An Index-Based Method for Timestamped Event Sequence Matching*

Sanghyun Park[1], Jung-Im Won[1], Jee-Hee Yoon[2], and Sang-Wook Kim[3]

[1] Department of Computer Science,
Yonsei University, Korea
{sanghyun, jiwon}@cs.yonsei.ac.kr
[2] Division of Information Engineering and Telecommunications
Hallym University, Korea
jhyoon@hallym.ac.kr
[3] College of Information and Communications
Hanyang University, Korea
wook@hanyang.ac.kr

Abstract. This paper addresses the problem of timestamped event sequence matching, a new type of sequence matching that retrieves the occurrences of interesting patterns from a timestamped event sequence. Timestamped event sequence matching is useful for discovering temporal causal relationships among timestamped events. In this paper, we first point out the shortcomings of prior approaches to this problem and then propose a novel method that employs an R^*-tree to overcome them. To build an R^*-tree, it places a time window at every position of a timestamped event sequence and represents each window as an n-dimensional rectangle by considering the first and last occurrence times of each event type. Here, n is the total number of disparate event types that may occur in a target application. When n is large, we apply a grouping technique to reduce the dimensionality of an R^*-tree. To retrieve the occurrences of a query pattern from a timestamped event sequence, the proposed method first identifies a small number of candidates by searching an R^*-tree and then picks out true answers from them. We prove its robustness formally, and also show its effectiveness via extensive experiments.

Keywords: Event sequence, indexing, similarity search.

1 Introduction

A *sequence database* is a set of data sequences, each of which comprises an ordered list of symbols or numeric values [1]. *Similar sequence matching* is an operation that finds sequences similar to a query sequence from a sequence database [1,2,5,8]. During the past decades, many useful techniques have been proposed for similar sequence matching [1,2,4,5,6,7,9,11,12,16].

* This work was partially supported by Korea Research Foundation Grant funded by Korea Government (MOEHRD, Basic Research Promotion Fund) (KRF-2005-206-D00015), by the ITRC support program(MSRC) of IITA, and by the research fund of Korea Research Foundation with Grant KRF-2003-041-D00486.

K.V. Andersen, J. Debenham, and R. Wagner (Eds.): DEXA 2005, LNCS 3588, pp. 493–502, 2005.

Event	Timestamp
:	:
CiscoDCDLinkUp	19:08:01
MLMSocketClose	19:08:07
MLMStatusUp	19:08:20
:	:
MiddleLayerManagerUp	19:08:37
TCPConnectionClose	19:08:39
:	:

Fig. 1. Example of a timestamped event sequence in a network environment

Wang et al. [14] defined a new type of similar sequence matching that deals with timestamped event sequences. To exemplify this type of matching, let us examine an event management system in a network environment. Here, items, each of which is a pair of (event type, timestamp), are sequentially added into a log file in their chronological order whenever special events arise. Figure 1 shows a sequence of items in a log file.

In such environment, the queries to identify *temporal causal relationships* among events are frequently issued as follows [14]: "Find all occurrences of CiscoDCDLinkUp that are followed by MLMStatusUp within 20±2 seconds as well as TCPConnectionClose within 40±3 seconds."

In reference [14], they call a sequence of items as in Figure 1 a *timestamped event sequence* T, and regard the above query as a query sequence $Q = \langle$(CiscoDCDLinkUp, 0), (MLMStatusUp, 20), (TCPConnectionClose, 40)$\rangle$ with 2 as a tolerance for the interval between CiscoDCDLinkUp and MLMStatusUp and 3 as a tolerance for the interval between CiscoDCDLinkUp and TCPConnectionClose. Then, the above query is converted to the problem of *timestamped event sequence matching* which is to retrieve from T all subsequences, possibly *non-contiguous*, that are matched with Q. Since a non-contiguous subsequence can be an answer in timestamped event sequence matching, previous methods proposed for similar sequence matching are not appropriate for this new problem.

Reference [14] tackled the problem and proposed a method that employs a new index structure called an *iso-depth index*. This method uses the concept of a *time window*, a contiguous subsequence whose time interval is not larger than ξ, a *window size* determined by a user for indexing. The method extracts a time window from every possible position of a timestamped event sequence, and builds a trie [13] from a set of time windows. Then, it creates *iso-depth links*, which connects all items equally apart from the first items of their time windows, by referencing the trie via a depth-first traversal. This index structure enables to directly jump to a qualified descendent node without traversing the tree, and thus provides the capability to support efficient matching of non-contiguous event subsequences. In this paper, however, we point out two problems of this method as follows.

1. This method performs well when a query specifies the exact values for the time intervals between the first and the following events in a query sequence (i.e., the tolerance of 0). When the tolerance is larger than 0, however, it has to access multiple chains of iso-depth links for each event specified in a query sequence. Thus, the performance of the method becomes worse as the tolerance gets larger.
2. When constructing the iso-depth index from a trie, the method assigns a sequential ID to each node of the trie in a depth-first fashion. Such IDs are not changeable once the iso-depth index has been built [14]. Therefore, the method is not appropriate for dynamic situations where events continuously arrive into a sequence.

We identified that the above two problems are mainly due to the structural characteristics of an iso-depth index. Based on this identification, we propose a novel method that replaces it with a multidimensional index. Our method places a time window on every item in a timestamped event sequence. Next, it represents each time window as a rectangle over n-dimensional space by considering the first and last occurrence times of each event type. Here, n is the total number of disparate event types that may occur in a given application. Because there may be a quite large number of rectangles extracted from an event sequence, it builds an R*-tree [3], a widely-accepted multidimensional index, for indexing them. When n is large, we apply a grouping technique to reduce the dimensionality of an index. To retrieve the occurrences of a query pattern from a timestamped event sequence, the proposed method first identifies a small number of candidates by searching an R^*-tree and then picks out true answers from them. We verify the robustness and effectiveness of the proposed method.

2 Problem Definition

In this section, we define the notations necessary for further presentation and formulate the problem of timestamped event sequence matching.

Definition 1: Timestamped event sequence T

T, a timestamped event sequence, is a list of pairs of $(e_i,\ t_i)$ $(1 \leq i \leq n)$ defined as follows.

$$T = \langle (e_1, t_1), (e_2, t_2), ..., (e_n, t_n) \rangle$$

where e_i is an event type and t_i is the timestamp at which e_i has occurred. Also, a pair of $(e_i,\ t_i)$ is referred to as the i^{th} item of T. The number of items in T is denoted as $|T|$. The time interval between the first and the last events, i.e., $t_n - t_1$, is denoted as $\|T\|$. The i^{th} item of T is denoted as T_i, and its event type and timestamp are denoted as $e(T_i)$ and $ts(T_i)$, respectively. A non-contiguous subsequence T' is obtained by eliminating some items from T. Hereafter, we simply call a non-contiguous subsequence a subsequence. We also assume that all the items are listed in the ascending order of their timestamps, i.e., $t_i \leq t_j$ for $i < j$. ■

Definition 2: Query pattern QP

QP, a query pattern, consists of an event list EL and a range list RL defined below.

$$QP.EL = \langle e_1, e_2, ..., e_k \rangle$$
$$QP.RL = \langle [min_1, max_1], [min_2, max_2], ..., [min_k, max_k] \rangle$$

Each element e_i in EL expresses the i^{th} event, and each element $[min_i, max_i]$ in RL denotes the time range within which e_i has to occur following the first event. $|QP.EL|$ is the number of events (=k) in $QP.EL$, and $\|QP\|$ is the time interval covered by QP, which is $max_k - min_1$. Also, $QP.EL_i$ is the i^{th} event in $QP.EL$. ■

Definition 3: Matching of subsequence T' and query pattern QP

Given T' and QP, they are considered to be matched if the following three conditions are all satisfied.

- **Condition 1**: $|T'| = |QP.EL|$
- **Condition 2**: $e(T'_i) = QP.EL_i$ for all i (=1, 2, ..., $|T'|$)
- **Condition 3**: $min_i \leq ts(T'_i) - ts(T'_1) \leq max_i$ for all i (=1, 2, ..., $|T'|$)

■

Definition 4: Timestamped event sequence matching

Timestamped event sequence matching, shortly event sequence matching, is the problem of retrieving from an event sequence T all subsequences T' that are matched to a query pattern QP. ■

3 Proposed Method

This section proposes a new indexing method that overcomes the problems of aforementioned approaches, and suggests an algorithm that uses the proposed indexing method for event sequence matching.

3.1 Indexing

The proposed index construction algorithm is given in Algorithm 1. To support efficient event sequence matching, we first build an empty R*-tree (Line 1). The dimensionality of the tree is equal to the number of disparate event types that can occur in a target application. Next, we extract a time window at every possible position of an event sequence (Line 3). The maximum value of the time intervals covered by query patterns of a target application is used as ξ, the size of a time window. Let DW denote the time window extracted at the i^{th} position of T. Then, we construct a rectangle from DW (Line 4) and insert it into the

Algorithm 1: Index construction.

Input : Event sequence T
Output : R*-tree I

```
1 I := createEmptyRstarTree();
2 for (i=1; i ≤ |T|; i++) do
3     DW := extractTimeWindow(T, i);
4     DR := constructDataRectangle(DW);
5     insertRectangle(I, DR, i);
6 return I;
```

R*-tree using i as its identifier (Line 5). Finally, we return the R*-tree containing all rectangles (Line 6).

Let $E = \{E_1, E_2, ..., E_n\}$ denote a set of n disparate event types. The rectangle from the time window DW is expressed as ($[min_1, max_1]$, $[min_2, max_2]$, ..., $[min_n, max_n]$). Here, min_i $(i = 1, 2, ..., n)$ denotes the difference between $ts(DW_1)$ and $first\_ts(E_i)$ where $ts(DW_1)$ is the timestamp of the first item of DW and $first\_ts(E_i)$ is the timestamp of the first occurrence of event type E_i in DW. Similarly, max_i denotes the difference between $ts(DW_1)$ and $last\_ts(E_i)$ where $last\_ts(E_i)$ is the timestamp of the last occurrence of event type E_i in DW. We assign $\|DW\|$ to both min_i and max_i when there are no instances of event type E_i in DW.

In practice, we maintain multiple R*-trees in order to reduce the search space before starting an index search. More specifically, we insert the rectangle for the time window DW into an R*-tree I_j when the first item of DW is of event type E_j. Therefore, the proposed index structure consists of n R*-trees, I_1, I_2, ..., I_n, and an index table. The index table consists of n entries, each of which points to the root node of the corresponding R*-tree. Since the index table is not large in most cases, it is kept in main memory.

3.2 Event Sequence Matching

As explained in Section 2, a query pattern QP with k events has the following format: $QP.EL = \langle e_1, e_2, ..., e_k \rangle$, $QP.RL = \langle [min_1, max_1], [min_2, max_2], ..., [min_k, max_k] \rangle$. This query pattern requires that the events should occur in the order of e_1, e_2, ..., e_k, and there should be an event e_i within the time range of $[min_i, max_i]$ after the occurrence of the first event e_1. The event sequence matching algorithm that uses the proposed index structure is given in Algorithm 2.

We first construct a query rectangle from a given query pattern (Line 2). The minimum and maximum values of each dimension of a query rectangle are obtained from the corresponding time ranges specified in a query pattern.

Algorithm 2: Event sequence matching.

Input : Query pattern $QP = (\langle e_1, ..., e_k \rangle, \langle [min_1, max_1], ..., [min_k, max_k] \rangle)$, Target index I, Event sequence T
Output : Set of answers A

```
1 A := {};
2 QR := constructQueryRectangle(QP);
3 C := findOverlappingRectangles(I, QR);
4 for (each identifier id ∈ C) do
5     if (isTrueAnswer(T, id)) then
6         addAnswer(A, id);
7 return A;
```

1. If there are no time ranges specified for the event type E_i in a query pattern, then the range of the i^{th} dimension becomes $[0, \xi]$.
2. If there is only one time range specified for the event type E_i in a query pattern, then the minimum and maximum values on that time range become the minimum and maximum values of the i^{th} dimension, respectively.
3. If there are more than one time range specified for the event type E_i in a query pattern, then the time range with the smallest interval determines the range of the i^{th} dimension.

After constructing the query rectangle from a query pattern, we search the multi-dimensional index I for the rectangles overlapping with the query rectangle (Line 3). As mentioned in Section 3.1, each query pattern has an associated R*-tree according to the type of its first event. Therefore, the search is performed only on the R*-tree dedicated to the type of e_1. Its location is obtained by looking up the index table. The post-processing step begins after obtaining a set of candidate rectangles from the index search. Using the identifier of each candidate rectangle, this step reads the event sequence to verify whether the candidate actually matches the query pattern (Line 5). Only the candidate rectangles which are actually matched with the query pattern are added into the result set (Line 6) and then returned to the user (Line 7). The theorem in [10] shows that the proposed matching algorithm retrieves without false dismissal [1,5] all time windows containing subsequences matched with a query pattern.

3.3 Dimensionality Reduction

The proposed index becomes very high dimensional when n is large. To prevent the problem of *dimensionality curse* [15] in this case, we apply *event type grouping* that combines n event types into a smaller number, say m, of event type groups. The composition of m groups from n event types enables the proposed n-dimensional index to be m-dimensional. Such event type grouping, however,

Algorithm 3: Event type grouping.

Input : Set of n event types $E = \{E_1, E_2, ..., E_n\}$, Event sequence T, Number of desired event type groups m
Output : Set of m event type groups G

1 Using T, compute the distance of every pair of event types;
2 Construct a complete graph with n nodes and $n(n-1)/2$ edges;
3 Attach a unique label to each node using the integer values from 1 to n. The node labeled with the integer i represents the event type E_i;
4 Attach a weight to each edge. The distance between E_i and E_j is used as a weight of the edge connecting the node i and the node j;
5 **while** *the number of components in the graph is greater than* m **do**
  Remove from the graph the edge with the largest weight;
6 Extract G, a set of m event type groups, from m components;
7 Return G;

tends to enlarge the rectangles to be stored in the index, lowering a filtering ratio in searching. Therefore, in this paper, we suggest a systematic algorithm (See Algorithm 3) that performs event type grouping with the least enlargement of the rectangles. For this algorithm, we define a distance metric [10] that measures the degree of enlargement incurred by merging each pair of event types.

4 Performance Evaluation

4.1 Experimental Environment

For the experiments, we generated synthetic event sequences with various $|T|$ and n values. The event types were generated uniformly within the range of $[1, n]$, and their interarrival times followed an exponential distribution. We used queries with 3 event items as a standard query pattern. We submitted 100 queries for an individual experiment and computed their average elapsed time. The machine for the experiments was a personal computer with a Pentium-IV 2GHz CPU, the main memory of 512 MB, and the operating system of Windows 2000 Server.

We compared the performances of the following three methods: (1) The first one is the proposed method using a 5-dimensional R^*-tree with the page size of 1KB. (2) The second one is the sequential-scan-based method. (3) The third one is the method based on an iso-depth index.

4.2 Results and Analyses

While fixing the size of the time window at 50, the first experiment compared the proposed index and the iso-depth index in terms of the index size. Figure 2 shows their sizes with increasing data set sizes and different numbers of event types. The X-axis is for the number of items in the event sequence, and the Y-axis is

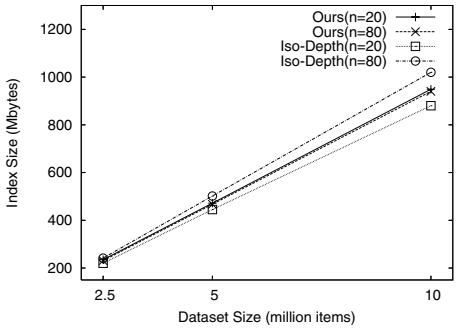

Fig. 2. Sizes of the proposed index and the iso-depth index with increasing data set sizes and different numbers of event types

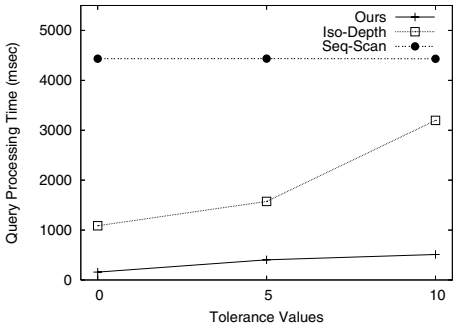

Fig. 3. Query processing times of the three methods with various tolerance values of query patterns

for the index sizes when the event sequence has 2.5 million items (20M bytes), 5 million items (40M bytes), and 10 million items (80M bytes), respectively.

As clearly shown in the figure, both the proposed index and the iso-depth index grow linearly as the data set increases. It is also evident that the number of event types increases the size of the iso-depth index conspicuously but does not affect much the size of the proposed index due to event type grouping.

The second experiment compared the query processing times of the three methods with various tolerance values of query patterns. The data set had 20 event types and 5 million items, and the size of the time window was 50. The tolerance values were set 0% (=0), 10% (=5), and 20% (=10) of the time window size. The result is shown in Figure 3. The X-axis is for the tolerance values and the Y-axis is for the query processing times.

Since the sequential-scan-based method reads the entire event sequence in any cases, its query processing time does not change much with the tolerance values. On the contrary, the query processing times of our method and the iso-depth index increase conspicuously as the tolerance value grows. This is because both methods have to access more portions of the indexes when the tolerance value becomes larger. As compared with the sequential-scan-based method, our method is about 11 and 8.6 times faster when the tolerance values are 5 and 10, respectively. As compared with the iso-depth index, our method is about 3.9 and 6.3 times faster when the tolerance values are 5 and 10, respectively.

The third experiment compared the query processing times of the three methods with various data set sizes. The size of the time window was 50, and the tolerance value of query patterns was 5 (i.e., 10% of the time window size). The number of event types was 20 at first and then 80. The result is shown in Figure 4. The X-axis is for the number of items in the event sequence and the Y-axis is for the query processing times.

The result shows that the query processing times of all three methods increase linearly with the data set sizes. The query processing time of the sequential-scan-

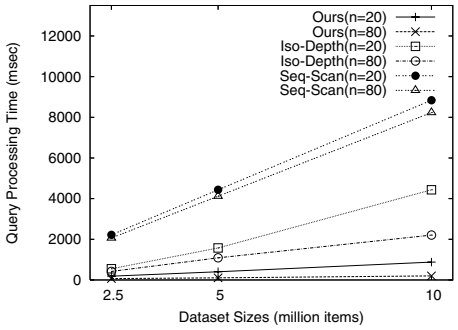

Fig. 4. Query processing times of the three methods with increasing data set sizes and different numbers of event types

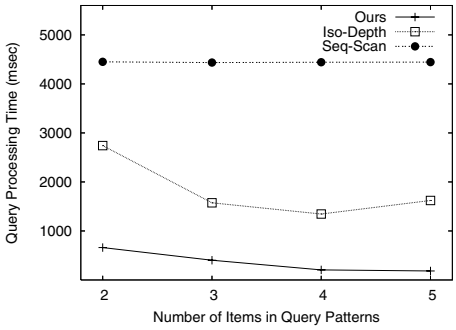

Fig. 5. Query processing times of the three methods with various numbers of event items in query patterns

based method does not change much with the number of event types. However, the query processing times of our method and the iso-depth index decrease when the number of event types increases from 20 to 80. This is because both methods generate a smaller number of candidates as the number of event types grows. As compared with the sequential-scan-based method and the iso-depth index, our method performs about 40.9 times and 10.9 times faster, respectively, when the data set has 10 million items and 80 event types.

The fourth experiment compared the query processing times of the three methods with various numbers of event items in query patterns. The data set had 20 event types and 5 million data items. The size of the time window was 50 and the tolerance value was 5. The result is shown in Figure 5.

The query processing time of the sequential-scan-based method does not change much with the number of events in query patterns. However, the query processing times of our method and the iso-depth index decrease when the number of events in query patterns increases. This is because both methods produce a smaller number of candidates as query patterns have more events inside. Especially in our method, the query rectangles become smaller when the query patterns have more events inside, which significantly reduces the number of candidates retrieved by the index search.

5 Conclusions

Timestamped event sequence matching is useful for discovering temporal causal relationships among timestamped events. In this paper, we have proposed a novel method for effective processing of timestamped event sequence matching. According to the performance results, the proposed method shows a significant speedup by up to a few orders of magnitude in comparison with the previous ones. Furthermore, the performance improvement becomes bigger (1) as toler-

ance ranges get larger, (2) as the number of event types increases, (3) as a data set grows in size, and (4) a query sequence gets longer.

References

1. R. Agrawal, C. Faloutsos, and A. Swami, "Efficient Similarity Searchin Sequence Databases", *In Proc. Int'l. Conf. on Foundations of Data Organization and Algorithms*, FODO, pp. 69-84, 1993.
2. R. Agrawal, K. Lin, H. S. Sawhney, and K. Shim, "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases", *In Proc. Int'l. Conf. on Very Large Data Bases*, VLDB, pp. 490-501, Sept. 1995.
3. N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger, "The R*-tree: An Efficient and Robust Access Method for Points and Rectangles", *In Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, pp. 322-331, 1990.
4. K. K. W. Chu and M. H. Wong, "Fast Time-Series Searching with Scaling and Shifting", *In Proc. Int'l. Symp. on Principles of Database Systems*, ACM PODS, pp. 237-248, May 1999.
5. C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast Subsequence Matching in Time-series Databases", *In Proc. Int'l. Conf. on Management of Data*, ACM SIGMOD, pp. 419-429, May 1994.
6. D. Q. Goldin and P. C. Kanellakis, "On Similarity Queries for Time-Series Data: Constraint Specification and Implementation", *In Proc. Int'l. Conf. on Principles and Practice of Constraint Programming*, CP, pp. 137-153, Sept. 1995.
7. S. W. Kim, S. H. Park, and W. W. Chu, "An Index-Based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases", *In Proc. Int'l. Conf. on Data Engineering*, IEEE ICDE, pp. 607-614, 2001.
8. Y. S. Moon, K. Y. Whang, and W. K. Loh, "Duality-Based Subsequence Matching in Time-Series Databases", *In Proc. Int'l. Conf. on Data Engineering*, IEEE ICDE, pp. 263-272, 2001.
9. S. Park, W. W. Chu, J. Yoon, and C. Hsu "Efficient Searches for Similar Subsequences of Different Lengths in Sequence Databases", *In Proc. Int'l. Conf. on Data Engineering*, IEEE ICDE, pp. 23-32, 2000.
10. S. Park, J. Won, J. Yoon, and S. Kim, "An Index-Based Method for Timestamped Event Sequence Matching", Technical Report, Yonsei University, 2004.
11. D. Rafiei and A. Mendelzon, "Similarity-Based Queries for Time-Series Data", *In Proc. Int'l. Conf. on Management of Data*, ACM SIGMOD, pp. 13-24, 1997.
12. D. Rafiei, "On Similarity-Based Queries for Time Series Data", *In Proc. Int'l. Conf. on Data Engineering*, IEEE ICDE, pp. 410-417, 1999.
13. G. A. Stephen, *String Searching Algorithms*, World Scientific Publishing, 1994.
14. H. Wang, C. Perng, W. Fan, S. Park, and P. Yu, "Indexing Weighted Sequences in Large Databases", *In Proc. Int'l Conf. on Data Engineering*, IEEE ICDE, pp. 63-74, 2003.
15. R. Weber, H.-J. Schek, and S. Blott, "A Quantitative Analysis and Performance Study for Similarity Search Methods in High-Dimensional Spaces", *In Proc. Int'l. Conf. on Very Large Data Bases*, VLDB, pp. 194-205, 1998.
16. B. K. Yi and C. Faloutsos, "Fast Time Sequence Indexing for Arbitrary Lp Norms", *In Proc. Int'l. Conf. on Very Large Data Bases*, VLDB, pp. 385-394, 2000.