IPSJ Transactions on Databases Vol. 2 No. 4 1–12 (Dec. 2009)

Regular Paper

An Efficient Depth-first Search Algorithm for Extracting Frequent Diamond Episodes from Event Sequences

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

In this paper, 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 $O(|\Sigma|^2\ell)$ time per an episode and in $O(|\Sigma| + \ell)$ space, where Σ and ℓ 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.

1. Introduction

It is one of the important tasks in data mining to discover frequent patterns from time-related data. For such a task, Mannila, et al.[7] have introduced the *episode mining* 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., Refs. 3), 9)). 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.

†1 Graduate School of Information Science and Technology, Hokkaido University
†2 Department of Artificial Intelligence, Kyushu Institute of Technology

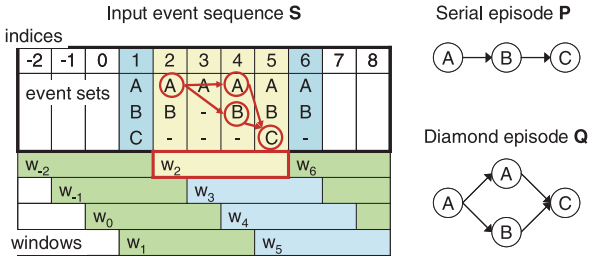

Fig. 1 (Left) An input sequence $\mathcal{S} = (S_1, \ldots, S_6)$ of length $\ell = 6$ over $\Sigma = \{A, B, C\}$ and their k-windows. (Right) Serial episode $P = A \mapsto B \mapsto C$ and a diamond episode $Q = A \mapsto \{A, B\} \mapsto C$. In the sequence $\mathcal{S}$, we indicate an occurrence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

Mannila, et al.[7] 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. Unfortunately, their algorithm is general but inefficient. Then, several efficient algorithms[5),6),8)] have been developed by introducing the specific forms of episodes for every target area.

As such specific forms of episodes, Katoh, et al. have introduced *diamond episodes*[6] and *elliptic episodes*[5]. In **Fig. 1**, we show examples of an input event sequence, a serial episode, and a diamond episode over an alphabet. Both episodes have the special event types, 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[5),6)], they have succeeded in finding frequent diamond episodes and frequent elliptic ones 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 algorithms designed by Katoh, et al.[5),6)] 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 the

frequent itemset mining algorithm AprioriTid [1] 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 (in particular, appropriately applied to the bacterial culture data), it is difficult to give a theoretical guarantee of the efficiency to the level-wise algorithms from the view of enumeration. In this paper, 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.

From experiments, we can also evaluate that the implementation of the algorithm 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 paper is organized as follows. In Section 2, we introduce diamond episodes and other notions necessary to the later discussion. In Section 3, we present the algorithm PolyFreqDmd and show its correctness and the computational complexity. In Section 4, we give a theoretical analysis to compare the algorithm PolyFreqDmd and the level-wise algorithm in Ref. 6). In Section 5, we give some experimental results from randomly generated and real-world event sequences to evaluate the practical performance of the algorithms. In Section 6, we conclude this paper and discuss the future works.

This paper is an extended version of the talk presented at the 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD2009), Bangkok, Thailand, 2009 [10].

2. Diamond Episode

In this section, we introduce diamond episodes and the related notions necessary to later discussion. We denote the sets of all integers and all natural numbers by $\mathbf{Z}$ and $\mathbf{N}$, respectively. For a set S, we denote the cardinality of S by $|S|$. Let $\Sigma = \{1, \ldots, s\}$ $(s \geq 1)$ be a finite alphabet with the total order $\leq$ over $\mathbf{N}$. Each element $e \in \Sigma$ is called an *event* ⋆1. An *input event sequence* (*input sequence*, for short) $\mathcal{S}$ on Σ is a finite sequence $\langle S_1, \ldots, S_\ell \rangle \in (2^\Sigma)^*$ of sets of events $(\ell \geq 0)$, where $S_i \subseteq \Sigma$ is called the i-th *event set* for every $1 \leq i \leq \ell$. Then, we call ℓ the *length* of $\mathcal{S}$ and denote it by $|\mathcal{S}|$, and define the *total size* of $\mathcal{S}$ by $||\mathcal{S}|| = \sum_{i=1}^{\ell} |S_i|$. Clearly, $||\mathcal{S}|| = O(|\Sigma|\ell)$, but the converse $|\Sigma|\ell = O(||\mathcal{S}||)$ is not always true.

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 1 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

⋆1 Mannila, et al. [7] 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*.

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. 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 $\ell = 6$ 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 [7]. 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 2 (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.

For an episode P, we define the *occurrence list* for P in $\mathcal{S}$ by $\mathbf{W}_{\mathcal{S},k}(P) = \{ -k + 1 \leq i \leq \ell \,|\, P \sqsubseteq W_i \}$, the set of the occurrences of P in an input $\mathcal{S}$. We may call the element $i \in \mathbf{W}_{\mathcal{S},k}(P)$ a label or a position. Note that we allow negative integers for labels. If $i \in \mathbf{W}_{\mathcal{S},k}(P)$, then we say that an episode P *occurs in* $\mathcal{S}$ at position i or at the i-th window.

Example 2 Consider an input event sequence $\mathcal{S} = (\{A, B, C\}, \{A, B\}, \{A\}, \{A, B\}, \{A, B\}, \{A, B, C\}, \{A, B\})$ in Fig. 1 again. Then, if the window width k is 4, $\mathcal{S}$ has nine 4-windows from W_{-2} to W_6, i.e., $\mathbf{W}_{\mathcal{S},4} = \{ W_i \,|\, -2 \leq i \leq 6 \}$. 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 $\mathbf{W}_{\mathcal{S},k}$. 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 $\mathbf{W}_{\mathcal{S},k}$. 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 thresh-

old 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 3 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 paper, we design an algorithm for efficiently solving the frequent diamond episode mining problem in the sense of enumeration algorithms [2),4)]. Let N be the total input size and M the number of all solutions. An enumeration algorithm $\mathcal{A}$ is of *output-polynomial time* if $\mathcal{A}$ finds all solutions $S \in \mathcal{S}$ in total polynomial time both in N and M. Also $\mathcal{A}$ is of *polynomial delay* if the *delay*, which is the maximum computation time between two consecutive outputs, is bounded by a polynomial in N alone. It is obvious that if $\mathcal{A}$ is of polynomial delay, then it is of output-polynomial.

3. A Polynomial-Delay and Polynomial-Space 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.

In **Fig. 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 $\mathbf{W}(Q_{ab})$.

```
algorithm PolyFreqDmd(S, k, Σ, σ)
input: input event sequence S ∈ (2^Σ)* of length ℓ, window width k > 0,
alphabet of events Σ, the minimum frequency 1 ≤ σ ≤ ℓ + k;
output: frequent diamond episodes; {
1  Σ0 := the set of all events apparing no less than σ windows (Σ0 ⊆ Σ);
2  foreach ( a ∈ Σ0 ) do
3      output a;
4      foreach ( b ∈ Σ0 ) do
5          Q0 := (a↦∅↦b); //2-serial episode
6          W0 := the occurrence list W_{S,k}(Q0) for Q0;
7          FreqDmdRec(Q0, W0, S, k, Σ0, σ);
8      end for
}
procedure FreqDmdRec(Q = (a↦E↦b), W, S, k, Σ, σ)
output: all frequent diamond episodes of the form a↦F↦b; {
1  if ( |W| ≥ σ ) then
2      output Q; // (*) output Q if the depth is odd (alternating output);
3      foreach ( e ∈ Σ such that e > max(E) ) ) do
4          R := a↦(E ∪ {e})↦b;
5          U := UpdateDmdOcc(Q, e, W, k, S);    // Computing U = W_{S,k}(R)
6          FreqDmdRec(R, U, S, k, Σ, σ);
7      end for
8      // (*) output Q if the depth is even (alternating output);
9  end if
}
```

Fig. 2 The main algorithm PolyFreqDmd and a recursive subprocedure FreqDmdRec for mining frequent diamond episodes in a sequence.

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**, 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

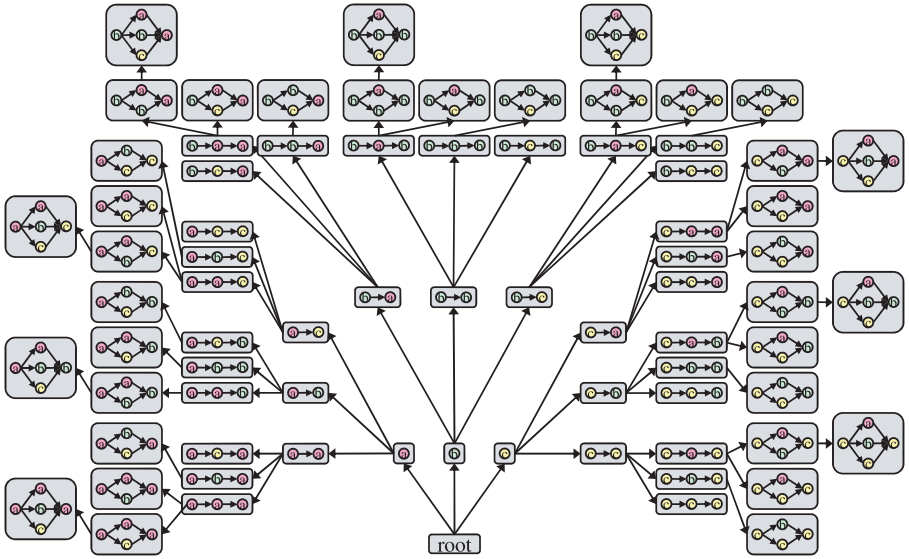

Fig. 3 The parent-child relationships on the alphabet $\Sigma = \{a, b, c\}$.

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. □

In the recursive procedure FreqDmdRec in Fig. 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 $O(||W_i||)$. □

```
algorithm UpdateDmdOcc(Q, e, W, k, S)
input: a parent diamond episode Q = (a↦E↦b), a new event e > max(E),
the old occurrence list W for Q, k ≤ 1, an input sequence S;
output: the new occurrence list U for the child R = (a↦E∪{e}↦b); {
1  V := FindSerialOcc(P = (a↦e↦b), k, S);
2  return U := W ∩ V;
}
procedure FindSerialOcc(P = (a↦e↦b), k, S) {
1  return the occurrence list W_{S,k}(P) for P in S;
}
```

Fig. 4 The algorithm UpdateDmdOcc for incremental update of the occurrence list.

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. 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:

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

(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

```
procedure FastFindSerialOcc(P = (a↦e↦b), k, S = ⟨S_1, ..., S_ℓ⟩)
input: serial episode P = (a↦e↦b), window width k > 0, an input sequence S;
output: the occurrence list W for P; {
1  W := ∅; (x, y, z) := (0, 0, 0);
2  for (i := −k + 1, ..., ℓ ) do
3      last := i − 1; end := i + k
4      while x < end and (not (x > last and a ∈ S_x)) do x := x + 1;
5      while y < end and (not (y > x and e ∈ S_y)) do y := y + 1;
6      while z < end and (not (z > y and b ∈ S_z)) do z := z + 1;
7      if ( last < x < y < z < end ) then W := W ∪ {i};
8          // (x, y, z) is the lexicographically first occurrence of P in W_i;
9  end for
10 return W;
}
```

Fig. 5 An improved algorithm FastFindSerialOcc for computing the occurrence list of a serial episode.

UpdateDmdOcc in Fig. 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. 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. 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. 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 $N = ||\mathcal{S}||$.

```
global variable: a hash table TABLE : DE → 2^{−k+1,...,n};
initialization: TABLE := ∅;
procedure LookupSerialOcc(P = (a↦e↦b), k, S) {
1  if (TABLE[P] = UNDEF) then
2      V := FindSerialOcc(P, k, S);
3      if |V| ≥ σ then TABLE := TABLE ∪ {⟨ P, V⟩ };
4  end if;
5  return TABLE[P];
}
```

Fig. 6 Practical speed-up of FindSerialOcc using memoization technique.

(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. 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 $N = ||\mathcal{S}||$.

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. 6**, we show the code for practical speed-up method using memoization technique. Then, we modify PolyFreqDmd in Fig. 2 and UpdateDmdOcc in Fig. 4 as follows:

- Before Line 5 of PolyFreqDmd, insert the **initialization** line in Fig. 6.
- Replace the call of FindSerialOcc$(P, k, \mathcal{S})$ in FreqDmdRec by the call of LookupSerialOcc$(P, k, \mathcal{S})$ in Fig. 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 5 below, we will see that this modification will be useful in practice.

The running time of the algorithm FreqDmdRec in Fig. 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. 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 [11], which can be realized by replacing Lines 2 and 8 in the algorithm FreqDmdRec in Fig. 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. 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. □

```
1  R := a↦(E ∪ {e})↦b;
2  Δ := FindSerialOcc(P, k, S);
3  W := W − Δ;
4  FreqDmdRec(R, U, S, k, Σ, σ);
5  W := W ∪ Δ;
6  R := a↦(E − {e})↦b;
```

Fig. 7 The diffset technique in PolyFreqDmd.

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 [12] for itemset mining, which can be realized by replacing Line 4, Line 5, and Line 6 of FreqDmdRec in Fig. 2 with the code in **Fig. 7**. Hence, we can reduce the space complexity in Theorem 10 to $O(m + \ell) = O(|\Sigma| + \ell)$.

4. 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 [6] 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 conlude 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 [6] 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.

5. Experimental Results

5.1 Data

In this section, we give experimental results for the combinations of the algorithms in Section 3 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.

5.2 Method

We adopt the following implementations of the algorithms, where BF is the breadth-first search algorithm in Ref. 6) and DF and the others are the depth-first search algorithms presented in Section 3.

BF	:	FreqDmd [6] with breadth first itemset mining algorithm.
DF	:	PolyFreqDmd (Fig. 2) with FindSerialOcc (Fig. 4).
DF-ALT	:	DF with alternating output (ALT) (Fig. 2 with (*)).
DF-FFS	:	DF with fast update by FastFindSerialOcc (FFS) (Fig. 5).
DF-DIFF	:	DF with diffset technique (DIFF) (Fig. 7).
DF-MEM	:	DF with memoization technique (MEM) (Fig. 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+, 1.81 GHz, 2.00 GB memory) with 32-bit x86 instruction set. If it is not explicitly described, we assume that the length of the sequence is $\ell = |\mathcal{S}| = 2{,}000$, 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 $\sigma = 0.4\ell$.

5.3 Experiments

Figure 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 $s = 30$, $k = 10$,

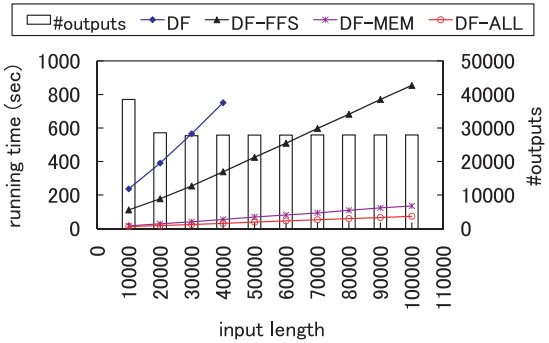

Fig. 8 Running time for the input length ℓ, where $s = 30$, $k = 10$, and $\sigma = 0.05n$.

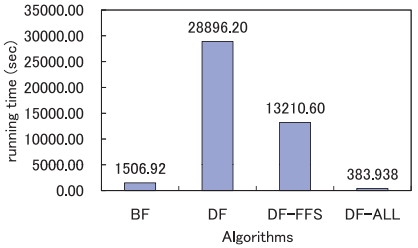

Fig. 9 Running time for algorithms, where $n = 1{,}000$ $s = 30$, $k = 10$, and $\sigma = 0.01n$.

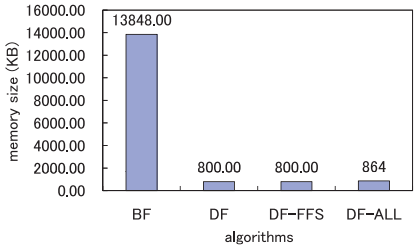

Fig. 10 Memory size for algorithms, where $n = 1{,}000$ $s = 30$, $k = 10$, and $\sigma = 0.01n$.

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. 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.

Figure 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.

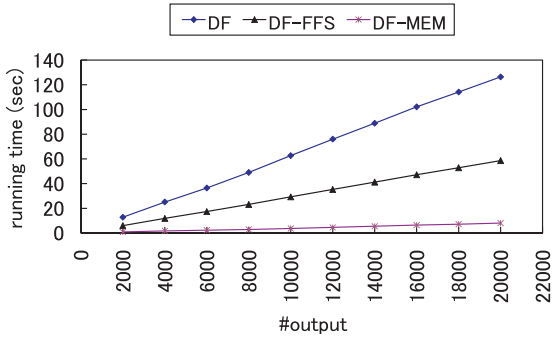

Fig. 11 Running time for the number of outputs, where $\ell = 1{,}000$, $s = 30$, $k = 10$, and $\sigma = 0.05\ell$.

Figure 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.

Figure 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.

Figure 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 $\ell = 2{,}000$. We see that the number of outputs, and thus, the running time is increasing when σ is decreasing.

Figures 13, **14** and **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.

Figures 16 and **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,

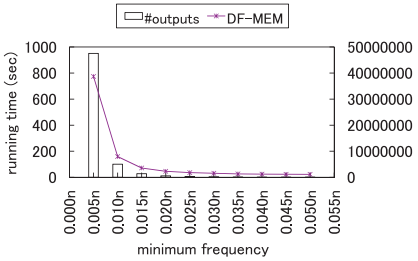

Fig. 12 Running time for the minimum frequency threshold $0.5\ell \leq \sigma \leq 5\ell$ with span 0.5ℓ, where $\ell = 2{,}000$ and $k = 10$.

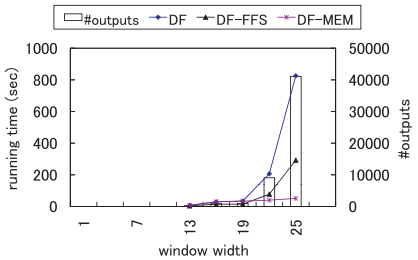

Fig. 13 Running time for the window width $13 \leq k \leq 25$, where $\ell = 2{,}000$ and $\sigma = 0.4\ell$.

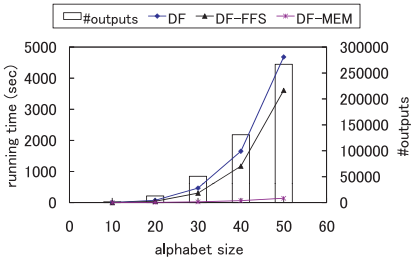

Fig. 14 Running time for the alphabet size $10 \leq s \leq 50$ with span 10, where $\ell = 2{,}000$, $\sigma = 0.4\ell$.

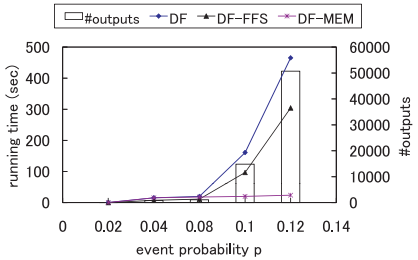

Fig. 15 Running time for the occurrence probability of events $0.02 \leq p \leq 0.12$ with span 0.02, where $\ell = 2{,}000$ and $\sigma = 0.4\ell$.

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. 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.

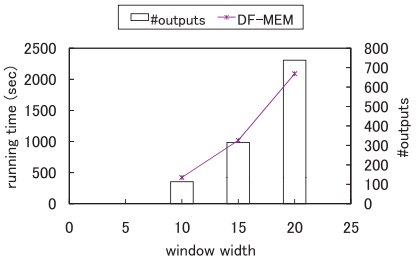

Fig. 16 Running time for the window width $10 \leq k \leq 20$, where data set is real-world event sequence of $\ell = 70{,}606$, $s = 174$, and $\sigma = 0.1\ell$.

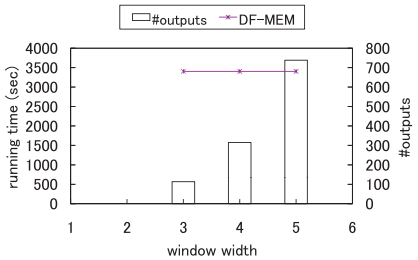

Fig. 17 Memory size for the window width $10 \leq k \leq 20$, where data set is real-world event sequence of $\ell = 70{,}606$, $s = 174$, and $\sigma = 0.1\ell$.

$$Q = \texttt{yeast} \mapsto \{\texttt{yeast}, \texttt{Enterobacter-cloacae}\} \mapsto \texttt{Staphylococcus-aureus}$$

Fig. 18 An example of the diamond episodes extracted from a bacterial culture data.

Overall, we conclude that the proposed algorithm FindDmdMain with the practical speed-up by memoization technique in Fig. 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.

6. Conclusion

This paper 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. Possible future problems are extension of PolyFreqDmd for general fragments of directed acyclic graphs [7),8)], and efficient mining of closed patterns [2),3),8),12)] for diamond episodes and their generalizations. Also, we plan to apply the proposed algorithm to bacterial cul-

ture data [5),6)].

Acknowledgments This work is partially supported by Grand-in-Aid for JSPS Fellows (20·3406). The authors would like to thank Dr. Kimiko Matsuoka of Ikagaku Co.,Ltd and Mr. Shigeki Yokoyama of Koden Industry Co., Ltd for providing bacterial culture data sets at Osaka Prefectural General Medical Center. The authors also would like thank to the anonymous referees for their careful readings of the paper and comments that greatly improve the correctness and the quality of this paper.

References

1) Agrawal, R. and Srikant, R.: Fast algorithms for mining association rules in large databases, *Proc. 20th International Conference on Very Large Data Bases*, pp.487–499 (1994).
2) Arimura, H.: Efficient algorithms for mining frequent and closed patterns from semi-structured data, *Proc. 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (*PAKDD2008*), *Lecture Notes in Artificial Intelligence 5012*, pp.2–13 (2008).
3) Arimura, H. and Uno, T.: A polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence, *Proc. 16th Annual International Symposium on Algorithms and Computation* (*ISAAC2005*), *Lecture Notes in Computer Science 3827*, pp.724–737 (2005).
4) Avis, D. and Fukuda, K.: Reverse search for enumeration, *Discrete Applied Mathematics*, Vol.65, pp.21–46 (1996).
5) Katoh, T. and Hirata, K.: Mining frequent elliptic episodes from event sequences, *Proc. 5th Workshop on Learning with Logic and Logics for Learning* (*LLLL2007*), pp.46–52 (2007).
6) Katoh, T., Hirata, K. and Harao, M.: Mining frequent diamond episodes from event sequences, *Proc. 4th International Conference on Modeling Decisions for Artificial Intelligence* (*MDAI2007*), *Lecture Notes in Artificial Intelligence 4617*, pp.477–488 (2007).
7) Mannila, H., Toivonen, H. and Verkamo, A.I.: Discovery of frequent episodes in event sequences, *Data Mining and Knowledge Discovery*, Vol.1, No.3, pp.259–289 (1997).
8) Pei, J., Wang, H., Liu, J., Wang, K., Wang, J. and Yu, P.S.: Discovering frequent closed partial orders from strings, *IEEE Trans. Knowledge and Data Engineering*, Vol.18, No.11, pp.1467–1481 (2006).
9) Pei, J., Han, J., Mortazavi-Asi, B. and Wang, J.: Mining sequential patterns by pattern-growth: The PrefixSpan approach, *IEEE Trans. Knowledge and Data Engineering*, Vol.16, No.11, pp.1–17 (2004).
10) Katoh, T., Arimura, H. and Hirata, K.: A Polynomial-Delay Polynomial-Space Algorithm for Extracting Frequent Diamond Episodes from Event Sequences, *Proc. 13th Pacific-Asia Conference on Knowledge Discovery and Data Mining* (*PAKDD2009*), *Lecture Notes in Artificial Intelligence 5476*, pp.172–183 (2009).
11) Uno, T.: Two general methods to reduce delay and change of enumeration algorithms, Technical report, National Institute of Informatics (NII) (2003).
12) Zaki, M.J. and Hsiao, C.-J.: CHARM: An efficient algorithm for closed itemset mining, *Proc. Second SIAM International Conference on Data Mining* (*SMD2002*), pp.457–478 (2002).

(Received June 18, 2009)
(Accepted October 9, 2009)

(Editor in Charge: *Kunihiko Sadakane*)

Takashi Katoh is a graduate student of the Graduate School of Information Science and Technology of Hokkaido University, Hokkaido, Japan. He received his B.S. and M.S. degrees in Information Engineering from Kyushu Institute of Technology, Iizuka, Japan, in 2006 and 2008, respectively. His research interests include data mining, knowledge discovery and bioinformatics.

Hiroki Arimura is a professor of the Graduate School of Information Science and Technology of Hokkaido University. He received his B.S. degree in Physics, M.S. and Dr.Sci. degrees in Information Systems all from Kyushu University, Fukuoka, Japan, in 1988, 1990 and 1994, respectively. His research interests include data mining, computational learning theory, and combinatorial pattern matching with applications to the Web and texts. Since 2007, he has been the director of the Global COE (Centers of Excellence) Program of "Center for Next-Generation Information Technology Based on Knowledge Discovery and Knowledge Federation" at Hokkaido University by MEXT.

Kouichi Hirata is an associate professor of Department of Artificial Intelligence, Kyushu Institute of Technology, Iizuka, Japan. He received his B.S. degree in Mathematics, M.S. and Dr.Sci. degrees in Information Systems all from Kyushu University, Fukuoka, Japan, in 1990, 1992 and 1995, respectively. His research interests include data mining, knowledge discovery and bioinformatics.