

TCS Technical Report

Mining Frequent Bipartite Episodes from Event Sequences

by

TAKASHI KATOH, HIROKI ARIMURA, AND KOUICHI
HIRATA

Division of Computer Science

Report Series A

May 21, 2009



Hokkaido University
Graduate School of
Information Science and Technology

Email: arim@ist.hokudai.ac.jp

Phone: +81-011-706-7680

Fax: +81-011-706-7680

Mining Frequent Bipartite Episodes from Event Sequences ^{*}

Takashi Katoh¹, Hiroki Arimura¹, and Kouichi Hirata²

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, first we introduce a *bipartite episode* of the form $A \mapsto B$ for two sets A and B of events, which means that every event of A is followed by every event of B . Then, we present an algorithm that finds all frequent bipartite episodes from an input sequence without duplication in $O(|\Sigma| \cdot N)$ time per an episode and in $O(|\Sigma|^2 n)$ space, where Σ is an alphabet, N is total input size of S , and n is the length of S . Finally, we give experimental results on artificial and real sequences to evaluate the efficiency of the algorithm.

1 Introduction

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.* [10] have introduced *episode mining* to discover frequent *episodes* in an event sequence. Here, an episode is formulated as an acyclic labeled 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.*, [3, 12]). Furthermore, 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 [8, 9, 5, 10], a number of efficient algorithms have been developed so far (in Fig. 1). Mannila *et al.* [10] presented efficient mining algorithm for subclasses of episodes, called *parallel episodes* and *serial episodes*. Mannila *et al.* [10] have designed an algorithm to construct general class of episodes from serial and parallel episodes, which is general but inefficient. On the other hand, in order to capture the direct relationship between premises

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

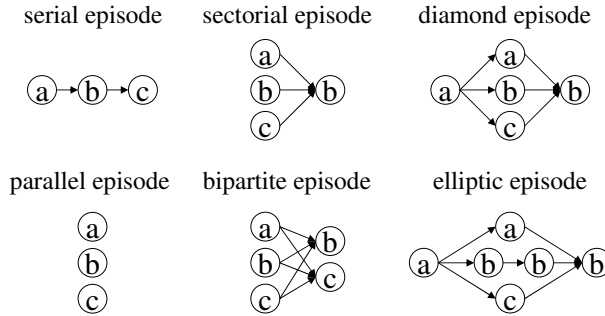


Fig. 1. Examples of subclasses of episode. serial episode (Mannila *et al.* [10]), parallel episode (Mannila *et al.* [10]), sectorial episode (Katoh *et al.* [8]), bipartite episode (this paper), diamond episode (Katoh *et al.* [9]), and elliptic episode (Katoh *et al.* [5]).

and consequences, Katoh *et al.* have introduced *sectorial episodes* [8], *diamond episodes* [9], and *elliptic episodes* [5]. Both episodes have the special events, a source as a premise and a sink as a consequence. In particular, from bacterial culture data [5, 9], they have succeeded to find frequent diamond and elliptic episodes concerned with the replacement of bacteria and the changes for drug resistance, which are valuable from the medical viewpoint. Here, the source and the sink are set to the bacteria and another bacteria for the former episodes, and the sensitivity of antibiotic and the resistant of the same antibiotic for the latter episodes.

Since both diamond and elliptic episodes have just a single source and a single sink, they can represent no relationship between plural premises and plural consequences. On the other hand, plural sources or plural sinks are necessary to represent episodes that plural premises or plural consequences simultaneously occur. Hence, as the simplest forms of episodes with plural sources and plural sinks, in this paper, we newly introduce *bipartite episodes* of the form $A \mapsto B$, where A and B are sets of events. The bipartite episode $A \mapsto B$ means that every event of A is followed by every event of B , so A and B are regarded as the sets of sources and sinks, respectively, and the graph representation of it forms a directed bipartite graph from A to B .

By paying our attention to enumeration methods, Katoh *et al.* [5, 9] have designed so called level-wise algorithms for enumerating all of the frequent diamond or elliptic episodes, based on the frequent itemset mining algorithm APRIORI-TID [1]. While the level-wise algorithms are sufficient to find frequent episodes efficiently in practice, it is difficult to give theoretical guarantee of the efficiency. Recently, in order to give such theoretical guarantee, Katoh *et al.* [7] have developed the enumeration algorithm for frequent diamond episodes in polynomial delay and in polynomial space, based on depth-first search.

In this paper, we design the algorithm BIPAR to enumerate frequent bipartite episodes efficiently based on depth-first search. Then, it finds all of the frequent

bipartite episodes in an input sequence \mathcal{S} without duplication in $O(|\Sigma|N)$ time per an episode and in $O(|\Sigma|^2n)$ space, where $|\Sigma|$, n , and N are an alphabet size, the length of \mathcal{S} , and the total size of \mathcal{S} , respectively. Hence, we can enumerate frequent bipartite episodes in polynomial delay and in polynomial space. We also give the incremental computation for occurrences, and practical speed-up by dynamic programming and prefix-based classes.

This paper is organized as follows. In Section 2, we introduce bipartite episodes and other notions necessary to the later discussion. In Section 3, we discuss several properties of bipartite episodes. In Section 4, we present the algorithm BIPAR and show its correctness and complexity. In Section 5, we give some experimental results from randomly generated event sequences to evaluate the practical performance of the algorithms. In Section 6, we conclude this paper and discuss the future works.

2 Preliminaries

In this section, we introduce the class of bipartite episodes 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 (*arcs*). A *directed acyclic graph* (*dag*, for short) is a digraph without cycles.

2.1 An input event sequence and its windows

Let $\Sigma = \{1, \dots, m\}$ ($m \geq 1$) be a finite alphabet with the total order \leq over \mathbf{N} . Each element $e \in \Sigma$ is called an *event*³. Let *null* be the special, smallest event, called the *null event*, such that $a < \text{null}$ for all $a \in \Sigma$. Then, we define $\max \emptyset = \text{null}$. An *input event sequence* (*input sequence*, for short) \mathcal{S} on Σ is a finite sequence $\langle S_1, \dots, S_n \rangle \in (2^\Sigma)^*$ 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 define $S_i = \emptyset$. Then, we define n the *length* of \mathcal{S} by $|\mathcal{S}| = n$ and define the *total size* of \mathcal{S} by $\|\mathcal{S}\| = \sum_{i=1}^n |S_i|$. Clearly, $\|\mathcal{S}\| = O(|\Sigma|n)$, but the converse is not always true.

2.2 Episodes

Mannila *et al.* [10] defined an episode as a partially ordered set of labeled nodes.

Definition 1 (Mannila *et al.* [10]). A labeled acyclic digraph $X = (V, E, g)$ is an *episode* over Σ where V is a set of nodes, $E \subseteq V \times V$ is a set of arcs and $g : V \rightarrow \Sigma$ is a mapping associating each node with an event.

An episode is an acyclic digraph in the above definition, while it is define as a partial order in Mannila *et al.* [10]. It is not hard to see that two definitions are essentially same each other. For an arc set E on a vertex set V , let E^+ be the *transitive closure* of E such that $E^+ = \{(u, v) \mid \text{there is some directed path from } u \text{ to } v\}$.

³ Mannila *et al.* [10] 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*.

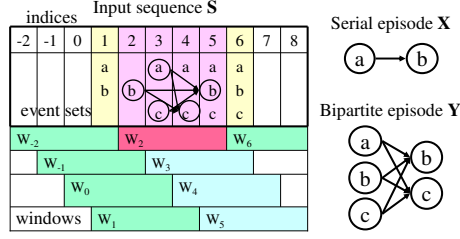


Fig. 2. (Left) An input sequence $\mathcal{S} = (S_1, \dots, S_6)$ of length $n = 6$ over $\Sigma = \{a, b, c\}$ and their k -windows. (Right) Serial episode $X = a \mapsto b$ and a bipartite episode $Y = (\{a, b, c\} \mapsto \{b, c\})$. In the sequence \mathcal{S} , we indicate an occurrence (embedding) of Y in the second window W_2 in circles and arrows. See Example 1 and 2 for details.

Definition 2 (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 some mapping $f : V_1 \rightarrow V_2$ such that (i) f preserves vertex labels, i.e., for all $v \in V_1$, $g_1(v) = g_2(f(v))$, and (ii) f preserves 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 $\mathcal{S} = \langle S_1, \dots, S_n \rangle \in (2^\Sigma)^*$, an *window* in \mathcal{S} is a contiguous subsequence $W = \langle S_i \cdots S_{i+k-1} \rangle \in (2^\Sigma)^*$ of \mathcal{S} for some i , where $k \geq 0$ is the *width* of W .

Definition 3 (occurrence for an episode). An episode $X = (V, E, g)$ occurs in an window $W = \langle S_1 \cdots S_k \rangle \in (2^\Sigma)^*$, denoted by $X \sqsubseteq W$, if there exists some mapping $h : V \rightarrow \{1, \dots, k\}$ such that (i) h preserves vertex labels, i.e., for all $v \in V$, $g(v) \in S_{h(v)}$, and (ii) h preserves 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 in the above definition is called an *embedding* of X into W .

An *window width* is a fixed positive integer $1 \leq k \leq n$. For any $-k+1 \leq i \leq n$, we say that an episode X occurs at position i in \mathcal{S} if $X \sqsubseteq W_i$, where $W_i = \langle S_i, \dots, S_{i+k-1} \rangle$ is the i -th window of width k 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 $\mathbf{W}_i^{\mathcal{S}, k}$. Let $\mathbf{W}_{\mathcal{S}, k} = \{i \mid -k+1 \leq i \leq n\}$ be the domain of occurrences. For an episode X , we define the *occurrence list* for X in \mathcal{S} by $\mathbf{W}_{\mathcal{S}, k}(X) = \{-k+1 \leq i \leq n \mid X \sqsubseteq W_i\}$, the set of occurrences of X in an input \mathcal{S} .

Example 1. Consider an alphabet $\Sigma = \{a, b, c\}$ and an input event sequence $\mathcal{S} = \langle \{a, b\}, \{b\}, \{a, c\}, \{a, c\}, \{a, b, c\}, \{a, b, c\} \rangle$ in Figure 2. Then, if the window width k is 4, has nine 4-windows from W_{-2} to W_6 for all $-2 \leq i \leq 6$, i.e., $\mathbf{W}_{\mathcal{S}, 5} = \{W_i \mid -2 \leq i \leq 6\}$.

Let \mathcal{C} be a subclass of episodes, \mathcal{S} be an input sequence, and $k \geq 1$ a window width. Let $X \in \mathcal{C}$ be an episode in the class \mathcal{C} . The *frequency* of X in \mathcal{S} is defined

by the number of k -windows $\text{freq}_{\mathcal{S},k}(X) = |\mathbf{W}_{\mathcal{S},k}(X)| = O(n)$. A *minimum frequency threshold* is any positive integer $\sigma \geq 1$. Without loss of generality that $\sigma \leq |\mathbf{W}_{\mathcal{S},k}|$ for the length n of \mathcal{S} . Then, the episode X is σ -frequent in \mathcal{S} if $\text{freq}_{\mathcal{S},k}(X) \geq \sigma$. We denote by $\mathcal{F}_{\mathcal{S},k,\sigma}$ be the set of all σ -frequent episodes occurring in \mathcal{S} . Let \mathcal{C} be a subclass of episodes we consider.

Definition 4. FREQUENT EPISODE MINING PROBLEM FOR \mathcal{C} :

Given an input sequence $\mathcal{S} \in (2^\Sigma)^*$, an window width $k \geq 1$, and a minimum frequency threshold $\sigma \geq 1$, the task is to find all σ -frequent episodes X within class \mathcal{C} that occur in \mathcal{S} with window width k without duplicates.

Our goal is to design an efficient algorithm for the frequent episode mining problem in the framework of enumeration algorithms [2, 4]. 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 Bipartite Episodes

3.1 Definition

Definition 5. For $m \geq 1$, m -serial episode (or serial episode) over Σ is a sequence $P = (a_1 \mapsto \dots \mapsto a_m)$ of events $a_1, \dots, a_m \in \Sigma$. This P represents an episode $X = (V, E, g)$, where $V = \{v_1 \dots v_m\}$, $E = \{(v_i, v_{i+1}) \mid 1 \leq i < m\}$, and $g(i) = a_i$ for every $i = 1, \dots, m$.

Definition 6. An episode $X = (V, E, g)$ is a *partial bipartite episode* (or *partial bi-episode*) if the underlying acyclic digraph X is bipartite, i.e., (i) $V = V_1 \cup V_2$ for mutually disjoint sets V_1, V_2 , (ii) for every arc $(x, y) \in E$, $(x, y) \in V_1 \times V_2$. Then, we call V_1 and V_2 the source and sink sets.

Definition 7. A *bipartite episode* (*bi-episode, for short*) is an episode $X = (V, E, g)$ that satisfies the following conditions (i) – (iii):

- (i) X is a partial bipartite episode with $V = V_1 \cup V_2$.
- (ii) X is *complete*, i.e., $E = V_1 \times V_2$ holds.
- (iii) X is *partwise-linear*, that is, for every $i = 1, 2$, the set V_i contains no distinct vertices with the same labeling by g .

In what follows, we represent a bipartite episode $X = (V_1 \cup V_2, E, g)$ by a pair $(A, B) \in 2^\Sigma \times 2^\Sigma$ of two subsets $A, B \subset \Sigma$ of events, or equivalently, an expression $(A \mapsto B)$, where $A = g(V_1)$ and $B = g(V_2)$ are the images of V_1 and V_2 by label mapping g . We also write (a, b) or $(a \mapsto b)$ for a 2-serial episode. In what follows, then, we define the *size* of a bipartite episode X by $\|X\| = |A| + |B| = |V_1| + |V_2|$.

Example 2. In Figure 2, we show examples of an input event sequence $\mathcal{S} = \langle \{a, b\}, \{b\}, \{a, c\}, \{a, c\}, \{a, b, c\}, \{a, b, c\} \rangle$ of length $n = 6$, a serial episode $X = a \mapsto b$ and a bipartite episode $Y = (\{a, b, c\} \mapsto \{b, c\})$ on an alphabet of events $\Sigma = \{a, b, c\}$. Then, the window list for a bipartite episode $Y = (\{a, b, c\} \mapsto \{b, c\})$ is $\mathbf{W}(Y) = \{W_2, W_3, W_4\}$.

In what follows, we denote by \mathcal{SE}_k , $\mathcal{SE} = \cup_{k \geq 1} \mathcal{SE}_k$, \mathcal{PE} , \mathcal{SEC} , \mathcal{BE} , \mathcal{DE} , and \mathcal{EE} , respectively, the classes of k -serial, serial, parallel, sectorial, bipartite, diamond, and elliptic episodes over Σ . For subclasses of episodes, the following inclusion relation hold: (i) $\mathcal{SE}_2 \subseteq \mathcal{SEC} \subseteq \mathcal{BE}$ and (ii) $\mathcal{PE} \subseteq \mathcal{BE}$.

3.2 Serial constructibility

In this section, we introduce properties of bipartite episode that are necessary to devise an efficient algorithm for the frequent bipartite episode mining problem. We define the set of all serial episodes embedded in episode X by $Ser(X) = \{S \in \mathcal{SE} \mid S \sqsubseteq X\}$. An episode X is said to be *serially constructible* on Σ if for any input event sequence \mathcal{S} on Σ and for any window W of \mathcal{S} , $X \sqsubseteq W$ holds iff for every serial episode $S \in Ser(X)$, $S \sqsubseteq W$ holds.

Katoh and Hirata [6] gave a necessary and sufficient condition for serially constructibility, called the parallel-freeness.

Definition 8 (Katoh and Hirata [6]). An episode $X = (V, E, g)$ is *parallel-free* if any pair of vertices labeled by the same event are reachable, that is, for any pair of mutually distinct vertices $u, v \in V$ ($u \neq v$), if $g(u) = g(v)$ then there exists a directed path from u to v or v to u in X .

Theorem 1 (Katoh and Hirata [6]). Let Σ be any alphabet. X is parallel-free iff X is serially constructible.

Theorem 2. Let X be partial bi-episode. If X is bipartite then X is parallel-free.

Corollary 1. Any bipartite episode is serially constructible.

Let $X_i = (A_i \mapsto B_i)$ be a bipartite episode for every $i = 1, 2$. We define that $X_1 \subseteq X_2$ if $A_1 \subseteq A_2$ and $B_1 \subseteq B_2$.

Lemma 1 (anti-monotonicity of frequency). Let σ be any frequency threshold and $k \geq 1$ be a window width. Let $X_i = (A_i \mapsto B_i)$ ($i = 1, 2$) be a bipartite episode. If $X_1 \subseteq X_2$ then $freq_{\mathcal{S}, k}(X_1) \geq freq_{\mathcal{S}, k}(X_2)$.

Proof. Let W be any window in \mathcal{S} . Suppose that $X_2 \sqsubseteq W$. By Corollary 1, $S \sqsubseteq W$ for all serial episodes $S \in Ser(X_2)$. Since $X_1 \subseteq X_2$, we can show that $Ser(X_1) \subseteq Ser(X_2)$. For all serial episodes $S \in Ser(X_1)$, it holds that $S \sqsubseteq W$. By Corollary 1, $X_1 \sqsubseteq W$. From the above, if $X_2 \sqsubseteq W$ then $X_1 \sqsubseteq W$ for any W . Therefore, $\mathbf{W}_{\mathcal{S}, k}(W_1) \supseteq \mathbf{W}_{\mathcal{S}, k}(W_2)$. Then, $freq(X_1) \geq freq(X_2)$. \square

Now, we have shown the serial constructibility for bipartite episodes. In the following, however, we further make detailed analysis on the serial constructibility for bipartite episodes by giving a simpler proof of Corollary 1 that does not use Theorem 1. For a window W and an event $e \in \Sigma$, we denote by $st(e, W)$ and $et(e, W)$, respectively, the first and the last positions in W at which e occurs.

Lemma 2 (characterization of the occurrences for a bipartite episode). Let $X = (U, V, A, g)$ be any bipartite episode and W any window in $\mathbf{W}_{\mathcal{S},k}$. Then, $X \sqsubseteq W$ iff $(\max_{u \in U} st(g(u), W)) < (\min_{v \in V} et(g(v), W))$ holds.

Lemma 3 (characterization of the occurrences for a bipartite episode). Let $U = \{u_1, \dots, u_n\}$ and $V = \{v_1, \dots, v_m\}$ ($0 \leq n, 0 \leq m$) be sets of vertices, $X = (U, V, A, g)$ be a bipartite episode and $W = \langle S_1, \dots, S_k \rangle$ a window in $\mathbf{W}_{\mathcal{S},k}$. Then, $X \sqsubseteq W$ iff for any $n, m \geq 0$, $\max(st(g(u_1)), \dots, st(g(u_n))) < \min(et(g(v_1)), \dots, et(g(v_m)))$ hold.

Proof. Let $U = \{u_1, \dots, u_n\}$ and $V = \{v_1, \dots, v_m\}$ ($0 \leq n, 0 \leq m$) and $W = \langle S_1, \dots, S_k \rangle$. We write $st(u)$ for $st(u, W)$ and same for $et(u, W)$. (Only if-direction) If $X \sqsubseteq W$ then there exists some mapping $h : (U \cup V \rightarrow \{1, \dots, k\})$ satisfying (i) for all vertex $v \in (U \cup V)$, $g(v) \in S_{h(v)}$ and (ii) for all vertices $u \in U$ and $v \in V$, $h(u) < h(v)$. For all $1 \leq i \leq n$ and $1 \leq j \leq m$, it holds that $st(g(u_i)) \leq h(u_i)$ and $h(u_j) \leq et(g(u_j))$, respectively. Then, it holds that $\max_{u \in U} st(g(u)) \leq \max_{u \in U} h(u)$ and $\min_{v \in V} h(v) \leq \min_{v \in V} et(g(v))$. By the condition (ii), for all $1 \leq i \leq n$ and $1 \leq j \leq m$, it holds that $h(u_i) < h(v_j)$. Therefore, we obtain the claim. (If-direction) Suppose that $\max_{u \in U} st(g(u)) < \min_{v \in V} et(g(v))$ holds. Then, we can build a mapping h by For all $1 \leq i \leq n$ and $1 \leq j \leq m$, $h(u_i) = st(g(u_j))$ for every $1 \leq i \leq n$ and $h(v_j) = et(g(v_j))$ for every $1 \leq j \leq m$. Then, the claim holds. \square

Lemma 4. For any bipartite episode $X = (A \mapsto B)$, $Ser(X) = A \cup B \cup \{(a \mapsto b) \mid (a, b) \in A \times B\}$

Theorem 3 (a detailed version of serial construction). Let X be a bipartite episode and $W = \langle S_1, \dots, S_k \rangle$ a window in $\mathbf{W}_{\mathcal{S},k}$. Let, $A, B \subseteq \Sigma$ be non-empty sets. Then,

- (1) If $X = (A \mapsto B)$ then, $X \sqsubseteq W$ iff $\forall (a, b) \in A \times B$, $(a \mapsto b) \sqsubseteq W$.
- (2) if $X = (A \mapsto \emptyset)$ or $X = (\emptyset \mapsto B)$, $X \sqsubseteq W$ iff $\forall a \in A \cup B$, $a \sqsubseteq W$.

We define the *merge* of two bipartite episodes $X_i = (A_i \mapsto B_i)$ ($i = 1, 2$) by $X_1 \cup X_2 = (A_1 \cup A_2 \mapsto B_1 \cup B_2)$, such that the edge set is the set unions of their edge sets. The *downward closure property* for a class \mathcal{C} of episodes says that for any episodes $X_1, X_2 \in \mathcal{C}$, the condition $\mathbf{W}_{\mathcal{S},k}(X_1 \cup X_2) = \mathbf{W}_{\mathcal{S},k}(X_1) \cap \mathbf{W}_{\mathcal{S},k}(X_2)$ holds. Unfortunately, the class of bipartite episodes does not satisfy this property in general. The next lemma is essential to fast incremental computation of occurrence lists for the class of bipartite episodes in the next section.

Theorem 4 (downward closure property). Let $X_i = (A_i \mapsto B_i)$ ($i = 1, 2$). For any input sequence \mathcal{S} and any $k \geq 1$, if $A_1 = A_2$ then $\mathbf{W}_{\mathcal{S},k}(X_1 \cup X_2) = \mathbf{W}_{\mathcal{S},k}(X_1) \cap \mathbf{W}_{\mathcal{S},k}(X_2)$.

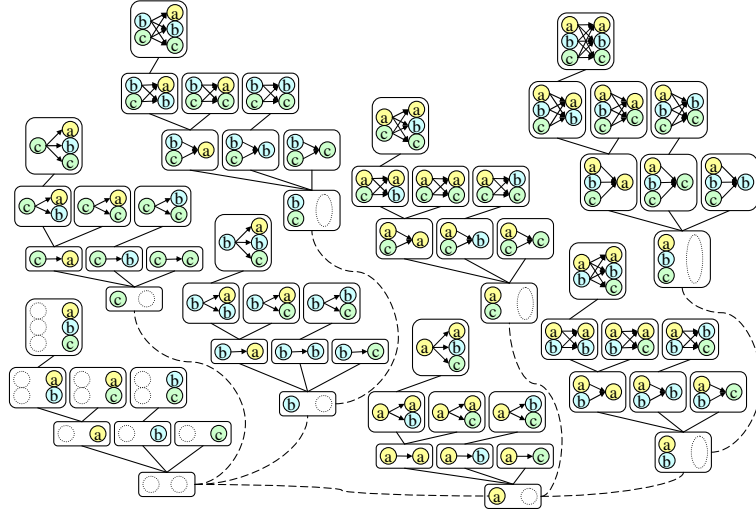


Fig. 3. The parent-child relationships on the alphabet $\Sigma = \{a, b, c\}$, where each white empty circle indicates the empty set.

4 A Polynomial-Delay and Polynomial-Space Algorithm

4.1 The outline of the algorithm

In this section, we present a polynomial-delay and polynomial-space algorithm BIPAR for extracting all frequent bipartite episodes in a given input sequence. Let $\mathcal{S} = (S_1, \dots, S_n) \in (2^\Sigma)^*$ be an input sequence of length n and total input size $N = \|\mathcal{S}\|$, $k \geq 1$ be the window width, and $\sigma \geq 1$ be the minimum frequency threshold.

4.2 Enumeration of bipartite episodes

The main idea of our algorithm is to enumerate all frequent bipartite episodes by searching the whole search space from general to specific using depth-first search. For the search space, we define the parent-child relationships for bipartite episodes.

Definition 9. The bi-episode $\perp = (\emptyset \mapsto \emptyset)$ is the *root*. Then, the parent of a non-root bipartite episode $X = A \mapsto B$ is defined by

$$\text{parent}(A \mapsto B) = \begin{cases} (A - \{\max(A)\}) \mapsto B & (\text{if } B = \emptyset) \\ A \mapsto (B - \{\max(B)\}) & (\text{otherwise}) \end{cases}$$

We define the set of all children of X by $\text{Children}(X) = \{Y \mid \text{parent}(Y) = X\}$. Then, we define the *family tree* for \mathcal{BE} by the rooted digraph $\mathcal{T}(\mathcal{BE}) = (V, E, \perp)$ with the root \perp , the vertex set $V = (\mathcal{BE})$, and the edge set $E =$

algorithm BIPAR($\mathcal{S}, k, \Sigma, \sigma$)
input: input event sequence $\mathcal{S} = \langle S_1, \dots, S_n \rangle \in (2^\Sigma)^*$ of length $n \geq 0$,
window width $k > 0$, alphabet of events Σ , the minimum frequency $1 \leq \sigma \leq n + k$;
output: the set of all σ -frequent bipartite episodes in \mathcal{S} with window width k ;
method:
1 $\perp := (\emptyset \mapsto \emptyset)$; // The root bipartite episode \perp ;
2 FREQBIPARREC($\perp, \mathcal{S}, k, \Sigma, \sigma$);

procedure FREQBIPARREC($X, \mathcal{S}, k, \Sigma, \sigma$)
input: bipartite episode $X = (A \mapsto B)$ and \mathcal{S}, k, Σ , and k are same as in BIPAR.
output: the set of all σ -frequent bipartite episodes in \mathcal{S} that are descendants of X ;
method:
1 **if** ($|\mathbf{W}_{\mathcal{S},k}(X)| \geq \sigma$) **then**
2 **output** X ;
3 // Execute in the special case that the right hand side of X is empty.
4 **if** ($B = \emptyset$) **then**
5 **foreach** $e \in \Sigma$ **such that** $e > \max(A)$ **do**
6 // Expand the left hand side.
7 FREQBIPARREC($((A \cup \{e\}) \mapsto B), \mathcal{S}, k, \Sigma, \sigma)$);
8 **end if**
9 // Execute always.
10 **foreach** ($e \in \Sigma$ ($e > \max(B)$)) **do**
11 // Expand the right hand side.
12 FREQBIPARREC($(A \mapsto (B \cup \{e\})), \mathcal{S}, k, \Sigma, \sigma)$);
13 **end if**

Fig. 4. The main algorithm BIPAR and a recursive subprocedure FREQBIPARREC for mining frequent bipartite episodes in a sequence.

$\{ (X, Y) \mid X \text{ is the parent of } Y, Y \neq \perp \}$. As shown in Fig. 3, we can show that the family tree $\mathcal{T}(\mathcal{BE})$ forms the spanning tree for all bi-episodes of \mathcal{BE} .

In Fig. 4, we show the basic version of our polynomial-delay and polynomial-space algorithm BIPAR and its subprocedure FREQBIPARREC for extracting frequent bipartite episodes from input sequence \mathcal{S} . The algorithm is a backtracking algorithm that traverses the spanning tree $\mathcal{T}(\mathcal{BE})$ based on depth-first search starting from the root \perp using the parent-child relationships over \mathcal{BE} .

The subprocedure BIPAROC is a straightforward algorithm that computes the occurrence list $\mathbf{W}_{\mathcal{S},k}(X)$ for bi-episode X by testing the embedding $X \sqsubseteq \mathbf{W}_i^{\mathcal{S},k}$ for each position i while scanning the input sequence. Its definition is omitted here. We can show that BIPAROC computes $\mathbf{W}_{\mathcal{S},k}(X)$ from X of size $m = \|X\|$ and an input sequence \mathcal{S} of length n in $O(|\Sigma|kmn)$ time.

In Fig. 5, we show the algorithm for computing the occurrence list of a bipartite episode.

Lemma 5. *Let \mathcal{S} be any input sequence of length n . For any window width $k \geq 1$ and minimum frequency threshold $\sigma \geq 1$, the algorithm BIPAR in Fig. 4*

```

procedure BIPAROCC( $X, k, \mathcal{S}$ )
input: bipartite episode  $X = (A \mapsto B)$ ,
window width  $k > 0$ , an input sequence  $\mathcal{S} = \langle S_1, \dots, S_n \rangle$ ;
output: the occurrence list  $W$  for  $X$ ;
method:
1  $W := \emptyset$ ;
2 for ( $i := -k + 1, \dots, n$ ) do
3   if ( $X \sqsubseteq \mathbf{W}_i^{\mathcal{S}, k}$ ) then  $W := W \cup \{i\}$ ;
4 return  $W$ ;

```

Fig. 5. An algorithm BIPAROCC for computing the occurrence list of a bipartite episode.

```

procedure BIPAROCCINC( $X, X_0, W_0, k, \mathcal{S}$ )
input: bipartite episodes  $X = (A \mapsto (B_0 \cup \{e\}))$  and  $X_0 = (A \mapsto B_0) = \text{parent}(X)$ ,
the occurrence list  $W_0$  for  $X_0$ , window width  $k > 0$ 
, an input sequence  $\mathcal{S} = \langle S_1, \dots, S_n \rangle$ ;
output: the occurrence list  $W$  for  $X$ ;
method:
1  $W := W_0$ ;
2 if ( $A = \emptyset$ ) then  $W := W \cap \text{SERIALOCC}((\emptyset \mapsto \{e\}), k, \mathcal{S})$ ;
3 else
4   foreach ( $a \in A$ )  $W := W \cap \text{SERIALOCC}((a \mapsto e), k, \mathcal{S})$ ;
5 return  $W$ ;

```

Fig. 6. An improved algorithm BIPAROCCINC for computing the occurrence list of a bipartite episode.

with BIPAROCC finds all σ -frequent bipartite episodes occurring in \mathcal{S} without duplicates in $O(|\Sigma|^4 kn)$ delay and $O(|\Sigma|^2 + n)$ space.

4.3 Incremental computation of occurrences

The algorithm BIPAROCCINC in Fig.6 computes the occurrence list $W = \mathbf{W}_{\mathcal{S}, k}(Y)$ for the newly created child episode $Y = (A \mapsto B \cup \{b\})$ from the list $\mathbf{W}_{\mathcal{S}, k}(X)$ for its parent $X = (A \mapsto B)$ by calling the subprocedure SERIALOCC. The next lemma is derived from Lemma 5 on the downward closure property for \mathcal{BE} .

Lemma 6 (correctness of BIPAROCCINC). *Let $X = (A \mapsto B)$ be a bipartite episode and $e \in \Sigma$ be an event. Then, we have the next (1) and (2):*

- (1) *If $A = \emptyset$ then $\mathbf{W}(A \mapsto (B \cup \{e\})) = \mathbf{W}(X) \cap \mathbf{W}(\emptyset \mapsto \{e\})$.*
- (2) *if $A \neq \emptyset$ then $\mathbf{W}(A \mapsto (B \cup \{e\})) = \mathbf{W}(X) \cap \bigcap_{a \in A} \mathbf{W}(a \mapsto \{e\})$.*

The algorithm BIPAROCCINC uses the subprocedure SERIALOCC for computing the occurrence list for a 2-serial episode. This algorithm is a modification of FASTSERIALOCC for 3-serial episodes in [7] and its definition is omitted here. We can show that SERIALOCC can be implemented to run in $O(N) = \|\mathcal{S}\|$ time in the total input size $N = \|\mathcal{S}\|$ regardless window width k .

```

procedure SERIALOCC( $X, k, \mathcal{S}$ )
input: serial episode  $X = (a \mapsto b)$ , window width  $k > 0$ ,
an input sequence  $\mathcal{S} = \langle S_1, \dots, S_n \rangle$ ;
output: the occurrence list  $W$  for  $X$ ;
method:
1  $W := \emptyset; (x, y) := (0, 0);$ 
2 for ( $i := -k + 1, \dots, n$ ) do
3    $last := i - 1; end := i + k$ 
4   while  $x < end$  and (not ( $x > last$  and  $a \in S_x$ )) do  $x := x + 1;$ 
5   while  $z < end$  and (not ( $z > x$  and  $b \in S_y$ )) do  $y := y + 1;$ 
6   if ( $last < x < y < end$ ) then  $W := W \cup \{i\};$ 
7      $//(x, y)$  is the lexicographically first occurrence of  $X$  in  $W_i$ ;
8 end for
9 return  $W;$ 

```

Fig. 7. A linear-time algorithm SERIALOCC for the occurrence list of a 2-serial episode.

Fig. 7 shows the algorithm SERIALOCC for computation of the occurrence list for a 2-serial episode, which is a modification of FASTSERIALOCC in [7]. We can show that SERIALOCC can be implemented to run in $O(N) = \|\mathcal{S}\|$ time in the total input size $N = \|\mathcal{S}\|$ regardless window width k .

Lemma 7. *The algorithm SERIALOCC in Fig. 7 computes the occurrence list for a 2-serial episodes $X = a \mapsto b$ in an input sequence \mathcal{S} of length n in $O(N) = O(|\Sigma|n)$ time regardless window width k , where $N = \|\mathcal{S}\|$.*

Lemma 8. *The algorithm BIPAROCCINC in Fig.6 computes the new occurrence list $W = \mathbf{W}_{\mathcal{S},k}(Y)$ for the child episode $Y = (A \mapsto B \cup \{b\})$ in $O(N|A|) = O(|\Sigma|^2 n)$ time from a bi-episode $X = (A \mapsto B)$, $\mathbf{W}_{\mathcal{S},k}(X)$, any event $b \in \Sigma$, and k , where $n = |\mathcal{S}|$ and $N = \|\mathcal{S}\|$.*

4.4 Practical improvement by dynamic programming

We can further improve the computation of occurrence list by BIPAROCCINC using dynamic programming technique as follows.

During the execution of the algorithm FREQBIPARREC the subprocedure SERIALOCC for \mathcal{SE} are called many times inside BIPAROCCINC with the same arguments $(a \mapsto b, k, \mathcal{S})$ ($a, b \in \Sigma$). Fig. 8 shows the algorithm LOOKUPSERIALOCC that is a modification version of SERIALOCC using dynamic programming. This algorithm uses a hash table *TABLE* in Fig. 8 that stores pairs $\langle X, \mathbf{W}(X) \rangle$ of a 2-serial episode $X = (a \mapsto b)$ and its occurrence list $\mathbf{W}(X)$

We modify the main algorithm BIPAR and BIPAROCCINC such that after initializing the hash table, we call LOOKUPSERIALOCC instead of SERIALOCC. This modification does not change the behavior, while it reduces the total number of the calls for SERIALOCC from at most $|\Sigma||\mathcal{F}|$ to at most $|\Sigma|^2$, where $\mathcal{F} \subseteq \mathcal{BE}$ is the set of solutions.

global variable: a hash table $TABLE : \Sigma^2 \rightarrow 2^{\{-k+1, \dots, n\}}$;
initialization: $TABLE := \emptyset$;

procedure LOOKUPSERIALOCC($X, k \in \mathbf{N}, \mathcal{S}$)
input: serial episode $X = (a \mapsto b)$, window width $k > 0$,
an input sequence $\mathcal{S} = \langle S_1, \dots, S_n \rangle$;
output: the occurrence list W for X ;
method:

- 1 **if** ($TABLE[(a, b)] = UNDEF$) **then**
- 2 $W := \text{SERIALOCC}((a \mapsto b), k, \mathcal{S})$;
- 3 $TABLE := TABLE \cup \{(a, b), W\}$;
- 4 **end if**;
- 5 **return** $TABLE[(a, b)]$;

Fig. 8. Practical speed-up for computing occurrence lists of serial episodes using dynamic programming.

Lemma 9. *After initializing the hash table $TABLE$, the algorithm LOOKUPSERIALOCC calls SERIALOCC at most $O(|\Sigma|^2)$ times during the execution of the main procedure BIPAR using $O(|\Sigma|^2 n)$ memory.*

4.5 Reducing the number of scan on the an input sequence by prefix-based classes

We can improve the computation of occurrence list by BIPAROCCINC using the idea of prefix-based classes, which is originally invented by Zaki [14, 15].

For a bipartite episode $P = (A, B)$, called a *common prefix*, we define the *prefix-based class* related to P by the set of bi-episodes

$$\mathcal{C}_P = \{ X = (A \mapsto B \cup \{b\}) \mid P = (A \mapsto B), b \in \Sigma, \max B < b \}.$$

In our modified algorithm BIPARFAST, we enumerate each prefix-based classes for \mathcal{BE} instead of each episode in \mathcal{BE} . We start with defining enumeration procedure of bi-episodes using prefix-based classes induced in a new class of family trees for \mathcal{BE} . We define the parent function $parent : \mathcal{BE} \setminus \{\perp\} \rightarrow \mathcal{BE}$.

Definition 10. For any non-root bipartite episode $X = A \mapsto B$,

$$parent(A \mapsto B) = \begin{cases} ((B - \{\max B\}) \mapsto \emptyset) & \text{if } A = \emptyset, B \neq \emptyset \\ ((A - \{\max A\}) \mapsto B) & \text{if } A \neq \emptyset, |B| = 0 \text{ or } |B| = 1 \\ (A \mapsto (B - \{\max B\})) & \text{if } A \neq \emptyset, |B| \geq 2 \end{cases}$$

By using the parent function above, we can define the family tree $\mathcal{T} = (V, E, \perp)$ in a similar way as in Section 4.2. As shown in Fig. 9, we can show that the family tree $\mathcal{T}(\mathcal{BE})$ forms the spanning tree for all bi-episodes of \mathcal{BE} .

Next, we give a procedure to enumerate all bi-partite episodes based on depth-first search on \mathcal{T} . Starting with $\perp = (\emptyset, \emptyset)$, we enumerate bi-episodes in \mathcal{BE} by the following rules.

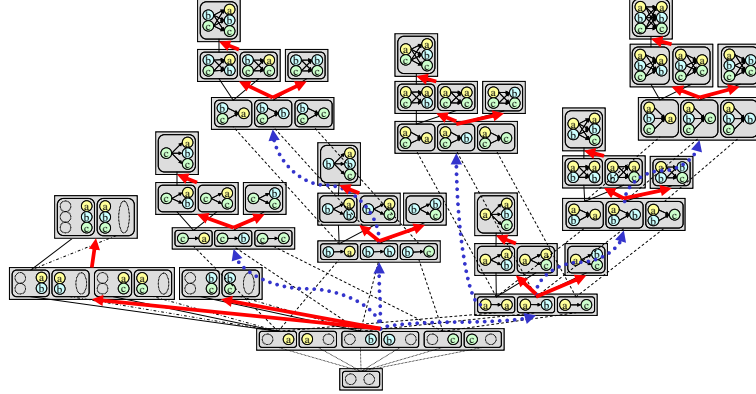


Fig. 9. The improved parent-child relationships on the alphabet $\Sigma = \{a, b, c\}$ (Lines) and the image of recursive calls in algorithm BIPARFAST (Arrows).

Lemma 10. For any bi-episodes $X, Y \in \mathcal{BE}$, Y is a child of X if and only if Y is obtained from X by applying one of the following rules to X . The new occurrence list $\mathbf{W}(Y)$ is also obtained by the corresponding rule.

- (i) If $X = (A \mapsto \emptyset)$, then for any $e \in \Sigma$, $Y = (\emptyset \mapsto A)$ and $\mathbf{W}(Y) = \mathbf{W}(X)$.
- (ii) If $X = (A \mapsto \emptyset)$, then for any $e \in \Sigma$, $Y = (A \cup \{e\} \mapsto \emptyset)$ and $\mathbf{W}(Y) = \mathbf{W}(X) \cap \mathbf{W}(e)$.
- (iii) If $X = (A \mapsto \{b\})$, then for any $e \in \Sigma$, $Y = (A \cup \{e\} \mapsto \{b\})$ and $\mathbf{W}(Y) = \mathbf{W}(X) \cap \mathbf{W}((e \mapsto b))$.
- (iv) If $X = (A \mapsto C \cup \{a\}) \in \mathcal{C}_P$, then for any $Z = (A \mapsto C \cup \{b\}) \in \mathcal{C}_P$ such that $a < b$, $Y = (A \mapsto C \cup \{a, b\})$, where \mathcal{C}_P is the unique prefix-based class to which X belongs, and $\mathbf{W}(Y) = \mathbf{W}(X) \cap \mathbf{W}(Z)$.

Proof. The statements (i) – (iii) are easily proved by construction of the parents. In statements (iv), it follows from the condition $\max B < a, b$ and $a < b$ that the parent for Y is uniquely determined to be X . The proof for the property $\mathbf{W}(Y) = \mathbf{W}(X) \cap \mathbf{W}(Z)$ follows from Theorem 5 on downward closure property for \mathcal{BE} . \square

By the above lemma, provided that each prefix-based class \mathcal{C}_P is available, we do not need to compute the new occurrence lists $\mathbf{W}(Y)$ for each child in the cases of (i) and (iv). We have to explicitly compute the occurrence lists only for a single event in case (i) and for a 2-serial episodes in case (iii).

Example 3. Consider an alphabet $\Sigma = \{a, b, c\}$ and an input event sequence $\mathcal{S} = \langle \{a, b\}, \{b\}, \{a, c\}, \{a, c\}, \{a, b, c\}, \{a, b, c\} \rangle$ in Figure 2. If the window width k is 4, we show that $\mathbf{W}(X_1 = \{a, b\} \mapsto \{a\}) = \{0, 1, 2, 3, 4, 5\}$, $\mathbf{W}(X_2 = \{a, b\} \mapsto \{b\}) = \{-1, 0, 1, 2, 3, 4, 5\}$, and $\mathbf{W}(X_3 = \{a, b\} \mapsto \{c\}) = \{0, 1, 2, 3, 4, 5\}$. Then, we compute the occurrence lists of $Y \in \bigcup_{i=1,2,3} \text{children}(X_i)$ in Table 1.

bipartite episodes	occurrence lists
$X_1 = \{a, b\} \mapsto \{a\}$	$\mathbf{W}(X_1) = \{0, 1, 2, 3, 4, 5\}$
$X_2 = \{a, b\} \mapsto \{b\}$	$\mathbf{W}(X_2) = \{-1, 0, 1, 2, 3, 4, 5\}$
$X_3 = \{a, b\} \mapsto \{c\}$	$\mathbf{W}(X_3) = \{0, 1, 2, 3, 4, 5\}$
$Y_1 = \{a, b\} \mapsto \{a, b\}$	$\mathbf{W}(Y_1) = \mathbf{W}(X_1) \cap \mathbf{W}(X_2) = \{0, 1, 2, 3, 4, 5\}$
$Y_2 = \{a, b\} \mapsto \{a, c\}$	$\mathbf{W}(Y_2) = \mathbf{W}(X_1) \cap \mathbf{W}(X_3) = \{0, 1, 2, 3, 4, 5\}$
$Y_3 = \{a, b\} \mapsto \{b, c\}$	$\mathbf{W}(Y_3) = \mathbf{W}(X_2) \cap \mathbf{W}(X_3) = \{0, 1, 2, 3, 4, 5\}$
$Y_4 = \{a, b, c\} \mapsto \{a\}$	$\mathbf{W}(Y_4) = \mathbf{W}(X_1) \cap \mathbf{W}(c \mapsto a) = \{1, 2, 3, 4, 5\}$
$Y_5 = \{a, b, c\} \mapsto \{b\}$	$\mathbf{W}(Y_5) = \mathbf{W}(X_2) \cap \mathbf{W}(c \mapsto b) = \{2, 3, 4, 5\}$
$Y_6 = \{a, b, c\} \mapsto \{c\}$	$\mathbf{W}(Y_6) = \mathbf{W}(X_3) \cap \mathbf{W}(c \mapsto c) = \{1, 2, 3, 4, 5\}$

Table 1. Bipartite episodes X_i ; ($i = 1, 2, 3$) and Y_i ; ($1 \leq i \leq 6$) in Fig. 2 and whose occurrence lists.

algorithm BIPARFAST($\mathcal{S}, k, \Sigma, \sigma$)
input: input event sequence $\mathcal{S} = \langle S_1, \dots, S_n \rangle \in (2^\Sigma)^*$ of length $n \geq 0$,
window width $k > 0$, alphabet of events Σ ,
the minimum frequency $1 \leq \sigma \leq n + k$;
output: the set of all σ -frequent bipartite episodes ($A \mapsto B$)
such that $B \neq \emptyset$ in \mathcal{S} with window width k ;
method:
1 **if** ($|W_{\mathcal{S},k}(\emptyset \mapsto \emptyset)| \geq \sigma$) **then**
2 $\mathbf{X}_r := \emptyset$; // the set of pairs of a bipartite episode X and whose occurrence list;
3 **foreach** ($e \in \Sigma$) $\mathbf{X}_r := \mathbf{X}_r \cup (\emptyset \mapsto \{e\}, W_{\mathcal{S},k}(e))$;
4 FASTRECA($\mathbf{X}_r, \mathcal{S}, k, \Sigma, \sigma$);
5 FASTRECB($\mathbf{X}_r, \mathcal{S}, k, \Sigma, \sigma$);
6 **end if**

Fig. 10. The main algorithm BIPARFAST for mining frequent bipartite episodes in a sequence.

Based on the above arguments, in Fig. 10, we present the modified version of our backtracking algorithm, called BIPARFAST, for finding all frequent bipartite episodes in a given input sequence. In Fig. 11, we also show its recursive subprocedures FASTRECA and FASTRECB.

We can apply further improvements to our algorithm BIPARFAST as shown in [7]. We can improve the delay of the algorithm BIPARFAST by the factor of the height of the search tree \mathcal{T} using *alternating output* technique of [13]. We can also reduce the space complexity of the algorithm BIPARFAST from $O(|\Sigma|^3 n)$ to $O(|\Sigma|^2 n)$ space by using the *diffset* technique, of Zaki [15] for itemset mining.

Combining all improvements discussed above, we can modify the basic version of our backtracking algorithm BIPAR. In what follows, we call this modified algorithm by BIPARFAST. Now, we have the main theorem of this paper on the delay and the space complexities of the modified algorithm BIPARFAST.

```

procedure FASTRECA( $\mathbf{X}, \mathcal{S}, k, \Sigma, \sigma$ )
input: the set of pairs  $\mathbf{X} = \{(X, W_{\mathcal{S}, k}(X))\}$ 
of a bipartite episode  $X$  and the occurrence window list for  $X$ ,
 $\mathbf{X}, \mathcal{S}, k, \Sigma$ , and  $k$  are same as in BIPARFAST.
output: the set of all  $\sigma$ -frequent bipartite episodes  $Y$ 
that are descendants of  $X$  in  $(X, Y)$ , where  $(X, W)$  is any element of  $\mathbf{X}$ ;
method:
1   $\mathbf{X}_f := \emptyset$ ;
2   $\mathbf{Y} := \emptyset$ ;
2  foreach (  $(X, W) \in \mathbf{X}$  ) do
3    if (  $|W| \geq \sigma$  )  $\mathbf{X}_f := \mathbf{X}_f \cup (X, W)$ ;
4  foreach (  $((A \mapsto B), W) \in \mathbf{X}_f$  ) bf then
5  output  $A \mapsto B$ ;
6    if (  $A = \emptyset$  ) output  $B \mapsto \emptyset$ ;
7  if (  $\mathbf{X}_f \neq \emptyset$  ) then
8     $A_e \mapsto \{b\} :=$  one of a bipartite episode in  $\mathbf{X}_f$ ;
9     $\mathbf{Y} := \emptyset$ ;
10   foreach (  $e \in \Sigma$  (  $e > \max(A_e)$  ) ) do
11     foreach (  $(A \mapsto \{b\}, W) \in \mathbf{X}_f$  ) do
12        $Y := (A \cup \{e\}) \mapsto \{b\}$ ;
13        $U := W \cap \text{SERIALOCC}(e \mapsto b, k, \mathcal{S})$ ;
14        $\mathbf{Y} := \mathbf{Y} \cup \{(Y, U)\}$ ;
15     end foreach
16     FASTRECA( $\mathbf{Y}, \mathcal{S}, k, \Sigma, \sigma$ );
17     FASTRECB( $\mathbf{Y}, \mathcal{S}, k, \Sigma, \sigma$ );
18   end foreach
19 end if

procedure FASTRECB( $\mathbf{X}, \mathcal{S}, k, \Sigma, \sigma$ )
input and output are same as in FASTRECA.
method:
1   $\mathbf{X}_f := \emptyset$ ;
2   $\mathbf{Y} := \emptyset$ ;
2  foreach (  $(X, W) \in \mathbf{X}$  ) do
3    if (  $|W| \geq \sigma$  )  $\mathbf{X}_f := \mathbf{X}_f \cup (X, W)$ ;
4  foreach (  $(X, W) \in \mathbf{X}_f$  ) output  $X$ ;
5  foreach (  $(A_p \mapsto B_p, W_p) \in \mathbf{X}_f$  ) do
6     $\mathbf{Y} := \emptyset$ ;
7    foreach (  $(A_s \mapsto B_s, W_s) \in \mathbf{X}_f$  (  $\max(B_s) > \max(B_p)$  ) ) do
8       $Y := A_p \mapsto (B_p \cup \{\max(B_s)\})$ ;
9       $U := W_p \cap W_s$ ;
10      $\mathbf{Y} := \mathbf{Y} \cup \{(Y, U)\}$ ;
11   end foreach
12   FASTRECB( $\mathbf{Y}, \mathcal{S}, k, \Sigma, \sigma$ );
13 end foreach

```

Fig. 11. Recursive subprocedures FASTRECA and FASTRECB of the algorithm BIPARFAST.

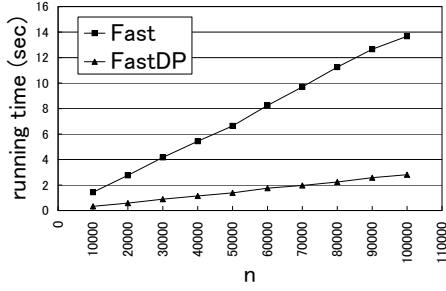


Fig. 12. Running time for the input length n , where $s = 10$, $p = 0.1$, $r = 0.0$, $k = 10$, and $\sigma = 0.1n$.

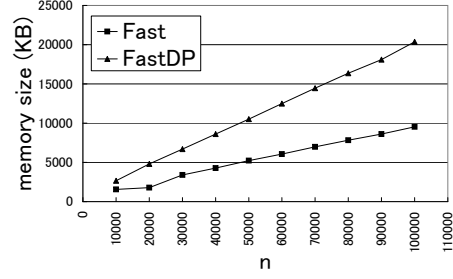


Fig. 13. Memory size for the input length n , where $s = 10$, $p = 0.1$, $r = 0.0$, $k = 10$, and $\sigma = 0.1n$.

Theorem 5. Let \mathcal{S} be any input sequence of length n on event alphabet Σ . For any window width $k \geq 1$ and minimum frequency threshold $\sigma \geq 1$, the algorithm BIPARFAST can be implemented to find all σ -frequent bipartite episodes occurring in \mathcal{S} without duplicates in $O(|\Sigma|N)$ delay (time per frequent episode) and $O(|\Sigma|^2n)$ space, where $N = \|\mathcal{S}\|$ is the total size of input.

5 Experimental Results

In this section, we give the experimental results for the following combinations of the algorithms given in Section 4, by applying to the randomly generated event sequences and the real event sequence.

Data: We made randomly generated event sequences $\mathcal{S} = (S_1, \dots, S_n)$ over an alphabet $\Sigma = \{1, \dots, s\}$ from four parameters (n, s, p, r) , where each event set S_i ($i = 1, \dots, n$) is generated by a probability measures $P(e \in S_i) = p(e/s)^r$ for each $e \in \Sigma$. The real event sequence is made from bacterial culture data provided from Osaka Prefectural General Medical Center from 2000 to 2005 by concatenating detected bacterium for the sample of sputum for every patients.

Method: We implemented the following three depth-first search (DFS) algorithms given in Section 4:

Basic : the basic DFS algorithm BIPARBASIC with OIPAROCC in Fig. 4 (sec. 4.1).

Fast : the modified DFS algorithm BIPARFAST with SREALOCC (sec. 4.5).

FastDP : the modified DFS algorithm BIPARFAST with LOOKUPSREALOCC based on dynamic programming (sec. 4.4).

All experiments were run in a PC (AMD Mobile Athlon64 Processor 3000+, 1.81GHz, 2.00GB memory, Window XP, Visual C++) with window width $K \geq 1$ and minimum frequency threshold $\sigma \geq 1$.

Experiment A: Fig. 12 and Fig. 13 show the running time and the size of virtual memory usage of the algorithms Fast and FastDP for the randomly

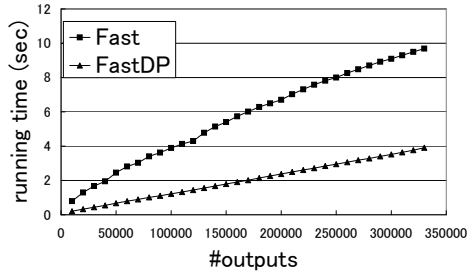


Fig. 14. Running time for the number of outputs, where $n = 10,000$, $s = 10$, $p = 0.1$, $r = 0.0$, $k = 10$, and $\sigma = 0.001n$.

generated event sequences from the parameter ($10000 \leq n \leq 100000$, $s = 10$, $p = 0.1$, $r = 0.0$), where $k = 10$ and $\sigma = 0.1n$. Then, we know that, the running time and the size of virtual memory usage of these algorithms seems to be linear in the input size and thus expected to scales well on large datasets. For this event sequence, **FastDP** is five hundred times as faster as **Fast**. On the other hand, **FastDP** used more memory than **Fast**.

Experiment B: Fig. 14 shows the running time of the algorithms **Fast** and **FastDP** for the number of outputs for the randomly generated event sequences from the parameter ($n = 10,000$, $s = 10$, $p = 0.1$, $r = 0.0$), where $k = 10$ and $\sigma = 0.001n$. Then, we see that the slopes are almost constant and thus the delays are just determined by the input size as indicated by Theorem 5.

exp	exp1	exp2	exp3	exp4	exp5	exp6	exp7
type	rand	rand	rand	rand	rand	bact	bact
n	1,000	10,000	10,000	1,000	100,000	70,606	70,606
s	10	10	1,000	10	10	174	174
p	0.1	0.1	0.1	0.1	0.001		
r	0.0	0.0	10.0	0.0	0.0		
k	10	10	10	100	1,000	15	15
σ	$0.1n$	$0.001n$	$0.25n$	$0.1n$	$0.1n$	$0.01n$	1
#outputs	2,330	334,461	3,512	1,048,576	1,780	162	177,216

Table 2. Parameter settings for Experiment C, where rand and bact indicates a randomly generated data and a bacterial culture data, respectively. The first, second, third, and fourth rows show the name of setting, the data, the parameters, and the number of output episodes, respectively.

Experiment C: Next, we ran seven experiments to compare the algorithms **Basic**, **Fast**, and **FastDP** for various parameter settings par1 – par7 shown in Table 2. As input data, exp1 to exp5 used the randomly generated event se-

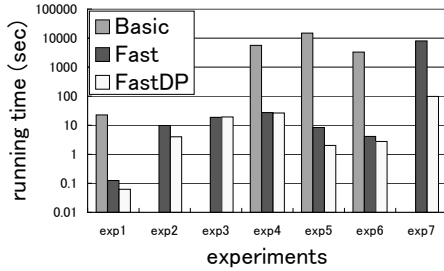


Fig. 15. Running time for exp1 to exp7.

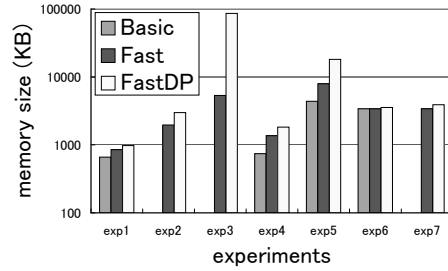


Fig. 16. Memory size for exp1 to exp7.

$$\begin{aligned}
 X &= \{\text{Serratia-marcescens}, \text{Staphylococcus-aureus}\} \\
 &\mapsto \{\text{yeast}, \text{Stenotrophomonas-maltophilia}\}
 \end{aligned}$$

Fig. 17. An example of bipartite episode extracted from a bacterial culture data.

quences with parameters (n, s, p, r) , and exp6 and exp7 used the bacterial culture data.

Fig. 15 and Fig. 16 show the running time and the virtual memory size of the algorithms. For exp2, exp3, and exp7, we did not have the results for Basic because the running time was greater than 20,000 (sec). Then, for exp1, exp4, exp5, and exp6, we know that Fast and FastDP were faster than Basic. especially, for exp5, FastDP was 7300 times faster than Basic. On the other hand, Fast and FastDP were using more memory than Basic. The algorithm FastDP was using more memory than Fast for exp1 to exp7. The algorithm FastDP was the fastest algorithm for exp1 to exp7 expect exp3. For exp3 that was a experiment with large alphabet size $|\Sigma| = 1000$, FastDP was using sixteen (16) times large memory than Fast and Fast was faster than FastDP in this case.

In Fig. 17, we show an example of the bipartite episode X with frequency 21 extracted from the bacterial culture data in exp7, where names in type writer font are the names of bacteria. This episode says that a group of bacteria yeast and Stenotrophomonas-maltophilia are detected after another group of bacteria Serratia-marcescens and Staphylococcus-aureus within fifteen days.

6 Conclusion

This paper studied the problem of frequent bipartite episode mining, and presented an efficient algorithm BIPAR that finds all frequent bipartite episodes in an input sequence in polynomial delay and polynomial space in the input size. We have further studied several techniques for reducing the time and the space complexities of the algorithm.

Possible future problems are extension of BIPAR for general fragments of DAGs [10, 11]. Also, we plan to apply the proposed algorithm to bacterial culture data [5, 9].

References

1. R. Agrawal, R. Srikant: Fast algorithms for mining association rules in large databases, *Proc. 20th VLDB*, 487–499, 1994.
2. H. Arimura: Efficient algorithms for mining frequent and closed patterns from semi-structured data, *Proc. PAKDD'08*, LNAI 5012, 2–13, 2008.
3. 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.
4. D. Avis, K. Fukuda: Reverse search for enumeration, *Discrete Applied Mathematics*, **65**, 21–46, 1996.
5. T. Katoh, K. Hirata: Mining frequent elliptic episodes from event sequences, *Proc. 5th LLLL*, 46–52, 2007.
6. T. Katoh, K. Hirata: A simple characterization on serially constructible episodes, *Proc. PAKDD'08*, LNAI 5012, 600–607, 2008.
7. 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.
8. T. Katoh, K. Hirata, M. Harao: *Mining sectorial episodes from event sequences*, *Proc. 10th DS*, LNAI 4265, 137–145, 2006.
9. T. Katoh, K. Hirata, M. Harao: Mining frequent diamond episodes from event sequences, *Proc. 4th MDAI*, LNAI 4617, 477–488, 2007.
10. H. Mannila, H. Toivonen, A. I. Verkamo: Discovery of frequent episodes in event sequences, *Data Mining and Knowledge Discovery*, **1**, 259–289, 1997.
11. 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.
12. 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 Trans. Knowledge and Data Engineering*, **16**, 1–17, 2004.
13. T. Uno: Two general methods to reduce delay and change of enumeration algorithms, NII Technical Report, NII-2003-004E, April 2003.
14. M. J. Zaki: Scalable Algorithms for Association Mining, *IEEE TKDE*, **12**, 372–390, 2000.
15. M. J. Zaki, C.-J. Hsiao: CHARM: An efficient algorithm for closed itemset mining, *Proc. 2nd SDM*, 457–478, SIAM, 2002.