Noname manuscript No.
(will be inserted by the editor)

RLC: Ranking Lag Correlations with Flexible Sliding Windows in Data Streams

Received: date / Accepted: date

Abstract Lag correlation refers to the correlation between two time series shifted in time relative to one another. Existing research on lag correlation is mainly based on the entire stream or the current substream. However, the entire stream cannot get rid of the stale data or highlight the importance of the recent data and there is no steady way to define the proper substream length on different application needs. In this paper, we propose a new way to analyze the lag correlation which is based on flexible sliding windows. In view of that, a new query called RLC(Ranking Lag Correlations with flexible sliding windows in data streams)is proposed. The key challenge in answering the RLC query is that the number of window lengths to be analyzed will grow quadratically with the growth of the stream, giving rise to a quadratic increase in computation cost. To tackle the challenge, we use the running sum technique and the geometric probing idea to dramatically reduce the computation cost of lag correlation. This work presents a comprehensive study to this problem by designing an exact as well as an approximate solution. The extensive experiments validate the efficiency of our proposed methods. Some insightful lag correlations discovered from real datasets demonstrate the practicality of this work.

Keywords Lag correlation · Flexible sliding windows · Data streams

1 Introduction

Time series are used in statistics, signal processing, pattern recognition, econometrics, mathematical finance, weather or earthquake forecasting, intelligent transport, etc. As a result, the processing and analysis of time series have received much attention. Of particular interest within this realm are the problems of correlation analysis [3], [4], [12], [13], [14]. Traditional correlation

analyses between two time series are aligned in time [13], [14]. However, in many real-world applications, such as traffic system and climate, the time lag is a key feature of hidden temporal dependencies [23]. Lag correlation refers to the correlation between two time series with a time lag.

Most of current methods for lag correlation analysis in a relatively crude manner: the entire stream [18], [23] or the current substream based on the sliding window [20], [21]. Considering the entire stream cannot get rid of the stale data or emphasize the importance of the recent data. Even though the sliding window considers the recent data, it only works within the window of a fixed length. Unfortunately, there is no steady way to define the proper length on different application needs and it is almost impossible to give the window length that is the most appropriate for each pairwise stream. Hence, no single optimal choice for the parameter exists, as the window length heavily depends on queries, time, data, and application domains. In addition, the most appropriate length may be stream-dependent. Actually, the presence of free window length represents additional burden on the user instead of providing more freedom.

The RLC query in this paper is to rank lag correlations with flexible sliding windows between a given query stream and multiple objective streams. Let's focus on two streams first. Given a query stream q and an objective stream o, a substream pair is maximum lag correlation if and only if the lag correlation $R((q,\tau_q,l),(o,\tau_o,l),w)$ is maximized subject to the length of the sliding window $w_{min} \leq w \leq m$ and the length of the substream $w/2 \leq l \leq w$, where $R((q,\tau_q,l),(o,\tau_o,l),w)$ is the lag correlation between q in the segment $[\tau_q,\tau_q+l-1]$ and o in the segment $[\tau_o,\tau_o+l-1]$, m represents the maximum length of the stream data at current time t and w_{min} represents the minimal window length given by users. Surprisingly, there has been limited work on this problem.

1.1 Motivation

The RLC query is particularly useful in helping those analyses without priori knowledge of the query window length. For instance, a stock analyst wants to find the top-2 lagged correlated stocks to Google in recently 50 days. This question can be answered easily by the method which can find lag correlations in a sliding window with a fixed length in [20], [21] since the window length has been given. The query result is shown in Table 1. The query result may be more meaningful to analysts if it becomes the RLC query, i.e., 'find top-2 lagged correlated stocks to Google in a flexible sliding window whose length is no less than 50 days given by the stock analyst'. The query result is shown in Table 2. We claim that the minimal length is a more natural parameter than a fixed appropriate length since analysts can evaluate the lower limit in their application domain. From these tables, we have three discoveries as follows:

Firstly, setting the most appropriate window length with maximum lag correlation for one pairwise stream is difficult. For example, the window length

50 days given by the fixed window length query is not appropriate for Google and Amazon. Because the length of maximum lag correlation between Google and Amazon should be 630 days, which can be automatically identified by our RLC query.

Secondly, the most appropriate windows for the different pairwise streams are different. The most appropriate sliding window in which has the maximum lag correlation between Google and Apple is the sliding window [04/02/2012, 31/12/2012] whose length is 696 days, while the most appropriate sliding window between Google and Amazon is the sliding window [11/04/2012, 31/12/2012] whose length is 630 days. So it is not possible to fix a window length that is most appropriate for each pairwise stream.

Finally, the returned stocks by these two queries are different.

Above all, it is obvious that the result of our RLC query is more meaningful and suitable in real applications.

Table 1 The top-2 query of fixed length 50 days

Rank	sliding window	correlation	lagged length	query subseries	objective subseries
1	**50 days** [12/11/2012, 31/12/2012]	0.834	14 days	Google [12/11/2012, 17/12/2012]	**Amazon** [26/11/2012, 31/12/2012]
2	**50 days** [12/11/2012, 31/12/2012]	0.832	0 days	Google [12/11/2012, 31/12/2012]	**Netflix** [12/11/2012, 31/12/2012]

Table 2 The top-2 RLC query of minimal length 50 days

Rank	sliding window	correlation	lagged length	query subseries	objective subseries
1	**696 days** [04/02/2011, 31/12/2012]	0.985	302 days	Google [04/02/2011, 04/03/2012]	**Apple** [03/12/2011, 31/12/2012]
2	**630 days** [11/04/2011, 31/12/2012]	0.965	298 days	Google [11/04/2011, 09/03/2012]	**Microsoft** [03/02/2012, 31/12/2012]

1.2 Challenges

The RLC query, i.e., ranking lag correlations with flexible sliding windows between a query stream and multiple objective streams, poses great challenges.

To begin with, the length of the sliding window, i.e., the time span which will be computed lag correlation, is variable-length and the range of the lengths is very large, from the minimal length given by users to m(the maximum length of all streams at current time t). As shown in [13], the hourly stock price variances may be longer than ∼2500 per year.

Next, the lagged length l_{ag} is also unknown and its maximum value is set to the half of the window length as suggested in Box et al. [2]. The range of the lagged lengths will expand as the length of the sliding window increases. The lag correlation between pairwise substreams must be re-computed for each lagged length. Due to the variable-length window and lag, the high computational complexity makes the design of an efficient solution difficult.

Finally, another key challenge in the RLC query is the number of objective streams, giving rise to an increase computation cost. As shown in [13], the number of signatures in hyperspectral image databases may be more than $100k$.

1.3 Our Proposals

In this paper, an exact algorithm Prog_RLC and an approximate algorithm Appro_RLC are proposed.

Prog_RLC Algorithm(cf. Section 3): The lag correlation of a subseries pair can be computed in $O(1)$ time, by utilizing running sums technology [12], [13], [14]. That is to say, Prog_RLC Algorithm leverages on the previous computations to compute lag correlations (see Lemma 1 of Section 3.2). Hence, Prog_RLC Algorithm only uses $O(Cnm)$ to find the exact results, where C is the complexity of a correlation computation, n and m are the cardinality and the maximum length of streams.

Appro_RLC Algorithm(cf. Section 4): Instead of computing lag correlation for every possible value of the lag, i.e., $O(m)$, we propose to keep track of a geometric progression sequence of values, where each value after the first value is found by multiplying the previous one by a fixed, non-zero number. In this algorithm, we adapt a geometric progression with the lags, $0, 1, 2, 4, ..., 2^i, ...$, which only needs to compute $O(\log m)$ lag correlations (cf. Observation 1 of Section 4). This technique can achieve a dramatic reduction in computation time and a small error when the lag is small, because we have many points to interpolate when the lag is small. However, it leads to larger error when the lag is large. It is obvious that a new maximum lag correlation may emerge at maximum lag values of every sliding window. That is because that the effect caused by the new data point is largest when the length computed is shortest, hence a new maximum lag correlation may emerge at the largest lag values of every sliding window, which has the shortest length to compute. Fortunately, the estimation error caused by Observation 1 can be reduced by considering this technique (cf. Observation 2 of Section 4). We propose an approximate algorithm, Appr_RLC based on these two observations and running sum technique.

1.4 Summary of Contributions and Outline

Our major contributions can be summarized as follows:

- We propose a novel and general RLC query, i.e., ranking lag correlations with flexible sliding windows between a query stream and multiple objective streams. The lag correlation with flexible sliding windows between two streams is defined as its maximum lag correlation over all possible windows whose lengths satisfy a minimal length constraint.
- We study a progressive computation by the running sums technique (see Lemma 1) and propose an exact algorithm called Prog_RLC that leverages on the previous computations to compute lag correlations. Instead of computing lag correlation for every possible value of the lag, only $O(\log m)$ lag values are tracked (see Observation 1). Moreover, the estimation error is reduced by considering Observation 2. Base on these two observations and running sum, an approximate Appro_RLC algorithm is proposed.
- Extensive experiments on synthetic and real datasets verify the efficiency of our proposed methods. We also show some insightful lag correlations discovered from real datasets, to highlight the practicality of this work.

The rest of this paper is organized as follows. In Section 2, we introduce the notations and formally define the problem. We present an exact as well as an approximate solution in Section 3 and Section 4, respectively. We provide the experiment results in Section 5. The related work is reviewed in Section 6, followed by the conclusions in Section 7.

2 Problem Formulation

Table 3 lists the main symbols we use throughout this paper. We consider a set of N synchronized streams $\{S^1, S^2, ..., S^N\}$, where each stream $S^i = (S^i[1], S^i[2], ..., S^i[t])$ is a sequence of discrete observation at the current time t.

Table 3 Symbols and Definitions

Symbol	Definition and Description
S^i	the ith data stream
$S^i[j]$	The value of S^i at time j
$S^i(\tau, l)$	the substream in stream S^i,i.e., $S^i(\tau, l) =< S^i[\tau], ..., S^i[\tau + l - 1] >$
w_{min}	the minimal length of sliding window
w	the length of the sliding window which is being considered
n	the number of all objective streams
m	the maximum length of all the streams at current time t
$A_i[x]$	the sum of all the values in $S^i(0, x + 1)$, $A_i[x] = S^i[0] + ... + S^i[x]$
$A_i^2[x]$	the sum of squares in $S^i(0, x + 1)$, $A_i^2[x] = S^i[0]^2 + ... + S^i[x]^2$
$\mu^i(\tau, l)$	the mean values in $S^i(\tau, l)$
$\sigma^i(\tau, l)$	the standard deviations in $S^i(\tau, l)$

Different correlation measures can be employed for this problem, such as the Pearson correlation coefficient, extended Jaccard coefficient, etc. In this paper, we use the Pearson correlation coefficient to illustrate our methods.

The definition of Pearson correlation between two substreams on a specific lag l_{ag} is defined by Definition 1.

Definition 1 (**Lag correlation for a specific lag**) At the current time t, the lag correlation between S^q and S^i at lag l_{ag} in the sliding window of length w, denoted as $R_w^{qi}(l_{ag})$.

$$R_w^{qi}(l_{ag}) = \begin{cases} \frac{\sum_{j=0}^{l-1} S^q[\tau_q+j]S^i[\tau_i+j] - l\mu^q(\tau_q,l)\mu^i(\tau_i,l)}{l\sigma^q(\tau_q,l)\sigma^l(\tau_i,l)} & l_{ag} \geq 0 \\ R_w^{qi}(-l_{ag}) & l_{ag} < 0 \end{cases} \quad (1)$$

where $\tau_q = t - w + 1$, $\tau_i = t - w + 1 - l_{ag}$, and $l = w - l_{ag}$.

The lagged length l_{ag} is unknown and its maximum value is set to the half of the window length. The definition of Pearson correlation between two substreams during a sliding window is defined by Definition 2.

Definition 2 (**Lag correlation for a specific sliding window**) At the current time t, the lag correlation between S^q and S^i in the sliding window of length w, denoted as R_w^{qi}, is defined as its maximum lag correlation over all lagged lengths.

$$R_w^{qi} = max_{l_{ag}=0,\ldots,w/2}\{R_w^{qi}(l_{ag})\} \quad (2)$$

Definition 3 (**Lag correlation with flexible sliding windows**) At the current time t, given the minimal length w_{min} of the sliding window, the lag correlation between S^q and S^i, denoted as R^{qi}, is defined as its maximum lag correlation over all possible windows whose lengths satisfy a minimal length constraint w_{min}.

$$R^{qi} = max_{w=w_{min},\ldots,t}\{R_w^{qi}\} \quad (3)$$

Noted that, we will return the longer subsequence if the lag correlations of two pairs of subsequences are equal.

Problem Definition. Given a query stream S^q and a set of objective streams $\{S^1, S^2, ..., S^N\}$, and a minimal length w_{min} of sliding window, the RLC query return top-k pairs of substreams $(S^q(\tau_q, l), S^i(\tau_i, l))$ according to R^{qi} at current time t. We will explain our problem by Example 1.

Example 1. Given a query stream $S^q =< 100, 101, 102, 1, 2, 13, 1, 9, 17, 18 >$, an objective stream $S^o =< 22, 23, 24, 101, 32, 19, 108, 1, 2, 3 >$ (Here we use an objective stream just for ease of expression), a minimal length of sliding window $w_{min} = 8$, result size $k = 2$. To answer the RLC query, we need to calculate lag correlations as follows:

$$R_8^{qo} = max\{R_8^{qo}(0), R_8^{qo}(1), R_8^{qo}(2), R_8^{qo}(3), R_8^{qo}(4)\} = R_8^{qo}(4) = 0.995$$

$$R_9^{qo} = max\{R_9^{qo}(0), R_9^{qo}(1), R_9^{qo}(2), R_9^{qo}(3), R_9^{qo}(4)\} = R_9^{qo}(4) = 0.734$$

$$R_{10}^{qo} = max\{R_{10}^{qo}(0), R_{10}^{qo}(1), R_{10}^{qo}(2), R_{10}^{qo}(3), R_{10}^{qo}(4), R_{10}^{qo}(5)\} = R_{10}^{qo}(4) = 0.6823$$

Therefore, $R_8^{qo}(4) = 0.995$ and $R_9^{qo}(4) = 0.734$ are returned as the result of top-2 RLC query.

3 Answering the RLC Query Exactly

3.1 Adapting State-of-the-art Techniques

We first introduce a state-of-the-art solution, which is based on approach proposed in [20], [21]. Since the length of the sliding window is fixed in [20], [21], we have adapted it to the flexible sliding windows, which we dub SOTA algorithm. SOTA calculates the correlations between query and objects for every possible subsequence combination, i.e., all windows and corresponding lags. The pseudo code is shown in Algorithm 1. The complexity of SOTA is $O(Cnm^2)$.

Algorithm 1 SOTA algorithm

Input: a query stream S^q, a set of objective streams $\{S^1, S^2, ..., S^N\}$, a minimal length w_{min} of sliding window, up to current time t, result size k
Output: top-k pairs of substreams $(S^q(\tau_q, l), S^i(\tau_i, l))$ according to R^{qi}

```
1: Result set RS = ∅
2: Update statistics needed for correlation computation
3: for every stream S^i, i = 1 to N do
4:     for every sliding-window length w = w_min to t do
5:         for every lagged length l_ag = 0 to w/2 do
6:             compute R_w^qi(l_ag) entirely                    # O(l_ag)
7:         end for
8:         R_w^qi = max_{l_ag=0,...,w/2}{R_w^qi(l_ag)}
9:     end for
10:    R^qi = max_{w=w_min,...,t}{R_w^qi}
11:    if |RS| < k or R^qi > kth result in RS then
12:        Insert R^qi into RS
13:    end if
14: end for
15: return RS as result
```

3.2 An progressive algorithm

SOTA is computationally intensive, particularly in evolving time series. Hence, we need to reduce the computational cost. The Pearson correlation for a pair $(S^q(\tau_q, l), S^i(\tau_i, l))$ for the same offset, i.e., $\tau_q = \tau_i$, can be computed in $O(1)$ time, by utilizing running sums [12], [13], [14]. We find that the lag correlation for a pair $(S^q(\tau_q, l), S^i(\tau_i, l))$ for the different offsets, i.e., $\tau_q \neq \tau_i$, can also be computed in $O(1)$ time by utilizing running sums, which is shown in Lemma 1.

Lemma 1 *Given the product sum, i.e.,* $\sum_{j=0}^{l-1} S^q[\tau_q + j]S^i[\tau_i + j]$*, and two running sum arrays, i.e.,* $A_i[x]$ *and* $A_i^2[x]$*, the lag correlation* $R_{w+1}^{qi}(l_{ag})$ *can be calculated progressively from* $R_w^{qi}(l_{ag})$ *in* $O(1)$ *time.*

Proof The lag correlation is defined by Equation 1. We keep two running sum arrays for every stream S^i as follows:

$$A_i[x] = \sum_{j=0}^{x} S^i[j], A_i^2[x] = \sum_{j=0}^{x} S^i[j]^2, 0 < x < m \tag{4}$$

By using these two running sum arrays, the mean and standard deviation of a substream can be calculated in $O(1)$ time, which are shown in Equation 5 and Equation 6, respectively.

$$\begin{aligned} \mu^i(\tau_i - 1, l + 1) &= \frac{1}{l+1}(\sum_{j=\tau_i-1}^{\tau_i+l-1} S^i[j]) \\ &= \frac{1}{l+1}(A_i[\tau_i + l - 1] - A_i[\tau_i - 1] + S^i[\tau_i - 1]) \end{aligned} \tag{5}$$

$$\begin{aligned} \sigma^i(\tau_i - 1, l + 1) &= \frac{1}{l+1}(\sum_{j=\tau_i-1}^{\tau_i+l-1} S^i[j]^2) - \mu^i(\tau_i - 1, l + 1)^2 \\ &= \frac{1}{l+1}(A_i^2[\tau_i + l - 1] - A_i^2[\tau_i - 1] + S^i[\tau_i - 1]^2) - \mu^i(\tau_i - 1, l + 1)^2 \end{aligned} \tag{6}$$

To calculate the lag correlation in constant time, we need to calculate the *product sum* (i.e., $\sum_{j=0}^{l-1} S^q[\tau_q + j]S^i[\tau_i + j]$) of Equation 1 in $O(1)$ time. We can maintain the *product sum* in $O(1)$ time progressively by Equation 7.

$$\sum_{j=0}^{l} S^q[\tau_q - 1 + j]S^i[\tau_i - 1 + j] = \sum_{j=0}^{l-1} S^q[\tau_q + j]S^i[\tau_i + j] + S^q[\tau_q - 1 + j]S^i[\tau_i - 1 + j] \tag{7}$$

In summary, given $\sum_{j=0}^{l-1} S^q[\tau_q + j]S^i[\tau_i + j]$ and two running sum arrays, $R_{w+1}^{qi}(l_{ag})$ can be calculated progressively from $R_w^{qi}(l_{ag})$ in $O(1)$ time. Similarly, $R_{w-1}^{qi}(l_{ag})$ can be also calculated progressively from $R_w^{qi}(l_{ag})$ in $O(1)$ time. ■

By Lemma 1, we can calculate lag correlations of different pairs of substreams for the same lag l_{ag} (i.e., $R_{w_{min}}^{qi}(l_{ag})$, $R_{w_{min}+1}^{qi}(l_{ag})$, ..., $R_t^{qi}(l_{ag})$) progressively. That is to say, for the same lag l_{ag}, lag correlations of all the pairs of substreams during all sliding windows can be calculated in $O(1)$ time.

Inspired by the running sum technique, we study a progressive algorithm, Prog_RLC, to answer the RLC query. The pseudo code is shown in Algorithm 2. Note that we do not need to compute each lag correlation $R_w^{qi}(l_{ag})$. For each l_{ag}, we calculate the lag correlation of the first sliding window from scratch in lines 6-7 and progressively derive the lag correlation of the remaining sliding windows with the same l_{ag} in line 9.

Algorithm 2 Prog_RLC algorithm

Input: a query stream S^q, a set of objective streams $\{S^1, S^2, ..., S^N\}$, a minimal length w_{min} of sliding window, up to current time t, result size k
Output: top-k pairs of substreams $(S^q(\tau_q, l), S^i(\tau_i, l))$ according to R^{qi}

```
 1: Result set RS = ∅
 2: Update statistics needed for correlation computation
 3: for every stream S^i, i = 1 to N do
 4:    for every sliding-window length w from w_min to m do
 5:       for every lagged length l_ag = 0 to w/2 do
 6:          if R_w^qi(l_ag) is firstly computed for l_ag then
 7:             compute R_w^qi(l_ag) entirely                    # O(l_ag)
 8:          else
 9:             compute R_w^qi(l_ag) progressively by Lemma 1    # O(1)
10:          end if
11:       end for
12:       R_w^qi = max_{l_ag=0,...,w/2}{R_w^qi(l_ag)}
13:    end for
14:    R^qi = max_{w=w_min,...,t}{R_w^qi}
15:    if |RS| < k or R^qi > kth result in RS then
16:       Insert R^qi into RS
17:    end if
18: end for
19: return RS as result
```

4 Answering the RLC Query Approximately

Although important, the running sum technique still needs linear time to compute all the lag correlations for a sliding window. In order to reduce the computation time, an approximation is applied. Our approximate solution is based on two observations.

Observation 1. We compute the lag correlation at values of l_{ag} form a geometric progression. Thus, we need only $O(\log m)$ numbers to estimate $R_w^{qi}(l_{ag})$.

Observation 1 is similar to Observation 2 in [18], instead of computing correlation for every possible value of the lag l_{ag}, a geometric progression of the lag values: $l_{ag} = 0, 1, 2, 4, ..., 2^i, ...$, are tracked. This technique can achieve a dramatic reduction in computation time and a small error when the lag is small, but there is sometimes large error when the lag is large, exactly because we have many points to interpolate when the lag is small.

Inspired by [20], [21] and a lot of experiments, we have Observation 2 as follows, which can reduce the estimation error.

Observation 2. A new maximum lag correlation may emerge at maximum lag values of every sliding window.

Observation 2 can be explained as follows. For a specific sliding window of length w, the lag correlation $R_w^{qi}(l_{ag})$ is computed on the window of length $(w - l_{ag})$ at time t. With the increase of l_{ag} from 0 to $w/2$, the length of substream, on which $R_w^{qi}(l_{ag})$ is computed, will decrease. Compared with the previous time $t-1$, we will add a new data point to and delete an old data point from the sliding window. This causes the value of $R_w^{qi}(l_{ag})$ at time t to be different from $R_w^{qi}(l_{ag})'$ at time $(t-1)$. It is obvious that the effect caused by the new data point is different at different lag l_{ag}. With the increase of l_{ag}, the length of the substream, on which $R_w^{qi}(l_{ag})$ is computed, will become smaller, hence the effect of the new data point will be larger, which results in a larger difference of $R_w^{qi}(l_{ag})$ and $R_w^{qi}(l_{ag})'$. This explains why a new maximum lag correlation may emerge at maximum lag values of every sliding window. By using Observation 2, we can compute the maximum lag values of every sliding window to reduce the error caused by Observation 1.

Considering the above two observations and running sum technique (see Lemma 1) in our approximate solution, an approximate algorithm, Appr_RLC, is presented in Algorithm 3. It first checks all the lag values for each sliding window in a geometric progression style in lines 6-13. Since a new maximum lag correlation may emerge at maximum lag values of every sliding window (see Observation 2), so we need to compute the lag correlation at the maximum lag value in lines 14-15 to reduce the error caused by the geometric progression.

5 Discussions

5.1 Selection of the minimal window size

The minimal length is an important parameter which should be set based on the characteristic of data streams and applications. If the value is too large, some meaningful correlations will be ignored. The returned correlation may be very small, which is meaningless. On the other hand, if the value is too small, the trend of correlations between two sequences cannot be discovered.

Moreover, the minimal length is a more natural parameter than a fixed length since analysts can evaluate the lower limit in their application domain.

5.2 Variant of the flexible window

The flexible window can be defined in diverse approach. Not only the lag correlation of sequences no less than the given length can be of interest for an analyst, also the lag correlation of sequences *no more than the given length* can sometimes reveal interesting knowledge. However, ranking the lag correlation

Algorithm 3 Appr_RLC algorithm

Input: a query stream S^q, a set of objective streams $\{S^1, S^2, ..., S^N\}$, a minimal length w_{min} of sliding window, up to current time t, result size k
Output: top-k pairs of substreams $(S^q(\tau_q, l), S^i(\tau_i, l))$ according to R^{qi}

Result set $RS = \emptyset$
Update statistics needed for correlation computation
for every stream S^i, $i = 1$ to N **do**
  for every sliding-window length w from w_{min} to m **do**
    $l_{ag} = 0$
    while $l_{ag} \leq w/2$ **do**
      if $R_w^{qi}(l_{ag})$ is firstly computed for l_{ag} **then**
        compute $R_w^{qi}(l_{ag})$ entirely # $O(l_{ag})$
      else
        compute $R_w^{qi}(l_{ag})$ progressively by Lemma 1 # $O(1)$
      end if
      $l_{ag} = l_{ag} \times 2$
    end while
    if $R_w^{qi}(\frac{w}{2})$ is not computed **then**
      compute $R_w^{qi}(\frac{w}{2})$
    end if
    $R_w^{qi} = max_{l_{ag}=0,...,w/2}\{R_w^{qi}(l_{ag})\}$
  end for
  $R^{qi} = max_{w=w_{min},...,t}\{R_w^{qi}\}$
  if $|RS| < k$ or $R^{qi} >$ kth result in RS **then**
    Insert R^{qi} into RS
  end if
end for
return RS as result

of sequences no more than the given length does not pose a new challenge. It is not difficult to find that the running sums and geometric progression technologies can also be applied to this problem, hence our proposed algorithms can still solve it.

Moreover, the minimal lag correlation with no less (more) than the given length maybe also meaningful. Nevertheless, this new problem does not also bring about a new challenge.

6 Experiments

In this section, we present an evaluation of the proposed algorithms on synthetic and real datasets. All the algorithms are implemented in C++ and compiled using Microsoft Visual Studio 2010 complier. All the experiments are conducted on a 3.20 GHz Pentium PC machine with 8GB main memory running Windows 7 Professional Edition. The experiments are designed to answer the following three questions:

- *Efficiency*: how does the proposed algorithms scale with different values of each parameter w.r.t the response time?
- *Effectiveness*: how the proposed RLC query is useful in real applications?

– *Estimation error of the approximate algorithm*: how accurate is the approximate algorithm Prog_RLC?

6.1 Datasets

Table 4 shows the ranges of the investigated parameters and their default values (in bold). In each experiment, we vary a single parameter, while setting the others to their default values. We use a workload of 100 queries to evaluate the performance in all experiments and we compare the average response time of the query workload for each algorithm.

Table 4 Range of parameter values

Parameter	Values
Number of objective streams, n	Synthetic: **1k**, 5k, 10k, 15k, 20k Tao: **28399** Stock: **2187** Kursk: **200**
Sequence length, m	Synthetic: 2500, **5000**, 7500, 10000 Tao: **1008** Stock: **1037** Kursk: **300**
The minimal length, w_{min}	**1%m**, 5%m, 10%m, 15%m, 20%m
The result size, top-k	**1**, 2, 4, 8, 16

We perform experiments on the following synthetic data sets shown in [13] and real data sets shown in [18].

This synthetic data sets is generated by a random walk technique [1]. For each stream S^i, we randomly generate the first value (i.e., $S^i[0]$) in [-1.0, 1.0]. Each stream value of S^i is generated by $S^i[j+1] = S^i[j] \times (1+N(\mu,\sigma))$, where $N(\mu,\sigma)$ is a normal distribution function. By default, we set the mean $\mu = 0$ and the standard deviation $\sigma = 0.2$.

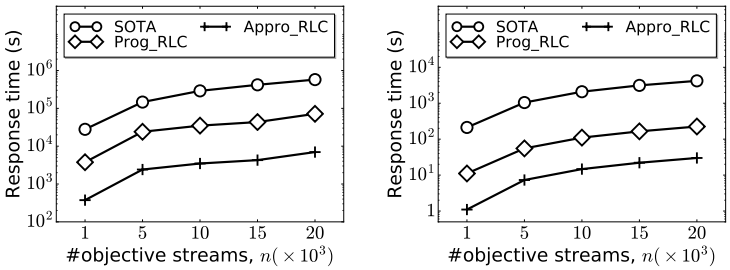
Real datasets used in the experiments are as follows:

– **Tao**: It contains 28399 streams of weekly sea surface temperature of 55 array sensors on the Tropocal Atmosphere Ocean [2]. The length of each stream is 1008.
– **Stock**: It contains 2187 streams of Daily closing prices in NYSE from 2008 to 2012 [3]. The length of each stream is 1037.
– **Kursk**: It contains seismic recording from multiple sensors, showing the explosion of the Russian submarine 'Kursk' [9]. Each stream has a single burst. We extract a stream of length $m = 300$ as a query stream and 200 streams as objective streams.

[1] Monte Carlo simulated stock price generator. http://25yearsofprogramming.com/blog/20070412c-montecarlostockprices.html
[2] http://www.pmel.noaa.gov/tao/
[3] http://finance.yahoo.com

6.2 Efficiency

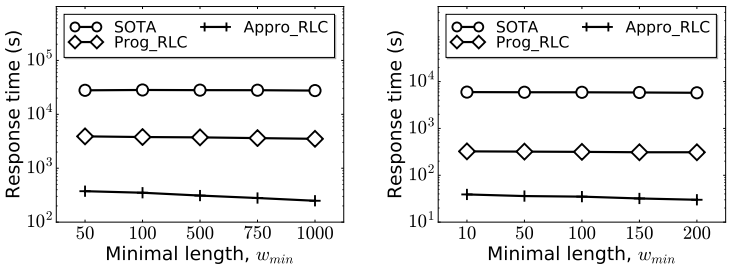
In this section, all experiments compare three algorithms (SOTA algorithm based on the algorithm proposed in [20], [21], our exact algorithm Prog_RLC and our approximate algorithm Appr_RLC) in terms of response time under varying parameter values.



(a) on Synthetic (b) on Tao

Fig. 1 Effect of the number of objective streams, n

Effect of the number of objective streams. Figure 1 shows that the response time of three algorithms as a function of the number of objective streams n, after setting all other parameters to their default values on Synthetic and Tao, respectively. The trend on Synthetic is similar to that on Tao. Not surprisingly, the response time increases linearly as n becomes larger for all algorithms. From Figure 1(a) and Figure 1(b), we can also find that Prog_RLC is up to one order of magnitude faster than SOTA, and Appro_RLC is up to about two orders of magnitude faster than SOTA.



(a) on Synthetic (b) on Tao

Fig. 2 Effect of the minimal length, w_{min}

Effect of the minimal length. Figure 2 shows the response time versus the minimal lengths w_{min}. Intuitively, the response time decrease slowly as the value of w_{min} increases. Again, the performance of Appro_RLC is superior to

Prog_RLC and SOTA. As an example, when w_{min} is 200 on Figure 2(b), Appro_RLC only costs 30.11 seconds to find the lag correlations, while Prog_RLC costs 310.23 seconds and SOTA costs 5768 seconds.

Effect of k. The response time of the algorithms versus k is shown in Figure 3. The response time increases linearly with k, as more results are discovered. Our Appro_RLC still performs the best, followed by Prog_RLC and SOTA.

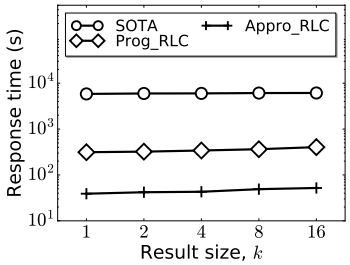


Fig. 3 Effect of result size on Tao

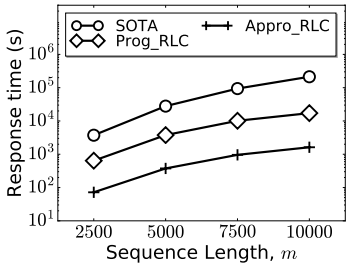


Fig. 4 Effect of sequence length on Synthetic

Effect of the sequence length. Figure 4 shows the response time versus stream length m. The longer the sequence length, the more Appro_RLC and the Prog_RLC have the advantage. For instance, Prog_RLC (Appro_RLC) is 15 (150) times faster than SOTA when m is 10000. Hence, the improvement of performance grows as the value of m increases for our proposed algorithms.

6.3 Estimation error of the approximate algorithm

Since all the lag correlations can be detected in the exact Prog_RLC algorithm, in which estimation errors of the captured lag correlations is 0%.

We will define the estimation error of our proposed approximate algorithm Appr_RLC as follows.

Let us focus on the Top-1 lag correlation between a query stream and multiple objective streams.

If the returned objective stream by the Appr_RLC algorithm is different from that by the exact algorithm, then we think $Error_{stream} = 100\%$.

If the objective stream returned by approximate algorithm is the same to that returned by exact algorithm, then $Error_{stream} = 0\%$. The window $W_{exact} =< S_{exact}, E_{exact} >$ represents the exact window in which lag correlation is maximum. The window $W_{appr} =< S_{appr}, E_{appr} >$ represents the approximate window founded by the Appr_RLC algorithm, in which lag correlation is maximum. $Error_{lag}$ is defined by Equation 8 as follows.

$$Error_{lag} = \frac{|S_{appr} - S_{exact}| + |E_{appr} - E_{exact}|}{E_{exact} - S_{exact} + 1}. \tag{8}$$

Table 5 shows the estimation error of the captured lag correlations on four datasets. The experiment clearly demonstrates that Appr_RLC detects the correct lag perfectly, most of the time. $Error_{stream} = 100\%$ happens rarely and the largest relative error $Error_{lag}$ is about 3%.

Table 5 Estimation error of lag correlations

Datasets	$Error_{stream}(\%)$	$Error_{lag}(\%)$
kursk	0	0.815
Stock	0	2.302
Tao	100	3.614
Synthetic	0	0.404

6.4 Effectiveness

In this section, we demonstrate the practicality of the proposed RLC query in real applications. The Stock dataset is tested for this task. We adopt the Prog_RLC algorithm to answer our proposed RLC query and use the algorithm which is proposed by Wu et. al [20], [21] to answer the query with a fixed window length. We test the fixed-length query under different fixed-size windows to compare the performance of the comparison algorithms. Table 1 (see Section 1.1), Table 6, Table 7 and Table 8 show the result with the fixed-size windows, 50 days, 100 days, 150 days, and 200 days respectively. Table 2 (see Section 1.1) shows the result of the RLC query with a given minimal window length, 50 days. From these tables, we can find that setting the most appropriate window length with maximum lag correlation for one pairwise stream is difficult. For instance, the most appropriate window length of the maximum lag correlation between Google and Apple is 696 days, not 50, 100, 150, or 200 days given by users. In addition, the most appropriate window length for each pairwise stream may be different. As an example, the most appropriate window length between Google and Apple is 696 days and that between Google and Microsoft is 630 days. It is obvious that setting the same window length for each pairwise stream is not suitable. In addition, the returned stocks by these two queries are different.

Table 6 The top-2 query of fixed length 100 days

Rank	sliding window	correlation	lagged length	query subseries	objective subseries
1	**100 days** [22/09/2012, 31/12/2012]	0.961	33 days	Google [22/09/2012, 29/11/2012]	**Netflix** [25/10/2012, 31/12/2012]
2	**100 days** [22/09/2012, 31/12/2012]	0.954	37 days	Google [22/09/2012, 25/11/2012]	**Facebook** [29/10/2012, 31/12/2012]

Table 7 The top-2 query of fixed length 150 days

Rank	sliding window	correlation	lagged length	query subseries	objective subseries
1	**150 days** [04/08/2012, 31/12/2012]	0.907	19 days	Google [04/08/2012, 22/12/2012]	**Netflix** [23/08/2012, 31/12/2012]
2	**150 days** [04/08/2012, 31/12/2012]	0.886	67 days	Google [04/08/2012, 26/10/2012]	**Facebook** [09/10/2012, 31/12/2012]

Table 8 The top-2 query of fixed length 200 days

Rank	sliding window	correlation	lagged length	query subseries	objective subseries
1	**200 days** [15/06/2012, 31/12/2012]	0.952	90 days	Google [15/06/2012, 01/10/2012]	**Netflix** [13/09/2012, 31/12/2012]
2	**200 days** [15/06/2012, 31/12/2012]	0.951	98 days	Google [15/06/2012, 24/09/2012]	**Facebook** [25/09/2012, 31/12/2012]

7 Related work

The processing and analysis of time series have attracted on increasing amount of interest in the last two decades, such as similarity search and subsequence matching [1], [6], [7], [8], [17], [19], classification and clustering [5], [15], motif discovery [14], [16], top-k queries [11], [22], etc. Of particular interest within this realm are the problems of correlation analysis [3], [4], [10], [12], [13], [14], [18], [20], [21].

Traditional correlation analyses between two time series are aligned in time [12], [13], [14]. However, in many real-world applications, the time lag is a key feature between the cause and the effect or the spread of a common cause [18], [23]. Unfortunately, most of methods for lag correlation analysis in a relatively crude manner: the entire stream or the substream based on the sliding window. In [18] and [23], entire stream is considered, which cannot get rid of the stale data or emphasize the importance of the recent data. Even though the sliding window model is considered in [20], [21], it only works within the window with a fixed length. In many applications, however, it is not possible to fix a window length that is the most appropriate for each pairwise stream. Even worse, in many cases no single optimal choice for the parameter exists, as the most logical window to measure correlation over may be series-dependent. Moreover, the window length heavily depends on queries, time, data, and application domains. Actually, the presence of free window length represents additional burden on the user instead of providing more freedom. Therefore, these methods [18], [23], [20], [21] cannot answer the proposed RLC query. In our experiments, we have adapted the algorithm proposed in [20], [21] to solve the RLC query (see Section 3.1).

To be the best of our knowledge, we are the first work to explore jointly lag correlation and flexible sliding windows such that we propose the RLC query and there is no existing methods on efficient answering the RLC query.

8 Conclusion

In this paper, we propose a novel and general RLC query of ranking lag correlations with flexible sliding windows between a query stream and multiple objective streams. We propose an exact algorithm and an approximate algorithm to answer the RLC query. The exact Prog_RLC Algorithm applies the running sums technique to leverage on the previous computations, which makes Prog_RLC enjoy a linear scalability w.r.t. stream size. The approximate algorithm Appr_RLC uses two observations found by experiments to reduce the number of lag correlations computed and the estimation error.

Our experiments on real and synthetic datasets show that the efficiency and effectiveness of our proposed algorithms. More specifically, the improvement of performance grows as the stream size increases. Prog_RLC (Appro_RLC) outperform the state-of-the-art sulution by 2 (3) orders of magnitude when the stream size is 1100. The estimation error of the Appro_RLC is low when the lag is small. Moreover, we use the real dataset Stock to show the proposed RLC query is more meaning than the query based on the sliding window of a fixed length in real applications.

In the future, we plan to study the RLC query based on other distance measurements, such as dynamic time warping with normalization.

Acknowledgements We thank Yasushi Sakurai and Yuhong Li for providing us the data sets. This work was supported by National Science and Technology Supporting plan (2013BAH62F02 and 2013BAH27F01), the public key plan of Zhejiang Province (2014C23005), China mobile research fund of education ministry (mcm20130671), the cultural relic protection science and technology project of Zhejiang Province, and the University of Macau RC (MYRG2014-00106-FST).

References

1. Athitsos, V., Papapetrou, P., Potamias, M., Kollios, G., Gunopulos, D.: Approximate embedding-based subsequence matching of time series. In: SIGMOD, pp. 365–378 (2008).
2. Box,G. E., Jenkins,G. M., Reinsel,G. C.: Time series analysis: forecasting and contro. (1994).
3. Cai, Y., Tong, H., Fan, W., Ji, P.: Fast mining of a network of coevolving time series, In: SDM, (2015).
4. Cai, Y., Tong, H., Fan, W., Ji, P., He, Q.: Facets: Fast comprehensive mining of coevolving high-order time series. In: SIGKDD, pp. 79–88 (2015).
5. Cao, J., Zhou, Y., Wu, M.: Adaptive Grid-Based k-median Clustering of Streaming Data with Accuracy Guarantee. In: DASFAA, pp. 75–91 (2015).
6. Faloutsos, C., Ranganathan, M., Manolopoulos, Y.: Fast subsequence matching in time-series databases. In: SIGMOD, pp. 419–429 (1994).
7. Gong, X., Xiong, Y., Huang, W., Chen, L., Lu, Q., Hu, Y.: Fast Similarity Search of Multi-Dimensional Time Series via Segment Rotation. In: DASFAA, pp. 108–124 (2015).

8. Kahveci, T., Singh, A. K.: Optimizing similarity search for arbitrary length time series queries. TKDE, 16, pp. 418–433 (2004).
9. Koper, K. D., Wallace, T. C., Taylor, S. R., Hartse, H. E.: Forensic seismology and the sinking of the kursk. Eos, Transactions American Geophysical Union, 82, pp. 37–46 (2001).
10. Kusmierczyk, T., Nørvåg, K.: Mining Correlations on Massive Bursty Time Series Collections. In: DASFAA, pp. 55–71 (2015).
11. Lee, M. L., Hsu, W., Li, L., Tok, W. H.: Consistent top-k queries over time. In: DASFAA, pp. 51–65 (2009).
12. Li, Y., Leong Hou, U., Yiu, M. L., Gong, Z.: Quick-motif: An efficient and scalable framework for exact motif discovery. In: ICDE, (2015).
13. Li, Y., Yiu, M. L., Gong, Z.: Discovering longest-lasting correlation in sequence databases. Proceedings of the VLDB Endowment, 6, pp. 1666–1677 (2013).
14. Mueen, A.: Enumeration of time series motifs of all lengths. In: ICDM, pp. 547–556 (2013).
15. Mueen, A., Keogh, E., Young, N.: Logical-shapelets: an expressive primitive for time series classification. In: SIGKDD, pp. 1154–1162 (2011).
16. Mueen, A., Keogh, E. J., Zhu, Q., Cash, S., Westover, M. B.: Exact discovery of time series motifs. In: SDM, pp. 473–484 (2009).
17. Rakthanmanon, T., Campana, B., Mueen,A., Batista,G., Westover,B., Zhu, Q., Zakaria, J., Keogh, E.: Searching and mining trillions of time series subsequences under dynamic time warping. In: SIGKDD, pp. 262–270 (2012).
18. Sakurai, Y., Papadimitriou, S., Faloutsos, C.: Braid: Stream mining through group lag correlations. In: SIGKDD, pp. 599–610 (2005).
19. Traina, A., Traina Jr, C., Faloutsos, C.: Similarity search without tears: the omni-family of all-purpose access methods, In: ICDE, pp. 623–630 (2001).
20. Wu, D., Ke, Y., Yu, J. X., Philip, S. Y., Chen, L.: Detecting leaders from correlated time series. In: DASFAA, pp. 352–367 (2010).
21. Wu, D., Ke, Y., Yu, J. X., Philip, S. Y., Chen, L.: Leadership discovery when data correlatively evolve. World wide web, 14, pp. 1–25 (2011).
22. Xu, E., Hsu, W., Lee, M. L.,Patel, D.: k-Consistent Influencers in Network Data. In: DASFAA, pp. 452–468 (2015).
23. Zhou, X., Hong, H., Xing, X., Huang, W., Bian, K., Xie, K.: Mining dependencies considering time lag in spatio-temporal traffic data. Web-Age Information Management, Springer, pp. 285–296 (2015).