Monitoring Complex Patterns of Events

Lilian Harada **Yuuji Hotta**

Fujitsu Laboratories Ltd.
{harada.lilian, yhotter}@jp.fujitsu.com

Abstract

The fast changing business environment today makes high demands on the organizations for timely strategic decisions and actions. In this paper we present our approach to efficiently detect problematic sequences of events that require action from the decision-maker.

Keywords: monitoring, events, patterns.

1. Introduction

In order to survive and flourish in the ever faster changing business world, organizations and companies are required to make strategic decisions and take business actions at the right time. However, today's business complexity makes it difficult to analyze all the business events that should be taken into account for those decisions and actions. In such complex analysis, the examination of only a single business event is usually not enough. In this paper we present our approach to efficiently detect sequential patterns of events that represent situations that may affect the company business performance and thus should be promptly notified to the decision-makers [2].

This paper is organized as follows. Section 2 presents the kind of sequential patterns of events focused in our work, and Section 3, the algorithm for detecting those patterns. Section 4 presents evaluation results showing the efficiency of the proposed algorithm. Finally, Section 5 concludes the paper.

2. Pattern of Events

Let R be a set of records composed of n attributes $\{a_1, a_2, \ldots, a_n\}$. By specifying attribute a_k as the Group Coordinate (GC) and a_l as the Order Coordinate (OC), where $1 \leqq k \leqq n$, $1 \leqq l \leqq n$, and $k \neq l$, then R is partitioned according to the GC into a set of partitions and each partition's records are sorted according to the OC. This results in a set of sequences of events that are ordered on the a_l values, where there is one sequence for each value of a_k.

Let's suppose a set of records of 4 attributes {custID, day, item, price}. By specifying custID as the GC and day as the OC, we logically have two sequences of events as illustrated in Fig. 1. One sequence is for cust1 that has purchased 9 items, and the other sequence is for cust2 that has purchased 6 items.

After a set of sequences is defined by the specifications of the GC and OC, we define the conditions of the pattern to be searched over these sequences.

Grouping Coordinate (GC)

Ordering Coordinate (OC)

custID	day	item	price	
cust1	10/1	C	150	r1
cust1	10/2	D	120	r2
cust1	10/5	A	180	r3
cust2	10/5	A	180	r4
cust2	10/6	C	200	r5
cust1	10/7	E	90	r6
cust2	10/8	E	90	r7
cust1	10/10	C	190	r8
cust1	10/12	A	220	r9
cust2	10/15	B	270	r10
cust1	10/17	H	230	r11
cust2	10/18	B	250	r12
cust1	10/19	B	250	r13
cust1	10/21	B	260	r14
cust2	10/23	C	220	r15

Fig. 1: Set of sequences.

Given a set of sequences, an m-pattern $[e_1\ \Theta_{1,2}\ e_2\ \Theta_{2,3}\ e_3\ \ldots\ e_{m-1}\ \Theta_{m-1,m}\ e_m]$, where e_i represents an event and $\Theta_{i,i+1}$ represents the distance relationship between consecutive events e_i and e_{i+1}, is defined. This distance relationship specifies whether two consecutive events e_i and e_{i+1} of the pattern occur simultaneously, contiguously, or non-contiguously in the input sequence of events. Specifically:

- $e_i = e_{i+1}$ means that both events occur simultaneously;
- $e_i - e_{i+1}$ means that the events are contiguous; that is, e_{i+1} occurs immediately after e_i;

- $e_i < e_{i+1}$ means that e_{i+1} occurs after but not immediately after e_i.

Each event e_i ($1 \leqq i \leqq m$) of the pattern satisfies some intra-event conditions; that is, some conditions for the values of any of the attributes a_j of event e_i ($1 \leqq j \leqq n$) that require only the examination of the event itself. Besides these intra-event conditions, there are also inter-event conditions; that is, conditions that specify how different events in the pattern are inter-related with specifications for the values of any of the attributes a_j for $1 \leqq j \leqq n$, $j \neq k$, where a_k is the GC. A special inter-event condition gives the window size ws of the pattern; that is, the maximum distance allowed between the first event e_1 and the last event e_m of the pattern, i.e., $ws \geqq e_m.a_l - e_1.a_l$, where a_l is the OC.

Consider a simple example that searches for a pattern composed of three events e_1, e_2 and e_3, where e_1 specifies the purchase of an item "A"; e_3 specifies a later purchase of an item "B" within 15 days of e_1; and e_2 specifies all the purchases of items different to either "A" and "B" that occur between e_1 and e_3. Let's also include the condition that the price of the item purchased in e2 is higher than the price of item "A" in e_1, and lower than the price of item "B" in e_3. This pattern can be expressed as shown in Fig. 2. In DEFINE CONTEXT the GC, OC as well as the other attributes of the events composing the sequence are specified by GROUP, ORDER and DATA, respectively. In this case, custID is the GC, day is the OC and there are also the item and price attributes. Then, in DEFINE EVENT, the intra-event conditions are specified and in DEFINE CONDITION, the inter-event conditions as well as the information of the events to output are specified.

```
DEFINE CONTEXT {
  GROUP custID AS VCHAR
  ORDER day AS DATE
  DATA item AS VCHAR
  DATA price AS INTEGER
}
DEFINE EVENT {
  e1 AS item == "A"
  e2 AS item != "A" and item != "B"
  e3 AS item == "B"
}
DEFINE CONDITION {
  CASE ( e1 < e2 < e3
    AND
    e2.price > e1.price
    AND
    e3.price > e2.price
    AND
    e3.day – e1.day ≦15
    }
  OUPUT (e1.custID, e1.item, e1.price, e2.item, e2.price,
          e3.item, e3.price)
}
```

Fig. 2: Pattern specification in our proposed query language.

custID	e1.item	e1.price	e2.item	e2.price	e3.item	e3.price	
cust1	A	180	C	190	B	250	r3, r8, r13
cust1	A	180	H	230	B	250	r3, r11, r13
cust1	A	220	H	230	B	250	r9, r11, r13
cust1	A	220	H	230	B	260	r9, r11, r14
cust2	A	180	C	200	B	250	r4, r5, r12

Fig. 3: Resultant patterns.

Fig. 3 shows the four patterns that are detected for cust1, and the single pattern that is detected for cust2 from the input of Fig. 1.

3. Pattern Detection Processing

As presented in the previous section, the patterns to be detected can be very complex. They can specify conditions on multiple attributes, and those conditions can have inter-relationships with previous events in the pattern. Also, multiple events can occur simultaneously (i.e., have the same OC value), and consecutive events of the pattern do not necessarily occur contiguously in the sequence. Therefore, previous approaches [1],[3] that extend traditional string-matching algorithms cannot be applied.

Since the patterns are frequently specified in a context where the focus is on the time-ordered relationship, the records are already sorted on OC values (in case the OC is specified as an attribute different to the timestamp, the records are first sorted on OC values). Then, each event is hashed on the GC value so that a sequence of events is generated for each value of GC.

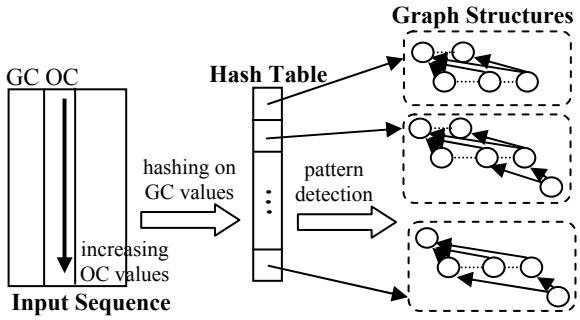

Fig. 4: Pattern detection processing.

As illustrated in Fig. 4, the pattern detection processing that utilizes an on-memory graph structure is basically composed of the following steps:

(a) filtering of an event that satisfies the pattern conditions,
(b) introduction of the filtered event as a node in the graph structure,
(c) output of a detected pattern, and

(d) deletion of unnecessary nodes from the graph structure.

Fig. 5 presents a simplified pseudocode for the detection of an m-pattern over a sequence in which the events cannot have the same order value and therefore $\Theta_{ei,ei+1} \in \{-, <\}$. Except the first event e_1, which is specified by an intra-event condition — i.e., $e_1 = f_{intra1}(C_1)$, where C_1 is a constant — each event e_i can be specified by a conjunction of intra-event and inter-event conditions — i.e., $e_i = f_{intrai}(C_i)$ AND $f_{interi}(e_{i-1})$, where $2 \leqq i \leqq m$. A special inter-event condition that we will consider separately is the maximum distance allowed between two consecutive events e_i and e_{i+1}, which we denote as $\Delta_{i,i+1}$. For instance, when the OC attribute is the day attribute, then $\Delta_{i,i+1} \geqq e_{i+1}.day - e_i.day$. When the distance $\Delta_{i,i+1}$ is not specified, we take $\Delta_{i,i+1}$ as the window size ws.

Due to space limitations in this paper, we will not present details of the extensions that are necessary for the more complex case in which multiple events can have the same order value (e.g., multiple simultaneous results) and therefore $\Theta_{ei,ei+1} \in \{-, <, =, \leqq\}$. Basically, all simultaneous input events must be checked once against each of the pattern events e_i, and a check must be made to prevent the same input event *r* from appearing in different events e_i of the pattern.

```
While there is a new input event r to process
  /* level 1 */
    If r satisfies f_intra1(C_1)
        Create a node e_1 that represents r
  /* levels 2 to m-1 */
    For ( i = 2, ..., m-1 )
        Take the newest node e_i-1
        While r satisfies Δ_ei-1, ei AND Θ_ei-1,ei
            If r satisties f_intrai(C_i) AND f_interi(e_i-1)
              /* f_intrai( ) and f_interi( ) = TRUE if not explicitly */
        /*  specified in the pattern */
                Create a node e_i that represents r (if it does
                not exist yet), and link it to the node e_i-1
                Take the previous node e_i-1
  /* last level m */
    While the distance between r and the oldest node e_1 does
    not satisfy the window size ws
          Discard the oldest node e_1 and all nodes linked only
to it
    Take the newest node e_m-1
    While r satisfies Δ_em-1,em AND Θ_em-1,em
        If r satisties f_intram(C_m) AND f_interm(e_m-1)
            /* f_intram( ) and f_interm( ) = TRUE if not explicitly */
     /* specified in the pattern */
            Create a node e_m that represents r (if it does not
            exist yet), and link it to the node e_m-1
            Take the previous node e_m-1
    Output all patterns composed by the subgraph rooted at
        node e_m
    Discard the node e_m
```

Fig. 5: Simplified pseudocode for pattern detection.

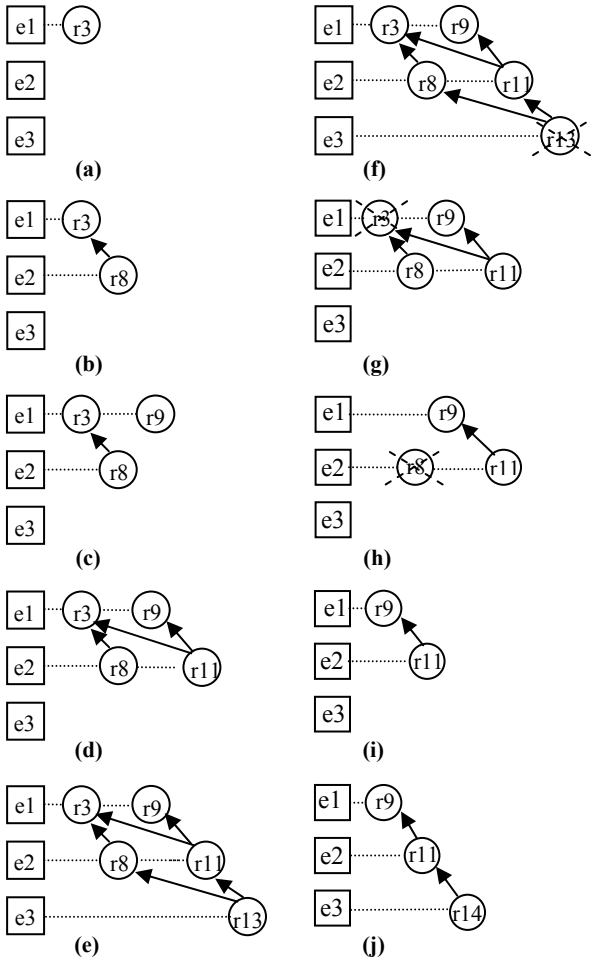

Fig. 6: Graph construction for pattern detection.

Fig. 6 illustrates how the graph evolves in the detection of the 4 resultant patterns for cust1 as shown in Fig. 3. Patterns [r3,r8,r13], [r3,r11,r13], [r9,r11,r13] are output by searching the graph in Fig.6(e), and [r9,r11,r14] from the graph in Fig.6(j).

4. Evaluation Results

Here we present evaluation results for experiments ran in a Primepower2000 with SPARC64 III 248 MHz×1 and 512 MB of memory.

Fig. 7 shows results for the processing time required to detect the example-pattern shown in Fig. 2 over a real-life retail data corresponding to the purchases of one customer in a supermarket during 12 months, 18 months and 24 months. The average number of items purchased per day by this customer is 1.2 items. We compare the proposed method to a more straightforward approach that scans the sequence once for the checking of each specified event. Our proposed approach requires only about 1 second for the pattern detection in all the three input sequences. We can see

that since it detects the pattern by scanning the events' sequence only once, it is much more efficient than the straightforward approach.

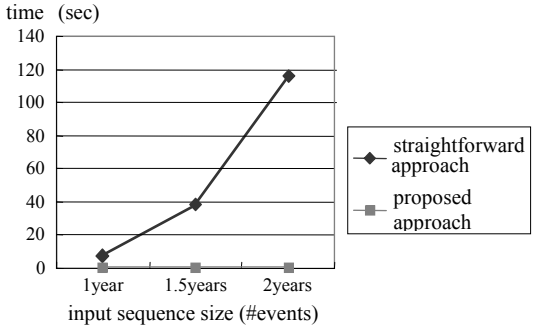

Fig. 7: Processing times for detection of pattern in Fig. 2.

We have also performed extensive evaluation analysis with artificially generated data since they can be more easily manipulated for analysis. Fig. 8 presents the processing rate (in #events/sec) in the detection of a pattern composed of 2 events, where each 2 consecutive input events compose a resultant output pattern. As expected, the processing rate is inversely proportional to the number of searching patterns that are simultaneously processed. For a single pattern, the processing rate is of 45,075 events/sec; for 2 simultaneous patterns, 23,320 events/sec; and for 3 simultaneous patterns, 15,656 events/sec. Note that those rates are sufficiently high to satisfy the requirements of many applications that require on-line event monitoring.

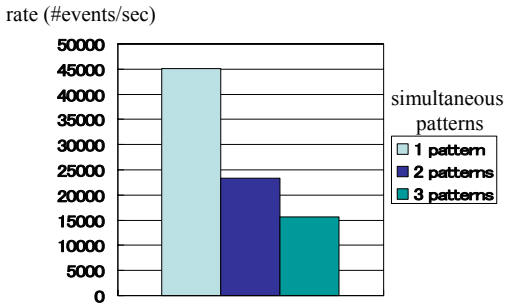

Fig. 8: Processing rates for simultaneous pattern detection.

Fig. 9 shows the processing rate when increasing the pattern length from 2 events to 4 events. As expected, the rate decreases for longer patterns. While for 2 events' pattern the rate is 45,075 events/sec, for 3 events is 24,786 events/sec, and for 4 events is 16,514 events/sec.

Fig. 10 shows the processing times for the pattern detection of 2 different selectivities: one in which only one pattern is detected for each window, and another when 3 patterns are detected for the same window. As expected, the time required for higher selectivity is longer, and the component with the most relative increase is the graph searching component.

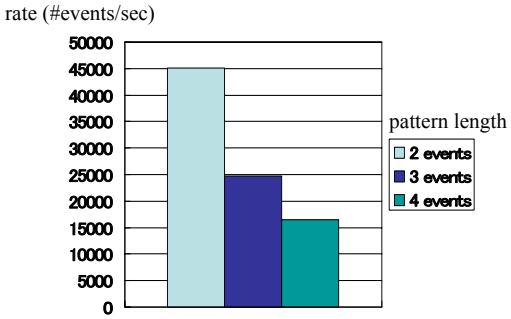

Fig. 9: Processing rates for different patterns' length.

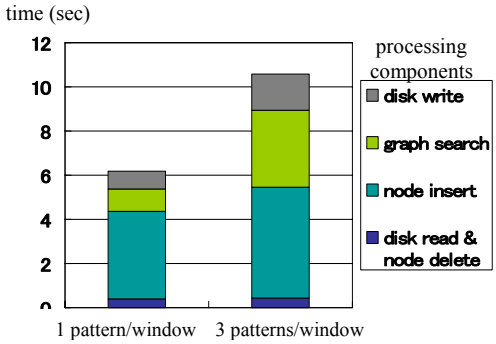

Fig. 10: Processing times for different selectivities.

5. Conclusions

In this paper we presented our approach for searching complex patterns in sequences of events. Evaluation results showed the proposed method to be very efficient and we believe it can be applied in many new applications that require on-line monitoring of events. We have plans to make a trial for network monitoring, and also for banking transactions monitoring to detect suspicious behavior that can be pre-defined by patterns of events.

6. References

[1] L.Harada, "Complex Temporal Patterns Detection over Data Streams", *Proc. Of ADBIS'02*, pp.401-414, 2002.

[2] L.Harada and Yuuji Hotta, "Detection of Sequential Patterns of Events for Business Intelligence Solutions", *Proc. Of IDEAS'04*, pp.475-479, 2004.

[3] R.Sadri, C.Zaniolo, A.Zarkesh, and J.Adibi, "Optimization of Sequence Queries in Database Systems", *Proc. Of PODS'01* , pp.71-81, 2001.