IEICE TRANS. INF. & SYST., VOL.E103–D, NO.5 MAY 2020

PAPER Special Section on Data Engineering and Information Management

Multiple Regular Expression Pattern Monitoring over Probabilistic Event Streams

Kento SUGIURA[†a)], ***Nonmember and*** **Yoshiharu ISHIKAWA**[†], ***Member***

SUMMARY As smartphones and IoT devices become widespread, probabilistic event streams, which are continuous analysis results of sensing data, have received a lot of attention. One of the applications of probabilistic event streams is monitoring of time series events based on regular expressions. That is, we describe a monitoring query such as "*Has the tracked object moved from* `RoomA` *to* `RoomB` *in the past 30 minutes?*" by using a regular expression, and then check whether corresponding events occur in a probabilistic event stream with a sliding window. Although we proposed the fundamental monitoring method of time series events in our previous work, three problems remain: 1) it is based on an unusual assumption about slide size of a sliding window, 2) the grammar of pattern queries did not include "negation", and 3) it was not optimized for multiple monitoring queries. In this paper, we propose several techniques to solve the above problems. First, we remove the assumption about slide size, and propose adaptive slicing of sliding windows for efficient probability calculation. Second, we calculate the occurrence probability of a negation pattern by using an inverted DFA. Finally, we propose the merge of multiple DFAs based on disjunction to process multiple queries efficiently. Experimental results using real and synthetic datasets demonstrate effectiveness of our approach.

key words: *probabilistic event streams, regular expressions, pattern matching, sliding windows*

1. Introduction

It has become important to process *probabilistic event streams* because of the development of sensing technologies and machine learning. Machine learning techniques, particularly supervised learning, can recognize the real world events from raw sensing data, such as accelerometer of smartphones [1] and RFID and Wi-Fi signals [2]. Although recognized events are useful and intuitive, the uncertainty of classification is a crucial problem for utilization. In other words, analysis results of machine learning include errors because of some reasons, such as sensing noises and unpredictable incidents in the real world. We consider such non-definite event streams as probabilistic event streams. For instance, Fig. 1 shows a probabilistic event stream of indoor location tracking. The probabilities indicate the possibility of presence at corresponding locations. That is, the uncertainty of classification is expressed by probabilistic distributions at every time step.

To utilize probabilistic event streams, existing methods proposed monitoring of time series events based on *a regular expression* [3]–[9]. In other words, they challenge *complex event processing* [10], [11] over probabilistic event streams. Although machine learning can recognize raw sensing data as the real world events, each event relates to only one time step. Since a probabilistic event cannot recognize continuous event occurrence directly, existing work tried to detect a *time series event* that is an event sequence with a specific order [12], [13]. For example, consider a time series event that is "*a tracked object moves from* `RoomA` *to* `RoomB`" in Fig. 1. In this case, the following regular expression shows such a time series event:

$$q = \langle \mathtt{a}^{+} \; .^{*} \; \mathtt{b}^{+} \rangle,$$

where `a` and `b` are abbreviations of `RoomA` and `RoomB`, respectively. Note that a wild card (a dot symbol) indicates any event symbol, and so pattern q accepts any route from `RoomA` to `RoomB`.

event symbol	time step 1	2	3	4	5	6	7	···
`RoomA (a)`	0.60	0.60	0.10	0.05	0.05	0.05	0.05	
`RoomB (b)`	0.05	0.05	0.05	0.05	0.60	0.60	0.60	
`HallC (c)`	0.15	0.15	0.45	0.45	0.10	0.10	0.10	···
`HallD (d)`	0.10	0.10	0.20	0.25	0.15	0.15	0.15	
`HallE (e)`	0.10	0.10	0.20	0.20	0.10	0.10	0.10	

Fig. 1 A probabilistic event stream

The existing work [7] uses a sliding window with regular expressions to describe monitoring queries. A sliding window limits the maximum duration w between the first event occurrence and the last one to prevent too long time series events from detecting. Furthermore, a sliding window can change its update frequency l according to user's requirements. Let us continue the above example. Consider that a user wants to detect movement events q in the past 30 minutes and update detection results at every 5 minutes. When a unit of time is 5 minutes in Fig. 1, window and sliding sizes are $w = 6$ and $l = 1$, respectively. Note that we denote a certain time window as $[t_s : t_e]$ in this paper. Thus, a sliding window changes such as $[1 : 6]$, $[2 : 7]$, and $[3 : 8]$.

Existing work [3]–[7], [9], however, does not calculate the appropriate probabilities of time series events for monitoring queries. In a definite event stream, matching results (i.e., *matches*) show the occurrence of specified time series events. On the other hand, since matches show only the possibility of occurrence of time series events in probabilistic event streams, we have to check whether a time series

Manuscript received June 26, 2019.
Manuscript revised November 4, 2019.
Manuscript publicized February 3, 2020.

[†]The authors are with Graduate School of Informatics, Nagoya University, Nagoya-shi, 464–8601 Japan.
a) E-mail: sugiura@db.is.i.nagoya-u.ac.jp

DOI: 10.1587/transinf.2019DAP0009

event occurs or not in a window. Consider the previous example. When we answer the query "*has a tracked object moved from* `RoomA` *to* `RoomB` *in the past 30 minutes?*", we need to calculate the occurrence probability of such a movement in each time window. However, almost all existing methods focused on the detection of matches and did not propose the calculation methods for the probabilities of time series events. For instance, when we apply the above query to Fig. 1, existing methods detect some matches from [1 : 6] as follows:

$$m_1 = \langle \mathtt{a}_1, .2, .3, .4, \mathtt{b}_5 \rangle \qquad P(m_1) = 0.36,$$
$$m_2 = \langle \mathtt{a}_1, .2, .3, .4, .5, \mathtt{b}_6 \rangle \qquad P(m_2) = 0.36, \text{ and}$$
$$m_3 = \langle \mathtt{a}_2, .3, .4, \mathtt{b}_5, \mathtt{b}_6 \rangle \qquad P(m_3) = 0.216.$$

Although each match has its occurrence probability, it is the probability of a specific movement from `RoomA` to `RoomB`. It is difficult for a user to infer the occurrence of a time series event from these matches because they are correlated and may take small probabilities with Kleene closures [9].

We thus propose an efficient method to calculate the probabilities of time series events with a sliding window. The proposed method does not detect matches but calculates the probabilities of time series events in a sliding window directly. Let us continue the above example. We calculate the occurrence probability of pattern q in window [1 : 6] by using our approach:

$$P_{[1:6]}(q) \simeq 0.75.$$

That is, the occurrence probability of a movement from `RoomA` to `RoomB` is about 75% over window [1 : 6]. Unlike in the case of matches, this probability helps a user to infer the occurrence of time series events.

Although we proposed the fundamental solution for a monitoring query in our previous work [14], we extend the proposed method in this paper.

- We extend the problem definition of a monitoring query to accept user-specific slide size. Moreover, we adaptively apply a slicing technique [15], [16] for efficient processing according to a slide size. (Sect. 4)
- To deal with negation in pattern queries, we propose the calculation method based on an inverted deterministic finite automaton (DFA). (Sect. 5)
- To process multiple queries simultaneously, we merge queries into one regular expression by using disjunction. Since a naïve approach cannot calculate occurrence probabilities of each query separately, we propose the two-step integration of DFAs. (Sect. 6)

This paper is constructed as follows. First, we define basic concepts to discuss the proposed method in Sect. 2. In Sect. 3, we explains a probability calculation method for time series events in a sliding window. Sections 4, 5, and 6 describe the above contributions, respectively. Then, we evaluate the proposed method by experiments in Sect. 7. We introduce the related work in Sect. 8 and conclude the paper in Sect. 9.

2. Preliminaries

In this section, we explain the basic concepts of pattern matching in a probabilistic event stream. First, we introduce the definitions of a probabilistic event stream and a query pattern. Then, we define the probability of a time series event within a window by using the possible world semantics. Finally, we define pattern monitoring queries with a sliding window.

2.1 Probabilistic Event Streams

First, we define a *probabilistic event* as a component of a probabilistic event stream.

Definition 1. A *probabilistic event* e_t is a probabilistic distribution to express the occurrence of an event at time step t. Let Σ be an universal set of event symbols. Each event symbol $\alpha \in \Sigma$ has an occurrence probability $P(e_t = \alpha)$, and the probabilities satisfy the following properties.

$$\forall \alpha \in \Sigma,\ 0 \leq P(e_t = \alpha) \leq 1 \tag{1}$$
$$\forall \alpha, \beta \in \Sigma,\ \alpha \neq \beta \rightarrow P(e_t = \alpha \wedge e_t = \beta) = 0 \tag{2}$$
$$P(\bigvee_{\alpha \in \Sigma} e_t = \alpha) = \sum_{\alpha \in \Sigma} P(e_t = \alpha) = 1 \tag{3}$$

■

In the rest of the paper, we abbreviate $e_t = \alpha$ as α_t for simplicity.

We define a *probabilistic event stream* in terms of probabilistic events.

Definition 2. A *probabilistic event stream* $PES = \langle e_1, e_2, \ldots \rangle$ is an infinite sequence of probabilistic events.

■

For instance, we can represent the probabilistic event stream in Fig. 1 as $PES = \langle e_1, e_2, e_3, e_4, e_5, e_6, e_7, \ldots \rangle$, where $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}, \mathtt{e}\}$.

2.2 Query Pattern and Matches

We define the grammar of query patterns.

Definition 3. Let α, ϵ, and "." be an event symbol in Σ, an empty symbol, and a wildcard (i.e., an arbitrary symbol), respectively. A query pattern is generated by the following grammar:

$$q ::= \alpha \mid \epsilon \mid . \mid q\ q \mid q \vee q \mid q^* \mid q^+ \mid (q) \mid \neg q \tag{4}$$

■

In other words, we assume that query patterns are specified by regular expressions [17].

We define a match as a concrete event sequence for a query pattern.

Definition 4. A *match* $m = \langle \alpha_{t_s}, \ldots, \alpha_{t_e} \rangle$ is a sequence of event symbols that fits a specified query pattern. Let $m.t_s$ and $m.t_e$ be the start and end time step of match m, respectively. The probability of match m is calculated as follows:

$$P(m) = \prod_{t=m.t_s}^{m.t_e} P(\alpha_t) \tag{5}$$

■

2.3 Probabilities of Time Series Events in a Window

We define the occurrence probability of a time series event in a certain window.

First, Definition 5 defines possible worlds of a probabilistic event stream in a window.

Definition 5. Let $PES_{[t-w+1:t]} = \langle e_{t-w+1}, e_{t-w+2}, \ldots, e_t \rangle$ be a subsequence of PES in a window, and let Σ_t be the set of event symbols that have positive occurrence probabilities at time step t. Given a probabilistic event stream PES and a window with size w, the set of possible worlds $\Omega_{[t-w+1:t]}$ is the universal set of event sequences that can occur in $PES_{[t-w+1:t]}$, and it is specified by the cartesian product of every Σ_t in a window.

$$\Omega_{[t-w+1:t]} = \Sigma_{t-w+1} \times \Sigma_{t-w+2} \times \cdots \times \Sigma_t \tag{6}$$

The probability of each possible world $\omega = \langle \alpha_{t-w+1}, \ldots, \alpha_t \rangle \in \Omega_{[t-w+1:t]}$ is calculated as follows:

$$P(\omega) = \prod_{t'=t-w+1}^{t} P(\alpha_{t'}) \tag{7}$$

■

For instance, when we apply window [1 : 3] to the probabilistic event stream in Fig. 1, the set of possible worlds $\Omega_{[1:3]}$ has 125 event sequences including:

$$\begin{aligned}
&\omega_1 = \langle \mathtt{a}_1, \mathtt{a}_2, \mathtt{b}_3 \rangle, \quad P(\omega_1) = 0.018, \\
&\omega_2 = \langle \mathtt{a}_1, \mathtt{a}_2, \mathtt{c}_3 \rangle, \quad P(\omega_2) = 0.162, \quad \text{and} \\
&\omega_3 = \langle \mathtt{a}_1, \mathtt{c}_2, \mathtt{b}_3 \rangle, \quad P(\omega_2) = 0.0045.
\end{aligned}$$

We define a probability space for a probabilistic event stream by using possible worlds.

Definition 6. Let 2^X be the power set of X. Given a windowed stream $PES_{[t-w+1:t]}$, the probability space is defined as $(\Omega_{[t-w+1:t]}, 2^{\Omega_{[t-w+1:t]}}, P)$. Note that P is a probability measure and satisfies the following properties:

$$P(\Omega_{[t-w+1:t]}) = 1 \tag{8}$$

$$X \in 2^{\Omega_{[t-w+1:t]}}, P(X) = \sum_{\omega \in X} P(\omega) \tag{9}$$

■

Definition 7 defines the occurrence probability of a time series event.

Definition 7. Let $\Omega_{[t-w+1:t]}(q)$ be the set of possible worlds that include a match of query pattern q as a subsequence. Given a windowed stream $PES_{[t-w+1:t]}$ and pattern q, the occurrence probability of a time series event of pattern q is calculated as follows:

$$\begin{aligned}
P_{[t-w+1:t]}(q) &= P(\Omega_{[t-w+1:t]}(q)) \\
&= \sum_{\omega \in \Omega_{[t-w+1:t]}(q)} P(\omega)
\end{aligned} \tag{10}$$

■

We continue the above example. Given pattern $q = \langle \mathtt{a}^+ .^* \mathtt{b}^+ \rangle$, the set of corresponding possible worlds $\Omega_{[1:3]}(q)$ includes ω_1 and ω_3. We can calculate the probability of a time series event by summing up their probabilities.

$$\begin{aligned}
P_{[1:3]}(q) &= P(\Omega_{[1:3]}(q)) \\
&= P(\omega_1) + P(\omega_3) + \cdots \\
&= 0.0705
\end{aligned}$$

2.4 Problem Definition

We define *pattern monitoring queries with a sliding window*.

Definition 8. Let PES, Q, w, and l be a probabilistic event stream, query patterns, a window size, and a sliding size, respectively. A *pattern monitoring query* requires (PES, Q, w, l) as input and continuously outputs every occurrence probability of a pattern $q_i \in Q$ at each sliding window:

$$\begin{aligned}
&\{\{P_{[1:w]}(q_1), P_{[1:w]}(q_2), \ldots\}, \\
&\{P_{[1+l:w+l]}(q_1), P_{[1+l:w+l]}(q_2), \ldots\}, \\
&\{P_{[1+2l:w+2l]}(q_1), P_{[1+2l:w+2l]}(q_2), \ldots\}, \ldots\}.
\end{aligned} \tag{11}$$

■

For example, if an input set is (PES in Fig. 1, $\{q = \langle \mathtt{a}^+ .^* \mathtt{b}^+ \rangle\}$, $w = 6, l = 1$), the proposed method calculates every probability of q at each window and outputs as follows:

$$\{\{P_{[1:6]}(q) \simeq 0.75\}, \{P_{[2:7]}(q) \simeq 0.66\}, \ldots\}. \tag{12}$$

In our previous work [14], we do not deal with a sliding size as a parameter. That is, slide size l is fixed to 1. Although we optimized the proposed method in such a setting, a sliding size is usually tuned by a user. For example, consider indoor location tracking. If a location event occurs at every second, we need window size $w = 3600$ to monitor the occurrence of a certain time series event past one hour. In such a case, however, we do not need to update a window at every second (i.e., $l = 1$). To monitor time series events, it is sufficient to update a window with longer intervals, such as at every minute (i.e., $l = 60$). We thus extend the problem definition in this paper, and optimize query processing in this setting.

3. Probability Calculation Based on Probabilistic Transition Matrices

In this section, we describe a calculation method for the probabilities of time series events in a time window. In the following, we explain enumeration-based and DFA-based approaches.

We assume that a windowed stream $PES_{[t-w+1:t]}$ is retained as an array at each time step. That is, every probabilistic event $e_t \in PES_{[t-w+1:t]}$ is accessible during calculation.

3.1 Naïve Approach

We explain the enumeration-based approach to calculate the probability of a time series event in a window. As shown in Definition 7, $P_{[t-w+1:t]}(q)$ is the sum of the probabilities of possible worlds that include a match of q as a subsequence. Thus, we can calculate $P_{[t-w+1:t]}(q)$ by enumerating all the possible worlds in a window and extracting matching ones.

However, this naïve method is inefficient. The number of possible worlds increases $O(|\Sigma|^w)$ with the universal set of event symbols Σ and window size w. As we need exponential computation time whenever a window slides, this naïve method is inappropriate for stream processing.

3.2 DFA-Based Approach

We propose an efficient calculation method based on a DFA. This method does not enumerate possible worlds but calculates the sum of their probabilities of arrival at each state in a DFA. Note that we proposed the basic idea of this method in our previous work [8], [14].

First, we construct a DFA that accepts matching possible worlds. Since each matching possible world $\omega \in \Omega_{[t-w+1:t]}(q)$ includes a match of pattern q as a subsequence, we can describe them by a regular expression $\langle .^* \, q \, .^* \rangle$. Thus, we can construct a corresponding DFA by using basic methods, such as the Thompson's construction and the subset construction [17]. For instance, Fig. 2 shows a DFA of a regular expression $\langle .^* \, q \, .^* \rangle$ with pattern $q = \langle \mathtt{a}^+ \, .^* \, \mathtt{b}^+ \rangle$.

Given a DFA that accepts matching possible-worlds $\Omega_{[t-w+1:t]}(p)$, we can avoid enumerating all the possible-worlds. When we input a probabilistic event stream to a DFA, all the possible worlds arrive at any one of states because of the deterministic property of DFAs. That is, the probability of arrival at a specific state is identical to the sum of the probabilities of arriving possible worlds. Since all the matching possible-worlds arrive at any one of the final states, the probability of arrival at the final states is the occurrence probability of a time series event.

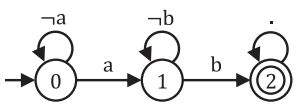


Fig. 2 DFA of $\langle .^* \, \mathtt{a}^+ \, .^* \, \mathtt{b}^+ \, .^* \rangle$

We calculate the probabilities of arrival at each state by using probabilistic transition matrices. The following matrix is an example of a probabilistic transition matrix for the DFA in Fig. 2:

$$T_p(e_t) = \begin{bmatrix} P(\neg \mathtt{a}_t) & P(\mathtt{a}_t) & 0 \\ 0 & P(\neg \mathtt{b}_t) & P(\mathtt{b}_t) \\ 0 & 0 & P(.) \end{bmatrix}. \tag{13}$$

In transition matrices, rows and columns correspond departure and arrival states, respectively. We set each transition probability by an input event. For instance, we generate the following transition matrix by using e_1 in Fig. 1:

$$T_p(e_1) = \begin{bmatrix} 0.40 & 0.60 & 0 \\ 0 & 0.95 & 0.05 \\ 0 & 0 & 1.0 \end{bmatrix}. \tag{14}$$

To calculate the probabilities of arrival at each state V_p, we initialize the probability of arrival at the initial state to 1.0, and then product transition matrices in a window. For instance, consider the calculation of $P_{[1:6]}(q)$ where the probabilistic event stream in Fig. 1 and pattern $q = \langle \mathtt{a}^+ \, .^* \, \mathtt{b}^+ \rangle$ are given. First, we initialize the probabilities of arrival at each state:

$$V_{init} = \begin{bmatrix} 1.0 & 0 & 0 \end{bmatrix}. \tag{15}$$

Next, we calculate the probabilities of arrival at each state over window $[1 : 6]$ by multiplying V_{init} and probabilistic transition matrices together:

$$V_p(PES_{[1:6]}) = V_{init} \prod_{t=1}^{6} T_p(e_t) \tag{16}$$

$$\simeq \begin{bmatrix} 0.12 & 0.13 & 0.75 \end{bmatrix}. \tag{17}$$

As $V_p(PES_{[1:6]})$ shows that the probability of arrival at the final state (i.e., state 2 in Fig. 2) is 0.75, the occurrence probability of a time series event is as follows:

$$P_{[1:6]}(q) \simeq 0.75. \tag{18}$$

When we answer pattern monitoring queries with a slicing window, we apply the DFA-based method to every query for each time window. That is, we prepare DFAs of every query and calculate the vector-matrix products for each windowed probabilistic stream. However, we can calculate the occurrence probabilities of time series events more efficiently in some parameter settings. We propose an efficient calculation for a sliding window in Sect. 4 and multiple queries in Sect. 6.

4. Adaptive Slicing for Efficient Calculation

When we use a sliding window for time series event monitoring, we need duplicate matrix products in the probability calculation. For instance, consider that (PES in Fig. 1, $\{q = \langle \mathtt{a}^+ \, .^* \, \mathtt{b}^+ \rangle\}$, $w = 5$, $l = 2$) is given. As the first and

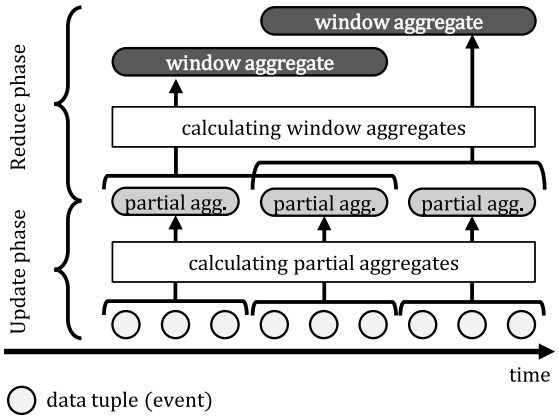


Fig. 3 Slicing a stream with slide size $l = 3$

the second windows are $[1:5]$ and $[3:7]$, respectively, we calculate the occurrence probabilities of q in these windows based on the DFA-based approach.

$$V_p(PES_{[1:5]}) = V_{init} \prod_{t=1}^{5} T_p(e_t) \tag{19}$$

$$\simeq \begin{bmatrix} 0.13 & 0.31 & 0.56 \end{bmatrix}$$

$$\therefore P_{[1:5]}(q) \simeq 0.56 \tag{20}$$

$$V_p(PES_{[3:7]}) = V_{init} \prod_{t=3}^{7} T_p(e_t) \tag{21}$$

$$\simeq \begin{bmatrix} 0.73 & 0.07 & 0.20 \end{bmatrix}$$

$$\therefore P_{[3:7]}(q) \simeq 0.20 \tag{22}$$

However, Eqs. (19) and (21) include duplicate products $\prod_{t=3}^{5} T_p(e_t)$.

Thus, we apply a *slicing technique* [15], [16] to omit redundant matrix products. A slicing technique is used for window-aggregate in data stream processing, and it consists of two phases: an update phase and a reduce phase. Figure 3 shows these phases in a slicing technique graphically. In the update phase, we slices a window into small chunks, and then computes partial aggregates incrementally at each time step. When a stream reaches the end of a certain window, the reduce phase calculates the window aggregate by using partial ones of corresponding chunks. Note that the optimal chunk size is derived by the existing work [15]; we can use slide size l as the optimal chunk size.

In the case of the above example, we slice a window into chunks with chunk size 2. That is, Eqs. (19) and (21) become the following equations, respectively.

$$V_p(PES_{[1:5]}) = V_{init} \prod_{t=1}^{2} T_p(e_t) \prod_{t=3}^{4} T_p(e_t) \prod_{t=5}^{5} T_p(e_t) \tag{23}$$

$$V_p(PES_{[3:7]}) = V_{init} \prod_{t=3}^{4} T_p(e_t) \prod_{t=5}^{6} T_p(e_t) \prod_{t=7}^{7} T_p(e_t) \tag{24}$$

The equations shows that we can reuse the results of matrix products such as $\prod_{t=3}^{4} T_p(e_t)$. To calculate these equations with a slicing technique, the update phase calculates the products between transition matrices (i.e., partial aggregates), such as $\prod_{t=3}^{4} T_p(e_t)$, at each time step. When a stream reaches the end of a certain window, such as $t = 5$ in this case, the reduce phase calculate the corresponding equation by using the calculated products of transition matrices.

4.1 Adaptive Slicing

Although a slicing technique is an useful calculation strategy, we should not use it anytime for our probability calculation. A slicing technique assumes that update of chunks and reduce of partial aggregates have the same computation cost. In our case, however, the computation costs are different. Let n be the size of an objective DFA. In the update phase, as we calculate the probabilistic transition matrix of the newest chunk, we need a matrix-matrix product with $O(n^3)$. On the other hand, we calculate vector-matrix products with $O(n^2)$ in the reduce phase. This different computation costs may make a slicing technique inefficient. In other words, if an input query construct a gigantic DFA, the DFA-based method without a slicing technique may more efficient.

Thus, we calculate the computation costs with/without a slicing technique, and then adaptively apply the most efficient one in them. We consider the computation costs are based on the number of unit operations (i.e., float multiplication operators). Let $|PES|$ be the length of a finite event stream. In the DFA-based method without a slicing technique, we calculate the arrival probabilities for all the windows at each time step. As it is a vector-matrix product, we need n^2 float multiplications for each window. Since we deal with $\lceil w/l \rceil$ windows simultaneously, the overall computation cost is as follows:

$$Cost_{naive} = \frac{wn^2}{l}|PES|. \tag{25}$$

In the slicing approach, we update one partial aggregate at each time step. As it is a matrix-matrix product, the number of float multiplications is n^3. Moreover, in the slicing approach, we need reducing of partial aggregates to output windows. Since the number of windows is $\lceil |PES|/l \rceil$ and the number of chunks is $\lceil w/l \rceil$, the entire computation cost becomes as follows:

$$Cost_{slicing} = n^3|PES| + \frac{wn^2}{l^2}|PES|. \tag{26}$$

That is, we can determine whether to use a slicing technique by using the following equation:

$$Cost_{naive} > Cost_{slicing}. \tag{27}$$

This equation is simplified as follows:

$$\frac{w}{l}\left(1 - \frac{1}{l}\right) > n. \tag{28}$$

Since all the parameters (i.e., w, l, and n) are specified by a user, we can choose an appropriate approach before starting computation.

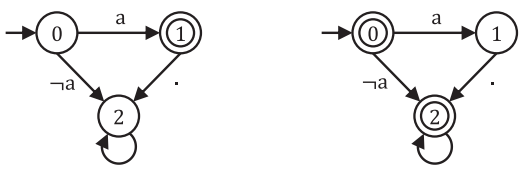
5. Probability Calculation with Negation

Since *negation* is one of the regular expressions, we can calculate the occurrence probabilities of negation patterns. In other words, we can represent negation patterns by using disjunction. A simple example is the negation of a certain event symbol. Let Σ be $\{\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}, \mathsf{e}\}$ and consider the probability of $e_t = \neg\mathsf{a}$. In this case, the negation of a is equivalent to the disjunction of all the other event symbols:

$$\neg\mathsf{a} = \mathsf{b} \vee \mathsf{c} \vee \mathsf{d} \vee \mathsf{e}. \tag{29}$$

Thus, we can calculate the probability of $e_t = \neg\mathsf{a}$ as follows:

$$P(\neg\mathsf{a}_t) = P(\mathsf{b}_t \vee \mathsf{c}_t \vee \mathsf{d}_t \vee \mathsf{e}_t). \tag{30}$$



(a) DFA of $\langle\mathsf{a}\rangle$ (b) DFA of $\langle\neg\mathsf{a}\rangle$

Fig. 4 Conversion of a DFA based on negation

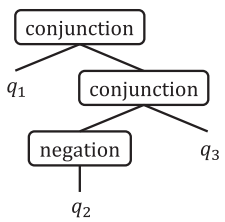
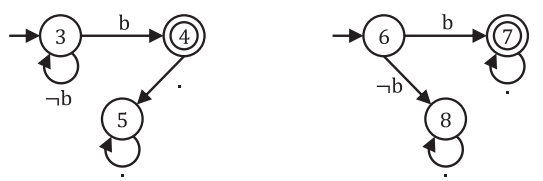
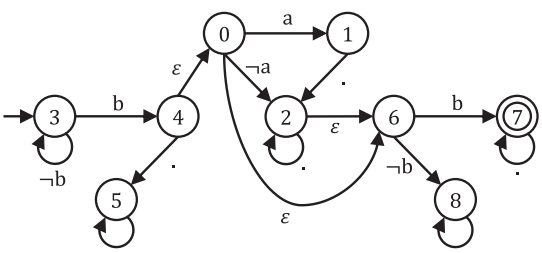


Fig. 5 Syntax tree of $q = q_1\ \neg q_2\ q_3$



(a) DFA of $q_1 = \langle .^*\ \mathsf{b}\rangle$ (b) DFA of $q_3 = \langle \mathsf{b}\ .^*\rangle$



(c) ϵ-NFA of $q = \langle .^*\ \mathsf{b}\ \neg\mathsf{a}\ \mathsf{b}\ .^*\rangle$

Fig. 6 Construction of an ϵ-NFA from Fig. 5

We can generalize the calculation of negation patterns $\neg q$ by using the negation of a DFA. If a sequence does not reach the final states of the DFA of q, such a sequence is accepted by the DFA of $\neg q$. Let us continue the above example. In the above example, the query pattern is $q = \langle\neg\mathsf{a}\rangle$. Since q is the negation of pattern $q' = \langle\mathsf{a}\rangle$, first we construct a DFA of q' as shown in Fig. 4 (a). This DFA accepts sequences that have only one event a. The other sequences, such as $\langle\mathsf{b}\rangle$ and $\langle\mathsf{a}, \mathsf{c}\rangle$, are rejected and reach the states 0 or 2. Thus, we can construct a DFA of $q = \neg q'$ by inverting the final states in Fig. 4 (a), as shown in Fig. 4 (b), and calculate the occurrence probability of $\neg q$ for any windows by using the DFA in Fig. 4 (b).

We can construct a DFA with negation by using the Thompson's construction [17]. For instance, consider a pattern $q = q_1\ \neg q_2\ q_3$, where q_1, q_2 and q_3 are subquery patterns. To apply the Thompson's construction, we generate a syntax tree as shown in Fig. 5. This syntax tree shows that we can invert the final states in the "negation" node for subquery q_2. In more detail, we consider subqueries $q_1 = \langle .^*\ \mathsf{b}\rangle$, $q_2 = \langle\mathsf{a}\rangle$, and $q_3 = \langle\mathsf{b}\ .^*\rangle$. That is, a query pattern is $q = \langle .^*\ \mathsf{b}\ \neg\mathsf{a}\ \mathsf{b}\ .^*\rangle$. We already show the DFA of $\neg q_2$ in Fig. 4 (b), and Figs. 6 (a) and (b) show corresponding DFAs for q_1 and q_3, respectively. Note that the Thompson's construction uses NFAs in its algorithm, but we use these DFAs in this example for simplicity. Since a conjunction node connects the final states in a left child with the initial state in a right child by using ϵ-transitions, we generate an ϵ-NFA in Fig. 6 (c) from the syntax tree. Thus, we convert this ϵ-NFA into a DFA by using the subset construction [17] and use the converted DFA for probability calculation.

6. Simultaneous Multiple Query Processing

In some use cases, we want to query multiple patterns simultaneously. For example, consider indoor location tracking. A certain user monitors that there is no objects in `RoomA` in the past 30 minutes. Moreover, if any object enters `RoomA`, the user may need to check whether an object visits `RoomB` without going through a certain area, such as `HallC`. These monitoring queries are represented as follows:

$$q_4 = \langle .^*\ \mathsf{a}^+\ .^*\rangle \text{ and} \tag{31}$$

$$q_5 = \langle .^*\ \mathsf{a}^+\ \neg(.^*\ \mathsf{c}^+\ .^*)\ \mathsf{b}^+\ .^*\rangle. \tag{32}$$

In order to reduce the total number of states in DFAs and achieve more efficient processing, multiple DFAs can be merged into one DFA. In the literature of XML matching, YFilter [18] proposed the use of a merged NFA for multiple XML queries with the same prefix. Here, since pattern q_4 is a prefix subsequence of q_5 and their DFAs include the same structure, we can reduce the total number of states by merging these DFAs.

Although we can merge multiple queries by using disjunction, it disables the merged DFA from calculating occurrence probabilities. Let us continue the above example. To integrate multiple DFAs, the simple solution is to de-

scribe original regular expressions as one regular expression by using disjunction. In the above case, we can construct a merged DFA from the disjunction of q_4 and q_5 (i.e., $q_4 \vee q_5$). Such a DFA, however, does not distinguish accepted sequences. That is, as a merged DFA accepts all the sequences for q_4 and q_5, the calculated probability does not distinguish the occurrence of q_4 from that of q_5.

To distinguish the occurrence probabilities of multiple queries, we construct a merged DFA in two steps: 1) merging DFAs of each query $q \in Q$ by using ϵ-transitions and 2) converting the merged ϵ-NFA into a DFA while distinguishing the final states of every query. That is, we hold the correspondence between the final states in a merged DFA and those of original DFAs. Note that we follow the disjunction rule of the Thompson's construction [17] to merge DFAs in the first step, and it makes a merged automaton non-deterministic. For instance, we continue the above example with q_4 and q_5. Figure 7 shows a merged ϵ-NFA of q_4 and q_5. States $\{1, 2\}$ and $\{3, 4, 5, 6\}$ correspond to q_4 and q_5, respectively. These states are connected via state 0 and ϵ-transitions. We can translate an ϵ-NFA into an equivalent DFA, as shown in Fig. 8, by reducing ϵ-transitions and merging states [17]. Note that each state in Fig. 8 represents the subset of states in Fig. 7; state $\{2, 4\}$ in Fig. 8 is equivalent to states 2 and 4 in Fig. 7. That is, we can distinguish each final state of the original DFAs in the merged one. In this case, since all the final states in Fig. 8 includes state 2 (i.e., the final state of q_4), the occurrence probability of q_4 is the sum of the probabilities of arrival at states $\{2, 4\}, \{2, 5\}$, and $\{2, 6\}$. On the other hand, as only state $\{2, 6\}$ includes state 6 (i.e., the final state of q_5), the occurrence probability of q_5 is the probability of arrival at state $\{2, 6\}$. If we calculate the probabilities of arrival at each state in Fig. 8 with PES in Fig. 1 and window $[1 : 6]$, the results are as follows:

$$V_p(PES_{[1:6]}) \simeq \begin{bmatrix} \{0,1,3\} & \{2,4\} & \{2,5\} & \{2,6\} \\ 0.12 & 0.03 & 0.61 & 0.25 \end{bmatrix}. \quad (33)$$

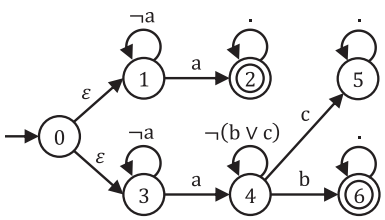


Fig. 7 Merged ϵ-NFA of $q_4 = \langle .^* \, \mathsf{a}^+ \, .^* \rangle$ and $q_5 = \langle .^* \, \mathsf{a}^+ \, \neg(.^* \, \mathsf{c}^+ \, .^*) \, \mathsf{b}^+ \, .^* \rangle$

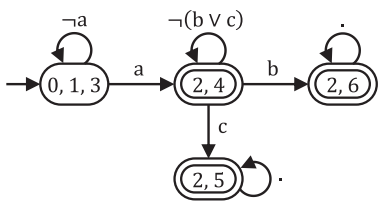


Fig. 8 Equivalent DFA of Fig. 7

Thus, we can calculate the probabilities of q_4 and q_5:

$$P_{[1:6]}(q_4) \simeq 0.03 + 0.61 + 0.25 = 0.89 \text{ and} \quad (34)$$

$$P_{[1:6]}(q_5) \simeq 0.25. \quad (35)$$

Note that simultaneous query processing is not always efficient. In the above example, since query q_5 includes q_4 as a prefix subsequence, we can construct a compressed DFA. However, if queries have different prefix, the size of a merged DFA may become larger than those of original DFAs. Thus, we need to check whether a merging DFA reduces the number of states before starting computation.

7. Experiments

In this section, we evaluate the effectiveness of our approach by experiments. To evaluate our approach, we have implemented the proposed method by using Java language. Our experimental settings are summarized in Table 1.

7.1 Effect of Window-Based Detection

We evaluate the effectiveness of our approach by using the real dataset. The real dataset is indoor location event streams that are collected in the RFID ecosystem project [19]. In this dataset, an indoor space is represented as a graph, and the locations (i.e., nodes in a graph) of a tracked object are estimated by RFID sensing data at every second. Since the dataset includes definite event streams (i.e., truth locations), we can detect correct time series events from them. Note that the dataset has twelve probabilistic event streams, but we omit the results of eleven streams in the paper because they have the same tendency with introduced one.

In this experiments, we use pattern $q = \langle (\mathsf{a} \vee \mathsf{b})^{3+} \rangle$ with window size $w = 30$ and slide size $l = 1$ where $(\mathsf{a} \vee \mathsf{b})$ indicates that a tracked object is in a certain room. Note that α^{k+} means k or more occurrence of event α. Hence, the above query means that a tracked object is in a certain room over 3 seconds in the last 30 seconds.

We compare the proposed method (i.e., window-based detection) with the match-based detection [4]–[7] and the suffix-based detection [3]. In the match-based detection, we detect a match with the maximum occurrence probability in each window $[t - w + 1 : t]$ by using the existing method [7], and then output its probability as the probability of a time series event. On the other hand, Rè et al. [3] calculates the occurrence probabilities of time series events at each time step. Since we can calculate these probabilities by using a suffix pattern $\langle .^* \, q \rangle$ in our method, we show them as the suffix-based detection.

Table 1 Experimental settings

Item	Value
OS	Ubuntu 18.04.2 LTS
CPU	Intel(R) Xeon(R) CPU E5-2637 v4
RAM	128GB
JDK	Oracle OpenJDK 12.0.1

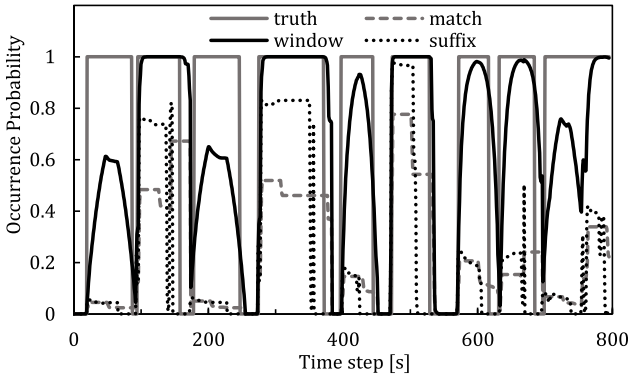


Fig. 9 Estimated occurrence probabilities

Table 2 Root mean square error

Method	RMSE
Match-based	0.677
Suffix-based	0.688
Window-based	0.369

Our approach can detect time series events with high probabilities. Figure 9 shows the accuracy of our approach graphically. A gray solid line shows truth probabilities; that is, a time series event occurs in a window when the line reaches the top. Gray and black dotted lines are the results of the match-based and the suffix-based detection, respectively. As the probabilities of the existing methods are small, we probably miss the occurrence of time series events, such as the first room entry/exit event around [20 : 60]. In contrast to the existing methods, our approach (i.e., a black solid line) can detect time series events with high probabilities. Besides, as Table 2 shows the root mean square error (RMSE) between the truth probabilities and the estimated ones, the RMSE error significantly decreases by using the proposed method.

7.2 Effect of Parameters

In this subsection, we evaluate the effect of the parameters. Note that we do not evaluate probability calculation for negation patterns (described in Sect. 5) because it does not affect efficiency. In addition, we also omit the evaluation of simultaneous query processing (described in Sect. 6) because it affects only DFA size n. That is, the effect of simultaneous query processing depends on the number of reduced states by merging multiple DFAs.

In this experiments, we use a synthetic dataset. That is, we generate a probabilistic event stream by simulation. The stream has a million probabilistic events (i.e., $|PES| = 10^6$) and each probabilistic event has a hundred event symbols (i.e., $|\Sigma| = 100$). We use natural numbers as event symbols (i.e., $\Sigma = \{1, 2, \ldots, 100\}$) and set their occurrence probabilities randomly. Since we prepare the synthetic dataset in memory, the following experimental results do not include storage I/O. Note that we use random occurrence probabilities as the worst-case setting. If the occurrence probabilities of some events are zero, the throughputs may increase because transition matrices may be sparser. However, it does not affect the experimental results significantly.

Table 3 Parameter settings

Parameter	Range
window size: w	{500, 1000, 1500, 2000, 2500, **3000**, 3500, 4000, 4500, 5000}
slide size: l	{25, **50**, 75, 100, 125, 150, 175, 200, 225, 250}
DFA size: n	{11, 21, **31**, 41, 51, 61, 71, 81, 91, 101}

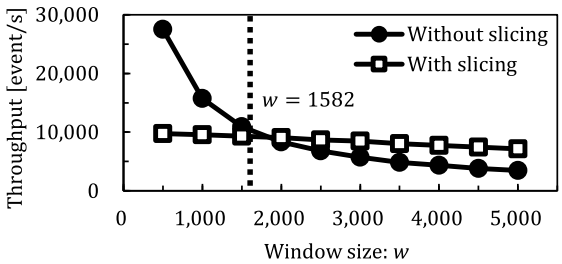


Fig. 10 Throughputs over different window size w

In this experiment, we measure the throughputs by using different values of each parameter. We summarize parameter settings in Table 3 and the default parameter values are displayed in bold. Each DFA is constructed from simple regular expressions. To construct a DFA with a specific size, we increase the number of natural numbers $i \in \Sigma$ in regular expression $\langle .^* \; i^+ \; .^* \rangle$. This regular expression generates a DFA with size $n = i + 1$. For instance, we use $q = \langle .^* \; 1^+ \; 2^+ \; 3^+ \; \ldots \; 10^+ \; .^* \rangle$ to construct a DFA with size $n = 11$. Note that the throughputs of our approach depends on the number of states in a DFA as described in Eqs. (25) and (26). Although more complex patterns may generate complex DFAs, we can estimate how such DFAs affect the efficiency from the number of their states.

We compare the throughputs of the DFA-based method with/without a slicing technique. Since the naïve method, as described in Sect. 3.1, enumerates the exponential number of possible worlds, it takes too long running time. Thus, we omit it from the experiment. In the following, we denote "without slicing" and "with slicing" to specify the results of the DFA-based method without/with a slicing technique.

Figure 10 shows throughputs over different window sizes. When we do not use a slicing technique, window size w significantly affects the efficiency. On the other hand, a slicing technique makes throughput stable. As described in Eqs. (25) and (26), w affects computation cost linearly. However, when we set slide size l to 50, a major bottleneck with slicing is matrix-matrix products (i.e., the first term in Eq. (26)), as described in the next paragraph. That is, since the effect of w is minor in this parameter settings, the throughput of a slicing technique is stable over different window sizes. Note that a dotted line in Fig. 10 shows the estimated switching point for slicing, but we explain the detail in the next subsection.

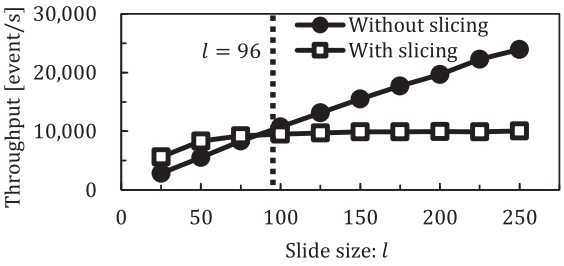


Fig. 11 Throughputs over different slide size l

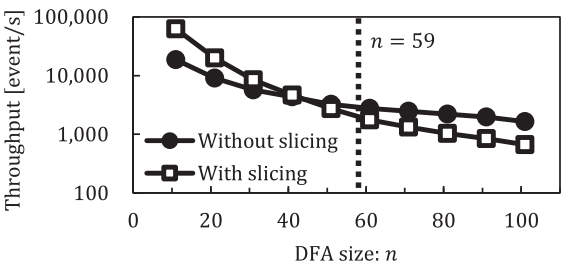


Fig. 12 Throughputs over different DFA size n

Figure 11 shows throughputs over different slide sizes. When we do not use a slicing technique, the throughput improves linearly. However, while the throughput with slicing increases in small slide sizes ($l \leq 75$), it becomes stable over large slide sizes ($l > 75$). As described above, slide size l reduces the computation cost. Although the effect of l increases by the slicing technique, matrix-matrix products (the first term in Eq. (26)) becomes a major bottleneck with large slide sizes. Thus, the throughput with slicing does not change in large slide sizes.

Figure 12 shows throughputs over different DFA sizes. With the slicing technique, DFA size has more impact on throughput because we need matrix-matrix products. On the other hand, the throughput without slicing more moderately decreases as DFA size increases because it uses only vector-matrix products.

7.3 Effect of Adaptive Slicing

Next, we evaluate the effect of adaptive slicing. As described in Sect. 4, we can determine whether to use a slicing technique by using the proposed cost functions. In this subsection, we check the appropriateness of the cost functions experimentally.

In Figs. 10, 11, and 12, dotted lines show the estimated switching points for slicing. That is, we calculate each value by using the following equation:

$$\frac{w}{l}\left(1 - \frac{1}{l}\right) = n, \tag{36}$$

where we set the default values to non-varying parameters and round up the estimated values.

Each result shows that our cost functions work satisfactorily for adaptive slicing. Although the switching points are different from intersection points of line graphs, the difference is slight. This difference is caused from the sparsity of transition matrices. As described above, we use simple regular expressions to construct DFAs. Since such regular expressions generate simple DFAs, transition matrices are sparse. Thus, the real computation cost without slicing becomes smaller than the estimated one. On the other hand, we calculate the products of transition matrices with a slicing technique. As the multiplied matrices become denser, the difference between the real computation and the estimated one becomes smaller.

8. Related Work

We introduce the exiting methods that detect time series events from probabilistic event streams. Note that event recognition in probabilistic even streams [11] is divided into automaton-based methods [3]–[9] and rule-based methods [20], [21], but we describe only the former because our research is closely related to automata.

Almost all the existing methods try to detect appropriate matches from probabilistic event streams [4]–[7], [9]. To detect appropriate matches, the existing methods use two criteria: an occurrence probability and the deviation of information amounts. The methods of Shen et al. [4], Kawashima et al. [5], Wang et al. [6] are probabilistic extension of the SASE's approach [12], [22], which proposed a detection method of time series events in definite event streams, and use the occurrence probabilities of matches as a criterion. Li et al. [7] follow this approach and propose the efficient top-k pattern matching method with a sliding widow. On the other hand, one of our previous work [9] shows that the occurrence probabilities of matches become a misleading criterion with Kleene closure (i.e., q^* and q^+), and propose the use of the deviation of information amounts for alternative criteria.

On the other hand, some existing methods consider the probabilities of time series events. Rè et al. [3] calculate the occurrence probabilities of time series events at each time step. However, since they consider the occurrence of a time series event not within window $[t - w + 1 : t]$ but just at time step t, their method is insufficient to answer monitoring queries with a sliding window. Li et al. [7] also discussed the calculation of the probability of a time series event by using top-k matches, but calculated probabilities are approximate values and need exponential computation time. In our previous work [8], we proposed a grouping (i.e., clustering) method of matches and calculated the probabilities of time series events from grouped matches. However, our previous method assumes batch processing and is not suitable for stream processing.

In comparison with these existing methods, the proposed method can monitor time series events accurately with a sliding window. Moreover, the proposed method can monitor time series events efficiently by using a slicing technique and simultaneous multiple query processing.

9. Conclusion

In this paper, we proposed the calculation method for pattern monitoring queries with a sliding window. To calculate probabilities efficiently, our method adaptively choose the use of a slicing technique based on the proposed cost functions. Moreover, we proposed the two step integration of DFAs for simultaneous multiple query processing. We also described the probability calculation with negation patterns, and it enable a user to express query patterns more flexibly. We evaluated the effectiveness of our approach by the experiments based on the real and the synthetic datasets.

Our future work includes the introduction of more complex conditions for pattern matching. In this paper, we assume that a user specifies his/her monitoring queries by using only regular expressions. However, as described in the existing work in the literature of definite event streams [13], [22], various conditions based on declarative language are useful for complex event recognition. Although Agrawal et al. [22] deal with such complex conditions by using the proposed NFA^b, we cannot apply their approach directly because a DFA is required to calculate probabilities in probabilistic event streams.

Acknowledgments

This study was partly supported by KAKENHI (18H06461, 16H01722).

References

[1] Z.S. Abdallah, M.M. Gaber, B. Srinivasan, and S. Krishnaswamy, "Activity recognition with evolving data streams: A review," ACM Comput. Surv., vol.51, no.4, pp.71:1–71:36, 2018.
[2] J. Xiao, Z. Zhou, Y. Yi, and L.M. Ni, "A survey on wireless indoor localization from the device perspective," ACM Comput. Surv., vol.49, no.2, pp.25:1–25:31, 2016.
[3] C. Ré, J. Letchner, M. Balazinska, and D. Suciu, "Event queries on correlated probabilistic streams," Proc. SIGMOD, pp.715–728, 2008.
[4] Z. Shen, H. Kawashima, and H. Kitagawa, "Lineage-based probabilistic event stream processing," Proc. 9th Int. Conf. Mobile Data Management Workshops (MDMW), pp.106–113, 2008.
[5] H. Kawashima, H. Kitagawa, and X. Li, "Complex event processing over uncertain data streams," Proc. 5th Int. Conf. P2P Parallel Grid Cloud Internet Comput. (3PGCIC), pp.521–526, 2010.
[6] Y.H. Wang, K. Cao, and X.M. Zhang, "Complex event processing over distributed probabilistic event streams," Comput. Math. Appl., vol.66, no.10, pp.1808–1821, 2013.
[7] Z. Li, T. Ge, and C.X. Chen, "ε-matching: Event processing over noisy sequences in real time," Proc. SIGMOD, pp.601–612, 2013.
[8] K. Sugiura, Y. Ishikawa, and Y. Sasaki, "Grouping methods for pattern matching over probabilistic data streams," IEICE Trans. Inf. & Syst., vol.E100.D, no.4, pp.718–729, 2017.
[9] K. Sugiura and Y. Ishikawa, "Top-k pattern matching using an information-theoretic criterion over probabilistic data streams," Proc. APWeb-WAIM Joint Conf. Web Big Data, LNCS 10366, pp.511–526, 2017.
[10] G. Cugola and A. Margara, "Processing flows of information: From data stream to complex event processing," ACM Comput. Surv., vol.44, no.3, pp.15:1–15:62, 2012.
[11] E. Alevizos, A. Skarlatidis, A. Artikis, and G. Paliouras, "Probabilistic complex event recognition: A survey," ACM Comput. Surv., vol.50, no.5, pp.71:1–71:31, 2017.
[12] E. Wu, Y. Diao, and S. Rizvi, "High-performance complex event processing over streams," Proc. SIGMOD, pp.407–418, 2006.
[13] A. Demers, J. Gehrke, and B. Panda, "Cayuga: A general purpose event monitoring system," Proc. CIDR, pp.412–422, 2007.
[14] K. Sugiura and Y. Ishikawa, "Regular expression pattern matching with sliding windows over probabilistic event streams," Proc. 6th IEEE Int. Conf. Big Data Smart Comput. (BigComp), pp.1–8, 2019.
[15] P. Carbone, J. Traub, A. Katsifodimos, S. Haridi, and V. Markl, "Cutty: Aggregate sharing for user-defined windows," Proc. CIKM, pp.1201–1210, 2016.
[16] J. Traub, P. Grulich, A.R. Cuéllar, S. Breß, A. Katsifodimos, T. Rabl, and V. Markl, "Efficient window aggregation with general stream slicing," Proc. EDBT, pp.97–108, 2019.
[17] J.E. Hopcroft, R. Motwani, and J.D. Ullman, "Introduction to Automata Theory, Languages, and Computation, 2nd ed.," Addison Wesley, vol.32, no.1, 2001.
[18] Y. Diao, P. Fischer, M.J. Franklin, and R. To, "YFilter: Efficient and scalable filtering of XML documents," Proc. ICDE, pp.341–342, 2002.
[19] The RFID Ecosystem Project - Projects - University of Washington, CSE: https://rfid.cs.washington.edu/projects.html (accessed: Sept. 10, 2018).
[20] A. Skarlatidis, A. Artikis, J. Filippou, and G. Paliouras, "A probabilistic logic programming event calculus," Theory Pract. Logic Program., vol.15, no.2, pp.213–245, 2015.
[21] G. Cugola, A. Margara, M. Matteucci, and G. Tamburrelli, "Introducing uncertainty in complex event processing: Model, implementation, and validation," Computing, vol.97, no.2, pp.103–144, 2015.
[22] J. Agrawal, Y. Diao, D. Gyllstrom, and N. Immerman, "Efficient pattern matching over event streams," Proc. SIGMOD, pp.147–160, 2008.

Kento Sugiura is a research associate in Graduate School of Informatics, Nagoya University. He received the B.S., M.S., and Ph.D. degrees from Nagoya University in 2013, 2015, and 2018, respectively. His research interests include data stream processing, uncertain data management, and spatio-temporal data processing. He is a member of DBSJ, IPSJ, and ACM.

Yoshiharu Ishikawa is a professor in Graduate School of Informatics, Nagoya University. His research interests include spatio-temporal databases, mobile databases, scientific databases, data mining, and Web information systems. He is a member of the Database Society of Japan, IPSJ, IEICE, JSAI, ACM, and IEEE Computer Society.