Grouping Methods for Pattern Matching in Probabilistic Data Streams

Kento Sugiura[1(✉)], Yoshiharu Ishikawa[1], and Yuya Sasaki[2]

[1] Graduate School of Information Science, Nagoya University, Nagoya, Japan
[2] Institute of Innovation for Future Society, Nagoya University, Nagoya, Japan
{sugiura,yuya}@db.ss.is.nagoya-u.ac.jp, ishikawa@is.nagoya-u.ac.jp

Abstract. In recent years, *complex event processing* has attracted considerable interest in research and industry.*Pattern matching* is used to find complex events in data streams. In probabilistic data streams, however, the system may find multiple matches in a given time interval. This may result in inappropriate matches, because multiple matches may correspond to a single event. We therefore propose *grouping methods* of matches for probabilistic data streams, and call such merged matches a *group*. We describe the definitions and generation methods of groups, propose an efficient approach for calculating an occurrence probability of a group, and compare the proposed approach with a naïve one by experiment. The results demonstrate the properties and effectiveness of the proposed method.

Keywords: Complex event processing · Pattern matching · Grouping · Probabilistic data streams

1 Introduction

In recent years, *complex event processing* (CEP) has been a topic of great interest in research and industry. *Pattern matching* is of particular interest because of its usefulness [2–4,10–14,16,19]. The majority of the existing research, however, does not consider data source uncertainty. Data sources such as sensor devices are uncertain because they may contain measurement error, communication error, or both. A data stream has a probabilistic nature when the data source is uncertain. Figure 1 shows an example of a stream that corresponds to an uncertain data source. We call such a data stream a *probabilistic data stream*. In our research, we investigate pattern matching in probabilistic data streams.

However, pattern matching in probabilistic data streams is difficult because the system may find multiple matches in a given interval. For example, Fig. 2 shows results of pattern matching over the stream in Fig. 1 when the pattern $\langle \mathtt{a}\ \mathtt{b}^{+}\mathtt{c} \rangle$ is given. Methods presented in the existing research remove such matches with low probability because such matches are not important [9]. Such an approach, however, may not be appropriate because every match implies the possibility that the pattern occurred in the interval. We therefore propose

M. Renz et al. (Eds.): DASFAA 2015, Part I, LNCS 9049, pp. 92–107, 2015.
DOI: 10.1007/978-3-319-18120-2_6

time		1	2	3	4	5	6
event	a	1.0	0.3	0.1	0.1	0	0
	b	0	0.7	0.8	0.7	0.9	0
	c	0	0	0.1	0.2	0.1	1.0

Fig. 1. A probabilistic data stream

match	time						probability
	1	2	3	4	5	6	
m_1	a	b	c				0.07
m_2	a	b	b	b	b	c	0.3528
m_3		a	b	b	c		0.0168
m_4				a	b	c	0.09

Fig. 2. Pattern matching result for pattern $\langle \mathtt{a}\ \mathtt{b}^{+}\mathtt{c} \rangle$

grouping methods for matches in a given interval. We call such a set of matches a *group*. For example, we merge all matches in Fig. 1 into one group and calculate the probability that the pattern $\langle \mathtt{a}\ \mathtt{b}^{+}\mathtt{c} \rangle$ exists in the time interval $[1, 6]$.

The remainder of the paper is organized as follows. In Sect. 2, we describe the background of our research. Section 3 describes the definition of a group and Sect. 4 explains how to generate groups. In Sect. 5, we introduce an effective approach for calculating probabilities of groups. Section 6 describes the settings and results of experiments. Section 7 introduces related work and Sect. 8 concludes the paper.

2 Preliminaries

Assumptions. We make two assumptions here:

1. Each event occurs every unit time and arrives in a data stream engine in order.
2. A probability of an event at time t_i is independent of that of an event at time t_j $(i \neq j)$.

For example, in the probabilistic data stream in Fig. 1, probability $P(\mathtt{a}_2 \wedge \mathtt{b}_3) = P(\mathtt{a}_2) \times P(\mathtt{b}_3) = 0.24$ according to the second assumption.

Probability Space. We first define a *probabilistic event* as an entry of a probabilistic data stream.

Definition 1. *A* probabilistic event e_t *is an event with its probability. The probability that the value of* e_t *is* α *is denoted as* $P(e_t = \alpha)$*. For a discrete domain of events* V*, the properties*

$$\forall \alpha \in V,\ 0 \leq P(e_t = \alpha) \leq 1$$

and

$$\sum_{\alpha \in V} P(e_t = \alpha) = 1$$

hold.

For example, in Fig. 1 the occurrence probability of e_3 is $\sum_{\alpha \in \{\mathtt{a},\mathtt{b},\mathtt{c}\}} P(e_3 = \alpha) = P(\mathtt{a}_3) + P(\mathtt{b}_3) + P(\mathtt{c}_3) = 1$. We may use $P(\alpha_t)$ as a shorthand of $P(e_t = \alpha)$.

Next, we define a *probabilistic data stream* in our research.

Definition 2. *A* probabilistic data stream $PDS = \langle e_1, e_2, ..., e_t, ... \rangle$ *is a sequence of probabilistic events.*

For instance, the probabilistic data stream in Fig. 1 is represented by $PDS = \langle e_1, e_2, e_3, e_4, e_5, e_6 \rangle$.

Then, we define the notion of *sequence* $s_{[t_i,t_j]}$.

Definition 3. $s_{[t_i,t_j]} = \langle \alpha_{t_i}, \alpha_{t_i+1}, ..., \alpha_{t_j} \rangle$ *is a sequence of events from* t_i *to* t_j*. A probability of* $s_{[t_i,t_j]}$ *is defined as the product of the probabilities of the events in* $s_{[t_i,t_j]}$*:* $P(s_{[t_i,t_j]}) = \prod_{k=t_i}^{t_j} P(e_k = \alpha_k)$.

For example, one of the sequences in Fig. 1 is $s_{[1,3]} = \langle \mathtt{a}_1, \mathtt{a}_2, \mathtt{b}_3 \rangle$ and the probability of $s_{[1,3]}$ is $P(s_{[1,3]}) = P(\mathtt{a}_1) \times P(\mathtt{a}_2) \times P(\mathtt{b}_3) = 0.24$.

If a *window* is specified as $w = [t_i, t_j]$, we denote $s_{[t_i,t_j]}$ as s_w. In addition, we represent the universal set of s_w as S_w. For instance, Fig. 1 is a data stream for the window $w = [1, 6]$ and examples of the elements of S_w are $\langle \mathtt{a}_1, \mathtt{a}_2, \mathtt{a}_3, \mathtt{a}_4, \mathtt{a}_5, \mathtt{a}_6 \rangle$ and $\langle \mathtt{a}_1, \mathtt{a}_2, \mathtt{a}_3, \mathtt{a}_4, \mathtt{a}_5, \mathtt{b}_6 \rangle$.

Next, we define a probability space using S_w.

Definition 4. *Given a window* w*,* $(2^{S_w}, P)$ *is the probability space for a probabilistic data stream, where* 2^{S_w} *is the power set of* S_w*.* P *gives a probability* $P(x)$ *to each element* $x \in 2^{S_w}$ *by summing the probabilities of all sequences in* x *such as* $P(x) = \sum_{s_w \in x} P(s_w)$.

Query Pattern. We use a *regular expression* for representing a query pattern. For example, we use $\langle \mathtt{a\ b\ c} \rangle$ if we want to find matches that include a, b, and c with this order: events a, b, and c must be contiguous in the stream. In a query pattern, we can use the Kleene plus ($^+$) as an option for the regular expression. For instance, for the pattern $\langle \mathtt{a\ b}^+\mathtt{c} \rangle$, we accept matches such as $\langle \mathtt{a}_t\ \mathtt{b}_{t+1}\ \mathtt{c}_{t+2} \rangle$ and $\langle \mathtt{a}_t\ \mathtt{b}_{t+1}\ \mathtt{b}_{t+2}\ \mathtt{c}_{t+3} \rangle$.

Matches. A *match* is an instance of a pattern found in the target probabilistic stream. For example, in Fig. 1 one of the matches for the pattern $\langle \mathtt{a\ b\ c} \rangle$ is $\langle \mathtt{a}_1\ \mathtt{b}_2\ \mathtt{c}_3 \rangle$. We define the notion of a match and its probability in a consistent manner with the probability space.

Definition 5. *A match* m *is a set of sequences that include the pattern occurrence as a subsequence. A probability of match* m *is given as* $P(m) = \sum_{s_w \in m} P(s_w)$.

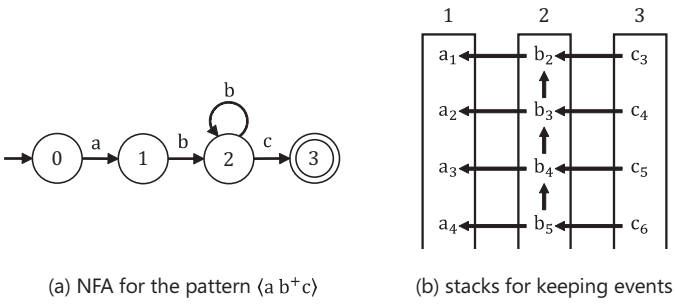

Fig. 3. An NFA and stacks for generating matches in the stream in Fig. 1

For instance, we consider the probability of $m_1 = \langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{c}_3 \rangle$ in Fig. 2. Suppose the window $w = [1, 4]$ is specified for the stream in Fig. 1. In this case, S_w holds sequences such as $\langle \mathtt{a}_1, \mathtt{a}_2, \mathtt{a}_3, \mathtt{a}_4 \rangle$ and $\langle \mathtt{a}_1, \mathtt{a}_2, \mathtt{a}_3, \mathtt{b}_4 \rangle$. In S_w, there are three sequences that include m_1:

$$s_1 = \langle \mathtt{a}_1, \mathtt{b}_2, \mathtt{c}_3, \mathtt{a}_4 \rangle$$
$$s_2 = \langle \mathtt{a}_1, \mathtt{b}_2, \mathtt{c}_3, \mathtt{b}_4 \rangle$$
$$s_3 = \langle \mathtt{a}_1, \mathtt{b}_2, \mathtt{c}_3, \mathtt{c}_4 \rangle$$

Thus, the probability of m_1 is $P(m_1) = P(s_1) + P(s_2) + P(s_3) = 0.07$.

We follow the NFA-based approach to generate matches [1]. This approach represents a pattern as a non-deterministic finite automaton (NFA) and manages events and matches using stacks. For example, Fig. 3 shows an NFA and stacks for generating matches over the probabilistic data stream in Fig. 1 for the pattern $\langle \mathtt{a}\ \mathtt{b}^+\mathtt{c} \rangle$. The stacks correspond to the respective states of the NFA and store each event that has an occurrence probability. In this example, stack 1 stores events $\{\mathtt{a}_1, \mathtt{a}_2, \mathtt{a}_3, \mathtt{a}_4\}$ and does not contain $\{\mathtt{a}_5, \mathtt{a}_6\}$ because their probabilities are 0. We connect the events using pointers according to the edges of the NFA. We can generate matches by tracing the pointers from the events in the stack of the final state. For example, $\langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{b}_3\, \mathtt{c}_4 \rangle$ and $\langle \mathtt{a}_2\, \mathtt{b}_3\, \mathtt{c}_4 \rangle$ are generated by tracing the pointers from $\mathtt{c}_4$. In the following, we call a candidate of matches under construction a *run*.

3 Grouping Policies

In our framework, a group is defined by a *grouping policy*. In this section, we introduce two policies. Intuitively, it is natural to merge matches in a given time interval into a group. Thus, we consider the time intervals of matches to decide whether to merge them. The time interval of a match is given by its start and end times. For example, the time interval of m_1 in Fig. 2 is $[1, 3]$.

For considering grouping policies, we use the complete link method and the single link method in hierarchical clustering [6]. The complete link method

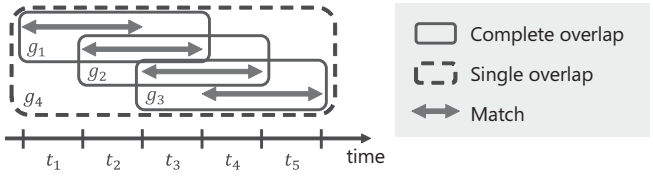

Fig. 4. Group generation based on complete overlap and single overlap

requires that every document is similar to (linked to) all other documents in the same cluster. In contrast, the single link method requires that every document is similar to at least one other document in the same cluster. For our context, we propose *complete overlap* and *single overlap*, inspired by the two methods, and give their definitions below. In the following definition, $ts\_overlap(m, m')$ is a predicate that is true when the time interval of m overlaps with that of m'.

Definition 6. *A set of matches M has a property of* complete overlap *when M satisfies the following condition:*

$$\forall m, m' \in M, ts\_overlap(m, m'). \tag{1}$$

Definition 7. *A set of matches M has a property of* single overlap *when M satisfies the following condition:*

$$\forall m \in M, \exists m' \in M, m \neq m' \wedge ts\_overlap(m, m'). \tag{2}$$

Now, we define a group using overlaps:

Definition 8. *A group g is a set of matches. g should have a property of complete overlap or single overlap and g should not be a subset of other groups. A group is represented as a tuple $g = (t_s, t_e, p)$ that contains the starting time, the end time, and the corresponding probability.*

Complete overlap ensures that all matches in a group overlap with each other. In contrast, single overlap ensures that each match overlaps with at least one other match in the same group. Figure 4 shows an example of group generation using complete overlap and single overlap. For the case of complete overlap, three groups $g_1 = (t_1, t_3, p_1), g_2 = (t_2, t_4, p_2)$, and $g_3 = (t_3, t_5, p_3)$ are generated. In contrast, one group $g_4 = (t_1, t_5, p_4)$ is generated for the case of single overlap.

The above example shows a tendency of group generation based on complete overlap and single overlap. In this case, complete overlap may generate more useful groups in general, because single overlap excessively merges matches. Suppose the first and last matches have high occurrence probability in Fig. 4. We should distinguish among them in such a case, but single overlap merges them into g_4. On the other hand, complete overlap can distinguish among them as g_1 and g_3. Single overlap is, however, more useful than complete overlap in some ways. For example, it may be appropriate to merge all the matches in Fig. 2 but complete overlap cannot merge them. In contrast, single overlap can merge all the

matches into a group. Therefore, we should selectively use complete or single overlap according to the usage scenario.

Next we define the probability of a group. A group is a set of matches according to Definition 8. Moreover, a match is a set of sequences according to Definition 5. A group is therefore also a set of sequences, so we can define the probability of a group as follows:

Definition 9. *A probability of a group g is given as*

$$P(g) = P\left(\bigcup_{m_i \in g} m_i\right) = \sum_{s_w \in \bigcup_{m_i \in g} m_i} P(s_w). \tag{3}$$

In the following, we use the term *group probability* for simplicity.

4 Algorithms for Generating Groups

In this section, we explain how to generate groups for each type of overlap. Moreover, we introduce the use of a probability threshold for efficient group generation.

4.1 The Case of Complete Overlap

As described in Definition 6, complete overlap requires that all matches overlap with each other. Thus, small groups such as g_1, g_2, and g_3 in Fig. 4 are generated. We can identify such groups when a group finds a match for the first time. For example, we consider the groups $g_1 = \{m_1, m_2, m_3\}$, and $g_2 = \{m_2, m_3, m_4\}$ in Fig. 2. g_1 does not have m_4 because m_4 does not overlap with m_1. In other words, all matches in g_1 are fixed when we detect m_1 at time 3. In more detail, g_1 has $\langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{c}_3 \rangle, \langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{b}_3 \rangle$, and $\langle \mathtt{a}_2\, \mathtt{b}_3 \rangle$ at time 3, and any runs and matches are not added to g_1 after time 3 due to the condition of complete overlap. Thus, we can distinguish g_1 and the other groups such as g_2 at time 3.

Figure 5 shows the algorithm for generating groups based on complete overlap. Note that we omit explanation of lines 10 and 15 in this section; they are covered in Sect. 5. Suppose the pattern $\langle \mathtt{a\ b\ c} \rangle$ is given for the stream in Fig. 1. First, we initialize R and G (lines 2 and 3). R is a temporal set of runs and G holds the candidates of groups. We process the events in order (line 4) and add new runs to R to generate candidates. At time 1, a new run $r_1 = \langle \mathtt{a}_1 \rangle$ is generated and added to R (line 5). The conditions at lines 6, 11, and 14 are not satisfied in this iteration. Then R becomes $\{r_1 = \langle \mathtt{a}_1\, \mathtt{b}_2 \rangle, r_2 = \langle \mathtt{a}_2 \rangle\}$ at time 2 and $\{m_1 = \langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{c}_3 \rangle, r_2 = \langle \mathtt{a}_2\, \mathtt{b}_3 \rangle, r_3 = \langle \mathtt{a}_3 \rangle\}$ at time 3 at line 5. As we find a match m_1, we generate a copy of R as g_1 and add g_1 to G (line 7). Hereafter, we update only $\{r_2, r_3\}$, the remaining runs of g_1 (line 5). Note that we remove $m_1 = \langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{c}_3 \rangle$ from R (line 8) because R cannot get a new run like $r_4 = \langle \mathtt{a}_4 \rangle$ for the condition of complete overlap. We output groups and remove them from G when they have no runs (lines 16 and 17). In this example, g_1 is output at

```
 1: procedure GenerateGroupsForCompleteOverlap(PDS)
 2:   R = ∅ // set of runs
 3:   G = ∅ // candidates of groups
 4:   for each e_t ∈ PDS do
 5:     update runs and generate a new run ⟨e_t⟩ then add it to R
 6:     if R has a match then
 7:       generate a copy g_copy of R and add g_copy to G
 8:       remove matches from R
 9:     end if
10:     update the group probability of R using (5)
11:     if R has runs that are out of the window next time then
12:       remove such runs from R
13:     end if
14:     for each g ∈ G do
15:       update the group probability of g using (5)
16:       if g does not have a run then
17:         output g and remove it from G
18:       else if g has runs or matches that are out of the window next time then
19:         output g
20:         remove such runs and matches from g
21:       end if
22:     end for
23:   end for
24: end procedure
```

Fig. 5. Group generation based on complete overlap

time 5 because $g_1 = \{\langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{c}_3\rangle, \langle \mathtt{a}_2\, \mathtt{b}_3\, \mathtt{c}_4\rangle, \langle \mathtt{a}_3\, \mathtt{b}_4\, \mathtt{c}_5\rangle\}$ does not have a run. This process continues until the data stream terminates. Note that lines 11 to 13 and lines 18 to 21 are for window processing. We remove runs and matches that are out of the window next time (lines 12 and 20). When a group has matches, we output it (line 19).

4.2 The Case of Single Overlap

In a group based on single overlap, each match should overlap with at least one of the other matches in the same group. The group formation process is not simple, as described below. Consider the situation where the match and the runs in Fig. 6 are generated for the pattern $\langle \mathtt{a}\ \mathtt{b}^{+}\mathtt{c}\rangle$.

In this case, we can formulate the groups $g_1 = \{m_1, r_1\}, g_2 = \{r_2\}$, and $g_3 = \{r_3\}$. However, we cannot yet merge g_1 and g_2 because r_2 overlaps with only r_1, and r_2 may not overlap with g_1 if r_1 is rejected in the future. Similarly we cannot merge g_1 and g_3, nor g_2 and g_3, because they overlap with the runs only. Then, we merge groups into one group when an overlap between them is confirmed. For example, when r_2 becomes $m_2 = \langle \mathtt{a}_4\, \mathtt{b}_5\, \mathtt{c}_6\rangle$ at time 6, we can merge g_2 and g_3 into $g = \{m_2, r_3\}$. Note that we can merge only the groups generated after g_2 because g_2 overlaps with only the run in g_1. If r_2 becomes $m_3 = \langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{b}_3\, \mathtt{b}_4\, \mathtt{b}_5\, \mathtt{c}_6\rangle$ at time 6, we can merge all the groups into one group.

match / run	time				
	1	2	3	4	5
m_1	a	b	c		
r_1	a	b	b	b	b
r_2				a	b
r_3					a

Fig. 6. A match and runs for the pattern $\langle \mathtt{a}\ \mathtt{b}^{+}\mathtt{c} \rangle$

Figure 7 shows the algorithm for generating groups based on single overlap. Suppose that the pattern $\langle \mathtt{a}\ \mathtt{b}\ \mathtt{c} \rangle$ is given for the stream in Fig. 1. A new run $r_1 = \langle \mathtt{a}_1 \rangle$ is generated at time 1 (line 4). As G is an empty set, we do not execute lines 5 to 19 in this iteration. At line 21, we generate a new group $g_1 = \{r_1\}$ and add it to G because r_1 is not added to any group at time 1. At time 2, a new group $g_2 = \{r_2 = \langle \mathtt{a}_2 \rangle\}$ is generated at line 21 because $g_1 = \{r_1 = \langle \mathtt{a}_1\, \mathtt{b}_2 \rangle\}$ does not have matches. g_2 is merged into g_1 at time 3 because g_1 gets the match $m_1 = \langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{c}_3 \rangle$ (lines 6 to 11). We can merge g_1 and g_2 because $g_1 = \{\langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{c}_3 \rangle\}$ and $g_2 =$

```
 1: procedure GenerateGroupsForSingleOverlap(PDS)
 2:   G = ∅ // candidates of groups
 3:   for each e_t ∈ PDS do
 4:     update runs and generate a new run r_new = ⟨e_t⟩
 5:     for each g_i ∈ G do // subscript means the generation order
 6:       if g_i found a match this time then
 7:         for each g_j ∈ G (j > i) do
 8:           g_i = g_i ∪ g_j and remove g_j from G
 9:         end for
10:         g_i = g_i ∪ {r_new}
11:       end if
12:       update the group probability of g_i using (5)
13:       if g_i does not have a run then
14:         output g_i and remove it from G
15:       else if g_i has runs or matches that are out of the window next time then
16:         output g_i
17:         remove such runs and matches from g_i
18:       end if
19:     end for
20:     if r_new is not added to any group then
21:       generate a new group g_new = {r_new} and add it to G
22:       update the group probability of g_new using (5)
23:     end if
24:   end for
25: end procedure
```

Fig. 7. Group generation based on single overlap

$\{\langle \mathtt{a}_2\, \mathtt{b}_3\rangle\}$ certainly overlaps. We output groups that have no runs and remove them (line 14). In this example, $g_1 = \{\langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{c}_3\rangle, \langle \mathtt{a}_2\, \mathtt{b}_3\, \mathtt{c}_4\rangle, \langle \mathtt{a}_3\, \mathtt{b}_4\, \mathtt{c}_5\rangle\, \langle \mathtt{a}_4\, \mathtt{b}_5\, \mathtt{c}_6\rangle\}$ is output at time 6. This process continues until the data stream terminates.

4.3 Use of Threshold of Match Probability

We consider the threshold of a match probability to generate groups efficiently. The runtime of group generation is large when we generate all matches. Thus, we remove matches whose probabilities are lower than the specified threshold. We can remove runs and matches at line 5 in Fig. 5 and line 4 in Fig. 7.

For instance, suppose the pattern $\langle \mathtt{a}\ \mathtt{b}^+\mathtt{c}\rangle$ is given and the match threshold is $\theta = 0.1$ for single overlap matching for the stream in Fig. 1. Ten matches are found from Fig. 1, but matches satisfying θ are $\langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{b}_3\, \mathtt{c}_4\rangle$, $\langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{b}_3\, \mathtt{b}_4\, \mathtt{b}_5\, \mathtt{c}_6\rangle$, and $\langle \mathtt{a}_2\, \mathtt{b}_3\, \mathtt{b}_4\, \mathtt{b}_5\, \mathtt{c}_6\rangle$. Therefore, we construct a group from these three matches.

Although we prune matches with low probabilities, we do not ignore those probabilities. That is, we remove matches such as $\langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{c}_3\rangle$ and $\langle \mathtt{a}_1\, \mathtt{b}_2\, \mathtt{b}_3\, \mathtt{b}_4\, \mathtt{c}_5\rangle$ in our example, but we compute the group probability according to (3). We explain the details in the next section.

5 Efficient Calculation of Group Probability

Using (3), we can calculate a group probability by summing probabilities of all sequences in the group. However, such an approach is not efficient because the number of sequences increases with order $O(|V|^w)$. Therefore, we propose an efficient method using a transducer.

A *finite state transducer* is a finite state automaton which produces output as well as reading input. Figure 8 shows an example of a transducer for the pattern $\langle \mathtt{a}\ \mathtt{b}^+\mathtt{c}\rangle$. This transducer is generated to accept all sequences that contain the pattern as a subsequence. Thus, a probability of arriving at the final state is the sum of probabilities of the sequences. That is, the probability of arriving at the final state becomes the group probability.

We can generate the transducer by adding edges to the NFA in Fig. 3. The rules for adding edges are as follows:

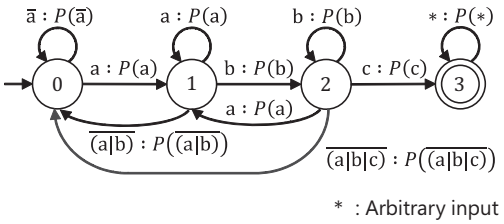

Fig. 8. Transducer for the pattern $\langle \mathtt{a}\ \mathtt{b}^+\mathtt{c}\rangle$

1. We add an edge from the final state to itself with arbitrary inputs.
2. If the state does not have edges with the first event of the pattern, we add an edge that shifts the state to state 1 with the first event.
3. We add edges that shift each state to the initial state with other inputs.

For example, we add an edge with the label "$* : P(*)$" to the final state according to rule 1. This edge enables the transducer to keep accepted sequences in the final state. As the first event of $\langle \mathtt{a}\ \mathtt{b}^{+}\mathtt{c} \rangle$ is $\mathtt{a}$, we add edges with the label "$\mathtt{a} : P(\mathtt{a})$" to state 1 and 2 according to rule 2. We add those edges to accept sequences that contain a part of the pattern such as $\langle \mathtt{a}_1, \mathtt{b}_2, \mathtt{a}_3, \mathtt{b}_4, \mathtt{c}_5 \rangle$. According to rule 3, we add edges with the labels "$\overline{\mathtt{a}} : P(\overline{\mathtt{a}})$", "$\left(\overline{\mathtt{a}|\mathtt{b}}\right) : P\left(\left(\overline{\mathtt{a}|\mathtt{b}}\right)\right)$", and "$\left(\overline{\mathtt{a}|\mathtt{b}|\mathtt{c}}\right) : P\left(\left(\overline{\mathtt{a}|\mathtt{b}|\mathtt{c}}\right)\right)$." Those edges enable rejected sequences to start again from the initial state.

In the following, we explain how to use a transducer for single and complete overlaps.

5.1 The Case of Single Overlap

A group $g = (t_s, t_e, p)$ based on single overlap has all matches in the interval $[t_s, t_e]$. Figure 4 shows an example where g_4 consists of all the matches in $[t_1, t_5]$. Therefore, in single overlap, a group probability is equal to the sum of the probabilities of all sequences that contain the pattern as a subsequence in the interval $[t_s, t_e]$. As described above, such a probability is the probability of arriving at the final state of the transducer. Thus, we can compute a group probability using a transducer instead of computing the probabilities of all sequences.

We use a *transition matrix* of a transducer to calculate the probability of arriving at the final state. Equation (4) is an example of the transition matrix of the transducer in Fig. 8:

$$T_{t_{i-1},t_i} = \begin{bmatrix} P(\overline{\mathtt{a}_{\mathtt{t_i}}}) & P(\mathtt{a}_{\mathtt{t_i}}) & 0 & 0 \\ P\left(\overline{\mathtt{a}|\mathtt{b}_{\mathtt{t_i}}}\right) & P(\mathtt{a}_{\mathtt{t_i}}) & P(\mathtt{b}_{\mathtt{t_i}}) & 0 \\ P\left(\overline{\mathtt{a}|\mathtt{b}|\mathtt{c}_{\mathtt{t_i}}}\right) & P(\mathtt{a}_{\mathtt{t_i}}) & P(\mathtt{b}_{\mathtt{t_i}}) & P(\mathtt{c}_{\mathtt{t_i}}) \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4}$$

Each row corresponds to the previous states and each column corresponds to the present states. For example, $T_{t_{i-1},t_i}(0,1) = P(\mathtt{a}_{\mathtt{t_i}})$ means a probability that shifts state 0 to state 1 is $P(\mathtt{a}_{\mathtt{t_i}})$. Let St be a vector that contains state probabilities. Then we can update the probabilities as follows:

$$St_{t_i} = St_{t_{i-1}} \times T_{t_{i-1},t_i}. \tag{5}$$

In other words, we can calculate the probability of the final state by updating the vector at each time. This process corresponds to lines 12 and 22 in Fig. 7. For example, Fig. 9 shows the change of vector St for computing the group probability of $g = (1, 6, p)$ in Fig. 1. Note that g is the set of all matches found

time		init	1	2	3	4	5	6
state	$St[0]$	1.0	0	0	0.03	0.047	0.0563	0.0563
	$St[1]$	0	1.0	0.3	0.10	0.093	0	0
	$St[2]$	0	0	0.7	0.80	0.630	0.6507	0
	$St[3]$	0	0	0	0.07	0.230	0.2930	0.9437

Fig. 9. Updating state vector St

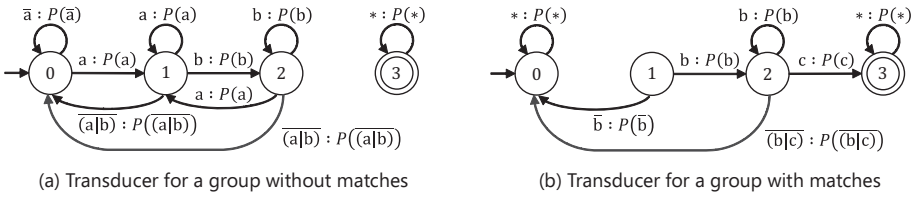

Fig. 10. Additional transducers to calculate a group probability in complete overlap

in Fig. 1. When a group is generated, we initialize the vector such that $St_{init}[0] = 1.0$ and the others are 0. We update the vector using (5) at each time until the group is output. In this case, the group probability is 0.9437 because we output the group at time 6.

This approach can compute a group probability even if we prune matches using a match threshold. As we use a transducer and its transition matrix, we can compute the sum of probabilities of all sequences that contain matches regardless of whether matches are generated or not. Let us continue the above example with $g = (1, 6, p)$. Suppose the matches such as m_1 in Fig. 2 are pruned by a match threshold. In such a case, however, we can compute the same transition matrix at each time using (4). For example, $T_{1,2}$ does not change regardless of whether the matches are generated or not. As we use the transition matrix for calculating the group probability, we can compute the same group probability even if the matches are pruned.

5.2 The Case of Complete Overlap

We cannot calculate a group probability using the former approach for complete overlap. Recall Fig. 2, where two groups $g_1 = \{m_1, m_2, m_3\}$ and $g_2 = \{m_2, m_3, m_4\}$ are generated. If we use the transducer in Fig. 8 to calculate the group probability of g_1, the calculated probability is not correct because it contains the probability of m_4. Similarly, the probability of g_2 is also not correct because of m_1.

Therefore, we use two additional transducers to solve the problem. Figure 10 shows the transducers for the pattern $\langle \texttt{a b}^{+}\texttt{c} \rangle$. The transducer (a) accepts no matches because it does not have edges that shift states to the final state. On the other hand, the transducer (b) generates no runs because it does not have edges that shift states to state 1. We use the transducer (a) while the group does not have matches (line 10 in Fig. 5). The transducer (b) is used after the group

finds matches (line 15 in Fig. 5). Note that we use the transducer in Fig. 8 only once, when a group finds matches for the first time (line 15 in Fig. 5).

Let us continue the above example with $g_1 = \{m_1, m_2, m_3\}$ and $g_2 = \{m_2, m_3, m_4\}$. Let us denote the transition matrix of the transducer in Fig. 8 as T. Similarly, we represent the transition matrices of transducers (a) and (b) in Fig. 10 as T^A and T^B, respectively. We consider the case of calculating the probability of g_1. The system uses T^A before time 3 and T at time 3. T^B is used after time 3 to avoid including the probability of m_4. At $t = 6$, the vector St of g_1 is as follows:

$$St_6 = St_{init} \times T^A_{init,1} \times T^A_{1,2} \times T_{2,3} \times T^B_{3,4} \times T^B_{4,5} \times T^B_{5,6}.$$

Similarly, for g_2 we use T^A before time 5 to avoid including the probability of m_1. Then T is used at time 5 and T^B is used after time 5. Thus, St of g_2 is as follows:

$$St_6 = St_{init} \times T^A_{init,1} \times T^A_{1,2} \times T^A_{2,3} \times T^A_{3,4} \times T_{4,5} \times T^B_{5,6}.$$

If the system uses a match threshold, our approach can compute group probabilities as well as the case of single overlap. In the case of complete overlap, the system decides whether to use the transducers according to the time t_f that a group gets matches for the first time. Thus, the system can compute the group probability of $g = (t_s, t_e, p)$ only if it has a tuple (t_s, t_f, t_e). t_f is easily determined in the process of group generation, because the system can recognize it at line 6 in Fig. 5. Therefore, our approach can calculate correct group probabilities for complete overlap.

6 Experiments

In this section, we analyze the performance of our approach. We constructed a system that generates groups and computes group probabilities using the described approach. The system is an extension of SASE+ [1], a Java-based system for pattern matching queries in a non-uncertain data stream. We performed all measurements on a computer with an Intel Core i7-2600 3.40 GHz CPU, 4.0 GB main memory, and the Windows 7 Professional 64-bit operating system. The system runs under Java Hotspot VM 1.5 with the JVM allocation pool set to 1.5 GB.

The experiments are performed based on simulations. We generate a synthetic probabilistic data stream and use it as an input stream. The generation process is as follows. First, we generate a non-uncertain data stream $\langle \alpha_1, \alpha_2, ..., \alpha_{10000} \rangle$ with 10,000 events. Each event value α_t is taken from the domain $V = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}$. The probability distribution for each event is set as follows. Consider the case of $\alpha_1 = \mathtt{a}_1$. We randomly choose the occurrence probabilities of $\mathtt{b}_1, \mathtt{c}_1$, and $\mathtt{d}_1$ from the range $[0, 0.1]$. Then the probability of $\mathtt{a}_1$ is given as $P(\mathtt{a}_1) = 1 - \sum_{\alpha_1 \in \{\mathtt{b}_1, \mathtt{c}_1, \mathtt{d}_1\}} P(\alpha_1)$. For $t = 2, 3, ...$, we follow the same procedure.

We evaluate the performance of the proposed method using throughput (the number of events processed per second). In the experiments, we use the pattern $\langle \mathtt{a}\ \mathtt{b}^+\mathtt{c} \rangle$ and the parameters in Table 1. In the following, we represent the setting

Table 1. Parameters in the experiments

Parameters	Values
o: overlap type	*comp*: complete overlap *sing*: single overlap
θ: threshold of match probabilities	$\{0, 0.01, 0.02, 0.04, 0.06, 0.08, 0.1\}$
w: window size	$\{5, 10, 15, 20, 25, 50\}$

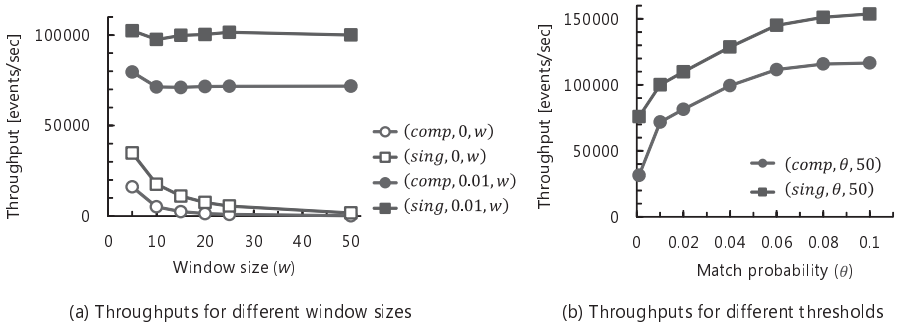

Fig. 11. Throughputs for different window sizes and thresholds

of the parameters as a tuple (o, θ, w). For example, $(comp, 0.01, 50)$ means that the system uses complete overlap, the match threshold is 0.01, and the window size is 50.

6.1 Effect of Parameters

Figure 11(a) shows the throughputs for different window sizes. We show the cases of only $\theta = 0$ and 0.01 because the tendencies for larger thresholds are similar to that of $\theta = 0.01$. We can observe three properties. First, the throughput decreases rapidly if we do not use a threshold ($\theta = 0$). This is due to the number of generated matches. When we use the pattern $\langle \mathtt{a\ b^{+}c} \rangle$, the number of matches increases with order $O(w^2)$. Second, the throughput is independent of the window size if we use a threshold, because most matches are pruned early by the threshold. As described above, the number of matches increases rapidly, but most matches do not have high probabilities. High throughput is achieved because many matches are pruned before their generation. Furthermore, the throughput of single overlap is larger than that of complete overlap due to the number of generated groups. As shown in Fig. 4, complete overlap generates more groups than single overlap. Thus, we need more computation time for complete overlap.

We next study the effect of the threshold setting. Figure 11(b) shows the throughputs for different thresholds. We show the case of only $w = 50$ because

Table 2. Throughputs of the transducer-based approach and the naive approach for different window sizes

w	5	10	15
$(comp, 0.01, w)$+proposed	79,581	71,356	71,076
$(comp, 0.01, w)$+naïve	265	14	Out of memory
$(sing, 0.01, w)$+proposed	102,389	97,575	99,865
$(sing, 0.01, w)$+naïve	267	14	Out of memory

the tendencies for other window sizes are similar to those of $w = 50$. The throughput increases as the threshold becomes higher. This is also due to the number of generated matches. As described above, the higher the threshold, the more we can prune matches. Therefore, we can process each event rapidly if we use a high threshold.

6.2 Effect of Transducer-Based Approach

In this experiment, we study the efficiency of our method with transducers. We compare the proposed method with a naïve one. In the naïve method, we generate all sequences in a window and summarize their probabilities according to (3).

Table 2 shows the throughput measurements for different window sizes, where "proposed" means that we use the proposed method to compute group probabilities. Similarly, "naïve" means the use of the naïve method. Table 2 shows that the throughput of the naïve method decreases exponentially. The naïve method generates all sequences in a window, but their number increases with order $O(|V|^w)$. Therefore, the naïve method cannot compute group probabilities due to memory shortage for $w = 15$. On the other hand, the proposed method can compute group probabilities regardless of the window size. The proposed method uses (5) to compute group probabilities. The computational complexity of (5) is $O\left(|p|^2\right)$, where $|p|$ is the length of pattern p. Note that the length of a pattern means the number of events in the pattern (e.g., $|\langle \mathtt{a\ b^{+}c} \rangle| = 3$). Thus, the computation time of the proposed method is much smaller than that of the naïve one.

7 Related Work

In the literature of non-uncertain data streams, many methods for pattern matching are proposed [1–5,10–14,16–19]. The SASE project [1,5,17–19], proposes an NFA-based approach for finding matches as described in Sect. 2. Moreover, they propose a method to efficiently process the Kleene closure [1,5]. We have implemented our system by extending their CEP system SASE+ [1] and their methods. Including the SASE project, however, all the methods do not consider and process uncertain data streams.

Some researchers have taken on pattern matching in uncertain data streams [7–9,15]. The Lahar project [7,8,15] considers correlated probabilistic data streams. In correlated streams, every event has a conditional probability as an occurrence probability because an underlying Markov process is assumed. To compute match probabilities, they use an NFA translated from a query. They keep probabilities of the states and regard the probability of the final state as the match probability. Their approach, however, merges only simultaneously accepted matches. On the other hand, our approach considers a time interval of matches and merges them according to complete overlap or single overlap. Therefore, we can group matches more flexibly. We do not consider correlations between events in this paper, but we will be able to extend our approach to correlated streams.

[9] proposes a method to find top k matches in probabilistic streams. In [9], probabilistic streams are generated by their system using an error model that translates a non-uncertain event to a probabilistic event. Moreover, the system computes probabilities for the top k matches using the error model, and can merge them. Their approach, however, merges only matches that are among the top k simultaneously accepted matches. On the other hand, we can merge all probabilities in a group using transducers.

8 Conclusion

We proposed a grouping method of matches for probabilistic data streams. We proposed complete overlap and single overlap and defined a group using them. Then, we explained the two algorithms for generating groups. To compute a group probability efficiently, we proposed an approach that uses transducers. We evaluated the efficiency of our approach in simulation-based experiments. Future work will include refinement of the grouping policy and method for group generation, extension to correlated probabilistic data streams, support of other options for the regular expression such as negation, and re-evaluation of our approach using real data sets.

Acknowledgments. This research is partially supported by KAKENHI (25280039, 26540043) and the Center of Innovation Program from the Japan Science and Technology Agency (JST).

References

1. Agrawal, J., Diao, Y., Gyllstrom, D., Immerman, N.: Efficient pattern matching over event streams. In: Proc. ACM SIGMOD, pp. 147–160 (2008)
2. Akdere, M., Çetintemel, U., Tatbul, N.: Plan-based complex event detection across distributed sources. Proc. VLDB Endow. **1**(1), 66–77 (2008)
3. Chandramouli, B., Goldstein, J., Maier, D.: High-performance dynamic pattern matching over disordered streams. Proc. VLDB Endow. **3**(1–2), 220–231 (2010)
4. Demers, A., Gehrke, J., Panda, B.: Cayuga: A general purpose event monitoring system. In: Proc. CIDR, pp. 412–422 (2007)

5. Gyllstrom, D., Agrawal, J., Diao, Y., Immerman, N.: On supporting Kleene closure over event streams. In: Proc. ICDE, pp. 1391–1393 (2008)
6. Jain, A.K., Murty, M.N., Flynn, P.J.: Data clustering: A review. ACM Comput. Surv. **31**(3), 264–323 (1999)
7. Letchner, J., Ré, C., Balazinska, M., Philipose, M.: Access methods for Markovian streams. In: Proc. ICDE, pp. 246–257 (2009)
8. Letchner, J., Ré, C., Balazinska, M., Philipose, M.: Approximation trade-offs in Markovian stream processing: An empirical study. In: Proc. ICDE, pp. 936–939 (2010)
9. Li, Z., Ge, T., Chen, C.X.: ε-matching: Event processing over noisy sequences in real time. In: Proc. ACM SIGMOD, pp. 601–612 (2013)
10. Liu, M., Golovnya, D., Rundensteiner, E.A., Claypool, K.T.: Sequence pattern query processing over out-of-order event streams. In: Proc. ICDE, pp. 784–795 (2009)
11. Majumder, A., Rastogi, R., Vanama, S.: Scalable regular expression matching on data streams. In: Proc. ACM SIGMOD, pp. 161–172 (2008)
12. Mei, Y., Madden, S.: ZStream: A cost-based query processor for adaptively detecting composite events. In: Proc. ACM SIGMOD, pp. 193–206 (2009)
13. Mozafari, B., Zeng, K., Zaniolo, C.: High-performance complex event processing over XML streams. In: Proc. ACM SIGMOD, pp. 253–264 (2012)
14. Qi, Y., Cao, L., Ray, M., Rundensteiner, E.A.: Complex event analytics: Online aggregation of stream sequence patterns. In: Proc. ACM SIGMOD, pp. 229–240 (2014)
15. Ré, C., Letchner, J., Balazinksa, M., Suciu, D.: Event queries on correlated probabilistic streams. In: Proc. ACM SIGMOD, pp. 715–728 (2008)
16. Woods, L., Teubner, J., Alonso, G.: Complex event detection at wire speed with FPGAs. Proc. VLDB Endow. **3**(1–2), 660–669 (2010)
17. Wu, E., Diao, Y., Rizvi, S.: High-performance complex event processing over streams. In: Proc. ACM SIGMOD, pp. 407–418 (2006)
18. Zhang, H., Diao, Y., Immerman, N.: Recognizing patterns in streams with imprecise timestamps. Proc. VLDB Endow. **3**(1–2), 244–255 (2010)
19. Zhang, H., Diao, Y., Immerman, N.: On complexity and optimization of expensive queries in complex event processing. In: Proc. ACM SIGMOD, pp. 217–228 (2014)