博 士 論 文

Efficient Algorithms for Extracting Frequent Episodes from Event Sequences

(イベント列からの頻出エピソード抽出に対する効率良いアルゴリズム)

北海道大学大学院情報科学研究科

河 東 孝

Takashi Katoh

Abstract

Episode mining is one of the data mining method for time-related data introduced by Mannila *et al.* in 1997. The purpose of episode mining is to extract all frequent *episodes* from input *event sequences*. Here, the episode is formulated as an *acyclic labeled digraph* in which labels correspond to events and edges represent temporal precedent-subsequent relations in an event sequence. Then, an episode gives a richer representation of temporal relationship than a *subsequence*, which represents just a linearly ordered relation in *sequential pattern mining*.

For the episodes and the subclasses of episodes, several mining algorithms have been developed by several researchers. As such subclasses of episodes, Katoh *et al.* have introduced *sectorial episodes*, *diamond episodes* and *elliptic episodes*. These episodes are simpler than general episode but useful to represent the real-world information that are not represented by *subsequences*.

The algorithms designed by Katoh *et al.* are *level-wise*; The algorithms first find the occurrence information of the serial episodes in an input event sequence, by scanning it just once. After regarding the serial episodes as itemsets, the algorithms then construct the frequent episodes by using one of the frequent itemset mining algorithm that uses breadth-first search over the space of candidate patterns.

While the level-wise algorithms are sufficient to find frequent episodes efficiently in practice it is difficult to give a theoretical guarantee of the efficiency to the level-wise algorithms from the view of enumeration. Moreover, since level-wise algorithms are exponential in memory complexity, the algorithms may not run on the physical

machine with large input.

In this thesis, as a space-efficient episode mining algorithm, we newly design the *episode-growth* algorithms, to enumerate frequent diamond episodes and more general episodes in polynomial time per an output and polynomial space. The algorithms adopts the depth-first search instead of the level-wise search. In Chapter 3, we study the problem of mining *frequent diamond episodes efficiently* from an input event sequence with sliding a window. Here, a diamond episode is of the form $a \mapsto E \mapsto b$, which means that every event of E follows an event a and is followed by an event b. Then, we design a polynomial-delay and polynomial-space algorithm PolyFreqDmd that finds all of the frequent diamond episodes without duplicates from an event sequence in $O(|\Sigma|^2 n)$ time per an episode and in $O(|\Sigma| + n)$ space, where Σ and n are an alphabet and the length of the event sequence, respectively. Finally, we give experimental results on artificial and real-world event sequences with varying several mining parameters to evaluate the efficiency of the algorithm.

In Chapter 4, 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 $\mathcal{S}$, and n is the length of S. Finally, we give experimental results on artificial and real sequences to evaluate the efficiency of the algorithm.

In Chapter 5, 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 time per an output and polynomial space in total input size.

In Chapter 6, we give a simple characterization of episodes in episode mining

that are constructible from just information for occurrences of serial episodes, called *serially constructible* episodes. First, we formulate an episode as an *acyclic transitive labeled digraph* of which label is an event type in episode mining. Next, we introduce a *parallel-free* episode that always has an arc between vertices with the same label. Also we formulate a *serially constructible* episode as an episode embedded into every parallel-free episode containing all of the serial episodes occurring in it. Then, we show that an episode is parallel-free if and only if it is serially constructible.

In Chapter 7, we apply episode mining algorithms to the bacterial culture data and extracted the episodes as the time-related rules representing *replacements of bacteria* and *changes for drug resistance* as the factors of hospital-acquired infection. A *sectorial episode* is of the form $C \mapsto r$, where C is a set of events and r is an event. This sectorial episode means every event type in C is precedent to an event type r. A *sequential episode*, which is the simplest form of serial episodes, is an episode of the form $A \to B$. This sequential episode means that an event type A is precedent to an event type B. An *aligned bipartite episode between the genera of bacteria* is of the form $A \to B$, where A and B are the sets of genera of bacteria, satisfying that every genus in A has the same earliest occurrence time, every genus in B has the same earliest occurrence time, and the former is precedent to the latter. In this chapter, we extract the sectorial episodes, the sequential episodes, and aligned bipartite episodes representing changes for drug resistance and replacements of bacteria from bacterial culture data provided from from Osaka Prefecture General Medical Center.

Finally, we conclude this thesis and discuss future researches in Chapter 8.

Acknowledgments

First and foremost I would like to thank Professor Hiroki Arimura who supervised me from the beginning of my research activity. I would like to thank Associate Professor Kouichi Hirata. He supervised me very kindly since I was an undergraduate student of Kyushu Institute of Technology.

I would thank Dr. Kimiko Matsuoka and Mr. Shigeki Yokoyama for providing bacterial culture data sets at Osaka Prefectural General Medical Center. They gave me a lot of comments from medical view point. I am deeply grateful to Associate Professor Takuya Kida for his guidance and strong support. I would thank Professor Thomas Zeugmann and Professor Shin-ichi Minato for their helpful advice and comments for my research activity.

I wish to thank to Professor Makoto Haraguchi, Professor Hideyuki Imai, Professor Mineichi Kudo, Professor Yuzuru Tanaka, Professor Masaaki Miyakoshi, Professor Miki Haseyama, and Professor Masahiko Onosato for their comments for improving and completing the thesis.

I am deeply indebted to many people who helped and supported me. I particularly appreciate to Professor Masateru Harao, Associate Professor Shougo Ozaki, Professor Takeshi Shinohara, Associate Professor Shinichi Shimozono, Associate Professor Hiroshi Sakamoto, Associate Professor Kimihito Ito, Professor Takeaki Uno, Professor Takashi Washio, Associate Professor Akihiro Inokuchi, Professor Shusaku Tsumoto, Professor Akihiro Yamamoto, Dr. Seishi Okamoto, and Dr. Tatsuya Asai for their helpful discussions.

I appreciate the help from Ms. Kyouko Nanno and Ms. Rieko Kanemitsu. This research was partially supported by Grant-in-Aid for JSPS Fellows. Finally, I thank my parents for their encouragement and support.

Contents

Chapter 1

Introduction

It is one of the most important tasks in data mining [1, 2, 10, 16, 39, 46, 49, 59, 26, 25, 24, 23, 27, 63, 14, 12, 13, 66, 43, 15] to discover frequent patterns from time-related data. For such a task, Mannila *et al.* have introduced the *episode mining* [40] to discover frequent *episodes* in an *event sequence*. Here, the episode is formulated as an *acyclic labeled digraph* in which labels correspond to events and edges represent temporal precedent-subsequent relations in an event sequence. Then, an episode gives a richer representation of temporal relationship than a *subsequence*, which represents just a linearly ordered relation in *sequential pattern mining* (*cf.*, [3, 52, 5, 47, 61, 65]). 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.

Mannila *et al.* [40] have designed an algorithm to construct episodes from a *parallel episode* as a set of events and a *serial episode* as a sequence of events. In Fig. 1.1, we show examples of the parallel episodes and the serial episodes. Unfortunately, their algorithm is general but inefficient, because for each candidate of output episodes, their algorithm needs complex methods to compute its frequency. Since then, several efficient algorithms [48, 9, 7, 8, 45, 70] have been developed by introducing subclasses of episodes in the specific forms that are restricted but useful in various application

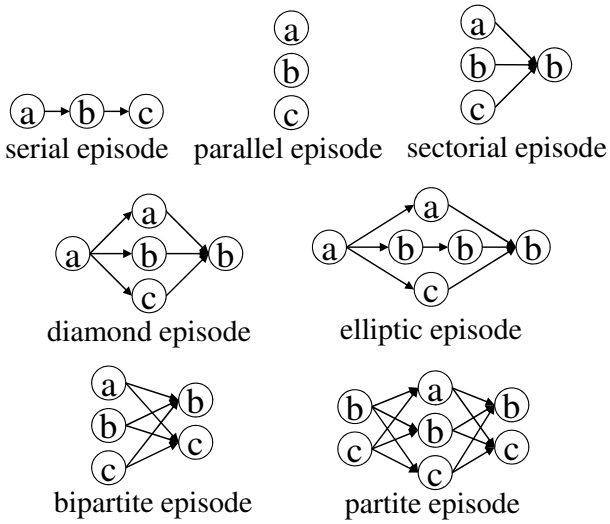


Figure 1.1: Examples of subclasses of episode; serial episode (Mannila *et al.* [40]), parallel episode (Mannila *et al.* [40]), sectorial episode (Katoh *et al.* [36]), diamond episode (Chapter 3), elliptic episode (Katoh *et al.* [32]). bipartite episode (Chapter 4), and partite episode (Chapter 5).

area.

As such restricted but useful subclasses of episodes, Katoh *et al.* have introduced the class of *sectorial episodes* [36], *diamond episodes* [37] and *elliptic episodes* [32]. In Fig. 1.1, we show examples of sectorial episodes, diamond episodes and elliptic episodes. These episodes are simpler than general episode but useful to represent real-world information that are not represented by the class of *subsequences*.

For example, diamond episodes have the special nodes, called a *source* and a *sink*. Then, by setting the source and the sink to the specified event types, we can find frequent episodes with the source as a premise and the sink as a consequence. In particular, from bacterial culture data [32, 37], they have succeeded in finding frequent diamond 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.

Note that the previous algorithms designed by Katoh *et al.* [32, 37] are *level-wise*; algorithms or breadth-first algorithms The algorithms first find the occurrence information of the serial episodes in an input event sequence, by scanning it just once. After regarding the serial episodes as itemsets, the algorithms then construct the frequent episodes by using the frequent itemset mining algorithm AprioriTid [2] that uses breadth-first search over the space of candidate patterns. Although the level-wise algorithms are sufficient to find frequent episodes efficiently in practice, it is difficult to give a theoretical guarantee of the efficiency to the level-wise algorithms from the view of enumeration. Moreover, since level-wise algorithms require exponentially large memory, the algorithms may not run on a large input data with limited amount of memory.

In this thesis, as a space-efficient episode mining algorithm, we newly design the *episode-growth* algorithms, to extract frequent diamond episodes and more general episodes in polynomial time per an output and polynomial space. The algorithms adopts the depth-first search instead of the level-wise search.

In Chapter 3, we study the problem of mining *frequent diamond episodes efficiently* from an input event sequence with sliding a window. Here, a diamond episode is of the form $a \mapsto E \mapsto b$, which means that every event of E follows an event a and is followed by an event b. Then, we design a polynomial-delay and polynomial-space algorithm PolyFreqDmd that finds all of the frequent diamond episodes without duplicates from an event sequence in $O(|\Sigma|^2 n)$ time per an episode and in $O(|\Sigma| + n)$ space, where Σ and n are an alphabet and the length of the event sequence, respectively. Finally, we give experimental results on artificial and real-world event sequences with varying several mining parameters to evaluate the efficiency of the algorithm.

In Chapter 4, 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. In Fig. 1.1, we show examples of the bipartite episode. 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 $\mathcal{S}$, and n is the length of S. Finally, we give experimental results on artificial and real sequences to evaluate the efficiency of the algorithm.

In Chapter 5, 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 $1 \leq i < k$. In Fig. 1.1, we show examples of the partite episode. 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 time per an output and polynomial space in total input size.

In Chapter 6, we give a simple characterization of episodes in episode mining that are constructible from just information for occurrences of serial episodes, called *serially constructible* episodes. First, we formulate an episode as an *acyclic transitive labeled digraph* of which label is an event type in episode mining. Next, we introduce a *parallel-free* episode that always has an arc between vertices with the same label. Also we formulate a *serially constructible* episode as an episode embedded into every parallel-free episode containing all of the serial episodes occurring in it. Then, we show that an episode is parallel-free if and only if it is serially constructible.

In Chapter 7, we apply episode mining algorithms to the bacterial culture data and extracted the episodes as the time-related rules representing *replacements of bacteria* and *changes for drug resistance* as the factors of hospital-acquired infection. A *sectorial episode* is an episode of $C \mapsto r$, where C is a set of events and r is an event. This means every event type in C is precedent to an event type r. A *sequential episode*, which is the simplest form of serial episodes, is an episode of the form $A \to B$. This sequential episode means that an event type A is precedent to an event type B. An *aligned bipartite episode between the genera of bacteria* is of the form $A \to B$, where A and B are the sets of genera of bacteria, satisfying that every genus in A has the same earliest occurrence time, every genus in B has the same

earliest occurrence time, and the former is precedent to the latter. In this chapter, we extract the sectorial episodes, the sequential episodes, and aligned bipartite episodes representing changes for drug resistance and replacements of bacteria from bacterial culture data provided from from Osaka Prefecture General Medical Center.

Finally, we conclude this thesis and discuss future researches in Chapter 8.

Chapter 2

Preliminaries

In this chapter, we introduce the frequent episode mining problem and the related notions necessary to later discussion.

2.1 An input event sequence

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.

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* [*]. Let *null* be the special, smallest event, called the *null event*, such that $a < null$ for all $a \in \Sigma$. Then, we define $\max \emptyset = null$. 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 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, 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 $\mathcal{S}$.

[*] Mannila *et al.* [40] 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*.

2.2 Episodes

Mannila *et al.* [40] defined an episode as a partially ordered set of labeled nodes.

Definition 1 (Mannila *et al.* [40]) 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 $g : V \to \Sigma$ is a mapping associating each vertices 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.* [40]. 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 $E^+ = \{ (u, v) \mid \text{there is some directed path from } u \text{ to } v \}$.

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 $f : V_1 \to V_2$ 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 $\mathcal{S} = \langle S_1, \ldots, S_n \rangle \in (2^\Sigma)^*$, 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 3 (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 4 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.

Let $\mathcal{C}$ be a subclass of episodes. Our goal is to design an efficient algorithm for the frequent episode mining problem for $\mathcal{C}$ in the framework of enumeration algorithms [6]. 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. It is obvious that if $\mathcal{A}$ is of polynomial delay, then so is of output-polynomial.

Chapter 3

Frequent Diamond Episode Mining Problem

In this chapter, we present an efficient mining algorithm for the class of *diamond episodes* from event sequences. For the alphabet Σ and input length ℓ, this algorithm runs in $O(|\Sigma|^2\ell)$ time per an output and in $O(|\Sigma| + \ell)$ space.

As a space-efficient episode mining algorithm, we newly design the *episode-growth* algorithm, called PolyFreqDmd, to enumerate frequent diamond episodes. The algorithm PolyFreqDmd adopts the depth-first search instead of the level-wise search. Then, the algorithm finds all of the frequent diamond episodes in an input sequence $\mathcal{S}$ over an alphabet Σ of events without duplication in $O(|\Sigma|^2\ell)$ time per episode and in $O(|\Sigma| + \ell)$ space, where ℓ is the length of $\mathcal{S}$. Hence, we can guarantee that the episode-growth algorithm PolyFreqDmd enumerates frequent diamond episodes in polynomial delay and in polynomial space. Further, we present some practical optimization techniques such as fast serial episode discovery or memorization for reducing the running time and the required space of the algorithm PolyFreqDmd. In Fig. 3.1, we show examples of an input event sequence, a serial episode, and a diamond episode over an alphabet.

From experiments, we can also evaluate that the implementation of the algorithm

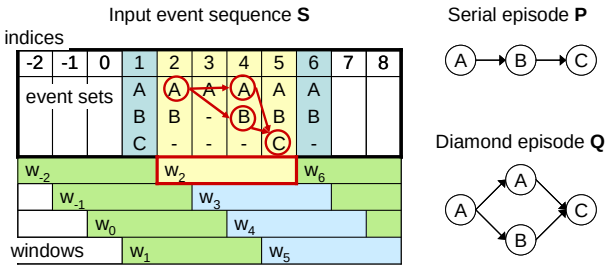


Figure 3.1: (Left) An input sequence $\mathcal{S} = (S_1, \ldots, S_6)$ of length $\ell = 6$ over $\Sigma = \{A, B, C\}$ and their k-windows. (Right) Serial episode $P = A \mapsto B \mapsto C$ and a diamond episode $Q = A \mapsto \{A, B\} \mapsto C$. In the sequence $\mathcal{S}$, we indicate an occurrence (embedding) of Q in the second window W_2 in circles and arrows. See Example 1 and 2 for details.

PolyFreqDmd is efficient on artificial and real-world event sequences. In particular, the implementation DF with MEM (memoization technique) is quite efficient and stable on most data sets. For instance, DF-MEM is one hundred times as fast as others for some parameters.

This chapter is organized as follows. In Section 3.1, we introduce diamond episodes and other notions necessary to the later discussion. In Section 3.2, we present the algorithm PolyFreqDmd and show its correctness and the computational complexity. In Section 3.3, we give a theoretical analysis to compare the algorithm PolyFreqDmd and the level-wise algorithm in [37]. In Section 3.4, we give some experimental results from randomly generated and real-world event sequences to evaluate the practical performance of the algorithms. In Section 3.5, we conclude this chapter and discuss the future works.

This chapter is based on the papers [28, 30].

3.1 Diamond Episodes

For a fixed input sequence $\mathcal{S} = \langle S_1, \ldots, S_\ell \rangle \in (2^\Sigma)^*$, a *position* or an *index* on $\mathcal{S}$ is any integer, where we define S_i for any index such that $i \leq 0$ or $i > \ell$ by $S_i = \emptyset$. Let $1 \leq k \leq \ell$ be a fixed positive integer, called the *window width*. For any index $-k+1 \leq i \leq \ell$, we define the *k-window* $W_i^{\mathcal{S},k}$ *at position* i *in* $\mathcal{S}$ by the contiguous subsequence of length k of $\mathcal{S}$ as follows: $W_i^{\mathcal{S},k} = w_{\mathcal{S}}(i,k) = \langle S_i, \ldots, S_{i+k-1} \rangle \in (2^\Sigma)^k$. We denote the set $\{ W_i^{\mathcal{S},k} \mid -k+1 \leq i \leq \ell \}$ of all k-windows in $\mathcal{S}$ by $\mathbf{W}_{\mathcal{S},k}$. We simply write W_i and $\mathbf{W}$ instead of $W_i^{\mathcal{S},k}$ and $\mathbf{W}_{\mathcal{S},k}$ by omitting the scripts $\mathcal{S}$ and k, respectively, when they are clear from the context. Moreover, we may identify the set of all k-windows by the set $\{ -k+1 \leq i \leq \ell \mid W_i^{\mathcal{S},k} \in \mathbf{W}_{\mathcal{S},k} \} \subseteq \mathbf{Z}$ of their indices.

A *serial episode* over Σ of length $m \geq 0$ (or, *m-serial episode*) is a sequence $P = \langle a_1, \ldots, a_m \rangle \in \Sigma^*$ of events.

Definition 5 A *diamond episode* over Σ is either an event $a \in \Sigma$ (a 1-serial episode) or a triple $Q = \langle a, E, b \rangle \in \Sigma \times 2^\Sigma \times \Sigma$ (called a *proper* diamond episode), where $a, b \in \Sigma$ are events and $E \subset \Sigma$ is a subset of Σ. Then, we call a, b, and E the *source*, the *sink*, and the *body* of Q, respectively. We define the *size* of Q by its body size $||Q|| = |E|$. For the body E, we denote the maximum element in E (with respect to Σ) by $\max(E)$, where $\max(\emptyset)$ is assumed to be the special smallest number ϵ such that $\epsilon < e$ for all $e \in \Sigma$.

To emphasize the chronological dependencies of events, we often write $(e_1 \mapsto \cdots \mapsto e_m)$ and $(a \mapsto E \mapsto b)$ for an m-serial episode $\langle e_1, \ldots, e_m \rangle$ and a diamond episode $\langle a, E, b \rangle$, respectively. Also we denote the classes of m-serial episodes, proper diamond episodes and diamond episodes (over Σ) by $\mathcal{SE}_m$, $\mathcal{PDE}$ and $\mathcal{DE}$, respectively. Since any $(a \mapsto b) \in \mathcal{SE}_2$ and any $(a \mapsto e \mapsto b) \in \mathcal{SE}_3$ are equivalent to $(a \mapsto \emptyset \mapsto b)$ and $(a \mapsto \{e\} \mapsto b) \in \mathcal{PDE}$, respectively, we see that $\mathcal{SE}_1 \cup \mathcal{SE}_2 \cup \mathcal{SE}_3 \cup \mathcal{PDE} = \mathcal{DE}$.

Example 1 In Fig. 3.1, we show examples of an event sequence $\mathcal{S} = (\{A, B, C\}, \{A, B\}, \{A\}, \{A, B\}, \{A, B, C\}, \{A, B\})$ of length 6 over an alphabet $\Sigma = \{A, B, C\}$

of events, a serial episode $P = A \mapsto B \mapsto C$, and a diamond episode $Q = A \mapsto \{A, B\} \mapsto C$.

Next, we introduce the concept of the occurrences of episodes in a window. Then, we give the formal definition of the occurrences of episodes, which is consistent with the original definition in the literature [40]. Let $P = e_1 \mapsto \cdots \mapsto e_m$ be a serial episode, $Q = a \mapsto \{e_1, \ldots, e_m\} \mapsto b$ a diamond episode and $W = \langle S_1, \cdots, S_k \rangle \in \mathbf{W}_{\mathcal{S},k}$ a window, respectively. A serial episode $P = e_1 \mapsto \cdots \mapsto e_m$ *occurs in* a window $W = \langle S_1, \cdots, S_k \rangle \in \mathbf{W}_{\mathcal{S},k}$, denoted by $P \sqsubseteq W$, if and only if there exists some mapping $h : \{1, \ldots, m\} \to \{1, \ldots, k\}$ satisfying (i) $1 \leq h(1) < \cdots < h(m) \leq k$, and (ii) $e_i \in S_{h(i)}$ holds for every $1 \leq i \leq m$.

Definition 6 (occurrence for a diamond episode) A diamond episode $Q = a \mapsto \{e_1, \ldots, e_m\} \mapsto b$ $(m \geq 0)$ *occurs in* a window $W = \langle S_1, \cdots, S_k \rangle \in \mathbf{W}_{\mathcal{S},k}$, denoted by $Q \sqsubseteq W$, if and only if there exists some mapping $h : \{a, b, 1, \ldots, m\} \to \{1, \ldots, k\}$ satisfying (i) for every $i \in \{1, \ldots, m\}$, $1 \leq h(a) < h(i) < h(b) \leq k$ holds, and (ii) $a \in S_{h(a)}$, $b \in S_{h(b)}$ and $e_i \in S_{h(i)}$ holds for every $i \in \{a, b, 1, \ldots, m\}$.

For a window W and an event $e \in \Sigma$, we denote the first and the last position in W at which e occurs by $st(e, W)$ and $et(e, W)$, respectively. The matching algorithm for diamond episodes will be studied in Section 3.

Example 2 Consider an input event sequence $\mathcal{S} = (\{A, B, C\}, \{A, B\}, \{A\}, \{A, B\}, \{A, B, C\}, \{A, B\})$ in Fig. 3.1 again. Then, if the window width k is 4, $\mathcal{S}$ has nine 4-windows from W_{-2} to W_6, i.e., $\mathbf{W}_{\mathcal{S},4} = \{\, W_i \mid -2 \leq i \leq 6 \,\}$. Among them, the occurrence list $\mathbf{W}(Q)$ for a diamond episode $Q = A \mapsto \{A, B\} \mapsto C$ is $\{W_2, W_3\}$.

Lemma 1 *Let Q be a proper diamond episode $(a \mapsto E \mapsto b)$ and W a window in $\boldsymbol{W}_{\mathcal{S},k}$. Then, $Q \sqsubseteq W$ if and only if for every $e \in E$, there exists some position p in W for e such that $st(a, W) < p < et(b, W)$ hold.*

(proof) (Only if-direction) If $Q \sqsubseteq W$ then there exists some embedding h from Q to W. By restricting h to the serial episode $(a \mapsto e \mapsto b)$, we obtain the claim. (If-direction) Suppose that for every $e \in E$, there exists a position p_e for e with $st(a, W) < p_e < et(b, W)$. Then, we can build a mapping h by $h(a) = st(a, W)$, $h(b) = et(b, W)$, and satisfying the claimed property $h(e) = p_e$ for every $e \in E$. Then, the claim holds. □

Lemma 1 implies the following two important lemmas.

Lemma 2 (serial construction for diamond episodes) *Let $E = \{e_1, \ldots, e_m\}$ $(m > 0)$ be a set of events, $Q = (a \mapsto E \mapsto b)$ a partial diamond episode, and $W = \langle S_1, \ldots, S_k \rangle$ a window in $\boldsymbol{W}_{\mathcal{S},k}$. Then, $Q \sqsubseteq W$ if and only if $(a \mapsto e \mapsto b) \sqsubseteq W$ for every $e \in E$.*

(proof) By Lemma 1, we have that $Q \sqsubseteq W$ if and only if there exists some mapping $h : \{a, b, e_1, \ldots, e_m\} \to \{1, \ldots, k\}$ satisfying (i) for every $i \in \{1, \ldots, m\}$, $st(a, W) = h(a) < h(i) < h(b) = et(b, W)$ holds, and (ii) $e_i \in S_{h(i)}$ holds for every $i \in \{a, b, 1, \ldots, m\}$. By the definition of the occurrence for a diamond episode, the result immediately follows. □

Let $\mathcal{S}$ be an input sequence, $k \geq 1$ a window width and Q a diamond episode $a \mapsto E \mapsto b$. The *frequency* of Q in $\mathcal{S}$ is defined by the number of k-windows at which Q occurs $freq_{\mathcal{S},k}(Q) = |\mathbf{W}_{\mathcal{S},k}(Q)|$. A *minimum frequency threshold* (*minimum frequency*, for short) is any integer $1 \leq \sigma \leq |\mathbf{W}_{\mathcal{S},k}|$. A diamond episode Q is *σ-frequent in $\mathcal{S}$* if $freq_{\mathcal{S},k}(Q) \geq \sigma$. We denote by $\mathcal{F}_{\mathcal{S},k,\sigma}$ be the set of all σ-frequent diamond episodes occurring in $\mathcal{S}$.

Lemma 3 (anti-monotonicity for diamond episodes) *Let $a, b \in \Sigma$ be events and $E, F \subseteq \Sigma$ the set of events. For every minimum frequency threshold σ and window width $k \geq 1$, if $E \supseteq F$, then $(a \mapsto E \mapsto b) \in \mathcal{F}_{\mathcal{S},k,\sigma}$ implies $(a \mapsto F \mapsto b) \in \mathcal{F}_{\mathcal{S},k,\sigma}$.*

(proof) From Lemma 2, we have $\mathbf{W}_{\mathcal{S},k}(a \mapsto E \mapsto b) = \bigcap_{e \in E} \mathbf{W}_{\mathcal{S},k}(a \mapsto e \mapsto b)$. Then, we have $|\mathbf{W}_{\mathcal{S},k}(a \mapsto E \mapsto b)| \leq |\mathbf{W}_{\mathcal{S},k}(a \mapsto F \mapsto b)|$. Hence, the result immediately holds. □

Definition 7 Frequent Diamond Episode Mining Problem: Given an input sequence $\mathcal{S}$, a window width $k \geq 1$, and a minimum frequency threshold $\sigma \geq 1$, the task is to find all σ-frequent diamond episodes $Q \in \mathcal{F}_{\mathcal{S},k,\sigma}$ occurring in $\mathcal{S}$ with window width k without duplicates.

In the remainder of this chapter, we design an algorithm for efficiently solving the frequent diamond episode mining problem in the sense of enumeration algorithms.

3.2 Algorithm

In this section, we present a polynomial-delay and polynomial-space algorithm PolyFreqDmd for mining all frequent diamond episodes in a given input sequence. Let Σ be an alphabet of size $s \geq 1$. Let $\mathcal{S} = (S_1, \ldots, S_\ell) \in (2^\Sigma)^*$ be an input sequence of length ℓ and total input size $N = ||\mathcal{S}||$, $k \geq 1$ be the window width, and $\sigma \geq 1$ be the minimum frequency threshold.

3.2.1 Depth first search of frequent diamond episodes

In Fig. 3.2, we show an outline of our polynomial-delay and polynomial-space algorithm PolyFreqDmd and its subprocedure FreqDmdRec for mining frequent diamond episodes in $\mathcal{DE}$ appearing in an input sequence $\mathcal{S}$.

The algorithm PolyFreqDmd is a backtracking algorithm that adopts the depth-first search over the class $\mathcal{FDE}$ of frequent diamond episodes from general to specific. For every pair of events $(a, b) \in \Sigma^2$, PolyFreqDmd starts the depth-first search by calling the recursive procedure FreqDmdRec with the smallest (complete) diamond episode $Q_{ab} = (a \mapsto \emptyset \mapsto b) \in \mathcal{DE}$ and its occurrence list $\mathbf{W}(Q_{ab})$.

By Lemma 3, in each iteration of FreqDmdRec, the algorithm checks whether or not the current candidate $Q = (a \mapsto E \mapsto b)$ is frequent. If so, then FreqDmdRec outputs Q with a body E, and furthermore adds e to E for every $e \in \Sigma$ such that $e > \max(E)$. to its body E. Otherwise, it prunes the search below Q and backtracks

to the parent of Q. Here, we call this process the *tail expansion* for diamond episodes. For episodes Q and R, if R is generated from Q by adding a new event e as above, then we say that Q is a *parent* of R, or R is a *child* of Q. In Fig. 3.3, we show the parent-child relationship on the alphabet $\Sigma = \{a, b, c\}$.

Lemma 4 *For any window width $k > 0$ and any minimum frequency threshold σ, the algorithm* PolyFreqDmd *enumerates all of the frequent diamond episodes from $\mathcal{S}$ without duplicates.*

(proof) Suppose that $R = (a \mapsto E \cup \{e\} \mapsto b) \in \mathcal{DE}$ is a child of some $Q = (a \mapsto E \mapsto b) \in \mathcal{DE}$ obtained by the tail expansion such that $e > \max(E)$. From Lemma 3, we see that any frequent R can be obtained by expanding some frequent parent Q. Furthermore, since $e > \max(E)$, the parent Q is unique for each R. This means that the parent-child relationship forms a spanning tree $\mathcal{T}$ for all frequent diamond episodes in $\mathcal{DE}$. Since FreqDmdRec makes the depth-first search on $\mathcal{T}$ by backtracking, the result immediately follows. □

3.2.2 Incremental algorithm

In the recursive procedure FreqDmdRec in Fig. 3.2, the procedure newly creates a child episode $R = (a \mapsto E \cup \{e\} \mapsto b)$ from the parent $Q = (a \mapsto E \mapsto b)$ by tail expansion with $e \in \Sigma$ at Line 4. Then, at Line 5, it computes the new occurrence list $U = W_{\mathcal{S},k}(R)$ for R in $\mathcal{S}$. To compute the new list U, we can use a straightforward procedure FindSerialOcc that scans all of the k-windows in $\mathcal{S}$ one by one for checking the occurrences of a 3-serial episode P.

Lemma 5 *There is an algorithm that computes an occurrence of a given 3-serial episode $P = a \mapsto e \mapsto b$ in a given window W_i of width k in $O(||W_i||) = O(|\Sigma|k)$, where $||W_i|| = \sum_{j=i}^{i+k-1} |S_j|$.*

(proof) Given a window W_i, we first find the leftmost position x of a with $x \geq 0$, a position y of e with $y \geq x$, and finally a position z of b with $z \geq y$. Such a triple

exists in W_i if and only if P occurs in W_i. The time complexity is obviously $O(||W_i||)$. □

From Lemma 2 and Lemma 5, there is an algorithm that computes the occurrence list of a given diamond episode R in $O(|\Sigma|km\ell)$ time, where k is the window width, $m = ||R||$ is the episode size, and $\ell = |\mathcal{S}|$ is the input length.

In Fig. 3.4, we show an improved algorithm UpdateDmdOcc that computes the new occurrence list $U = W_{\mathcal{S},k}(R)$ from the old one in $O(|\Sigma|k\ell)$ time, by dropping the factor of $m = ||R||$, with incrementally updating the old list W for the parent Q. To see the validity of the improved algorithm, we require the *downward closure property* for $\mathcal{DE}$ shown in Lemma 6 below. In the proof of the property, the *serial construction* for $\mathcal{DE}$ shown in Lemma 2 is used.

Lemma 6 (downward closure property) *Let $a, b \in \Sigma$ and $E \subseteq \Sigma$. Then, for any input sequence $\mathcal{S}$ and any $k \geq 1$, the following statement holds:*

$$\boldsymbol{W}_{\mathcal{S},k}(a \mapsto (E_1 \cup E_2) \mapsto b) = \boldsymbol{W}_{\mathcal{S},k}(a \mapsto E_1 \mapsto b) \cap \boldsymbol{W}_{\mathcal{S},k}(a \mapsto E_2 \mapsto b).$$

(proof) By Lemma 2, we have that $\mathbf{W}_{\mathcal{S},k}(a \mapsto E \mapsto b) = \bigcap_{e \in E}(a \mapsto e \mapsto b)$. Thus, $\mathbf{W}_{\mathcal{S},k}(a \mapsto (E_1 \cup E_2) \mapsto b) = (\bigcap_{e_1 \in E_1} \mathbf{W}_{\mathcal{S},k}(a \mapsto e_1 \mapsto b)) \cap (\bigcap_{e_2 \in E_2} \mathbf{W}_{\mathcal{S},k}(a \mapsto e_2 \mapsto b))$ holds. □

From Lemma 5 and Lemma 6, we see the correctness of the improved algorithm UpdateDmdOcc in Fig. 3.4, and have the next lemma. Note in the following that the computation time of UpdateDmdOcc does not depend on the size $m = ||R||$ of the child episode. If we implement the procedure FindSerialOcc by an algorithm of Lemma 5, we have the next lemma.

Lemma 7 *The algorithm* UpdateDmdOcc *in Fig. 3.4, given the old list W for the parent diamond episode Q and a newly added event e, computes the new occurrence list $U = W_{\mathcal{S},k}(R)$ for a new child R in $O(kN) = O(|\Sigma|k\ell)$ time, where $\ell = |\mathcal{S}|$ and $N = ||\mathcal{S}||$ are the length and the total size of input $\mathcal{S}$, respectively.*

(proof) By Lemma 5, the matching for the i-th window takes $||W_i|| = \sum_{j=i}^{i+k-1} |S_j|$ time for every index i. Summing up this amount of time for all of $\ell + k$ indices $i = -k+1, \ldots, n$, we have total amount $H = \sum_{i=-k+1}^{\ell} ||W_i|| = \sum_{i=-k+1}^{\ell} \sum_{j=i}^{i+k-1} |S_j| \leq \sum_{i=-k+1}^{\ell} k|S_i| = O(kN)$. Thus, the result follows. □

Next, we present a faster algorithm for implementing the procedure FindSerialOcc for serial episodes than that of Lemma 5. In Fig. 3.5, we show the modified algorithm FastFindSerialOcc that computes $\mathbf{W}(P)$ for a 3-serial episode $P = a \mapsto e \mapsto b$ by a single scan of an input sequence $\mathcal{S}$ from left to right.

Lemma 8 *The algorithm* FastFindSerialOcc *in Fig. 3.5 computes the occurrence list of a 3-serial episode $P = a \mapsto e \mapsto b$ in an input sequence $\mathcal{S}$ of length ℓ in $O(N) = O(|\Sigma|\ell)$ time regardless of window width k, where $N = ||\mathcal{S}||$.*

(proof) At each execution of for-loop (from Lines 2 to 9) with position i, we see that after executing three while-loops parameters x, y, and z (in Lines 4 to 6) are the lexicographically first triple of positions such that (i) $x < y < z$, and (ii) $a \in S_x$, $e \in S_y$, and $b \in S_z$. If $last < x < y < z < end$, then this fact implies that an input episode $P = (a \mapsto e \mapsto b)$ occurs in the window W_i. On the contrary, if P occurs in W_i, then we see that the algorithm finds such a triple. For the time complexity, we observe that during the scan of input $\mathcal{S}$ from left to right, each of positions x, y, and z visits each position in $\mathcal{S}$ at most once. This shows that the running time of the algorithm is $O(N)$. This completes the proof. □

Corollary 9 *Equipped with* FastFindSerialOcc *in Fig. 3.5, the modified algorithm* UpdateDmdOcc *computes $U = W_{\mathcal{S},k}(R)$ for a child $R \in \mathcal{DE}$ from the list $W = W_{\mathcal{S},k}(Q)$ for the parent $Q \in \mathcal{DE}$ and $e \in \Sigma$ in $O(N) = O(|\Sigma|\ell)$ time, where $\ell = |\mathcal{S}|$ and $N = ||\mathcal{S}||$.*

3.2.3 Dynamic programming

During the execution of the algorithm FreqDmdRec, the subprocedure FindSerialOcc (or FastFindSerialOcc) for updating occurrence lists are called many

times with the same arguments $(P = (a \mapsto e \mapsto b), k, \mathcal{S})$ $(e \in \Sigma)$. In the worst case, the number of calls may be $|\Sigma|$ in the search paths. Therefore, we can achieve the reduction of the number of calls for FindSerialOcc by memorizing the results of the computation in a hash table $TABLE$.

In Fig. 3.6, we show the code for practical speed-up method using memoization technique. Then, we modify PolyFreqDmd in Fig. 3.2 and UpdateDmdOcc in Fig. 3.4 as follows:

- Before Line 5 of PolyFreqDmd, insert the **initialization** line in Fig. 3.6.
- Replace the call of FindSerialOcc$(P, k, \mathcal{S})$ in FreqDmdRec by the call of LookupSerialOcc$(P, k, \mathcal{S})$ in Fig. 3.6.

This modification does not change the behavior of the procedures PolyFreqDmd, FreqDmdRec, and UpdateDmdOcc. Moreover, this modification makes the total number of calls of FindSerialOcc to be bounded by $|\Sigma|^3$, while it uses $O(|\Sigma|\ell)$ space in main memory. In Section 3.4 below, we will see that this modification will be useful in practice.

The running time of the algorithm FreqDmdRec in Fig. 3.2 mainly depends on the time $T(m, N)$ for the subprocedure UpdateDmdOcc at Line 5 to compute the occurrence list $U = \mathbf{W}_{\mathcal{S},k}(Q)$ of a candidate $Q \in \mathcal{DE}$ in $\mathcal{S}$, where $m = ||Q||$ and $N = ||\mathcal{S}||$.

Unfortunately, if the height of the search tree is $d = \Theta(m) = \Theta(|\Sigma|)$, then the straightforward execution of the algorithm FastFindSerialOcc in Fig. 3.5 yields the delay of $O(d \cdot |\Sigma| \cdot T(m, N))$, where factor d follows from the fact that at least d recursive calls are necessary to come back to the root from the leaf of depth d. We can remove this factor $d = \Theta(m)$ by using a technique called an *alternating output* in backtracking [60], which can be realized by replacing Lines 2 and 8 in the algorithm FreqDmdRec in Fig. 3.2 with the corresponding lines (*) in the comments.

Theorem 10 *Let $\mathcal{S}$ be any input sequence of length ℓ. For any window width $k \geq 1$ and minimum frequency threshold $\sigma \geq 1$, the algorithm* PolyFreqDmd *in Fig. 3.2 finds all σ-frequent diamond episodes Q in $\mathcal{DE}$ occurring in $\mathcal{S}$ without duplicates in $O(|\Sigma|N) = O(|\Sigma|^2\ell)$ delay (time per frequent episode) and $O(m\ell + N) = O(|\Sigma|\ell)$ space, where $N = ||\mathcal{S}||$ and $m = ||Q||$ is the maximum size of frequent episodes.*

(proof) At each iteration of the algorithm FreqDmdRec, in the foreach-loop, the algorithm computes the occurrence list in $O(N) = O(|\Sigma|\ell)$ time by Corollary 9, and executes instructions except invocation of FreqDmdRec within the same cost. Since, each frequent diamond episode has at most $O(|\Sigma|)$ infrequent children, the running time per frequent diamond episode is $O(|\Sigma|N) = O(|\Sigma|^2\ell)$. At each iteration of the algorithm FreqDmdRec, it uses $O(\ell)$ space for occurrence list. Therefore, the algorithm FreqDmdRec takes at most $O(m)$ recursive calls to come back to the root from the leaf of depth $O(m)$. Since, the algorithm PolyFreqDmd uses $O(N)$ space to store an input sequence, we see that PolyFreqDmd runs in $O(m\ell + N)$ space. □

Corollary 11 *The frequent diamond episode mining problem is solvable in linear delay with respect to the total input size using polynomial space.*

Finally, we can reduce the space complexity of the algorithm PolyFreqDmd by using the *diffset* technique introduced by Zaki [69] for itemset mining, which can be realized by replacing Line 4, Line 5, and Line 6 of FreqDmdRec in Fig. 3.2 with the code in Fig. 3.7. Hence, we can reduce the space complexity in Theorem 10 to $O(m + \ell) = O(|\Sigma| + \ell)$.

3.3 Theoretical Analysis

In the last of the previous section, we have shown that the space complexity of the algorithm PolyFreqDmd is bounded by a polynomial in the total input size. In

this section, we show that the lower bound on the space complexity of the previous breadth-first algorithm FreqDmd [37] is exponential in the total input size in the worst case for unbounded alphabet Σ.

Let $\mathcal{S}$ be an input sequence, k the window width, and σ the minimum frequency threshold. For every $m \geq 0$, we denote by $\mathcal{F}_{\mathcal{S},k,\sigma}(m) \subseteq \mathcal{F}_{\mathcal{S},k,\sigma}$ the set of all frequent diamond episodes such that the size of every body is exactly m in $\mathcal{S}$.

Theorem 12 *Let $k \geq 3$ and $\sigma \geq 0$ be any integers that represent a window width and a minimum frequency threshold, respectively. For every $n \geq 1$, there exists a pair of an alphabet Σ_n and an input sequence $\mathcal{S}$ such that*

$$|\mathcal{F}_{\mathcal{S},k,\sigma}(m)| = 2^{\Omega(n)},$$

where $m = \lceil n/2 \rceil$, $|\Sigma_n| = n + 1$, $|\mathcal{S}_n| = k\sigma = \ell$, and $||\mathcal{S}_n|| = O(n\ell/k)$.

(proof) For any positive integer $n \geq 1$, we build Σ_n and $\mathcal{S}_n$ as follows. Let $\Sigma_n = \{\$, 1, \ldots, n\}$ be an alphabet consisting of $n+1$ events. Let k be any window width such that $k \geq 3$. We define a block $B = \langle T_1, \ldots, T_k \rangle$ of length k by $T_1 = T_k = \{\$\}$, $T_2 = \{1, \ldots, n\}$, and $T_i = \emptyset$ for every $i = 3, \ldots, k-1$. Then, we define an input sequence by the concatenation of θ copies of the block B, i.e., $\mathcal{S}_n = B^{(1)} \cdots B^{(\theta)}$. Clearly, the input sequence $\mathcal{S}_n$ has the length $\ell = k\theta$ and the total size $(n+2)\ell/k = O(n\ell/k)$. Let $\mathcal{C}_{m,n}$ be the set of all frequent diamond episodes of the form $Q = (\$ \mapsto E \mapsto \$)$ such that the size of E is exactly m and let $\mathcal{C}_n = \bigcup_{m \geq 0} \mathcal{C}_{m,n}$ be their union. Now, we fix the size of bodies to be $m = \lceil n/2 \rceil$. Let $f_{m,n} = |\mathcal{C}_{m,n}|$. In this case, we can easily see that there exist at least

$$f_{m,n} \;=\; \binom{n}{m} \geq \binom{2m}{m} = \left(\frac{2m}{m}\right)\left(\frac{2m-1}{m-1}\right)\cdots\left(\frac{m}{1}\right) \geq 2^m = 2^{n/2} = 2^{\Omega(n)}$$

subsets of Σ_n. Therefore, there exist at least $f_{m,n} = 2^{\Omega(n)}$ mutually distinct diamond episodes in the class $\mathcal{C}_{m,n}$. Since any $Q \in \mathcal{C}_n$ appears in the block B exactly once, we can easily see that Q also appears in the whole $\mathcal{S}_n$ exactly σ times. This implies that

if $Q \in \mathcal{C}_{m,n}$ then $Q \in \mathcal{F}_{\mathcal{S},k,\sigma}(m)$. Since $\mathcal{C}_{m,n} \subseteq \mathcal{F}_{\mathcal{S},k,\sigma}(m)$, we can conclude that for $m = n/2$

$$|\mathcal{F}_{\mathcal{S},k,\sigma}(m)| \geq |\mathcal{C}_{m,n}| = f_{m,n} = 2^{\Omega(n)}$$

holds. This completes the proof. □

Corollary 13 *The space complexity of* FreqDmd *algorithm [37] is at least exponential in the total input size in the worst case.*

(proof) Since the algorithm FreqDmd is a breadth-first algorithm, for any $m \geq 0$, it has to store all of the frequent diamond episodes in the m-th level $\mathcal{F}_{\mathcal{S},k,\sigma}(m)$ in the main memory. By Theorem 12, the size of $\mathcal{F}_{\mathcal{S},k,\sigma}(m)$ is exponential in $|\Sigma|$, and the total input size $||\mathcal{S}||$. This completes the proof. □

From Corollary 13 and Theorem 10, we see that the proposed algorithm PolyFreqDmd is more exponentially efficient than the previous algorithm FreqDmd for the space complexity.

3.4 Experimental Results

Data

In this section, we give experimental results for the combinations of the algorithms in Section 3.2 on both the artificial data set and the real-world data set.

The artificial data set consists of the randomly generated event sequence $\mathcal{S} = \langle S_1, \ldots, S_\ell \rangle$ ($\ell \geq 1$) over an alphabet $\Sigma = \{1, \ldots, s\}$ ($s \geq 1$) as follows. Let $0 < p \leq 1$ be any number called an event probability. Let $i = 1, \ldots, \ell$ be any index. For every event $e \in \Sigma$, we add e into S_i independently with probability p. By repeating this process, we build the i-th set S_i. On the other hand, the real-world data set is made from bacterial culture data provided from Osaka Prefectural General Medical Center from 2000 to 2005 by concatenating a detected bacterium for the sample of sputum

for every patients, where the length ℓ and the alphabet size s of the real-world event sequence are $\ell = 70,606$ and $s = 174$, respectively.

Method

We adopt the following implementations of the algorithms, where BF is the breadth-first search algorithm in [37] and DF and the others are the depth-first search algorithms presented in Section 3.2.

BF	:	FreqDmd [37] with breadth first itemset mining algorithm.
DF	:	PolyFreqDmd (Fig. 3.2) with FindSerialOcc (Fig. 3.4).
DF-ALT	:	DF with alternating output (ALT) (Fig. 3.2 with (*)).
DF-FFS	:	DF with fast update by FastFindSerialOcc (FFS) (Fig. 3.5).
DF-DIFF	:	DF with diffset technique (DIFF) (Fig. 3.7).
DF-MEM	:	DF with memoization technique (MEM) (Fig. 3.6).
DF-ALL	:	DF with all techniques (ALT, FFS, DIFF, and MEM).

All the experiments were run on a PC (AMD Mobile Athlon64 Processor 3000+, 1.81GHz, 2.00GB memory) with 32-bit x86 instruction set. If it is not explicitly described, we assume that the length of the sequence is $\ell = |\mathcal{S}| = 2000$, the alphabet size is $s = |\Sigma| = 30$, the probability of each event is $p = 0.1$, the window width is $k = 10$, the minimum frequency threshold is $\sigma = 0.4\ell$.

Experiments

Fig. 3.8 shows the running time and the number of solutions of the algorithms DF, DF-FFS, DF-MEM and DF-ALL for the input length ℓ, where $s = 30$, $k = 10$, and $\sigma = 0.05\ell$. Then, we see that DF-FFS is twice, DF-MEM is one hundred, and DF-ALL is two hundred as fast as DF. Here, since we can find no difference in the running time between DF-ALT, DF-DIFF and DF, we do not depict the results of DF-ALT and DF-DIFF in Fig. 3.8. Note that, the techniques of DF-ALT and DF-DIFF are useful

in technical improvements for time complexity and space complexity, respectively. Moreover, the running time of these algorithms DF, DF-FFS, DF-MEM and DF-ALL is almost linear in the input size and thus expected to scale well on large datasets.

Fig. 3.9 shows the running time for BF, DF, DF-FFS and DF-ALL, where $\ell = 1,000$, $s = 30$, $k = 10$, and $\sigma = 0.01\ell$. In this data set, we see that BF is faster than DF and DF-FFS. We see that DF-ALL is faster than BF.

Fig. 3.10 shows the size of memory usage for BF, DF, DF-FFS and DF-ALL, where $\ell = 1,000$, $s = 30$, $k = 10$, and $\sigma = 0.01\ell$. We can find no difference in the size of memory usage between DF, DF-FFS and DF-ALL on this data set. On the other hand, the size of memory usage for BF is larger than one for DF.

Fig. 3.11 shows the running time for the algorithm DF, DF-FFS and DF-MEM, where $\ell = 1,000$, $s = 30$, $k = 10$, and $\sigma = 0.05\ell$. Then, we see that the slopes of all algorithms are almost constant, and thus we can conclude that the delay is just determined by the input size as indicated by Theorem 10.

Fig. 3.12 shows the running time of DF-MEM, with varying the minimum frequency threshold $0.5\ell \leq \sigma \leq 5.0\ell$ with the input size $\ell = 2000$. We see that the number of outputs, and thus, the running time is increasing when σ is decreasing.

Fig. 3.13, 3.14 and 3.15 show the running time of the algorithms DF, DF-FFS and DF-MEM with varying the window width $13 \leq k \leq 25$, the size of alphabet $10 \leq s \leq 50$ and the event probability $0.02 \leq p \leq 0.12$, respectively. Then, we see that DF-MEM outperforms other algorithms in most cases. The performance of DF-MEM is stable in most datasets and the parameter settings. We also see that DF-FFS is from 20% to 60% faster than DF.

Fig. 3.16 and 3.17 show the running time and the size of memory usage of the algorithm DF-MEM with varying the window width $10 \leq k \leq 20$ and minimum frequency threshold $\sigma = 0.1\ell$ for the real-world event sequence. Then, we observe that the running time of the algorithm DF-MEM on the real-world data set shows a similar behavior as on artificial data set. Also, we observe that the size of memory usage of the algorithm DF-MEM is independent of the window width on this data set.

In Fig. 3.18, we show an example of the diamond episode Q with frequency 136 extracted from the real-world event sequence by the algorithm DF-MEM with the window width $k = 20$ and the minimum frequency threshold $\sigma = 0.1\ell$, where the typewriter fonts describe the names of bacteria. This episode says that the group of the bacteria of `yeast` and `Enterobacter-cloacae` occur in data, after the bacteria of `yeast` occurs and before the bacteria of `Staphylococcus-aureus` occurs within 20 days.

Overall, we conclude that the proposed algorithm FindDmdMain with the practical speed-up by memoization technique in Fig. 3.6 (DF-MEM) is quite efficient on the artificial data set used in these experiments. The fast linear-time update by FastFindSerialOcc (DF-FFS) achieves a speed-up of twice. Algorithm DF-MEM on the real-world data set shows a similar behavior as on artificial data set.

3.5 Chapter Summary

This chapter studied the problem of frequent diamond episode mining, and presented an efficient algorithm PolyFreqDmd that finds all frequent diamond episodes in an input sequence in polynomial delay and polynomial space in the input size. We have further studied several techniques for reducing both time complexity and space complexity of the algorithm.

algorithm PolyFreqDmd($\mathcal{S}, k, \Sigma, \sigma$)

input: input event sequence $\mathcal{S} \in (2^{\Sigma})^*$ of length ℓ, window width $k > 0$,

alphabet of events Σ, the minimum frequency $1 \leq \sigma \leq \ell + k$;

output: frequent diamond episodes; {

1 Σ_0 := the set of all events appearing no less than σ windows ($\Sigma_0 \subseteq \Sigma$);

2 **foreach** ($a \in \Sigma_0$) **do**

3 **output** a;

4 **foreach** ($b \in \Sigma_0$) **do**

5 $Q_0 := (a \mapsto \emptyset \mapsto b)$; //2-serial episode

6 W_0 := the occurrence list $W_{\mathcal{S},k}(Q_0)$ for Q_0;

7 FreqDmdRec($Q_0, W_0, \mathcal{S}, k, \Sigma_0, \sigma$);

8 **end for**

}

procedure FreqDmdRec($Q = (a \mapsto E \mapsto b), W, \mathcal{S}, k, \Sigma, \sigma$)

output: all frequent diamond episodes of the form $a \mapsto F \mapsto b$; {

1 **if** ($|W| \geq \sigma$) **then**

2 **output** Q; // (*) **output** Q if the depth is odd (*alternating output*);

3 **foreach** ($e \in \Sigma$ such that $e > \max(E)$)) **do**

4 $R := a \mapsto (E \cup \{e\}) \mapsto b$;

5 U := UpdateDmdOcc($Q, e, W, k, \mathcal{S}$); // Computing $U = W_{\mathcal{S},k}(R)$

6 FreqDmdRec($R, U, \mathcal{S}, k, \Sigma, \sigma$);

7 **end for**

8 // (*) **output** Q if the depth is even (*alternating output*);

9 **end if**

}

Figure 3.2: The main algorithm PolyFreqDmd and a recursive subprocedure FreqDmdRec for mining frequent diamond episodes in a sequence.

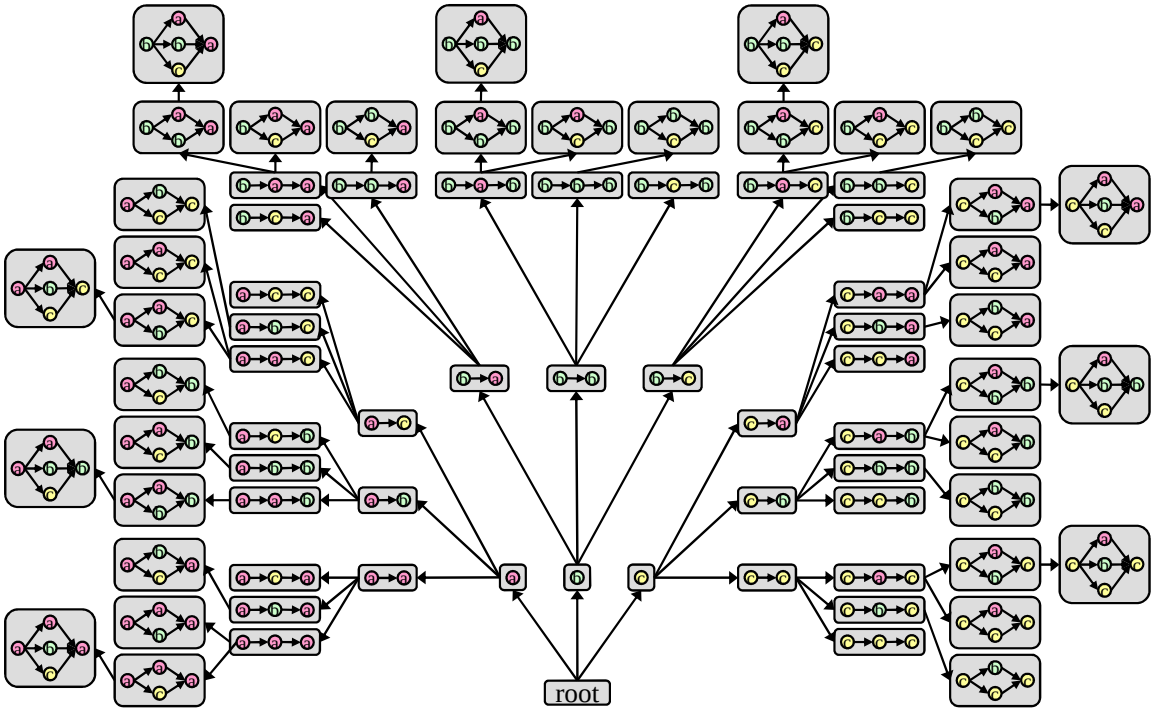


Figure 3.3: The parent-child relationships on the alphabet $\Sigma = \{a, b, c\}$.

---

algorithm UpdateDmdOcc($Q, e, W, k, \mathcal{S}$)

input: a parent diamond episode $Q = (a \mapsto E \mapsto b)$, a new event $e > \max(E)$,
the old occurrence list W for Q, $k \leq 1$, an input sequence $\mathcal{S}$;

output: the new occurrence list U for the child $R = (a \mapsto E \cup \{e\} \mapsto b)$; {

1 $V :=$ FindSerialOcc($P = (a \mapsto e \mapsto b), k, \mathcal{S}$);

2 **return** $U := W \cap V$;

}

procedure FindSerialOcc($P = (a \mapsto e \mapsto b), k, \mathcal{S}$) {

1 **return** the occurrence list $\mathbf{W}_{\mathcal{S},k}(P)$ for P in $\mathcal{S}$;

}

Figure 3.4: The algorithm UpdateDmdOcc for incremental update of the occurrence list.

procedure FastFindSerialOcc($P = (a \mapsto e \mapsto b), k, \mathcal{S} = \langle S_1, \ldots, S_\ell \rangle$)
input: serial episode $P = (a \mapsto e \mapsto b)$, window width $k > 0$, an input sequence $\mathcal{S}$;
output: the occurrence list **W** for P; {
1 **W** $:= \emptyset$; $(x, y, z) := (0, 0, 0)$;
2 **for** $(i := -k + 1, \ldots, \ell\ )$ **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** $y < end$ **and** (**not** $(y > x$ **and** $e \in S_y))$ **do** $y := y + 1$;
6 **while** $z < end$ **and** (**not** $(z > y$ **and** $b \in S_z))$ **do** $z := z + 1$;
7 **if** ($last < x < y < z < end$) **then** **W** $:=$ **W** $\cup \{i\}$;
8 // (x, y, z) is the lexicographically first occurrence of P in W_i;
9 **end for**
10 **return** **W**;
}

Figure 3.5: An improved algorithm FastFindSerialOcc for computing the occurrence list of a serial episode.

global variable: a hash table $TABLE : \mathcal{DE} \to 2^{\{-k+1,\ldots,n\}}$;
initialization: $TABLE := \emptyset$;
procedure LookupSerialOcc($P = (a \mapsto e \mapsto b), k, \mathcal{S}$) {
1 **if** $(TABLE[P] = UNDEF)$ **then**
2 $V :=$ FindSerialOcc$(P, k, \mathcal{S})$;
3 **if** $|V| \geq \sigma$ **then** $TABLE := TABLE \cup \{\langle P, V \rangle\}$;
4 **end if**;
5 **return** $TABLE[P]$;
}

Figure 3.6: Practical speed-up of FindSerialOcc using memoization technique.

1 $R := a \mapsto (E \cup \{e\}) \mapsto b$;

2 $\Delta :=$ FindSerialOcc$(P, k, \mathcal{S})$;

3 $W := W - \Delta$;

4 FreqDmdRec$(R, U, \mathcal{S}, k, \Sigma, \sigma)$;

5 $W := W \cup \Delta$;

6 $R = a \mapsto (E - \{e\}) \mapsto b$;

Figure 3.7: The diffset technique in PolyFreqDmd.

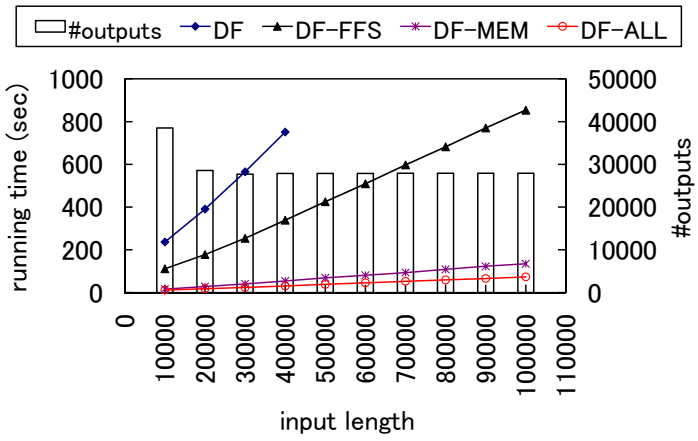


Figure 3.8: Running time for the input length ℓ, where $s = 30$, $k = 10$, and $\sigma = 0.05n$.

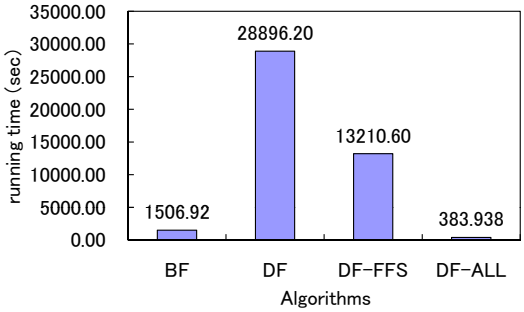


Figure 3.9: Running time for algorithms, where $n = 1,000$ $s = 30$, $k = 10$, and $\sigma = 0.01n$.

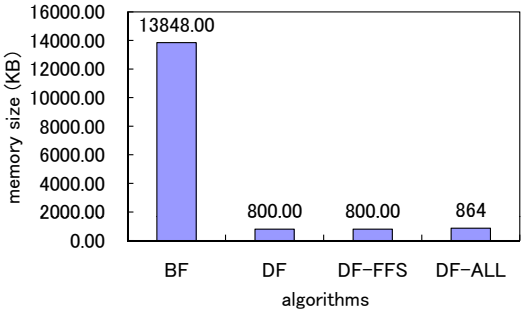


Figure 3.10: Memory size for algorithms, where $n = 1,000$ $s = 30$, $k = 10$, and $\sigma = 0.01n$.

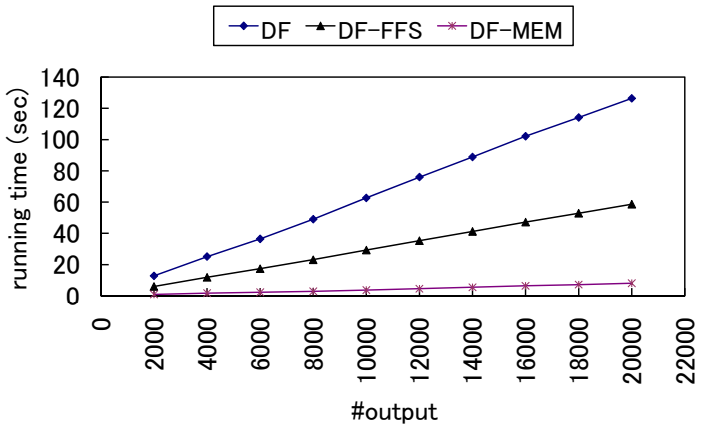


Figure 3.11: Running time for the number of outputs, where $\ell = 1,000$, $s = 30$, $k = 10$, and $\sigma = 0.05\ell$.

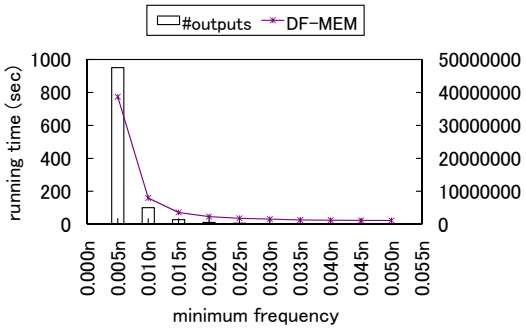


Figure 3.12: Running time for the minimum frequency threshold $0.5\ell \leq \sigma \leq 5\ell$ with span 0.5ℓ, where $\ell = 2,000$ and $k = 10$.

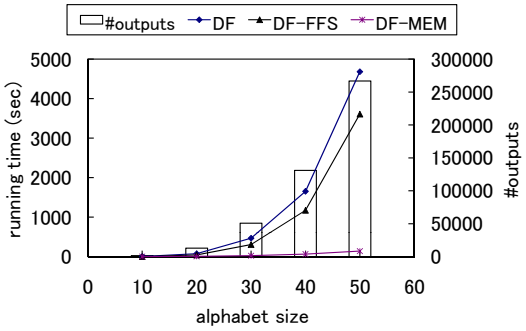


Figure 3.14: Running time for the alphabet size $10 \leq s \leq 50$ with span 10, where $\ell = 2,000$, $\sigma = 0.4\ell$.

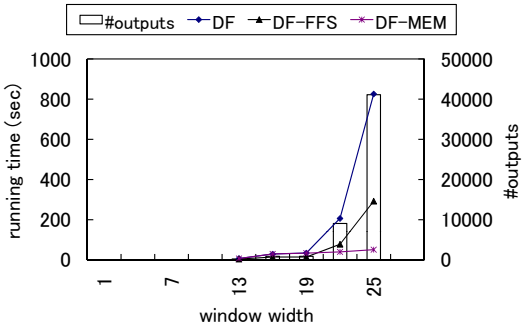


Figure 3.13: Running time for the window width $13 \leq k \leq 25$, where $\ell = 2,000$ and $\sigma = 0.4\ell$.

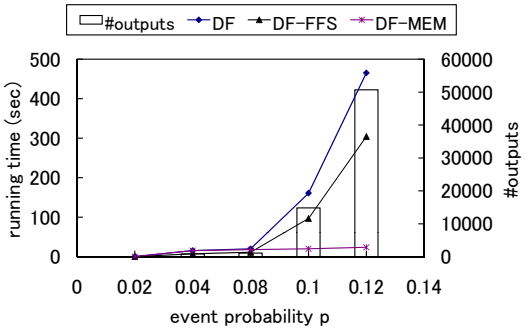


Figure 3.15: Running time for the occurrence probability of events $0.02 \leq p \leq 0.12$ with span 0.02, where $\ell = 2,000$ and $\sigma = 0.4\ell$.

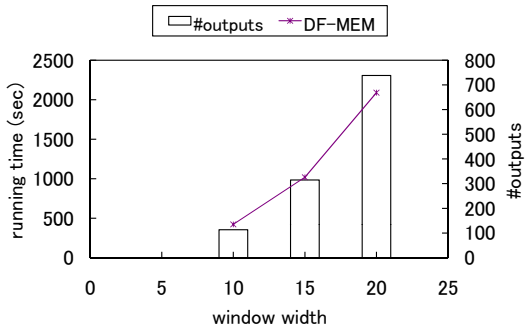


Figure 3.16: Running time for the window width $10 \leq k \leq 20$, where data set is real-world event sequence of $\ell = 70,606$, $s = 174$, and $\sigma = 0.1\ell$.

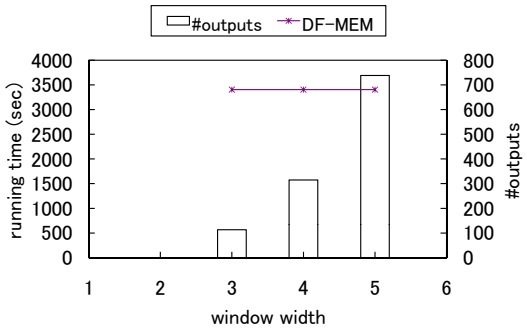


Figure 3.17: Memory size for the window width $10 \leq k \leq 20$, where data set is real-world event sequence of $\ell = 70,606$, $s = 174$, and $\sigma = 0.1\ell$.

$$
\begin{aligned}
Q \quad = \quad & \texttt{yeast} \\
& \mapsto \{\texttt{yeast}, \texttt{Enterobacter-cloacae}\} \\
& \mapsto \texttt{Staphylococcus-aureus}
\end{aligned}
$$

Figure 3.18: An example of the diamond episodes extracted from a bacterial culture data.

Chapter 4

Frequent Bipartite Episode Mining Problem

In this chapter, we present an efficient mining algorithm for the class of *bipartite episodes* from event sequences. For the alphabet Σ, input length n, and the total input size N, this algorithm runs in $O(|\Sigma|N)$ time per an output and in $O(|\Sigma|^2 n)$ space.

As a space-efficient episode mining algorithm, we newly 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 $O(|\Sigma|^2 n)$ 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 chapter is organized as follows. In Section 4.1, we introduce bipartite episodes and discuss several properties of bipartite episodes. In Section 4.2, we present the algorithm Bipar and show its correctness and complexity. In Section 4.3, we give some experimental results from randomly generated event sequences to evaluate the

practical performance of the algorithms. In Section 4.4, we conclude this chapter and discuss the future works.

This chapter is based on the paper [29].

4.1 Bipartite Episodes

In this section, we introduce the class of bipartite episodes and other notions and discuss their properties.

Definition

Definition 8 For $m \geq 1$, *m-serial episode* (or *serial episode*) over Σ is a sequence $P = (a_1 \mapsto \cdots \mapsto a_m)$ of events $a_1, \ldots, a_m \in \Sigma$. This P represents an episode $X = (V, E, g)$, where $V = \{v_1 \ldots v_m\}$, $E = \{(v_i, v_{i+1}) \mid 1 \leq i < m\}$, and $g(i) = a_i$ for every $i = 1, \ldots, m$.

Definition 9 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 10 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

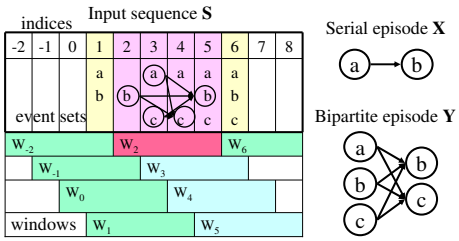


Figure 4.1: (Left) An input sequence $\mathcal{S} = (S_1, \ldots, S_6)$ of length $n = 6$ over $\Sigma = \{a, b, c\}$ and their k-windows. (Right) Serial episode $X = a \mapsto b$ and a bipartite episode $Y = (\{a, b, c\} \mapsto \{b, c\})$. In the sequence $\mathcal{S}$, we indicate an occurrence (embedding) of Y in the second window W_2 in circles and arrows. See Example 3 and 4 for details.

$(A \mapsto B)$, 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 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 4.1. 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., $\mathbf{W}_{\mathcal{S},5} = \{\, W_i \mid -2 \leq i \leq 6 \,\}$.

Example 4 In Figure 4.1, 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) $\mathcal{PE} \subseteq \mathcal{BE}$.

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 if and only if for every serial episode $S \in Ser(X)$, $S \sqsubseteq W$ holds.

Katoh and Hirata [33] gave a necessary and sufficient condition for serially constructibility, called the parallel-freeness.

Definition 11 (Katoh and Hirata [33]) 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 14 (Katoh and Hirata [33]) *Let Σ be any alphabet. X is parallel-free if and only if X is serially constructible.*

Theorem 15 *Let X be partial bi-episode. If X is bipartite then X is parallel-free.*

(proof) Since $X = (V_1, V_2, A, g)$ is a bipartite episode, X is a complete and partwise-linear. Since X is a partwise-linear, for every verteces $u, v \in V_1$ such that $u \neq v$, $g(u) \neq g(v)$ holds. Similarly, for every it holds that $g(u) \neq g(v)$. Since X is complete, for every $u \in V_1$ and $v \in V_2$, it holds that $(u, v) \in A$. Therefore, X is parallel-free. □

Corollary 16 *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 17 (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 16, $S \sqsubseteq W$ 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 16, $X_1 \sqsubseteq W$. 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, $freq(X_1) \geq freq(X_2)$. □

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 16 that does not use Theorem 14. 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 18 (characterization of the occurrences) *Let $X = (U, V, A, g)$ be any bipartite episode and W any window in $\boldsymbol{W}_{\mathcal{S},k}$. Then, $X \sqsubseteq W$ if and only if $(\max_{u \in U} st(g(u), W)) < (\min_{v \in V} et(g(v), W))$ holds.*

Lemma 19 *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 20 (a detailed version of serial construction) *Let X be a bipartite episode and $W = \langle S_1, \ldots, S_k \rangle$ a window in $\boldsymbol{W}_{\mathcal{S},k}$. Let, $A, B \subseteq \Sigma$ be non-empty sets. Then,*

(1) If $X = (A \mapsto B)$ then, $X \sqsubseteq W$ if and only if $\forall (a, b) \in A \times B$, $(a \mapsto b) \sqsubseteq W$.

(2) if $X = (A \mapsto \emptyset)$ or $X = (\emptyset \mapsto B)$, $X \sqsubseteq W$ if and only if $\forall a \in A \cup B$, $a \sqsubseteq W$.

(proof) We first consider the case that both of A and B are non-empty. Only if-direction immediately follows from Lemma 18. We show if-direction, that is, for

non empty sets of event A and B, if $(a \mapsto b) \sqsubseteq W$ for every $(a, b) \in A \times B$ then there exist a bipartite episode $A \mapsto B$. For every 2-serial episode $(a \mapsto b)$ such that $a \in A$ and $b \in b$, then there exists some mapping $h_{a,b} : (\{1, 2\} \to \{1, \ldots k\})$ satisfying $a \in S_{h(1)}$, $b \in S_{h(2)}$, and $st(a, W) \leq h(1) < h(2) \leq et(b, W)$. Therefore, from the Lemma 18, the claim holds. In the case that one of A and B is empty, the result is easily proved. □

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 $\mathbf{W}_{\mathcal{S},k}(X_1 \cup X_2) = \mathbf{W}_{\mathcal{S},k}(X_1) \bigcap \mathbf{W}_{\mathcal{S},k}(X_2)$ 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 21 (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* $\boldsymbol{W}_{\mathcal{S},k}(X_1 \cup X_2) = \boldsymbol{W}_{\mathcal{S},k}(X_1) \bigcap \boldsymbol{W}_{\mathcal{S},k}(X_2)$.

4.2 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 $\mathcal{S} = (S_1, \ldots, S_n) \in (2^{\Sigma})^*$ 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.

4.2.1 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.

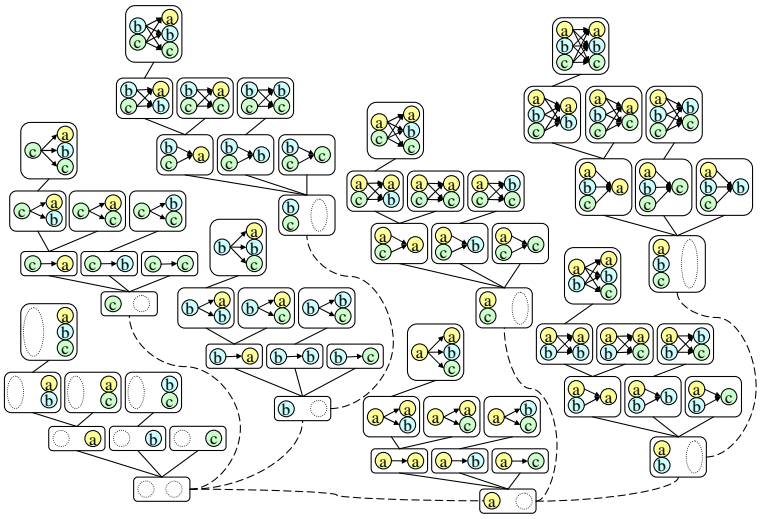


Figure 4.2: The parent-child relationships on the alphabet $\Sigma = \{a, b, c\}$, where each white empty circle indicates the empty set.

Definition 12 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

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

We define the set of all children of X by $Children(X) = \{Y \mid parent(Y) = X\}$. 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 $E = \{(X, Y) \mid X \text{ is the parent of } Y, Y \neq \perp\}$. As shown in Fig. 4.2, we can show that the family tree $\mathcal{T}(\mathcal{BE})$ forms the spanning tree for all bi-episodes of $\mathcal{BE}$.

In Fig. 4.3, 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 BIPAROCC 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 BiparOcc 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.

Lemma 22 *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. 4.3 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.*

4.2.2 Incremental computation of occurrences

The algorithm BiparOccInc in Fig.4.4 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 Theorem 20 on the downward closure property for $\mathcal{BE}$.

Lemma 23 (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 $\boldsymbol{W}(A \mapsto (B \cup \{e\})) = \boldsymbol{W}(X) \cap \boldsymbol{W}(\emptyset \mapsto \{e\})$.

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

The algorithm BiparOccInc uses the subprocedure SerialOcc for computing the occurrence list for a 2-serial episode. This algorithm is a modification of Fast-SerialOcc for 3-serial episodes in [30] 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.

Lemma 24 *The algorithm* BiparOccInc *in Fig.4.4 computes the new occurrence list $W = \boldsymbol{W}_{\mathcal{S},k}(Y)$ for the child episode $Y = (A \mapsto B \cup \{b\})$ in $O(N|A|) = O(|\Sigma|^2 n)$ time from a bi-episode $X = (A \mapsto B)$, $\boldsymbol{W}_{\mathcal{S},k}(X)$, any event $b \in \Sigma$, and k, where $n = |\mathcal{S}|$ and $N = ||\mathcal{S}||$.*

4.2.3 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. 4.5 shows the algorithm LookupSerialOcc that is a modification version of SerialOcc using dynamic programming. This algorithm uses a hash table $TABLE$ in Fig. 4.5 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 initializating 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 $|\Sigma||F|$ to at most $|\Sigma|^2$, where $\mathcal{F} \subseteq \mathcal{BE}$ is the set of solutions.

Lemma 25 *After initializating 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.*

4.2.4 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 [68, 69].

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 $parent : \mathcal{BE}\backslash\{\perp\} \to \mathcal{BE}$.

Definition 13 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.

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.

Lemma 26 *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 $\boldsymbol{W}(Y)$ is also obtained by the corresponding rule.*

(i) If $X = (A \mapsto \emptyset)$, then for any $e \in \Sigma$, $Y = (\emptyset \mapsto A)$ and $\boldsymbol{W}(Y) = \boldsymbol{W}(X)$.

(ii) If $X = (A \mapsto \emptyset)$, then for any $e \in \Sigma$, $Y = (A \cup \{e\} \mapsto \emptyset)$ and $\boldsymbol{W}(Y) = \boldsymbol{W}(X) \cap \boldsymbol{W}(e)$.

(iii) If $X = (A \mapsto \{b\})$, then for any $e \in \Sigma$, $Y = (A \cup \{e\} \mapsto \{b\})$ and $\boldsymbol{W}(Y) = \boldsymbol{W}(X) \cap \boldsymbol{W}((e \mapsto b))$.

(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 $\boldsymbol{W}(Y) = \boldsymbol{W}(X) \cap \boldsymbol{W}(Z)$.

(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 28 on downward closure property for $\mathcal{BE}$. □

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).

We can apply further improvements to our algorithm BiparFast as shown in [30]. 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 [60]. 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 [69] 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 chapter on the delay and the space complexities of the modified algorithm BiparFast.

Theorem 28 *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 $O(|\Sigma|^2 n)$ space, where $N = ||\mathcal{S}||$ is the total size of input.*

4.3 Experimental Results

In this section, we give the experimental results for the following combinations of the algorithms given in Section 4.2, by applying to the randomly generated event sequences and the real event sequence.

Data: As randomly generated data, we adopt an event sequence $\mathcal{S} = (S_1, \ldots, S_n)$ over an alphabet $\Sigma = \{1, \ldots, s\}$ from four parameters (n, s, p, r), by generating each

event set S_i $(i = 1, \ldots, n)$ under the probability $P(e \in S_i) = p(e/s)^r$ for each $e \in \Sigma$. On the other hand, as the real event sequence, we adopt bacterial culture data provided from Osaka Prefectural General Medical Center from 2000 to 2005. In particular, we adopt an event sequence obtained by regarding a detected bacterium as an event type, fixing the sample of sputum and connecting data of every patient with same span.

Method: We implemented the following three depth-first search (DFS) algorithms given in Section 4.2:

Basic	:	the basic DFS algorithm BiparBasic with OiparOcc in Fig. 4.3 (sec. 4.2.1).
Fast	:	the modified DFS algorithm BiparFast with SerealOcc (sec. 4.2.4).
FastDP	:	the modified DFS algorithm BiparFast with LookUpSerealOcc based on dynamic programming (sec. 4.2.3).

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. 4.6 and Fig. 4.7 show the running time and the size of virtual memory usage of the algorithms Fast and FastDP for the randomly 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, both time and space complexity of these algorithms seem to be linear in the input size and thus expected to scales well on large datasets. Furthermore, FastDP is five hundred times as faster as Fast. On the other hand, FastDP tends to occupy more memory than Fast.

Experiment B: Fig. 4.8 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 $\sigma = 0.001n$.

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 4.1: 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.

Then, the slopes are almost constant and thus the delays are just determined by the input size as indicated by Theorem 28.

Experiment C: Table 4.1 gives the seven experiments for the algorithms Basic, Fast, and FastDP under the various parameter settings par1 – par7. The input data from exp1 to exp5 are randomly generated event sequences with parameters (n, s, p, r), and ones of exp6 and exp7 are the bacterial culture data.

Fig. 4.9 and Fig. 4.10 show the running time and the virtual memory size of the algorithms. Here, for exp2, exp3, and exp7, we give no results for Basic, because the running time is over $20,000$ (sec). Then, for exp1, exp4, exp5, and exp6, Fast and FastDP are faster than Basic. Especially, for exp5, FastDP was 7300 times faster than Basic. On the other hand, Fast and FastDP occupy more memory space than Basic. From exp1 to exp7, the algorithm FastDP occupy more memory space than Fast. Then, the algorithm FastDP is the fastest algorithm except exp3. For exp3 with large alphabet size $|\Sigma| = 1000$, FastDP occupies sixteen (16) times large memory

space than Fast, and Fast is faster than FastDP.

Fig. 4.11 shows an example of the bipartite episode X with frequency 21 extracted from the bacterial culture data for exp7, where an event in type writer fonts denotes the names of bacteria. This episode represents that the bacteria of `Serratia-marcescens` and `Staphylococcus-aureus` are followed by another bacteria of `yeast` and `Stenotrophomonas-maltophilia` within fifteen days.

4.4 Chapter Summary

This chapter 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.

algorithm BIPAR($\mathcal{S}, k, \Sigma, \sigma$)

input: input event sequence $\mathcal{S} = \langle S_1, \ldots S_n \rangle \in (2^{\Sigma})^*$ 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  ⊥ := (∅↦∅); // The root bipartite episode ⊥;
2  FreqBiparRec(⊥, S, k, Σ, σ);
```

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 ( |W_{S,k}(X)| ≥ σ ) then
2      output X;
3      // Execute in the special case that the right hand side of X is empty.
4      if ( B = ∅ ) then
5          foreach e ∈ Σ such that e > max(A) do
6              // Expand the left hand side.
7              FreqBiparRec(((A ∪ {e}) ↦ B), S, k, Σ, σ);
8      end if
9      // Execute always.
10     foreach ( e ∈ Σ (e > max(B) ) ) do
11         // Expand the right hand side.
12         FreqBiparRec((A ↦ (B ∪ {e})), S, k, Σ, σ);
13 end if
```

Figure 4.3: The main algorithm BIPAR and a recursive subprocedure FREQBIPARREC for mining frequent bipartite episodes in a sequence.

procedure BIPAROCCINC$(X, X_0, W_0, k, \mathcal{S})$

input: bipartite episodes $X = (A \mapsto (B_0 \cup \{e\})$ and $X_0 = (A \mapsto B_0) = \mathit{parent}(X)$,
the occurrence list W_0 for X_0, window width $k > 0$
, an input sequence $\mathcal{S} = \langle S_1, \ldots S_n \rangle$;

output: the occurrence list W for X;

method:

```
1  W := W_0;
2  if ( A = ∅ ) then W := W∩SERIALOCC((∅↦{e}), k, S);
3  else
4      foreach (a ∈ A) W := W∩SERIALOCC((a↦e), k, S);
5  return W;
```

Figure 4.4: An improved algorithm BIPAROCCINC for computing the occurrence list of a bipartite episode.

global variable: a hash table $TABLE : \Sigma^2 \to 2^{\{-k+1,\ldots,n\}}$;
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 $\mathcal{S} = \langle S_1, \ldots S_n \rangle$;

output: the occurrence list W for X;

method:

```
1  if (TABLE[(a, b)] = UNDEF) then
2      W :=SERIALOCC((a↦b), k, S);
3      TABLE := TABLE ∪ {⟨ (a, b), W⟩ };
4  end if
5  return TABLE[(a, b)];
```

Figure 4.5: Practical speed-up for computing occurrence lists of serial episodes using dynamic programming.

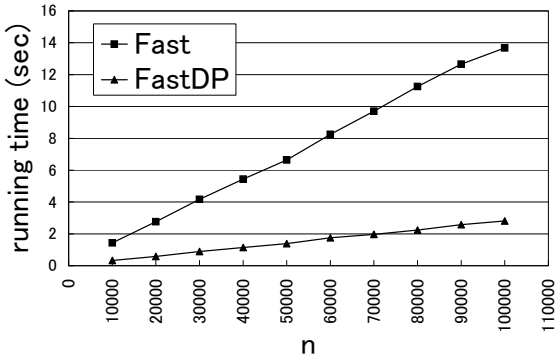


Figure 4.6: Running time for the input length n, where $s = 10$, $p = 0.1$, $r = 0.0$, $k = 10$, and $\sigma = 0.1n$.

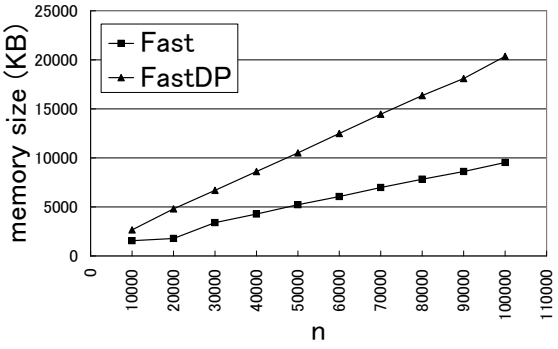


Figure 4.7: Memory size for the input length n, where $s = 10$, $p = 0.1$, $r = 0.0$, $k = 10$, and $\sigma = 0.1n$.

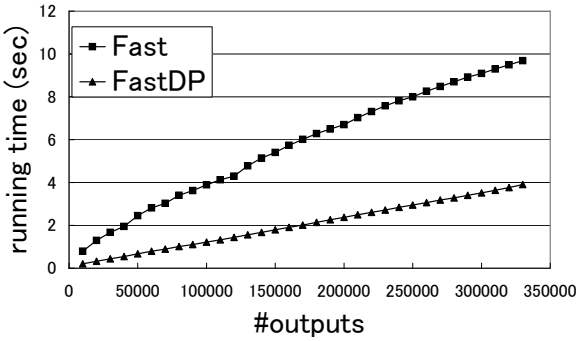


Figure 4.8: Running time for the number of outputs, where $n = 10,000$, $s = 10$, $p = 0.1$, $r = 0.0$, $k = 10$, and $\sigma = 0.001n$.

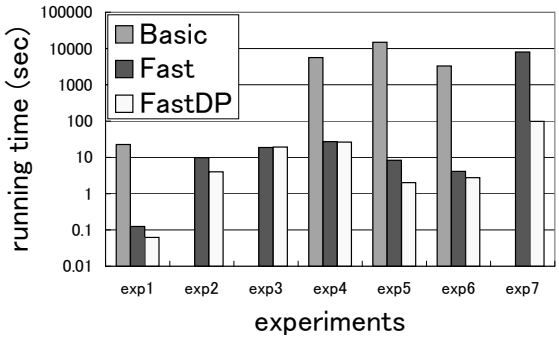


Figure 4.9: Running time for exp1 to exp7.

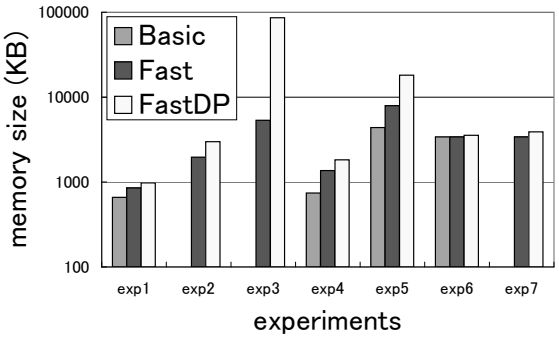


Figure 4.10: Memory size for exp1 to exp7.

$$X = \{\texttt{Serratia-marcescens}, \texttt{Staphylococcus-aureus}\}$$
$$\mapsto \{\texttt{yeast}, \texttt{Stenotrophomonas-maltophilia}\}$$

Figure 4.11: An example of bipartite episode extracted from a bacterial culture data.

Chapter 5

Frequent Partite Episode Mining Problem

In this chapter, we present an efficient mining algorithm for the class of *partite episodes* from event sequences. For the alphabet Σ, maximum length k of output episodes, and the total input size N, this algorithm runs in $O(|\Sigma|N^2k)$ time per an output and in $O(|\Sigma|k + N)$ space.

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.

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.

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 $O(N^2k)$ time by using the event lists. The modified algorithm Kpar2 with CountByList runs in $O(|\Sigma|N^2k)$ 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.

This chapter is organized as follows. In Section 5.1, we introduce k-partite episodes and other notions and discuss their properties. In Section 5.2, we present the algo-

rithms KPAR and KAPR2 to extract all of the frequent k-partite episodes. In Section 5.3, we give the experimental results for the algorithms given in Section 5.2, by applying to the randomly generated event sequences. In Section 5.4, we conclude this chapter and discuss the future works.

This chapter is based on the paper [31].

5.1 k-Partite Episodes

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.

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

In this chapter, 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 $1 \leq i \leq n$ [40]. Then, we call an episode $X = (\emptyset, \emptyset, g)$ with an empty vertex set an *empty episode* and denote X by $\emptyset$.

Definition 14 For $k \geq 1$, a *k-serial episode* (or a *serial episode*) over Σ is an episode $X = (V, E, g)$ satisfying that $V = \{v_1, \ldots, v_k\}$, $E = \{(v_i, v_{i+1}) \mid 1 \leq i < k\}$ and $g(v_i) = a_i$ for every $1 \leq i \leq k$. We denote such a k-serial episode by a sequence $(a_1 \mapsto \cdots \mapsto a_k)$ of events $a_1, \ldots, a_k \in \Sigma$.

Definition 15 For $k \geq 1$, 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 $V_i \cap V_j = \emptyset$ 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.

(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 k-tuple $X = \langle A_1, \ldots, A_k \rangle$, where $A_i \subseteq \Sigma$ is a set of events for every $1 \leq i \leq k$.

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. 5.1. In what follows, we denote the classes of k-serial, parallel, sectorial [36], diamond [30, 37] 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) $\mathcal{SE}_k \subseteq \mathcal{PTE}_k$ ($k \geq 0$). In particular, for the 3-partite episode $X = \langle A_1, A_2, A_3 \rangle$, if both $|A_1| = 1$ and $|A_3| = 1$ then X is a diamond episode. Also, for the 2-partite episode $X = \langle A_1, A_2 \rangle$, if $|A_2| = 1$ then X is a sectorial episode.

5.2 Algorithm

5.2.1 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.

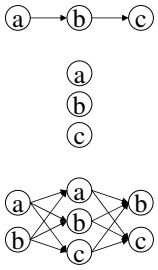


Figure 5.1: The examples of a 3-serial episode (top), a parallel episode (center), and a 3-partite episode (bottom) on the alphabet $\Sigma = \{a, b, c\}$.

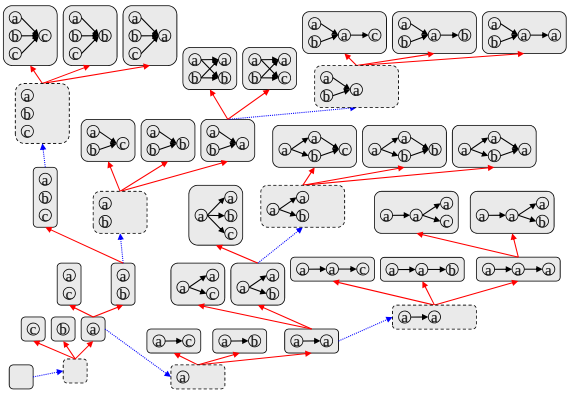


Figure 5.2: The parent-child relationships on the alphabet $\Sigma = \{a, b, c\}$, where episodes in dashed boxes are pre-partite episodes.

Definition 16 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 29 *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. □

In Fig. 5.2, 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. 5.3, 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 $\mathcal{PTE}_k$.

5.2.2 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 30 *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 $p_i = \max_{e \in A_i} \min\{\, j \mid p_{i-1} < j \leq n, e \in S_j \,\}$.*
3. *The list P can be computed in $O(|\Sigma|n)$ time.*

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 ⟨⟩;
2  KparRec(⟨∅⟩, 1, S, w, Σ, σ);
```

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, S, w);        // f = |W_{S,w}(X)|;
3  if ( f < σ ) then return;
4  if (A_i ≠ ∅) then
5      output X;
6      KparRec(⟨A_1, ..., A_i, ∅⟩, i + 1, S, w, Σ, σ);
7  end if
8  foreach ( e ∈ Σ such that e > max(A_i)) do
9      KparRec(⟨A_1, ..., A_i ∪ {e}⟩, i, S, w, Σ, σ);
```

Figure 5.3: 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, ..., n ) do
3      Test if X ⊑ W_i^{S,w} = ⟨S_i, ..., S_{i+w−1}⟩ by the procedure in Lemma 30.
4      if the answer is "Yes" then f := f + 1;
5  end for
6  return f;
```

Figure 5.4: An algorithm CountByScan for computing the frequency of a k-partite episode.

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. 5.4 computes the frequency count for a k-partite episode X based on the above lemma.

Lemma 31 Let $\mathcal{S}$ be an input event sequence of length n and w a window width. Then, the algorithm CountByScan in Fig. 5.4 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 [60] to reduce the delay by factor of the depth of the search tree. Then:

Theorem 32 *Let $\mathcal{S}$ be an input event sequence of length n. Then, the algorithm* Kpar *in Fig. 5.3 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.*

algorithm KPAR2($\mathcal{S}, k, w, \Sigma, \sigma$)

input: an input event sequence $\mathcal{S} = \langle S_1, \ldots S_n \rangle \in (2^{\Sigma})^*$ 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;

Figure 5.5: The modified algorithm KPAR2 for mining frequent k-partite episodes in a sequence.

(proof) At each iteration of the algorithm KPARREC, the algorithm computes the frequency count f in $O(|\Sigma|wn)$ by Lemma 31 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. □

5.2.3 Modified algorithm with frequency counting on event lists

In Fig. 5.5, 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 $L[e] = (p_1, \ldots, p_m)$ of positions

method:

1 **foreach** $e \in \Sigma$ **do** $last_e := -1$; $free_e := 0$; $Head_e := 0$; **end foreach**

2 **for** $(i := i_1, \ldots, i_h)$ **do**

// $(i_1, \ldots, i_h) = \mathcal{A}$ 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** $\mathcal{L} = (Head_e, POS_e, NEXT_e)_{e \in \Sigma}$;

Figure 5.6: The algorithm for computing the event list table $\mathcal{L}$ in $\mathcal{S}$.

$1 \leq p \leq n$ 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 $\mathcal{A} = (i_1, \ldots, i_h)$ $(h \geq 0)$ of the appropriate positions in $\mathcal{S}$ is given.

Lemma 33 *The event list table $\mathcal{L}$ in $\mathcal{S}$ can be computed in $O(N)$ time.*

(proof) We encode a event list $L[e] = (p_1, \ldots, p_h)$ $(h \geq 0)$ 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 algorithm in Fig. 5.6 computes $\mathcal{L}$ in $\mathcal{S}$. 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 $p = \max_{e \in A} \min\{\, j \mid p \leq j \leq n, e \in S_j \,\}$.

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**

Figure 5.7: 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 ≥ ℓ then return (ℓ_max, (LM[·], π[·], i_last, ℓ_max));
3  while i < n do
4      Let i be the smallest appropriate position such that i > i_last and S_i ≠ ∅;
5      foreach e ∈ S_i ∩ A do
6          LM[e] := POS_e[π[e]]; ℓ_max := max(ℓ_max, LM[e]); π[e] = NEXT_e[π[e]];
7      end foreach
8      i_last := i;
9      if i ≥ ℓ then return (ℓ_max, (LM[·], π[·], i_last, ℓ_max));
10 end while
11 return (n + 1, (LM[·], π[·], i_last, n + 1)); //failed!
```

Figure 5.8: An incremental version of LeftMostTail that computes the leftmost tail position of a set A of events w.r.t. a given position ℓ.

Lemma 34 *Suppose that $S_0 = \Sigma$. Then, the algorithm* LeftMostTail *in Fig. 5.7 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. 5.9, we describe the algorithm CountByList for computing frequency of episodes with an incremental version IncLeftMostTail of LeftMostTail in Lemma 34. 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 35 *Let $\mathcal{S}$ be an input sequence of length n and w a window width. Then, the algorithm* CountByList *in Fig. 5.9 computes the frequency $freq_{\mathcal{S},w}(X)$ of a*

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;
```

Figure 5.9: An algorithm CountByList for computing the number of windows in which a k-partite episode occurs.

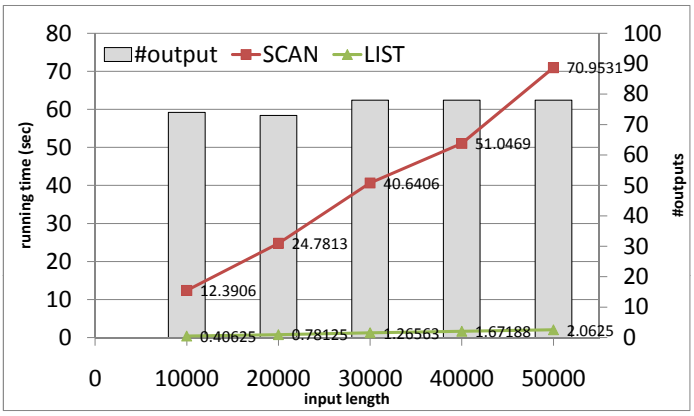


Figure 5.10: Running time for the input length n, where $s = 4$, $p = 0.1$, $k = 4$, $w = 4$, and $\sigma = 0.01n$.

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 [60], we show the following theorem on the complexity of the modified algorithm Kpar2.

Proposition 36 *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 $\mathcal{S}$.*

5.3 Experimental Results

In this section, we give the experimental results for the following combinations of the algorithms given in Section 5.2, by applying to the randomly generated event

sequences.

Data: As randomly generated data, we adopt an event sequence $\mathcal{S} = (S_1, \ldots, S_n)$ over an alphabet $\Sigma = \{1, \ldots, s\}$ from four parameters (n, s, p), by generating each event set S_i $(i = 1, \ldots, n)$ under the probability $P(e \in S_i) = p$ for each $e \in \Sigma$.

Method: We implemented the following two algorithms given in Section 5.2:

SCAN	:	the algorithm KPAR with COUNTBYSCAN in Fig. 5.3
LIST	:	the algorithm KPAR2 with COUNTBYLIST in Fig. 5.5

All experiments were run in a PC (AMD Mobile Athlon64 Processor 3000+, 1.81GHz, 2.00GB memory, Window XP, Visual C++) with window width $w \geq 1$, maximum length of output partite episode $k \geq 0$ and minimum frequency threshold $\sigma \geq 1$.

Experiments Fig. 5.10 shows the running time of the algorithms SCAN and LIST for the randomly generated event sequences from the parameter $(10000 \leq n \leq 50000, s = 4, p = 0.1)$, where $k = 4$, $w = 4$ and $\sigma = 0.01n$. Then, time complexity of these algorithms seem to be linear in the input size and thus expected to scales well on large datasets. Furthermore, SCAN is 35 times as faster as LIST.

5.4 Chapter Summary

This chapter 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. Then, we have implemented our two algorithms KPAR and KPAR2 and given empirical results to compare the time and space efficiencies of the algorithms.

Chapter 6

Simple Characterization of Serially Constructible Episodes

In this chapter, we study a property, called the *serially constructibility* for the class of episodes, which is crucial to design of efficient frequent episode mining algorithms. For a syntactic property, called *parallel-free* condition, we show that for any episode D, D is serially constructible if and only if D is parallel-free.

It is one of the important tasks for data mining to discover frequent patterns from time-related data. For such a task, Mannila *et al.* [40] have introduced the *episode mining* to discover frequent *episodes* in an *event sequence*. Here, the episode is formulated as an *acyclic labeled digraphs* of which label is an event type and of which edges specify the temporal precedent-subsequent relation in an event sequence, which proposes a richer representation of temporal relationship than a *subsequence* in *sequential pattern mining* (*cf.*, [47]).

Then, Mannila *et al.* [40] have designed an algorithm to construct episodes from a *parallel episode* as a set of events and a *serial episode* as a sequence of events. Note that their algorithm is general but inefficient. In order to avoid such inefficiency, Katoh *et al.* have introduced the specific forms of episodes, that is, *sectorial episodes* [36], *diamond episodes* in Chapter 3 and *elliptic episodes* [32], and designed

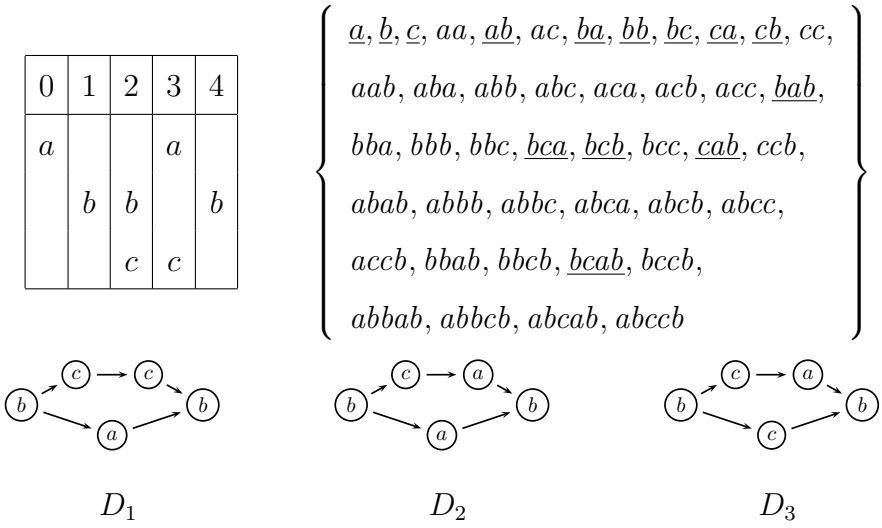


Figure 6.1: An event sequence W (upper left), all of the serial episodes occurring in W (upper right), and episodes D_1, D_2 and D_3 (lower).

efficient algorithms to extract them. Such efficiency follows from the construction of episodes from just information for occurrences of serial episodes in an event sequence, which is obtained by scanning an event sequence just once. On the other hand, there has remained an open problem of *what form of episodes is constructible from just information for occurrences of serial episodes*.

Consider the event sequence W consisting of a pair (e, t) of an event type e and the occurrence time t of e described as Figure 6.1 (upper left), where all of the serial episodes occurring in W are described as Figure 6.1 (upper right). Also consider the episodes D_1, D_2 and D_3 described in Figure 6.1 (lower) as acyclic labeled digraphs. Note first that all of D_i are embedded into W.

Since D_2 is *not* embedded into W with distinguishing an event type a in D_2, D_2 is *not* constructible from just information for occurrences of serial episodes in D_2. On the other hand, while D_3 is embedded into W with distinguishing every event type in D_3, D_3 is *not* constructible from just information for occurrences of serial episodes in D_3, because all of the serial episodes occurring in D_3 coincide with ones in a serial episode *bcab* in W underlined in Figure 6.1 (upper right).

In order to solve the problem of *what form of episodes is constructible from just information for occurrences of serial episodes* with capturing the above situations, we formulate both an episode and an event sequence as an acyclic *transitive* labeled digraph (*ATL-digraph*, for short) of which label is an event type. We say that an ATL-digraph D is *parallel-free* if D always has an arc between vertices with the same label. Also we say that an ATL-digraph D is *serially constructible* if D is embedded into every parallel-free ATL-digraph containing all of the serial episodes occurring in D. Hence, we show that *an episode* (as *an ATL-digraph*) *is parallel-free if and only if it is serially constructible*.

This chapter is based on the paper [33].

6.1 Episodes as Digraphs

As similar as Mannila's episode mining [40], we assume that an event has an associated time of occurrence as a natural number. Formally, let $\mathcal{E}$ be a set of *event types*. Then, a pair (e, t) is called an *event*, where $e \in \mathcal{E}$ and t is a natural number which is the (*occurrence*) *time* of the event. For a set $E \subseteq \mathcal{E}$ of event types, we denote $\{(e, t) \mid e \in \mathcal{E}\}$ by (E, t).

An *event sequence* $\mathcal{S}$ on $\mathcal{E}$ is a triple $\langle S, T_s, T_e \rangle$, where

$$S = \langle (E_1, t_1), \ldots, (E_n, t_n) \rangle$$

is an ordered sequence of events satisfying the following conditions.

1. $E_i \subseteq \mathcal{E}$ $(1 \leq i \leq n)$,
2. $T_s \leq t_1 < \cdots < t_n \leq T_e$.

Here, T_s and T_e are called the *starting* time and the *ending* time of $\mathcal{S}$. In particular, a *window* in an event sequence $\mathcal{S} = (S, T_s, T_e)$ is an event sequence $W = (w, t_s, t_e)$ such that $t_s < T_e$, $T_s < t_e$ and w consists of all of the events (e, t) in S where $t_s \leq t < t_e$.

Mannila *et al.* [40] have formulated an episode as an *acyclic labeled digraph*. On the other hand, in this chapter, we formulate an episode as an acyclic *transitive* labeled digraph. Note that we adopt the transitivity to represent all of the serial episodes contained in an episode explicitly. Then, we prepare some notions for digraphs necessary for discussion bellow according to [32].

A *digraph* (or a *directed graph*) $D = (V, A)$ consists of a finite, nonempty set V of *vertices* and a (possibly empty) set A of ordered pairs of distinct vertices. We sometimes denote V by $V(D)$ and A by $A(D)$. We denote $|V|$ by $|D|$. A digraph $(\emptyset, \emptyset)$ is called *empty* and denoted by $\emptyset$.

An element of A is called an *arc*. For an arc $(u, v) \in A$, u is said to be *adjacent to* v and v is *adjacent from* u. For a digraph $D = (V, A)$ and a vertex $v \in V$, the *outdegree* of v in D, denoted by $od_D(v)$, is the number of vertices adjacent from v in D and the *indegree* of v in D, denoted by $id_D(v)$, is the number of vertices adjacent to v in D. Then, we define $ini(D) = \{v \in V \mid id(v) = 0\}$ and $fin(D) = \{v \in V \mid od(v) = 0\}$.

For two digraphs $D_1 = (V_1, A_1)$ and $D_2 = (V_2, A_2)$, we denote a digraph $(V_1 \cup V_2, A_1 \cup A_2)$ by $D_1 \cup D_2$. For a digraph $D = (V, A)$ and $W \subseteq V$, we denote a digraph $(V - W, A - \{(v, u) \in A \mid v \in W \text{ or } u \in W\})$ by $D - W$. Furthermore, we denote $in(D, W) = \{v \in V \mid (v, w) \in A, w \in W\}$ and $out(D, W) = \{v \in V \mid (w, v) \in A, w \in W\}$.

For digraphs $D_1 = (V_1, A_1)$ and $D_2 = (V_2, A_2)$, D_1 is a *subgraph* of D_2 if $V_1 \subseteq V_2$ and $A_1 \subseteq A_2$. For a digraph $D = (V, A)$ and a non-empty set $S \subseteq V$, the *subgraph of* D *induced by* S, denoted by $\langle S \rangle_D$, is the maximal subgraph of D of which vertices is S, that is, $\langle S \rangle_D = (S, \{(u, v) \in A \mid u, v \in S\})$.

Let D be a digraph (V, A). Then, a *walk* in D is an alternating sequence $W = v_0 a_1 v_1 \cdots a_n v_n$ of vertices and arcs, beginning and ending with vertices, such that $a_i = (v_{i-1}, v_i)$ for $1 \leq i \leq n$, and refer to W as a v_0-v_n walk. For vertices u and v in V, u is *accessible to* v (in D) if there exists a u-v walk in D.

A digraph D is *acyclic* if there exists no v-v walk in D. Also a digraph D is *transitive* if, for $u, v, w \in V$, it holds that $(u, w) \in A$ whenever it holds that $(u, v) \in A$

and $(v, w) \in A$. Furthermore, for a set L of labels, a digraph D is *labeled* (by L) if every vertex $v \in V$ has a label $l(v) \in L$. We call an acyclic transitive labeled digraph an *ATL-digraph*. For an ATL-digraph $D = (V, A)$, D^- denotes an acyclic labeled digraph obtained by removing an arc $(u, v) \in A$ such that there exist arcs $(u, w) \in A$ and $(w, v) \in A$ from A as possible. Note that D^- is uniquely determined for every ATL-digraph D.

Two digraphs $D_1 = (V_1, A_1)$ and $D_2 = (V_2, A_2)$ are *isomorphic as labeled digraphs*, denoted by $D_1 \cong D_2$, if there exists a bijection φ from V_1 to V_2 such that $(u, v) \in A_1$ if and only if $(\varphi(u), \varphi(v)) \in A_2$, and $l(v) = l(\varphi(v))$ for every $v \in V_1$. A digraph $D_1 = (V_1, A_1)$ is *embedded into* a digraph $D_2 = (V_2, A_2)$ *as labeled digraphs*, denoted by $D_1 \sqsubseteq D_2$, if there exists an injection from V_1 to V_2 such that $(\varphi(u), \varphi(v)) \in A_2$ whenever $(u, v) \in A_1$, and $l(v) = l(\varphi(v))$ for every $v \in V_1$.

In this chapter, we formulate an *episode* as an ATL-digraph of which label is an event type. Also Mannila *et al.* [40] have introduced a *serial episode* as a sequence of events. In this chapter, we formulate a serial episode $a_1 \cdots a_n$ as an ATL-digraph $S = (\{v_1, \ldots, v_n\}, \{(v_i, v_j) \mid 1 \leq i < j \leq n\})$ such that $l(v_i) = a_i$. We sometimes identify a serial episode S with a label sequence $a_1 \cdots a_n$ of S. For an episode D, we denote the set of all serial episodes embedded into D by $se(D)$, that is, $se(D) = \{S \mid S \sqsubseteq D \text{ and } S : \text{serial episode}\}$.

Furthermore, we also formulate an event sequence $\mathcal{S}$ as an ATL-digraph $d(\mathcal{S}) = (V, A)$ satisfying the following conditions.

1. For every event $(e, t) \in \mathcal{S}$, there exists a vertex $v_{e,t} \in V$ such that $l(v_{e,t}) = e$.

2. For every pair $((e, t), (e', t')) \in \mathcal{S} \times \mathcal{S}$ of events, $(v_{e,t}, v_{e',t'}) \in A$ if and only if $t < t'$.

It is obvious that, for an event sequence $\mathcal{S}$, $d(\mathcal{S})$ is determined uniquely.

6.2 Equivalence between Parallel-Free and Serially Constructible Episodes

In this section, we newly introduce a *parallel-free* and a *serially constructible* ATL-digraphs. Then, we show the main result of this chapter that an episode as an ATL-digraph is parallel-free if and only if it is serially constructible.

Definition 17 (Katoh & Hirata [32]) For ATL-digraphs $W = (V_1, A_1)$ and $D = (V_2, A_2)$, we say that D is *parallel-free in* W if for every pair $(u, v) \in V_2 \times V_2$ such that $u \neq v$ and $l(u) = l(v)$, it holds that either $(u, v) \in A_1$ or $(v, u) \in A_1$.

Also we say that D is *parallel in* W if there exists a pair $(u, v) \in V_2 \times V_2$ such that $u \neq v$, $l(u) = l(v)$ and $(u, v), (v, u) \notin A_1$.

Furthermore, we say that D is *parallel-free* (*resp.*, *parallel*) if D is parallel-free (*resp.*, parallel) in D itself.

Note that the notion of "a parallel episode" in this chapter is different from one in Mannila *et al.* [40]. For example, every serial episode is parallel-free. Also $d(\mathcal{S})$ for an event sequence $\mathcal{S}$ is parallel-free. Furthermore, if an ATL-digraph D is parallel-free, then D is parallel-free in an ATL-digraph W such that $D \sqsubseteq W$.

Definition 18 (Katoh & Hirata [32]) An ATL-digraph D is *serially constructible* if it holds that $D \sqsubseteq W$ for every parallel-free ATL-digraph W such that $se(D) \subseteq se(W)$.

Definition 18 requires that, for ATL-digraphs D and W, every serial episode in W is corresponding to exactly one serial episode in D. Hence, by regarding D as an episode and W as a window, Definition 18 claims that a window W contains the information of occurrences of serial episodes in D.

Example 5 (Katoh & Hirata [32]) Let D be a serial episode. Since $D \sqsubseteq D$ and by the definition of $se(D)$, it holds that $D \in se(D)$. Then, for every parallel-free ATL-digraph W such that $se(D) \subseteq se(W)$, it holds that $D \in se(W)$. By the definition of

$se(W)$, it holds that $D \sqsubseteq W$, which implies that D is serially constructible. Hence, every serial episode is serially constructible.

Example 6 (Katoh & Hirata [32]) Let D and W be ATL-digraphs such that D^- and W^- are described as Figure 6.2. Then, it holds that W is parallel-free and $se(D) = se(W) = \{a, b, c, ab, bb, bc, abc, bbc, abbc\}$. However, there exists no injection from $V(D)$ to $V(W)$, so $D \not\sqsubseteq W$. Hence, D is *not* serially constructible.

D^- W^-

Figure 6.2: D^- and W^- in Example 6.

In order to show the equivalence between parallel-free and serially constructible episodes, it is necessary to decompose a digraph into a certain collection of digraphs. Then, we introduce a *decomposition* of digraphs.

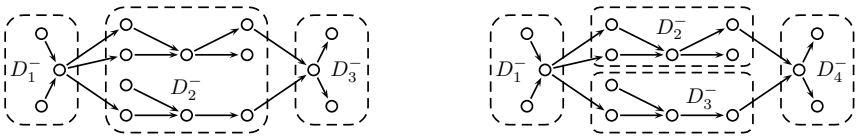
For three digraphs $D_i = (V_i, A_i)$ $(i = 1, 2, 3)$ (that are possibly empty) and two sets $B_1 = \{(u, v) \mid u \in V_1, v \in V_2\}$ and $B_2 = \{(u, v) \mid u \in V_2, v \in V_3\}$ of arcs, $D_1 \oplus_{B_1} D_2 \oplus_{B_2} D_3$ denotes a digraph (V, A) such that $V = V_1 \cup V_2 \cup V_3$, $V_i \cap V_j = \emptyset$ $(1 \leq i < j \leq 3)$, and $A = A_1 \cup A_2 \cup A_3 \cup B_1 \cup B_2$.

Definition 19 Let $D = (V, A)$ be an ATL-digraph.

1. We say that D has a *serial decomposition* if there exist ATL-digraphs $D_i = (V_i, A_i)$ $(i = 1, 2, 3)$ and sets B_i $(i = 1, 2)$ of arcs such that $D^- = D_1^- \oplus_{B_1} D_2^- \oplus_{B_2} D_3^-$. In this case, we denote D by $[D_1, D_2, D_3]$.

2. We say that D has a *diamond decomposition* if there exist ATL-digraphs $D_i = (V_i, A_i)$ $(i = 1, 2, 3, 4)$ and sets B_i $(i = 1, 2, 3, 4)$ of arcs such that $D^- = (D_1^- \oplus_{B_1} D_2^- \oplus_{B_2} D_4^-) \cup (D_1^- \oplus_{B_3} D_3^- \oplus_{B_4} D_4^-)$ and $D_2 \cup D_3$ is parallel-free in D. In this case, we denote D by $[D_1, \langle D_2, D_3 \rangle, D_4]$.

Example 7 Consider the ATL-digraph D such that D^- is described as Figure 6.3, where every label in D is assumed to be distinct. Then, D has a serial decomposition $[D_1, D_2, D_3]$ and a diamond decomposition $[D_1, \langle D_2, D_3 \rangle, D_4]$, where D_i^- is described as the dashed boxes in Figure 6.3.



serial decomposition diamond decomposition

Figure 6.3: D^- in Example 7

Lemma 37 *Let W be an ATL-digraph. Also let D and E be ATL-digraphs embedded into W having serial decompositions $[D_1, D_2, D_3]$ and $[E_1, E_2, E_3]$, respectively. If $D_1 \cong E_1$, $D_3 \cong E_3$ and $D_2 \cup E_2$ are parallel-free in W, then there exists an ATL-digraph F that is parallel-free in W, embedded into W and has a diamond decomposition $[F_1, \langle F_D, F_E \rangle, F_3]$ satisfying the following conditions. See Figure 6.4.*

1. *$[F_1, F_D, F_3] \cong [D_1, D_2, D_3]$ and $[F_1, F_E, E_3] \cong [E_1, E_2, E_3]$.*
2. *$F_1 \cong D_1 (\cong E_1)$ and $F_3 \cong D_3 (\cong E_3)$.*
3. *$F_D \cong D_2$ and $F_E \cong E_2$.*
4. *For every $v \in V(F_1)$, it holds that either $v \in V(D_1)$ or $v \in V(E_1)$.*
5. *For every $v \in V(F_3)$, it holds that either $v \in V(D_3)$ or $v \in V(E_3)$.*

(proof) We show the statement by induction on $|D_1|$ and $|D_3|$.

Consider the case that $|D_1| = |D_3| = 0$, that is, D_1 and D_3 are empty. Since $D_1 \cong E_1$ and $D_3 \cong E_3$, E_1 and E_3 are empty. Let F_1 and F_3 be empty digraphs.

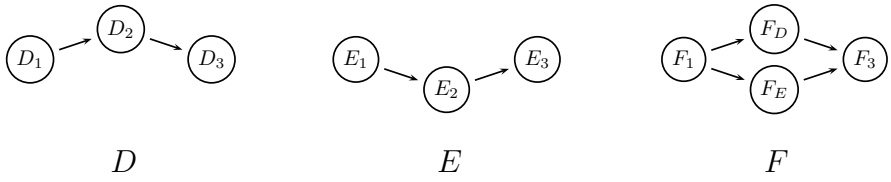


Figure 6.4: Intuitive figures of D, E and F in Lemma 37.

Then, the condition 1 obviously holds. Furthermore, since $D_2 \cup E_2$ is parallel-free, it holds that $[\emptyset, \langle D_2, E_2 \rangle, \emptyset]$ is parallel-free.

Suppose that the statement holds for $|D_1| = |E_1| < n$ and $|D_3| = |E_3| < m$, and consider the case that $|D_3| = |E_3| = m$. Let φ be a bijection on $D_1 \cong E_1$ and $D_3 \cong E_3$. Also let v_1 be a vertex in $\mathit{fin}(D_3)$ and v_2 be $\varphi(v_1) \in E_3$. It is obvious that $v_2 \in \mathit{fin}(E_3)$. Furthermore, let $D'_1 = D_1$, $D'_2 = D_2$, $D'_3 = D_3 - \{v_1\}$ and $D'_4 = \langle \{v_1\} \rangle_{D_3}$. Then, we can write D as Figure 6.5. Here, for a set $A_{i,j} = \{(u, v) \in A(D) \mid u \in D_i, v \in D_j\}$ $(1 \leq i < j \leq 3)$ of arcs in D, every set $A'_{i,j}$ of arcs from $V(D'_i)$ to $V(D'_j)$ in D $(1 \leq i < j \leq 4)$ satisfies the following statements.

$$A'_{1,4} = A_{1,3} \cap \{(v, v_1) \in A(D) \mid v \in V(D_1)\}, \quad A'_{1,3} = A_{1,3} - A'_{1,4}, \quad A'_{1,2} = A_{1,2},$$
$$A'_{2,4} = A_{2,3} \cap \{(v, v_1) \in A(D) \mid v \in V(D_2)\}, \quad A'_{2,3} = A_{2,3} - A'_{2,4},$$
$$A'_{3,4} = \{(v, v_1) \in A(D) \mid v \in V(D_3)\}.$$

Also let $E'_1 = E_1$, $E'_2 = E_2$, $E'_3 = E_3 - \{v_2\}$ and $E'_4 = \langle \{v_2\} \rangle_{E_3}$. Then, we can write E as Figure 6.5. Here, for a set $B_{i,j} = \{(u, v) \in A(E) \mid u \in E_i, v \in E_j\}$ $(1 \leq i < j \leq 3)$ of arcs in E, every set $B'_{i,j}$ of arcs from $V(E'_i)$ to $V(E'_j)$ in E $(1 \leq i < j \leq 4)$ satisfies the following statements.

$$B'_{1,4} = B_{1,3} \cap \{(v, v_2) \in A(E) \mid v \in V(E_1)\}, \quad B'_{1,3} = B_{1,3} - B'_{1,4}, \quad B'_{1,2} = B_{1,2},$$
$$B'_{2,4} = B_{2,3} \cap \{(v, v_2) \in A(E) \mid v \in V(E_2)\}, \quad B'_{2,3} = B_{2,3} - B'_{3,4},$$
$$B'_{3,4} = \{(v, v_2) \in A(E) \mid v \in V(E_3)\}.$$

Let $D' = D - \{v_1\}$ and $E' = E - \{v_2\}$. Then, it holds that $D' = [D'_1, D'_2, D'_3]$ and $E' = [E'_1, E'_2, E'_3]$. Since $|D'_3| = |E'_3| < n$ and by induction hypothesis, there exist ATL-digraphs F'_1 and F'_3 satisfying the following conditions. See Figure 6.6.

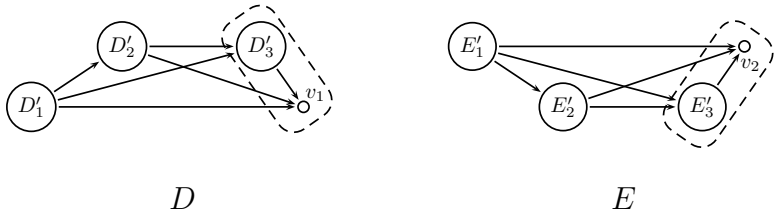


Figure 6.5: Intuitive figures of D and E, where the dashed boxes mean D_3 and E_3, respectively.

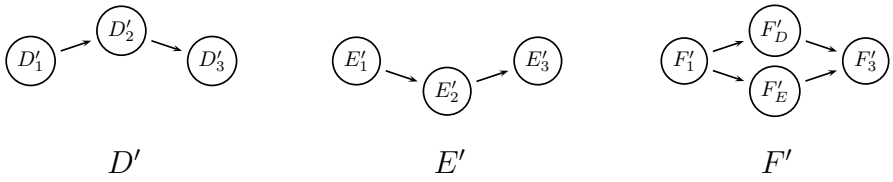


Figure 6.6: Intuitive figures of D', E' and F'.

1. $F' = [F'_1, \langle F'_D, F'_E \rangle, F'_3]$ is parallel-free and embedded into D.
2. $[F'_1, F'_D, F'_3] \cong [D'_1, D'_2, D'_3]$ and $[F'_1, F'_E, F'_3] \cong [E'_1, E'_2, E'_3]$.
3. $F'_1 \cong D'_1 (\cong E'_1)$ and $F'_3 \cong D'_3 (\cong E'_3)$.
4. $F'_D \cong D'_2$ and $F'_E \cong E'_2$.
5. For every $v \in V(F'_1)$, either $v \in V(D'_1)$ or $v \in V(E'_1)$.
6. For every $v \in V(F'_3)$, either $v \in V(D'_3)$ or $v \in V(E'_3)$.

In the following, we explain how to construct F_3 satisfying the statements. Since W is parallel-free and by the definition of v_1 and v_2, v_1 and v_2 satisfy one of the conditions (a) $v_1 = v_2$, (b) $(v_1, v_2) \in A(W)$ or (c) $(v_2, v_1) \in A(W)$. We denote $A'_{1,4} \cup A'_{2,4} \cup A'_{3,4}$ and $B'_{1,4} \cup B'_{2,4} \cup B'_{3,4}$ by $A'_{*,4}$ and $B'_{*,4}$, respectively.

(a) In the case that $v_1 = v_2$, construct the following ATL-digraph F_3, see Figure 6.7 (a).

$$F_3 = (V(F'_3) \cup \{v_1\}, A(F'_3) \cup \{(v, v_1) \in A(W) \mid v \in V(F'_3)\}).$$

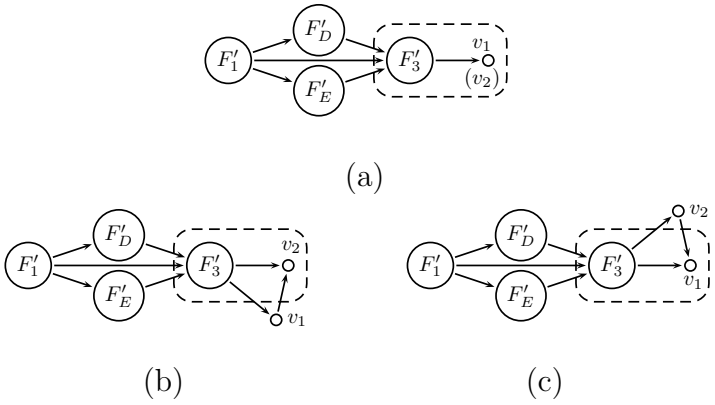


Figure 6.7: Intuitive figures of F, where the dashed box is F_3 and we omit the arcs from F'_1, F'_D and F'_E to v_1 and v_2.

We denote an ATL-digraph by adding arcs $A'_{*,4} \cup B'_{*,4}$ to $[F'_1, \langle F'_D, F'_E \rangle, F_3]$ by F. Since F' is parallel-free, so is F. Furthermore, we check the conditions in the statement as follows.

Since $[F'_1, F'_D, F'_3] \cong D'$ and by the definition of D'_3 and D'_4, it holds that $[F'_1, F'_D, F_3] \cong D$. Also, since $[F'_1, F'_E, F'_3] \cong E'$ and by the definition of E'_3 and E'_4, it holds that $[F'_1, F'_E, F_3] \cong E$. Hence, the condition 1 holds. By induction hypothesis, the conditions 2, 3 and 4 also hold. Finally, we show the condition 5. For every $v \in V(F_3)$, if $v \neq v_1$, then either $v \in V(D'_3)$ or $v \in V(E'_3)$. Since $D'_3 = D_3 - \{v_1\}$ and $E'_3 = E_3 - \{v_1\}$, it holds that either $v \in V(D_3)$ or $v \in V(E_3)$. If $v = v_1$, then it holds that $v_1 \in V(D_3)$, since $v_1 \in \mathit{fin}(D_3)$. Hence, for every $v \in F_3$, either $v \in V(D_3)$ or $v \in V(E_3)$.

(b) Consider the case that $(v_1, v_2) \in A(W)$. Since $F'_3 \cong D'_3 \cong E'_3$ and $D_3 \cong E_3$, for a given bijection φ, $(v, v_1) \in A(D_3)$ if and only if $(\varphi(v), v_2) \in A(E_3)$, where $v_2 = \varphi(v_1)$. Since either $v \in V(D'_3)$ or $v \in V(E'_3)$ for every $v \in V(F'_3)$, there exists a vertex $v \in V(F'_3)$ such that either $(v, v_1) \in A(D_3)$ or $(v, v_2) \in A(E_3)$. For the former case, if $(v, v_1) \in A(D_3)$, then there exists an arc $(v, v_2) \in A(W)$, since W is transitive and by the supposition that $(v_1, v_2) \in A(W)$. Then, in both cases, there exists a vertex $v \in V(F'_3)$ such that $(v, v_2) \in A(W)$.

Hence, construct the following ATL-digraph F_3, see Figure 6.7 (b).

$$F_3 = (V(F'_3) \cup \{v_2\}, A(F'_3) \cup \{(v, v_2) \in A(W) \mid v \in V(F'_3)\}).$$

We denote an ATL-digraph by adding arcs $A'_{*,4} \cup B'_{*,4}$ to $[F'_1, \langle F'_D, F'_E \rangle, F_3]$ by F. Then, we can check that F satisfies the conditions as similar as the case (a).

(c) Consider the case that $(v_2, v_1) \in A(W)$. By the same reason of the case (b), there exists a vertex $v \in V(F_3)$ such that either $(v, v_1) \in A(D_3)$ or $(v, v_2) \in A(E_3)$. For the latter case, if $(v, v_2) \in A(E_3)$, then there exists an arc $(v, v_1) \in A(W)$, since W is transitive and by the supposition that $(v_2, v_1) \in A(W)$. Then, in both cases, there exists a vertex $v \in V(F_3)$ such that $(v, v_1) \in A(W)$.

Hence, construct the following ATL-digraph F_3, see Figure 6.7 (c).

$$F_3 = (V(F'_3) \cup \{v_1\}, A(F'_3) \cup \{(v, v_1) \in A(W) \mid v \in V(F'_3)\}).$$

We denote an ATL-digraph by adding arcs $A'_{*,4} \cup B'_{*,4}$ to $[F'_1, \langle F'_D, F'_E \rangle, F_3]$ by F. Then, we can check that F satisfies the conditions as similar as the case (a).

Furthermore, we can give the similar proof of the case that $|D_1| = |E_1| = n$. Hence, the statement holds. □

Theorem 38 *Every parallel-free ATL-digraph is serially constructible.*

(proof) For a parallel-free ATL-digraph F, we show the statement by induction on $|F|$. If $|F| \leq 1$, then F is a serial episode, so the statement holds.

Suppose that the statement holds for $|F| < n$ and consider the case that $|F| = n$. If F is a serial episode, then the statement obviously holds, so suppose that F is not a serial episode. Then, there exist vertices u and v in F such that (1) $(u, v) \notin A(F)$ and (2) $(v, u) \notin A(F)$. Let $acc(F, v) = \{v\} \cup in(F, \{v\}) \cup out(F, \{v\})$. Since F is transitive, $u \in acc(F, v)$ implies that either u is accessible to v in F or v is accessible to u in F. Then, for this v, we construct the following ATL-digraphs F_1, F_2, F_3 and F_4:

$$
\begin{aligned}
F_2 &= \langle V(F) - acc(F, v)\rangle_F, \\
F_1 &= \langle in(F, \{v\}) \cap in(F_2, \{v\})\rangle_F, \\
F_4 &= \langle out(F, \{v\}) \cap out(F_2, \{v\})\rangle_F, \\
F_3 &= \langle V(F) - (V(F_1) \cup V(F_2) \cup V(F_4))\rangle_F.
\end{aligned}
$$

For example, consider an ATL-digraph F such that F^- is described in Figure 6.8 (left). Suppose that $v \in V(F)$ in Figure 6.8 (left) satisfies the above condition. Then, we obtain F_i^- $(i = 1, 2, 3, 4)$ as the dashed boxes in Figure 6.8 (right).

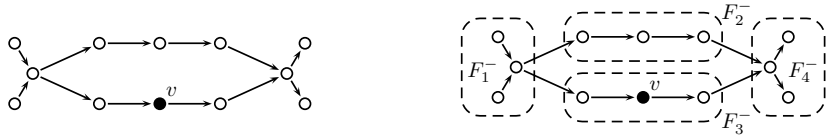


Figure 6.8: ATL-digraphs F^- (left) and F_i^- $(i = 1, 2, 3, 4)$ (right).

Then, from F, we can construct the ATL-digraphs $X = [F_1, F_2, F_4]$ and $Y = [F_1, F_3, F_4]$. By the construction of F_i, it is obvious that $v \in V(F_3)$ and $u \in V(F_2)$, so it holds that $|X| < n$ and $|Y| < n$. Also it holds that $se(X) \subseteq se(F)$ and $se(Y) \subseteq se(F)$. Since F is parallel-free, $F_2 \cup F_3$ is also parallel-free

Let W be a parallel-free ATL-digraph such that $se(F) \subseteq se(W)$. Then, it holds that $se(X) \subseteq se(W)$ and $se(Y) \subseteq se(W)$. Since X and Y are serially constructible and by induction hypothesis, it holds that $X \sqsubseteq W$ and $Y \sqsubseteq W$.

By regarding X and Y as D and E in Lemma 37, there exists an ATL-digraph $Z = [F_1, \langle F_D, F_E\rangle, F_4]$ that is parallel-free in W and embedded into W. In particular, it holds that $F \cong Z$, so it holds that $F \sqsubseteq W$. Hence, F is serially constructible. □

Theorem 39 *Every serially constructible ATL-digraph is parallel-free.*

(proof) By contraposition, it is sufficient to show that, for every parallel ATL-digraph $D = (V, A)$, there exists an ATL-digraph W such that $se(D) \subseteq se(W)$ but $D \not\sqsubseteq W$.

Since D is parallel, there exists a pair $(u, v) \in V \times V$ such that $u \neq v$, $l(u) = l(v)$ and $(u, v), (v, u) \notin A$. For such u and v, let A_u and $A_{u \to v}$ be the following sets of arcs.

$$\begin{aligned} A_u &= \{(w, u) \in A \mid w \in in(D, \{u\})\} \cup \{(u, w) \in A \mid w \in out(D, \{u\})\}, \\ A_{u \to v} &= \{(w, v) \mid w \in in(D, \{u\})\} \cup \{(v, w) \mid w \in out(D, \{u\})\}. \end{aligned}$$

Furthermore, let W be an ATL-digraph $W = (V - \{u\}, (A - A_u) \cup A_{u \to v})$. Then, $D - \{u\} (= (V - \{u\}, A - A_u))$ is a subgraph of W.

Let $S_k^n(u) = v_1 \cdots v_{k-1} u v_{k+1} \cdots v_n$ and $S_k^n(v) = v_1 \cdots v_{k-1} v v_{k+1} \cdots v_n$ be serial episodes containing u and v at k with length n ($n \geq 1, 1 \leq k \leq n$). By the definition of W, for every serial episode $S_k^n(u)$ embedded into D, there exists a serial episode $S_k^n(v)$ embedded into W. Since $l(u) = l(v)$, it holds that $S_k^n(u) \cong S_k^n(v)$. Furthermore, since $D - \{u\}$ is a subgraph of W, every serial episode not containing u and embedded into D is also embedded into W. Hence, it holds that $se(D) \subseteq se(W)$.

On the other hand, since $|W| = |D| - 1$, there exists no injection from $V(D)$ to $V(W)$. Hence, it holds that $D \not\sqsubseteq W$. □

By summarizing Theorem 38 and 39 and by regarding an episode as an ATL-digraph, we obtain the following main result of this chapter.

Theorem 40 *An episode is parallel-free if and only if it is serially constructible.*

6.3 Chapter Summary

In this chapter, by formulating both an episode and an event sequence as an ATL-digraph, we have introduced the concept of a *parallel-free* and a *serially constructible episodes*. Then, we have shown that *an episode is parallel-free if and only if it is serially constructible*. Since a serially constructible episode is an episode that is constructible from just information for occurrences of serial episodes, this equivalence result gives one of the theoretical limitations on efficiently constructing episodes.

Chapter 7

Practical Applications for Bacterial Culture Data

In this chapter, we study the applications of episode mining from medical data, called bacterial culture data. we use episode mining algorithms proposed so far including some algorithms in the previous chapters.

In analysis of bacterial culture data, several data mining algorithms have been applied [64, 58, 41, 42]. For example, Hirata *et al.* [51, 21, 50] have paid attention to *data mining from bacterial culture data*. Then, they have developed to extract *disjunctive rules*, called *frequent monotone DNF formulas* [21], *frequent closed monotone DNF formulas* [51] and *frequent few-overlapped monotone DNF formulas* [50], as hypotheses to explain bacterial culture data nearly overall. Also they have designed the algorithm to construct decision trees classifying MRSA to MSSA [20] and then implemented the prototype of risk management system for hospital-acquired infection [22]. In medical data mining [54, 19, 44, 18, 57, 55, 56, 17] such as the analysis of medical data, the analysis of temporal information is very important. However, it remains open how to deal with temporal information in bacterial culture data and how to extract frequent episodes representing temporal rules. To solve this problem, we apply *episode mining* to the bacterial culture data and extracted the *episodes*

as the time-related rules representing *replacements of bacteria* and *changes for drug resistance* as the factors of hospital-acquired infection.

This chapter is based on the papers [38, 34, 35].

7.1 Sectorial Episodes

As the form of an episode, Katoh *et al.* [36] have introduced a *sectorial episode* of the form $C \mapsto r$, where C is a set of event types and r is an event type. The sectorial episode $C \mapsto r$ means that every event of C is followed by an event r, so we can regard every event in C as a candidate of *causation* of r. Note that, the class of sectorial episodes is the subclass of diamond episodes.

Let $\mathcal{S}$ be an event sequence k an window width. In this section, by regarding a sectorial episode $X = C \mapsto r$ as an association rule [2], we can formulate the *support* $supp_{\mathcal{S},k}(X)$ and the *confidence* $conf_{\mathcal{S},k}(X)$ of X as follows:

$$supp_{\mathcal{S},k}(X) = \frac{freq_{\mathcal{S},k}(X)}{|\mathbf{W}_{\mathcal{S},k}(X)|}, \quad conf_{\mathcal{S},k}(X) = \frac{freq_{\mathcal{S},k}(X)}{freq_{\mathcal{S},k}(C)}.$$

In the following, the subscripts $\mathcal{S}$ and k are omitted if they are clear by the context.

In this section, for the *minimum support* σ $(0 < \sigma < 1)$, we say that a sectorial episode X is *frequent* if $supp(X) \geq \sigma$. Also, for the *minimum confidence* γ $(0 < \gamma < 1)$, we say that a sectorial episode X is *accurate* if $conf(X) \geq \gamma$. Then, the algorithm Sect [36] is the mining algorithm to extract *sectorial episodes that are frequent and accurate*.

7.1.1 Experiments for bacterial culture data

In this section, we regard bacterial culture data as event sequences, one of the time-related data. Then, we apply the algorithm Sect [36] to bacterial culture data. By the form $C \mapsto r$ of sectorial episodes, we can regard an element of C as cause of an event r.

The purpose of this section is to extract the sectorial episodes representing *changes for drug resistance* and *replacements of bacteria*. In the former, we extract sectorial episodes of the form $C \mapsto r$ such that C contains an event denoting some antibiotics is susceptibility and an event r is an event denoting the antibiotics is resistant. In the latter, we extract sectorial episodes of the form $C \mapsto r$ such that C contains an event denoting the occurrence of bacterium and an event r is an event denoting the occurrence of different bacterium.

In this section, we apply the algorithm SECT to bacterial culture data from Osaka Prefectural General Medical Center from 1995 to 1998, which are complete data in [53]. Then, we extract sectorial episodes representing *changes for drug resistance* and *replacements of bacteria*. Here, we regard a pair of "attribute=value" as an event type.

In the following, we describe the event sequence by using the parameters (d, p, w), where d is the number of records, p is the number of patients and w is the number of windows, respectively. Furthermore, we denote the number of extracted sectorial episodes by e.

7.1.2 The frequent sectorial episodes for changes for drug resistance

First, we extract sectorial episodes representing changes for drug resistance*. By fixing the category of bacteria and the sample, we connect data of every patient with the span of 30 days, which is the width of windows. Then, we investigate the sectorial episode of the following form:

$$C \mapsto (\texttt{Ant=R}),$$

*In the previous work [36], we have already reported the experimental results from the same data, but the implementation of the algorithm SECT contains some errors. Hence, the result given in this section is different from one given by [36].

where (Ant=S) $\in C$. The attribute values S and R denote "susceptibility" and "resistant," respectively. We call such an episode a *target episode* (of an antibiotic Ant), which means that the drug resistance of Ant changes from S to R.

In this section, Ant(n) denotes that the number of extracted target episodes of an antibiotic Ant is n. Here, antibiotics are benzilpenicillin (PcB), augmentin (Aug), anti-pseudomonas penicillin (PcAP), 1st generation cephems (Cep1), 2nd generation cephems (Cep2), 3rd generation cephems (Cep3), aminoglycosides (AG), macrolides (ML), tetracyclines (TC), and carbapenems (CBP).

Staphylococci

Consider the case that the category of bacteria is "Staphylococci." Then, we can extract episodes from the samples "catheter/others," "respiratory organs," and "blood." The following table describes the parameters of the event sequence for the above samples.

sample	d	p	w
catheter/others	775	249	41521
respiratory organs	1014	311	46062
blood	89	33	2830

For the first two samples, we set the minimum support $\sigma = 0.01$. Then, for the minimum confidence γ, we obtain the following target episodes.

sample	γ	e	antibiotics
catheter/others	0.05	24	PcS(16), Cep1(8)
	0.10	0	–
respiratory organs	0.05	10776	ML(4672), Cep1(3792), PcS(2128), TC(184)
	0.10	5644	Cep1(3512), PcS(1860), ML(272)

On the other hand, for the sample "blood," we set the minimum support $\sigma = 0.04$ and 0.03. Then, for the minimum confidence γ, we obtain the following target episodes.

σ	γ	e	antibiotics
0.04	0.10	560	AG(544), CBP(16)
	0.15	16	AG(16)
0.03	0.10	7208	AG(3008), PcS(2048), Cep1(1728), CBP(400), TC(24)
	0.15	4992	PcS(2048), Cep1(1728), AG(976), CBP(224), TC(16)

Hence, for the above three samples, we can extract frequent target episodes of PcS and Cep1 in "Staphylococci." Also, for the sample "blood," we can extract ones of AG and CBP. On the other hand, the frequency of target episodes of ML for the sample "respiratory organs" and AG for the sample "blood" rapidly decreases when the minimum confidence increases.

Enterococci

Consider the case that the category of bacteria is "Enterococci." Then, we can extract episodes from the samples "catheter/others" and "urinary/genital organs." The following table describes the parameters the event sequence for the above samples.

sample	d	p	w
catheter/others	206	70	9318
urinary/genital organs	120	45	7601

We set the minimum support $\sigma = 0.01$. Then, for the minimum confidence γ, we obtain the following target episodes.

sample	γ	e	antibiotics
catheter/others	0.05	2112	PcB(1792), TC(320)
	0.10	640	PcB(512), TC(128)
urinary/genital organs	0.05	96	PcB(96)
	0.10	0	–

Hence, for the above two samples, we can extract frequent target episodes of PcB and TC in "Enterococci," where the former is more frequent than the latter.

Enteric bacteria

Consider the case that the category of bacteria is "Enteric bacteria." Then, we can extract episodes from the samples "catheter/others" and "respiratory organs." The following table describes the parameters of the event sequence for the above samples.

sample	d	p	w
catheter/others	441	133	14379
respiratory organs	987	308	39917

For the sample "catheter/others," we set the minimum support $\sigma = 0.04$ and 0.03. Then, for the minimum confidence γ, we obtain the following target episodes.

σ	γ	e	antibiotics
0.04	0.10	384	`Aug`(384)
	0.15	352	`Aug`(352)
0.03	0.10	38784	`Aug`(32768), `Cep1`(6016)
	0.15	9764	`Aug`(9764)

On the other hand, for the sample "respiratory organs," we set the minimum support $\sigma = 0.03$ and 0.02. Then, for the minimum confidence γ, we obtain the following target episodes.

σ	γ	e	antibiotics
0.02	0.10	168396	`Aug`(78832), `Cep1`(76016), `Cep2`(13548)
	0.15	42104	`Aug`(23152), `Cep1`(18952)
0.03	0.10	1536	`Aug`(880), `Cep1`(656)
	0.15	0	–

Hence, for the above two samples, we can extract frequent target episodes of `Aug` and `Cep1` in "Enteric bacteria," where the former is more frequent than the latter.

Glucose-nonfermentative gram-negative bacteria

Consider the case that the category of bacteria is "Glucose-nonfermentative gram-negative bacteria." Then, we can extract episodes from the samples "catheter/others" and "respiratory organs." The following table describes the parameters of the event sequence for the above samples.

sample	d	p	w
catheter/others	238	81	12585
respiratory organs	1030	302	48195

For these samples, we set the minimum support $\sigma = 0.01$. Then, for the minimum confidence γ, we obtain the following target episodes.

sample	γ	e	antibiotics
catheter/others	0.05	1096	CBP(1088), TC(8)
	0.10	372	CBP(368), TC(8)
	0.15	0	–
respiratory organs	0.05	15176	CBP(12224), TC(2088), Cep3(544), Aug(320)
	0.10	14260	CBP(11788), TC(2088), Aug(320), Cep3(64)
	0.15	6652	CBP(4308), TC(2024), Aug(320)

Hence, for the above two samples, we can extract frequent target episodes of CBP and TC in "Glucose-nonfermentative gram-negative bacteria," where the former is more frequent than the latter. Also, for the sample "respiratory organs," we can extract ones of Aug, but they are more infrequent than ones of CBP and TC.

Other gram-negative bacilli

Consider the case that the category of bacteria is "Other gram-negative bacilli." Then, we can extract episodes from the sample "respiratory organs." The parameters of the event sequence for the sample "respiratory organs" is that $(d, p, w) = (484, 179, 32759)$.

We set the minimum support $\sigma = 0.01$. Then, for the minimum confidence $\gamma = 0.05$ and 0.10, we obtain the target episodes of which antibiotics is PcB(6524) and PcB(1536), respectively. Also for the minimum confidence $\gamma = 0.15$, we can obtain no target episodes.

Hence, for the sample "respiratory organs," we can extract frequent target episodes of just PcB in "Other gram-negative bacilli."

Anaerobes

Consider the case that the category of bacteria is "Anaerobes." Then, we can extract episodes from the samples "catheter/others," "punctual fluid," and "blood." The following table describes the parameters of the event sequence for the above samples.

sample	d	p	w
catheter/others	763	218	19032
punctual fluid	70	27	941
blood	114	45	1785

For the sample "catheter/others," we set the minimum support $\sigma = 0.05$ and 0.04. Then, for the minimum confidence γ, we obtain the following target episodes.

σ	γ	e	antibiotics
0.05	0.10	416	`Cep1`(352), `PcB`(64)
	0.15	192	`Cep1`(128), `PcB`(64)
0.04	0.10	41152	`PcB`(27008), `Cep1`(14144)
	0.15	14216	`PcB`(13408), `Cep1`(808)

For the last two samples "punctual fluid" and "blood," we set the minimum support $\sigma = 0.05$. Then, for the minimum confidence γ, we obtain the following target episodes.

sample	γ	e	antibiotics
punctual fluid	0.10	114688	`PcB`(57344), `Cep1`(57344)
	0.30	65536	`PcB`(32768), `Cep1`(32768)
	0.50	16384	`PcB`(8192), `Cep1`(8192)
blood	0.05	1728	`PcAP`(1216), `ML`(512)
	0.10	1088	`PcAP`(576), `ML`(512)
	0.15	512	`PcAP`(512)

Hence, for the samples "catheter/others" and "punctual fluid," we can extract frequent target episodes of `PcB` and `Cep1` in "Anaerobes." On the other hand, for the sample "blood," we can extract frequent target episodes of `PcAP` and `ML` in "Anaerobes."

Note that, for the sample "punctual fluid," the minimum confidence is very large than other experimental results in this section. Furthermore, in this case, the frequency of target episodes of `PcB` is same as ones of `Cep1` for every minimum confidence.

Summary

Finally, we summarize the above experimental results for the viewpoint of samples. The following table describes the changes for drug resistance for every samples.

sample	bacterium	antibiotics
catheter/others	Staphylococci	PcS, Cep1
	Enterococci	PcB, TC
	Enteric bacteria	Aug, Cep1
	Glucose-nonfermentative gram-negative bacteria	TC, CBP
	Anaerobes	PcB, Cep1
punctual fluid	Anaerobes	PcB, Cep1
urinary/genital organs	Enterococci	PcB
respiratory organs	Staphylococci	Cep1, ML PcS, TC
	Enteric bacteria	Aug, Cep1, Cep2
	Glucose-nonfermentative gram-negative bacteria	Aug, Cep3, TC, CBP
	other gram-negative bacilli	PcB
blood	Staphylococci	PcS, Cep1, AG, TC, CBP
	Anaerobes	PcAP, ML

The previous work [67] has reported that the drug resistance of CBP changes from S to R in Anaerobes from this data. While we can observe no changes for drug resistance of CBP in "Anaerobes," we can newly observe ones in "Glucose-nonfermentative gram-negative bacteria" for the samples "catheter/others" and "respiratory organs," and in "Staphylococci" for the sample "blood." Furthermore, in "Anaerobes," we can newly observe the changes for the drug resistance of PcB and Cep1 for the samples "catheter/others" and "punctual fluid," and ones of PcAP and ML for the sample "blood."

7.1.3 The frequent sectorial episodes for replacements of bacteria

Next, we extract the sectorial episodes representing *replacements of bacteria*. By fixing the sample, we connect data of every patient with the span of 30 days. Then, the following table describes the parameters of the event sequence for samples.

sample	d	p	w
catheter/others	3642	834	94568
punctual fluid	226	67	3322
urinary/genital organs	1152	386	39139
digestive tract	418	141	11465
respiratory organs	5639	1337	170604
blood	468	162	8986

In this subsection, we pay our attention to the sectorial episode of the following form:

$$C \mapsto (\texttt{Bac=bac2}),$$

where (Bac=bac1) $\in C$ such that bac1$\neq$bac2. We call the attribute values bac1 and bac2 a *precedent bacterium* and a *subsequent bacterium*, respectively. We call such an episode a *target episode*.

Catheter/others

Consider the case that the sample is "catheter/others." We set the minimum support $\sigma = 0.005$ and the minimum confidence $\gamma = 0.10$. Then, we obtain the following 215248 target episodes.

precedent	subsequent	e
Enterococcus faecalis	Pseudomonas aeruginosa	214768
Bacteroides fragilis	Enterococcus faecalis	480

Note that we can obtain no target episodes under the minimum confidence $\gamma = 0.20$.

Punctual fluid

Consider the case that the sample is "punctual fluid." We set the minimum support $\sigma = 0.0175$. In this case, we just extract the following target episodes.

precedent	subsequent
Enterococcus faecalis	Bacteroides fragilis

For the minimum confidence γ, the number of target episodes is 65536 ($\gamma = 0.10$), 56832 ($\gamma = 0.20$), 18432 ($\gamma = 0.30$) and 0 ($\gamma = 0.40$).

Urinary/genital organs

Consider the case that the sample is "urinary/genital organs." We set the minimum support $\sigma = 0.0025$. Then, we obtain the following target episodes.

$\gamma = 0.10$

precedent	subsequent	e
Klebsiella pneumoniae	Enterococcus faecalis	24448
Morganella morganii	Pseudomonas aeruginosa	16352
Citrobacter freundii	Enterococcus faecalis	4072
Staphylococcus aureus	Enterococcus faecalis	3352
Enterococcus faecalis	Pseudomonas aeruginosa	1552
Pseudomonas aeruginosa	Enterococcus faecalis	384

$\gamma = 0.20$

precedent	subsequent	e
Morganella morganii	Pseudomonas aeruginosa	8192
Staphylococcus aureus	Enterococcus faecalis	256

$\gamma = 0.30$

precedent	subsequent	e
Morganella morganii	Pseudomonas aeruginosa	8192

For the minimum confidence γ, the total number of target episodes is 50160 ($\gamma = 0.10$), 8448 ($\gamma = 0.20$), 8192 ($\gamma = 0.30$) and 0 ($\gamma = 0.40$).

Digestive tract

Consider the case that the sample is "digestive tract." We set the minimum support $\sigma = 0.0075$. In this case, we just extract the following target episodes.

precedent	subsequent
Staphylococcus aureus	Pseudomonas aeruginosa

For the minimum confidence γ, the number of target episodes is 11168 ($\gamma = 0.10$), 3016 ($\gamma = 0.20$), 1272 ($\gamma = 0.30$), 736 ($\gamma = 0.40$), 240 ($\gamma = 0.50$) and 0 ($\gamma = 0.60$).

Respiratory organs

Consider the case that the sample is "respiratory organs." We set the minimum support $\sigma = 0.01$. Then, we obtain the following target episodes.

$\gamma = 0.10$

precedent	subsequent	e
Pseudomonas aeruginosa	Staphylococcus aureus	517480
Staphylococcus aureus	Pseudomonas aeruginosa	149100

$\gamma = 0.20$

precedent	subsequent	e
Pseudomonas aeruginosa	Staphylococcus aureus	71712

For the minimum confidence γ, the number of target episodes is 666580 ($\gamma = 0.10$), 71712 ($\gamma = 0.20$) and 0 ($\gamma = 0.30$).

Blood

Consider the case that the sample is "blood." We set the minimum support $\sigma = 0.02$. In this case, we just extract the following target episodes.

precedent	subsequent
Streptococcus intermedius	Staphylococcus aureus

For the minimum confidence γ, the number of target episodes is 16384 ($\gamma = 0.10$), 16384 ($\gamma = 0.20$), 8128 ($\gamma = 0.30$) and 0 ($\gamma = 0.40$).

Summary

We summarize the above experimental results. The following table describes the replacements of bacterium for every samples.

sample	precedent	subsequent
catheter/others	Bacteroides fragilis	Enterococcus faecalis
	Enterococcus faecalis	Pseudomonas aeruginosa
punctual fluid	Enterococcus faecalis	Bacteroides fragilis
urinary/genital organs	Enterococcus faecalis	Pseudomonas aeruginosa
	Morganella morganii	Pseudomonas aeruginosa
	Pseudomonas aeruginosa	Enterococcus faecalis
	Staphylococcus aureus	Enterococcus faecalis
	Citrobacter freundii	Enterococcus faecalis
	Klebsiella pneumoniae	Enterococcus faecalis
digestive tract	Staphylococcus aureus	Pseudomonas aeruginosa
respiratory organs	Pseudomonas aeruginosa	Staphylococcus aureus
	Staphylococcus aureus	Pseudomonas aeruginosa
blood	Streptococcus intermedius	Staphylococcus aureus

Hence, the bacteria "Enterococcus faecalis," "Staphylococcus aureus," "Pseudomonas aeruginosa," and "Bacteroides fragilis" occur in a target episode as a precedent or a subsequent bacteria.

7.2 Sequential Episodes

In this section, we adopt the bacterial culture data from Osaka Prefectural General Medical Center in years from 2000 to 2005 consisting of 35827 records and, in particular, of which number of samples is 30. As the simplest form of *serial episode* [40], we introduce a *sequential episode* of the form $A \to B$ where A and B are event types. Then, we extract a sequential episode $A \to B$ *representing the replacements of bacteria*, where A and B are event types of the occurrences of bacteria, from the new data.

Let $\mathcal{E}$ be the set of *event types*. Then, a pair (e, t) is called an *event*, where $e \in \mathcal{E}$ and t is a natural number which is the *occurrence time* of e. An *event sequence* $\mathcal{S}$ on $\mathcal{E}$ is the set of events, where we deal with bacterial culture data as an event sequence.

A *sequential episode* is of the form $A \to B$, where A and B are event types. We say that a sequential episode $A \to B$ *occurs* in an event sequence $\mathcal{S}$ if both A and B occur in $\mathcal{S}$ and A is precedent to B.

7.2.1 Experiments for bacterial culture data

In this section, by using the following naïve algorithm, we extract the frequent sequential episodes of the from $A \rightarrow B$ *representing the replacements of bacteria* that the occurrence of the bacterium A is followed by one of B from the bacterial culture data.

1. For each of 30 samples, construct an event sequence by fixing the sample and connecting data of every patient with the span of 15 days.

2. Apply the procedure from 3) to 5) to the event sequence constructed by 1).

3. Compute the number of windows in an event sequence and the number of windows that a bacterium occurs.

4. For every pair (A, B) of bacteria, compute the number of windows that a sequential episode $A \rightarrow B$ representing the replacement of bacteria and then compute $supp(A \rightarrow B)$.

5. Output the frequent sequential episodes and their information of occurrences.

In this section, we give the experimental results for extracting frequent sequential episodes representing the replacements of bacteria. First, we summary the number of windows and all of the sequential episodes under the minimum support 0%. Next, we give the frequent sequential episodes for the samples of which number of windows is greater than 10000. Finally, we give the frequent sequential episodes under the minimum support 1%.

The data are provided from Osaka Prefectural General Medical Center in years from 2000 to 2005 consisting of 35827 records. Figure 7.1 describes the number of windows (#windows) and the extracted sequential episodes under the minimum support 0% (#episodes) for each sample sorted by #windows.

In the remainder of this section, we refer both the sample and the number of windows of the sample to the form of "sample (#windows)."

7.2.2 The most 5 frequent sequential episodes

In this subsection, we describe the most frequent 5 sequential episodes extracted from 6 samples, blood, respiratory mucus, pus, stool, urine and sputum, of which number of windows is more than 10000. Here, #patients is the number of different patients included by the sequential episode.

Sputum

The most 5 frequent sequential episodes for the sample of sputum (185745) are described as follows.

replacement of bacteria	frequency (%) #patients
Staphylococcus aureus → Pseudomonas aeruginosa	3010 (1.621%) 179
Pseudomonas aeruginosa → Staphylococcus aureus	2662 (1.433%) 168
yeast → Staphylococcus aureus	2609 (1.405%) 192
Staphylococcus aureus → yeast	2566 (1.381%) 201
yeast → Pseudomonas aeruginosa	1189 (0.640%) 101

Pus

The most 5 frequent sequential episodes for the sample of pus (109103) are described as follows.

replacement of bacteria	frequency (%) #patients
Staphylococcus aureus → Pseudomonas aeruginosa	384 (0.352%) 36
Staphylococcus aureus → Enterococcus faecalis (Group D)	377 (0.346%) 42
Pseudomonas aeruginosa → Staphylococcus aureus	376 (0.345%) 29
Enterococcus faecalis (Group D) → Staphylococcus aureus	373 (0.342%) 38
Staphylococcus aureus → Escherichia coli	252 (0.231%) 21

Respiratory mucus

The most 5 frequent sequential episodes for the sample of respiratory mucus (94689) are described as follows.

replacement of bacteria	frequency (%) #patients
Staphylococcus aureus → yeast	298 (0.315%) 29
Staphylococcus aureus → Pseudomonas aeruginosa	215 (0.227%) 18
Staphylococcus aureus → Staphylococcus coagulase (-)	185 (0.195%) 25
yeast → Staphylococcus aureus	166 (0.175%) 17
Pseudomonas aeruginosa → Staphylococcus aureus	151 (0.159%) 13

Urine

The most 5 frequent sequential episodes for the sample of urine (58699) are described as follows.

replacement of bacteria	frequency (%) #patients
Escherichia coli → yeast	124 (0.211%) 12
Escherichia coli → Enterococcus faecalis (Group D)	119 (0.203%) 12
Staphylococcus aureus → Pseudomonas aeruginosa	115 (0.196%) 17
Enterococcus faecalis (Group D) → Staphylococcus aureus	109 (0.186%) 14
Enterococcus faecalis (Group D) → Pseudomonas aeruginosa	104 (0.177%) 12

Blood

The most 5 frequent sequential episodes for the sample of blood (19117) are described as follows.

replacement of bacteria	frequency (%) #patients
Staphylococcus aureus → Streptococcus constellatus	117 (0.612%) 9
Staphylococcus aureus → Streptococcus intermedius	112 (0.586%) 8
Streptococcus intermedius → Staphylococcus aureus	111 (0.581%) 9
Streptococcus constellatus → Staphylococcus aureus	96 (0.502%) 7
Klebsiella pneumoniae → Bacteroides distasonis	50 (0.262%) 4

Stool

The most 5 frequent sequential episodes for the sample of stool (18820) are described as follows.

replacement of bacteria	frequency (%) #patients
Staphylococcus aureus → yeast	335 (1.780%) 29
yeast → Staphylococcus aureus	206 (1.094%) 17
Pseudomonas aeruginosa → yeast	122 (0.648%) 9
Pseudomonas aeruginosa → Staphylococcus aureus	109 (0.579%) 10
yeast → Pseudomonas aeruginosa	71 (0.377%) 6

Summary

We can observe the replacements of bacteria between Staphylococcus aureus and (1) Streptococcus constellatus or Streptococcus intermedius for the sample of blood and (2) Pseudomonas aeruginosa for the samples except blood. Also the occurrences of bacteria of Enterococcus faecalis (Group D) and Escherichia coli are characterized as the samples of pus and urine. Furthermore, we can observe the replacement from/to yeast for the samples of respiratory mucus, stool, urine and sputum.

On the other hand, by focusing the number of different patients, we can observe the sequential episodes with more than 100 different patients for the sample of sputum, and ones with more than 10 different patients for the 3 samples of respiratory mucus, pus and urine.

7.2.3 The frequent sequential episodes for replacements of bacteria

In this subsection, we describe all of the frequent sequential episodes under the minimum support 1%. We can extract them for the following 10 samples; joint fluid, pleural effusion, sputum, bile, pertoneal fluid, catheter, stool, heart pacer, gastric fluid and secretion (prostate gland, urethra).

Joint fluid

For the sample of joint fluid (425), we can extract 10 frequent sequential episodes under the minimum support 1% as follows.

replacement of bacteria	frequency (%) #patients
Streptococcus agalactiae (Group B) → Streptococcus constellatus	12 (2.824%) 1
Streptococcus constellatus → Staphylococcus aureus	10 (2.353%) 1
Bacteroides fragilis → Streptococcus constellatus	7 (1.647%) 1
Bacteroides fragilis → Prevotella loescheii/denticola	7 (1.647%) 1
Bacteroides uniformis → Prevotella oralis	7 (1.647%) 1
Bacteroides uniformis → Streptococcus constellatus	7 (1.647%) 1
Bacteroides fragilis → Prevotella oralis	7 (1.647%) 1
Bacteroides uniformis → Prevotella loescheii/denticola	7 (1.647%) 1
Prevotella oralis → Prevotella loescheii/denticola	7 (1.647%) 1
Prevotella oralis → Streptococcus constellatus	7 (1.647%) 1

Pleural effusion

For the sample of pleural effusion (2782), we can extract 6 frequent sequential episodes under the minimum support 1% as follows.

replacement of bacteria	frequency (%) #patients
Escherichia coli → Enterococcus faecalis (Group D)	37 (1.330%) 2
Bacteroides distasonis → Escherichia coli	33 (1.186%) 2
Bacteroides distasonis → Enterococcus faecalis (Group D)	33 (1.186%) 2
Escherichia coli → Fusobacterium varium	30 (1.078%) 2
Candida albicans → Enterococcus faecalis (Group D)	29 (1.042%) 2
Enterococcus faecalis (Group D) → Escherichia coli	29 (1.042%) 2

Sputum

For the sample of sputum (185745), we can extract 4 frequent sequential episodes under the minimum support 1% as follows.

replacement of bacteria	frequency (%) #patients
Staphylococcus aureus → Pseudomonas aeruginosa	3010 (1.621%) 179
Pseudomonas aeruginosa → Staphylococcus aureus	2662 (1.433%) 168
yeast → Staphylococcus aureus	2609 (1.405%) 192
Staphylococcus aureus → yeast	2566 (1.381%) 201

Bile

For the sample of bile (3413), we can extract 3 frequent sequential episodes under the minimum support 1% as follows.

replacement of bacteria	frequency (%) #patients
Staphylococcus aureus → Candida albicans	46 (1.348%) 4
Candida albicans → Staphylococcus aureus	40 (1.172%) 3
Escherichia coli → Enterococcus faecalis (Group D)	39 (1.143%) 1

Pertoneal fluid

For the sample of pertoneal fluid (4604), we can extract 3 frequent sequential episodes under the minimum support 1% as follows.

replacement of bacteria	frequency (%) #patients
Enterococcus faecalis (Group D) → Escherichia coli	54 (1.173%) 6
Escherichia coli → Pseudomonas aeruginosa	53 (1.151%) 5
Escherichia coli → Fusobacterium varium	48 (1.043%) 6

Catheter

For the sample of catheter (460), we can extract 2 frequent sequential episodes under the minimum support 1% as follows.

replacement of bacteria	frequency (%) #patients
Enterobacter cloacae → Staphylococcus coagulase (-)	6 (1.304%) 1
Enterococcus faecium (Group D) → Staphylococcus coagulase (-)	6 (1.304%) 1

Stool

For the sample of stool (18820), we can extract 2 frequent sequential episodes under the minimum support 1% as follows.

replacement of bacteria	frequency (%) #patients
Staphylococcus aureus → yeast	335 (1.780%) 29
yeast → Staphylococcus aureus	206 (1.094%) 17

Heart pacer, gastric fluid and secretion (prostate gland, urethra)

For the samples of heart pacer (171), gastric fluid (245) and secretion (prostate gland, urethra) (120), we can extract just one frequent sequential episodes under the minimum support 1% as follows under this order of samples.

replacement of bacteria	frequency (%) #patients
Staphylococcus aureus → Staphylococcus epidermidis	9 (5.263%) 1
Candida glabrata → Pseudomonas aeruginosa	10 (4.082%) 1
Enterobacter cloacae → Staphylococcus coagulase (-)	10 (1.020%) 1

Summary

We can first observe that the smaller number of windows tends to give sequential episodes with larger frequency, except for the sample of sputum and stool, of which number of windows is over 10000. Also we can observe that the occurrences of bacteria of Enterococcus faecalis (Group D) and Escherichia coli are characterized as the samples of pleural effusion, bile and pertoneal fluid. Furthermore, we can observe that the occurrences of bacteria of Streptococcus constellatus and Prevotella oralis/loescheii/denticola, Pseudomonas aeruginosa, and Candida albicans are characterized as the samples of joint fluid, sputum, and bile, respectively.

7.3 Aligned Bipartite Episodes

In this section, we introduce an *aligned bipartite episode between the genera of bacteria*. Then, we extract aligned bipartite episodes from bacterial culture data provided from Osaka Prefectural General Medical Center in years from 1999 to 2007.

An *aligned bipartite episode between the genera of bacteria* is of the form $A \to B$, where A and B are the sets of event types representing the occurrences of the genera of bacteria. In particular, we call an aligned bipartite episode $A \to B$ such that $A \neq B$ *proper*.

The word "aligned" in the aligned bipartite episode comes from the following *earliest occurrence*. Let $\mathcal{S}$ be an event sequence, s and t time stamps, w the width of windows and $A \to B$ an aligned bipartite episode between the genera of bacteria. Then, we say that $A \to B$ *occurs at* (s, t) *in* $\mathcal{S}$ *within* w if $(A, s), (B, t) \in \mathcal{S}$ and $0 < t - s \leq w$. Furthermore, we say that a pair (s, t) is the *earliest occurrence* of $A \to B$ if $A \to B$ occurs at (s, t) in $\mathcal{S}$ within w such that s and t are minimum. In this case, we say that an aligned bipartite episode $A \to B$ occurs *through the earliest occurrence*.

7.3.1 Experiments for bacterial culture data

In this section, by using the following naïve algorithm, we extract the aligned bipartite episodes of the from $A \to B$ representing the replacements of genera of bacteria that every genus in A is precedent to one in B through the earliest occurrence from the bacterial culture data.

1. For each of samples, construct an event sequence by fixing the sample and connecting data of every patient with the span of 15 days. Also collect the genera of bacteria stated as above.

2. Apply the procedure from 3) to 5) to the event sequence constructed by 1).

3. Compute the number of windows in an event sequence and the number of windows that the genera of bacteria occur.

4. For every pair (A, B) of the genera of bacteria, compute the number of windows that an aligned bipartite episode $A \to B$ occurs through the earliest occurrence, and then compute the frequency of it.

5. Output the aligned bipartite episodes and their information of occurrences.

In this section, we adopt the bacterial culture data provided from Osaka Prefectural General Medical Center in years from 1999 to 2007, of which number of records is 164713.

Furthermore, in this section, we use two kinds of data, one is called *type-1* data that a detected bacterium always exists, another is called *type-2* data that contains "none of detected bacteria," where we denote it by $\emptyset$. The number of type-1 and 2 data 54260 and 75137, respectively.

Figure 7.2 describes the number of different patient (in the left column) and the possible maximum frequency of episodes (in the right column) in type-1 and type-2 data for every sample.

7.3.2 The number of extracted aligned bipartite episodes for replacements of bacteria

In the remainder of this section, we denote the number of aligned bipartite episodes between the genera of bacteria by #abe and the number of proper ones by #pabe. Then, Figure 7.3 describes #abe and #pabe for every sample.

7.3.3 The frequent aligned bipartite episodes for replacements of bacteria

In this subsection, we describe the extracted aligned bipartite episodes between genera of bacteria for the samples of which number of episodes is more than 100 (from type-1 data) in Figure 7.3.

Pleural effusion

For the sample of pleural effusion, #abe and #pabe for type-1 data are 139 and 125, and ones for type-2 data are 229 and 215, respectively. Then, the most 5 frequent aligned bipartite episodes between the genera of bacteria for type-1 and type-2 data are described as follows.

freq.	type-1 data		
5	Staphylococcus Streptococcus	→	Staphylococcus
4	Streptococcus	→	Staphylococcus
3	Staphylococcus Streptococcus	→	Staphylococcus
3	Staphylococcus	→	Pseudomonas
2	Staphylococcus	→	Peptostreptococcus Staphylococcus
2	Peptostreptococcus Staphylococcus	→	Staphylococcus

freq.	type-2 data		
25	∅	→	Staphylococcus
18	Staphylococcus	→	∅
9	Streptococcus	→	∅
5	Staphylococcus	→	Staphylococcus Streptococcus
5	∅	→	Staphylococcus Streptococcus

Blood

For the sample of blood, #abe and #pabe for type-1 data are 292 and 270, and ones for type-2 data are 576 and 553, respectively. Then, the most 5 frequent aligned bipartite episodes between the genera of bacteria for type-1 and type-2 data are described as follows.

freq.	type-1 data		
14	Staphylococcus	→	Staphylococcus Streptococcus
12	Staphylococcus Streptococcus	→	Staphylococcus
10	Escherichia	→	Staphylococcus
8	Peptostreptococcus Staphylococcus	→	Staphylococcus
6	Staphylococcus	→	Escherichia
6	Enterococcus	→	Staphylococcus

freq.	type-2 data		
274	Staphylococcus	→	$\emptyset$
261	$\emptyset$	→	Staphylococcus
49	Escherichia	→	$\emptyset$
36	Streptococcus	→	$\emptyset$
36	$\emptyset$	→	Escherichia

Indwelling vessel catheter

For the sample of indwelling vessel catheter, #abe and #pabe for type-1 data are 119 and 109, and ones for type-2 data are 262 and 249, respectively. Then, the most 5 frequent aligned bipartite episodes between the genera of bacteria for type-1 and type-2 data are described as follows.

freq.	type-1 data		
8	Enterococcus	→	Staphylococcus
7	Staphylococcus	→	Enterococcus Staphylococcus
7	Staphylococcus	→	Enterococcus
7	Candida	→	Staphylococcus
5	Staphylococcus Streptococcus	→	Staphylococcus
5	Staphylococcus	→	Pseudomonas
5	Staphylococcus	→	Candida

freq.	type-2 data		
182	$\emptyset$	→	Staphylococcus
162	Staphylococcus	→	$\emptyset$
24	Enterococcus	→	$\emptyset$
22	$\emptyset$	→	Candida
20	$\emptyset$	→	Enterococcus

Respiratory mucus

For the sample of respiratory mucus, #abe and #pabe for type-1 data are 562 and 523, and ones for type-2 data are 881 and 740, respectively. Then, the most 5 frequent aligned bipartite episodes between the genera of bacteria for type-1 and type-2 data are described as follows.

freq.	type-1 data
93	MRSA screening → Staphylococcus
23	Staphylococcus → MRSA screening
21	Staphylococcus → Staphylococcus yeast
18	Staphylococcus → Pseudomonas Staphylococcus
15	Staphylococcus → yeast

freq.	type-2 data
243	Staphylococcus → ∅
162	∅ → Staphylococcus
148	MRSA screening → ∅
98	∅ → ∅ Staphylococcus
74	MRSA screening → Staphylococcus

The reason why the frequency of the aligned bipartite episode "MRSA screening → Staphylococcus" between type-1 data and type-2 data changes is the occurrences of the following episodes containing ∅ in type-2 data; The part of the following episodes are extracted as the episodes from type-1 data.

freq.	type-2 data
26	MRSA screening → ∅ Staphylococcus

Tissue

For the sample of tissue, #abe and #pabe for type-1 data are 133 and 123, and ones for type-2 data are 262 and 249, respectively. Then, the most 5 frequent aligned bipartite episodes between the genera of bacteria for type-1 and type-2 data are described as follows.

freq.	type-1 data
4	Staphylococcus → Enterococcus Staphylococcus
3	Enterococcus → Staphylococcus Staphylococcus
2	Staphylococcus → Staphylococcus Streptococcus
2	Pseudomonas → Staphylococcus Staphylococcus Streptococcus
2	Pseudomonas → Staphylococcus Staphylococcus
2	Escherichia → Staphylococcus Staphylococcus
2	Enterococcus → Staphylococcus Staphylococcus Streptococcus

freq.	type-2 data
18	Staphylococcus → ∅
10	∅ → Staphylococcus
5	Candida → ∅
3	Staphylococcus → Enterococcus Staphylococcus
3	∅ → Candida

The reason why the frequency of the aligned bipartite episode "Staphylococcus → Enterococcus, Staphylococcus" between type-1 data and type-2 data changes is the occurrences of the following episodes containing ∅ in type-2 data; The following episode is extracted as the episode from type-1 data.

freq.	type-2 data
1	Staphylococcus → ∅ Enterococcus Staphylococcus

Drain

For the sample of drain, #abe and #pabe for type-1 data are 284 and 271, and ones for type-2 data are 405 and 390, respectively. Then, the most 5 frequent aligned bipartite

episodes between the genera of bacteria for type-1 and type-2 data are described as follows.

freq.	type-1 data		
3	Pseudomonas	→	Staphylococcus
3	Enterococcus Staphylococcus	→	Staphylococcus
3	Candida Staphylococcus	→	Candida
3	Candida	→	Staphylococcus
2	Staphylococcus	→	Staphylococcus Streptococcus
2	Staphylococcus	→	Pseudomonas
2	Staphylococcus	→	Prevotella Staphylococcus
2	Peptostreptococcus Staphylococcus	→	Staphylococcus
2	Enterococcus	→	Staphylococcus
2	Enterococcus	→	Enterococcus Staphylococcus
2	Enterococcus	→	Candida
2	Enterobacter	→	Candida Enterobacter
2	Candida Staphylococcus Streptococcus	→	Staphylococcus
2	Candida Staphylococcus	→	Staphylococcus
2	Candida	→	Candida Enterococcus

freq.	type-2 data		
54	$\emptyset$	→	Staphylococcus
31	Staphylococcus	→	$\emptyset$
6	$\emptyset$ Staphylococcus	→	$\emptyset$
6	$\emptyset$	→	Streptococcus
6	$\emptyset$	→	Enterococcus

Pus

For the sample of pus, #abe and #pabe for type-1 data are 3214 and 3138, and ones for type-2 data are 3917 and 3841, respectively. Then, the most 5 frequent aligned bipartite episodes between the genera of bacteria for type-1 and type-2 data are described as follows.

freq.	type-1 data		
33	Staphylococcus	→	Pseudomonas Staphylococcus
33	Pseudomonas Staphylococcus	→	Staphylococcus
32	Enterococcus	→	Staphylococcus
31	Staphylococcus	→	Pseudomonas
27	Pseudomonas	→	Staphylococcus

freq.	type-2 data		
220	Staphylococcus	→	∅
162	∅	→	Staphylococcus
52	Streptococcus	→	∅
40	∅	→	Streptococcus
35	Pseudomonas	→	∅

Stool

For the sample of stool #abe and #pabe for type-1 data are 163 and 148, and ones for type-2 data are 310 and 293, respectively. Then, the most 5 frequent aligned bipartite episodes between the genera of bacteria for type-1 and type-2 data are described as follows.

freq.	type-1 data
19	Staphylococcus → yeast
14	Staphylococcus → yeast yeast
13	Staphylococcus → Staphylococcus yeast
12	yeast → Staphylococcus
11	Staphylococcus → Staphylococcus yeast

freq.	type-2 data
138	Staphylococcus → ∅
95	∅ → Staphylococcus
47	yeast → ∅
42	Salmonella → ∅
38	∅ → yeast

Bile

For the sample of bile, #abe and #pabe for type-1 data are 193 and 180, and ones for type-2 data are 215 and 201, respectively. Then, the most frequent aligned bipartite episodes between the genera of bacteria for type-1 and type-2 data are described as follows. Here, the frequency of the other aligned bipartite episode is just 1.

freq.	type-1 data
2	Escherichia → Enterococcus Escherichia

freq.	type-2 data
2	Klebsiella → ∅
2	Escherichia → Enterococcus Escherichia
2	∅ → Enterococcus

Urine

For the sample of urine, #abe and #pabe for type-1 data are 1163 and 1109, and ones for type-2 data are 1499 and 1444, respectively. Then, the most 5 frequent

aligned bipartite episodes between the genera of bacteria for type-1 and type-2 data are described as follows.

freq.	type-1 data
20	yeast → Candida
13	Candida → yeast
13	Candida → Escherichia
12	Staphylococcus → Pseudomonas
10	Escherichia → yeast

freq.	type-2 data
91	∅ → Escherichia
83	∅ → Candida
80	Escherichia → ∅
76	Staphylococcus → ∅
71	∅ → Staphylococcus

Peritoneal fluid

For the sample of peritoneal fluid, #abe and #pabe for type-1 data are 203 and 195, and ones for type-2 data are 283 and 274, respectively. Then, the most 5 frequent aligned bipartite episodes between the genera of bacteria for type-1 and type-2 data are described as follows. Here, the frequency of the other aligned bipartite episode for type-1 data is just 1.

freq.	type-1 data
2	Candida → Streptococcus

freq.	type-2 data
15	∅ → Staphylococcus
9	Staphylococcus → ∅
5	Candida → ∅
5	∅ → Candida
4	Streptococcus → ∅

Sputum

For the sample of sputum, #abe and #pabe for type-1 data are 7636 and 7474, and ones for type-2 data are 9543 and 9375, respectively. Then, the most 5 frequent

aligned bipartite episodes between the genera of bacteria for type-1 and type-2 data are described as follows.

freq.	type-1 data		
108	Staphylococcus	→	Pseudomonas Staphylococcus
103	Staphylococcus yeast	→	Staphylococcus
92	Staphylococcus	→	Staphylococcus yeast
92	Staphylococcus	→	Pseudomonas
90	Pseudomonas	→	Staphylococcus

freq.	type-2 data		
396	$\emptyset$	→	Staphylococcus
325	Staphylococcus	→	$\emptyset$
169	$\emptyset$	→	Pseudomonas
136	Pseudomonas	→	$\emptyset$
128	$\emptyset$	→	Pseudomonas Staphylococcus

Summary

In type-2 data, both "Staphylococcus → $\emptyset$" and "$\emptyset$ →Staphylococcus" are the most 2 frequent episodes for the samples except bile and urine. On the other hand, "Klebsiella → $\emptyset$," "Escherichia → Enterococcus, Escherichia" and "$\emptyset$ → Enterococcus" are the most frequent episode for the sample of bile, and "$\emptyset$ → Escherichia," "$\emptyset$ → Candida" and "Escherichia → $\emptyset$" are the most 3 frequent episodes for the sample of urine.

By pay our attention to the genera of bacteria, the genus of Pseudomonas occurs in the episodes for the samples except blood, stool, bile and peritoneal fluid. Also the genera of (1) Streptococcus, (2) Enterococcus, (3) Escherichia and (4) Peptostreptococcus occur in the episodes for the samples of (1) pleural effusion, blood, indwelling vessel catheter, tissue and drain, (2) indwelling vessel catheter, tissue, drain, pus and bile, (3) blood, tissue, bile and urine, and (4) pleural effusion and blood, respectively.

On the other hand, the yeast occurs in the episodes for the samples of stool, urine

and sputum. In particular, for the sample of stool, the most 5 frequent episodes for type-1 data always contain the yeast.

Also Candida occurs in the episodes for the samples of indwelling vessel catheter, drain, urine and peritoneal fluid. In particular, for the sample of urine, both "yeast $\rightarrow$ Candida" and "Candida $\rightarrow$ yeast" occur uniquely.

Finally, the episodes containing MRSA screening and of "Salmonella$\rightarrow \emptyset$" occur uniquely for the sample of respiratory mucus and stool, respectively.

7.4 Chapter Summary

In this chapter, by applying episode mining algorithms to bacterial culture data provided from Osaka Prefecture General Medical Center, we have extracted sectorial episodes, sequential episodes, and aligned bipartite episodes representing changes for drug resistance and replacements of bacteria.

sample	#windows	#episodes
sputum	185745	1627
pus	109103	1700
respiratory mucus	94689	273
urine	58699	265
blood	19117	169
stool	18820	85
indwelling vascular catheter	9772	70
tissue	7111	87
drain	5993	390
pertoneal fluid	4604	483
internal organ	4240	0
bile	3413	152
pleural effusion	2782	164
puncture fluid	1893	106
perfusate	1654	7
spinal fluid	1033	1
secretion (prostate gland, urethra)	980	2
amnion liquid	840	0
catheter	460	10
joint fluid	425	10
gastric fluid	245	2
heart pacer	171	1
secretion (cornea, conjunctiva)	120	0
intestinal fluid	120	0
attachment	77	0
fluid of bone marrow	57	0
lymph node	45	0
duodenal fluid	45	0
breast milk	30	0
nasal cavity	0	–

Figure 7.1: The number of windows and extracted sequential episodes under the minimum support 0% for each of 30 samples.

sample	type-1		type-2	
catheter	44	6	56	8
pleural effusion	163	58	1091	302
blood	1328	322	5995	2305
indwelling vessel catheter	729	149	3115	1514
respiratory mucus	3796	631	7095	1328
internal organ	82	2	147	6
tissue	455	76	727	121
drain	621	148	1725	433
pus	3973	1000	6953	1663
heart pacer	15	1	239	13
lymph node	3	0	19	1
gastric fluid	27	3	59	5
joint fluid	42	12	320	58
breast milk	2	0	2	0
duodenal fluid	5	0	5	0
spinal fluid	78	15	1096	186
puncture fluid	123	25	295	55
stool	1140	248	3934	937
bile	161	56	233	71
intestinal fluid	12	0	20	1
urine	2174	620	4463	1407
attachment	6	2	10	2
peritoneal fluid	415	82	904	202
secretion	78	2	94	3
amnion liquid	113	0	155	0
sputum	4342	1996	7283	3300
perfusate	20	12	38	24
nasal cavity	212	10	286	15

Figure 7.2: The number of different patients and the possible maximum frequency of episodes for every sample.

	type-1		type-2	
sample	#abe	#pabe	#abe	#pabe
catheter	11	9	12	9
pleural effusion	139	125	229	215
blood	292	270	576	553
indwelling vessel catheter	119	109	262	249
respiratory mucus	562	523	881	740
internal organ	2	1	5	3
tissue	133	123	183	171
drain	284	271	405	390
pus	3214	3138	3917	3841
heart pacer	1	0	4	2
lymph node	0	0	1	0
gastric fluid	3	3	5	4
joint fluid	12	7	32	27
spinal fluid	8	5	31	27
puncture fluid	52	46	78	71
stool	163	148	310	293
bile	193	180	215	201
intestinal fluid	0	0	1	1
urine	1163	1109	1499	1444
attachment	2	1	2	1
peritoneal fluid	203	195	283	274
secretion	4	4	8	7
sputum	7636	7474	9543	9375
perfusate	30	24	61	54
nasal cavity	10	8	15	12

Figure 7.3: The number of aligned bipartite and proper aligned bipartite episodes between the genera of bacteria.

Chapter 8

Conclusion

In this thesis, we have studied frequent episode mining problem. To reduce memory consumption, we have newly designed the *episode-growth* algorithms to enumerate frequent diamond episodes and more general episodes in polynomial time per an output and polynomial space. For the algorithms, we have adopted the depth-first search instead of the level-wise search.

8.1 Summary of the Results

In Chapter 3, we have studied the problem of mining *frequent diamond episodes efficiently* from an input event sequence with sliding a window. Then, we have designed a polynomial-delay and polynomial-space algorithm PolyFreqDmd that finds all of the frequent diamond episodes without duplicates from an event sequence in $O(|\Sigma|^2 n)$ time per an episode and in $O(|\Sigma| + n)$ space, where Σ and n are an alphabet and the length of the event sequence, respectively. Finally, we have given experimental results on artificial and real-world event sequences with varying several mining parameters to evaluate the efficiency of the algorithm.

In Chapter 4, first we have introduced a *bipartite episode*. Then, we have presented 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 $\mathcal{S}$, and n is the length of S. Finally, we have given experimental results on artificial and real sequences to evaluate the efficiency of the algorithm.

In Chapter 5, we have introduced the class of *k-partite episodes*. Then, we have presented 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 have shown that these two algorithms run in polynomial time per an output and polynomial space in total input size.

In Chapter 6, we have given a simple characterization of episodes in episode mining that are constructible from just information for occurrences of serial episodes, called *serially constructible* episodes. First, we have formulated an episode as an *acyclic transitive labeled digraph* of which label is an event type in episode mining. Next, we have introduced a *parallel-free* episode that always has an arc between vertices with the same label. Also we have formulated a *serially constructible* episode as an episode embedded into every parallel-free episode containing all of the serial episodes occurring in it. Then, we have shown that an episode is parallel-free if and only if it is serially constructible.

In Chapter 7, we have applied episode mining algorithms to the bacterial culture data provided from Osaka Prefecture General Medical Center. Then, we have extracted the episodes as the time-related rules representing replacements of bacteria and changes for drug resistance as the factors of hospital-acquired infection.

8.2 Future Researches

First, it is a future work to give more exhaustive experiments on the proposed algorithms. Secondly, more detailed theoretical analysis of the proposed algorithms are required. For example, Lemma 35 in Chapter 5 says that the time complexity of COUNTBYLIST is $O(N^2 w)$ time. We conjecture that this can be improved to $O(Nw)$

time by amortized analysis on the execution of the subprecedure IncLeftMost-Tail over the whole execution. Thirdly, it is a possible future problem to introduce the class of *closed patterns* [4, 5, 48, 69, 62, 11] designed for a class of episodes and develop efficient closed episode mining algorithms. It is also a future work to extend our algorithms to parallel-free episodes for general fragments of directed acyclic graphs [40, 48]. Finally, it is a future work to apply our episode mining algorithms in Chapter 3, Chapter 4, and Chapter 5 to the bacterial culture data.

Bibliography

[1] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the ACM SIGMOD Conference on Management of Data (SIGMOD1993)*, pages 207–216. ACM, 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB1994)*, pages 487–499. VLDB, 1994.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering*, pages 3–14. IEEE, 1995.

[4] H. Arimura. Efficient algorithms for mining frequent and closed patterns from semi-structured data. In *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD2008), Lecture Notes in Artificial Intelligence 5012*, pages 2–13. Springer-Verlag, 2008.

[5] H. Arimura and T. Uno. A polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. In *Proceedings of the 16th Annual International Symposium on Algorithms and Computation (ISAAC2005), Lecture Notes in Computer Science 3827*, pages 724–737. Springer-Verlag, 2005.

[6] D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65:21–46, 1996.

[7] R. Bathoorn, M. C. M. Welten, M. Richardson, A. Siebes, and F. J. Verbeek. Frequent episode mining to support pattern analysis in developmental biology. In *Proceedings of the 5th IAPR International Conference on Pattern Recognition in Bioinformatics (PRIB2010)*, pages 253–263. IAPR, 2010.

[8] M. Baumgarten, A. G. Büchner, and J. G. Hughes. Tree growth based episode mining without candidate generation. In *IC-AI Proceedings of the 2003 International Conference on Artificial Intelligence (ICAI2003)*, pages 108–114. CSREA Press, 2003.

[9] C. Bettini, S. Wang, S. Jajodia, and J.-L. Lin. Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):222–237, 1998.

[10] E. Boros, V. Gurvich, L. Khachiyan, and K. Makino. On the complexity of generating maximal frequent and minimal infrequent sets. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science (STACS2002), Lecture Notes in Computer Science 2285*, pages 133–141. Springer-Verlag, 2002.

[11] D. Burdick, M. Calimlim, and J. Gehrke. MAFIA: A maximal frequent itemset algorithm for transactional databases. In *Proceedings of the 17th IEEE International Conference on Data Engineering (ICDE2001)*, pages 443–452. IEEE, 2001.

[12] C. Chen, C. X. Lin, X. Yan, and J. Han. On effective presentation of graph patterns: A structural representative approach. In *Proceeding of the 17th ACM Conference on Information and Knowledge Management (CIKM2008)*, pages 299–308. ACM, 2008.

[13] C. Chen, X. Yan, P. S. Yu, J. Han, D. Zhang, and X. Gu. Towards graph containment search and indexing. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB2007)*, pages 926–937. VLDB, 2007.

[14] C. Chen, X. Yan, F. Zhu, J. Han, and P. S. Yu. Graph OLAP: Towards online analytical processing on graphs. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM2008)*, pages 103–112. IEEE, 2008.

[15] J. Gao, W. Fan, and J. Han. On appropriate assumptions to mine data streams: Analysis and practice. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM2007)*, pages 143–152. IEEE, 2007.

[16] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the ACM SIGMOD Conference on Management of Data (SIGMOD2000)*, pages 1–12. ACM, 2000.

[17] S. Hirano and S. Tsumoto. Cluster analysis of temporal trajectories of hospital laboratory examinations. In *Proceedings of the Society of Photo-Optical Instrumentation Engineers 6570*, pages 150–158. SPIE, 2007.

[18] S. Hirano and S. Tsumoto. Temporal data mining in hospital information systems: Analysis of clinical courses of chronic hepatitis. *International Journal of Intelligent Computing in Medical Sciences and Image Processing*, 1:9–20, 2007.

[19] S. Hirano and S. Tsumoto. Trajectory analysis of laboratory tests as medical complex data mining. In *Proceedings of the 3rd ECML/PKDD International Conference on Mining Complex Data (MCD2007), Lecture Notes in Computer Science 4944*, pages 27–41. Springer-Verlag, 2008.

[20] K. Hirata, M. Harao, M. Wada, S. Ozaki, S. Yokoyama, and K. Matsuoka. Attribute selection measures with possibility and their application to classifying MRSA from MSSA. In *Complex Medical Engineering*, pages 143–151. Springer-Verlag, 2007.

[21] K. Hirata, Y. Shima, M. Harao, S. Yokoyama, K. Matsuoka, and T. Izumi. Disjunctive rules extracted from MRSA data with verification. In *Proceedings of*

the 1st International Conference on Complex Medical Engineering (CME2005), pages 326–330. IEEE/ICME, 2005.

[22] Y. Ikenaga, K. Hirata, M. Harao, S. Yokoyama, and K. Matusoka. Risk management system for hospital-acquired infection based on bacterial culture database. In *Proceedings of the 1st International Conference on Complex Medical Engineering (CME2005)*, pages 423–427. IEEE/ICME, 2005.

[23] A. Inokuchi and T. Washio. A fast method to mine frequent subsequences from graph sequence data. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM2008)*, pages 303–312. IEEE, 2008.

[24] A. Inokuchi and T. Washio. Mining frequent graph sequence patterns induced by vertices. In *Proceedings of 10th SIAM International Conference on Data Mining (SDM2010)*, pages 466–477. SIAM, 2010.

[25] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. In *Proceedings of the 4th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD2000), Lecture Notes in Computer Science 1910*, pages 13–23. Springer-Verlag, 2000.

[26] A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs: Mining graph data. *Machine Learning*, 50:321–354, 2003.

[27] H. Kashima and A. Inokuchi. Kernels for graph classification. In *Proceedings of the 1st ICDM Workshop on Active Mining (AM2002)*, pages 31–36. IEEE, 2002.

[28] T. Katoh, H. Arimura, and K. Hirata. An efficient depth-first search algorithm for extracting frequent diamond episodes from event sequences. *IPSJ Transactions on Databases*, 2:1–12, 2009.

[29] T. Katoh, H. Arimura, and K. Hirata. Mining frequent bipartite episodes from event sequences. In *Proceedings of the 12th International Conference on Dis-*

covery Science (DS2009), Lecture Notes in Artificial Intelligence 5808, pages 136–151. Springer-Verlag, 2009.

[30] T. Katoh, H. Arimura, and K. Hirata. A polynomial-delay polynomial-space algorithm for extracting frequent diamond episodes from event sequences. In *Proceedings of the 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD2009), Lecture Notes in Artificial Intelligence 5476*, pages 172–183. Springer-Verlag, 2009.

[31] T. Katoh, H. Arimura, and K. Hirata. Mining frequent k-partite episodes from event sequences. In *New Frontiers in Artificial Intelligence, Lecture Notes in Artificial Intelligence 6284*, pages 331–344. Springer-Verlag, 2010.

[32] T. Katoh and K. Hirata. Mining frequent elliptic episodes from event sequences. In *Proceedings of the 5th Workshop on Learning with Logic and Logics for Learning (LLLL2007)*, pages 46–52. JSAI, 2007.

[33] T. Katoh and K. Hirata. A simple characterization on serially constructible episodes. In *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD2008), Lecture Notes in Artificial Intelligence 5012*, pages 600–607. Springer-Verlag, 2008.

[34] T. Katoh, K. Hirata, H. Arimura, S. Yokoyama, and K. Matsuoka. Extracting sequential episodes representing replacements of bacteria from bacterial culture data. In *Proceedings of the 2009 IEEE/ICME International Conference on Complex Medical Engineering (CME2009)*, pages 1–4. IEEE/ICME, 2009.

[35] T. Katoh, K. Hirata, H. Arimura, S. Yokoyama, and K. Matsuoka. Aligned bipartite episodes between the genera of bacteria. In *Proceedings of the 2010 IEEE/ICME International Conference on Complex Medical Engineering (CME2010)*, pages 193–197. IEEE/ICME, 2010.

[36] T. Katoh, K. Hirata, and M. Harao. Mining sectorial episodes from event sequences. In *Proceedings of the 9th International Conference on Discovery Science (DS2006), Lecture Notes in Artificial Intelligence 4265*, pages 137–145. Springer-Verlag, 2006.

[37] T. Katoh, K. Hirata, and M. Harao. Mining frequent diamond episodes from event sequences. In *Proceedings of the 4th International Conference on Modeling Decisions for Artificial Intelligence (MDAI2007), Lecture Notes in Artificial Intelligence 4617*, pages 477–488. Springer-Verlag, 2007.

[38] T. Katoh, K. Hirata, M. Harao, S. Yokoyama, and K. Matsuoka. Extraction of sectorial episodes representing changes for drug resistant and replacements of bacteria. In *Proceedings of the 2007 IEEE/ICME International Conference on Complex Medical Engineering (CME2007)*, pages 304–309. IEEE/ICME, 2007.

[39] H. Mannila and H. Toivonen. Multiple uses of frequent sets and condensed representations. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD1996)*, pages 189–194. AAAI, 1996.

[40] H. Mannila, H. Toivonen, and A. I. Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289, 1997.

[41] K. Matsuoka, S. Yokoyama, K. Watanabe, and S. Tsumoto. Mining rules for risk factors on blood stream infection in hospital information system. In *Proceedings of the 2007 IEEE International Conference on Bioinformatics and Biomedicine (BIBM2007)*, pages 181–187. IEEE, 2007.

[42] K. Matsuoka, S. Yokoyama, K. Watanabe, and S. Tsumoto. Data mining analysis of relationship between blood stream infection and clinical background in patients undergoing lactobacillus therapy. In *New Frontiers in Artificial Intelligence, Lecture Notes in Artificial Intelligence 4914*, pages 277–288. Springer-Verlag, 2008.

[43] L. Mendes, B. Ding, and J. Han. Stream sequential pattern mining with precise error bounds. In *Proceedings of the 8th IEEE International Conference on Data Mining (ICDM2008)*, pages 941–946. IEEE, 2008.

[44] M. Ohsaki, H. Abe, S. Tsumoto, H. Yokoi, and T. Yamaguchi. Valuation of rule interestingness measures in medical knowledge discovery in databases. *Artificial Intelligence in Medicine*, 41:177–196, 2007.

[45] H. Ohtani, T. Kida, T. Uno, and H. Arimura. Efficient serial episode mining with minimal occurrences. In *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication (ICUIMC2009)*, pages 457–464. ACM, 2009.

[46] J. Pei, J. Han, and R. Mao. CLOSET: An efficient algorithm for mining frequent closed itemsets. In *Proceedings of the ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD2000)*, pages 21–30. ACM, 2000.

[47] J. Pei, J. Han, B. Mortazavi-Asi, and J. Wang. Mining sequential patterns by pattern-growth: The PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1–17, 2004.

[48] J. Pei, H. Wang, J. Liu, K. Wang, J. Wang, and P. S. Yu. Discovering frequent closed partial orders from strings. *IEEE Transactions on Knowledge and Data Engineering*, 18(11):1467–1481, 2006.

[49] B. Possas, N. Ziviani, W. Meira Jr., and B. A. Ribeiro-Neto. Set-based model: a new approach for information retrieval. In *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR2002)*, pages 230–237. ACM, 2002.

[50] Y. Shima, K. Hirata, and M. Harao. Extraction of frequent few-overlapped monotone DNF formulas with depth-first prunning. In *Proceedings of the 9th Pacific-*

Asia Conference on Knowledge Discovery and Data Mining (PAKDD2005), Lecture Notes in Artificial Intelligence 3518, pages 50–60. Springer-Verlag, 2005.

[51] Y. Shima, K. Hirata, M. Harao, S. Yokoyama, K. Matsuoka, and T. Izumi. Extracting disjunctive closed rules from MRSA data. In *Proceedings of the 1st International Conference on Complex Medical Engineering (CME2005)*, pages 321–325. IEEE/ICME, 2005.

[52] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology (EDBT1996), Lecture Notes in Computer Science 1057*, pages 3–17. Springer-Verlag, 1996.

[53] S. Tsumoto. Guide to the bacteriological examination data set. In *Proceedings of the International Workshop of KDD Challenge on Real-World Data (KDD Challenge 2000)*, pages 8–12. ACM, 2000.

[54] S. Tsumoto. Mining diagnostic taxonomy and diagnostic rules for multi-stage medical diagnosis from hospital clinical data. In *Proceedings of the 2007 IEEE International Conference on Granular Computing (GrC2007)*, pages 611–616. IEEE, 2007.

[55] S. Tsumoto. Mining risk information in hospital information system as risk mining. In *Proceedings of the 2007 IEEE/ICME International Conference on Complex Medical Engineering (CME2007)*, pages 1948–1957. IEEE/ICME, 2007.

[56] S. Tsumoto, K. Matsuoka, and S. Yokoyama. Risk mining for infection control. In *Communications and Discoveries from Multidisciplinary Data*, pages 283–297. Springer-Verlag, 2008.

[57] S. Tsumoto and Y. Tsumoto. Data mining-based hospital management. *Journal of Chongqing University of Post and Telecommunications*, 20:309–323, 2008.

[58] S. Tsumoto, Y. Tsumoto, K. Matsuoka, and S. Yokoyama. Risk mining in medicine: Application of data mining to medical risk management. In *Web Intelligence Meets Brain Informatics, Lecture Notes in Computer Science 4845*, pages 471–493. Springer-Verlag, 2006.

[59] T. Uno. Fast algorithms for enumerating cliques in huge graphs. Technical report, Institute of Electronics, Information and Communication Engineers, 2003.

[60] T. Uno. Two general methods to reduce delay and change of enumeration algorithms. Technical report, National Institute of Informatics (NII), 2003.

[61] J. Wang and J. Han. BIDE: Efficient mining of frequent closed sequences. In *Proceedings of the 20th International Conference on Data Engineering (ICDE2004)*, pages 79–90. IEEE, 2004.

[62] J. Wang, J. Han, Y. Lu, and P. Tzvetkov. TFP: An efficient algorithm for mining top-k frequent closed itemsets. *IEEE Transactions on Knowledge and Data Engineering*, 17:652–664, 2005.

[63] T. Washio, A. Fujimoto, and H. Motoda. Extension of basket analysis and quantitative association rule mining. In *Proceedings of the Joint Workshop of Vietnamese Society of AI, SIGKBS-JSAI, ICS-IPSJ and IEICE-SIGAI on Active Mining*, pages 117–122. JSAI/IPJS/IEICE, 2004.

[64] J. L. Wu, K. Ito, S. Tobimatsu, T. Nishida, and H. Fukuyama, editors. *Complex Medical Engineering*. Springer-Verlag, 2007.

[65] X. Yan, J. Han, and R. Afshar. CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of the 3rd SIAM International Conference on Data Mining (SDM2003)*, pages 166–177. SIAM, 2003.

[66] X. Yan, F. Zhu, P. S. Yu, and J. Han. Feature-based substructure similarity search. *ACM Transactions on Database Systems*, 31:1418–1453, 2006.

[67] S. Yokoyama, K. Matsuoka, S. Tsumoto, M. Harao, T. Yamakawa, K. Sugahara, C. Nakahama, S. Ichiyama, and K.Watanabe. Study on the association between the patients' clinical background and the anaerobes by data mining in infectious diseases database. *Biomedical Soft Computing and Human Sciences*, 7(1):69–75, 2001.

[68] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12:372–390, 2000.

[69] M. J. Zaki and C.-J. Hsiao. CHARM: An efficient algorithm for closed itemset mining. In *Proceedings of the 2nd SIAM International Conference on Data Mining (SMD2002)*, pages 457–478. SIAM, 2002.

[70] W. Zhou, H. Liu, and H. Cheng. Mining closed episodes from event sequences efficiently. In *Proceedings of the 14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD2010), Lecture Notes in Artificial Intelligence 6118*, pages 310–318. Springer-Verlag, 2010.