Efficient Correlation Discovery in Time-Series Streams

Aleksandr Eismont

Karlsruhe Institute of Technology, Kaiserstr. 12, 76131 Karlsruhe, Germany
`ukqln@student.kit.edu`

Abstract. This paper addresses the challenges in detecting the correlation among time-series data streams, which facilitates the research of data stream mining, pattern discovery, and potential anomaly detection. I focus on two types of correlations: global correlation and local correlation with time delay. The traditional method is unsuitable for real-time processing, especially when the number of stream sequences is enormous. This paper proposes effective methods based on different mathematical and approximation techniques to detect the correlation among time-series data streams continuously. Extensive experiments on real and synthetic data sets verify the efficiency, effectiveness, and scalability of the following possible solutions: up to 40k times efficiency improvement.

Keywords: Correlation Detection · Data Streams

1 Introduction

Correlation analysis for all pairs of time series is often the first step of analysis of such data. The notion of correlation allows us to explore the interactional link and dependency relationship between data streams, to discover groups of objects with similar behavior, and, consequently, to detect potential anomalies that a change in correlation may reveal.

In recent years, more and more individuals and companies are collecting data over time. The increasing instrumentation of physical and computing processes has given us unprecedented capabilities to manage massive volumes of data. Time-series exist in almost every field of human activity; they may be information about social and socio-technological processes. For example, In housing and plumbing, there is an analyzing system that can track the amount of fluoride in the drinking water in real time because it may lead to fewer dental cavities some years later. The second example is that almost all online services like YouTube and Apple Music use some recommendation system to find customers with similar shopping patterns and then provide dynamic recommendations based on how a given customer's behavior correlates with others. Data centers have a monitoring system to recognize the servers with correlated performance patterns [6]. And the last one is financial application [18]. The leading stock traders can spot investment opportunities based on the found correlation between currency rates and the stock market. For example, the increase in the exchange rate of the Australian dollar may result in an increase in the New Zealand dollar, which can be delayed by several hours.

The main goal is quantifying these data and studying how they change over time. However, a vast number of devices permanently produce these time-series data. It is not feasible to load these real-time time series data into a stream processing system, which cannot handle the rapidly increasing amount of time-series data. Moreover, users or higher-level applications require immediate responses and cannot afford post-processing. The correlation set discovery problem is NP-hard [4]. So, a concise but powerful model can capture various trend or pattern types of correlation. That is why this discovery is important.

There are different notions of correlation: Pearson correlation coefficient, Spearman correlation coefficient [7], Kendall correlation coefficient, extended Jaccard coefficient [8], mutual information [9], Cosine similarity, Euclidean distance and dynamic time warping [12]. But in this paper, I focus on Pearson correlation coefficients measure. However, some approaches allow us to choose another correlation measure.

The rest of the paper is organised as follows: In Section 2, I describe the problem of discovery of global correlation for time-series and give a view of 3 approaches to detect it. Section 3 overviews local correlation and explains three efficient approaches. Finally, Section 4 gives a brief conclusion.

2 Global Correlation

In this section, I introduce the term *global correlation*, the base *naive* approach, explain *iBRAID* algorithm, which uses an efficient principle of calculating the correlation coefficient, and show *PriCe* for interactive pairwise correlation of subsequences of time-series data.

2.1 Foundations

Correlation may occur in a burst, last for a certain duration, and then disappear. Without loss of generality, the data streams are running synchronously and sampled at a fixed rate, data are numeric, and all time-series are of the same length so that they can be treated as discrete signal sequences. It means that the two data streams are aligned with each other in the time dimension, and there are no missing timestamps. To simplify the problem and the algorithm's explanations, it will discuss only how to detect a correlation between two data streams, but then it is possible to extend the detection to multiple time-series data streams.

To continuously detect the correlation of two streams, a sliding window is applied to keep the most recent synchronous subsequences of the streams. [2]

The numerical time-series data streams are denoted as X and Y, so $x = \{x(0), x(1), \ldots, x(N-1)\}$ stands for the subsequence of X with limited length N, where $x(n) \in \mathbb{R}$ is the value of the $(n+1)$-th data point of the subsequence. $len(x)$ is the length of x, and $st(x)$ is the starting time unit of x. Similar concepts are also applied to Y. Here, the correlation is evaluated by the widely applied Pearson correlation coefficient. For two sequences X and Y, which have the same length N and the same starting time unit, the Pearson correlation coefficient is calculated as, where $\bar{x}$ and $\bar{y}$ stand for the mean value of x and y respectively:

$$r_{xy} = \frac{\sum_{n=0}^{N-1} (x_n - \bar{x})(y_n - \bar{y})}{\sqrt{\sum_{n=0}^{N-1} (x_n - \bar{x})^2}\sqrt{\sum_{n=0}^{N-1} (y_n - \bar{y})^2}} \tag{1}$$

The term $r_{xy} \in [-1, 1]$ indicates the linear correlation of the sequences and can reflect the similar behavior of sequence trend. Theoretically, when the evaluated parts of two sequences are correlated, r_{xy} will return a value close to 1 or -1.

2.2 Naive Solution

The naive solution is the most apparent approach to finding and reporting correlated subsequences in a time-series data set. First, the approach starts with a single pair of time series, picks subsequences of them, which start at the same point in time, and calculates the Pearson correlation coefficient for them. Subsequently, it continues the Pearson correlation coefficient calculations for other subsequences of the same pair of time-series, or moves to calculate the Pearson correlation coefficient for the corresponding subsequences of different pairs of time-series until all possible pairs of subsequences in the time-series data set are examined. Finally, it returns all pairs of subsequences whose Pearson correlation coefficient meets a threshold. As a result, each pair decides correlation.

There is an advantage that it is pretty easy to implement, but also a disadvantage that there is a complex calculation using the central math equation (1) for all pairs of time-series in the data set. In other words, the naive approach illustrates the crux of the problem, which is the high number of calculations required to explore the data sets. This algorithm does not have any prior knowledge about the data and does not use any results as a decision-making input to speed up the production of results. Additionally, each point in each time series is touched as many times as the different pairs of time-series in the data set are.

2.3 iBRAID Solution [2,14]

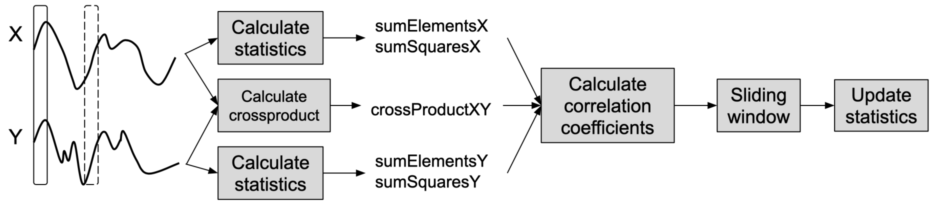

Fig. 1. The iBRAID approach.

The iBRAID algorithm originated from the work BRAID, described in 2005. [15] This solution is based on the hypothesis that integrating caching can achieve interactivity. The iBRAID technique partially solved the problem of the naive approach by efficiently calculating the Pearson correlation coefficient based on basic and computationally cheap five sufficient statistics — the sum of the data points in each window of length m (sum_x), the sum of the squares of the data points of each window of length m (sum_{xx}), the inner cross-product of the data points of the two windows of length m ($sumprod_{xy}$), the variance for time-series (var_x), and the covariance of the two time-series (cov_{xy}) for which the correlation is calculated. The covariance of the two time-series X and Y is

$$cov_{xy} = sumprod_{xy} - \frac{sum_x \times sum_y}{m} \tag{2}$$

and the variance of the window of length m can be calculated according to the following equation

$$var_x = sum_{xx} - \frac{(sum_x)^2}{m} \tag{3}$$

Similarly, the variance for time-series Y will be denoted var_y. Then, the Pearson correlation coefficient can be calculated by applying the following equation

$$r_{xy} = \frac{cov_{xy}}{\sqrt{var_x \times var_y}} \tag{4}$$

First, the approach analyzes the pairs of data streams sequentially, starting from the first point for all data streams. It calculates the sufficient statistics needed to calculate the Pearson correlation coefficient efficiently. Next, it calculates the Pearson correlation coefficient for the first point for all pairs of windows. Once this is done, the windows are slid further by one point, and the sufficient statistics are updated incrementally — the first point is expired/subtracted from them, and the new point is added. The Pearson correlation coefficient is calculated again for all pairs. Then, it analyzes all data streams by a single point, augmenting the sufficient statistics incrementally and recalculating the Pearson correlation coefficient. This is done until the whole pair is analyzed.

Now, I focus on the idea of an incremental approach. These sufficient statistics can be computed either from scratch or incrementally each time a new point of data streams explores a pair of data streams. In this case, the sums stored in memory are incremented by the new values and decremented by those not part of the windows anymore. The same operations are performed for the sums of the squares and the inner cross products using the respective points of data streams. So, iBRAID is a round-robin scanning algorithm that uses the incremental computation of the Pearson correlation coefficient.

Method iBRAID has the following advantages: it is easy to implement, it reduces the computations by half due to the usage of sufficient statistics, and it uses an incremental approach. iBRAID is experimentally shown to perform well for data streams whose data is uniformly distributed and for low correlation thresholds. On the other hand, it might underperform on skewed data sets.

Evaluation The authors have used Yahoo Finance Historical Data Set[1]: It consists of 318 time series and reflects the trading of 56 companies on the New York Stock Exchange for the last 28 years. The length of each time series is about 7100 timestamps.

Naive and iBRAID have the same number of operations due to exhaustive processing of the pairs. The authors noticed that the higher the Pearson correlation coefficient, the more operations the algorithm executes. This is due to fewer windows that are highly correlated according to the Pearson correlation coefficient. This results in higher latency in detecting them by the algorithm. On the other hand, in the case of a low Pearson correlation coefficient, the authors saw that iBRAID terminates the analysis process early, as soon as it reaches the criterion of the number of correlated sliding windows.

2.4 PriCe Solution [3,1]

PriCe is a more informed search algorithm. It uses a priority function to explore the pairs of subsequences while reusing partial Pearson correlation coefficient computations as the iBRAID approach. This solution is based on the hypothesis that interactivity can be achieved by integrating caching and scheduling principles. The idea of PriCe is to explore the most promising pair first, the one with the highest priority function value. The priority function:

$$Pr = PCC * (M/totalExp)/C \tag{5}$$

[1] Yahoo Inc. Yahoo Finance Historical Data (2016). https://finance.yahoo.com/quote/YHOO/history

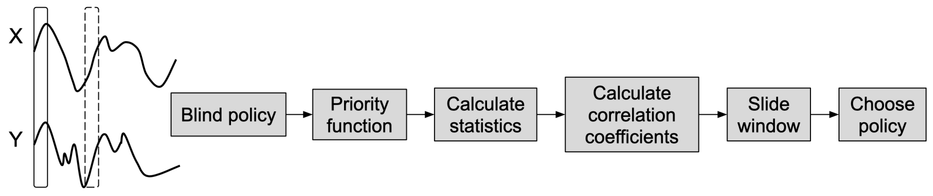

Fig. 2. The PriCe approach.

where *PCC* is the most recently calculated Pearson correlation coefficient for a pair of sliding windows that belong to the same pair of data streams, M is the number of correlated sliding windows found in the corresponding pair of data streams so far, *totalExp* is the total number of analyzed pairs of sliding windows, and C is the cost of analyzing a pair of sliding windows in terms of number of computations. The default values are $PCC = 1$, $M = 0$, $totalExp = 1$, and $C = 1$.

PriCe prioritizes the pair of time series with a history of increased results produced so far and a highly recently calculated Pearson correlation coefficient. This captures the idea of space locality along with temporal locality, where the recently estimated Pearson correlation coefficient captures space locality, and the ratio of correlated sliding windows captures temporal locality to the total number of analyzed pairs of sliding windows. Moreover, the cost in the priority function is the number of operations needed to calculate the sufficient statistics for a pair of sliding windows. For example, if a pair of data streams share one data stream with another pair. Then, the more advanced one (i.e., the one with a higher timestamp in the interval) has already calculated the sums and the sum of the squares for the tuples of that data stream. This leaves the lagging behind pair with lower cost to slide its windows since the more advanced one has already computed some of the sufficient statistics for that shared data stream.

The authors of this approach implemented and evaluated nine policies that initialize and/or tune the priority function. It is based on the concepts of early termination and pruning. When the first pair arrives at the system, the system does not know any correlated pairs of streams.

1. *Blind*: when the analysis starts with no prior knowledge of correlated pairs of streams, Blind policy initializes PriCe's priority function to its default values.
2. *Informed*: the priority function is initialized based on the latest analysis results. In the Informed starting phase, PriCe's priority function is initialized to the same parameter values of the correlated pairs used by the immediately previous subsequences. The rationale behind this policy is to keep analyzing

closely those pairs that already exhibited a high correlation in the earlier subsequences, potentially indicating an insight of interest.

3. *Untouched*: the focus is on the pairs not processed in the previous iterations (i.e., not chosen for analysis due to their low values of Pearson correlation coefficient) at the beginning of PriCe's execution. Specifically, such pairs are jumpstarted by altering their previous number of correlated windows. This increases their priority, preventing starvation and giving them another chance to be analyzed in the new iteration. The rationale behind this policy is to allow such pairs another chance, potentially identifying different behaviors that remained undetected in the previous iteration.

4. *Alternating*: this policy gives the pairs that were not correlated in the previous iteration a chance to be explored through a hybrid round-robin fashion. In the Alternating starting phase, alternately, a pair from those that are not correlated in the previous iteration is picked and explored using PriCe, followed by a pair from those that were correlated. By doing this, the idea is to reduce starvation's effect for those not correlated in the previous iteration.

5. *X% Non-correlated*: this policy tries to achieve fairness of exploration through jumpstarting the lowest X% pairs in priority. The pair with the highest priority among those lowest X% is picked and explored. This continues until all those X% pairs are jumpstarted. Consecutively, PriCe carries out the exploration process naturally.

6. *Decaying History*: this policy regards the significance of the whole historical correlation information of a pair differently from recent iterations information. In the priority function, it alters the parameter M, which reflects the number of correlated sliding windows found for a corresponding pair, such that it becomes weighted. It gives the historical correlation information a weight and then a higher weight to the most recent correlation information. Then, the total of both becomes the new parameter M. This policy aims to consider the most recent information along the exploration process instead of older ones.

7. *Shared Stream*: Its focus is on the group of pairs not correlated in the previous iteration but sharing a data stream that was part of a correlated one in the preceding iteration. The idea in this policy is that a data stream that is correlated with another one might also be correlated with a third different stream. Thus, this policy picks a pair from this group of non-correlated pairs according to PriCe and explores it. Shared Streams starts all those pairs and then carries on using PriCe.

8. *X% Probing*: this policy explores the first few windows for all the pairs in a round-robin fashion. This is done to set the priority function with actual current values instead of artificial hand-crafted ones. After those few windows, PriCe kicks in and continues the exploration process using the priority

function with its parameters filled with actual data through the probing process.

9. *P-Alternating*: this policy mimics the multilevel queue scheduling, whereby the pairs are explored in a round-robin fashion between two groups. Those are the previously correlated pairs and the non-correlated ones. This gives the pairs not correlated in the previous iteration a chance to be persistently processed. It picks a pair from the non-correlated ones and processes it using PriCe; then, it picks a pair from those correlated. It does that until the end of the iteration and carries on in this fashion by selecting the pair from each group of pairs according to the priority function. By doing this, the hope is to alleviate the effect of starvation for those pairs that were not persistently correlated in the previous iteration, for they might exhibit some correlation beyond the starting phase.

Evaluation The authors have used the same data set from Yahoo (as described in Sect.2.3). Naive and PriCe have the same number of operations due to the exhaustive processing of the pairs. The authors noticed that the higher the Pearson correlation coefficient, the more operations the algorithm executes. This is due to fewer windows that are highly correlated according to the Pearson correlation coefficient. On the other hand, in the case of a low Pearson correlation coefficient, the authors saw that terminates the analysis process early.

The authors noticed that PriCe outperformed the naive algorithm and the iBRAID approach in experiments. This is a speed-up of more than 15 times than the iBRAID solution. This is attributed to the pruning and early termination features, which allow the algorithm to analyze other pairs and detect more correlated data streams. The authors saw that PriCe scheduling demonstrated the effectiveness of its priority function in capturing more correlated pairs at an early deadline, especially when the Pearson correlation coefficient is high. That means it elects the pairs to explore more intelligently than the naive and iBRAID approaches.

The authors noticed that PriCe with policy *P-Alternating* is the best policy for detecting correlated live data streams for high values of the Pearson correlation coefficient. On the other hand, the *1% Probing* outperforms the rest when the Pearson correlation coefficient has low values. This is attributed to the fact that with higher Pearson correlation coefficient values, it is highly likely to have fewer correlated windows in a given pair and vice versa. Finally, the explorative policies (i.e., *Blind*) achieved the highest diversity in detecting new correlated pairs than the exploitative ones (i.e., *Informed*, *Untouched*, and *P-Alternating*). This can be explained because the exploitative policies have some informative approach to analyzing the data streams, whether from previous iterations or some other source. Thus, they will keep exploring the pairs already correlated in previous iterations.

3 Local Correlation

In this section, I introduce the term *local correlation*, the base *naive* approach, explain *DFT-Based* algorithm for interactive pairwise correlation of subsequences of time-series data and show *BRAID*, which is using an efficient principle of calculating the correlation coefficient.

3.1 Foundations

Correlation may occur in a burst, last for a particular duration, and then disappear. However, real data sets' signals often correlate with unknown lag. Intuitively, two signals have a local correlation with time delay τ if they look very similar when τ time ticks delay one signal. Without loss of generality, the data streams are sampled at a fixed rate, data are numeric, and all time-series are of the same length so that they can be treated as discrete signal sequences. It means that there are no missing timestamps. To simplify the problem and the algorithm's explanations, only one time-series has been lagged or shifted, and the correlation detection occurs only between two data streams. Still, it is possible to extend the detection to multiple time-series data streams [13].

The numerical time-series data streams are denoted as X and Y, so $x = \{x(0), x(1), \ldots, x(N-1)\}$ stands for the subsequence of X with limited length N, where $x(n) \in \mathbb{R}$ is the value of the $(n+1)$-th data point of the subsequence. $len(x)$ is the length of x, and $st(x)$ is the starting time unit of x. Similar concepts are also applied to Y. Here, the correlation is evaluated by the widely applied Pearson correlation coefficient. Note that to compute the local correlation of X and Y with a time delay τ, one time-series is first shifted by τ time ticks while keeping the other time-series fixed, and then the correlation is computed over their trimmed, common parts of length $(T-\tau)$, where T is the maximum possible time delay. For two sequences X and Y, which have the same length N and time delay τ, the Pearson correlation coefficient is calculated as:

$$\begin{aligned} r_{xy}(\tau) = &\frac{\sum_{n=0}^{N-\tau-1}\left(x_n-\bar{x}\right)\left(y_{n+\tau}-\bar{y}\right)}{\sqrt{\sum_{n=0}^{N-\tau-1}\left(x_n-\bar{x}\right)^2}\sqrt{\sum_{n=\tau}^{N-1}\left(y_n-\bar{y}\right)^2}}, \\ \text{where}\quad \bar{x} = &\frac{1}{N-\tau}\sum_{n=0}^{N-\tau-1} x_n, \quad \bar{y} = \frac{1}{N-\tau}\sum_{n=0}^{N-\tau-1} y_n \end{aligned} \tag{6}$$

Term $r_{xy}(\tau) \in [-1,1]$ indicates the linear correlation of the sequences and can reflect the similar behavior of sequence trend. Theoretically, when the evaluated parts of two sequences are correlated, $r_{xy}(\tau)$ will return a value close to 1 or -1.

3.2 Naive Solution

The naive solution is the most apparent approach to finding and reporting correlated subsequences in a time-series data set. First, the approach starts with a single pair of time-series and picks subsequences from them. Then, for every possible time delay $\tau \in [-T, T]$, it calculates the Pearson correlation coefficient according to math equation (6). When it gets the Pearson correlation coefficients for all possible τ values in the predefined period, the maximal value is chosen as the Pearson correlation coefficient and the corresponding τ. If the Pearson correlation coefficient exceeds the predefined threshold, it reports the correlation, and the related τ is returned as time delay. Note that to compute the local correlation of two time-series with a time delay τ, one time-series is first shifted by τ time ticks while keeping the other time-series fixed. After processing the points in the current window, the current correlated subsequences should be saved if the correlation condition is satisfied. When the following data points come in, the Pearson correlation coefficient is calculated until the correlation is lost. Otherwise, the sliding window reads the following data points and recalculates the Pearson correlation coefficient. As a result, each pair decides correlation.

There is an advantage that it is pretty easy to implement, but also a disadvantage that for each new data point and each possible time delay, it is necessary to recalculate the correlation coefficient using the central math equation (6), which will result in high computation complexity. For the naive algorithm, the space complexity is $O(L + T)$, and the time complexity is $O(L * T)$, where L is the length of the sliding window and T is the maximal time delay.

3.3 DFT-Based Solution [17]

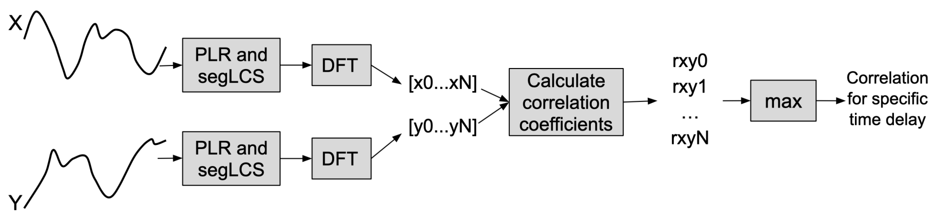

Fig. 3. The DFT-Based Approach.

This solution is based on the advantage of the Discrete Fourier Transform (DFT) properties. It calculates the correlation coefficient with time delay by the inner product of normalized sequences of inverse Fourier Transform of two data stream sequences. The main algorithm order is the same as *naive* but with another math logic calculation.

Now, I focus on the DFT foundation. Let $x = \{x(0), x(1), \ldots, x(n), \ldots, x(N-1)\}$ be a N-point sequence, and the *Discrete Fourier Transform* of x be $X = \{X(1),$

$X(2), \ldots, X(k), \ldots, X(N-1)\}$,:

$$X(k) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn} \quad k \in [0, N-1] \tag{7}$$

The inverse Fourier Transform of X is

$$x(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}kn} \quad n \in [0, N-1] \tag{8}$$

If applying normalization to the sequence x, it is possible to generate the normalized sequence $\hat{x}$, where each sequence point value is

$$\hat{x}(n) = \frac{x(n) - \bar{x}}{\sqrt{\frac{1}{N}\sum(x(n) - \bar{x})^2}} \quad n \in [0, N-1] \tag{9}$$

Similarly, it is possible also to normalize the N-point sequence y. The Pearson correlation coefficient of x and y can be calculated by the inner-product of normalized sequences $\hat{x}$ and $\hat{y}$:

$$r_{xy}(\tau) = \sum_{n=0}^{N-\tau-1} \hat{x}(n)\hat{y}(n+\tau) \tag{10}$$

But the Pearson correlation function $r_{xy}(\tau)$ is directly calculated by inverse DFT on R_{xy} after applying DFT on $\hat{x}$ and $\hat{y}$, because for two N-point sequences x and y, if $r_{xy}(\tau) = \sum x(n)y(n+\tau)$, then the DFT of $r_{xy}(\tau)$ is given by $R_{xy}(k) = X^*(k)Y(k)$, where $X(k)$ and $Y(k)$ are the DFT of x and y respectively, and $X^*(k)$ denotes the complex conjugate of $X(k)$ and $\hat{X}(k) = \frac{X(k)}{\sqrt{\frac{1}{N}\sum(x(n)-\bar{x})^2}}$. [11] Since it is possible to derive the Pearson correlation coefficient with all possible time directly delay τ, the computation complexity can be significantly improved.

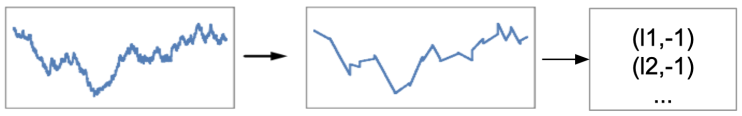

Fig. 4. The Piecewise Linear Representation and Segment trend sequence.

Now, I focus on the first step of this algorithm, utilizing linear representation to enhance the correlation detection, prune the correlation candidates, and help design efficient window sliding. Evaluating every new data point is a huge waste of time since there can be much redundancy. The authors defined piecewise linear representation for a data stream to reduce unnecessary calculations, make the window sliding more efficient, and accelerate the correlation analysis. It is to use line segments to approximate the data stream points continuously. [10] As shown in Figure 4, the data stream can be represented by several line segments, which can indicate the trend of the data stream. It leads to the following advantages: control of the approximation ratio and reduction in subsequence comparisons because the sliding procedure can go ahead segment by segment rather than point by point. Also, piecewise linear representation can prune the unnecessary calculation from an early stage.

Since the correlation is sensitive to the increasing and decreasing trend of data point values, the next step is an evaluation of the similarity of line segments from the view of data-changing trends. After generating line segments, it transfers the line segments into pairs of numbers. Each segment pair describes a line segment's characteristics, consisting of the length of the line segment and the changing trend. If the slope of the line segment is greater than 0, then the changing trend is 1; otherwise, the changing trend is -1. For example in Figure 4, $\{(l_1, -1), (l_2, -1)\}$ means the sequence is approximated by two segments. The first one has l_1 points and is decreasing; the length of the second one is l_2, and it is also decreasing. So, for the two subsequences in the sliding window, it is possible first to compare their similarity by the segment trend sequences. If two sequences are in correlation, their segment trend sequences should be similar since the correlation indicates the linear dependency of the sequences. If the two trend sequences are similar enough, the approach continues on the local correlation evaluation; otherwise, the sliding window moves on to receive new coming points.

There are three advantages: The Discrete Fourier Transform allows rapid correlation calculation with time delay — 30% reduction of running time. Based on the properties of the approximation technique, it is possible to make early pruning to avoid unnecessary calculations by estimating the correlation. Additionally, it reduces running time because it only concerns the XOR operation for similarity of line segments analysis. On the other hand, there are a lot of redundant evaluations of every new data point. For the DFT-based algorithm, the space complexity is $O(L)$, and the time complexity is $O(L * log(L))$, where L is the length of the sliding window.

Evaluation The authors test the algorithm on synthetic and real data streams so that the conclusion can be made based on a wide range of data distribution. For real data, they have chosen the Sunspots[2] and Temperature[3] data sets. Sunspots consist of the data set that describes the daily sunspot numbers from

[2] ftp://ftp.ngdc.noaa.gov/stp/solar_data/sunspot_numbers/
[3] http://lwf.ncdc.noaa.gov/oa/climate/ research/anomalies/index.html

01.01.1945 to 31.10.2009 with 23680 records. It randomly extracted two subsequences from the 23680 records as two data streams, each comprising around 1800 data points. Temperature consists of monthly data records measuring the land and ocean temperatures as data streams. Each of the streams consists of 1560 data points. The correlation between them indicates the link between land temperature and ocean temperature.

The authors found that the results detected by the DFT-based algorithm are mainly laid on the correlation score curve of the naive algorithm at the occurrence of correlation, demonstrating this solution's effectiveness. The DFT-Based algorithm can detect most of the correlation and effectively reduce the false alarm on the subsequences with low visual similarity. When the maximal time delay is slight, the running time of the DFT-Based algorithm is longer than the naive solution because the DFT-Based algorithm continuously calculates the Pearson correlation coefficient with every possible time delay, even if it exceeds T. However, with the growth of T, the DFT-Based algorithm significantly outperforms the naive algorithm, bringing improvement at about 80% off. Finally, the random data set test showed an impressive pruning rate of 95% at least.

3.4 BRAID Solution [15]

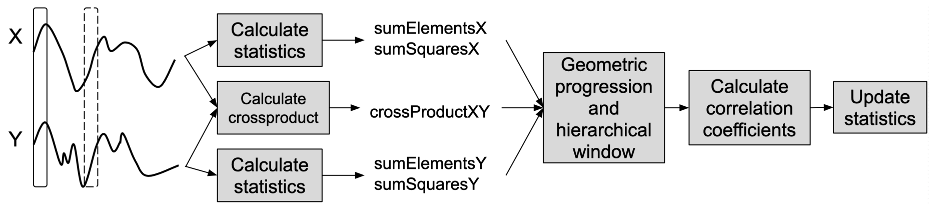

Fig. 5. The BRAID approach.

In section 2.3, I have already described that the BRAID solution is based on the hypothesis that interactivity can be achieved by integrating caching. This algorithm technique partially solved the problem of the naive approach by efficiently and incrementally calculating the Pearson correlation coefficient based on five basic and computationally cheap sufficient statistics. However, linear time is still needed to compute the Pearson correlation function between the two time-series.

The authors introduced an approximation using a geometrical progression to reduce the lag-estimation time. In other words, instead of computing correlation coefficients for every possible period value, it is necessary to track only the geometric progression of the period values with step 2 ($\tau = 0, 1, 2, 4, \ldots, 2^i$).

It means that only $O(log(n))$ numbers are necessary to estimate the Pearson correlation function, instead of $O(n)$ that the naive solution requires. Of course, this approximation introduces error.

This method will give good accuracy for small τ precisely because for small tau's, there are many points to interpolate. It may provide a larger error for large delay τ, but the relative error will probably be small. But the space required grows linearly with the length n because for computation of the Pearson correlation coefficients at any time τ, it is necessary to keep a sliding window of size l ($O(n)$ space required). So, the authors introduced the smoothed version of sequences to estimate the Pearson correlation function. It means $O(log(n))$ space and $O(1)$ time by the geometric probing and sequence smoothing.

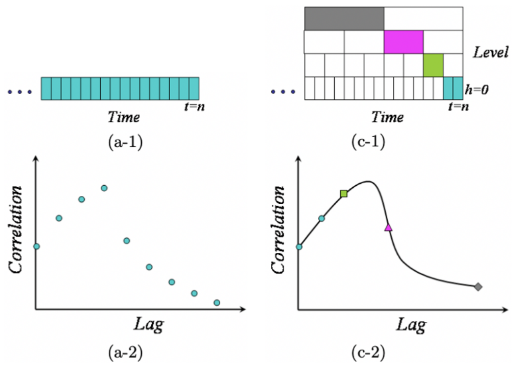

Fig. 6. Hierarchical windowing and cubic spline interpolation.

Instead of operating on the original time sequences, the approach computes their smoothed version by adding non-overlapping windows. The window widths will be powers of 2, although any other number would also be acceptable. Figure 6 (a) illustrates the naive approach: Figure 6 (a-1) denotes that the values for all the time delays are kept. Figure 6 (a-2) shows an illustration of the Pearson correlation function for two subsequences. Figure 6 (c) offers the BRAID solution. The redundant points were eliminated here, favoring the smallest window, which should give more accurate results. Figure 6 (c-2) shows the correlation coefficients obtained from the selected window averages shown in Figure 6 (c-1). The colored boxes represent BRAID's window averages for computing correlation coefficients and capturing time delay. BRAID ignores the white ones and does not calculate their window averages. It computes window averages for every level and correlation coefficients for several time delays incrementally. The missing Pearson correlation coefficients caused by the approximation are estimated by interpolation with a cubic spline. Firstly, there are nine blue points. Next, only 0, 1, 2, 4, and 8 points are chosen. As a result, an off-the-shelf interpolation method was used to obtain a smoother curve to find the local maximum of the Pearson correlation function (Brent's method [5]).

Evaluation The authors test the algorithm on synthetic and real data streams so that the conclusion can be made based on a wide range of data distribution. For real data, they have chosen humidity, height, and temperature readings, Kursk and Sunspots[4]. Humidity, Light, Temperature consist of humidity, illuminance, and temperature readings, from 55 sensors within several buildings. Each sensor gives a reading every 30 seconds. The authors have chosen two sequences for each data set, humidity and light, and used all sequences for temperature. Kursk consists of seismic recordings from multiple sensors, showing the Russian submarine "Kursk" explosion. Each sequence has a single burst. The authors extracted two subsequences of length n=70,000. Sunspots consist of several sunspots per day. The data set has a period of approximately 11 years. The authors have chosen two intervals from the data set, each length n=25,900, and treated them as two different time sequences.

The results show that BRAID ideally approximates the Pearson correlation coefficients of the sinusoidal wave. Also, they indicate that BRAID closely estimates the Pearson correlation coefficients using interpolation and captures the local correlations of the spike. The experiments demonstrate that BRAID detects the correct time delay perfectly most of the time. The most significant relative error was about 1%. Instead of the $O(n)$ that the naive implementation requires, BRAID can dramatically reduce computation time. Theoretically, BRAID requires time $O(1)$ for updating the sufficient statistics; the computation time does not depend on n. In this experiment, the time increases slightly as n grows. This increase is caused by interpolation through a more significant $O(log(n))$ number of points. Specifically, BRAID is up to about 40,000 times faster than the naive implementation.

4 Conclusions

Data streams have received considerable attention in various communities. The naive methods are unsuitable for real-time processing, especially when the number of stream sequences is vast. In this paper, I have explained six different methods for efficiently discovering correlation in time-series. Specifically, I first discussed the algorithms that detect global correlation in time-series. Then, I discussed the local correlation algorithms. Moreover, I described the evaluation for all approaches on several data sets, indicating that they can discover correlations more efficiently and effectively. If you have a task to explore, then you can choose one of them, or you can always combine these methods. For example, you do not want to implement the whole DFT logic, so you take a naive approach

[4] http://csep10.phys.utk.edu/astr162/lect/sun/sscycle.html

and improve it with piecewise linear representation. As a result, you get your algorithm with improved time and space complexity. You can always extend your knowledge about discovering correlation by the papers from the references list and then find others when using the ontological technique like snowballing in systematic literature studies and a replication in software engineering [16].

References

1. Alseghayer, R., Petrov, D., Chrysanthis, P.K.: Strategies for Detection of Correlated Data Streams. Proceedings of the 5th International Workshop on Exploratory Search in Databases and the Web (2018). https://doi.org/10.1145/3214708.3214714
2. Alseghayer, R., Petrov, D., Chrysanthis, P.K., Sharaf, M., Labrinidis, A.: Detection of Highly Correlated Live Data Streams. Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics (2017). https://doi.org/10.1145/3129292.3129298
3. Alseghayer, R., Petrov, D., Chrysanthis, P.K., Sharaf, M., Labrinidis, A.: DCS: A Policy Framework for the Detection of Correlated Data Streams. Real-Time Business Intelligence and Analytics Lecture Notes in Business Information Processing p. 191–210 (2019). https://doi.org/10.1007/978-3-030-24124-7_12
4. Amagata, D., Hara, T.: Correlation Set Discovery on Time-Series Data. Database and Expert Systems Applications pp. 275–290 (2019). https://doi.org/10.1007/978-3-030-27618-8_21
5. Brent, R.P.: Algorithms for Minimization without Derivatives. Englewood Cliffs, Prentice Hall (2002). https://doi.org/10.2307/2005713
6. Gehrke, J., Madden, S.: Query Processing in Sensor Networks. IEEE Pervasive Computing pp. 46–55 (2004). https://doi.org/10.1109/MPRV.2004.1269131
7. Guo, T., Calbimonte, J.P., Zhuang, H., Aberer, K.: SigCO: Mining Significant Correlations via a Distributed Real-time Computation Engine. 2015 IEEE International Conference on Big Data (2015). https://doi.org/10.1109/bigdata.2015.7363819
8. Guo, T., Sathe, S., Aberer, K.: Fast Distributed Correlation Discovery over Streaming Time-Series Data. Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (2015). https://doi.org/10.1145/2806416.2806440
9. Ho, N.T.T., Vo, H., Vu, M., Pedersen, T.B.: AMIC: An Adaptive Information Theoretic Method to Identify Multi-Scale Temporal Correlations in Big Time Series Data. IEEE Transactions on Big Data p. 1–1 (2019). https://doi.org/10.1109/tbdata.2019.2907987
10. Keogh, E., Chu, S., Hart, D., Pazzani, M.: An Online Algorithm for Segmenting Time Series. Proceedings 2001 IEEE International Conference on Data Mining (2001). https://doi.org/10.1109/icdm.2001.989531
11. Lathi, B.P.: Signal Processing and Linear Systems. Berkeley Cambridge Press (1998), `http://galia.fc.uaslp.mx/~mlr/Lathi1.pdf`
12. Li, Z., Guo, J., Li, H., Wu, T., Mao, S., Nie, F.: Speed Up Similarity Search of Time Series under Dynamic Time Warping. IEEE Access p. 163644–163653 (2019). https://doi.org/10.1109/access.2019.2949838
13. Mueen, A., Nath, S., Liu, J.: Fast Approximate Correlation for Massive Time-Series Data. Proceedings of the ACM SIGMOD International Conference on Management of Data pp. 171–182 (2010). https://doi.org/10.1145/1807167.1807188

14. Petrov, D., Alseghayer, R., Sharaf, M., Chrysanthis, P.K., Labrinidis, A.: Interactive Exploration of Correlated Time Series. Proceedings of the ExploreDB17 (2017). https://doi.org/10.1145/3077331.3077335
15. Sakurai, Y., Papadimitriou, S., Faloutsos, C.: BRAID: Stream Mining through Group Lag Correlations. Proceedings of the 2005 ACM SIGMOD international conference on Management of data (2005). https://doi.org/10.1145/1066157.1066226
16. Wohlin, C.: Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (2014). https://doi.org/10.1145/2601248.2601268
17. Xie, Q., Shang, S., Yuan, B., Pang, C., Zhang, X.: Local Correlation Detection with Linearity Enhancement in Streaming Data. Proceedings of the 22nd ACM international conference on Conference on information & knowledge management (2013). https://doi.org/10.1145/2505515.2505746
18. Zhu, Y., Shasha, D.: StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. Proceedings of 28th VLDB Conference pp. 358–369 (2002). https://doi.org/10.1016/B978-155860869-6/50039-1