HKU CS Tech Report TR-2007-01

Mining time-delayed associations from discrete event datasets

K. K. Loo Ben Kao

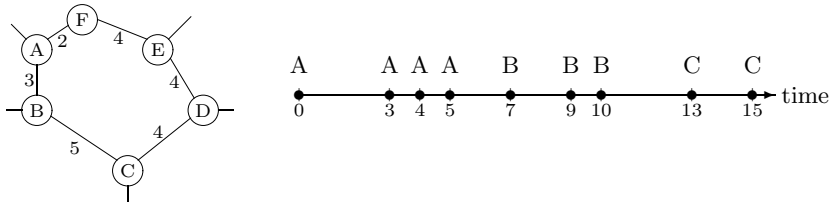
Department of Computer Science,
The University of Hong Kong, Hong Kong.
{kkloo, kao}@cs.hku.hk

Abstract. We study the problem of finding time-delayed associations among types of events from an event dataset. We present a baseline algorithm for the problem. We analyse the algorithm and identify two methods for improving efficiency. First, we propose pruning strategies that can effectively reduce the search space for frequent time-delayed associations. Second, we propose the breadth-first* (BF*) candidate-generation order. We show that BF*, when coupled with the least-recently-used cache replacement strategy, provides a significant saving in I/O cost. Experiment result show that combining the two methods results in a very efficient algorithm for solving the time-delayed association problem.

1 Introduction

Developments in sensor network technology have attracted vast amounts of research interest in recent years [1–3, 5–8]. One of the research topics related to sensor networks is to find correlations among the behaviour of different sensors. Such correlations are useful because they provide insight into the system monitored and can help to make decisions on what action to be taken when certain sensor value patterns are observed.

Consider a network monitoring system designed for collecting traffic data of a network of switches and links as shown in Figure 1(a). In the figure, nodes represent switches, whereas edges are links connecting switches. Under normal conditions, the time needed to pass through a link is represented by the number



(a) Network topology (b) Alerts issued by a network monitoring system

Fig. 1. An example showing a network monitoring system

on the corresponding edge. When the traffic at Switch X exceeds certain capacity, a congestion alert is raised. Figure 1(b) shows an example of alert signals.

By analysing an alert sequence, one may discover interesting correlations among different types of alerts. For example, one may find that a Switch-A alert is likely followed by a Switch-B alert within a certain time period. One may also find that if such an A-B pattern occurs, a Switch-C alert is likely to occur soon after. Such association information would be useful, for example, in congestion prediction, which could be applied to intelligent traffic redirection strategies.

In this paper, we model correlations of events in the form of *time-delayed associations*. In our model, an event e is a pair (E_e, t_e) where E_e is its type and t_e is the time at which e occurs. For example, with our network switch application, $(A, 5)$ represents the event that a Switch-A alert (type) occurs at time 5. We are interested in associations among events whose occurrences are time-constrained. A time-delayed association thus takes the form $I \xrightarrow{[u,v]} J$, where I, J are event types and u, v are two time values such that $0 < u \leq v$. The association captures the observation that when an event i of type I occurs at some time t_i, it is likely that an event j of type J occurs at time t_j such that $t_i + u \leq t_j \leq t_i + v$. If such an event j exists, event i is said to *match* the association and we call event j a *consequence* of i w.r.t. the association.

Associations can be "chained" to form longer associations that involve more than two event types. Chained associations are important because they can help detecting risk of unfavourable conditions early so that precautionary actions can be taken. Here, we can treat an association $I \xrightarrow{[u,v]} J$ as a *complex event type* $\mathbb{I}$. An association between a complex event type $\mathbb{I}$ and an ordinary event type K has the form $\mathbb{I} \xrightarrow{[u,v]} K$. Intuitively, such an association refers to the observation that if an event of type I occurs and is followed by one or more event of type J within a certain constrained time period, then at least one of the type-J consequences is likely followed by a type-K event within a constrained time period.

In [4], Mannila et al proposed the concept of *episode*, which is an ordered list of events. They proposed the use of minimal occurrences to find episode rules in order to capture temporal relationships between different event types. A *minimal occurrence* of an episode is a time interval $[t_s, t_e)$ such that the episode (an event sequence) occurs in the interval but not in any proper sub-interval of $[t_s, t_e)$. Let α and β be two episodes. An *episode rule* has the form $\alpha[w_1] \Rightarrow \beta[w_2]$, which specifies the following relationship: "if α has a minimal occurrence in the interval $[t_s, t_e)$ such that $t_e - t_s \leq w_1$, then there is a minimal occurrence of β in $[t_s, t'_e)$ such that $t'_e - t_s \leq w_2$". Let us consider Figure 1(b) as an example. Let $\alpha = \{A \rightarrow B\}$ be an episode. From the figure, we see that a minimal occurrence of α is $[5, 8)$. Similarly, let $\beta = \{A \rightarrow B \rightarrow C\}$ be another episode, which has a minimal occurrence of $[5, 14)$. Let $w_1 = 5$ and $w_2 = 10$, the minimal occurrence of α in $[5, 8)$ matches the episode rule $\alpha[5] \Rightarrow \beta[10]$ because there is a minimal occurrence of β within 10 time units since time 5. The goal is to discover episodes and episode rules that occur frequently in the data sequence.

In a sense, our problem is similar to episode discovery in that we are looking for frequently occurring event sequences. However, we remark that the use of

minimal occurrence to define the occurrence of an episode might in some cases fail to reflect the strength of an association. As an example, consider Figure 1(b) again. It is possible that the three type-B events that occur at time $t = 7, 8$ and 10 are "triggered" respectively by the three preceding A's that occur at $t = 3, 4$ and 5. Hence, the association, namely, $A \rightarrow B$ has occurred three times. However, only one period ($[5, 8)$) is qualified as a minimal occurrence of the episode $A \rightarrow B$. In other words, out of all 4 occurrences of A in the figure, there is only 1 occurrence of the episode $A \rightarrow B$, even though 3 of the A's have triggered B.

A major difference between our definition of time-delayed association and the episode's minimal occurrence approach is that, under our approach, every event that matches an association counts towards the association's support. This fairly reflects the strength of correlations among event types. Also, our definition allows the specification of a timing constraint $[u, v]$ between successive event types in an association. This helps removing those associations that are not interesting. For example, if it takes at least 2 time units for a packet to pass through a switch, then any type-B alert that occurs 1 time unit after a type-A alert should not count towards the association $A \rightarrow B$ (See Figure 1). We can thus use the timing constraint to filter false matches. The minimal occurrence approach used in episode does not offer such flexibility.

A straightforward approach to finding all frequent associations is to generate and verify them incrementally. First, we construct all possible length-2 associations $X \rightarrow Y$, where X and Y are any event types in the data sequence. We then scan the data sequence to count the associations' supports, that is, the number of event occurrences that match the associations. Those associations with high supports are considered frequent. Next, for each frequent association $X \rightarrow Y$, we consider every possible length-3 extension, i.e., we append every event type Z to $X \rightarrow Y$ forming $(X \rightarrow Y) \rightarrow Z$. The support of those length-3 associations are counted and those that are frequent will be used to generate length-4 associations, and so on. The process stops when we can no longer obtain any new frequent sequences. In Section 3 we will show how the above conceptual procedure is implemented in practice. In particular, we show how the computational problem is reduced to a large number of table joins. We call this algorithm the baseline algorithm.

The baseline algorithm is not particularly efficient. We address two methods to improve the algorithm's efficiency. First, the baseline algorithm extends a frequent association $\mathbb{I} \rightarrow Y$ by considering all possible extensions $(\mathbb{I} \rightarrow Y) \rightarrow Z$. Many of such extended associations could be infrequent and the effort spent on counting their supports is wasted. A better strategy is to estimate upper bounds of the associations' supports and prune away those that cannot meet the support requirement. Second, as we will explain later, the baseline algorithm generates $(\mathbb{I} \rightarrow Y) \rightarrow Z$ by considering two sub-associations, namely, $\mathbb{I} \rightarrow Y$ and $Y \rightarrow Z$. In the process, two tables that are associated with the two sub-associations are retrieved and joined. Since the number of such associations and their associated tables is huge, the tables will have to be disk-resident. A caching strategy that

can avoid disk accesses as much as possible would thus have a big impact on the algorithm's performance. In this paper we study an interesting coupling effect of a caching strategy and an association-generation order.

The rest of the paper is structured as follows. We give a formal definition of our problem in Section 2. In Section 3, we discuss some properties of time-delayed associations and propose a baseline algorithm for the problem. In Section 4, we discuss the pruning strategies and the caching strategies. We present experiment results in Section 5 and conclude the paper in Section 6.

2 Problem definition

In this section we define the problem of finding time-delayed associations from event datasets. We define an event e as a 2-tuple (E_e, t_e) where E_e is the event type and t_e is the time e occurs. Let $\mathcal{D}$ denote an event dataset and $\mathcal{E}$ denote the set of all event types that appear in $\mathcal{D}$. We define a time-delayed association as a relation between two event types $I, J \in \mathcal{E}$ of the form $I \xrightarrow{[u,v]} J$. We call I the *triggering event type* and J the *consequence event type* of the association. Intuitively, $I \xrightarrow{[u,v]} J$ captures the observation that if an event i such that $E_i = I$ occurs at time t_i, then it is "likely" that there exists an event j so that $E_j = J$ and $t_i+u \leq t_j \leq t_i+v$, where $v \geq u > 0$. The likelihood is given by the *confidence* of the association, whereas the statistical significance of an association is given by its *support*. We will define support and confidence shortly.

For an association $r = I \xrightarrow{[u,v]} J$, an event i is called a *match* of r (or i *matches* r) if $E_i = I$ and there exists another event j such that $E_j = J$ and $t_i + u \leq t_j \leq t_i + v$. In other words, event i matches r if it leads to a positive example of the association. The event j here is called a *consequence* of r. Note that a match can correspond to more than one consequence, and vice versa. We use the notations M_r to denote the set of all matches of r, $q_{r,i}$ to denote the set of all consequences that correspond to a match i of r and $m_{r,j}$ to denote the set of all matches of r that correspond to a consequence j. Also, we define $Q_r = \bigcup q_{r,i}\ \forall i \in M_r$. That is, Q_r is the set of all events that are consequences of r. The *support* of an association r is defined as the ratio of the number of matching events to the total number of events (i.e., $\frac{|M_r|}{|\mathcal{D}|}$). The *confidence* of r is defined as the fraction $\frac{|M_r|}{|\mathcal{D}_I|}$, where $\mathcal{D}_I$ is the set of all type-I events in $\mathcal{D}$. We use the notations $supp(r)$ and $conf(r)$ to represent the support and confidence of r, respectively. Finally, the *length* of an association r, denoted by $len(r)$, is the number of event types contained in r.

We can extend the definition to relate more than two event types. Consider an association $r = I \xrightarrow{[u,v]} J$ as a *complex* event type $\mathbb{I}$, an association between $\mathbb{I}$ and an ordinary event type K is of the form $r' = \mathbb{I} \xrightarrow{[u,v]} K$. Here, $\mathbb{I}$ is the triggering event type and K is the consequence event type. Intuitively, the association says that if an event of type I is followed by one or more event of type J within certain time constraints u and v, then at least one of the J's is likely to be followed by a type K event. A match for the association r' is a match i for r such that, for some j where $j \in q_{r,i}$, there exists an event k such that $E_k = K$

Symbol	Meaning
$\mathcal{D}$	The event dataset
$\mathcal{E}$	The set of all event types
$\mathcal{D}_I$	The set of all type-I events
e, E_e, t_e	An event, the event type and time of the event e
$I, J, ...$ (font face)	(Ordinary) event types
$\mathbb{I}, \mathbb{J}, ...$ (font face)	Complex event types
r	A time-delayed association
u, v	Time constraints of a time-delayed association
M_r, Q_r	The set of all matches and consequences of the association r
$q_{r,x}$	All consequences corresponding to the match x w.r.t. r
$m_{r,x}$	All matches corresponding to the consequence x w.r.t. r
$supp(r), conf(r), len(r)$	The support, confidence and length of r
ρ_s, ρ_c	The support and confidence thresholds

Table 1. Symbols and notations

and $t_j + u \leq t_k \leq t_j + v$. We say that event k is a consequence of event i w.r.t. the association r'. The support of r' is defined as the fraction of events in D that match r' (i.e., $\frac{|M_{r'}|}{|\mathcal{D}|}$). The confidence of r' is defined as the ratio of the number of events that match r' to the number of events that match r (i.e., $\frac{|M_{r'}|}{|M_r|}$). Given two user-specified thresholds ρ_s and ρ_c and a timing constraint $[u, v]$, the problem of mining time-delayed associations is to find all associations r such that $supp(r) \geq \rho_s$ and $conf(r) \geq \rho_c$.

In our model, we use the same timing constraint $[u, v]$ for all associations. Therefore, we will use a plain arrow "$\rightarrow$" instead of "$\xrightarrow{[u,v]}$" in the rest of the paper when the timing constraint is clear from the context or is unimportant. We summarize in Table 1 the symbols and notations used in the paper.

3 Baseline algorithm for finding frequent time-delayed associations

We start this section with two properties based on which the baseline algorithm is designed.

Property 1: If $|\mathcal{D}_I|$, i.e., the number of occurrences of type I events, is smaller than $\rho_s \times |\mathcal{D}|$, then any association of the form $r = I \rightarrow J$ must not be frequent.

Proof: By definition, the set of matches of r must be a subset of $\mathcal{D}_I$. Hence, $|M_r| \leq |\mathcal{D}_I| < \rho_s \times |\mathcal{D}|$. □

Property 2: For any associations x and $y = x \rightarrow K$, we have $supp(x) \geq supp(y)$.

Proof: By definition, $M_x \supseteq M_y$. Hence, $supp(x) \geq supp(y)$. □

From Property 2, we know that if association y is frequent, so is x. In other words, if an association x is not frequent, we do not need to consider any associations that are *right* extensions of x. The baseline algorithm (Figure 2) generates associations based on this observation.

```
algorithm BASELINE
1)  L := ∅;  C := ∅  n = 2;
2)  F := {all frequent event types}
3)  foreach I ∈ F, J ∈ E do
4)      C := C ∪ {I → J}
5)  end-for
6)  while C ≠ ∅ do
7)      C_n := C;  C := ∅
8)      foreach r ∈ C_n do
9)          if r = 𝕀 → J is frequent do
10)             L := L ∪ {r}
11)             C := C ∪ {(𝕀 → J) → K} ∀ K ∈ E
12)         end-if
13)     end-for
14)     n := n + 1
15) end-while
16) return L
```

Fig. 2. Algorithm BASELINE

$A \xrightarrow{[3,5]} B$

m	q
3	7
4	7
4	9
5	9
5	10

(a)

$B \xrightarrow{[3,5]} C$

m	q
9	13
10	13
10	15

(b)

$(A \xrightarrow{[3,5]} B) \xrightarrow{[3,5]} C$

m	q
4	13
5	13
5	15

(c)

Fig. 3. M-Q mappings for various time-delayed associations

First, the algorithm collects into the set $\mathcal{F}$ all frequent event types (Line 2). The algorithm then maintains two sets: $\mathcal{C}$ is a set of candidate associations which are potentially frequent, and $\mathcal{L}$ is a set that contains all frequent associations discovered so far. The set $\mathcal{C}$ is initialized to contain all possible length-2 associations (Lines 3-5). The support of a candidate association r is determined. (We will discuss how to compute the support shortly.) If r is verified to be frequent, we extend r to $r \rightarrow K$ for each event type $K \in \mathcal{E}$ and add them to $\mathcal{C}$. The algorithm terminates when all candidates are evaluated and no new candidates can be generated.

Now, we discuss how to compute an association's support. Consider an association $r = (\mathbb{I} \rightarrow J) \rightarrow K$. By definition, an event i is a match of r if i is a match of $\mathbb{I} \rightarrow J$ and for some consequence j of i, there exists an event k such that $E_k = K$ and $t_j + u \leq t_k \leq t_j + v$. In other words, the event j is both a consequence of $r_1 = \mathbb{I} \rightarrow J$ and a match of $r_2 = J \rightarrow K$. The set of all such events is given by $Q_{r_1} \cap M_{r_2}$. Let us call this set the *connecting set* between r_1 and r_2. We have the following properties.

Property 3: For any event $j \in Q_{r_1} \cap M_{r_2}$, every $i \in m_{r_1,j}$ is a match of r and every $k \in q_{r_2,j}$ is a consequence of event i w.r.t. r for every $i \in m_{r_1,j}$.

Proof: By definition, every $i \in m_{r_1,j}$ is a match of r because there exists k such that $t_j + u \leq t_k \leq t_j + v$. Indeed, every $k \in q_{r_2,j}$ fulfils this requirement. Hence, every $k \in q_{r_2,j}$ is a consequence of i w.r.t. r for every $i \in m_{r_1,j}$. □

Property 4: For any event $j \notin Q_{r_1} \cap M_{r_2}$, $\nexists i \in m_{r_1,j}, k \in q_{r_2,j}$ such that i is a match and k is a consequence of i w.r.t. r.

Proof: (i) Any event j not in Q_{r_1} cannot be a consequence of any $i \in M_{r_1}$ for the association r_1. So $m_{r_1,j} = \emptyset$. (ii) For any event $j \in Q_{r_1}$ but not in M_{r_2}, $q_{r_2,j} = \emptyset$. □

Given an association r and a match i of r, we can determine all consequences j of i w.r.t. r. If we put all these match-consequence i-j pairs in a relation, we obtain an M-Q mapping of the association r. Let us consider the network switch

example again (Figure 1). If $r = A \xrightarrow{[3,5]} B$, then the matching type-A event at $t = 4$ leads to two consequence type-B events at $t = 7$ and 9. Hence the tuples $\langle 4, 7\rangle$ and $\langle 4, 9\rangle$ are in the M-Q mapping of the association. Figures 3(a) and 3(b) show the M-Q mappings of the associations $A \xrightarrow{[3,5]} B$ and $B \xrightarrow{[3,5]} C$, respectively.

By Property 3, given the M-Q mappings for r_1 and r_2, denoted respectively by T_1 and T_2, we can derive the M-Q mapping of r by (1) performing an equi-join on T_1 and T_2 so that $T_1.q = T_2.m$, where the join result is projected on $T_1.m$ and $T_2.q$. removing the duplicate tuples in the mapping. Figure 3(c) shows the resultant M-Q mapping of $(A \xrightarrow{[3,5]} B) \xrightarrow{[3,5]} C$. The mapping is computed by joining the M-Q mappings shown in Figures 3(a) and 3(b).

Given the M-Q mapping of an association r, the support $supp(r)$ can be computed by counting the number of distinct elements in the m column. The confidence of r can then be easily determined by the supports of its sub-associations. In this paper we focus on computing the supports of associations and extracting those that are frequent.

4 Improving the baseline algorithm

The baseline algorithm described in the previous section offers a method to find frequent time-delayed associations. In this section, we propose methods to improve the efficiency of the algorithm by investigating two areas, namely the search space for frequent associations and the handling of intermediate results.

4.1 Pruning strategy

Frequent itemset mining from market-basket data is a well-studied data mining problem. Most algorithms for the problem are based on the apriori property (i.e., if an itemset is frequent, all its subsets must also be frequent), which serves as a pruning strategy that trims the search space for frequent itemsets and avoids unnecessary computation. In our problem, however, the only base for trimming the search space for frequent time-delayed associations is described in Properties 1 and 2 (Section 3), which is why the baseline algorithm takes all possible extensions of a frequent association as candidates. Nevertheless, if we can develop methods that quickly determine whether a candidate can be frequent without need to join the M-Q mapping, such methods can be regarded as alternative pruning strategies and substantially improve the efficiency of the baseline algorithm.

Multiplicity of consequences With respect to a time-delayed association, an event can be a consequence of one or more matches. We define, for the association r, the *multiplicity* of a consequence q as the number of matches such that q is a consequence. Getting the multiplicity for each distinct consequence is easy. By sorting the M-Q mapping of an association r by the consequence column, rows for a particular consequence are arranged consecutively. Then, the multiplicity of q can be obtained by counting the number of consecutive rows representing q.

m	q
1	3
4	7
5	7
6	7
8	10
9	10

q	Multi
3	1
7	3
10	2

(a) M-Q mapping and multiplicity of consequences for $I \rightarrow J$

seg	ST Vector
1	$\langle 1 \rangle$
2	$\langle 3, 5 \rangle$

(b) SectTop vectors for $I \rightarrow J$

m	q
7	9
11	13

seg	num_matches
1	0
2	1
3	1

(c) M-Q mapping and number of matches per segment for $J \rightarrow K$

Fig. 4. Multiplicity of consequences and SectTop

Figure 4(a) shows an example. Suppose the table on the left is the M-Q mapping for an association $I \rightarrow J$, the table on the right lists the multiplicity for each distinct consequence.

Given a frequent association r, the multiplicity of consequences of r can help us determine whether a candidate obtained by extending r can be frequent. Next, we propose two methods, namely, GlobalK and SectTop, for identifying candidates that cannot be frequent without actually finding the M-Q mappings of the candidates.

GlobalK The idea of *GlobalK* is as follows. Recall that, for an association $r = I \rightarrow J$, the multiplicity of a consequence q is the number of matches such that q is a consequence. Thus, the sum of the multiplicity values of n consequences gives an upper bound on the number of corresponding matches w.r.t. r (the sum is an upper bound because a match can correspond to more than 1 of the n consequences). Now, suppose r is frequent. When evaluating whether a candidate $(I \rightarrow J) \rightarrow K$, which is an extension of r, is frequent, we find its M-Q mapping by firstly getting the connecting set (as discussed in Section 3). If n of the consequences of r end up in the connecting set, the sum of their multiplicity values gives an upper bound on the number of matches in the M-Q mapping of $(I \rightarrow J) \rightarrow K$. If the upper bound is smaller than the support threshold, then $(I \rightarrow J) \rightarrow K$ cannot be frequent, and vice versa.

Here, for a time-delayed association r, if k is the minimum number of consequences such that sum of their multiplicity values is not smaller than the support threshold, then we call k the *GlobalK threshold* for r. For example, in Figure 4(a), if the support threshold is 4 matches, then the GlobalK threshold for $I \rightarrow J$ is 2. Intuitively, for a candidate $(I \rightarrow J) \rightarrow K$ to be frequent, the number of matches for $J \rightarrow K$ must not be smaller than the GlobalK threshold of $I \rightarrow J$.

To compute the GlobalK threshold for an association, one can first inversely sort the multiplicities of all consequences for r. Then, each of the multiplicity values is added to a sum, which is initialized to 0, in order until the sum is equal to or bigger than $\rho_s \times |\mathcal{D}|$ after adding the k-th multiplicity. The k is then the GlobalK threshold for r.

SectTop GlobalK offers a simple method for filtering out candidates that cannot be frequent. However, the GlobalK threshold is calculated based on the highest

multiplicity values for an association r, which may be too generous as a pruning strategy because the consequences with top multiplicity values may not always enter the connecting set.

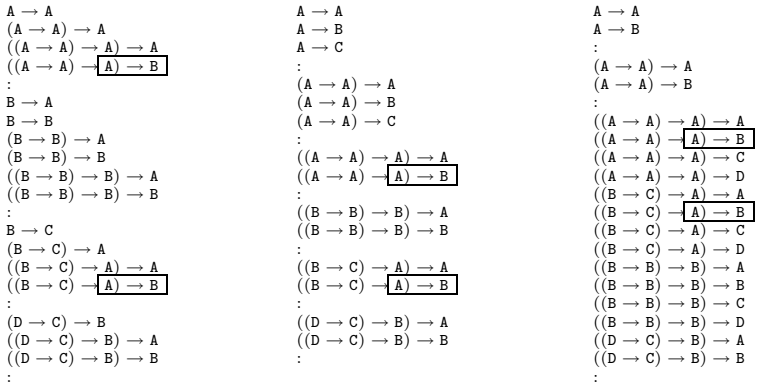
We address this issue in our second pruning strategy, *SectTop*. For a frequent association $I \rightarrow J$, SectTop "localizes" consequences with high multiplicities to a portion of the whole length of time covered by the dataset $\mathcal{D}$ so that, when evaluating whether a candidate $(I \rightarrow J) \rightarrow K$ can be frequent, the high multiplicities count only if the corresponding consequences can appear in the connecting set.

To check whether a candidate $(I \rightarrow J) \rightarrow K$ can be frequent, SectTop assumes that the whole period of time covered by the dataset is divided into n non-overlapping segments. Then, for $I \rightarrow J$, a SectTop vector is determined for each of the segments as follows. First, the multiplicity values of the consequences for r appearing in the segment are sorted inversely. Then, we keep the k highest multiplicity values such that k is minimum and sum of the k values exceeds $\rho_s \times |\mathcal{D}|$. If the sum of all the k values does not exceed $\rho_s \times |\mathcal{D}|$, all of the multiplicity values are kept. The multiplicity values are then transformed to a vector of k values such that the x-th element is the sum of the top-x multiplicity values in the segment. For example, in Figure 4(a), suppose the whole length of time covered by $\mathcal{D}$ is divided into segments of 5 time units so that events with time $t \leq 5$ belongs to the first segment, those with $5 < t \leq 10$ belongs to the second segment, and so on. In the first segment, only 1 consequence exists for $I \rightarrow J$ and so the SectTop vector contains only 1 value (1 in this case). For the second segment, since there are 2 consequences, the multiplicity values (namely, 3 and 2) are inversely sorted and the SectTop vector is obtained by keeping the cumulative sum of the values. Hence, the vector $\langle 3, 5 \rangle$ is obtained.

The SectTop vectors for a frequent association $r = I \rightarrow J$ can be interpreted as follows. When checking whether a candidate $r' = (I \rightarrow J) \rightarrow K$ can be frequent, for each segment, if up to x consequences for r are to appear in the connecting set, then the maximum number of matches for r' induced from the x consequences is given by x-th element of the vector. Sum of the maximum number of match for each segment gives an overall (upper bound) guess on the number of matches in the M-Q mapping of r'.

With the SectTop vectors for r found, we evaluate as follows whether a candidate r' can be frequent. With respect to the n segments when we calculate the SectTop vectors for r, we count the number of distinct matches for $J \rightarrow K$ appearing in each segment. If there are x matches for $J \rightarrow K$ in the i-th segment, then, in the segment, at most x consequences of r may appear in the connecting set. Hence, the maximum number of matches induced from the x consequences is given by the x-th element of the corresponding SectTop vector. The sum of the maximum number of matches for each segment thus gives an overall maximum number of matches of r'. If the overall maximum is smaller than $\rho_s \times |\mathcal{D}|$, r' cannot be frequent.

For example, the table on the left in Figure 4(c) is the M-Q mapping of $J \rightarrow K$, while that on the right lists the number of matches of $J \rightarrow K$ appearing



(a) Depth-first candidate generation (b) Breadth-first candidate generation (c) Breadth-first* candidate generation

Fig. 5. Candidate generation schemes

in each segment. When evaluating whether $r' = (I \rightarrow J) \rightarrow K$ can be frequent, we check the number of matches of $J \rightarrow K$ in each segment against the SectTop vectors of $I \rightarrow J$. It turns out that the maximum number of matches for r' is (0 + 3 + 0) = 3. If the support threshold is 4 matches, then we know immediately that r' cannot be frequent.

4.2 Cache management

The baseline algorithm generates a lot of associations during execution. Some of them are needed for candidate evaluation repeatedly later on. It is, of course, ideal if all these associations can be fitted in the main memory so that they can be instantly retrieved whenever needed. However, this is usually not the case because the volume of data being processed is often very large. Maintaining a cache is thus a compromise so that, while keeping some of the intermediate results in memory and reduce I/O accesses, memory can be made available for other operations.

When the cache overflows, part of the cached data is replaced by data fetched from disk. Two commonly used strategies for choosing data for replacement are "Least recently used" (LRU), i.e., data that have not been accessed for the longest time are replaced, and "Least frequently used" (LFU), i.e., least frequently accessed data in the cache are replaced.

The effectiveness of cache, i.e., the likeliness that the data accessed is in the cache, is often mentioned as the *hit-rate*. To achieve high cache hit-rate, an important factor is that, if a piece of information is repeatedly accessed, the accesses should be temporally close so that when the information is accessed again, it is less likely that the information have been swapped out of the cache.

Figure 5 shows three different orders that candidates are generated and evaluated. Figure 5(a) demonstrates a depth-first candidate generation, i.e., after an

association $r = I \rightarrow J$ is evaluated as frequent, the algorithm immediately generates candidates by extending r and evaluates them. Figure 5(b), on the other hand, shows a breadth-first candidate generation that all candidates of the same length are evaluated before longer candidates. These are common schemes for level-wise algorithms evaluating candidates for inclusion in results.

We argue that ordinary depth-first and breadth-first candidate generation schemes do not pay attention to their impact on cache management. For example, in Figure 5(a), $A \rightarrow B$ is referenced when the candidates $((A \rightarrow A) \rightarrow A) \rightarrow B$ and $((B \rightarrow C) \rightarrow A) \rightarrow B$ are evaluated (see the highlighted parts in the figure). In between the accesses, a number of other candidates are evaluated. Recall that, when a candidate $(\mathbb{I} \rightarrow J) \rightarrow K$ is evaluated, we compute its M-Q mapping from those of $\mathbb{I} \rightarrow J$ and $J \rightarrow K$. Hence, the more candidates evaluated between two accesses to $A \rightarrow B$, the more other associations are brought into the memory and so cache overflows are more likely. So, when $A \rightarrow B$ is accessed the second time, it is likely that $A \rightarrow B$ no longer resides in cache, and an I/O access is needed to bring it back to memory from disk. Similar problem exists in the breadth-first candidate generation shown in Figure 5(b).

It is noteworthy that, in the baseline algorithm, length-2 associations are frequently accessed when evaluating candidate associations. In particular, for an association $r \in L_i$ whose consequence event type is X, each of length-2 associations with triggering event type X, i.e., those associations of the form $X \rightarrow Y$ for some event type Y, is referenced when evaluating candidates generated from r. By processing as a batch all associations in L_i with the same consequence event type X (Figure 5(c)), we can have length-2 associations with triggering event type X accessed closely temporally, which favours cache hit-rate.

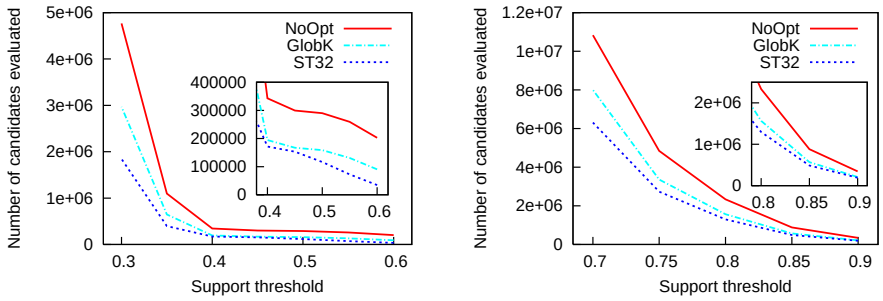
The special order for candidate evaluation can be easily fitted into the breadth-first candidate generation scheme. At the end of each iteration, we can sort the associations in L_i by their consequence event type. Then the sorted associations are fetched sequentially for candidate generation. We call this the *breadth-first** candidate generation scheme.

5 Experiment results

We conducted experiments using stock price data. Due to space limitation, we leave the discussion on how the raw data is transformed into an event dataset in Appendix A.1. The transformed dataset consists 99 different event types and around 45000 events.

5.1 Pruning strategy

In the first set of experiments, we want to study the effectiveness of the pruning strategy "GlobalK" and "SectTop". The effectiveness is best reflected by the number of candidate associations being evaluated. Figure 6 shows the number of candidate associations evaluated when ρ_s is set at different values. We comment that a candidate is regarded as "evaluated" only if the M-Q mapping of the



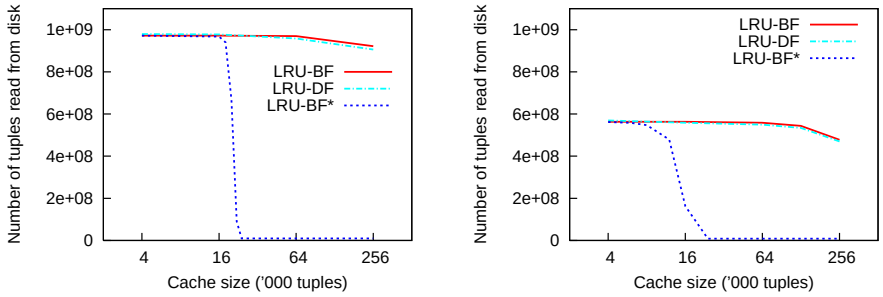
(a) $[u, v] = [\delta, 1]$ (b) $[u, v] = [\delta, 2]$

Fig. 6. No. of candidates evaluated at different ρ_s

candidate is enumerated. The lines labelled "NoOpt", "GlobalK" and "ST32" represent respectively the case that no pruning strategy (i.e., the original baseline algorithm) is used, that "GlobalK" is chosen and that "SectTop" is chosen with the time covered by $\mathcal{D}$ divided into 32 segments.

Figure 6(a) shows the results when u and v are set to δ (i.e., a value just bigger than 0) and 1 respectively. As shown in the figure, both GlobalK and SectTop save a major fraction of candidate evaluations performed. At high support (0.6%), savings of 55% and 82% are observed respectively with GlobalK and SectTop over the baseline algorithm while, at low support (0.3%), the savings are 32% and 63%. Similar trend is observed when we changed v to 2 (Figure 6(b)). Although the savings are not as dramatic as in the case when $v = 1$, at low support (0.7%), GlobalK and SectTop achieve savings of 26% and 41%, while at high support (0.9%), the savings are around 39% and 44% respectively.

As shown by the figures, SectTop always outperform GlobalK in terms of number of candidates being evaluated. A reason is that, for each candidate association, SectTop calculates for each segment an upper-bound on the number of matches that are associated to the consequences in the segment. Summation of the upper-bounds for each segment thus gives an upper-bound on the overall number of matches for the association. A reasonably fine segmentation of the time covered by $\mathcal{D}$ ensures that, when evaluating the candidate $(\mathbb{I} \rightarrow J) \rightarrow K$, the multiplicity of a consequence q for $\mathbb{I} \rightarrow J$ is counted into the upper bound only if there exists a match for $J \rightarrow K$ that occurs near q. Hence, we can obtain a relatively tight upper-bound on the number of matches and avoid unnecessary candidate evaluations. For GlobalK, however, the GlobalK threshold for a frequent association is calculated from the highest multiplicity values without considering where these values actually exist in the whole period of time covered by $\mathcal{D}$. So, the pruning ability of GlobalK is not as good as that of SectTop.



(a) LRU, $\rho_s = 0.3\%$, "NoOpt" (b) LRU, $\rho_s = 0.3\%$, "ST32"

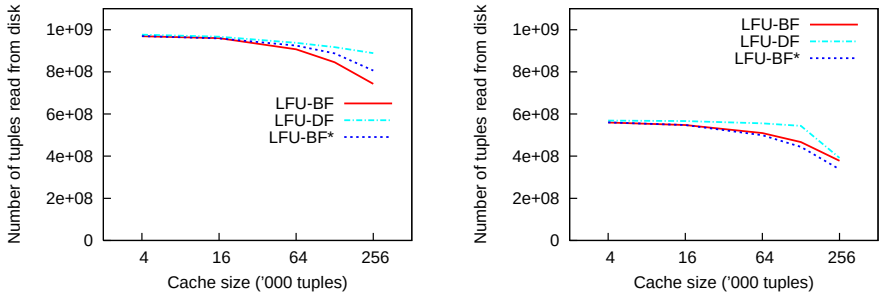
Fig. 7. I/O requirement for LRU ($[u, v] = [\delta, 1], \rho_s = 0.3\%$)

5.2 Candidate generation, cache replacement strategy and I/O access

In the second set of experiments, we want to study the effect of candidate generation orders on different cache replacement strategies. We plot the number of M-Q mapping tuples read from disk, reflecting total I/O requirement, against the size of the cache and the results in Figures 7 and 8.

We start the analysis with the LRU strategy and ρ_s set to a relative low value at 0.3%. Figure 7(a) shows the I/O performance when no pruning strategy is applied. From the figure, we find that the I/O performance of breadth-first and that depth-first strategies are very much close to each other. Interestingly, increasing the cache size from 4000 to 256000 tuples does not result in much improvement in performance. On the other hand, for *breadth-first** strategy, the I/O costs begins to drop at 16000 tuples and then drops dramatically. The improvement levels down when the cache size is increased to 24000 tuples.

The sharp improvement here is no coincidence. Recall that in *breadth-first** candidate generation, at the end of an iteration, we sort the frequent associations found in the iteration by their consequence event type. Candidates are formed and evaluated by extending each of the sorted associations sequentially. In other words, after candidates in the form $(\mathbb{I} \rightarrow X) \rightarrow Y$ are evaluated, for some complex/simple event type $\mathbb{I}$ and simple event types X and Y, next evaluated are those in the form $(\mathbb{I}' \rightarrow X) \rightarrow Z$ (if there exists such $\mathbb{I}'$). Hence, the whole set of length-2 associations with triggering event type X are accessed multiple times. If the cache is big enough to hold the M-Q mappings of the whole set of length-2 associations, it is very likely that the M-Q mappings are still in the cache next time when they are referenced. For the dataset in the experiment, we find that for a particular triggering event type, the maximum sum of the sizes of all M-Q mappings is around 22000 tuples (as shown in Figure 10 in Appendix A.3). A cache with 24000-tuple capacity can thus hold the M-Q mappings of any set of length-2 associations with the same triggering event type. Hence, the algorithm can take full advantage of the cache and save I/O access when the associations are accessed again. For cases with smaller cache size, however, the cache is not



(a) LFU, $\rho_s = 0.3\%$, "NoOpt" (b) LFU, $\rho_s = 0.3\%$, "ST32"

Fig. 8. I/O requirement for LFU ($[u, v] = [\delta, 1]$, $\rho_s = 0.3\%$)

big enough and so the M-Q mapping of each length-2 associations are repeatedly read from disk each time when it is needed.

Figure 7(b) shows the case when "ST32" is applied. The highest I/O required is about 60% of the case that no pruning strategy is applied because a lot of candidate evaluations are avoided by the pruning strategy. Like the case when no pruning strategy is applied, breadth-first and depth-first perform similarly in I/O requirement. Besides, a big drop in I/O access is observed with the line representing *breadth-first** although, in this case, the big drop begins at the cache size of 10000 tuples, instead of 16000 tuples in the previous case. This is because SectTop avoids evaluating candidates that cannot be frequent. So, when evaluating candidates generated by extending a frequent association, say $\mathbb{I} \rightarrow X$, it is not necessary to access every association $X \rightarrow Y$. A smaller cache is thus enough to hold the M-Q mappings of all length-2 association for candidate evaluation.

Figure 8 shows the case when LFU is adopted instead of LRU. From the figure, all three candidate generation methods are very similar in terms of I/O requirement. Both depth-first and breadth-first generation performed slightly better when LFU was adopted instead of LRU. However, the "big drop" with *breadth-first** is not observed and so the performance of *breadth-first** is much worse than the case with LRU. It is because the LFU strategy gives preference to data that are frequently accessed than those that are recently accessed when deciding on what to keep in the cache. This does not match the idea of *breadth-first** candidate generation, which works best when recently accessed data are kept in the cache. In addition, associations entered the cache early may reside in the cache for a long time because, when they are first used for evaluating candidates, a certain number of accesses have been accumulated. Associations newly added to the cache must be accessed even more frequently to stay in the cache. Besides, the LFU strategy works better when accesses to the associations are very skewed. In the stock data we used, however, the distribution of frequent associations are relatively even in the sense that different event types appear as the consequence event type (i.e., on the right-hand-side) of some association. So,

length-2 associations with different triggering event type (i.e., on the left-hand-side) are accessed, which does not favour LFU.

The same set of experiments has also been run with ρ_s set to 0.6%, a relatively high value. The graphs are similar in shape to those for the $\rho_s = 0.3\%$ case. Due to space limitation, we include the graphs in Appendix A.2.

6 Conclusion

We propose time-delayed association as a way to show time-delayed dependencies between types of events. Such kind of associations is particularly useful in monitoring systems since, based on mined associations, one can make predictions on events that are likely to occur in a designated time-frame. Precautious actions can be taken if unfavourable conditions are predicted.

We illustrate how time-delayed associations can be found from event datasets by starting with a simple but brute-force algorithm. We identify two areas for improvement in the simple algorithm. First, the fact that apriori property does not hold in time-delayed associations means that the simple algorithm has to evaluate a very large number of candidate associations. A good candidate pruning strategy is thus needed to prune the search space for frequent associations. We proposed two pruning strategies, namely, GlobalK and SectTop, and experiments show that they can reduce the number of candidates being evaluated.

Besides the candidate pruning issue, in a sensor environment, the volume of data being processed is likely to be high. Hence, swapping some of the intermediate results to disk is necessary in order to make room for other operations. When the intermediate results are brought back to memory, time-consuming I/O accesses are needed. A smart caching here can thus reduce the I/O requirement of the algorithm. We find that the order that candidate associations are formed and evaluated would affect the performance of the cache. Experiment results show that the *breadth-first** candidate generation scheme, coupled with a reasonably-sized cache and the LRU cache replacement strategy, can significantly reduce the I/O requirement of the algorithm.

References

1. Xiaonan Ji, James Bailey, and Guozhu Dong. Mining minimal distinguishing subsequence patterns with gap constraints. In *ICDM*, pages 194–201, 2005.
2. Daesu Lee and Wonsuk Lee. Finding maximal frequent itemsets over online data streams adaptively. In *ICDM*, pages 266–273, 2005.
3. Jin Li, David Maier, Kristin Tufte, Vassilis Papadimos, and Peter A. Tucker. Semantics and evaluation techniques for window aggregates in data streams. In *SIGMOD Conference*, pages 311–322, 2005.
4. Heikki Mannila and Hannu Toivonen. Discovering generalized episodes using minimal occurrences. In *KDD*, pages 146–151, 1996.
5. Spiros Papadimitriou, Jimeng Sun, and Christos Faloutsos. Streaming pattern discovery in multiple time-series. In *VLDB*, pages 697–708, 2005.

6. Yasushi Sakurai, Spiros Papadimitriou, and Christos Faloutsos. Braid: Stream mining through group lag correlations. In *SIGMOD Conference*, pages 599–610, 2005.
7. Mohammed Javeed Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1/2):31–60, 2001.
8. Rui Zhang, Nick Koudas, Beng Chin Ooi, and Divesh Srivastava. Multiple aggregations over data streams. In *SIGMOD Conference*, pages 299–310, 2005.

APPENDIX

A Supplementary experiment results

A.1 Data collection

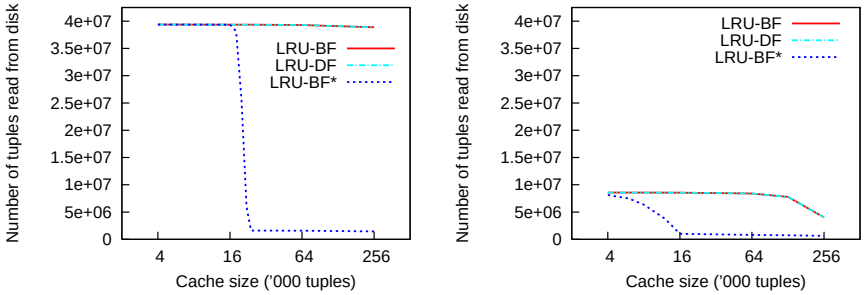
We conduct experiments using stock price data of the 33 constituent stocks of Hang Seng Index, the most indicative stock market index in Hong Kong. The stock price data are transformed to event data as follows. For each trading day, we compare the opening price of a stock against its closing price. For the stock X, an event "XA" is recorded for the trading day if the opening price of X is higher then its closing price, "XB" is recorded if both prices are equal and "XC" is recorded otherwise. Hence, there are 99 different event types. In practice, an event number is sequentially assigned to each of the event types while another sequence number is assigned to each of the transaction days in chronological order. Each event recorded are thus of the form (E, t) where E is the event number and t is the transaction day sequence number. The stock price dataset contains data from 01 Jan 2000 to 20 Apr 2006, which consists 1533 trading days. The resultant dataset contains around 45000 events. We comment that the size of the dataset is smaller than $33 \times 1533 = 50589$ because some of the HSI constituent stocks first became floated after year 2000.

A.2 I/O requirements for the high ρ_s case

Figure 9 shows the I/O performance when ρ_s is set to 0.6%, a relatively low value. Figure 9(a) and 9(b) show respectively the performance with LRU when no pruning strategy is used and when "SectTop-32" is applied, whereas Figure 9(c) and 9(d) are respectively the cases with LFU when no pruning strategy is used and when "SectTop-32" is applied. The shapes of the graphs are very much similar to their 0.3% counterparts and our arguments stated in the paper also apply to these cases.

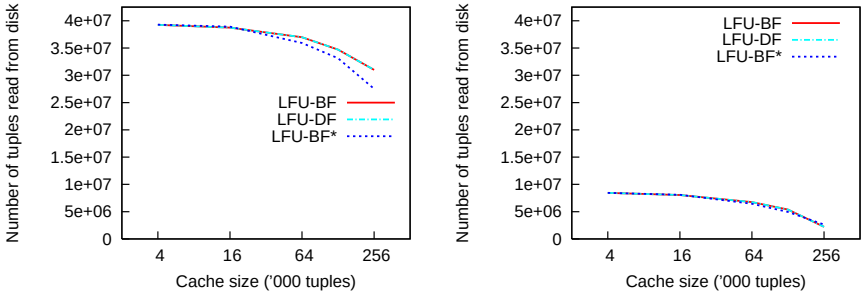
A.3 Sizes of M-Q mappings for length-2 associations

In Figure 10, we list the total size of M-Q mappings for length-2 associations that have common triggering event type. The total size is given by the total number of tuples in the M-Q mappings. X-axis of the graph is the sequence number of the event type. We comment that the maximum height of the bars reads 21926 tuples.



(a) LRU, $\rho_s = 0.6\%$, "NoOpt"

(b) LRU, $\rho_s = 0.6\%$, "ST32"



(c) LFU, $\rho_s = 0.6\%$, "NoOpt"

(d) LFU, $\rho_s = 0.6\%$, "ST32"

Fig. 9. I/O requirement for different settings ($[u, v] = [\delta, 1]$), $\rho_s = 0.6\%$

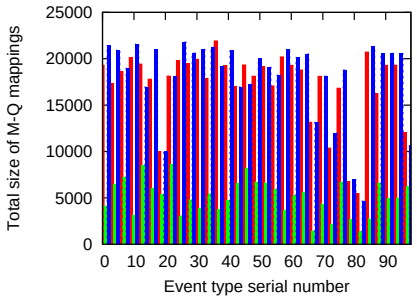


Fig. 10. Total size of M-Q mappings for length-2 associations grouped by the triggering event type