A Polynomial-Delay Polynomial-Space Algorithm for Extracting Frequent Diamond Episodes from Event Sequences

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

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

Abstract. 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 $O(|\Sigma|^2 n)$ time per an episode and in $O(|\Sigma| + n)$ space, where Σ and n are an alphabet and the length the event sequence, respectively. Finally, we give experimental results on artificial event sequences with varying several mining parameters to evaluate the efficiency of the algorithm.

1 Introduction

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.* [7] have introduced the *episode mining* to discover frequent *episodes* in an *event sequence*. Here, the episode is formulated as an *acyclic labeled digraphs* in which labels correspond to events and edges represent a temporal precedent-subsequent relation in an event sequence. Then, the episode is a richer representation of temporal relationship than a *subsequence*, which represents just a linearly ordered relation in *sequential pattern mining* (*cf.*, [3,9]). 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.

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. Note that their algorithm is general but inefficient. Then, the episode mining

T. Theeramunkong et al. (Eds.): PAKDD 2009, LNAI 5476, pp. 172–183, 2009.

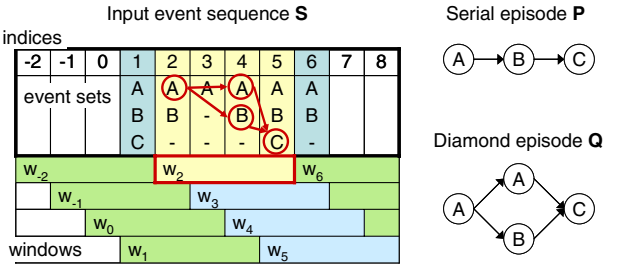

Fig. 1. (Left) An input sequence $\mathcal{S} = (S_1, \ldots, S_6)$ of length $n = 6$ over $\Sigma = \{A, B, C\}$ and their k-windows. (Right) Serial episode $P = A \mapsto B \mapsto C$ and a diamond episode $Q = A \mapsto \{A, B\} \mapsto C$. In the sequence $\mathcal{S}$, we indicate an occurrence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

has been developed by introducing the specific forms of episodes for every target area together with efficient algorithms [5,6,8].

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 to find frequent diamond and elliptic episodes concerned with *the replacement of bacteria* and *the changes for drug resistance*, which are valuable from the medical viewpoint. Here, the source and the sink are set to the bacteria and another bacteria for the former episodes, and the sensitivity of antibiotic and the resistant of the same antibiotic for the latter episodes.

Note that the algorithms designed by Katoh *et al.* [5,6] are so-called *level-wise*; The algorithms first find the information for the occurrences of serial episodes in an 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].

While the level-wise algorithms are sufficient to find frequent episodes efficiently in practice (in particular, appropriate to apply the bacterial culture data), it is difficult to give 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 $\mathcal{S}$ without duplication 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 $\mathcal{S}$, respectively. Hence, we can guarantee that the episode-growth algorithm enumerates

frequent diamond episodes in polynomial delay and in polynomial space. Further, we presents some practical optimization techniques for reducing the running time and the required space of the algorithm PolyFreqDmd.

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 time complexity. In Section 4, we give some experimental results from randomly generated event sequences to evaluate the practical performance of the algorithms. In Section 5, we conclude this paper and discuss the future works.

2 Diamond Episode

In this section, we prepare 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 $\Sigma = \{1, \ldots, m\}$ $(m \geq 1)$ be a finite alphabet with the total order $\leq$ over $\mathbf{N}$. Each element $e \in \Sigma$ is called an *event*[1]. 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 $1 \leq i \leq n$. Then, we call n the *length* of $\mathcal{S}$ and denote it by $|\mathcal{S}|$, and define the *total size* of $\mathcal{S}$ by $||\mathcal{S}|| = \sum_{i=1}^{n} |S|$. Clearly, $||\mathcal{S}|| = O(|\Sigma|n)$, but the converse is not always true.

For a fixed input sequence $\langle S_1, \ldots, S_n \rangle \in (2^{\Sigma})^*$, a *position* or an *index* on $\mathcal{S}$ is any integer, where we allow an index out of $\mathcal{S}$ by defining that $S_i = \emptyset$ if i is either $i \leq 0$ or $i > n$. Let $1 \leq k \leq n$ be a fixed positive integer, called the *window width*. For any index $-k+1 \leq i \leq n$, 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 $\{ W_i^{\mathcal{S},k} \mid -k+1 \leq i \leq n \}$ 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, when they are clear from the context. Moreover, we may identify the set of all k-windows by the set $\{ -k+1 \leq i \leq n \mid i \in \mathbf{W}_{\mathcal{S},k} \} \subseteq \mathbf{Z}$ 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 $P = \langle a, E, b \rangle \in \Sigma \times 2^{\Sigma} \times \Sigma$ (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 P, respectively. For the body E, we denote the maximum element in E (w.r.t. Σ) by $\max(E)$.

To emphasize the chronological dependencies of events, we often write $(a_1 \mapsto \cdots \mapsto a_m)$ and $(a \mapsto E \mapsto b)$ for an m-serial episode $\langle a_1, \ldots, a_m \rangle$ and a diamond episode $\langle a, E, b \rangle$, respectively. Also we denote the classes of m-serial episodes,

[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*.

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 $(a \mapsto b \mapsto c) \in \mathcal{SE}_3$ are equivalent to $(a \mapsto \emptyset \mapsto b)$ and $(a \mapsto \{b\} \mapsto c) \in \mathcal{PDE}$, respectively, we see that $\mathcal{SE}_1 \cup \mathcal{SE}_2 \cup \mathcal{SE}_3 \cup \mathcal{PDE} = \mathcal{DE}$.

Example 1. In Figure 1, we show examples of an event sequence $\mathcal{S} = (ABC, AB, A, AB, ABC, AB)$ of length $n = 6$, a serial episode $P = A \mapsto B \mapsto C$ and a diamond episode $Q = A \mapsto \{A, B\} \mapsto C$ on an alphabet of events $\Sigma = \{A, B, C\}$, where the body $\{A, C\}$ is written as a sequence AC.

Next, we introduce the concept of occurrences of episodes in a window. Then, we give the formal definition of the occurrences of episodes, which is consistent with the original definition by [7]. Let $P = e_1 \mapsto \cdots \mapsto e_n$ be a serial episode, $Q = e_{\mathrm{s}} \mapsto \{e_1, \ldots, e_m\} \mapsto e_{\mathrm{t}}$ 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* an window $W = \langle S_1 \cdots S_k \rangle \in \mathbf{W}_{\mathcal{S},k}$, denoted by $P \sqsubseteq W$, iff 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 $P = e_{\mathrm{s}} \mapsto \{e_1, \ldots, e_m\} \mapsto e_{\mathrm{t}}$ $(m \geq 0)$ *occurs in* an window $W = \langle S_1 \cdots S_k \rangle \in \mathbf{W}_{\mathcal{S},k}$, denoted by $D \sqsubseteq W$, iff there exists some mapping $h : \{\mathrm{s}, \mathrm{t}, 1, \ldots, m\} \to \{1, \ldots, k\}$ satisfying (i) for every $i \in \{1, \ldots, m\}$, $1 \leq h(\mathrm{s}) < h(i) < h(\mathrm{t}) \leq k$ holds, and (ii) $e_i \in S_{h(i)}$ holds for every $i \in \{\mathrm{s}, \mathrm{t}, 1, \ldots, m\}$.

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 $\mathbf{W}_{\mathcal{S},k}(P) = \{ -k + 1 \leq i \leq n \,|\, P \sqsubseteq W_i \}$, the set of occurrences of P in an input $\mathcal{S}$. We may call the element $i \in \mathbf{W}_{\mathcal{S},k}(P)$ a *label*. 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 $\mathcal{S} = (ABC, AB, A, AB, ABC, AB)$ in Figure 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., $\mathbf{W}_{\mathcal{S},5} = \{ W_i \mid -2 \leq i \leq 6 \}$. Among them, the window list for a diamond episode $P = A \mapsto AB \mapsto C$ is $\mathbf{W}(P) = \{W_2, W_3\}$.

Lemma 1. *Let P be a partial diamond episode $(e_{\mathrm{s}} \mapsto E \mapsto e_{\mathrm{t}})$ and W a window in $\mathbf{W}_{\mathcal{S},k}$. Then, $P \sqsubseteq W$ iff for every $e \in E$, there exists some position p for e such that $e \in S_p$ and $st(e_{\mathrm{s}}, W) < p < et(e_{\mathrm{t}}, W)$ hold.*

Proof. (Only if-direction) If $P \sqsubseteq W$ then there exists some embedding h from P to W. By restricting h to the serial episode $(e_{\mathrm{s}} \mapsto \cdots \mapsto e_{\mathrm{t}})$, we obtain the claim. (If-direction) Suppose that for every $e \in E$, there exists a position p_e for e with $st(e_{\mathrm{s}}, W) < p < et(e_{\mathrm{t}}, W)$. Then, we can build a mapping h by $h(e_{\mathrm{s}}) = st(e_{\mathrm{s}}, W)$, $h(e_{\mathrm{t}}) = et(e_{\mathrm{t}}, W)$, and $h(e) = p_e$ for every $e \in E$. Then, the claim holds. □

Lemma 1 implies the following two important corollaries.

Corollary 1 (serial construction of diamond episodes). *Let P be a partial diamond episode $(e_s \mapsto E \mapsto e_t)$ $(m \geq 0)$ and W a window in $\boldsymbol{W}_{\mathcal{S},k}$. Then, $P \sqsubseteq W$ iff $(e_s \mapsto e \mapsto e_s) \sqsubseteq W$ for every $e \in E$.*

Corollary 2 (anti-monotonicity for diamond episodes). *Let $a, b \in \Sigma$ be events, and $E, F \subseteq \Sigma$ be event sets. For every 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 $(a \mapsto F \mapsto b) \in \mathcal{F}_{\mathcal{S},k,\sigma}$.*

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

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 $P \in \mathcal{F}_{\mathcal{S},k,\sigma}$ 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 so 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 $\mathcal{S} = (S_1, \ldots, S_n) \in (2^{\Sigma})^*$ 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.

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 of $\mathcal{DE}$ appearing in an input sequence $\mathcal{S}$.

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

```
algorithm PolyFreqDmd(S, k, Σ, σ)
input: input event sequence S ∈ (2^Σ)* of length n, window width k > 0,
alphabet of events Σ, the minimum frequency 1 ≤ σ ≤ n + 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          D0 := (a ↦ ∅ ↦ b); //2-serial episode
6          W0 := the occurrence window list W_{S,k}(D0) for D0;
7          FreqDmdRec(D0, W0, S, k, Σ0, σ);
8      end for
}
procedure FreqDmdRec(D = (a ↦ E ↦ b), W, S, k, Σ, σ)
output: finds all frequent diamond episodes of the form a ↦ E ↦ b;
{
1  if ( |W| ≥ σ ) then
2      output D; //(*) output D if the depth is odd (alternating output);
3      foreach ( e ∈ Σ (e > max(E)) ) ) do
4          C = a ↦ (E ∪ {e}) ↦ b;
5          U := UpdateDmdOcc(W, e, D, k, S);    //Computing U = W_{S,k}(C)
6          FreqDmdRec(C, U, S, k, Σ, σ);
7      end for
8      //(*) output D 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 Corollary 2, in each iteration of FreqDmdRec, the algorithm tests if the current candidate $D = (a \mapsto E \mapsto b)$ is frequent. If so, FreqDmdRec output D, and furthermore, for every event $e \in \Sigma$ such that $e > \max(E)$, FreqDmdRec grows diamond episode D by adding the new event e to the body E. Otherwise, FreqDmdRec prunes the search below D and backtrack to its parent. We call this process the *tail expansion* for diamond episodes. For episodes P, Q, if Q is generated from P by adding new event e as above, then we say that P is a *parent* of Q, or Q is a *child* of P.

Lemma 2. *For any the window width $k > 0$ and any minimum support σ, the algorithm* PolyFreqDmd *enumerates all and only frequent diamond episodes from $\mathcal{S}$ without duplicates.*

Proof. Suppose that $Q = (a \mapsto E \cup \{e\} \mapsto b) \in \mathcal{DE}$ is a child of some $P = (a \mapsto E \mapsto b) \in \mathcal{DE}$ obtained by the tail expansion such that $e > \max(E)$. From Corollary 2, we see that any frequent Q can be obtained by expanding some

algorithm UpdateDmdOcc($D, e, W, k, \mathcal{S}$)
input: a parent serial episode $D = (a \mapsto E \mapsto b)$, a new event $e > \max(E)$, the old occurrence list W for D, $k \leq 1$, an input sequence $\mathcal{S}$;
output: the new occurrence list U for the child $C = (a \mapsto E \cup \{e\} \mapsto b)$;
{
  $V :=$ FindSerialOcc($P = (a \mapsto e \mapsto b), k, \mathcal{S}$);
  return $U := W \cap V$;
}
procedure FindSerialOcc($(a \mapsto e \mapsto b), k, \mathcal{S}$)
{ **return** the occurrence list $\mathbf{W}_{\mathcal{S},k}(P)$ for P in $\mathcal{S}$; }

Fig. 3. The algorithm UpdateDmdOcc for incremental update of the occurrence list

frequent parent P. Furthermore, since $e > \max(E)$, the parent P is unique for each Q. 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 DFS on $\mathcal{T}$ by backtracking, the result immediately follows. □

In the recursive procedure FreqDmdRec in Fig. 2, the procedure newly create a child episode $C = (a \mapsto E \cup \{e\} \mapsto b)$ from the parent $D = (a \mapsto E \mapsto b)$ by tail expansion with $e \in \Sigma$ at Line 4. Then, at Line 5, it computes the new occurrence window list $U = W_{\mathcal{S},k}(C)$ for C in $\mathcal{S}$. To compute the new list U, we can use a native procedure that scans all k-windows in $\mathcal{S}$ one by one for checking occurrences of C.

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

From Lemma 3, this naive algorithm requires $O(|\Sigma|kmn)$ time, where k is the window width, $m = ||D||$ is the episode size, and $n = |\mathcal{S}|$ is the input length.

In Fig. 3, we show an improved algorithm UpdateDmdOcc that computes the new occurrence list $U = W_{\mathcal{S},k}(C)$ in $O(|\Sigma|kn)$ time, by dropping the factor of $m = ||C||$, with incrementally updating the old list W for the parent D. To see the validity of the improved algorithm, we require two properties, called the *serial construction* for $\mathcal{DE}$ shown in Corollary 1 and the *downward closure property* for $\mathcal{DE}$ shown in Lemma 4 below. Here, Lemma 4 is an extension of the downward closure property for itemsets [1].

Lemma 4 (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:*

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

From Lemma 3 and Lemma 4, we see the correctness of the improved algorithm UpdateDmdOcc in Fig. 3, and have the next lemma. Note in the following that the computation time of UpdateDmdOcc does not depends on the size

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

Fig. 4. An improved algorithm FirstFindSerialOcc for computing the occurrence list of a serial episode

$m = ||C||$ of the child episode. If we implement the procedure FindSerialOcc by an algorithm of Lemma 3, we have the next result.

Lemma 5. *The algorithm* UpdateDmdOcc *in Fig. 3, given the old list W for the parent diamond episode D and a newly added event e, computes the new occurrence list $U = W_{\mathcal{S},k}(C)$ for a new child C in $O(kN) = O(|\Sigma|kn)$ time, where $n = |\mathcal{S}|$ and $N = ||\mathcal{S}||$ are the length and the total size of input $\mathcal{S}$, respectively.*

Next, we present a faster algorithm for implementing the procedure FindSerialOcc for serial episodes than that of Lemma 3. In Fig. 4, we show the faster 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 6. *The algorithm* FastFindSerialOcc *in Fig. 4 computes the occurrence list of a 3-serial episode $P = a \mapsto b \mapsto c$ in an input sequence $\mathcal{S}$ of length n in $O(N) = O(|\Sigma|n)$ time regardless window width k, where $N = ||\mathcal{S}||$.*

Corollary 3. *Equipped with* FastFindSerialOcc *in Fig. 4, the modified algorithm* UpdateDmdOcc *computes $U = W_{\mathcal{S},k}(C)$ for a child $C \in \mathcal{DE}$ from the list $W = W_{\mathcal{S},k}(D)$ for the parent $D \in \mathcal{DE}$ and $e \in \Sigma$ in $O(N) = O(|\Sigma|n)$ time, where $n = |\mathcal{S}|$ 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 $((a \mapsto e \mapsto b), k, \mathcal{S})$ $(e \in \Sigma)$. In the worst case, the number of calls may be $|\Sigma|$ times 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 $TABLE$.

```
1  global variable: a hash table TABLE : Σ → 2^{−k+1,...,n};
2  initialization: TABLE := ∅;
3  procedure LookupSerialOcc((a↦e↦b), k ∈ N, S)
{
4      if (TABLE[e] = UNDEF) then
5          V := FindSerialOcc((a↦e↦b), k, S);
6          if |V| ≥ σ then TABLE := TABLE ∪ {⟨ e, V⟩ };
7      end if;
8      return TABLE[e];
9  }
```

Fig. 5. Practical speed-up of FindSerialOcc using dynamic programming

In Fig. 5, we show the codes for practical speed-up method using dynamic programming. Then, we modify PolyFreqDmd in Fig. 2 and UpdateDmdOcc in Fig. 3 as follows:

- Before Line 5 of PolyFreqDmd, insert Line 2 (**initialization**) in Fig. 5.
- Replace the call of FindSerialOcc$((a\mapsto e\mapsto b), k, \mathcal{S})$ in FreqDmdRec by the call of LookupSerialOcc$((a\mapsto e\mapsto b), k, \mathcal{S})$ in Fig. 5.

This modification does not change the behavior of procedures PolyFreqDmd, FreqDmdRec and UpdateDmdOcc. Moreover, this makes the total number of the calls of FindSerialOcc to be bounded above by $|\Sigma|^3$, while it uses $O(|\Sigma|n)$ space in main memory. In Section 4 below, we know this technique 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 $U = \mathbf{W}_{\mathcal{S},k}(D)$ of a candidate $D \in \mathcal{DE}$ in $\mathcal{S}$, where $m = ||D||$ 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. 4 yields the delay of $O(d \cdot |\Sigma| \cdot T(n, N))$, where factor d comes from that it takes at least d recursive calls 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 *alternating output* in backtracking [10], which can be realized by replacing Line 2 and Line 8 in the algorithm FreqDmdRec with the corresponding lines (*) in the comments.

Theorem 1. *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* PolyFreqDmd *in Fig. 2 finds all σ-frequent diamond episodes D in $\mathcal{DE}$ occurring in $\mathcal{S}$ without duplicates in $O(|\Sigma|N) = O(|\Sigma|^2 n)$ delay (time per frequent episode) and $O(mn + N) = O(|\Sigma|n)$ space, where $N = ||\mathcal{S}||$ and $m = ||D||$ is the maximum size of frequent episodes.*

Corollary 4. *The frequent diamond episode mining problem is solvable in linear delay w.r.t. the total input size and in polynomial space.*

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

Fig. 6. The diffset technique in PolyFreqDmd

Finally, we can reduce the space complexity of the algorithm PolyFreqDmd by using the *diffset* technique introduced by Zaki [11] for itemset mining, which can be realized by replacing Line 5 and Line 6 of PolyFreqDmd with the code in Fig. 6. Hence, we can reduce the space complexity in Theorem 1 to $O(m+n) = O(|\Sigma| + n)$.

4 Experimental Results

In this section, we give the experimental results for the following combinations of the algorithms given in Section 3, by applying to the randomly generated event sequences $\mathcal{S} = (S_1, \ldots, S_n)$ over an alphabet $\Sigma = \{1, \ldots, s\}$, where each event set S_i $(i = 1, \ldots, n)$ is generated by a uniform distribution with letter probability $0 \leq p \leq 1/|\Sigma|$ and stopping probability $1 - p$.

DF : PolyFreqDmd (Fig. 2) with FindSerialOcc (Fig. 3).
DF-SWO : DF with alternative output (SWO) (Fig. 2 with (*)).
DF-FFS : DF with fast update by FastFindSerialOcc (FFS) (Fig. 4).
DF-DIFF : DF with diffset technique (DIFF) (Fig. 6).
DF-DP : DF-FFS with dynamic programming (DP) (Fig. 5).

All experiments were run in a PC (AMD Mobile Athlon64 Processor 3000+, 1.81GHz, 2.00GB memory) with 32-bit x86 instruction set. Without saying explicitly, we assume that the length of the sequence is $n = |\mathcal{S}| = 2000$, the alphabet size is $s = |\mathcal{S}| = 30$, the probability of each event is $p = 0.1$, the window width is $k = 10$, the minimum frequency threshold is $\sigma = 0.4$.

Fig. 7 shows the running time and the number of solutions of the algorithms DF, DF-FFS and DF-DP for the input length n, where $s = 20$, $k = 10$ and $\sigma = 0.1n$. Then, we know that DF-FFS is twice as faster as DF and DF-DP is one hundred times as faster as DF. On the other hand, we cannot find any difference between DF-SWO, DF-DIFF and DF on this data set, although the first two techniques are useful in technical improvements. Moreover, the running time of these algorithms seems to be linear in the input size and thus expected to scales well on large datasets.

Fig. 8 shows the running time for the number of outputs, where $n = 10,000$, $k = 30$ and $\sigma = 0.3n$. Then, we see that the slope is almost constant and thus the delay is just determined by the input size as indicated by Theorem 1.

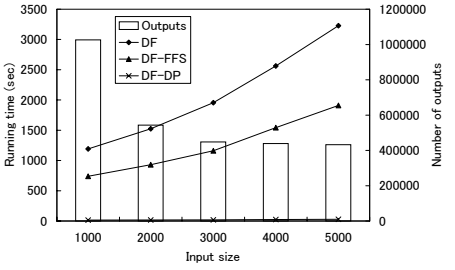

Fig. 7. Running time for the input length n, where $s = 20$, $k = 10$ and $\sigma = 0.1n$

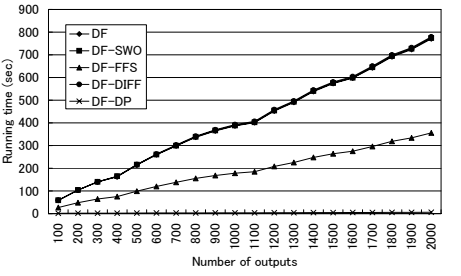

Fig. 8. Running time for the number of outputs, where $n = 10,000$, $k = 30$ and $\sigma = 0.3n$

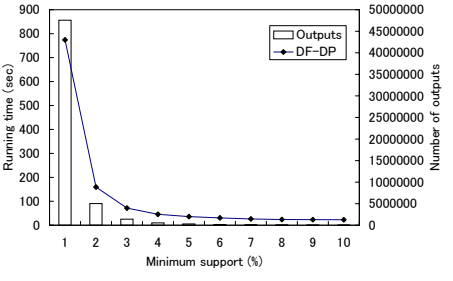

Fig. 9. Running time for the minimum support $0.5n \leq \sigma \leq 5n$ with span $0.5n$, where $n = 2,000$ and $k = 10$

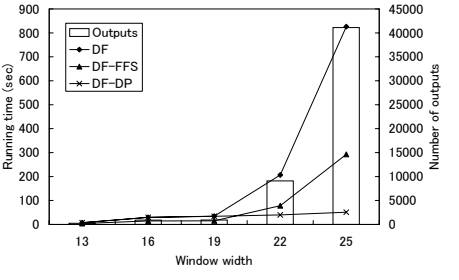

Fig. 10. Running time for the window width $13 \leq k \leq 25$, where $n = 2,000$ and $\sigma = 0.4n$

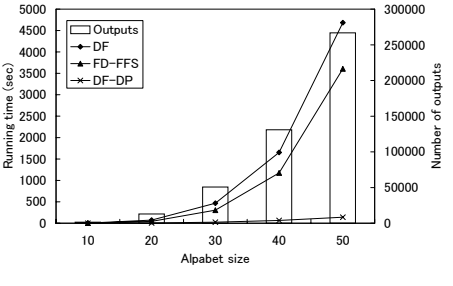

Fig. 11. Running time for the alphabet size $10 \leq |\Sigma| \leq 50$ with span 10, where $n = 2,000$, $\sigma = 0.4n$

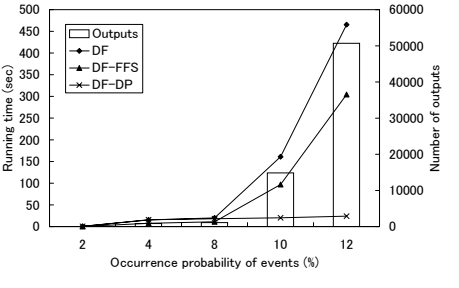

Fig. 12. Running time for the occurrence probability of events $0.02 \leq p \leq 0.12$ with span 0.02, where $n = 2,000$ and $\sigma = 0.4n$

Fig. 9 shows the running time of DF-DP, our fastest algorithm, with varying the minimum support $0.5n \leq \sigma \leq 5.0n$ with the input size $n = 2000$. We see that the number of outputs, and thus, the running time increase as σ decreases.

Figs. 10, 11 and 12 show the running time of the algorithms DF, DF-FFS and DF-DP with varying the window width $13 \leq k \leq 25$, the size of alphabet

$10 \leq |\Sigma| \leq 50$ and the event probability $0.02 \leq p \leq 0.12$, respectively. Then, we see that DF-DP outperforms other algorithms in most cases. The performance of DF-DP is stable in most datasets and the parameter settings. We also see that DF-FFS is from 20% to 60% faster than DF.

Overall, we conclude that the proposed algorithm FindDmdMain with the practical speed-up technique by dynamic programming in Fig. 5 (DF-DP) is quite efficient on the data sets used these experiments. The fast linear-time update by FastFindSerialOcc (DF-FFS) achieves twice speed-up.

5 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 the time and the space complexities of the algorithm.

Possible future problems are extension of PolyFreqDmd for general fragments of DAGs [7,8], and efficient mining of closed patterns [2,3,8,11] for diamond episodes and their generalizations. Also, we plan to apply the proposed algorithm to bacterial culture data [5,6].

References

1. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules in large databases. In: Proc. 20th VLDB, pp. 487–499 (1994)
2. Arimura, H.: Efficient algorithms for mining frequent and closed patterns from semi-structured data. In: Washio, T., Suzuki, E., Ting, K.M., Inokuchi, A. (eds.) PAKDD 2008. LNCS, vol. 5012, pp. 2–13. Springer, Heidelberg (2008)
3. Arimura, H., Uno, T.: A polynomial space and polynomial delay algorithm for enumeration of maximal motifs in a sequence. In: Deng, X., Du, D.-Z. (eds.) ISAAC 2005. LNCS, vol. 3827, pp. 724–737. Springer, Heidelberg (2005)
4. Avis, D., Fukuda, K.: Reverse search for enumeration. Discrete Applied Mathematics 65, 21–46 (1996)
5. Katoh, T., Hirata, K.: Mining frequent elliptic episodes from event sequences. In: Proc. 5th LLLL, pp. 46–52 (2007)
6. Katoh, T., Hirata, K., Harao, M.: Mining frequent diamond episodes from event sequences. In: Torra, V., Narukawa, Y., Yoshida, Y. (eds.) MDAI 2007. LNCS (LNAI), vol. 4617, pp. 477–488. Springer, Heidelberg (2007)
7. Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of frequent episodes in event sequences. Data Mining and Knowledge Discovery 1, 259–289 (1997)
8. Pei, J., Wang, H., Liu, J., Wang, K., Wang, J., Yu, P.S.: Discovering frequent closed partial orders from strings. IEEE TKDE 18, 1467–1481 (2006)
9. Pei, J., Han, J., Mortazavi-Asi, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Hsu, M.-C.: Mining sequential patterns by pattern-growth: The PrefixSpan approach. IEEE Trans. Knowledge and Data Engineering 16, 1–17 (2004)
10. Uno, T.: Two general methods to reduce delay and change of enumeration algorithms, NII Technical Report, NII-2003-004E (April 2003)
11. Zaki, M.J., Hsiao, C.-J.: CHARM: An efficient algorithm for closed itemset mining. In: Proc. 2nd SDM, pp. 457–478. SIAM, Philadelphia (2002)