy due to their heterogeneous origin, we design the above eight modules to make the raw data ready for the next mining steps. These modules can be roughly decomposed as the tasks of data cleaning, reduction, integration, and transformation. All these preprocessing steps are important in different aspects: Cleaning has high impact on the correctness on the mining results, while the other steps dealing with the data reduction, integration and transformation will affect the efficiency of the mining execution.

2) MODULES IN BATCH LAYER

a. Batch updating: Fetch the data stored in the speed layer into the batch layer and reset the storage of the speed layer. This ensures that whenever the batch layer works, the most updated data are contained within batch episode mining. The input of this module is the data storage of the speed layer, which is merged into the batch layer storage as the output.
b. Batch episode mining: Mine episode patterns in batches. The input of this module is a complex event sequence with segmentation symbols stored in batch layer storage. The output is episode patterns with support. The parameter of this module is the minimum support that user specified.

3) MODULES IN MERGE LAYER

a. Pattern merging: Merge the respective episode patterns from the batch episode mining module and those from the delta episode mining module, then find the patterns for the whole. This algorithm is described in the next section. The input of this module is the patterns with their support from the batch and speed layers, and the output is the merged pattern.
b. Rule generation: Rules are generated from merged patterns. The default rule generation mechanism is to split a k-pattern (k itemsets in the pattern) into a (k-1)-prefix pattern and a 1-postfix pattern, for example, a pattern $<A, D, C>$ will bring a rule $<A, D> \rightarrow <C>$. Then, calculate the confidence of the rule using the ratio of the supports of the k-pattern and the (k-1)-prefix pattern. Rules with low confidence, that is, confidence less than a user-specified minimum confidence, will be discarded.

In this module, the input is the merged patterns, and the output is the rule set presented in JSON format for further applications. A simple output sample is illustrated in Fig. 4. A rule is a set of elements, and each rule is composed of four parts: *LHS*, *RHS*, *sup* and *conf*, where *LHS* and *RHS* represent the segments of the rule, with support and confidence. For example, the first rule in Fig. 4 is $<A, D> \rightarrow <C>$, whose support is 0.55 and confidence is 0.75. The second rule in Fig. 4 is $<(B,C),(A,C,E),(D)> \rightarrow <A,E>$, whose support is 0.60 and confidence is 0.9.

4) DATA PIPELINE CONTROLLER

This is a module to intelligently balance the data flow for the speed and batch layers of the lambda architecture. as the experimental result shows, this module leads to a better processing performance than the previous designs.

In the lambda architecture, there are two independents but highly cooperative layers, namely, the batch process layer and incremental process layer. The processes within the two layers work independently; however, their respective results are well merged as the final output. The input data of the proposed system are complex event sequences used as streaming data. We observed that the system sometimes could not work robustly because of data loss during analytical processing. We learned that data dispatching is also a key factor in the system performance in terms of efficiency and effectiveness. To make the proposed system more robust and have wider use for diverse domain applications, we designed a data pipeline controller for users to dynamically adjust the data dispatching mechanism. In this module, users can control the frequency and scope of data dispatching to batch and incremental layers, respectively. The major benefit is that this module can work as an asynchronous buffer to balance the data input flow and processing capacity, thus avoiding data loss during the mining process.


```
{
 "RULES": [
  {
   "LHS": ["(A)", "(D)"],
   "RHS": ["(C)"],
   "sup": 0.55,
   "conf": 0.75
  },
  {
   "LHS": ["(B,C)", "(A,C,E)", "(D)"],
   "RHS": ["(A,E)"],
   "sup": 0.60,
   "conf": 0.9
  },
  …
 ]
}
```


FIGURE 4. A simple sample of rules presented in JSON format.

5) RULE ACCESS INTERFACE

This is a module to facilitate application developers to easily obtain the most updated analytic results and help domain experts retrieve valuable information in a timely manner.

For the above purpose, to allow users to use the analytical results more easily, we developed a series of application programming interfaces. Users can acquire frequent episode patterns by calling the APIs to obtain the results in JSON format, and then integrate those patterns with their applications or other information systems.

B. ALGORITHMS

The algorithms of the proposed framework are composed of three parts as listed in the following:

a. *BatchEpisodeMining* (batch layer)
b. *DeltaEpisodeMining* (speed layer)
c. *PatternMerging* (merge layer).

In batch episode mining, the patterns of the entire data are found. Because the cost of every time the batch layer execution is very high, the speed layer should be fast enough to meet the response time requirement of the users for real-time queries. Thus, delta-episode mining finds delta patterns from the new incoming data in the speed layer. Because the delta patterns cannot represent the behaviors of the total data,

we merged the patterns from the batch and speed layers to find the proper pattern set of the patterns of the total data.

We used *PrefixSpan* [56] as the basic mining approach and extended it to the MapReduce [57] framework. As discussed in [58], *PrefixSpan* outperforms other *Aprori*-like algorithms and can be extended to mining sequential patterns with user-specified constraints for various domain applications. In [59], Kijsanayothin et al. stated that MapReduce is a programming paradigm that enables parallel and distributed execution of massive data processing on large clusters of machines, and thus researchers can focus on building efficient algorithms to enhance performance.

1) BATCH EPISODE MINING

For batch episode mining, we used the *PrefixSpan* [56] approach with MapReduce. Both sliding-window based and minimum-occurrence based approaches cannot only consider the events appear around the "key event," which is highlight by Segmentation module. Before the mining process, the batch-updating module fetches the data from the speed-layer storage. This means that all data at this point are collected in batch layer storage. Then, the speed layer storage is reset to collect new incoming data. In batch episode mining, we first compute the support of each 1-item episode using a pair of Map and Reduce functions, where the count refers to the number of segments with different segmentation symbols and is the super-sequence of the counting 1-item episode. The 1-episode whose support is less than the user-specified minimum support is discarded as an unpromising episode. Then, we choose one of the promising episodes as a prefix and generate its projected complex event sequence, that is, discard the items located in segments without the prefix, or appear before the prefix. Then, we can take the projected complex event sequence as input to recursively repeat the counting step by MapReduce functions and find the episode patterns with the assigned prefix. After taking all the *1*-item episode pattern as prefixes, all episode patterns were found.

To check whether an episode is a pattern or not, there are two approaches to determine the support of an episode: 1) the number of sliding windows that contain the episode and 2) the number of minimal occurrences. According to our study, sliding window is basic and more useful, because for most real-world applications, users would like to know 'what's going on' within a period. Most existing episode mining algorithms with sliding windows adopt an *Apriori*-like approach that generates candidates and then calculates the support count. This takes too much time and does not fit the streaming environment and multivariate complex event sequences.

Because the *Aprori*-like approach does not meet our requirement owing to its low efficiency and re-scanning, we referred to the *PrefixSpan* approach and extended the approach to make it work well for episode pattern mining in complex event sequences. The basic steps are as follows.

1. Get the segment set from the complex event sequence by sliding window;
2. Set $k = 1$ and *pref* =<>;
3. Find the episode patterns that length equal to k from the segment set;
4. Generate the projected segment set for each pattern found in the previous step;
5. Set $k = k+ 1$ and go to step 2 until no more patterns found.

First, we cropped copies of the complex event sequence using a sliding window to obtain the set of complex event segments, that is, the subsequences of the complex event sequence that are included in the sliding windows. Before mining the patterns, we initialize the length parameter k as 1 and the prefix episode *pref* as an empty sequence.

Then, we count the support of each event to find the patterns by extending *pref* with each frequent event from the segment set and generate the projected segment set for each pattern with length k. Then, the same process is repeated with $k = k+ 1$. If no episode patterns are found in this step, we choose another *prefix* episode and mine with its projected segment set.

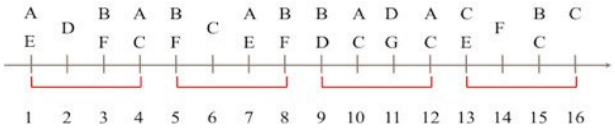

FIGURE 5. An example of a complex event sequence.

For example, the complex event sequence is illustrated in Fig. 5, and the sliding window was set to a size of 4 and sliding distance of 4. Without loss of generality, we assume that the events appearing within the same timestamp are sorted in alphabetical order. First, we obtained four segments using the sliding window, which are listed in Table 2. If *minsup* is 30%, {<*A*>, <*B*>, <*C*>, <*D*>, <*E*>, <*F* >} are episode patterns with $k = 1$. After generating their projected segment sets, < *A* > is chosen as the next *prefix* episode, and its projected segment set is presented in Table 3.

TABLE 3. Segment set of pref =<A>.

SID	Segment
1	<(_,1), (E,1), (D,2), (B,3), (F,3), (A,4), (C,4)>
2	<(_,7), (E,7), (B,8), (F,8)>
3	<(_,10), (C,10), (D,11), (A,12), (C,12)>

The underlined event is the first end of *pref* in the segment. After counting the support, we can find {<*B*>, <*C*>, <*D*>, <*F*>, <_*C*>, <_*E*>} are frequent episodes in <A>'s projected segment set, and represent the episode patterns as {<*A, B*>, <*A, C*>, <*A, D*>, <*A, F*>, <(*A, C*)>, <(*A, E*) >}. Then, we generate their projected segment sets from < *A* >'s projected segment set, choose <*A, B* > as the new pref*ix* episode, and proceed with $k = 3$. After all episode

patterns with $< A >$ as the first event are found, the process chooses a new *prefix* episode from the remaining episodes to mine the patterns until all patterns are found.

After the frequent episode patterns are mined from the complex event sequence, rules are generated from the patterns with at least two complex events to make predictions with new streaming data. A rule can be presented in the form of $LHS \rightarrow RHS$, which means that a complex event *RHS* may occur after a sequence of events *LHS*. A rule can be evaluated with its confidence, which is defined as $conf(LHS \rightarrow RHS) = sup(<LHS, RHS>) / sup(LHS)$, where $<LHS, RHS>$ is a sequence that contains only *LHS* and *RHS* and *LHS* occurs before *RHS*. For example, the confidence of rule $<A,E> \rightarrow < F >$ is equal to $sup(< A, E >) / sup(< F >)$. With a user-specified minimum confidence *minconf*, we can only retain the rules whose confidence is no less than *minconf* in the rule pool.

TABLE 4. An illustration of the rule pool.

No.	LHS	RHS	Conf.	Sup.
1	<A>	<B>	66.7%	3
2	<(A, E)>	<F>	100%	2
3	<C>	<B>	66.7%	3

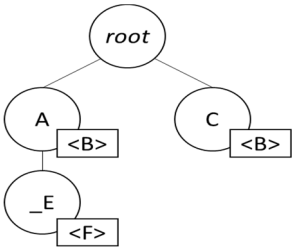

FIGURE 6. An illustration of the rule tree.

For example, there are three rules in the rule pool listed in Table 4. The straightforward approach loads these rules in the order (2, 1, 3), sorted by the keys, and the tree-based approach builds a rule tree, as shown in Fig. 6. Subsequently, a new segment $< (A, 12), (E, 13), (C, 13)>$ is obtained. The straightforward approach finds the first matched rule, which is rule No. 2, and predicts $< F >$. The tree-based approach reaches the end nodes of all three rules and predicts $< B >$ after voting.

2) DELTA EPISODE MINING

For Delta episode mining, the mining process is similar to batch episode mining. First, the speed layer storage, which collects the new incoming data, that is, the data never been mined by batch episode mining, is reset to the initial state (empty or a partial segment) based on the criteria defined in the segmentation module. Whenever new data come from the pre-processed data stream, they are fetched by the speed updating module and appended to the speed layer storage for Delta episode mining. In delta episode mining, a *PrefixSpan*-based approach with MapReduce for mining delta patterns was used. Basically, the main flow of the algorithm is like batch episode mining, but whenever it finds an episode, whose support is less than the user-specified minimum support, it has to check whether the episode is an episode pattern in batch episode mining. If so, the episode is still considered a promising pattern. After delta episode mining is performed, delta episode patterns are found. The pseudocode of *DeltaEpisodeMining* is shown in Fig. 7.

DeltaEpisodeMining(*S*, *minsup*, *pref*)
Input: Complex event sequence with segmentation symbols *S*, Minimum support threshold *minsup*, Prefix episode *pref*
Output: Frequent episode patterns *P*

```
1   SegmentRDD ← S.map(SegmentSpilit)
    //SegmentSplit() is a map function to split the input sequence by
    segmentation symbols
2   itemFlatRDD ← SegmentRDD.flatMap(ItemFlat)
    //ItemFlat() is a flatMap function to flat items in segments and
    remove repetitions
3   itemPairRDD ← itemFlatRDD.mapToPair(toPair)
    //toPair() is a mapToPair function to make items as (item, count)
4   pairs
    CountRDD ← itemPairRDD.reduceByKey(Counting)
    //Counting() is a reduceByKey function to compute the support
5   counts of items
    PromisingRDD ← CountRDD.filter(isPromising)
6   //isPromising() is a filter function to remove non-promising items
    foreach 1-episode e1 in PromisingRDD
7       newPref ←pref.append(e1)
8       P ← P ∪ {newpref}
9       ProjSegments ← SegmentRDD.map(SegProject)
        //SegProject() is a map function to remove non-promising
        items and the items appeared before the new prefix
10      ProjComplexSeq ← ProjSements.flatMap(SegFlat)
        //SegFlat() is a flatMap function to flat segments into a
        complex event sequence with segmentation symbols
12      DeltaEpisodeMining(ProjComplexSeq, minsup, newPref)
13  end for
```

FIGURE 7. Pseudo Code of DeltaEpisodeMining.

3) EPISODE MERGING

Once delta episode mining is performed, pattern merging is initialized. It quickly matches the patterns found in batch episode mining and delta episode mining. If a pattern appears in both batch and delta-episode mining, its support can be computed easily using a weighted average. The weights of the supports from the batch and delta layers are determined by the data size of the storage of the batch and speed layers.

If the pattern appears in Delta episode mining only, the support of the pattern in batch episode mining will be considered as 0 to compute the weighted average because, in most cases, the data size and weight of support in the batch

layer are larger, and it is costly and not necessary to gather the missing information of the supports in batch episode mining. The error caused by this assumption is minor when the data size of the batch layer storage is much larger than that of the speed layer storage. No episode patterns can appear in batch episode mining in the batch layer but not in Delta episode mining in the speed layer, because patterns appearing in the batch layer are considered promising patterns in the speed layer, even if their support in Delta episode mining is low.

After matching, the patterns with re-computed supports are again compared with the user-specified minimum support to find the final patterns in pattern merging.

C. USER INTERFACE

In addition to the efficiency and effectiveness of finding frequent episodes over complex event sequences, another primary goal of this system is to facilitate the episode mining process for people to use. In our previous work [27], [28], there was not a good user interface, so it was not easy for domain experts to make good use of the system. In this work, we hope to enhance this part to make it more user-friendly, so we designed a user interface to make it easier to solve real-world problems.

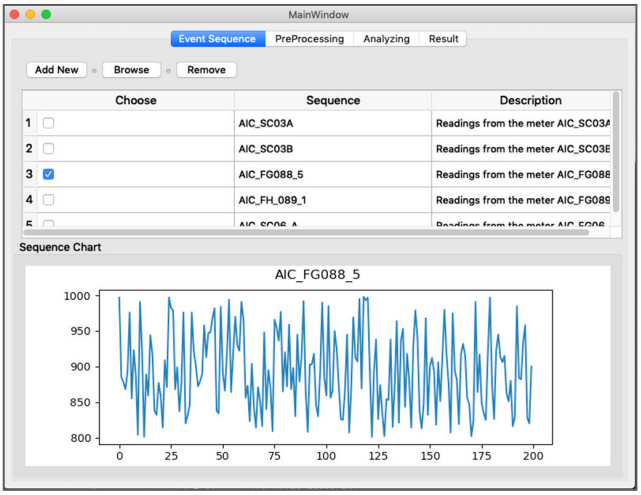

FIGURE 8. An screenshot of user interface.

Fig. 8 shows the user interface of the proposed application framework, domain experts or application developers can specify the complex event sequences to be analyzed and decide how to do the pre-processing jobs via the interface, instead of using hard-coding. After configuring the pre-processing jobs, the system will follow the configuration to process the data and then start the analytical mining process in the batch and the incremental perspective respectively.

V. EXPERIMENTAL EVALUATION

Since there is few similar study focused on developing a framework of complex event episode mining for various domain applications, as we did in this work, we consider three most relevant frameworks, namely *EM-CES* [27], *SCIEM* [28] and *TEM-SES* [1], for comparative evaluations with our proposed framework *SAAF*. To evaluate the performance of the tested frameworks, we used seven different datasets, namely temporal streaming data and data composed of multiple fields as typical complex event sequences, to conduct experiments as illustrated in this section.

TABLE 5. Experimental environment.

Roles	HW Spec.		
	Qty's	*CPU*	*Main Memory*
master	1	Intel i7-2600 @3.5GHz, 1600 MHz	32GB
slaves	10	Intel Celeron G1620 @2.7GHz, 1600 MHz	8GB

The experimental environment is composed of one master and 10 slaves. Cloudera CDH 6.3.4 with *Apache Spark* 2.4 is utilized to facilitate the cluster management job. The detailed hardware information of the cluster environment is presented in Table 5.

A. DATASETS AND EPISODE MINING SCENARIOS

The framework we proposed is mainly contributed to find useful patterns in solving real-life problems. The following three dataset were provided by different domain users. We help them find out interesting and useful patterns, and they take those patterns into their domain application to find out the valuable information.

The first dataset, denoted as DS1a, contains the electric power consumption data of a convenience store. The dataset was collected from 29 intelligent meters within 20 weeks, and the data collecting frequency was one record per minute. The total amount of records was almost 6.4 million, and each record consisted of six attributes: voltage(V), current(A), power(P), power factor(PF), kilowatt-hour(kWh), and recording_time. The problem we solved is how to find the episode patterns of excessive electricity usage. Users can easily find the episode patterns that led to the excessive electricity usage via this framework, and they take those patterns into their electricity monitoring application platform to help them not to use to more electricity than the contract allowed.

The second dataset, denoted as DS2a, was an exercise dataset that collected from wearable devices, including heart rate and GPS data, which have been collected via wearable devices over the years. The total amount of records was more than 13 billion. We only used those records with higher frequency and better data quality, so the current analysis of this dataset is approximately 600 million records. This dataset was provided by a wearable device that hoped to do healthcare business. The episode patterns were used in analyzing the relationship between sport behavior and health status. Our framework was incorporated with their healthcare platform and then provide information to their users.

The third dataset, denoted as DS3, is a type of streaming data from manufacturing devices. This dataset was collected in a fossil-fuel power station, and data collecting frequency was a record per second. The total amount of records was almost 53 million, and each record consisted of 384 attributes. This was a special application, and the domain user hoped to mine the episode patterns of events which are strongly related to the energy conversion performance.

We also used two benchmark datasets obtained from the UCI machine repository [60] for the evaluation. The first benchmark dataset, denoted as DS4, contains the recordings of 16 chemical sensors exposed to two dynamic gas mixtures at varying concentrations. For each mixture, signals were acquired continuously for 12 hours.

The second benchmark dataset, denoted as DS5, was recorded with two wearable devices: a chest-worn device (RespiBAN) and a wrist-worn device (Empatica E4). The RespiBAN device provides the following sensor data: electrocardiogram (ECG), electrodermal activity (EDA), electromyogram (EMG), respiration, body temperature, and three-axis acceleration.

Within the category ''energy-consumption'' and ''sports and healthcare'', other than the real-world dataset DS1 and DS3, we also found another two benchmark dataset from the UCI machine repository [60] for further evaluations. The dataset, denoted as DS1b, consists of measurements of electric power consumption in one household with a one-minute sampling rate over a period of almost 4 years. The second dataset, denoted as DS3b, comprises motion sensor data of 19 daily and sports activities each performed by 8 subjects in their own style for 5 minutes. Five Xsens MTx units are used on the torso, arms, and legs.

We used three datasets provided by domain users and four widely-used benchmark datasets to evaluate the framework in terms of three aspects: accuracy, efficiency, and scalability. Table 6 provides a brief description of the seven datasets, and the experimental results are presented in the following sections.

TABLE 6. The experimental dataset and description of the episodes.

	records	features	application domain	episodes to be mined
DS1a	6.4 million	6	energy consumption	To mine the episode patterns of the excessive electricity usage for electric power consumers then they can use that information for the management of power consumption
DS1b	2 million	9		
DS2a	600 million	24	sports and healthcare	To mine the episode patterns of abnormal heartbeat event sequences, which is very important to prevent some situations that may result in accidents.
DS2b	1.2 million	45		
DS3	53 million	384	manufacturing	To mine the episodes which are strongly related to the energy conversion performance, such as ash cleanups and cool system activities.
DS4	63 million	12	chemical	To mine the episodes of that contains an abnormal gas mixture of Methane, Ethylene and CO.
DS5	4.1 million	8	healthcare	To mine the episodes for wearable stress and affect detection, including physiological and motion data, recorded from both a wrist- and a chest-worn device.

B. EXPERIMENT ON THE NUMBER OF EPISODES AND ACCURACY

In this experiment, we first evaluated the number of episodes mined by the proposed framework *SAAF* and compared it with *TEM-SES, SCIEM and EM-CES,* which are taken as the baseline. The experiment was conducted under the same *minconf* of 0.8 and varied minimum support thresholds. The number of episodes mined by the four approaches is listed in Table 7. In Fig. 9, we can more easily see that the number of episodes found by different frameworks is very close.

The *EM-CES* is a basic full-mining framework that mines all qualified episodes that meet the criteria of *minsup* thresholds. Therefore, we take it as the baseline to validate the other two frameworks. In Table 8, we use the *EM-CES* framework as the baseline with 100% accuracy, and it shows the accuracy matrix among the seven different datasets under various minimum support thresholds. We found that the proposed *SAAF* performed better than the *TEM-SES* and *SCIEM* framework and was very close to the baseline.

C. EXPERIMENT ON THE EXECUTION TIME

In the second experiment, we evaluated the execution time of the proposed framework *SAAF* for mining frequent episodes under various minimum support thresholds. We provided a comparison with *TEM-SES* and the previous frameworks, *EM-CES* and *SCIEM*. The execution times for the four frameworks are listed in Table 9. In Fig. 10, the proposed *SAAF* has a better performance than the other three frameworks. The results of the first experiment, shown in Table 6, show that *SAAF* can consider both execution efficiency and accuracy simultaneously, as we expected.

D. EXPERIMENT ON SCALABILITY

In the third experiment, we conducted the experiment with different numbers of slaves (*Apache Spark* client) of 1, 2, 4, 8 and 10 to validate the scalability of the frameworks we proposed. The parameter *minsup* was 0.1 and *minconf* was 0.8.

TABLE 7. Number of episodes mined by four different frameworks among experimental datasets.

Dataset	Number of episodes / Framework	Minimum Support Threshold (%)						
		10%	15%	20%	25%	30%	35%	40%
DS1a	EM-CES(Baseline)	9982	1200	360	142	60	30	14
	TEM-SES	9375	1139	336	138	58	28	13
	SCIEM	9780	1109	345	135	50	26	13
	SAAF	9642	1186	353	138	58	29	14
DS1b	EM-CES(Baseline)	5638	655	224	85	33	16	8
	TEM-SES	5398	637	187	71	29	13	6
	SCIEM	5479	639	189	80	27	12	6
	SAAF	5558	646	214	81	30	15	7
DS2a	EM-CES(Baseline)	138925	39982	20183	8810	2530	1205	730
	TEM-SES	130149	38717	19039	8224	2459	1151	684
	SCIEM	134788	35800	19620	8443	2510	1187	709
	SAAF	137806	38804	19730	8754	2520	1193	722
DS2b	EM-CES(Baseline)	3437	446	145	55	22	14	6
	TEM-SES	3248	380	126	48	16	9	4
	SCIEM	3265	407	130	51	17	8	3
	SAAF	3352	424	138	53	18	11	5
DS3	EM-CES(Baseline)	39044	9924	2810	530	205	130	63
	TEM-SES	36289	9732	2645	504	200	123	60
	SCIEM	34800	9680	2656	510	187	109	58
	SAAF	38800	9730	2738	524	193	122	60
DS4	EM-CES(Baseline)	39087	10182	2692	524	190	127	64
	TEM-SES	34034	9290	2772	466	173	120	60
	SCIEM	34561	8457	2475	441	179	115	59
	SAAF	37846	8832	2350	486	180	97	61
DS5	EM-CES(Baseline)	10043	1143	336	139	61	29	13
	TEM-SES	8694	1062	318	137	53	25	10
	SCIEM	9221	1117	340	129	49	25	11
	SAAF	9501	1095	335	134	56	29	12

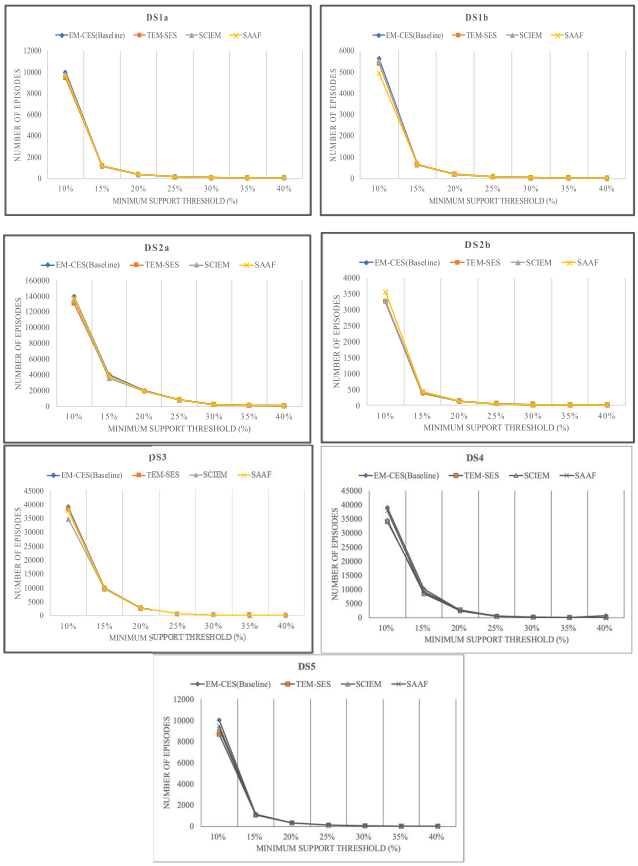

FIGURE 9. Chart of the episode number found by four different frameworks among seven datasets.

Table 10 shows that the execution time decreased when more slaves were involved in the mining process. We can see

TABLE 8. Accuracy of four different frameworks among experimental datasets.

Dataset	Accuracy / Framework	Minimum Support Threshold (%)						
		10%	15%	20%	25%	30%	35%	40%
DS1a	EM-CES(Baseline)	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	TEM-SES	93.9%	94.9%	93.3%	97.2%	96.7%	93.3%	92.9%
	SCIEM	98.0%	92.4%	95.8%	95.1%	83.3%	86.7%	92.9%
	SAAF	96.6%	98.8%	98.1%	97.2%	96.7%	96.7%	100.0%
DS1b	EM-CES(Baseline)	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	TEM-SES	95.7%	97.3%	83.5%	83.5%	87.9%	81.3%	75.0%
	SCIEM	97.2%	97.6%	84.4%	94.1%	81.8%	75.0%	75.0%
	SAAF	98.6%	98.6%	95.5%	95.3%	90.9%	93.8%	87.5%
DS2a	EM-CES(Baseline)	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	TEM-SES	93.7%	96.8%	94.3%	93.3%	97.2%	95.5%	93.7%
	SCIEM	97.0%	89.5%	97.2%	95.8%	99.2%	98.5%	97.1%
	SAAF	99.2%	97.1%	97.8%	99.4%	99.6%	99.0%	98.9%
DS2b	EM-CES(Baseline)	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	TEM-SES	94.5%	85.2%	86.9%	87.3%	72.7%	64.3%	66.7%
	SCIEM	95.0%	91.3%	89.7%	92.7%	77.3%	57.1%	50.0%
	SAAF	97.5%	95.1%	95.2%	96.4%	81.8%	78.6%	83.3%
DS3	EM-CES(Baseline)	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	TEM-SES	92.9%	98.1%	94.1%	95.1%	97.6%	94.6%	95.2%
	SCIEM	89.1%	97.5%	94.5%	96.2%	91.2%	83.8%	92.1%
	SAAF	99.4%	98.0%	97.4%	98.9%	94.1%	93.8%	95.2%
DS4	EM-CES(Baseline)	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	TEM-SES	87.1%	91.2%	103.0%	88.9%	91.1%	94.5%	93.8%
	SCIEM	88.4%	83.1%	91.9%	84.2%	94.2%	90.6%	92.2%
	SAAF	96.8%	86.7%	87.3%	92.7%	94.7%	76.4%	95.3%
DS5	EM-CES(Baseline)	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%
	TEM-SES	86.6%	92.9%	94.6%	98.6%	86.9%	86.2%	76.9%
	SCIEM	91.8%	97.7%	101.2%	92.8%	80.3%	86.2%	84.6%
	SAAF	94.6%	95.8%	99.7%	96.4%	91.8%	100.0%	92.3%

TABLE 9. Execution time of four different frameworks among experimental datasets.

DataSet	Execution Time(sec) / Framework	Minimum Support Threshold (%)						
		10%	15%	20%	25%	30%	35%	40%
DS1a	EM-CES	49	16	9	7	7	5	4
	TEM-SES	44	16	8	6	6	4	4
	SCIEM	36	13	7	5	4	3	2
	SAAF	25	11	5	3	2	2	2
DS1b	EM-CES	26	9	5	4	3	3	2
	TEM-SES	27	9	4	3	3	2	2
	SCIEM	20	8	4	3	2	2	1
	SAAF	14	6	3	2	1	1	1
DS2a	EM-CES	665	432	350	292	247	186	164
	TEM-SES	853	604	526	393	425	317	210
	SCIEM	443	302	248	219	189	173	124
	SAAF	357	248	190	161	141	127	107
DS2b	EM-CES	22	7	4	2	3	2	1
	TEM-SES	21	6	3	2	2	1	1
	SCIEM	16	5	3	2	1	1	0
	SAAF	10	4	2	1	1	1	1
DS3	EM-CES	185	96	75	68	48	38	31
	TEM-SES	213	134	103	73	65	55	41
	SCIEM	132	76	57	50	40	34	27
	SAAF	91	49	31	28	22	18	13
DS4	EM-CES	190	107	76	58	50	44	34
	TEM-SES	208	137	92	64	65	58	35
	SCIEM	141	80	53	46	35	29	26
	SAAF	115	87	48	39	37	29	18
DS5	EM-CES	53	16	8	7	7	5	4
	TEM-SES	45	14	9	7	7	3	3
	SCIEM	41	14	7	5	3	3	2
	SAAF	30	11	6	5	4	2	2

that the ratio of execution time improvement of dataset DS1a is not as good as that in DS2a and DS3. This may be because the data size of DS1a is smaller than that of the other two datasets, so when the number of slaves increases from four to ten, the improvement ratio is not as good as the others, such as increasing slaves from two to four.

Fig. 11 shows the scalability of the three frameworks, which comes from the design and development of these frameworks based on the development framework of *Apache Spark* and *Apache Spark Streaming*. Although there are only

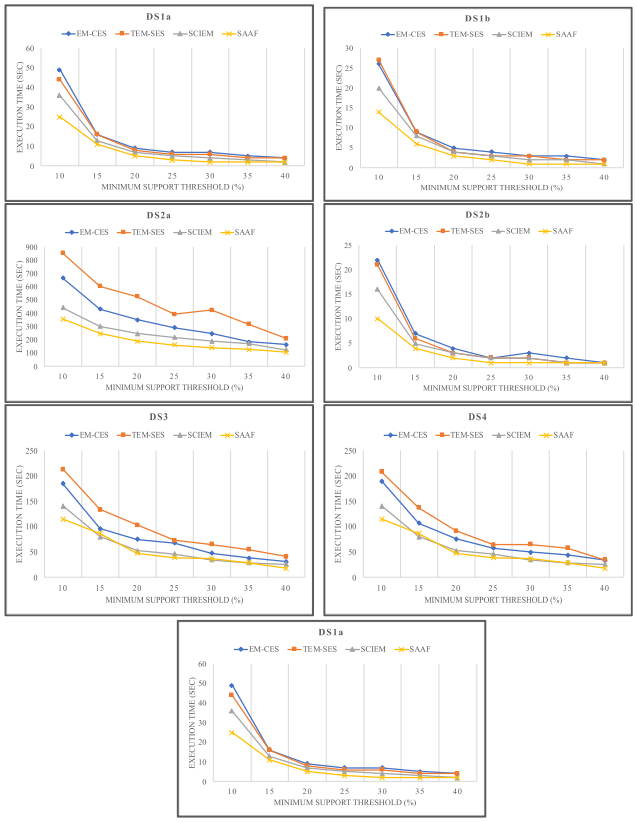

FIGURE 10. Chart of the execution time of four different frameworks among seven datasets.

TABLE 10. Execution time of different number of slaves.

Dataset	Execution Time(sec) / Approach	Number of slaves (Spark Client)				
		1	2	4	8	10
DS1a	EM-CES	221	128	72	53	52
	SCIEM	173	90	58	40	41
	SAAF	142	78	44	28	26
DS2a	EM-CES	3497	2082	1188	638	530
	SCIEM	2467	1430	793	439	378
	SAAF	1835	1008	660	369	320
DS3	EM-CES	1183	600	323	182	160
	SCIEM	885	521	291	142	131
	SAAF	528	310	169	96	80

ten slaves in our experiment, this shows that the proposed framework *SAAF* can work well with more slaves to achieve better scalability.

E. EXPERIMENT ON MODULE DATA PIPELINE CONTROLLER

We designed a module named data pipeline controller for users to dynamically adjust the data dispatching mechanism. In this module, users can control the frequency and scope of data dispatching to batch and incremental layers, respectively. The major benefit is that this module can work as an asynchronous buffer to balance the data input flow and processing capacity, thus avoiding data loss during the mining process.

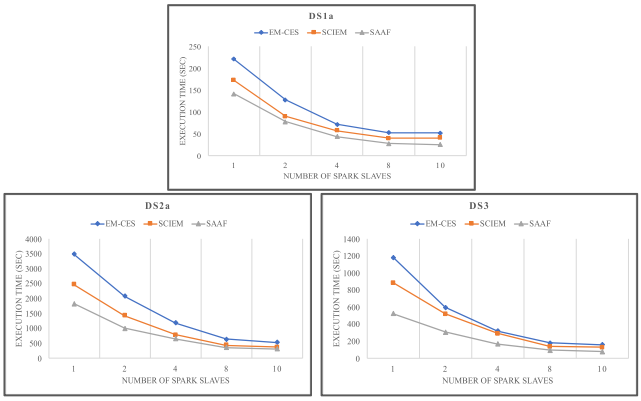

FIGURE 11. Chart of the execution time of different slave number of three different frameworks.

TABLE 11. Accuracy of different usage of data pipeline controller.

Dataset \ Accuracy	Pipeline Usage (%)				
	0%	25%	50%	75%	100%
DS2a	84.3%	88.4%	92.2%	94.7%	99.2%
DS3	84.4%	88.0%	92.6%	94.7%	96.8%
DS4	82.8%	86.3%	91.8%	95.2%	97.1%

TABLE 12. Execution time of different usage of data pipeline controller.

Dataset \ Execution Time(sec)	Pipeline Usage (%)				
	0%	25%	50%	75%	100%
DS2a	630	506	435	408	374
DS3	153	126	114	105	96
DS4	192	165	137	118	98

To evaluate the performance of the pipeline controller, we conducted an experiment using the dataset DS2a, DS3, DS4 with different usage percentage, from 0 $\sim$ 100%. Table 11 and 12 shows that the accuracy increases, and the execution time decreases when data pipeline controller was utilized more in the mining process. This result also shows that this module can help make the proposed framework more robust and efficient as we expected.

VI. CONCLUSION AND FUTURE WORKS

In this work, we have proposed a scalable analytical framework called *SAAF* for complex event episode mining in various domain applications. We have designed efficient algorithms consisting of three modules, namely *BatchEpisodeMining, DeltaEpisodeMining* and *PatternMerging* in correspondence with batch layer, speed layer and merge layer respectively. As far as we know, this is the first work that focuses on developing a scalable and easy-to-use framework for solving real-world problems in various domains. We extend the previous works, *EM-CES* and *SCIEM*, by developing new and important modules in *SAAF*,

including the data pipeline controller and the rule access interface.

We adopted the *Lambda* architecture to ensure the data safety and information trust, and used *Apache Spark* and *Apache Spark Streaming* as the development framework to boost the scalability and efficiency. To evaluate the accuracy and efficiency of *SAAF* framework, we used three real-world datasets from different domains and four benchmark datatsets to conduct the experiments. The results demonstrate that *SAAF* significantly outperforms other frameworks. The final experiment also shows that the proposed framework *SAAF* has excellent scalability in processing huge complex event episode mining jobs.

There exist some directions that we could explore in the future. First, we plan to develop a set of APIs for users to perform episode mining jobs with easy programming. Second, some optimizations can be conducted further to enhance the performance of the proposed modules.

ACKNOWLEDGMENT

The authors sincerely thank those who provided the real-world datasets for the experimental evaluation in this study.

REFERENCES

[1] Y. F. Lin, P. W. Jiang, and V. S. Tseng, "Efficient mining of frequent target episodes from complex event sequences," in *Frontiers in Artificial Intelligence and Applications*, vol. 274. Commack, NY, USA: Nova, 2015, pp. 501–510.

[2] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovery of frequent episodes in event sequences," *Data Mining and Knowledge Discovery*, vol. 1, pp. 259–289, Sep. 1997.

[3] H. Mannila and H. Toivonen, "Discovering generalized episodes using minimal occurrences," in *Proc. KDD*, 1996, pp. 146–151.

[4] H. Mannila, H. Toivonen, and A. I. Verkamo, "Discovering frequent episodes in sequences," in *Proc. Int. Conf. Knowl. Discov. Data Mining*, 1995, pp. 210–215.

[5] S. Laxman, P. Sastry, and K. Unnikrishnan, "Discovering frequent generalized episodes when events persist for different durations," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 9, pp. 1188–1201, Sep. 2007.

[6] T. You, Y. Li, B. Sun, and C. Du, "Multi-source data stream online frequent episode mining," *IEEE Access*, vol. 8, pp. 107465–107478, 2020.

[7] M.-Y. Su, "Discovery and prevention of attack episodes by frequent episodes mining and finite state machines," *J. Netw. Comput. Appl.*, vol. 33, no. 2, pp. 156–167, Mar. 2010.

[8] G. Casas-Garriga, "Discovering unbounded episodes in sequential data," in *Proc. Eur. Conf. Princ. Data Mining Knowl. Discovery*, 2003, pp. 83–94.

[9] C.-W. Wu, Y.-F. Lin, P. S. Yu, and V. S. Tseng, "Mining high utility episodes in complex event sequences," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2013, pp. 536–544.

[10] X. Ao, P. Luo, C. Li, F. Zhuang, and Q. He, "Discovering and learning sensational episodes of news events," *Inf. Syst.*, vol. 78, pp. 68–80, Nov. 2018.

[11] A. Dattasharma, P. K. Tripathi, and S. Gangadharpalli, "Identifying stock similarity based on episode distances," in *Proc. 11th Int. Conf. Comput. Inf. Technol.*, Dec. 2008, pp. 28–35.

[12] Y. F. Lin, C. F. Huang, and V. S. Tseng, "A novel episode mining methodology for stock investment," *J. Inf. Sci. Eng.*, vol. 30, no. 3, pp. 571–585, 2014.

[13] A. Ng and A. W. C. Fu, "Mining frequent episodes for relating financial events and stock trends," in *Proc. Pacific–Asia Conf. Knowl. Discovery Data Mining (PAKDD)*, in Lecture Notes in Computer Science, vol. 2637. Berlin, Germany: Springer, 2003, pp. 27–39.

[14] D. W. Cheung, J. Han, V. T. Ng, and C. Y. Wong, "Maintenance of discovered association rules in large databases: An incremental updating technique," in *Proc. 12th Int. Conf. Data Eng.*, Feb. 1996, pp. 106–114.

[15] D. W. Cheung, S. D. Lee, and B. Kao, "A general incremental technique for maintaining discovered association rules," in *Proc. Database Syst. Adv. Appl.*, Mar. 1997, pp. 185–194.

[16] S. Shan, X. Wang, and M. Sui, "Mining association rules: A continuous incremental updating technique," in *Proc. Int. Conf. Web Inf. Syst. Mining*, Oct. 2010, pp. 62–66.

[17] M.-Y. Lin and S.-Y. Lee, "Incremental update on sequential patterns in large databases by implicit merging and efficient counting," *Inf. Syst.*, vol. 29, no. 5, pp. 385–404, Jul. 2004.

[18] B. Mallick, D. Garg, and P. S. Grover, "Incremental mining of sequential patterns: Progress and challenges," *Intell. Data Anal.*, vol. 17, no. 3, pp. 507–530, May 2013.

[19] V. S. Tseng and C.-H. Lee, "Effective temporal data classification by integrating sequential pattern mining and probabilistic induction," *Expert Syst. Appl.*, vol. 36, no. 5, pp. 9524–9532, Jul. 2009.

[20] T. Matyashovskyy, *Lambda Architecture With Apache Spark DZone Big Data*. Durham, U.K.: Dzone.Com, 2016.

[21] *Apache Spark: A Fast and General Engine for Large-Scale Data Processing*, Spark.Apache.Org, Apache Spark, Berkeley, CA, USA, 2015.

[22] N. Deshai, B. V. D. S. Sekhar, and S. Venkataramana, "Mllib: Machine learning in apache spark," *Int. J. Recent Technol. Eng.*, vol. 8, no. 1, pp. 1235–1241, 2019.

[23] S. Salloum, R. Dautov, X. Chen, P. X. Peng, and J. Z. Huang, "Big data analytics on apache spark," *Int. J. Data Sci. Analytics*, vol. 1, nos. 3–4, pp. 145–164, 2016.

[24] G. P. Gupta and J. Khedwal, "Framework for error detection its localization in sensor data stream for reliable big sensor data analytics using apache spark streaming," *Proc. Comput. Sci.*, vol. 167, pp. 2337–2342, Jan. 2020.

[25] *Spark Streaming–Spark 2.4.4 Documentation*, Apache Software Foundation, Apache Spark, Berkeley, CA, USA, 2019.

[26] E. B. Johnsen, I. C. Yu, M. C. Lee, and J. C. Lin, "A configurable and executable model of spark streaming on apache YARN," *Int. J. Grid Utility Comput.*, vol. 11, no. 2, p. 185, 2020.

[27] J. C. C. Tseng, J. Y. Gu, P. F. Wang, C. Y. Chen, and V. S. Tseng, "A novel complex-events analytical system using episode pattern mining techniques," in *Proc. Intell. Sci. Big Data Eng. Big Data Mach. Learn. Techn. (IScIDE)*, 2015, pp. 487–498.

[28] J. C. C. Tseng, J.-Y. Gu, P. F. Wang, C.-Y. Chen, C.-F. Li, and V. S. Tseng, "A scalable complex event analytical system with incremental episode mining over data streams," in *Proc. IEEE Congr. Evol. Comput. (CEC)*, Jul. 2016, pp. 648–655.

[29] Y.-F. Lin, C.-W. Wu, C.-F. Huang, and V. S. Tseng, "Discovering utility-based episode rules in complex event sequences," *Expert Syst. Appl.*, vol. 42, no. 12, pp. 5303–5314, Jul. 2015.

[30] M. Atallah, R. Gwadera, and W. Szpankowski, "Detection of significant sets of episodes in event sequences," in *Proc. 4th IEEE Int. Conf. Data Mining (ICDM)*, Nov. 2004, pp. 3–10.

[31] K.-Y. Huang and C.-H. Chang, "Efficient mining of frequent episodes from complex sequences," *Inf. Syst.*, vol. 33, no. 1, pp. 96–114, Mar. 2008.

[32] N. Tatti and J. Vreeken, "The long and the short of it: Summarising event sequences with serial episodes," in *Proc. 18th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 2012, pp. 462–470.

[33] A. Hidri, A. Selmi, and M. S. Hidri, "Discovery of frequent patterns of episodes within a time window for alarm management systems," *IEEE Access*, vol. 8, pp. 11061–11073, 2020.

[34] X. Ao, P. Luo, C. Li, F. Zhuang, and Q. He, "Online frequent episode mining," in *Proc. IEEE 31st Int. Conf. Data Eng.*, Apr. 2015, pp. 891–902.

[35] L. Wan, J. Liao, and X. Zhu, "A frequent pattern based framework for event detection in sensor network stream data," in *Proc. 3rd Int. Workshop Knowl. Discovery From Sensor Data (SensorKDD)*, 2009, pp. 87–96.

[36] G. Xiao, A. Garg, D. Chen, D. Jiang, W. Shu, and X. Xu, "AHE detection with a hybrid intelligence model in smart healthcare," *IEEE Access*, vol. 7, pp. 37360–37370, 2019.

[37] V. S. Tseng, C.-H. Chou, K.-Q. Yang, and J. C. C. Tseng, "A big data analytical framework for sports behavior mining and personalized health services," in *Proc. Conf. Technol. Appl. Artif. Intell. (TAAI)*, Dec. 2017, pp. 178–183.

[38] M. H. Wong, V. S. Tseng, J. C. C. Tseng, S. W. Liu, and C. H. Tsai, "Long-term user location prediction using deep learning and periodic pattern mining," in *Proc. Int. Conf. Adv. Data Mining Appl.*, 2017, pp. 582–594.

[39] S. Moens, O. Jeunen, and B. Goethals, "Interactive evaluation of recommender systems with SNIPER: An episode mining approach," in *Proc. 13th ACM Conf. Recommender Syst.*, Sep. 2019, pp. 538–539.

[40] M. Amiri, L. Mohammad-Khanli, and R. Mirandola, "An online learning model based on episode mining for workload prediction in cloud," *Future Gener. Comput. Syst.*, vol. 87, pp. 83–101, Oct. 2018.

[41] M. Amiri, L. Mohammad-Khanli, and R. Mirandola, "A new efficient approach for extracting the closed episodes for workload prediction in cloud," *Computing*, vol. 102, no. 1, pp. 141–200, Jan. 2020.
[42] P. Fournier-Viger, P. Yang, J. C.-W. Lin, and U. Yun, "HUE-span: Fast high utility episode mining," in *Proc. Int. Conf. Adv. Data Mining Appl.*, 2019, pp. 169–184.
[43] N. F. Ayan, A. U. Tansel, and E. Arkun, "An efficient algorithm to update large itemsets with early pruning," in *Proc. 5th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining (KDD)*, 1999, pp. 287–291.
[44] G. S. Manku, "Approximate frequency counts over data streams," in *Proc. 28th VLDB Conf.*, 2000, pp. 346–357.
[45] D. Patnaik, S. Laxman, B. Chandramouli, and N. Ramakrishnan, "Efficient episode mining of dynamic event streams," in *Proc. IEEE 12th Int. Conf. Data Mining*, Dec. 2012, pp. 605–614.
[46] X. Ao, H. Shi, J. Wang, L. Zuo, H. Li, and Q. He, "Large-scale frequent episode mining from complex event sequences with hierarchies," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 4, pp. 36:1–36:26, 2019.
[47] O. Ouarem, F. Nouioua, and P. Fournier-Viger, "Mining episode rules from event sequences under non-overlapping frequency," in *Proc. Int. Conf. Ind., Eng. Other Appl. Appl. Intell. Syst.*, 2021, pp. 73–85.
[48] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proc. 8th ACM SIGMOD Workshop Res. Issues Data Mining Knowl. Discovery (DMKD)*, 2003, pp. 2–11.
[49] Y. Chen, P. Fournier-Viger, F. Nouioua, and Y. Wu, "Mining partially-ordered episode rules with the head support," in *Big Data Analytics and Knowledge Discovery (DaWak)* (Lecture Notes in Computer Science), vol. 12925. Cham, Switzerland: Springer, 2021, pp. 266–271.
[50] J. Qin, J. Wang, Q. Li, S. Fang, X. Li, and L. Lei, "Differentially private frequent episode mining over event streams," *Eng. Appl. Artif. Intell.*, vol. 110, Apr. 2022, Art. no. 104681.
[51] T. Guyet, W. Zhang, and A. Bifet, "Incremental mining of frequent serial episodes considering MultipleOccurrence," 2022, *arXiv:2201.11650*.
[52] J. Lin, E. Keogh, S. Lonardi, and B. Chiu, "A symbolic representation of time series, with implications for streaming algorithms," in *Proc. 8th ACM SIGMOD Workshop Res. Issues Data Mining Knowl. Discovery (DMKD)*, 2003, pp. 2–11.
[53] M. Hausenblas and N. Bijnens. (2013). *Lambda Architecture*. [Online]. Available: http://lambda-architecture.net/
[54] J. Kreps, N. Narkhede, and J. Rao, "Kafka: A distributed messaging system for log processing," in *Proc. ACM SIGMOD Workshop Netw. Meets Databases (NetDB)*, Athens, Greece, Jun. 2011.
[55] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Proc. IEEE 26th Symp. Mass Storage Syst. Technol. (MSST)*, May 2010, pp. 1–10.
[56] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "PrefixSpan,: Mining sequential patterns efficiently by prefix-projected pattern growth," in *Proc. 17th Int. Conf. Data Eng.*, Apr. 2001, pp. 215–224.
[57] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[58] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu, "Mining sequential patterns by pattern-growth: The PrefixSpan approach," *IEEE Trans. Knowl. Data Eng.*, vol. 16, no. 11, pp. 1424–1440, Nov. 2004.
[59] P. Kijsanayothin, G. Chalumporn, and R. Hewett, "On using MapReduce to scale algorithms for big data analytics: A case study," *J. Big Data*, vol. 6, no. 1, pp. 1–20, Dec. 2019.
[60] D. Dua and C. Graff, *UCI Machine Learning Repository: Data Sets*. Irvine, CA, USA: Univ. California, School of Information and Computer Science, 2019.

JERRY C. C. TSENG received the M.S. degree from the Department of Computer Science and Information Engineering, National Central University, Taoyuan, Taiwan, in 1996. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan. His current research interests include frequent pattern and episode mining, time-series data modeling, big data platform, some new topics of machine learning, and artificial intelligence.

SUN-YUAN HSIEH (Fellow, IEEE) received the Ph.D. degree in computer science from the National Taiwan University, Taipei, Taiwan, in June 1998. Then, he has served the compulsory two-year military service. From August 2000 to January 2002, he was an Assistant Professor with the Department of Computer Science and Information Engineering, National Chi Nan University. He joined the Department of Computer Science and Information Engineering, National Cheng Kung University, in February 2002, where he is currently a Chair Professor. His current research interests include design and analysis of algorithms, fault-tolerant computing, bioinformatics, parallel and distributed computing, and algorithmic graph theory. He is a fellow of the British Computer Society (BCS) and the Institution of Engineering and Technology (IET). He has an extremely impressive record of research achievements in areas of algorithms and fault-tolerant computing for interconnection networks. His awards include the 2020 ACM Distinguished scientist in 2019 Kwoh Ting Li Honorable Scholar Award in 2016, Outstanding Research Award of Taiwan Ministry of Science and Technology, President's Citation Award (American Biographical Institute) in 2007, Engineering Professor Award of Chinese Institute of Engineers (Kaohsiung Branch) in 2008, National Science Council's Outstanding Research Awards in 2009, IEEE Outstanding Technical Achievement Award (IEEE Tainan Section) in 2011, Outstanding Electronic Engineering Professor Award of Chinese Institute of Electrical Engineers in 2013, and Outstanding Engineering Professor Award of Chinese Institute of Engineers in 2014. He has served on organization committee and/or program committee of several dozen international conferences in computer science and computer engineering. He is also an experienced editor with editorial services to a number of journals, including serving as an Associate Editor for IEEE Transactions on Computers, IEEE Transactions on Reliability, IEEE ACCESS, *Journal of Computer and System Science* (Elsevier), *Theoretical Computer Science* (Elsevier), *Discrete Applied Mathematics* (Elsevier), *Journal of Supercomputing* (Springer), *Editor-in-Chiefs of International Journal of Computer Mathematics* (Taylor & Francis Group), *Parallel Processing Letters* (World Scientific), *Discrete Mathematics, Algorithms and Applications* (World Scientific), *and Managing editor of Journal of Interconnection Networks* (World Scientific).

VINCENT S. TSENG (Fellow, IEEE) received the Ph.D. degree with major in computer science from the National Chiao Tung University, in 1997. After that, he was a Postdoctoral Research Fellow with the Electrical Engineering and Computer Science Department, University of California at Berkeley, USA, from 1998 to 1999. He is currently a Chair Professor with the Department of Computer Science, National Yang Ming Chiao Tung University (NYCU). He was the Founding Director of the Institute of Data Science and Engineering, NYCU (2017–2020), the Chair of the IEEE CIS Tainan Chapter (2013–2015), and the President of the Taiwanese Association for Artificial Intelligence (2011–2012). He also acted as the Director of the Institute of Medical Informatics, National Cheng Kung University, from 2008 to 2011. He has authored more than 400 research papers in refereed journals and conferences and 15 patents (held and filed). His research interests include data mining, big data analytics, machine learning, biomedical informatics, and mobile and web technologies. He was a recipient of a number of prestigious awards, including the ACM Distinguished Scientist Member (2019), the Outstanding Research Award (2015 and 2019), the FutureTech Award by the National Science and Technology Council Taiwan (2022 and 2018), the Outstanding I. T. Elite Award (2018), and the K. T. Li Break-Through Award (2014). He served as the chair/a program committee member for a number of premier international conferences/institutions. He has been the Steering Committee Chair of PAKDD, since 2020. He was on the Editorial Board of a number of top journals, including the IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, the IEEE JOURNAL OF BIOMEDICAL AND HEALTH INFORMATICS, *IEEE Computational Intelligence Magazine*, and *ACM Transactions on Knowledge Discovery from Data*.