LETTER

Linear Detrending Subsequence Matching in Time-Series Databases*

Myeong-Seon GIL†, ***Nonmember***, **Yang-Sae MOON**†a), ***Member***, ***and*** **Bum-Soo KIM**†, ***Nonmember***

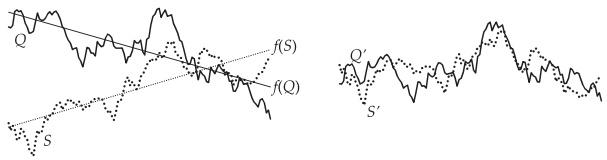
SUMMARY Every time-series has its own linear trend, the directionality of a time-series, and removing the linear trend is crucial to get more intuitive matching results. Supporting the linear detrending in subsequence matching is a challenging problem due to the huge number of all possible subsequences. In this paper we define this problem as the *linear detrending subsequence matching* and propose its efficient *index-based* solution. To this end, we first present a notion of *LD-windows* (LD means linear detrending). Using the LD-windows we then present a lower bounding theorem for the index-based matching solution and show its correctness. We next propose the index building and subsequence matching algorithms. We finally show the superiority of the index-based solution.

key words: *data mining, time-series databases, similar sequence matching, linear detrending, subsequence matching*

1. Introduction

Time-series data are of growing importance in data mining [5]. Typical examples of time-series data include stock prices, music data, moving object trajectories, and biomedical data [1], [3], [9]. Finding *data sequences* similar to the given *query sequence* from the database is called *similar sequence matching* [1], [5]. In many similar sequence matching models, two sequences $X = \{X[1], \ldots, X[n]\}$ and $Y = \{Y[1], \ldots, Y[n]\}$ are said to be *similar* if the distance $D(X, Y) \leq \epsilon$, where ϵ is the user-specified tolerance. In this paper we use the Euclidean distance ($= \sqrt{\sum_{i=1}^{n} |X[i] - Y[i]|^2}$) as the distance function of $D(X, Y)$.

Linear trend, a representative distortion of time-series data [3], shows the directionality of a time-series, and *linear detrending* in similar sequence matching is crucial to get more intuitive matching results. For example, [8] uses the linear detrending to remove the trend variation for comparison of global temperature changes, and [4] detrends time-series of stock prices for correlation analysis of stock items. Figure 1 shows an example of comparing two sequences before and after linear detrending: Fig. 1 (a) represents the original sequences Q and S; Fig. 1 (b) the linear detrended sequences Q' and S'. We obtain Q' and S' by linear detrending, i.e., by subtracting the corresponding trend lines $f(Q)$ and $f(S)$ from Q and S, respectively. In Fig. 1, there is a big distance between Q and S, and so these two sequences will be determined to be non-similar. In contrast, the distance between Q' and S' is very small in Fig. 1 (b), and they will be determined to be similar. This means that non-similar sequences can be identified as similar ones after linear detrending, and vice versa. Likewise, linear detrending is useful to know similarity of changes which is hidden by the linear trend of time-series data [3], [8]. Motivated by this example, we attack the problem of linear detrending in similar sequence matching, especially in subsequence matching [1], [6]. Supporting the linear detrending, however, is a challenging problem in subsequence matching because we need to consider a huge number of all possible data subsequences to be linear detrended. We call this matching scheme the *linear detrending subsequence matching*. For its formal definition, readers are referred to [2].



(a) Original time-series *before* linear detrending. (b) New time-series *after* linear detrending.

Fig. 1 Comparison of two sequences *before* and *after* linear detrending.

We propose an *index-based* solution for linear detrending subsequence matching. To this end, we first present a novel notion of *LD-windows*, linear detrending-windows. Suppose a subsequence $S[i : j]$ includes a window $S[a : b]$ (i.e., $i \leq a < b \leq j$), then we obtain the LD-window of $S[a : b]$ by eliminating the linearity of subsequence $S[i : j]$ rather than that of window $S[a : b]$ itself. This notion enables an LD-window to represent multiple subsequences of different lengths, and eventually, we can use only one index in subsequence matching [7]. Using the LD-windows we next present a lower bounding theorem for the index-based matching solution and prove its correctness. Based on this lower bounding theorem, we then propose the index building and subsequence matching algorithms, respectively. Experimental results show that, compared with the sequential scan, our solution improves the matching performance by one or two orders of magnitude.

2. Linear Detrending Subsequence Matching

For a time-series, its linear trend is a straight line that most likely reflects its directionality. The *least square method* is most widely used to obtain the line of a time-series [8]. For

Manuscript received July 21, 2010.
Manuscript revised November 26, 2010.

†The authors are with the Department of Computer Science, Kangwon National University, Korea.

*This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology (2010-0002518).

a) E-mail: ysmoon@kangwon.ac.kr (corresponding author)

DOI: 10.1587/transinf.E94.D.917

a sequence $X = \{X[1], \ldots, X[n]\}$, a linear function by the least square method is given by $g(k) = \alpha k + \beta$, where α and β are obtained by Eq. (1) [8].

$$\alpha = \frac{n\sum_{k=1}^{n} kX[k] - \sum_{k=1}^{n} k \cdot \sum_{k=1}^{n} X[k]}{n\sum_{k=1}^{n} k^2 - (\sum_{k=1}^{n} k)^2},$$
$$\beta = \frac{\sum_{k=1}^{n} X[k]}{n} - \alpha\frac{\sum_{k=1}^{n} k}{n}. \quad (1)$$

Linear detrending is the process of obtaining a new time-series from an original time-series by removing the corresponding linear trend. The following is the formal definition of linear detrending.

Definition 1: For a sequence $X = \{X[1], \ldots X[n]\}$ and its linear trend function $g(k)$, the *linear detrending sequence* of X, *LD-sequence* of X, is defined as $\overline{X} = \{\overline{X}[1], \ldots, \overline{X}[n]\}$, where $\overline{X}[k] = X[k] - g(k), k = 1, \ldots, n$. □

Linear detrending is simply solved in whole matching, but it is a challenging problem in subsequence matching. In whole matching, the lengths of data and query sequences are all identical, and we simply use the whole matching solution after linear detrending of all time-series. In contrast, the solution is not simple in subsequence matching by the following reasons: (1) data subsequences in different positions have different linear trend even though they have the same length; and (2) data subsequences of different lengths also have different linear trend even though they start from the same position. Therefore, we need to consider different linear trend for all possible query lengths and for all possible positions, and we cannot use the traditional solutions [1], [6] for linear detrending subsequence matching.

We formally define the problem of linear detrending subsequence matching as the following two definitions.

Definition 2: For two sequences X and Y of the same length and their LD-sequences $\overline{X}$ and $\overline{Y}$, we define that X and Y (or $\overline{X}$ and $\overline{Y}$) are *LD-similar* if $D(\overline{X}, \overline{Y}) \le \epsilon$. □

Definition 3: For a data sequence S, a query sequence Q, and the tolerance ϵ, *linear detrending subsequence matching* is the problem of finding all subsequences $S[i : j]$ which are LD-similar to Q, i.e., finding all subsequences $S[i : j]$ such that $D(\overline{Q}, \overline{S[i : j]}) \le \epsilon$. □

A simple solution to linear detrending subsequence matching is the *sequential scan*, which accesses every subsequence $S[i : j]$ sequentially and investigates its LD-similarity by computing $D(\overline{Q}, \overline{S[i : j]})$. Algorithm 1 shows the sequential scan algorithm, LDSeqScan, which is self-explanatory. LDSeqScan is simple, but it incurs severe CPU and I/O overhead due to accessing entire data sequences.

Algorithm 1 LDSeqScan(S, Q, ϵ)

```
1: Compute a line g(k) from Q using the least square method;
2: Obtain Q̄ from Q and g(k) through linear detrending;
3: for each subsequence S[i : j] of length Len(Q) do;
4:     Compute a line g'(k) from S[i : j] using the least square method;
5:     Obtain S̄[i : j] from S[i : j] and g'(k) through linear detrending;
6:     Return the subsequence S[i : j] if D(Q̄, S̄[i : j]) ≤ ε;  // LD-similar
7: end-for
```

3. Proposed Index-Based Solution

As in the traditional subsequence matching [1], [3], [6], we use the window construction mechanism that divides data and query sequences into disjoint/sliding windows of the fixed size. However, our solution quite differs from the traditional ones in constructing windows due to use of linear detrending. Each window should be mapped to multiple windows in the linear detrending subsequence matching while it does not in the traditional subsequence matching. This is because, in linear detrending subsequence matching, each window has multiple trend lines by different lengths and different positions of subsequences that include the window. Formally speaking, for a given window $S[a : b]$, there are many different subsequences $S[i : j]$'s that include $S[a : b]$; their trend lines are also different from each other; and the window $S[a : b]$ is mapped to multiple windows due to different trend lines. We call this complex property the *multiple mapping property*, which was already presented in the normalization-transformed subsequence matching [7]. The traditional solutions [1], [3], [6] do not have the multiple mapping property, but we need to support this property in linear detrending subsequence matching.

To support the multiple mapping property, for a given window, we do not remove the linear trend of the window itself, but we instead remove the linear trend of a subsequence including that window. To this end, we present a notion of LD-windows as follows:

Definition 4: Suppose $S[i : j]$ be a subsequence of a sequence S, $g(k)$ be a linear function of $S[i : j]$, and $S[a : b]$ be a window included in $S[i : j]$, then *LD-windows* of $S[a : b]$ against $S[i : j]$, denoted by $\overline{S_{\{i,j\}}[a : b]}$, is defined as a new window whose entry $\overline{S_{\{i,j\}}[k]}$ $(k = a, a + 1, \ldots, b)$ is set to $S[k] - g(k)$. □

Definition 4 means that a window $S[a : b]$ is mapped to an LD-window $\overline{S_{\{i,j\}}[a : b]}$ by the trend line of a subsequence $S[i : j]$ which includes $S[a : b]$. Because of the multiple mapping property, there are many subsequences $S[i : j]$'s that include $S[a : b]$, and thus, each window $S[a : b]$ is mapped to multiple LD-windows $\overline{S_{\{i,j\}}[a : b]}$'s for different subsequences $S[i : j]$'s.

Like the traditional subsequence matching algorithms [1], [3], [6], our index-based solution first transforms each *high-dimensional* window to a *low-dimensional* point and then stores the point into the multidimensional index. Unlike the traditional algorithms, however, our solution maps each high-dimensional window to a low-dimensional MBR† (minimum bounding rectangle) that bounds multiple low-dimensional points. This is due to the multiple mapping property that a window is mapped to multiple LD-

†An MBR is defined as the smallest rectangle parallel with the axis that completely encloses a time-series [1], [7].

windows. Constructing an MBR from a window is performed as follows: (1) the given window is mapped to multiple LD-windows; (2) LD-windows are transformed to low-dimensional points by the lower-dimensional transformation; and (3) a low-dimensional MBR is constructed by bounding the transformed points. We call this MBR *LD-MBR* and define it as follows:

Definition 5: Suppose s be a window of a sequence S, $\mathbb{S}$ be $\{\overline{s} \mid \overline{s}$ is an LD window of $s\}$, and $T(\cdot)$ be a function of lower-dimensional transformation, then *LD-MBR* of s, denoted by $\mathbb{M}(T(\mathbb{S}))$, is defined as a low-dimensional MBR that bounds all low-dimensional points $T(\overline{s})$ for all $\overline{s} \in \mathbb{S}$. □

Our index-based solution is developed from the following Theorem 1.

Theorem 1: *For a query sequence Q, a data subsequence $S[i:j]$, a tolerance ϵ, a function $T(\cdot)$ of lower-dimensional transformation, if Q and $S[i:j]$ are LD-similar, that is, if $D(\overline{Q}, \overline{S[i:j]}) \le \epsilon$, the distance between $T(\overline{q_k})$ and $\mathbb{M}(T(\mathbb{S}_k))$ $\le \epsilon/\sqrt{p}$, where $\overline{s_1}, \ldots, \overline{s_p}$ and $\overline{q_1}, \ldots, \overline{q_p}$ are p disjoint windows of $\overline{Q}$ and $\overline{S[i:j]}$, respectively, and $\mathbb{S}_k$ is the set of LD-windows of s_k. That is, the following Eq. (2) holds:*

$$D(\overline{Q}, \overline{S[i:j]}) \le \epsilon \Rightarrow \bigvee_{k=1}^{p} D(T(\overline{q_k}), \mathbb{M}(T(\mathbb{S}_k))) \le \epsilon/\sqrt{p}. \quad (2)$$

Proof: The proof is similar to that of normalization-transformed subsequence matching in the previous work [7]. Refer to [7] for the detailed proof. □

Theorem 1 guarantees correctness of our index-based solution to linear detrending subsequence matching. Like the traditional subsequence matching solutions, our index-based solution, called LDIndexMatch, also consists of two algorithms: (1) the index-building algorithm and (2) the subsequence matching algorithm.

Algorithm 2 shows the index-building algorithm. In Line 2 we divide the given data sequence into sliding or disjoint windows of size ω. For the first subsequence matching solution of [1], we use the sliding window; in contrast, for the recent Dual Match [6], we use the disjoint window. In Lines 4 to 14, we build a multidimensional index by repeating the following three steps for each window $S[a:b]$: (1) compute trend lines of all possible subsequences (Line 8); (2) obtain LD-windows using those trend lines (Line 9); and (3) map those LD-windows to an LD-MBR (Line 10). After obtaining an LD-MBR from a window, we store it into the index with its starting offset a (Line 13). Once we build an index by Algorithm 2, we use it repeatedly in the subsequence matching algorithm. However, we here note that the index building process will take much time if we handle a large time-series database. Thus, we leave the detailed analysis on the index building complexity and an efficient solution of constructing the index as the future work.

Algorithm 3 shows the subsequence matching algorithm. In Line 2 we first eliminate the linear trend from the query sequence Q. In Line 3 we divide the LD sequence $\overline{Q}$ into disjoint or sliding windows $\overline{q}$ of size ω. For the first solution of [1], we use the disjoint window; in contrast, for Dual Match [6], we use the sliding window. In Lines 5 to 11, we find candidate subsequences by repeating the following steps for each query window $\overline{q}$: (1) transform a high-dimensional window $\overline{q}$ to a low-dimensional point (Line 6); (2) make a range query using that point and the given tolerance (Line 7); and (3) find candidate subsequences by evaluating the range query on the index (Lines 8 and 9). After obtaining the candidate subsequences, we finally perform the post-processing step [1], [3], [6] to identify true LD-similar subsequences by accessing actual subsequences and eliminating false alarms.

Algorithm 2 LDIndexMatch-BuildIndex (data sequence S)

```
1: Let the window size be ω and the max/min query lengths be l_min, l_max;
2: Divide S into windows of size ω;
3:    // sliding windows for [1]; disjoint windows for Dual Match [6].
4: for each window S[a : b] in S do
5:    Make an f-dimensional MBR M which is initially empty;
6:    for each query length l ∈ [l_min, l_max] do
7:       for each subsequence S[i : j] of length l that includes S[a : b] do
8:          Compute a line of S[i : j] based on the least square method;
9:          Obtain the LD-window S̄_{i,j}[a : b]; // linear detrending
10:         Transform S̄_{i,j}[a : b] to an f-dimensional point and
                include it into M;
11:      end-for
12:   end-for
13:   Make a record <M, a> for S[a : b], and store it into the index;
14: end-for
```

Algorithm 3 LDIndexMatch-Matching (Q, S, ϵ)

```
1: Let the window size be ω; // ω is the same one used in Algorithm 2.
2: Obtain Q̄ from Q by eliminating the linear trend;
3: Divide Q̄ into windows of size ω;
4:    // disjoint windows for [1]; sliding windows for Dual Match [6].
5: for each window q̄ do
6:    Transform q̄ to an f-dimensional point;
7:    Construct a range query using that point and ε/√p;
8:       // p is the number of included windows in Q [6].
9:    Evaluate the query on the index and find the record <M, a>;
10:   Include in the candidate set S[i : j] obtained from <M, a>;
11: end-for
12: Perform the post-processing step [1], [3], [6] to eliminate false alarms;
```

4. Experimental Evaluation

We have performed experiments using three real data sets [3]. The first data set, *ECG-DATA*, contains electrocadiogram data; the second data set, *TAX-DATA*, contains data for tax growth rates; the third data set, *EXCH-DATA*, contains exchange rate data. The length of each data set is 100,000, i.e., each data set consists of 100,000 entries.

In the experiments we have compared two matching solutions: LDSeqScan and LDIndexMatch. For LDIndexMatch, we have adopted the first subsequence matching solution of [1]. In the first experiment we set the window size and the selectivity [1], [6] to 256 and 10^{-3}, respectively, and

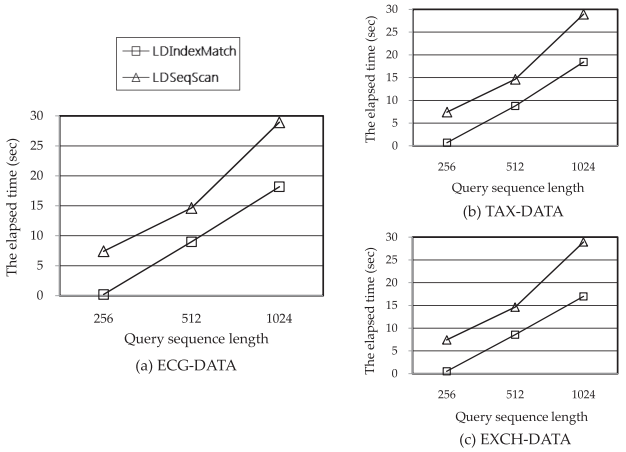


Fig. 2 Experimental results by varying the query sequence length.

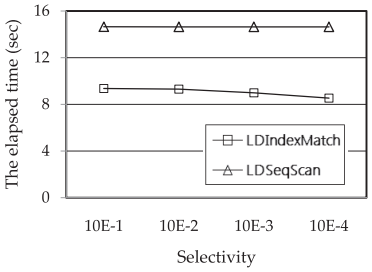


Fig. 3 Experimental result of ECG-DATA by varying the selectivity.

vary the query sequence length from 256 to 1024. In the second experiment we set the window size and the query sequence length to 256 and 512, respectively, and vary the selectivity from 10^{-1} to 10^{-4}. As the metric of efficiency, we measure the elapsed time of each solution. To avoid effects of noise, we experiment with 20 different query sequences of the same length and use their average as the result. In LDIndexMatch, we used PAA [3] as the lower-dimensional transformation and extracted eight features from an window of size 256; we used the R*-tree [1], [6] as the multidimensional index. Refer to [2] for the detailed hardware and software environments.

Figure 2 shows the results of the first experiment that uses different lengths of query sequences. We first note that, in Fig. 2 (a) of ECG-DATA, LDIndexMatch significantly outperforms LDSeqScan. This means that the notion of LD-windows works properly, and it prunes many unnecessary accesses on subsequences at the index level. As shown in Fig. 2 (a), as the query sequence length decreases, the performance difference between two solutions becomes larger. For example, compared with LDSeqScan, LDIndexMatch reduces the elapsed time by 38.0 times for the query sequence of length 256; in contrast, it reduces the elapsed time by 1.60 times only for the query sequence of length 1024. This is explained by the *window size effect* [6] that the performance of index-based solutions decreases as the query sequence length on the given window size increases. We can solve this problem by using the *index interpolation* technique [5] which uses multiple indexes (for multiple window sizes) to obtain the better performance. Figures 2 (b) and 2 (c) of TAX-DATA and EXCH-DATA show the very similar trend with Fig. 2 (a) of ECG-DATA. It means that the proposed LDIndexMatch exploits the pruning effect efficiently, regardless of data types.

Figure 3 shows the result of the second experiment on ECG-DATA that uses different selectivities (i.e., different tolerances). We omit the results of TAX-DATA and EXCH-DATA since they show very similar trend with ECG-DATA. As in Fig. 2, Fig. 3 also demonstrates that LDIndexMatch beats LDSeqScan in all selectivity ranges. This means that LDIndexMatch does not much depend on the selectivity values and shows better performance than LDSeqScan.

5. Conclusions

In this paper we formally defined the linear detrending subsequence matching first. We then presented a novel notion of *LD-windows*, and using LD-windows we next proposed an index-based solution. We provided a formal theorem that guaranteed correctness of our index-based solution. We also described the index-building and subsequence matching algorithms of the index-based solution. We finally showcased that, compared with the straightforward sequential scan, our index-based solution significantly improved the matching performance by one or two orders of magnitude. We believe that the linear detrending subsequence matching and its index-based solution will be very helpful to find meaningful time-series patterns hidden by the linear trend.

References

[1] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, "Fast subsequence matching in time-series databases," Proc. Int'l Conf. on Management of Data, pp.419–429, ACM SIGMOD, Minneapolis, Minnesota, May 1994.
[2] M.-S. Gil, Y.-S. Moon, and B.-S. Kim, "Linear detrending subsequence matching in time-series databases," Computing Research Repository (CoRR), 1006.5273, June 2010.
[3] E.J. Keogh, "A decade of progress in indexing and mining large time series databases," Proc. 32nd Int'l Conf. on Very Large Data Bases, p.1268, Seoul, Korea, Sept. 2006.
[4] Y. Liu, P. Cizeau, M. Meyer, C.-K. Peng, and H.E. Stanley, "Correlations in economic time series," Physica A, vol.245, no.3-4, pp.437–440, Nov. 1997.
[5] W.-K. Loh, Y.-S. Moon, and J. Srivastava, "Distortion-free predictive streaming time-series matching," Inf. Sci., vol.180, no.8, pp.1458–1476, April 2010.
[6] Y.-S. Moon, K.-Y. Whang, and W.-S. Han, "General match: A subsequence matching method in time-series databases based on generalized windows," Proc. Int'l Conf. on Management of Data, ACM SIGMOD, pp.382–393, Madison, Wisconsin, June 2002.
[7] Y.-S. Moon and J. Kim, "Fast normalization-transformed subsequence matching in time-series databases," IEICE Trans. Inf. & Syst., vol.E90-D, no.12, pp.2007–2018, Dec. 2007.
[8] R.H. Shumway and D.S. Stoffer, Time Series Analysis and Its Applications: With R Examples, 2nd ed., Springer Texts in Statistics, 2006.
[9] L. Singh and M. Sayal, "Privately detecting bursts in streaming, distributed time series data," Data and Knowledge Engineering, vol.68, no.6, pp.509–530, June 2009.