Enriching Complex Event Forecasting with Nested Patterns

Yuhui Chen, Ruihong Huang✉, Jinbo Xiong, Li Lin, and Jiayin Lin

College of Computer and Cyber Security, Fujian Normal University, Fujian, China
QSX20231392@student.fjnu.edu.cn,
{ruihong, jbxiong, li.lin, jy.lin}@fjnu.edu.cn

Abstract. Complex Event Recognition (CER) technology, as an important branch of Complex Event Processing (CEP), has driven the development of various related techniques through its progressive research advancements. A new direction in this field is to predict when a pattern might occur before the CER engine detects its actual occurrence, enabling proactive responses to potential events, which is referred to as Complex Event Forecasting (CEF). However, existing studies on CEF have primarily focused on sequential patterns, with limited attention given to effectively describing and handling complex nested patterns. To enhance CEF's capacity to predict nested patterns with enriched semantics, we propose a method to simplify nested patterns utilizing several flattening rules and employ Symbolic Finite Automaton (SFA) to encode different operators. We also delineate the characteristics of patterns that are conducive to accurate prediction. Additionally, we present two methods for deriving predictions using a probabilistic model, balancing the trade-offs between time and memory costs. Finally, we conduct a comparative analysis of the temporal and spatial efficiency of our optimizations and assess the prediction quality using two models, demonstrating the advantages of our approach.

Keywords: Complex Event Forecasting · Complex Event Recognition

1 Introduction

With the continuous development of Complex Event Processing technologies, Complex Event Forecasting (CEF) has emerged as a new and promising research direction, playing a significant role in multiple areas such as financial market prediction and traffic management [8]. Unlike Complex Event Recognition (CER), which focuses on detecting patterns in real-time event streams, CEF aims to forecast when a pattern might occur before it is actually detected by a CER engine, particularly when a specific pattern is undesirable. The primary goal of Complex Event Forecasting is to intervene at the right moment to minimize potential losses. For example, in maritime surveillance systems, CEF is crucial for predicting ship arrival times, which aids in the efficient arrangement of port resources, optimization of shipping schedules, and ensuring navigation

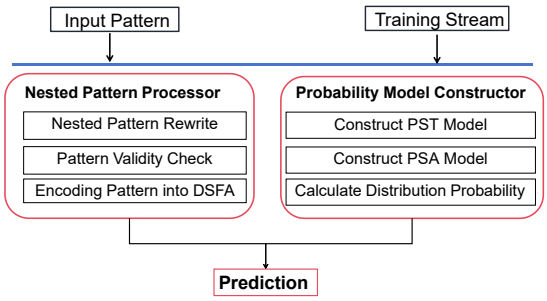


Fig. 1. System Overview

safety [4]. However, existing studies on Complex Event Forecasting are primarily based on sequential patterns (SEQ). There is a lack of methods for effectively forecasting patterns with more semantic complexity, such as conjunction (AND) or even more intricate nested patterns, which are commonly addressed in the field of Complex Event Recognition [12, 13]. The ability to express nested patterns with richer semantics is essential for advancing the capabilities of CEF. For instance, a forecasting pattern for extreme weather warnings in meteorology might involve a sequence of mild weather events followed by a sudden change involving both thunderstorms and hail. This scenario could be represented by a nested pattern like SEQ(Mild, Mild, Mild, AND(Thunderstorm, Hail)), where SEQ/AND denotes the sequential/concurrent occurrence of events, respectively.

In this paper, we present a formal framework for CEF with nested patterns, enabling users to define nested patterns with richer semantics. Figure 1 illustrates the overall components of our approach. To forecast whether and when a complex event is expected to occur in an event stream, we need to construct a probabilistic model for the input pattern. We begin by encoding the input pattern into a symbolic automaton [12]. Next, we use a portion of the input stream as a training dataset to learn a probabilistic model that captures dependencies among the events in the stream. Finally, this probabilistic model is used to derive forecasts about the expected occurrence of the complex event encoded by the automaton. Our contributions can be summarized as follows:

- We propose a framework that enriches the task of CEF by supporting the nesting patterns of AND, OR, Negation, and SEQ operators
- We present two methods for using the probabilistic model to derive predictions, balancing the trade-offs between time and memory costs.
- We conduct an extensive experimental evaluation to demonstrate the performance of the proposed methods.

2 Related Work

The rapid growth of large-scale event systems and the Internet has increased the demand for Complex Event Processing (CEP). CEP is critical technology [5,7,9],

but the need for nested Complex Event Forecasting (CEF) remains largely unaddressed. SQL supports nesting to any depth, with techniques like Kim's algorithm [10] converting nested SQL queries into simpler forms for efficient processing. However, SQL's reliance on predicates for AND/OR operators can result in cumbersome queries. Tree-based models [13, 16] handle nested queries but focus on basic event negation without optimizing nested expressions. Liu's work [11,12] extends SEQ nesting to AND/OR, employing decomposition and shared merging techniques to improve multi-pattern detection efficiency. Tools like [2] use symbolic automata and Markov chains for CEF but primarily focus on SEQ patterns. Similarities exist with other fields like time series forecasting [14] and anomaly detection [6], though CEF predicts future events, while these focus on quantitative forecasting or identifying anomalies. Approaches such as symbolic register automata (SRA) [3] encode patterns and use predictive suffix trees for probabilistic modeling. While SRA supports key operators (e.g., Seq, OR), it lacks negation support. DSFA-based models [4] inherently support OR/negation and efficiently handle AND through flattened patterns, which reduce memory usage by activating sub-patterns dynamically. These advancements improve pattern encoding and detection efficiency.

3 Transforming Nested Patterns into Deterministic Symbolic Finite Automaton

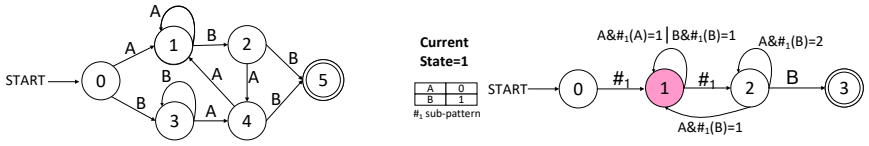
Our main idea is to use Symbolic Finite Automaton (SFA) [4] to represent nested patterns provided by users. Toward the target of expressing nested patterns with richer semantics for Complex Event Forecasting, we apply the Nested Complex Event Language (NEEL) [12], a commonly-used language in CER, which supports the nesting of AND, OR, Negation and SEQ operators at any level.

The SEQ Operator represents the sequence in which the events are expected to occur. The pattern SEQ($E_1\, e_1$, ..., $E_i\, e_i$, ..., $E_n\, e_n$) indicates that the event instances of interest $e_1, \ldots, e_i, \ldots, e_n$ must be captured in sequential order.

The OR Operator indicates the alternative occurrence of events. The pattern OR($E_1\, e_1$, ..., $E_i\, e_i$, ..., $E_n\, e_n$) means one event instance of types E_1, ..., E_i, ..., E_n will be detected.

The AND Operator is similar to SEQ operator but does not concern itself with the order of the events it accepts. The pattern AND($E_1\, e_1$, ..., $E_i\, e_i$, ..., $E_n\, e_n$) accepts a sequence of event instances of types E_1 , ..., E_i , ..., E_n without requiring a specific order of events. Figure 2(a) presents a naive enumeration approach to implementing the pattern = SEQ(AND(A, B), B), where the AND(A, B) is converted into two sequences, SEQ(A, B) and SEQ(B, A). This method is feasible when the number n of events within an AND operator is small, but as n increases, the number of states in the automaton grows exponentially.

We maintained a sub-pattern table to assist with matching for the AND operator. The sub-pattern table records the matching status of the AND operator, with numbers indicating the matching order of the AND components (0 represents no match). When the automaton transitions to matching events within



(a) Simple Enumeration Approach (b) Optimization with a Sub-pattern Table

Fig. 2. A comparison of implementing pattern = SEQ(AND(A, B), B).

the AND operator, it will activate the sub-pattern [17]. If an input event causes the sub-pattern to return true, the automaton will transition to the next state (i.e., one event within the AND operator is matched). Consequently, the number of states required for the AND operator will be equal to the number of events within the AND operator, significantly reducing system overhead.

Example 1. As illustrated in Figure 2(b), we denote the operation related to the sub-pattern AND(A, B) with the symbol "$\#_1$". When an input event of type A or B is received at state 0, we create a table in memory for the sub-pattern AND(A, B), labeled with "$\#_1$". The table is initialized with the two event types A and B, both marked with a number 0 to indicate their unmatched status. If an event of type A is first encountered at state 0, we update the table to change the number associated with A to 1, indicating its arrival order, and the automaton transitions from state 0 to state 1. Subsequently, if an event of type B is received at state 1, we update the number associated with B to 2 in the table, denoting its sequence, and transition the automaton to state 2.

At state 2, the reception of a type B event prompts the automaton to move to the final state 3. Conversely, if a type A event is received, we must consult the number associated with B in the table "$\#_1$". If $\#_1$(B) = 2, this indicates that we have detected the sequence "A, B" for AND(A, B). In such a scenario, we can remain at state 2 upon the arrival of a new type A event by revising $\#_1$(B) = 1 and $\#_1$(A) = 2 in the table, effectively updating the match for AND(A, B) to the latest sequence "B, A". After that, if an additional type A event is received and the current $\#_1$(B) = 1, we revert to state 1, resetting $\#_1$(B) = 0 and $\#_1$(A) = 1, and await a type B event to proceed to state 2.

The Negation is represented by a symbol "!" preceding an event E_i, signifying that E_i is prohibited from appearing in the specified position. In fact, the negation $!E_i$ is equivalent to the occurrence of every other event within the alphabet Σ. Therefore, negation can be handled using the OR operator by replacing $!E_i$ with OR(Σ - E_i).

4 Statistical Model and Prediction Execution

Once the nested input pattern is encoded into a DSFA, the main idea of our forecasting method is to employ the DSFA to develop a probabilistic model that captures dependencies among the events in an input stream.

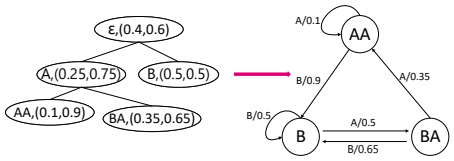


Fig. 3. An example of converting a PST into a PSA.

4.1 PST and PSA models

A Prediction Suffix Tree (PST) is constructed based on the inherent relationships within the data stream [18], and it can provide the probability of the next occurring complex event (CE) based on the CEs it has remembered. The left part of Figure 3 shows an example of a PST with an order $m = 2$. According to this tree, if the most recent event type observed in the stream is A, and we disregard any preceding events, the probability of the next input event type being A is 0.25. However, a more accurate estimation of the subsequent event's probability can be achieved by expanding the context and examining one additional event in the past. Thus, if the last two events observed are of types "B, A", the probability of encountering another type A event is 0.35. On the other hand, if the most recent event type encountered is B, the probability distribution for the next event type is (0.5, 0.5).

A Probabilistic Suffix Automaton (PSA) possesses a clearly defined state space and transition function, which enables the computation of the Markov property. Once a PST is acquired, it can be transformed into the corresponding PSA [18] With a PSA, we are able to process a sequence of events and, at any given moment, provide an estimation of the subsequent events likely to be encountered along with their respective probabilities. Figure 3 illustrates an example of converting a PST into a PSA, utilizing the tree's leaf nodes as the automaton's states. There is no necessity to expand state B into states BB and AB, as the PST suggests that such an expansion lacks statistical significance. According to this PSA, if the most recent event consumed from the stream is of type B, the PSA would reside in state B, with the probability of the next event being type A being 0.5. If the most recent event in the stream is of type A, it becomes necessary to expand this suffix to consider one additional event from the past. If the last two events are of types "B, A", the PSA would be in state BA, and the probability of the next event being type A would then be 0.35.

4.2 Generating Forecasts

For the purpose of complex event forecasting, we are interested in inferring whether and when the SFA of the input pattern will reach one of its final states. To determine the likelihood that a Complex Event (CE) will occur within the next k input events, we calculate the sum of the probabilities of the paths using the probability distribution constructed from the PST and PSA models. We

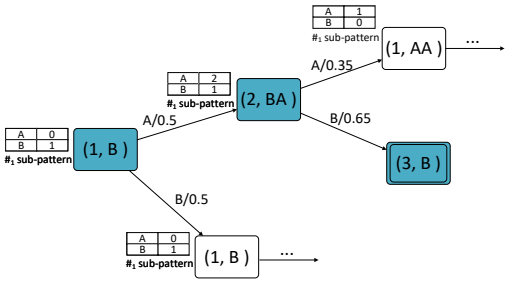


Fig. 4. Example of calculation of probability distribution for the PST model using the PST in Figure 3.

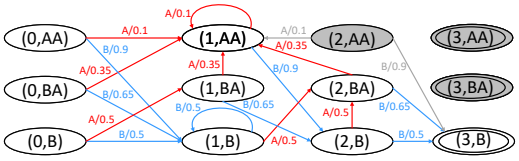


Fig. 5. Example embedding of the PSA in Figure 3, which is combined with the DSFA in Figure 2(b) of the nested pattern = SEQ(AND(A, B), B).

aggregate the probabilities of all paths with lengths that do not exceed k. If the probability of the current prediction exceeds our preset threshold, we emit a positive prediction, indicating that the automaton is anticipated to reach a final state within the next k events, implying the occurrence of a CE. Otherwise, we issue a negative prediction, suggesting that no CE is expected.

For the PST model, the probability distribution is derived through the iterative expansion of the PST. As depicted in Figure 4, assuming the current state of the automaton is 1 and the event that has been remembered is B, we must take into account all potential inputs to extend the path from the current node to a final state.

By using the PSA model for forecasting, we need to further build the embedding of the PSA according to the automaton encoded by the input event pattern. This embedding allows us to track the state of the automaton at every point in time, enabling us to estimate which future paths might lead to a final state. Figure 5 illustrates the embedding of the PSA from Figure 3 combined with the input pattern = SEQ(AND(A, B), B) It is noted that this embedding includes some redundant states and transitions, specifically those indicated in gray, which have no incoming transitions and are therefore inaccessible. The PSA model can then compute the probability distribution at each node by traversing its embedding structure. In Figure 5, the distribution for the node (1, AA) is computed by traversing the PSA model to identify all paths leading from the current node to the final state nodes. To avoid infinite path extensions caused by cycles, we can impose a constraint on the maximum path length.

5 Experiments

All experiments were conducted on a 64-bit Windows 10 machine equipped with an Intel Core i7-8750H CPU @ 2.20GHz. In each experiment, we repeated the process multiple times and averaged the results to evaluate the performance. The source code is available online[1].

5.1 Weather Forecast

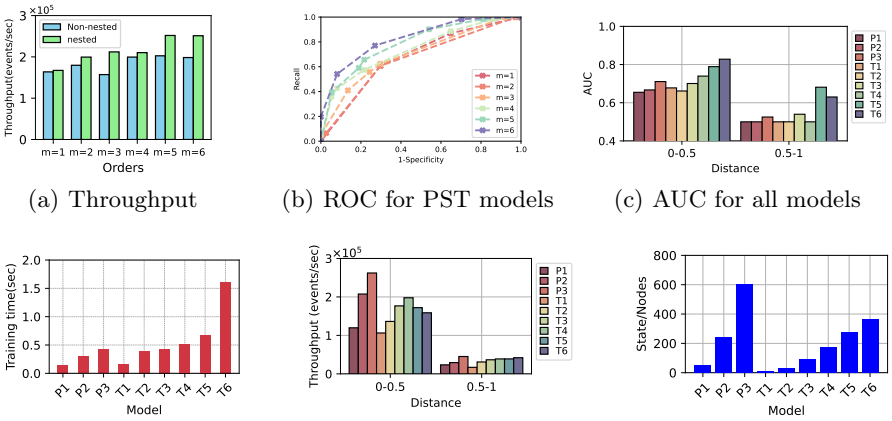
The first dataset used in this experiment is a real-world dataset [15] covering weather events from 49 U.S. states. It includes up to 8.6 million events, ranging from common weather conditions (such as rain and snow) to extreme phenomena (such as storms and extreme cold), referred to as abnormal weather. We processed the dataset by categorizing it into 1715 city-specific weather files. Each day was segmented into two time periods: 00:00-12:00 and 12:00-24:00. Since the dataset only contains records of abnormal weather, there were instances where data was absent for certain days. To address this, we interpolated normal weather events for the missing periods, ensuring that each time slot had at least one recorded event. We used 75% of the processed dataset for model training and the remaining 25% for prediction.

The event pattern utilized in this experiment is designed to detect shifts in weather conditions within a city and to forecast the onset of continuous rain or snow. This pattern offers significant value for urban planning and assists citizens in planning their travel and daily activities.

$$\text{Weather Pattern} = \text{SEQ(Soft, Soft, Soft, OR(Rain, Snow), OR(Rain, Snow), OR(Rain, Snow))}$$

The experimental results are shown in Figure 6. To compare with the CEF method focusing only on the sequential (non-nested) patterns [4], we conducted predictions for the following two patterns: $pattern_1$ = SEQ(Soft, Soft, Soft, Rain, Rain, Rain) and $pattern_2$ = SEQ(Soft, Soft, Soft, Snow, Snow, Snow). These patterns are both potential flattened sequential representations of our nested input pattern. In contrast, we executed forecasts on the original nested pattern using the CEF methodologies introduced in this study. Since both methods are based on a variable-order Markov model, we compared their performance across different orders m. As illustrated in Figure 6(a), the throughput of our proposed methods outperforms that of the method dedicated to non-nested patterns, even though there are additional possible flattened sequential patterns of our nested input pattern beyond $pattern_1$ and $pattern_2$. This superiority arises from the fact that the CEF method tailored for non-nested patterns needs to perform predictions on each possible flattened sequential pattern, while our proposed approach can directly process the original nested pattern, offering a more efficient and streamlined analysis.

[1] https://github.com/ChenYH-fjnu/complexEvent



(a) Throughput (b) ROC for PST models (c) AUC for all models

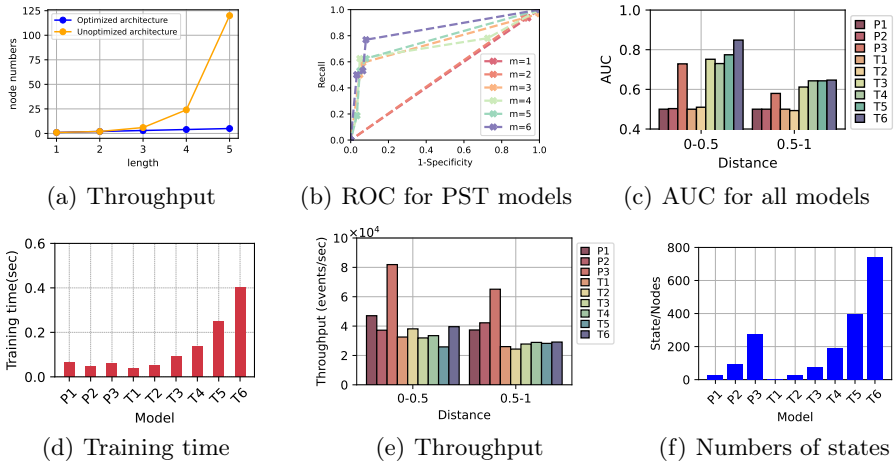
(d) Training time (e) Throughput (f) Numbers of states

Fig. 6. Results for nested CEF in the domain of weather prediction. Models Px and Tx correspond to the PSA and PST models with various orders m, respectively.

Figure 6(b) displays the ROC curve for the PST model across variable orders within the distance range [0, 0.5]. The results show improved performance as m increases, indicating that higher orders lead to better prediction. Figure 6(c) presents the AUC values, where higher values denote superior model performance. The model performance within the [0, 0.5] distance range exhibits a similar trend to that in Figure 6(b). In the [0.5, 1] distance range, most models have AUCs around 0.5, suggesting random predictions, which confirms that the forecast is more precise when the complex event is anticipated to occur relatively soon. Figure 6(d) reveals that the training time for the PST model increases with the order m due to the addition of more leaf nodes. Figure 6(e) demonstrates that for the same order m, the PSA model achieves significantly higher throughput than the PST model. This is because the PSA model directly traverses adjacent nodes in the embedding graph to calculate predictions, whereas the PST model requires more tree traversal. However, as illustrated in Figure 6(f), the PST model is more memory-efficient for high-order tasks, while the PSA model sacrifices memory efficiency for higher throughput.

5.2 Traffic Volume Prediction

The second dataset used in our experiments comprises vehicle data from several road intersections [1], recording the number of vehicles at each intersection on an hourly basis. We classified the data based on intersection IDs and chose those intersections that most frequently exhibited our predefined patterns for experimental purposes. We also categorized the congestion levels based on the vehicle counts into five degrees: Clear, Mild, Moderate, Severe, and Extreme. The prediction pattern utilized in this experiment is designed to identify conges-



(a) Throughput (b) ROC for PST models (c) AUC for all models

(d) Training time (e) Throughput (f) Numbers of states

Fig. 7. Results for nested CEF in the domain of traffic volume prediction. Models Px and Tx correspond to the PSA and PST models with various orders m, respectively.

tion at intersections. By accurately forecasting the occurrence of this pattern, drivers can be well-informed to make strategic road planning decisions, thereby mitigating congestion and reducing the risk of accidents at these critical points.

$$\text{Traffic Volume Pattern} = \text{SEQ}(\text{Mild}, \text{Mild}, \text{AND}(\text{Moderate}, \text{Severe}))$$

First, we compared the optimization of the AND operator when encoding the input event patterns as symbolic automaton, as introduced in Section 3. As shown in Figure 7(a), as the number of events included in the AND operation increases (i.e., the number n in AND$(E_1, E_2, ..., E_n)$ increases), both our implementation and the naive enumeration method lead to an increase in the number of nodes in the automaton. However, it is evident that our proposed optimization method significantly reduces the number of additional nodes in the automaton compared to the simple enumeration method. The other results presented in Figure 7 show similar trends to those observed in the previous section.

6 Conclusions

In this article, we present a framework for CEF with nested patterns, enabling users to define nested patterns with richer semantics. The experimental results demonstrate that the PST model requires less memory, allowing it to support higher-order models and ensure higher accuracy. In contrast, the PSA model is more efficient in terms of prediction time.

Acknowledgments. This work is supported in part by the Natural Science Foundation of Fujian Province (2023J05129, 2023J02014), and the National Natural Science Foundation of China (62272102, 62471139, 62307008).

References

1. Traffic prediction dataset, https://www.kaggle.com/datasets/fedesoriano/traffic-prediction-dataset
2. Alevizos, E., Artikis, A., Paliouras, G.: Wayeb: a tool for complex event forecasting. arXiv preprint arXiv:1901.01826 (2018)
3. Alevizos, E., Artikis, A., Paliouras, G.: Symbolic register automata for complex event recognition and forecasting. arXiv preprint arXiv:2110.04032 (2021)
4. Alevizos, E., Artikis, A., Paliouras, G.: Complex event forecasting with prediction suffix trees. The VLDB Journal **31**(1), 157–180 (2022)
5. Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N.: Real-time complex event recognition and reasoning–a logic programming approach. Applied Artificial Intelligence **26**(1-2), 6–57 (2012)
6. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM computing surveys (CSUR) **41**(3), 1–58 (2009)
7. Cugola, G., Margara, A.: Processing flows of information: From data stream to complex event processing. ACM Computing Surveys (CSUR) **44**(3), 1–62 (2012)
8. Dal Pozzolo, A., Boracchi, G., Caelen, O., Alippi, C., Bontempi, G.: Credit card fraud detection: a realistic modeling and a novel learning strategy. IEEE transactions on neural networks and learning systems **29**(8), 3784–3797 (2017)
9. Giatrakos, N., Alevizos, E., Artikis, A., Deligiannakis, A., Garofalakis, M.: Complex event recognition in the big data era: a survey. The VLDB Journal **29**, 313–352 (2020)
10. Kim, W.: On optimizing an sql-like nested query. ACM Transactions on Database Systems (TODS) **7**(3), 443–469 (1982)
11. Liu, M., Ray, M., Rundensteiner, E.A., Dougherty, D.J., Gupta, C., Wang, S., Ari, I., Mehta, A.: Processing nested complex sequence pattern queries over event streams. In: Proceedings of the Seventh International Workshop on Data Management for Sensor Networks. pp. 14–19 (2010)
12. Liu, M., Rundensteiner, E., Dougherty, D., Gupta, C., Wang, S., Ari, I., Mehta, A.: High-performance nested cep query processing over event streams. In: 2011 IEEE 27th International Conference on Data Engineering. pp. 123–134. IEEE (2011)
13. Mei, Y., Madden, S.: Zstream: a cost-based query processor for adaptively detecting composite events. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of data. pp. 193–206 (2009)
14. Montgomery, D.C., Jennings, C.L., Kulahci, M.: Introduction to time series analysis and forecasting. John Wiley & Sons (2015)
15. Moosavi, S., Samavatian, M.H., Nandi, A., Parthasarathy, S., Ramnath, R.: Short and long-term pattern discovery over large-scale geo-spatiotemporal data. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. pp. 2905–2913 (2019)
16. Perera, K., Ahangama, S.: A review of query optimization techniques for complex event processing. In: 2019 4th International Conference on Information Technology Research (ICITR). pp. 1–7. IEEE (2019)
17. Perera, K., Ahangama, S.: A review of query optimization techniques for complex event processing. In: 2019 4th International Conference on Information Technology Research (ICITR). pp. 1–7. IEEE (2019)
18. Ron, D., Singer, Y., Tishby, N.: The power of amnesia: Learning probabilistic automata with variable memory length. Machine learning **25**, 117–149 (1996)