Mining Serial Episode Rules with Time Lags over Multiple Data Streams

Tung-Ying Lee[1], En Tzu Wang[1], and Arbee L.P. Chen[2]

[1] Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan, R.O.C.
u902521@alumni.nthu.edu.tw, m9221009@em92.ndhu.edu.tw
[2] Department of Computer Science, National Chengchi University, Taipei, Taiwan, R.O.C.
alpchen@cs.nccu.edu.tw

Abstract. The problem of discovering *episode rules* from static databases has been studied for years due to its wide applications in prediction. In this paper, we make the first attempt to study a special episode rule, named *serial episode rule with a time lag* in an environment of multiple data streams. This rule can be widely used in different applications, such as traffic monitoring over multiple car passing streams in highways. Mining serial episode rules over the data stream environment is a challenge due to the high data arrival rates and the infinite length of the data streams. In this paper, we propose two methods considering different criteria on space utilization and precision to solve the problem by using a prefix tree to summarize the data streams and then traversing the prefix tree to generate the rules. A series of experiments on real data is performed to evaluate the two methods.

Keyword: Multiple data streams, Data mining, Serial episode rule, Time lag.

1 Introduction

The progress of technologies including communications and computations has led to a more convenient way of life and also brings huge amounts of commercial benefits. However, the high speed of communications and powerful capability of computations generate data as a form of *continuous data streams* rather than static persistent datasets, raising the complexity of data management. A data stream is an unbounded sequence of data continuously generated at a high speed. In such applications as network traffic management, sensor network systems and traffic management systems, we may need to handle different categories of the data streams.

Consider a scenario as follows. Roads are connected to each other in real road networks. Consequently, certain roads with heavy traffic may cause the other roads to be obstructed. Users may be interested in the following rule: *when road A and road B have heavy traffic, five minutes later, road C will most likely be congested.* This kind of rules can be applied to navigation systems, helping the users to avoid being blocked up in traffic. In the traffic monitoring systems, *flow* and *occupancy* of a road detected by sensors are employed in qualifying the traffic conditions [2]. The flow of a road is the

© Springer-Verlag Berlin Heidelberg 2008

number of cars passing by a sensor per minute. The occupancy of a zone is a ratio of a time interval in which the detection zone is occupied by a car. The collected data of the flow and occupancy are divided into several discrete classes [6]. Therefore, different traffic conditions can be represented by the different classes of the flow and occupancy. For example, if a road is under the condition of *low flow* and *high occupancy*, the traffic of this road can be qualified as "congested." Since many roads are simultaneously monitored by the sensors in a traffic monitoring system, a continuous multi-streams environment is formed. In order to find the rules as discussed above, a new problem of *finding serial episode rules with time lags over multiple streams* is addressed and solved in this paper. Different from our approach to design the new online mining algorithms, a framework using existing offline mining techniques to find frequent episodes from the historic data to monitor the current traffic status and to predict the coming traffic in time is proposed in [7].

We first introduce the problem of discovering episode rules from the static time series data [3] [4] [5] [10] [11] [12] in the following. Formally, an episode can be described using a *directed acyclic graph* (DAG). Each node in the graph represents an *event*. Suppose that the nodes *A* and *B* are kept in an episode *E*. If there is an edge from *A* to *B*, it means that *B* occurs after *A*. If there is no edge between *A* and *B*, it means that the order of the appearances of *A* and *B* is not important. The *time interval* of an occurrence of an episode *E* in the time series data is the interval from the occurring of the first event in *E* to that of the last event in *E* appearing in the time series. In addition, a parameter named *time bound* is set to limit the duration of the occurrence. *This means that we only concern about the occurrences of the episode, whose time interval are within the time bound.* We call these occurrences *valid occurrences* of the episode for the following discussion. The number of valid occurrences of an episode in the time series data is counted to determine whether it is *frequent*, and then from the frequent episodes to derive an episode rule. There are two ways of counting the number of valid occurrences of an episode, the *window-based strategy* and the *minimal occurrence strategy*.

The window-based strategy is to slide a window with a length equaling the time bound *T* over the time series data to compute the total number of the windows containing an episode *E*. In this strategy, a certain valid occurrence of *E* may be counted more than once due to being contained in the distinct windows. On the other hand, the minimal occurrence strategy is to count the number of valid occurrences satisfying the following constraint: *within its time interval, the other valid occurrences do not exist.* Given a user-defined threshold, named *minimum support*, if the number of the windows in the window-based strategy or the number of the counted valid occurrences in the minimal occurrence strategy of an episode exceeds or equals the minimum support, it is defined as *frequent*. The algorithms proposed in [10][11][12] for finding frequent episodes based on the window-based strategy and the minimal occurrence strategy are named WINEPI and MINEPI, respectively. In these approaches, only *serial episodes* and *parallel episodes* are considered. A serial episode is a sequence of events while a parallel episode is a set of events. The above methods based on the Apriori algorithm [1] are designed only for these two types of episodes because all episodes can be decomposed into their combinations. Finding frequent episodes is extended to finding *episode rules* in [11]. An episode rule describes that after an episode occurs another episode may occur. Harm et al. define a new episode rule by specifying a time lag between these two episodes and propose the MOWCATL approach to find the episode rules in multiple sequences [3][5]. The s*erial rule* discussed in [3] is a special episode rule restricted to have only two *serial episodes*. We call it *serial episode rules with time lags* (SERs) in this paper. An SER is represented in a form of $X \rightarrow_L Y$ where X and Y are *serial episodes* and L is a fixed time lag. The first serial episode X is defined as the *precursor* of the rule and the second serial episode Y is defined as the *successor*. The rule means that after X occurs, Y will probably occur in L time units later. As mining association rules, *support* and *confidence* are also used to be the measures of rule interestingness for mining SERs. The support of an SER is the number of the *tightest* interval in which the rule occurs, which is used to express the strength of the rule. The confidence of an SER is a conditional probability that the successor of the rule occurs within the time lag, given that the precursor occurs. The users can define their own interestingness of the rules by giving two thresholds, *minimum support* and *minimum confidence*. If both the support and confidence of a rule are no less than the minimum support and the minimum confidence respectively, the rule is defined as *significant*. Therefore, mining SERs is to return all the significant SERs to the users.

Refer to the scenario of the traffic prediction which motives us to *address this new problem of mining SERs over multi-streams.* We deal with the multi-streams by assuming that the events are generated from n streams with the same fixed sampling rate, and all the events generated at the same time form an *n-tuple event*. Let an *itemset* be a subset of an n-tuple event. The *serial episode* to be discussed in our problem is then defined as a sequence of itemsets. The challenges of solving this problem are described as follows. First, enormous amounts of episodes enumerated from the multiple streams may overload the memory utilization and incur enormous processing time. Second, in order to check whether the delay between the precursor and successor of a rule satisfies the time lag of a significant SER, the time intervals within which the episodes occur must be recorded. Obviously, as time goes by, the records for storing the time intervals of the episodes may use a huge amount of memory space.

In this paper, we propose a framework for mining significant SERs over multiple data streams, which counts the valid occurrences of the serial episodes using the minimal occurrence strategy. Different from the previous approaches which only focus on finding frequent episodes over event streams [9][13], in this paper, we also combine the serial episodes to generate the serial episode rules with time lags. Therefore, not only the support counts but also the time information of the occurrences of a serial episode need to be efficiently processed. In this framework, a prefix tree is employed to store the serial episodes enumerated from the n-tuple event stream. According to the considerations of different criteria on space utilization and precision, two methods storing different information in the prefix tree are proposed in this paper. Moreover, Lossy Counting [8] is applied to the two methods to save the memory required.

The remainder of the paper is organized as follows. Section 2 introduces the preliminaries and formulates the problem to be solved. The detailed algorithms are described in Section 3. The experiment results for evaluating the methods are presented in Section 4, and finally, Section 5 concludes this work.

2 Preliminaries

The definitions of the supports and confidences of SERs, the problem formulation, the data structure used in the methods, and several observations of SERs are described in this section.

2.1 Problem Formulation

Consider a centralized system which collects *n synchronized data streams* denoted as DS_1, DS_2,…, DS_n. A data stream in the system is an unbounded sequence of items (events). Moreover, the domain of the data in each data stream may be distinct from the others. The synchronized data streams mean that the data arrival rate for each data stream is consistent, that is, each data stream generates an item at the same time unit. Let $DS_j(i)$ represent the item arriving at time i and coming from the j^{th} stream. An *n-tuple event* $R(i)$ is defined as a set of items coming from all data streams at time i, i.e. $R(i) = \{DS_1(i), DS_2(i), ..., DS_n(i)\}$. Moreover, an *itemset* is defined as a subset of an n-tuple event. Therefore, a *serial episode* discussed in this paper is described as an ordered list of itemsets, for example $S = (I_1)(I_2), \ldots, (I_k)$ is a serial episode, where I_1, I_2, …, I_k are itemsets. The following first two definitions follow [11].

Definition 1: (Minimal Occurrence) *Given a serial episode S, a minimal occurrence of S can be identified by its time interval. A time interval* $[a, b]$ *is a minimal occurrence of S if it satisfies the following two constraints: 1) S occurs in the time interval* $[a, b]$ *and 2) S does not occur in any proper subinterval of* $[a, b]$*, that is, S does not occur in* $[c, d]$*, where* $a \le c$, $d \le b$ *and the duration of* $[a, b]$ > *the duration of* $[c, d]$. *The duration of* $[a, b]$ *is defined to equal* $b - a + 1$. *The set of all minimal occurrences of S is denoted as* $\mathrm{MO}(S) = \{[a, b] | [a, b]$ *is a minimal occurrence of S*$\}$. ■

For a minimal occurrence of a serial episode E, if it satisfies the time bound T, it is called a *valid minimal occurrence* of E in the following discussion.

Definition 2: (Support of Serial Episode) *Given a time bound T and a serial episode S, the support of S, supp(S), is the number of valid minimal occurrences of S i.e.,* $\mathrm{supp}(S) = |\{[a, b] \mid [a, b] \in \mathrm{MO}(S) \wedge (b - a + 1) \le T\}|$. ■

Definition 3: (Support of SER) *Given serial episode rule* R: $S_1 \rightarrow_{lag = L} S_2$ *with a time bound T, where the time lag is equal to L. The support of R representing the strength of R and helping to recognize the degree of significance of R is defined as* $\mathrm{supp}(R) = |\{[a, b] \mid [a, b] \in \mathrm{MO}(S_1) \wedge (b - a + 1) \le T \wedge \exists [c, d] \in \mathrm{MO}(S_2) \text{ s.t. } (d - c + 1) \le T \wedge (c - a) = L\}|$. ■

Definition 4: (Confidence of SER) *The confidence of the serial episode rule* R: $S_1 \rightarrow_{lag = L} S_2$ *is a conditional probability that* S_2 *occurs and satisfies the fixed time lag L, given that* S_1 *occurs, defined as* $\mathrm{conf}(R) = \mathrm{supp}(R)/\mathrm{supp}(S_1)$. ■

Given four parameters including the *maximum time lag, Lmax*, the *minimum support*, *minsup*, the *minimum confidence*, *minconf,* and the *time bound*, *T*. The problem of mining SERs is to find all the SERs, e.g. R: $S_1 \rightarrow_{lag = L} S_2$, satisfying the following constraints: 1) $L \le Lmax$, 2) $\mathrm{supp}(R) \ge N \times minsup$, where N is the number of the received n-tuple events generated from the multiple streams, and 3) $\mathrm{conf}(R) \ge minconf$.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
a	b	b	c	g	a	b	f	h	g	a	b	a	b	g	f	g	a	b
A	B	B	S	G	A	B	C	H	G	A	B	A	B	G	F	G	A	B

Fig. 1. An example of the 2-tuple events

Table 1. All parameters for Figure 1

Lmax	Minsup	Minconf	T
5	0.2	0.8	3

Table 2. Examples of serial episodes and serial episode rules with time lags

Serial episodes	**Minimal Occurrences**	**Support**
$(a\ A)(b\ B)$	[1, 2], [6, 7], [11, 12], [13, 14], [18, 19]	$5 > 19 \times 0.2 = 3.8$
$(g\ G)$	[5, 5], [10, 10], [15, 15], [17, 17]	$4 > 3.8$
Serial episode rules	**Minimal Occurrences**	**Support, Confidence**
$(a\ A)(b\ B) \rightarrow_4 (g\ G)$	[1, 2]→[5, 5], [6, 7]→[10, 10], [11, 12]→[15, 15], [13, 14]→[17, 17]	Supp: 4, Conf: 4/5 = 0.8

Moreover, the calculating of the supports for the serial episodes and the SERs has to take the time bound T into account.

Consider an example as in Figure 1. 19 *2-tuple* events have been received. All parameters are given as in Table 1 and some serial episodes, serial episode rules with time lags, and their corresponding minimal occurrences are listed in Table 2. The interval [1, 3] is not the minimal occurrence of $(a\ A)(b\ B)$ due to the interval [1, 2] which is a proper subset [1, 3]. The support of the serial episode $(a\ A)(b\ B)$ is 5 because each minimal occurrence of $(a\ A)(b\ B)$ are valid. By comparing all the time intervals of the valid minimal occurrences of the two serial episodes $(a\ A)(b\ B)$ and $(g\ G)$, the support of the serial episode rule R: $(a\ A)(b\ B) \rightarrow_4 (g\ G)$ is calculated.

2.2 Data Structure of the Methods: Prefix Tree

Since the serial episodes may have common prefixes, using the *prefix tree structure* to store the serial episodes is more space-efficient. The prefix tree structure consists of the nodes with labels and a root without a label. Moreover, each node in the prefix tree belongs to a *level*. The root is at the level 0 and if a node is at the level k, its children are at the level k + 1.The node in the prefix tree is used to represent a serial episode. By orderly combining all labels of those nodes in the path from the root to the node, the serial episode represented by the node can be derived. The label of a node has two forms including "X" and "_X," where X is an item. The node with *a label of "X"* represents a serial episode in which *X follows the label of the parent of the node*. Alternatively, the node with *a label of "_X"* represents a serial episode in which X *simultaneously occurs with the label of the parent of the node*. An example of using the prefix tree to represent the serial episodes is shown in Figure 2.

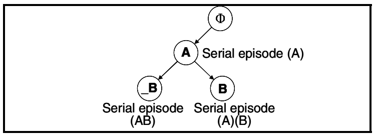


Fig. 2. An example of the prefix tree

Table 3. An example for Observation 1

Streams\Time unit	1	2	3	**4**
Stream 1	A	A	D	**A**
Stream 2	d	a	a	**b**

2.3 Observations of Serial Episode Rules with Time Lags

Several observations of the characteristics of SERs are described as follows, which are used in the methods detailed in Section 3. In order to simplify the display of the following discussion, the terms including $S^{+}X$ and S^{+}_X are first introduced. Given a serial episode S, $S^{+}X$ represents a serial episode *S followed by X*. The last itemset in $S^{+}X$ only contains an item, X. Alternatively, S^{+}_X represents another serial episode similar to S. The difference between S and S^{+}_X is that the last itemset in S^{+}_X consists of the last itemset in S and X.

Observation 1: *Given a serial episode S and a newly arrived item X which is one of the items in the current n-tuple event. If we want to know whether $S^{+}X$ has a new minimal occurrence, the last two minimal occurrences of S must be checked.*

Consider the data streams shown in Table 3. After the third time unit passed, the minimal occurrences of A are [1, 1] and [2, 2], that is, MO((A)) = {[1, 1], [2, 2]}. At the fourth time unit, since A and b are generated, MO((A)) becomes {[1, 1], [2, 2], [4, 4]}. We focus on the serial episode (A)(b). As can be seen, the new minimal occurrence [2, 4] of the serial episode (A)(b) is associated with the minimal occurrence [2, 2] of (A). If only the last minimal occurrence of A, [4, 4], is considered to join to the minimal occurrence [4, 4] of b, the minimal occurrence [2, 4] of (A)(b) will be ignored. Therefore, the last two minimal occurrences of (A) should be checked.

Observation 2: *Given two serial episode rules $X\rightarrow_L(AB)$ and $X\rightarrow_L A$, according to the Apriori property* [1], $\mathrm{supp}(X\rightarrow_L A) \geq \mathrm{supp}(X\rightarrow_L(AB))$. *Therefore, the rule $X\rightarrow_L(AB)$ is not significant if the rule $X\rightarrow_L A$ does not satisfy one of the minsup and the minconf.*

Observation 3: *Given two serial episode rules $(AB)\rightarrow_L(CD)$ and $A\rightarrow_L C$, obviously,* $\mathrm{supp}(A\rightarrow_L C) \geq \mathrm{supp}((AB)\rightarrow_L(CD))$. *Therefore, the rule $(AB)\rightarrow_L(CD)$ is not significant if* $\mathrm{supp}(A\rightarrow_L C) < \mathrm{supp}(AB) \times$ *minconf.*

Observation 4: *Given a serial episode rule $(A)(B)\rightarrow_L(CD)$ with a time bound T, the precursor of the rule has at most $T-1$ types, i.e. $(A)\rightarrow_p(B)$, where $0 < p < T$. While taking the time lag into account, the types of the rules are denoted as $(A)\rightarrow_p(B)\rightarrow_{L-p}(CD)$, where $0 < p < T$ and $L - p > 0$. Obviously, the support of $(A)\rightarrow_p(B)\rightarrow_{L-p}(CD)$ must be smaller than or equal to* $\mathrm{supp}(A\rightarrow_p B)$ *and* $\mathrm{supp}(B\rightarrow_{L-p}C)$. *Therefore,* $\mathrm{supp}((A)(B)\rightarrow_L C) \leq \sum_p \min(\mathrm{supp}(A\rightarrow_p B), \mathrm{supp}(B\rightarrow_{L-p}C))$. *Since* $\mathrm{supp}((A)(B)\rightarrow_L C) \geq \mathrm{supp}((A)(B)\rightarrow_L(CD))$, *we infer that the rule $(A)(B)\rightarrow_L(CD)$ is not significant if* $\sum_p \min(\mathrm{supp}(A\rightarrow_p B), \mathrm{supp}(B\rightarrow_{L-p}C)) < \mathrm{supp}(AB) \times$ *minconf.*

3 Methods

Two methods, LossyDL and TLT for finding significant SERs in the environment of multi- streams are proposed in this paper. Both of them use the prefix tree structure to store the information of the serial episodes. LossyDL keeps all valid minimal occurrences for each serial episode in the prefix tree. On the other hand, TLT only keeps 1) the last two valid minimal occurrences and the support of each serial episode in the prefix tree and 2) *the supports of the reduced rules in the additional tables,* to be detailed in Subsection 3.2. When the users want the SERs, to LossyDL, the valid minimal occurrences of any two serial episodes with enough supports are joined to check whether they satisfy the time lag. On the other hand, to TLT, any two serial episodes with enough supports will be combined into a *candidate rule* and then, the non-significant rules will be pruned by using the pruning strategies derived from Observations 2, 3, and 4. Since the prefix tree may increase as the time goes by, Lossy Counting [8] is used in these methods to avoid keeping the serial episodes with low supports. The detailed operations in LossyDL and TLT are respectively described in Subsections 3.1 and 3.2.

3.1 The LossyDL Method

The principle of LossyDL is to keep all the valid minimal occurrences of a serial episode under the *prefix tree* structure. Therefore, in addition to a label, each node (except the root node) of the prefix tree in LossyDL also keeps a *duration list* which is a set used to contain all the valid minimal occurrences in the MO set of its corresponding serial episode. The number of the valid minimal occurrences kept in the duration list is regarded as the support of the corresponding serial episode. Moreover, in order to avoid keeping too many serial episodes with low supports, the principle of Lossy Counting [8] is used in LossyDL. Given an error parameter ε, if the number of received n-tuple events is divisible by $\lceil 1/\varepsilon \rceil$, for each node, the oldest minimal occurrence is removed from its duration list. Moreover, the nodes with empty duration lists are removed from the prefix tree. When the users want the mining results, the SERs are generated from the prefix tree by comparing the duration lists of any two nodes with enough supports. The algorithms of LossyDL are shown in Figures 3 and 4 and explained as follows.

```
Input: multi-streams DSs, minsup, minconf, T, Lmax, and $\varepsilon$
Output: the prefix tree PT
Variable: a node b, the corresponding serial episode S, and its last two valid minimal occurrences $[t_{11}, t_{12}]$ and $[t_{21}, t_{22}]$
          in the duration list
1. Create a prefix tree PT with a root node containing $\varphi$
2. When an n-tuple event $R_i$ received from DSs at current time i
3.     for each item X in $R_i$
4.         for each node b in PT            // bottom-up traversing
5.             if ($i = t_{22}$)
6.                 if ($[t_{11}, i]$ is a valid minimal occurrence of $S^+X$)
7.                     Append it to the duration list of $S^+X$
8.                 if ($[t_{21}, t_{22}]$ is a valid minimal occurrence of $S^+\_X$)
9.                     Append it to the duration list of $S^+\_X$
10.            else
11.                if ($[t_{21}, i]$ is a valid minimal occurrence of $S^+X$)
12.                    Append it into the duration list of $S^+X$
13. If (the number of received records is divisible by $\lceil 1/\varepsilon \rceil$)
14.     Remove the oldest valid minimal occurrence for each node and remove the nodes with empty duration lists
```

Fig. 3. Maintaining the prefix tree in LossyDL

Initially, the root of the prefix tree is created. When a new n-tuple event arrives, all items in the n-tuple event are sequentially used to traverse the tree. The traversing order is *bottom-up*. The bottom-up traversing order means that *the nodes in the high levels are traversed before those in the low levels.* During the traversing process, the new serial episodes may be generated, that is, the new nodes may be created, and the new valid minimal occurrences for some serial episodes may be added. Suppose that $R(i)$ is an n-tuple event, where i is the current time unit and the item X is contained in $R(i)$. When X is processed, two cases need to be considered. Let d be a node corresponding to a serial episode S, the last two valid minimal occurrences of S stored in d are $[t_{11}, t_{12}]$ and $[t_{21}, t_{22}]$. When d is visited, (case 1) if t_{22} is equal to i, $[t_{21}, t_{22}]$ and $[t_{11}, i]$ become *the candidate occurrences* for S^+_X and S^+X, respectively; otherwise (case 2), $[t_{21}, i]$ is the candidate occurrence for S^+X. A candidate occurrence of S^+X (S^+_X) is recognized as valid and inserted into the duration list of the node corresponding to S^+X (S^+_X) if both of the last two valid minimal occurrences of S^+X (S^+_X) are not its proper subinterval and moreover, it satisfies the time bound T. Notice that if the node corresponding to S^+X (S^+_X) is not kept in the prefix tree, a new node with a label equal to X is created as a child of b.

Since LossyDL is rooted in Lossy Counting to reduce the memory usage, the supports of the serial episodes kept in the prefix tree and the significant SERs obtained by LossyDL must be equal to or less than their real supports. Therefore, when the users request the mining results, the duration lists of any two serial episodes with their supports equaling or exceeding $(minsup - \varepsilon) \times N$ are checked to see whether any valid minimal occurrences of the two serial episodes can be combined to contribute to the supports of a SER. Then, for each SER, R: $S_1 \rightarrow_L S_2$, satisfying the following two constraints are returned to avoid the *false dismissals*: 1) $\text{supp}(R) \geq (minsup - \varepsilon) \times N$ and 2) $(\text{supp}(R) + \varepsilon N) / \text{supp}(S_1) \geq minconf$.

An example for maintaining the prefix tree in the LossyDL method is shown in Figures 5 and 6. Suppose that the time bound is 3 and after some items were processed, the prefix tree is as the left-hand side in Figure 5. At time unit equal to 3, an item B contained in the n-tuple event is processed and the maintenance of the prefix tree is shown in Figure 6. The nodes in the prefix tree are traversed to be compared with B. The bottom-up traversing order is shown as the right-hand side of Figure 5. In Figure 6, notice that, [1, 3] is a minimal occurrence of (A)(B)(B), but [1, 3] is not a minimal occurrence of (A)(B) because [1, 2] is a proper subinterval of [1, 3].

```
Input: stream size $N$, $minsup$, $minconf$, $T$, $Lmax$, and $PT$
Output: significant SERs
Variable: a time lag, $lag$, nodes $N_1$, $N_2$ and their corresponding serial episodes $S_1$, $S_2$
1. for each two nodes $N_1$, $N_2$ in $PT$
2.   if ($\text{supp}(S_1) \geq (minsup - \varepsilon) \times N$ and $\text{supp}(S_2) \geq (minsup - \varepsilon) \times N$)
3.     Check the duration lists of $N_1$ and $N_2$
4.     for all $lag$ from 1 to $Lmax$
5.       Calculate $\text{supp}(S_1 \rightarrow_{lag} S_2)$
6.       if (($\text{supp}(S_1 \rightarrow_{lag} S_2) + \varepsilon N) / \text{supp}(S_1) \geq minconf$ and $\text{supp}(S_1 \rightarrow_{lag} S_2) \geq (minsup - \varepsilon) \times N$)
7.         Return $S_1 \rightarrow_{lag} S_2$
```

Fig. 4. Rule generation of LossyDL

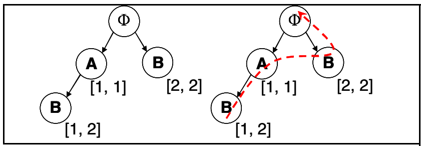
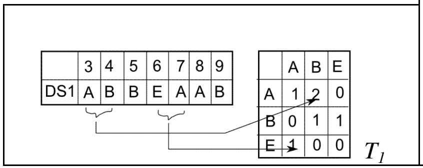


Fig. 5. The status of the prefix tree at time = 2 and the bottom-up traversing order at time = 3



	3	4	5	6	7	8	9
DS1	A	B	B	E	A	A	B

	A	B	E
A	1	2	0
B	0	1	1
E	1	0	0

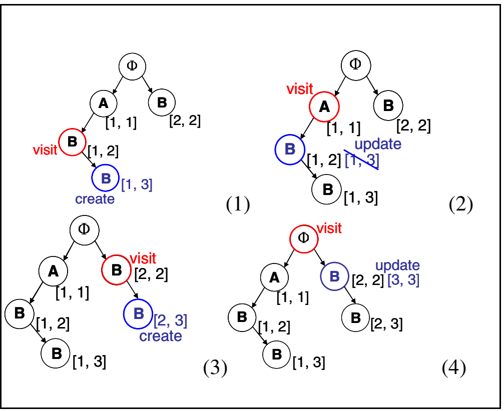


Fig. 7. An example of the time lag table T_1 **Fig. 6.** An example of maintaining the prefix tree after receiving an n-tuple event at time = 3

3.2 The TLT Method

Although Lossy Counting is used to reduce the required memory space for LossyDL, the duration list in each node of the prefix tree still needs much memory space. In order to avoid keeping almost all the valid minimal occurrences for a serial episode, another method named *TLT* is proposed. A more space-efficient structure, named *time lag table*, is used in TLT to summarize the time positions of the SERs. A time lag table is a two-dimension table used to store the supports of *the reduced SERs* which are SERs with their precursors and successors both containing a single item. For example, R: $A \rightarrow_L B$ is a reduced SER, if A and B are items. The row and column of a time lag table represent the items and moreover, a time lag table with a label of L keeps the supports of the reduced SERs with a time lag of L, e.g. the entry $T_L(A, B)$ in the table T_L keeps the support of R: $A \rightarrow_L B$. An example of the time lag table T_1 is shown in Figure 7.

Instead of keeping almost the whole duration list, each node of the prefix tree in TLT keeps only the last two valid minimal occurrences and the support for its corresponding serial episode. Moreover, since the time lag of the significant SERs is at most $Lmax$, such as $S_1 \rightarrow_{Lmax} S_2$, some reduced SERs with a time lag at most $Lmax + T - 1$ (from the first itemset of S_1 to the last itemset of S_2) may exist. Therefore, in addition to the prefix tree, TLT will also keep $Lmax + T - 1$ additional time lag tables, named T_1, T_2, …, $T_{Lmax+T-1}$, which respectively store the supports of the reduced SERs with a time lag equal to 1, 2, …, $Lmax + T - 1$.

The maintenance of the prefix tree in TLT is similar to that in LossyDL, and moreover, Lossy Counting is also employed in removing the nodes with low supports in TLT. The last two valid minimal occurrences kept in the node are used as in LossyDL to check whether a new valid minimal occurrence for its child node occurs or not. If it occurs, the support of its corresponding serial episode kept in the node of the prefix tree is increased by one and moreover, the original last two minimal occurrences of its corresponding serial episode will also be updated. When the number of the received n-tuple events of the data streams is divisible by $\lceil 1/\varepsilon \rceil$, TLT decreases the support of

```
Input: $N$, $minsup$, $minconf$, $\varepsilon$, $Lmax$, $PT$, and a set of time lag tables $T_i$, $i$ = 1 to $Lmax + T - 1$
Output: significant SERs
Variable: a time lag, $lag$, two nodes $N_1$, $N_2$, their corresponding serial episodes $S_1$, $S_2$, the first itemset of $S_1$,
        $I_{11}$, the first itemset of $S_2$, $I_{21}$, the second itemset of $S_1$, $I_{12}$, and a temporary variable $TempCount$
        for estimating the support boundary
1. for each two nodes $N_1$, $N_2$ in $PT$
2.     if (supp($S_1$) $\geq$ ($minsup - \varepsilon$) $\times N$ and supp($S_2$) $\geq$ ($minsup - \varepsilon$) $\times N$)
3.         for all $lag$ from 1 to $Lmax$
4.             for each item $I$ in $I_{11}$ and $J$ in $I_{21}$
5.                 Obtain supp($I \rightarrow_{lag} J$) from $T_{lag}$
6.                 if (supp($I \rightarrow_{lag} J$) < ($minsup - \varepsilon$) $\times N$)
7.                     Prune $S_1 \rightarrow_{lag} S_2$          // from Observation 2
8.                 if (supp($I \rightarrow_{lag} J$) < supp($S_1$) $\times$ $minconf$)
9.                     Prune $S_1 \rightarrow_{lag} S_2$          // from Observation 3
10.                for each item $X$ in $I_{12}$
11.                    $TempCount = 0$
12.                    for each $p$ s.t. $0 < p < T$ and $lag - p > 0$
13.                        $TempCount = TempCount$ + min(supp($I \rightarrow_p X$), supp($X \rightarrow_{lag-p} J$))
14.                    if ($TempCount$ < supp($S_1$) $\times$ $minconf$)
15.                        Prune $S_1 \rightarrow_{lag} S_2$      // from Observation 4
16.                if ($S_1 \rightarrow_{lag} S_2$ cannot be pruned)
17.                    Return $S_1 \rightarrow_{lag} S_2$
```

Fig. 8. Rule generation of TLT

each serial episode by one and remove the nodes with supports = 0 from the prefix tree. In addition to the prefix tree, by keeping the last $Lmax + T - 1$ n-tuple events of the stream, the time lag table T_1, T_2, …, $T_{Lmax+T-1}$ can be updated when a new n-tuple event is generated.

When the users want the significant SERs, any two serial episodes with enough supports, that is, greater than or equal to ($minsup - \varepsilon$) × N, will form a *candidate SER*. Then, the candidate SER is checked to see whether it satisfies the pruning rules derived from Observations 2, 3, and 4. That is, given a candidate SER R: $S_1 \rightarrow_{lag} S_2$, where the time lag = 1 to $Lmax$, and supp(S_1) and supp(S_2) both equal or exceed ($minsup - \varepsilon$) × N, R is returned to the users if it pass all of the pruning rules shown in Figure 8. The rule generation algorithm of TLT is described in Figure 8.

4 Performance Evaluations

This paper first addresses the problem of finding significant serial episode rules with time lags over multi-streams. Therefore, to evaluate the effectiveness of LossyDL and TLT, a series of experiments on real data are performed and the experiment results are provided in this section. All experiments are performed on a PC with the Intel Celeron(R) 2.0GHz CPU, 1GB of memory, and under the Windows XP operating system. The *memory* space utilization, the *updating time* for the summary, the *mining time* while requesting the significant SERs, and the *precision* are used to evaluate the two proposed methods. The error parameter ε is set to 0.1 × *minsup*, as usually done in the other papers [8]. The maximum time lag, *Lmax*, is set to 10.

Two datasets tested in the experiments are described as follows: 1) Dryness and Climate Indices, denoted as *PDOMEI*: the data set contains several dryness and climate indices which are derived by experts and usually used to predict droughts. For example, several serial episode rules related to these indices are discovered in [14]. We select four indices from these indices, including Standardized Precipitation Index (SPI1, SPI3) [17], Pacific Decadal Oscillation Index (PDO) [16], and Multivariate ENSO Index (MEI) [15]. *Therefore, there are four data streams with a length of 124, including SPI1, SPI3, PDO, and MEI in this data set.* Moreover, the values of these indices are divided into seven discrete categories [14], making *the total types of items are 28*. 2) Another data set is "Twin Cities' Traffic Data Archive," denoted as *Traffic*: the data set obtained from TDRL [18] is Twin Cities' traffic data near the 50th St. during the first week of February, 2006. *It contains three data streams with a length of 1440.* Each stream is generated by a sensor periodically reporting the occupancy and flow. A pair of the occupancy and the flow is divided into discrete classes and regarded as items. *There are 55 distinct items in this dataset.*

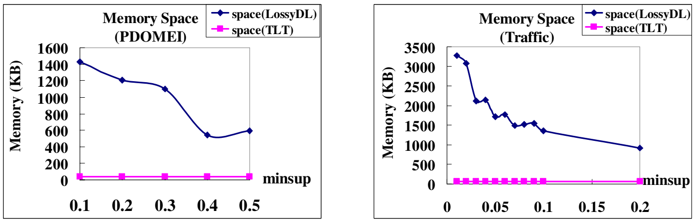


Fig. 9. Comparison of memory space between LossyDL and TLT

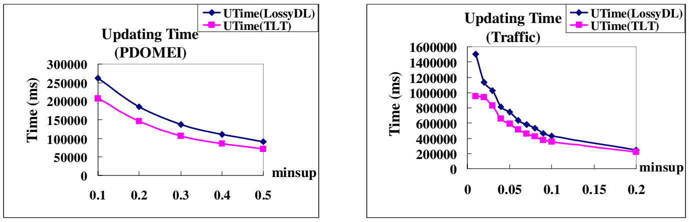
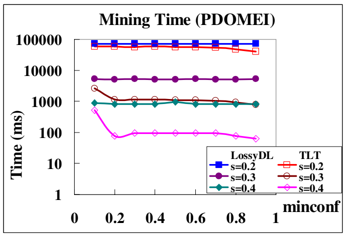


Fig. 10. Comparison of updating time between LossyDL and TLT

Since Lossy Counting is applied to both of the two methods, the required memory space of the two methods will increase as time goes by and drastically decrease when the sizes of the streams are divisible by $\lceil 1/\varepsilon \rceil$. Therefore, we concentrate on the maximal memory space used for the information of the occurrences of the serial episodes that is, the duration lists in LossyDL and the time lag tables in TLT, during the procedures, which is shown in Figure 9. As can be seen, the lower the *minsup* is, the more memory space the LossyDL method requires. This is because the error parameter ε is set to 0.1 × *minsup* and as ε decreases, the memory space required in LossyDL will increase. Alternatively, the memory space used in TLT is almost fixed under any values of the *minsup* because the sizes of the time lag tables are decided according to the number of items. Moreover, from Figure 9, it can find that the memory space used in TLT is substantially less than that in LossyDL. For example, in the Traffic dataset, the memory space used in TLT is around 64 KB, which is about 2% ~ 7% times that in LossyDL.

The Running time in the experiments can be divided into two types. One for *updating* such structures as the prefix tree, the duration lists, and the time lag tables and another one is for *mining* all the significant SERs from the structures. As shown in Figure 10, the updating time of TLT is less than that of LossyDL because increasing the counters in the time lag tables of TLT takes less time than frequently inserting/deleting the valid minimal occurrences into/from the duration lists of LossyDL. Moreover, the updating time of LossyDL and TLT roughly decreases as the *minsup* increases. Again, this is because ε is set to 0.1 × *minsup*. As ε (*minsup*) decreases, the size of the prefix trees will increase, thus making the updating time of two methods to be increased. In the Traffic dataset, the average processing time of LossyDL for each transaction is about 1.04 second. In other words, our methods can process the streaming data on time if the average arriving rate is less than 1 records/sec. This is acceptable in the traffic monitoring system because the sampling rates of the sensors cannot be set too high due to power consumption.



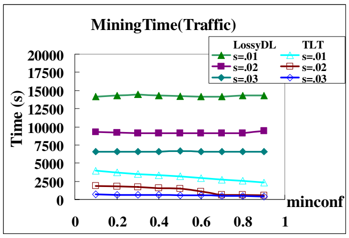
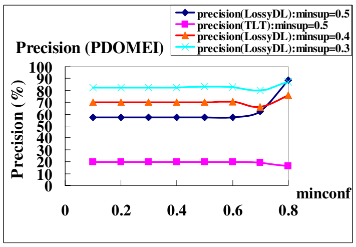


Fig. 11. Comparison of mining time between LossyDL and TLT



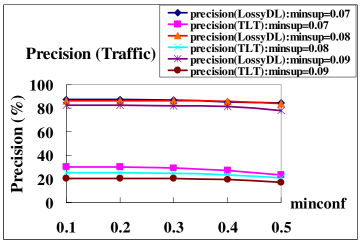


Fig. 12. Comparison of precision between LossyDL and TLT

The mining time of the two methods are shown in Figure 11. Since the lower the *minsup* (s) is, the more the serial episodes need to be checked, the mining time of the two methods will increase as the *minsup* decreases. Moreover, as can be seen, the mining time of the two methods will lightly decrease as *minconf* increases but the effects of *minconf* on the mining time seems to be marginal. This is because the mining time is almost decided according to the number of the serial episodes with enough supports. From Figure 11, it can also find that the mining time of TLT is lower than that of LossyDL because in LossyDL, the duration lists of two serial episodes need to be joined but to TLT, we only need to check the values in the time lag tables.

Since both of LossyDL and TLT are false-positive oriented approaches, their recall rates are equal to 100%. Therefore, we concentrate on the precision rates of the two methods, which are shown in Figure 12. The LossyDL method has the higher precision rates than the TLT method on both of the two datasets. This is because almost all the valid minimal occurrences are kept in the duration lists of LossyDL, making the support and confidence of a SER be precisely calculated by comparing the duration lists of the serial episodes.

5 Conclusions

In this paper, we address the problem of finding significant serial episode rules with time lags over multiple data streams and propose two methods, LossyDL and TLT, to solve the problem. The prefix tree structure is used to store the information of the support for each serial episode in the two methods. In order to limit the memory space for the prefix tree, the principle of Lossy Counting is integrated into the maintenance of the prefix tree. Since the number of the events (items) is much less than the number of the serial episodes, TLT is more space-efficient. Alternatively, since LossyDL keeps almost all the valid minimal occurrences for a serial episode, the precision of LossyDL is higher than that of TLT. In the near future, we will combine these two methods into a hybrid method to investigate the balance between the memory space used and the precision. On the other hand, we will also manage to extend the approach to mine general episode rules with time lags over multiple data streams.

References

[1] Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases. In: VLDB 1994, pp. 487–499 (1994)

[2] Hall, F.L.: Traffic stream characteristics. In: Traffic Flow Theory, U.S. Federal Highway Administration (1996)

[3] Harms, S.K., Deogun, J.S.: Sequential Association Rule Mining with Time Lags. Journal of Intelligent Information Systems 22(1), 7–22 (2004)

[4] Harms, S.K., Deogun, J., Saquer, J., Tadesse, T.: Discovering representative episodal association rules from event sequences using frequent closed episode sets and event constraints. In: ICDM 2001, pp. 603–606 (2001)

[5] Harms, S.K., Deogun, J., Tadesse, T.: Discovering Sequential Association Rules with Constraints and Time Lags in Multiple Sequences. In: ISMIS 2002, pp. 432–441 (2002)

[6] Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, San Francisco (2001)

[7] Liu, Y., Choudhary, A., Zhou, J., Khokhar, A.: A Scalable Distributed Stream Mining System for Highway Traffic Data. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD 2006. LNCS (LNAI), vol. 4213, pp. 309–321. Springer, Heidelberg (2006)

[8] Manku, G.S., Motwani, R.: Approximate Frequency Counts Over Data Streams. In: VLDB 2002, pp. 346–357 (2002)

[9] Laxman, S., Sastry, P.S., Unnikrishnan, K.P.: A Fast algorithm for Finding Frequent Episodes in Event Streams. In: KDD 2007, pp. 410–419 (2007)
[10] Mannila, H., Toivonen, H.: Discovering Generalized Episodes using Minimal Occurrence. In: KDD 1996, pp. 146–151 (1996)
[11] Mannila, H., Toivonen, H., Verkamo, A.I.: Discovery of Frequent Episodes in Event Sequences. Data Mining and Knowledge Discovery 1(3), 259–289 (1997)
[12] Mannila, H., Verkamo, A.I., Toivonen, H.: Discovering Frequent Episodes in Sequences. In: KDD 1995, pp. 210–215 (1995)
[13] Mielikäinen, T.: Discovery of Serial Episodes from Streams of Events. In: SSDBM 2004, p. 447 (2004)
[14] Tadesse, T., Wilhite, D.A., Hayes, M.J.: Discovering Associations between Climatic and Oceanic Parameters to Monitor Drought in Nebraska Using Data-Mining Techniques. Journal of Climate 18(10), 1541–1550 (2005)
[15] Multivariate ENSO Index (MEI), `http://www.cdc.noaa.gov/people/klaus.wolter/MEI/`
[16] Pacific Decadal Oscillation (PDO) Index, `http://jisao.washington.edu/data_sets/pdo/`
[17] Standardized Precipitation Index, `http://www.drought.unl.edu/monitor/archivedspi.htm`
[18] TDRL, `http://tdrl1.d.umn.edu/services.htm`