Mining Frequent k-Partite Episodes from Event Sequences *

Takashi Katoh[1], Hiroki Arimura[1], and Kouichi Hirata[2]

[1] Graduate School of Information Science and Technology, Hokkaido University
Kita 14-jo Nishi 9-chome, Sapporo 060-0814, Japan
{t-katou, arim}@ist.hokudai.ac.jp
Tel: +81-11-706-7678, Fax: +81-11-706-7890
[2] Department of Artificial Intelligence, Kyushu Institute of Technology
Kawazu 680-4, Iizuka 820-8502, Japan
hirata@ai.kyutech.ac.jp
Tel: +81-948-29-7622, Fax: +81-948-29-7601

Abstract. In this paper, we introduce the class of *k-partite episodes*, which are time-series patterns of the form $\langle A_1, \ldots, A_k \rangle$ for sets A_i $(1 \leq i \leq k)$ of events meaning that, in an input event sequence, every event of A_i is followed by every event of A_{i+1} for every $1 \leq i < k$. Then, we present a backtracking algorithm KPAR and its modification KPAR2 that find all of the frequent k-partite episodes from an input event sequence without duplication. By theoretical analysis, we show that these two algorithms run in polynomial delay and polynomial space in total input size.

1 Introduction

Episode Mining

It is one of the important tasks in data mining to discover frequent patterns from time-related data. For such a task, Mannila *et al.* [8] have introduced *episode mining* to discover frequent *episodes* in an event sequence. Here, an episode is formulated as a labeled acyclic digraphs in which labels correspond to events and arcs represent a temporal precedent-subsequent relation in an event sequence. Then, the episode is a richer representation of temporal relationship than a subsequence, which represents just a linearly ordered relation in sequential pattern mining (*cf.*, [1, 10]). Since the frequency of the episode is formulated by a window that is a subsequence of an event sequence under a fixed time span, the episode mining is more appropriate than the sequential pattern mining when considering the time span.

For subclasses of episodes [3, 6–8], a number of efficient algorithms have been developed so far. Mannila *et al.* [8] presented efficient mining algorithm for *parallel episodes* as a set of events and *serial episodes* a sequence of events. On the other hand, in order to capture the direct relationship between premises and consequences, Katoh *et al.* have introduced the episodes with the special events, a source as a premise and a sink as a consequence, as *sectorial episodes* [6], *diamond episodes* [7] and *elliptic episodes* [3]. Then, they have succeeded to find frequent their episodes concerned with the replacement of bacteria and the changes for drug resistance from bacterial culture data, which are valuable from the medical viewpoint [3, 6, 7].

Since their episodes have just a single source and a single sink, they can represent no relationship between plural premises and consequences. On the other hand, if we extend the diamond episodes with plural sources and sinks, we can deal with not only plural sources and sinks but also the precedent-subsequent relation between 3 sets of events as an episode with length 3. Furthermore, without restriction of the length of episodes, we can generalize such an episode by extending an event in a serial episode as a set of events. The generalized episode can represent the precedent-subsequent relation between sets of events that simultaneously occur.

Hence, in this paper, as a generalized form of episodes, we newly introduce *k-partite episodes* of the form $\langle A_1, \ldots, A_k \rangle$ $(k \geq 0)$, where A_i $(1 \leq i \leq k)$ are sets of events. The k-partite episode $\langle A_1, \ldots, A_k \rangle$ means that every event in A_i $(1 \leq i < k)$ is followed by every event in A_{i+1}. The name "k-partite" comes from the fact that we can represent a k-partite episode as a complete k-partite graph, by adding all of the transitive arcs and by ignoring the direction of arcs.

Main results

For the frequent k-partite episode mining problem, we present a backtracking algorithm KPAR and its modification KPAR2, which achieve polynomial delay and polynomial space complexity.

* This work is partially supported by Grand-in-Aid for JSPS Fellows (20·3406).

A key idea for the efficiency of these algorithms is an enumeration of the candidate k-partite episodes based on depth-first search. The algorithm KPAR searches for frequent k-partite episodes starting with the smallest empty episode, and then expands a candidate episode by attaching a new event one by one to the tail component. Once the algorithm reaches an infrequent episode, it stops to expand the current branch and backtracks for remaining branches in a search tree.

Let Σ be an alphabet of events, X a k-partite episode over Σ, $\mathcal{S}$ an event sequence of size N and of length n and w a window width. Then, *the matching problem of X in $\mathcal{S}$ against w* is to determine whether or not there exists a contiguous subsequence of $\mathcal{S}$ of length w that contains X satisfying the edge constraints. A straightforward generate-and-test method for the matching problem requires exponential time in the size M of a k-partite episode. By introducing the notion of the *leftmost tail occurrences*, we show that the matching problem is solvable in $O(|\Sigma|n)$ time, and we present an efficient scanning-based subprocedure COUNTBYSCAN that solves the matching problem of X in $\mathcal{S}$ in $O(w|\Sigma|n)$ time. Note here that $N \leq |\Sigma|n$. Then, we show that the algorithm KPAR runs in $O(|\Sigma|^2 wn)$ time per a k-partite episode and $O(|\Sigma|k)$ space.

In some real-world datasets, input sequences are often *sparse*, that is, the total number $N(= ||\mathcal{S}||)$ of the occurrences of events in $\mathcal{S}$ is much smaller than sn, where $s = |\Sigma|$ and $n = |\mathcal{S}|$. In such a case, the performance of a scanning-based algorithm such as COUNTBYSCAN may degenerate, since it is necessary to traverse many windows overall in order to solve the matching problem. To cope with this problem, we give a practical speed-up technique for frequency counting. We show that the matching problem of a k-partite episode in $\mathcal{S}$ is solvable in $O(N)$ time, and we present a practical algorithm COUNTBYLIST that computes the frequency counts in $O(N^2 k)$ time by using the event lists. The modified algorithm KPAR2 with COUNTBYLIST runs in $O(|\Sigma|N^2 k)$ time per a k-partite episode and in $O(|\Sigma|k + N)$ space, where $O(N)$ coincides with the space for storing all event sequences.

As a corollary, we show that the frequent episode mining problem is solvable in polynomial delay and in polynomial space in the total input size.

Organization

This paper is organized as follows. In Section 2, we introduce episodes and other notions necessary to the later discussion. In Section 3, we introduce k-partite episodes and other notions and discuss their properties. In Section 4, we present the algorithms KPAR and KAPR2 to extract all of the frequent k-partite episodes. In Section 5, we conclude this paper and discuss the future works.

2 Preliminaries

In this section, we introduce the frequent episode mining problem and the related notions necessary to later discussion. We denote the sets of all integers and all natural numbers by $\mathbf{Z}$ and $\mathbf{N}$, respectively. For a set S, we denote the cardinality of S by $|S|$. A *digraph* is a graph with directed edges (or, *arcs*). An *acyclic digraph* is a digraph without cycles.

2.1 An input event sequence and its windows

Let $\Sigma = \{1, \ldots, m\}$ $(m \geq 1)$ be a finite alphabet with the total order $\leq$ over $\mathbf{N}$. Each element $e \in \Sigma$ is called an *event* [3]. An *input event sequence* (*input sequence*, for short) $\mathcal{S}$ on Σ is a finite sequence $\langle S_1, \ldots, S_n \rangle \in (2^\Sigma)^*$ of events $(n \geq 0)$, where $S_i \subseteq \Sigma$ is called the i-th *event set* for every $1 \leq i \leq n$. For any $i < 0$ or $i > n$, we set $S_i = \emptyset$. For an input event sequence $\mathcal{S}$, we denote the *length* n by $|\mathcal{S}|$ and define the *total size* $||\mathcal{S}||$ of $\mathcal{S}$ by $\sum_{i=1}^{n} |S_i|$. Clearly, $||\mathcal{S}|| = O(|\Sigma|n)$, but the converse does not hold in general, that is, $O(||\mathcal{S}||) \neq |\Sigma|n$. Without loss of generality, we can assume that every event in Σ appears at least once in $\mathcal{S}$.

2.2 Episodes

Mannila *et al.* [8] have formulated an episode as a partially ordered. On the other hand, We formulate an episode as a labeled acyclic digraph as follows. An *episode* over Σ is a labeled acyclic digraph $X = (V, E, g)$, where V is a set of vertices, $E \subseteq V \times V$ is a set of arcs and $g : V \to \Sigma$ is a mapping associating each node with an event. It is not hard to see that two definitions of episodes by Mannila's partially ordered sets [8] and our labeled acyclic digraphs are essentially same each other.

Let $X = (V, E, g)$ be an episode. We define the *size* $||X||$ of X by $|V|$. For an arc set E on a vertex set V, let E^+ be the *transitive closure* of E such that $E^+ = \{ (u, v) \mid \text{there is a directed path from } u \text{ to } v \}$.

[3] Mannila *et al.* [8] originally referred to each element $e \in \Sigma$ itself as an *event type* and an occurrence of e as an *event*. However, we simply call both of them as *events*.

Definition 1 (embedding). For episodes $X_i = (V_i, E_i, g_i)$ $(i = 1, 2)$, X_1 *is embedded in* X_2, denoted by $X_1 \sqsubseteq X_2$, if there exists a mapping $f : V_1 \to V_2$ such that (i) f *preserves the labels of vertices*, i.e., for all $v \in V_1$, $g_1(v) = g_2(f(v))$, and (ii) f *preserves the precedence relation*, i.e., for all $u, v \in V$ with $u \neq v$, if $(u, v) \in E_1$ then $(f(u), f(v)) \in (E_2)^+$. The mapping f is called an *embedding* from X_1 to X_2.

Given an input sequence $\mathcal{S} = \langle S_1, \ldots, S_n \rangle \in (2^\Sigma)^*$, an *window* in $\mathcal{S}$ is a contiguous subsequence $W = \langle S_i \cdots S_{i+w-1} \rangle \in (2^\Sigma)^*$ of $\mathcal{S}$ for some i, where $w \geq 0$ is the *width* of W.

Definition 2 (occurrence for an episode). An episode $X = (V, E, g)$ *occurs in* an window $W = \langle S_1 \cdots S_w \rangle \in (2^\Sigma)^*$, denoted by $X \sqsubseteq W$, if there exists a mapping $h : V \to \{1, \ldots, w\}$ such that (i) h preserves the labels of vertices, i.e., for all $v \in V$, $g(v) \in S_{h(x)}$, and (ii) h preserves the precedence relation, i.e., for all $u, v \in V$ with $u \neq v$, if $(u, v) \in E$ then $h(u) < h(v)$. The mapping h is called an *embedding* of X into W.

A *window width* is a fixed positive integer $1 \leq w \leq n$. For any $-w+1 \leq i \leq n$, we say that an episode X *occurs at* position i in an event sequence $\mathcal{S}$ if $X \sqsubseteq W_i$, where $W_i = \langle S_i, \ldots, S_{i+w-1} \rangle$ is the i-th window of width w in $\mathcal{S}$. Then, we call i an *occurrence* or *label* of X in $\mathcal{S}$. In what follows, we denote the i-th window W_i by $\mathbf{W}_i^{\mathcal{S},w}$. Let $\mathbf{W}_{\mathcal{S},w} = \{\, i \mid -w+1 \leq i \leq n \}$ be the domain of the occurrences. For an episode X, we define the *occurrence window list* $\mathbf{W}_{\mathcal{S},w}(X)$ for X in $\mathcal{S}$ by the set $\{\, -w+1 \leq i \leq n \mid X \sqsubseteq W_i \,\}$ of the occurrences of X in $\mathcal{S}$.

2.3 Frequent episode mining problem

Let $\mathcal{C}$ be a subclass of episodes, X an episode in $\mathcal{C}$, $\mathcal{S}$ an input sequence and w (≥ 1) a window width. Then, the *frequency* $freq_{\mathcal{S},w}(X)$ of X in $\mathcal{S}$ is defined by the number of w-windows, that is, $freq_{\mathcal{S},w}(X) = |\mathbf{W}_{\mathcal{S},w}(X)| = O(|\mathcal{S}|)$. A *minimum frequency threshold* is any positive integer $\sigma \geq 1$. Without loss of generality, we can assume that $\sigma \leq |\mathcal{S}|$. An episode X is *σ-frequent in $\mathcal{S}$* if $freq_{\mathcal{S},w}(X) \geq \sigma$. We denote the set of all σ-frequent episodes occurring in $\mathcal{S}$ by $\mathcal{F}_{\mathcal{S},w,\sigma}$.

Definition 3. Frequent Episode Mining Problem for $\mathcal{C}$:
Let $\mathcal{C}$ be a subclass of episodes we consider. Given an input sequence $\mathcal{S} \in (2^\Sigma)^*$, a window width $w \geq 1$, and a minimum frequency threshold $\sigma \geq 1$, the task is to find all of the σ-frequent episodes X within class $\mathcal{C}$ that occur in $\mathcal{S}$ with a window width w without duplication.

Our goal is to design an efficient algorithm for the frequent episode mining problem for the class of k-partite episodes, which we will introduce in the next section, in the framework of enumeration algorithms [2]. Let N be the total input size and M the number of all solutions. An enumeration algorithm $\mathcal{A}$ is of *output-polynomial time*, if $\mathcal{A}$ finds all solutions $S \in \mathcal{S}$ in total polynomial time both in N and M. Also $\mathcal{A}$ is of *polynomial delay*, if the *delay*, which is the maximum computation time between two consecutive outputs, is bounded by a polynomial in N alone.

3 *k*-Partite Episodes

In this section, we introduce the class of k-partite episodes for any $k \geq 0$ and other notions and discuss their properties.

3.1 Definition

In this paper, we regard an event $e \in \Sigma$ as the episode $X = (\{v\}, \emptyset, g)$ such that $g(v) = e$. Similarly, we regard a set of event $\{e_1, \ldots, e_n\} \subseteq \Sigma$ $(n \geq 0)$ as a *parallel episode*, that is, an episode $X = (\{v_1, \ldots, v_n\}, \emptyset, g)$ such that $g(v_i) = e_i$ for every $1 \leq i \leq n$ [8]. Then, we call an episode $X = (\emptyset, \emptyset, g)$ with an empty vertex set an *empty episode* and denote X by $\emptyset$.

Definition 4. For $k \geq m \geq 1$, a *k-serial episode* (or a *serial episode*) over Σ is an episode $X = (V, E, g)$ satisfying that $V = \{v_1, \ldots, v_m\}$, $E = \{(v_i, v_{i+1}) \mid 1 \leq i < m\}$ and $g(v_i) = a_i$ for every $1 \leq i \leq m$. We denote such a k-serial episode by a sequence $(a_1 \mapsto \cdots \mapsto a_m)$ of events $a_1, \ldots, a_m \in \Sigma$.

Definition 5. For $k \geq m \geq 1$, a *k-partite episode* (or a *partite episode*) over Σ is an episode $X = (V, E, g)$ satisfying the following conditions (i) – (iii):

(i) $V = V_1 \cup \cdots \cup V_k$, where $V_i \neq \emptyset$ and $V_i \cap V_j \neq \emptyset$ for every i and j $(1 \leq i < j \leq k)$.
(ii) X is *complete*, i.e., $E = (V_1 \times V_2) \cup \cdots \cup (V_{k-1} \times V_k)$ holds.

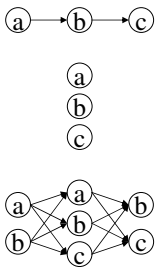

Fig. 1. The examples of a 3-serial episode (top), a parallel episode (center), and a 3-partite episode (bottom) on the alphabet $\Sigma = \{a, b, c\}$.

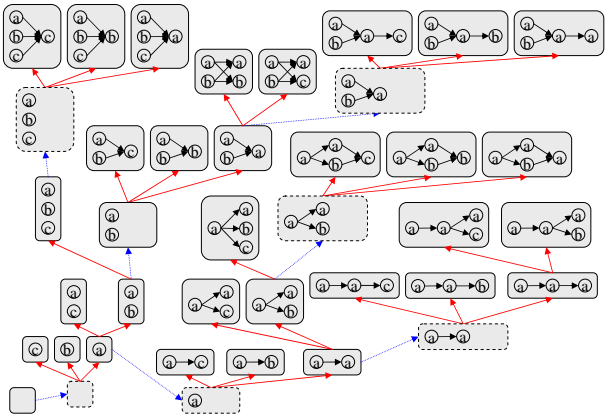

Fig. 2. The parent-child relationships on the alphabet $\Sigma = \{a, b, c\}$, where episodes in dashed boxes are pre-partite episodes.

(iii) X is *partwise-linear*, i.e., for every $1 \leq i \leq k$, the set V_i contains no distinct vertices with the same labeling by g.

We denote such a k-partite episode by an m-tuple $X = \langle A_1, \ldots, A_m \rangle$, where $A_i \subseteq \Sigma$ is a set of events for every $1 \leq i \leq m$.

For example, we describe a 3-serial episode, a parallel episode, and a 3-partite episode on the alphabet $\Sigma = \{a, b, c\}$ in Fig. 1. In what follows, we denote the classes of k-serial, parallel, sectorial [6], diamond [7] and k-partite episodes over Σ by $\mathcal{SE}_k$, $\mathcal{PE}$, $\mathcal{SEC}$, $\mathcal{DE}$ and $\mathcal{PTE}_k$, respectively. For these subclasses of episodes, the following inclusion relation hold: (i) $\mathcal{SE}_1 \subseteq \mathcal{PE} \subseteq \mathcal{PTE}_1$, (ii) $\mathcal{SE}_2 \subseteq \mathcal{SEC} \subseteq \mathcal{PTE}_2$, (iii) $\mathcal{SE}_3 \subseteq \mathcal{DE} \subseteq \mathcal{PTE}_3$ and (iv) $\mathcal{SE}_k \subseteq \mathcal{PTE}_k$ $(k \geq 0)$.

4 Algorithm

4.1 Depth-first enumeration of k-partite episodes

In this section, we present a polynomial-delay and polynomial-space algorithm KPAR for extracting all of the frequent k-partite episodes in an input event sequence. Throughout of this section, let $\mathcal{S} = (S_1, \ldots, S_n) \in (2^{\Sigma})^*$ be an input event sequence, where $|\mathcal{S}| = n$ and $||\mathcal{S}|| = N$, $w \geq 1$ a window width and $\sigma \geq 1$ the minimum frequency threshold.

For a partite episode $X = \langle A_1, \ldots, A_{k-1} \rangle$ of the length $k - 1 \geq 0$, we define the *k-pre-partite episode* of the length $k \geq 0$ as an episode $Y = \langle A_1, \ldots, A_{k-1}, \emptyset \rangle$. The main idea of our algorithm is to enumerate all of the frequent k-partite episodes by searching for the whole search space from general to specific by using depth-first search. For the search space, we introduce the parent-child relationship between k-partite episodes and k-pre-partite episodes, in order to output no k-pre-partite episodes.

Definition 6. The 0-partite episode $\perp = \langle\rangle$ is the *root*. For $1 \leq i \leq k$, the *parent* of the i-partite or i-pre-partite episode $X = \langle A_1, \ldots, A_i \rangle$ is defined by:

$$parent(\langle A_1, \ldots, A_i \rangle) = \begin{cases} \langle A_1, \ldots, A_{i-1} \rangle, & \text{if } A_i = \emptyset, \\ \langle A_1, \ldots, (A_i - \{\max A_i\}) \rangle, & \text{otherwise.} \end{cases}$$

Also we define the set of all children of X by $Children(X) = \{Y \mid parent(Y) = X\}$. Then, we define the *family tree* for $\mathcal{PTE}_k$ by a rooted digraph $\mathcal{T}(\mathcal{PTE}_k) = (V, E, \perp)$ with the root $\perp$, where $V = \mathcal{PTE}_k \cup \{ X \mid X \text{ is } k\text{-pre-partite episode} \}$ and $E = \{ (X, Y) \mid X \text{ is the parent of } Y, Y \neq \perp \}$.

Lemma 1. *The family tree $\mathcal{T}(\mathcal{PTE}_k) = (V, E, \perp)$ for $\mathcal{PTE}_k$ is a rooted tree with the root $\perp$.*

Proof. For any partite episode $X = \langle A_1, \ldots, A_k \rangle$ of length $k \geq 1$, it holds that $||parent(X)|| = ||X|| - 1$. For any pre-partite episode $X = \langle A_1, \ldots, A_{k-1}, \emptyset \rangle$ of length $k \geq 1$, it holds that $||parent(X)|| = ||X||$, and then, $parent(X) = \langle A_1, \ldots, A_{k-1} \rangle$ is a partite episode. Therefore, we can show that, for any vertex $X \in V$, X is reachable from $\perp$ by a path of the length at most $2||X||$. Moreover, we can show that $\mathcal{T}(\mathcal{PTE}_k)$ is acyclic. Hence, the statement holds. □

algorithm KPAR($\mathcal{S}, k, w, \Sigma, \sigma$)
input: input event sequence $\mathcal{S} = \langle S_1, \ldots S_n \rangle \in (2^{\Sigma})^*$ of length $n \geq 0$,
maximum length of output partite episode $k \geq 0$, window width $w > 0$,
alphabet of events $\Sigma = \{1, \ldots, s\}$ ($s \geq 1$), the minimum frequency $1 \leq \sigma \leq n + k$;
output: the set of all σ-frequent k-partite episodes in $\mathcal{S}$ with window width k;
method:
1 **output** $\langle\rangle$;
2 KPARREC($\langle\emptyset\rangle, 1, \mathcal{S}, w, \Sigma, \sigma$);

procedure KPARREC($X, i, \mathcal{S}, k, w, \Sigma, \sigma$)
input: parent partite episode $X = \langle A_1, \ldots, A_i \rangle$, and $\mathcal{S}$, k, w, Σ, and σ are same as in KPAR.
output: the set of all σ-frequent i-partite episodes in $\mathcal{S}$ that are descendants of X and $i \leq k$;
method:
1 **if** ($i > k$) **then return**;
2 $f :=$ COUNTBYSCAN($X, \mathcal{S}, w$); // $f = |W_{\mathcal{S},w}(X)|$;
3 **if** ($f < \sigma$) **then return**;
4 **if** ($A_i \neq \emptyset$) **then**
5 **output** X;
6 KPARREC($\langle A_1, \ldots, A_i, \emptyset \rangle, i + 1, \mathcal{S}, w, \Sigma, \sigma$);
7 **end if**
8 **foreach** ($e \in \Sigma$ such that $e > \max(A_i)$) **do**
9 KPARREC($\langle A_1, \ldots, A_i \cup \{e\} \rangle, i, \mathcal{S}, w, \Sigma, \sigma$);

Fig. 3. The main algorithm KPAR and a recursive subprocedure KPARREC for mining frequent k-partite episodes in a sequence.

procedure COUNTBYSCAN($X, \mathcal{S}, w$)
input: k-partite episode $X = \langle A_1, \ldots, A_k \rangle$, an input sequence $\mathcal{S} = \langle S_1, \ldots S_n \rangle$, window width $w > 0$;
output: the frequency of X;
method:
1 $f := 0$;
2 **for** ($i := -w + 1, \ldots, n$) **do**
3 Test if $X \sqsubseteq \mathbf{W}_i^{\mathcal{S},w} = \langle S_i, \ldots, S_{i+w-1} \rangle$ by the procedure in Lemma 2.
4 **if** the answer is "Yes" **then** $f := f + 1$;
5 **end for**
6 **return** f;

Fig. 4. An algorithm COUNTBYSCAN for computing the frequency of a k-partite episode.

In Fig. 2, we describe the part of family tree $\mathcal{T}(\mathcal{PTE}_3)$ forms the spanning tree for all 3-partite episodes of $\mathcal{PTE}_3$ on the alphabet $\Sigma = \{a, b, c\}$.

In Fig. 3, we describe the algorithm KPAR and its subprocedure KPARREC for extracting frequent k-partite episodes from an input event sequence $\mathcal{S}$. The algorithm is a backtracking algorithm that traverses the spanning tree $\mathcal{T}(\mathcal{PTE}_k)$ based on depth-first search starting from the root $\perp$ using the parent-child relationships over $\mathcal{PTE}_k$.

4.2 Basic algorithm with frequency counting by scanning

Let $\mathcal{S} = (S_1, \ldots, S_n) \in (2^{\Sigma})^*$ be an input event sequence of length n. For $1 \leq i \leq j \leq n$, we denote the subsequence $\mathcal{S}[i..j] = (S_i, \ldots, S_j)$ by $\mathcal{S}[i..j]$. Also let X be a k-partite episode over Σ and w a window width. Then, the *matching problem of X in $W = \mathcal{S}[i..j]$ against w* is to determine whether or not there exists a contiguous subsequence W' of W of length w such that $X \sqsubseteq W'$. Let $1 \leq i \leq n$ be any position. The *leftmost tail occurrence* of a k-partite episode $X = \langle A_1, \ldots, A_k \rangle$ w.r.t. the right boundary i is the smallest index $i \leq j \leq n$ such that X is contained in the prefix $\mathcal{S}[i..j]$, i.e., $X \sqsubseteq \mathcal{S}[i..j]$.

Lemma 2. *For a k-partite episode $X = \langle A_1, \ldots, A_k \rangle$ over Σ, an input sequence $\mathcal{S}$ of length n and a position $1 \leq r \leq n$, suppose that $P = (p_1, \ldots, p_k)$ is the list of positions such that, for every $1 \leq i \leq k$, p_i is the leftmost tail occurrence of i-partite episode $X_i = \langle A_1, \ldots, A_i \rangle$ w.r.t. the right boundary r. Also let $p_0 = r$. Then, we have the following statements:*

1. *P is increasing, i.e., it holds that $p_0 < p_1 < \cdots < p_k$.*
2. *For every $1 \leq i \leq k$, it holds that $p_i = \max_{e \in A_i} \min\{\, j \mid p_{i-1} < j \leq n, e \in S_j \,\}$.*

algorithm KPAR2($\mathcal{S}, k, w, \Sigma, \sigma$)
input: an input event sequence $\mathcal{S} = \langle S_1, \dots S_n \rangle \in (2^{\Sigma})^*$ of length $n \geq 0$,
an integer $k \geq 0$, an window width $w > 0$, an alphabet Σ, a minimum frequency $1 \leq \sigma \leq n + k$;
output: the set of all σ-frequent k-partite episodes in $\mathcal{S}$ with window width w;
method:
1 Compute the event list table $\mathcal{L} = \{L[e]\}_{e \in \Sigma}$ in $\mathcal{S}$;
2 **output** $\langle\rangle$;
3 KPARREC2($\langle\emptyset\rangle, 1, \mathcal{S}, \mathcal{L}, w, \Sigma, \sigma$);
// KPARREC2 is same as KPARREC except that this calls COUNTBYLIST with $\mathcal{L}$ instead of COUNTBYSCAN;

Fig. 5. The modified algorithm KPAR2 for mining frequent k-partite episodes in a sequence.

3. The list P can be computed in $O(|\Sigma|n)$ time.

Thus, the matching problem for k-partite episodes against a window width w can be solved in $O(|\Sigma|w)$ time. The algorithm COUNTBYSCAN in Fig. 4 computes the frequency count for a k-partite episode X based on the above lemma.

Lemma 3. Let $\mathcal{S}$ be an input event sequence of length n and w a window width. Then, the algorithm COUNTBYSCAN in Fig. 4 computes the frequency $freq_{\mathcal{S},w}(X) = |\mathbf{W}_{\mathcal{S},k}(X)|$ of a partite episode X in $O(|\Sigma|wn)$ time.

To estimate the delay of the algorithm precisely, we adopt the compact representation of the current episode X by storing only the difference of X from its parent. Moreover, we use the alternating output technique of [11] to reduce the delay by factor of the depth of the search tree. Then:

Theorem 1. *Let $\mathcal{S}$ be an input event sequence of length n. Then, the algorithm* KPAR *in Fig. 3 with* COUNTBYSCAN *finds all of the σ-frequent k-partite episodes occurring in $\mathcal{S}$ without duplication in $O(|\Sigma|^2 wn)$ delay and in $O(|\Sigma|k)$ space.*

Proof. At each iteration of the algorithm KPARREC, the algorithm computes the frequency count f in $O(|\Sigma|wn)$ by Lemma 3 and executes other instructions than the invocation of *KparRec* within the same cost. Since, each frequent episode has at most $O(|\Sigma|)$ infrequent children, the running time per a σ-frequent k-partite episode is $O(|\Sigma|^2 wn)$ time. □

4.3 Modified algorithm with frequency counting on event lists

In Fig. 5, we describe a modified version of our algorithm KPAR2 and its subprocedure KPARREC2 for extracting frequent k-partite episodes from input sequence $\mathcal{S}$.

The key idea of KPAR2 is a subprocedure COUNTBYLIST that computes the frequency counts using so-called event lists. For an event $e \in \Sigma$, the *event list* of e in an input event sequence $\mathcal{S}$ is an increasing list $L[e] = (p_1, \dots, p_m)$ of positions $1 \leq p \leq n$ such that $e \in S_i$. The *event list table* $\mathcal{L}$ in $\mathcal{S}$ is the set $\mathcal{L} = \{L[e]\}_{e \in \Sigma}$ of event lists for all events in Σ. Under the assumption that every event in Σ appears at least once in $\mathcal{S}$, the total number of elements in $\mathcal{L}$ equals to N. A position $1 \leq i \leq n$ is said to be *appropriate* in $\mathcal{S}$ if S_i is not an empty set. Without loss of generality, we can assume that the list $\mathcal{A} = (i_1, \dots, i_h)$ $(h \geq 0)$ of the appropriate positions in $\mathcal{S}$ is given.

Lemma 4. *The event list table $\mathcal{L}$ in $\mathcal{S}$ can be computed in $O(N)$ time.*

Proof. We encode a event list $L[e] = (p_1, \dots, p_h)$ $(h \geq 0)$ by a triple $Head = 0$, $POS[0..h-1]$ and $NEXT[0..h-1]$ such that, for each pointer $\pi \in [0..h-1]$, $POS[\pi]$ is the position pointed by π and $\pi' = NEXT[\pi]$ is the next pointer of π. Then, the following algorithm computes $\mathcal{L}$ in $\mathcal{S}$.

method:
1 **foreach** $e \in \Sigma$ **do** $last_e := -1$; $free_e := 0$; $Head_e := 0$; **end foreach**
2 **for** $(i := i_1, \dots, i_h)$ **do** // $(i_1, \dots, i_h) = \mathcal{A}$ is the list of the appropriate positions in $\mathcal{S}$;
3 **foreach** $e \in S_i$ **do**
4 $p_e := free_e$; $free_e := free_e + 1$; $POS_e[p_e] := i$;
5 **if** $last_e \neq -1$ **then** $NEXT_e[last_e] := p_e$;
6 $last_e := p_e$;
7 **end foreach**
8 **foreach** $e \in S_i$ **do** $NEXT_e[last_e] := last_e$;
9 **return** $\mathcal{L} = (Head_e, POS_e, NEXT_e)_{e \in \Sigma}$;

procedure LeftMostTail(A: a set of events, $\mathcal{S}$: an input sequence of total size N):
method:
1 $i := 0$;
2 **foreach** $e \in A$ **do** $\pi_e := 0$; $LM[e] := 0$; **end foreach**
3 **while** $i < n$ **do**
4 Let i be the next appropriate position such that $i > i_{last}$ and $S_i \neq \emptyset$;
5 **foreach** $e \in S_i \cap A$ **do**
6 $LM[e] := POS_e[\pi_e]$; $\ell_{\max} := \max(\ell_{\max}, LM[e])$; $\pi_e = NEXT_e[\pi_e]$;
7 **end foreach**
8 **output** $(h, \ell_{\max})$ as the pair of the position $h = i + 1$ and the leftmost tail position w.r.t. h;
9 $i_{last} := i$;
10 **end while**

Fig. 6. A streaming algorithm LeftMostTail that computes all leftmost tail positions of a set A of events by scanning an input sequence from left to rigth.

procedure IncLeftMostTail(A: a set of events, ℓ: the left boundary, $(LM[\cdot], \pi[\cdot], i_{last}, \ell_{\max})$: internal state):
method:
1 $i := i_{last}$;
2 **if** $i \geq \ell$ **then return** $(\ell_{\max}, (LM[\cdot], \pi[\cdot], i_{last}, \ell_{\max}))$;
3 **while** $i < n$ **do**
4 Let i be the smallest appropriate position such that $i > i_{last}$ and $S_i \neq \emptyset$;
5 **foreach** $e \in S_i \cap A$ **do**
6 $LM[e] := POS_e[\pi[e]]$; $\ell_{\max} := \max(\ell_{\max}, LM[e])$; $\pi[e] = NEXT_e[\pi[e]]$;
7 **end foreach**
8 $i_{last} := i$;
9 **if** $i \geq \ell$ **then return** $(\ell_{\max}, (LM[\cdot], \pi[\cdot], i_{last}, \ell_{\max}))$;
10 **end while**
11 **return** $(n + 1, (LM[\cdot], \pi[\cdot], i_{last}, n + 1))$; *//failed!*

Fig. 7. An incremental version of LeftMostTail that computes the leftmost tail position of a set A of events w.r.t. a given position ℓ.

The time complexity of the algorithm is obviously $O(N)$ time. □

By regarding a set $A \subseteq \Sigma$ of events as a parallel episode, we can define the leftmost tail occurrence q of A w.r.t. a position p in $\mathcal{S}$ as before. Then, we have that $p = \max_{e \in A} \min\{\, j \mid p \leq j \leq n, e \in S_j \,\}$.

Lemma 5. *Suppose that $S_0 = \Sigma$. Then, the algorithm* LeftMostTail *in Fig. 6 computes the set of all distinct pairs (h, ℓ_h) of a position $1 \leq h \leq n$ and the corresponding leftmost tail position ℓ_h w.r.t. h in the increasing order of h in $O(N)$ time.*

In Fig. 8, we describe the algorithm CountByList for computing frequency of episodes with an incremental version IncLeftMostTail of LeftMostTail in Lemma 5. The algorithm CountByList is an algorithm that computes the size of occurrence window list $\mathbf{W}_{\mathcal{S},k}(X)$ for a k-partite episode X by sweeping from the heads of the event lists $\mathcal{L}$ to the tails of $\mathcal{L}$.

Lemma 6. *Let $\mathcal{S}$ be an input sequence of length n and w a window width. Then, the algorithm* CountByList *in Fig. 8 computes the frequency $freq_{\mathcal{S},w}(X)$ of a k-partite episode X in $O(N^2K) = O(N^2w)$ time, where $N = ||\mathcal{S}||$ is the total size of an input event sequence $\mathcal{S}$.*

Proof. In the algorithm CountByList, the outer while-loop from line 6 to line 16 is executed $O(N)$ times and the inner for-loop from line 8 to line 9 is executed $O(K)$ times and the call to IncLeftMostTail takes $O(N)$ time in the worst case. Therefore, the running time of CountByList is $O(N^2K)$ time. □

By using alternating output technique [11], we show the following theorem on the complexity of the modified algorithm Kpar2.

Proposition 1. *Let $\mathcal{S}$ be an input event sequence of length n over Σ. Then, we can implement the algorithm* Kpar2 *to find all of the σ-frequent k-partite episodes in $\mathcal{S}$ without duplication in $O(|\Sigma|wN^2)$ delay and $O(|\Sigma|k + N)$ space, where $N = ||\mathcal{S}||$ is the total size of input event sequence $\mathcal{S}$.*

procedure CountByList($X = \langle A_1, \ldots, A_K \rangle, w, \mathcal{S}, \mathcal{L}$)
input: k-partite episode X, window width $w > 0$,
an input sequence $\mathcal{S} = \langle S_1, \ldots S_n \rangle$, and occurrence lists $\mathcal{L}$ for events in $\mathcal{S}$;
output: the number f of windows in which X occurs;
method:

```
1  for k := 1, ..., K do
2      foreach e ∈ A_i do LM[k][e] := 0; π_e = 0; i_last := 0; end foreach
3      State[k] := (LM[k][·], π[·], i_last, 0);
4  end for
5  i := 0;
6  while i < n do
7      Let i be the next appropriate position such that i > i_last and S_i ≠ ∅; ℓ_max[0] := i;
8      for k := 1, ..., K do // Note: The vector State[k] represents the internal state for k.
9          (ℓ_max[k], State[k]) := IncLeftMostTail(A_k, ℓ_max[k − 1], State[k]);
10     ℓ_max := ℓ_max[K];      // Note: ℓ_max is the leftmost tail occurrence for episode X w.r.t. position i;
11     if ℓ_max[k] > ℓ_last then
12         cont := i − max(0, ℓ_max[k] − w + 1); f := f + count;
13     end if
14     ℓ_last := i + w;
15     i_last := i;
16 end while
17 return f;
```

Fig. 8. An algorithm CountByList for computing the number of windows in which a k-partite episode occurs.

5 Conclusion

This paper studied the problem of frequent k-partite episode mining, and presented polynomial-delay and polynomial-space algorithms Kpar and kpar2 that find all frequent k-partite episodes in an input sequence. It is a future work to implement our two algorithms Kpar and kpar2 and give empirical results to compare the time and space efficiencies of the algorithms. Also, it is a future work to apply the proposed algorithm to bacterial culture data [3, 7].

Although Lemma 6 says that the time complexity of CountByList is $O(N^2K)$ time, this can be improved to $O(Nw)$ time by more detailed analysis on the amortized time of the repeated execution of the subprecedure IncLeftMostTail. This is another future work.

References

1. H. Arimura, T. Uno: A polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence, *Proc. ISAAC'05*, LNCS 3827, 2005.
2. D. Avis, K. Fukuda: Reverse search for enumeration, *Discrete Applied Mathematics*, **65**, 21–46, 1996.
3. T. Katoh, K. Hirata: Mining frequent elliptic episodes from event sequences, *Proc. 5th LLLL*, 46–52, 2007.
4. T. Katoh, K. Hirata: A simple characterization on serially constructible episodes, *Proc. PAKDD'08*, LNAI 5012, 600-607, 2008.
5. T. Katoh, H. Arimura, K. Hirata: A polynomial-delay polynomial-space algorithm for extracting frequent diamond episodes from event sequences *Proc. PAKDD'09*, LNAI 5476, 172-183, 2009.
6. T. Katoh, K. Hirata, M. Harao: *Mining sectorial episodes from event sequences*, *Proc. 10th DS*, LNAI 4265, 137–145, 2006.
7. T. Katoh, K. Hirata, M. Harao: Mining frequent diamond episodes from event sequences, *Proc. 4th MDAI*, LNAI 4617, 477–488, 2007.
8. H. Mannila, H. Toivonen, A. I. Verkamo: Discovery of frequent episodes in event sequences, *Data Mining and Knowledge Discovery*, **1**, 259–289, 1997.
9. J. Pei, H. Wang, J. Liu, K. Wang, J. Wang, P. S.. Yu: Discovering frequent closed partial orders from strings, *IEEE TKDE*, **18**, 1467–1481, 2006.
10. J. Pei, J. Han, B. Mortazavi-Asi, J. Wang, H. Pinto, Q. Chen, U. Dayal, M.-C. Hsu: Mining sequential patterns by pattern-growth: The PrefixSpan approach, *IEEE TKDE*, **16**, 1–17, 2004.
11. T. Uno: Two general methods to reduce delay and change of enumeration algorithms, NII Technical Report, NII-2003-004E, April 2003.