BRAID: Stream Mining through Group Lag Correlations

Yasushi Sakurai [*]
NTT Cyber Space Laboratories
sakurai.yasushi@lab.ntt.co.jp

Spiros Papadimitriou [†]
Carnegie Mellon University
spapadim@cs.cmu.edu

Christos Faloutsos [†]
Carnegie Mellon University
christos@cs.cmu.edu

ABSTRACT

The goal is to monitor multiple numerical streams, and determine which pairs are correlated with lags, as well as the value of each such lag. Lag correlations (and anti-correlations) are frequent, and very interesting in practice: For example, a decrease in interest rates typically precedes an increase in house sales by a few months; higher amounts of fluoride in the drinking water may lead to fewer dental cavities, some years later. Additional settings include network analysis, sensor monitoring, financial data analysis, and moving object tracking. Such data streams are often correlated (or anti-correlated), but with an unknown lag.

We propose BRAID, a method to detect lag correlations between data streams. BRAID can handle data streams of semi-infinite length, incrementally, quickly, and with small resource consumption. We also provide a theoretical analysis, which, based on Nyquist's sampling theorem, shows that BRAID can estimate lag correlations with little, and often with *no error* at all. Our experiments on real and realistic data show that BRAID detects the correct lag perfectly most of the time (the largest relative error was about 1%); while it is up to 40,000 times faster than the naive implementation.

1. INTRODUCTION

The processing and mining of data streams have attracted on increasing amount of interest recently. Data streams appear in a variety of settings, such as environmental, medical and socioeconomic systems. Typical data-stream applications include network analysis, sensor monitoring, financial data analysis, and moving object tracking. In all these situations, the data sources generate data with no end in sight, making it impossible to store all the historical data. Moreover, we want fast response times, in 'any time' fashion.

There are many, fascinating research problems in such settings, as we survey later, like clustering [20], summarization [34], and forecasting [33, 30]. Here we focus on a less-studied problem, namely on lag correlations. Our goal is to monitor k numerical sequences, $X_1, \ldots, X_k$, and to determine automatically all the pairs of sequences that have a lag correlation. That is, we want to report all the pairs of sequences X_i and X_j, for which sequence X_i follows sequence X_j, with an unknown, arbitrary, lag l. We also want BRAID to report the lag l for each such pair of sequences.

Let's focus on two sequences first ($k = 2$), and then we generalize for more. Without loss of generality, we can assume that the two sequences have the same length n, by zero-padding the shortest. Then, the sub-problem we want to solve is as follows:

Problem 1 (Pair-wise Lag Correlation). *Given two co-evolving sequences of equal length $n = 1, 2, \ldots$, determine, at any point of time, two things: (a) whether there is a lag correlation between them, and (b) if yes, what is the lag length l.*

The full problem we want to solve is as follows:

Problem 2 (Group Lag Correlation). *Given k co-evolving sequences of equal length n, determine, at any point of time, which pairs have a lag correlation, and report all such pairs, as well as the corresponding lags.*

Intuitively, two sequences have a lag correlation of l if they look very similar when one is delayed by l time-ticks. The formal definition of the lag correlation is given in Definition 1 (Section 3.1). Figure 1 (a) illustrates two lag-correlated sequences X and Y, with lag l=1300 time-ticks. Figure 1 (b) shows the correlation coefficient $R(l)$ of X and Y as a function of the lag l (=0,1, ...). The plot shown in Figure 1 (b) is called the *cross-correlation* function (CCF) in time-series literatures. We will give the exact definitions for all these symbols and terms later.

We continue the discussion focusing on two streams, for simplicity. If the sequences X and Y were static, the problem would be trivial: simply compute the CCF (for lag l=0, 1, 2, ...) and report the lag l, for which the CCF is maximized. However, when X and Y continuously increase in

[*]This work was done while this author was visiting Carnegie Mellon University.

[†]This material is based upon work supported by the National Science Foundation under Grants No. IIS-0083148, IIS-0113089, IIS-0209107 IIS-0205224 INT-0318547 SENSOR-0329549 EF-0331657IIS-0326322 CNS-0433540 by the Pennsylvania Infrastructure Technology Alliance (PITA) Grant No. 22-901-0001. Additional funding was provided by Intel and Northrop-Grumman Corporation. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, or other funding parties.

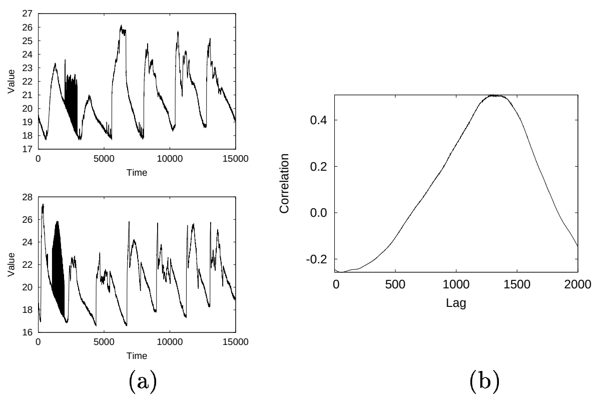


Figure 1: Example of correlated sequences. (a) Two sequences X and Y, with X lagging Y by 1300 time-ticks. (b) Their cross correlation function (CCF), peaking at l=1300.

length, the problem is challenging. We need a method which will monitor X and Y, and whenever the user wants, the method should determine whether there is a lag correlation, and if yes, the value l of the lag. Specifically, we need a method that has the following characteristics:

- '*Any-time*' processing, and fast. The computation time per time tick should be sub-linear (and, ideally, constant) on the length n of the sequences.
- *Nimble*: The memory space requirement should also be sub-linear on the length n.
- *Accurate*: Given that the exact results require too much space and time, we need approximations. Such approximation should introduce small error.

We propose BRAID, a lag capture method for two or more data streams. To our knowledge, BRAID is the first to possess all the above characteristics. The idea is to use careful approximations, exploiting the Nyquist sampling theorem. The net effect is that BRAID has dramatically better performance in terms of speed and memory, while it maintains excellent accuracy. Our experiments on real and realistic streams show that BRAID is up to 40,000 times faster than the straightforward lag computation, while maintaining relative error of 1% or less.

The rest of the paper is organized as follows. Section 2 gives related work on data streams and stream mining. Section 3 describes our method, BRAID. We show how lags of data streams can be captured. Section 4 presents an enhanced algorithm for better accuracy. Section 5 gives our theoretical analysis for BRAID. Section 6 reviews the results of the experiments, which clearly show the effectiveness of BRAID. Section 7 is a brief conclusion.

2. SURVEY

Time-series and sensor networks have been attracting much interest recently. Although there are numerous publications in these research topics, we have not seen any method for that can automatically determine *arbitrary* lag correlations. We provide a survey of the related literatures.

2.1 Indexing, compression, and DSMS

For indexing of time-series data, Agrawal et al. [2] proposed to use the DFT (Discrete Fourier Transform) to extract features for indexing, with R*-trees as the underlying spatial access method. Follow-up work examined several related problems, including subsequence matching [13], the Adaptive Piecewise Constant Approximation (APCA) [24], indexing for Dynamic Time Warping (DTW) [23], similarity query processing for multi-dimensional data streams [26]. Feature extraction, summarization and lossy compression are closely related, with powerful tools like wavelets [18, 19] and random projections [11, 15]. Remotely related is also the work on data stream management systems (DSMSs). Sample systems include *Aurora* [1], *Stream* [29], *Telegraph* [7] and *Gigascope* [9] *OSCAR* [8]. Algorithmic work includes query processing [28], scheduling [3, 6], load shedding [10, 31].

2.2 Pattern discovery - single sequence

Guha et al. [20] have proposed an algorithm that solves the k-median problem for data streams in a single pass. Domingos et al. [12] have presented an algorithm for constructing decision trees. The work in [22, 31] focuses on streams with concept drifting. Zhu et al. [35] study burst detection in streams. AWSOM [30] is one of the first streaming methods for forecasting, in a single time sequence.

2.3 Pattern discovery - multiple sequences

Multiple streams have also attracted significant interest. Ganti et al. [16] propose a generic framework for streaming mining. Zhu et al. [34] focus on monitoring multiple streams in real time. They use the "short window" Fourier Transform to summarize streams, and then compute all the pairwise correlations. However, the method will clearly miss any lag correlation that is longer than the window w of the short-window Fourier Transform. MUSCLES [33] applies multi-variate linear regression on multiple co-evolving time sequences; moreover, it expects the user to specify a constant, upper limit m on the longest possible lag that it will consider (the default is m=5).

In conclusion, none of the above methods satisfies the specifications that we listed in the introduction.

3. PROPOSED METHOD

3.1 Preliminaries

A data stream X is a discrete sequence of numbers $\{ x_1, \ldots, x_t, \ldots, x_n \}$, where x_n is the most recent value. Notice that n increases with every new time-tick.

The definition of the correlation coefficient $R(0)$ between two time sequences X and Y of equal length n and zero lag, is a traditional one, known as Pearson's ρ coefficient:

$$\rho = R(0) = \frac{\sum_t((x_t - \bar{x}) * (y_t - \bar{y}))}{\sigma(x) * \sigma(y)} \qquad (1)$$

where $\bar{x}$, $\bar{y}$ denote the mean of X and Y, respectively. For lag l ($l \geq 0$), we consider only the common part of X and the shifted Y; that is, only $n-l$ time ticks, and the equation

Symbol	Definition
n	Length of sequences
m	Maximum lag (e.g., $m = n/2$)
h	Level of BRAID $(0, \ldots, \lceil \log m \rceil)$
l	Lag $(0, \ldots, m)$
x_t	Value of a sequence X at time $t = 1, \ldots, n$
y_t	Value of a sequence Y at time $t = 1, \ldots, n$
$Ax_h(t)$, $Ay_h(t)$	Disjoint window average of each sequence
$R(l)$	Correlation of X and Y for the lag l
$R_h(l)$	Correlation coefficient computed from window averages X_h and Y_h at the lag l
$\hat{R(l)}$	Approximate correlation for the lag l
γ	Threshold for correlation coefficients
k	Number of sequences

Table 1: Symbols and definitions.

becomes

$$R(l) = \frac{\sum_{t=l+1}^{n}(x_t - \bar{x})(y_{t-l} - \bar{y})}{\sqrt{\sum_{t=l+1}^{n}(x_t - \bar{x})^2}\sqrt{\sum_{t=1}^{n-l}(y_t - \bar{y})^2}} \qquad (2)$$

$$\bar{x} = \frac{1}{n-l}\sum_{t=l+1}^{n} x_t, \quad \bar{y} = \frac{1}{n-l}\sum_{t=1}^{n-l} y_t$$

where $R(l)$ denotes the correlation coefficient, when X is delayed by l. Notice that we can easily handle the symmetric case; that is, when Y is the one delayed.

We are interested in high absolute values of $R(l)$. Thus, we call *score* at lag l the absolute value of $R(l)$, that is

$$score(l) = |R(l)| \qquad (3)$$

DEFINITION 1 (LAG CORRELATION). *Two sequences X and Y have a* lag correlation *of l, and specifically, that X lags Y by lag l, if*

1. *the score (=absolute value of the correlation coefficient) between x_t and y_{t-l} is above a threshold γ, say $\gamma = 0.4$, and is actually a local maximum.*
2. *and this is the earliest such maximum, if more maxima exist.*

The first requirement is straightforward. For the second requirement, the reasoning is more subtle: if the two sequences are periodic with the same period T, (which is often the case with real sequences – say, daily, or yearly periodicities), there will be many local maxima $(l, l+T, l+2*T, \ldots)$. Clearly, the earliest of these lags is the most important.

Another subtle point from time series analysis is that the estimate $R(l)$ for large values of the lag $l \approx n$ are undesirable, because the original and shifted time sequences will have too few overlapping time-ticks. Thus, following recommendations from the time series analysis [4], we restrict the maximum lag m to be $n/2$. Notice that even with this restriction, the maximum considered lag m increases with time, since we made it a function of the sequences' length. This changing nature of m creates subtle, but hard problems for all the earlier methods that we mentioned in the survey.

3.2 Main ideas behind BRAID

As we mentioned earlier, if we had infinite space and time, the problem would be trivial:

Naive Solution. At time n, we would access all the values of X and Y, we would compute the CCF, that is, the $R(l)$ for all values of the lag l (=0,1, ...), and we would choose the earliest maximum score above γ, or report that there is no lag correlation.

Our solution is based on the three major steps, each described next.

OBSERVATION 1. *The correlation coefficient $R()$ is an algebraic measure[21]. That is, it can be computed incrementally.*

More specifically, all we need is some *sufficient statistics* (namely, sums, sum of squares, sum products); then, R can be easily computed. Let $Sx(1, n)$ be the sum of X of length n (i.e., $Sx(1, n) = \sum_{t=1}^{n} x_t$), and $Sxx(1, n)$ be the sum of the squares of X (i.e., $Sxx(1, n) = \sum_{t=1}^{n} x_t^2$). $Sxy(l)$ means the inner-product for X and the shifted Y:

$$Sxy(l) = \sum_{t=l+1}^{n} x_t y_{t-l} \qquad (4)$$

We shall refer to all these values collectively as *sufficient statistics* . Given our sufficient statistics, the correlation coefficient $R(l)$ is obtained by:

$$R(l) = \frac{C(l)}{\sqrt{Vx(l+1, n) \cdot Vy(1, n-l)}} \qquad (5)$$

where $C(l)$ is the covariance of X and Y:

$$C(l) = Sxy(l) - \frac{Sx(l+1, n) \cdot Sy(1, n-l)}{n-l}$$

and $Vx(l+1, n)$ means the variance of the subsequence of X, starting from $t = l+1$:

$$Vx(l+1, n) = Sxx(l+1, n) - \frac{(Sx(l+1, n))^2}{n-l} \qquad (6)$$

The variance $Vy(1, n-l)$ of Y is computed similarly. In conclusion, for the given value of lag l, we only need to keep track of five numbers, the sufficient statistics, because they are enough to help us estimate the correlation $R(l)$, at any point of time.

Although important, this observation is not enough to have a streaming method: It still needs linear time to compute the cross-correlation function (CCF) between the two given sequences X and Y. To reduce the lag-estimation time, we introduce an approximation:

OBSERVATION 2 (GEOMETRIC PROBING). *We compute (probe) the CCF at values of the lag l that form a geometric progression. Thus, we need only $O(\log n)$ numbers to estimate the CCF.*

Specifically, instead of computing $R(l)$ for every possible value of the lag l, we propose to keep track of only a geometric progression of the lag values: l= 0,1, 2, 4, ..., 2^i, ... The justification is that it achieves a dramatic reduction in computation time, since we need only $O(\log n)$ numbers to keep track of, instead of $O(n)$ that the "Naive Solution" requires. As we show later (see Theorem 4), this approximation introduces little error, and occasionally zero error. The intuition is that our method will give good accuracy for small l, exactly because for small l's we have many points to interpolate; it may give a larger error for large lag l, but the relative error will probably be small.

There is only one remaining problem: The space required grows linearly with the length n. The reason is subtle: in

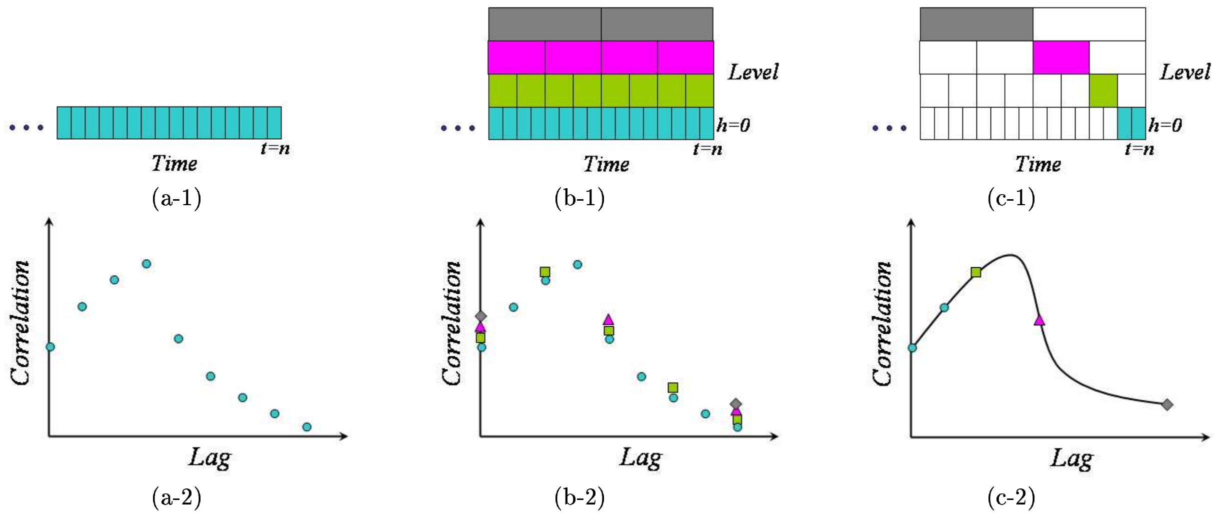


Figure 2: Illustration of BRAID. (a-1) (a-2) The "Naive Solution" computes all values of the lag $l = 0, \ldots,$ without smoothing. (b-1) (b-2) Using all versions of smoothed sequences allows a partial redundancy. (c-1) (c-2) BRAID keeps track of only a geometric progression of lag values.

order to have the ability to compute the correlation coefficient $R(l)$ at *any* time t ($t = n,\ n+1,\ \ldots$) we need to keep a sliding window of size l. Since l grows geometrically up to $m = n/2$, eventually we need $O(n)$ space.

Observation 3. *We use the smoothed version of sequences to estimate the CCF. Thus, we achieve $O(\log n)$ space and $O(1)$ time by geometric probing and sequence smoothing.*

We propose to solve the above problem with our third and final idea, an approximation: Instead of operating on the original time sequences, we also compute their smoothed version, by computing the means of non-overlapping windows. The window widths will be powers of g=2, although any other number would also be acceptable. Let X be the original time sequence, and Ax_h be its smoothed version with windows (*window average*) of length 2^h. That is, Ax_0 is the original sequence; Ax_1 consists of $n/2$ ticks, with the pair means; Ax_2 has $n/4$ ticks, with the quadruplet means, and so on. At time n, we need $O(\log n)$ levels; for each level, we compute the sufficient statistics. Ax_h and its sufficient statistics need to be computed every 2^h time ticks. On average, we require $O(1)$ time to update the sufficient statistics since $\sum_{h=0}^{\log m} 1/2^h \approx 2$.

Not only 'smoothing' saves space, but we can further prove that it results into a small or even zero error, under certain assumptions. Formally we approximate: $R(l) \approx \hat{R}_1(l/2)$, and in general

$$R(l) \approx \hat{R}_h(l/2^h) \qquad (7)$$

where $\hat{R}_h()$ is the correlation coefficient of the h-level smoothed sequences, and $\hat{R}_0() \equiv R()$. Lemma 1 in Section 5.1.1 gives a theoretical justification.

3.3 BRAID

As we mentioned, the correlation $R(l)$ of Equation (5) can be calculated incrementally. Moreover, we operate on increasingly coarser (smoother) representations of the sequences, by using window averages, with increasing window size. Figure 2 (b-1) shows an example of the window average. We compute the average of data points falling within a window, and organize all the windows hierarchically. The window size increases as the level of the hierarchy becomes higher.

BRAID is a lag capture method for data streams. Figure 2 (c-1) illustrates how BRAID uses the window averages. The colored (shaded, in B/W) boxes in this figure represent the window averages BRAID uses for computing correlation coefficients and capturing lags. BRAID ignores the white ones and does not calculate their window averages. BRAID maintains one value for each level in the figure, with one exception: the bottom level has double the number of windows.

BRAID approximates the correlation coefficients $R(l)$ ($l = 1,\ 2,\ 4,\ 8,\ \ldots$) of the two sequences from their window averages. It incrementally computes window averages for every level and correlation coefficients for several lags, as shown in Figure 2 (c-2) (circles, squares, triangles etc). To obtain a smoother curve to find the local maximum of the CCF, we can use an off-the-shelf interpolation method. We chose cubic splines [14], but the choice of interpolation method is orthogonal to BRAID.

We give some examples to explain how BRAID approximates the correlation coefficients and captures the lag for which the sequences are correlated. Figure 2 (a) illustrates the "naive" method: Figure 2 (a-1) denotes that we keep the values for all the time ticks (shaded). Figure 2 (a-2) shows an illustration of the CCF (cross correlation function) for two fictitious sequences. We need to capture the lag with a high correlation coefficient.

Figure 2 (b) illustrates a non-recommended method, which is only useful for explanations. Figure 2 (b-2) illustrates the correlation coefficients calculated from *all* window averages, that is, all the shaded/colored rectangles in Figure 2 (b-1). The window averages allow a partial redundancy.

Figure 2 (c) shows the proposed BRAID. Here, we eliminate the redundant points, favoring the smallest window, which should give more accurate results. Figure 2 (c-2) shows the correlation coefficients obtained from the selected window averages shown in Figure 2 (c-1). In contrast with the smaller lags, the larger lags are sparse because they

```
Algorithm BRAID
  input: new values at $t$ for $k$ sequences $X_1, \ldots, X_k$
  output: lag correlations for sequence pairs if any
  for each sequence $X$ do
    // Update the sums and sum squares of $X$
    SumKeeping($X$);
  for each pair of sequence $X$ and $Y$ do
    // Update the products of $X$ and $Y$
    ProductKeeping($X$, $Y$);
    if output is required at $t$ then
      // Spot a lag correlation if any
      LagDetecting($X$, $Y$);
      report a lag correlation if any;
    endif
  endfor

Algorithm LagDetecting($X$, $Y$)
  // Compute the correlation coefficient of $X$ and $Y$
  compute $R_0(0,t)$;
  delete $R_0(0,t-1)$;
  // Compute the correlation coefficient for the level $h$
  for $h = 0$ to $\lceil \log m \rceil$ do
    if $t$ mod $2^h = 0$ then
      compute $R_h(1,t_h)$;
      delete $R_h(1,t_h-1)$;
    else
      break;
    endif
  endfor
  // Fit splines on the graph $R_h(l/2^h)$ vs $l$
  //    The series of $l$ is a geometric progression
  //    $l = \{0,1,2,4, \ldots, 2^h, \ldots, 2^{\lceil \log m \rceil}\}$
  $\hat{R}_h := Spline(R_0(0), R_0(1/2^0), R_1(2/2^1), \ldots, R_h(l/2^h), \ldots)$;
  extract the optimal lag from $\hat{R}_h$ (if any);
```

Figure 3: Algorithm for detecting lag correlations.

```
Algorithm SumKeeping($X$)
  // Compute sum and sum square
  compute $Sx_0(1,t)$, $Sx_0(2,t)$, $Sxx_0(1,t)$, $Sxx_0(2,t)$;
  delete $Sx_0(1,t-2)$, $Sx_0(2,t-1)$, $Sxx_0(1,t-2)$, $Sxx_0(2,t-1)$;
  for $h = 1$ to $\lfloor \log n \rfloor$ do
    if $t$ mod $2^h = 0$ then
      // Compute window average for $h$
      compute $Ax_h(t_h)$;
      delete $Ax_{h-1}(t_{h-1}-3)$, $Ax_{h-1}(t_{h-1}-2)$;
      // Compute sums and sum squares for $h$
      compute $Sx_h(1,t_h)$, $Sx_h(2,t_h)$,
              $Sxx_h(1,t_h)$, $Sxx_h(2,t_h)$;
      delete $Sx_h(1,t_h-2)$, $Sx_h(2,t_h-1)$,
             $Sxx_h(1,t_h-2)$, $Sxx_h(2,t_h-1)$;
    else
      break;
    endif
  endfor

Algorithm ProductKeeping($X$, $Y$)
  // Compute inner-products
  compute $Sxy_0(0,t)$, $Sxy_0(1,t)$;
  delete $Sxy_0(0,t-1)$, $Sxy_0(1,t-1)$;
  for $h = 1$ to $\lfloor \log n \rfloor$ do
    if $t$ mod $2^h = 0$ then
      // Compute inner products for $h$
      compute $Sxy_h(1,t_h)$;
      delete $Sxy_h(1,t_h-1)$;
    else
      break;
    endif
  endfor
```

Figure 4: Algorithm for updating window averages and their sufficient statistics.

are computed from the window averages for higher levels. Thus, we use a cubic spline to interpolate the missing correlation coefficients between the approximated coefficients. It effectively estimates that the correlation coefficients vary between these lags. Finally, we can see the lag correlation (if any) from the local maximum of the cubic spline curve (solid line in Figure 2 (c-2)).

3.3.1 Algorithm

Before introducing our algorithm, we give some definitions. Let $Ax_h(t)$ be the window average at time tick t for level h. $Ax_h(t)$ is computed as:

$$\begin{aligned} Ax_h(t) &= \frac{Ax_{h-1}(2t-1) + Ax_{h-1}(2t)}{2} \\ & \quad t = 1, \ldots, n/2^h \end{aligned} \tag{8}$$

with $Ax_0(t) \equiv x_t$. We are now able to define the "sufficient statistics" for the window averages. The sum of Ax_h and the sum of the squares on Ax_h at t are obtained as:

$$\begin{aligned} Sx_h(1,t) &= Sx_h(1,t-1) + Ax_h(t) \\ Sxx_h(1,t) &= Sxx_h(1,t-1) + (Ax_h(t))^2 \end{aligned} \tag{9}$$

The inner-product of Ax_h and Ay_h for lag l at t is incrementally updated every 2^h time ticks.

$$Sxy_h(l,t) = Sxy_h(l,t-1) + Ax_h(t) \cdot Ay_h(t-l) \tag{10}$$

By using the obtained sufficient statistics, we can derive the correlation coefficient $R_h(l,t)$ as:

$$R_h(l,t) = \frac{C_h(l,t)}{\sqrt{Vx_h(l+1,t) \cdot Vy_h(1,t-l)}} \tag{11}$$

where

$$\begin{aligned} C_h(l,t) &= Sxy_h(l,t) - \frac{Sx_h(l+1,t) \cdot Sy_h(1,t-l)}{t-l} \\ Vx_h(l+1,t) &= Sxx_h(l+1,t) - \frac{(Sx_h(l+1,t))^2}{t-l} \end{aligned}$$

The basic algorithm for incremental computations is shown in Figures 3 and 4. In these figures, t_h denotes the time tick for level h (i.e., $t_h = t/2^h$).

For each incoming data point, we first incrementally update the window averages and their sufficient statistics for every level (See Figure 4). We then approximate the correlation coefficient of the two sequences at l, $R(l,t)$, from their window averages (See Figure 3). By using multiple window sizes, we can compute the correlation coefficients for only a geometric progression of lags l:

$$l = \{0, 1, 2, 4, 8, \ldots, 2^h, \ldots, 2^{\lceil \log m \rceil}\}$$

The missing correlation coefficients caused by the approximation are estimated by interpolation with a cubic spline. After interpolation, we can use any known method to find the local maxima - we chose Brent's method [5]. If a high enough local maximum exists, we report the corresponding lag.

4. ENHANCED BRAID

In Figures 3 and 4, the number of window averages that BRAID computes for each level, is 1. In fact, as described, BRAID is extremely nimble: for example, for two sequences of size $\approx 2^{20}$ (1 million long each), it requires about 5 *

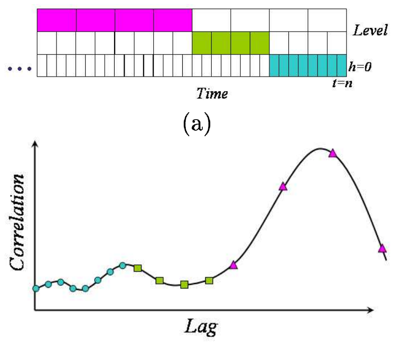


Figure 5: Illustration of the enhanced BRAID ($b = 4$). (a) The enhanced algorithm uses the colored windows. (b) It computes four coefficients for each level.

$\log 2^{20}$=5*20 = 100 float numbers, which is about 800 bytes - way less than 1KB!

The question is what can we do in the highly likely case that larger memory is available. The proposed solution is to enhance our probing scheme, so that we can probe in many more places (= lags), while still using $O(\log n)$ space.

To make our probing denser, one idea would be to have windows of powers of g, where $g \neq 2$. A simpler idea, that we propose and implemented, is the use a mix of arithmetic plus geometric probing. So far, BRAID uses only one window at each smoothing level $(0, 1, \ldots, h)$. We propose to use $b > 1$ such windows, say, $b = 4$ instead (See Figure 5).

The algorithm shown in Section 3.3.1 keeps one number for each level, that is $b = 1$, with one exception, namely that the bottom level has $2b$ coefficients. While computing the correlation coefficients at l, we end up sampling the CCF in a mixture of geometric and arithmetic progressions:

$$\begin{aligned} l &= \{0, 1, 2, \ldots, 2b-1;\ 2b,\ 2(b+1), \ldots, 2^h i, \ldots\} \\ & \quad (1 \le i \le b; 0 \le h \le \lceil \log(m/b) \rceil) \end{aligned}$$

Figure 5 shows an example of the enhanced BRAID of $b = 4$. We compute the correlation coefficients at l:

$$l = \{0, 1, \ldots, 7; 8, 10, 12, 14; 16, 20, 24, 28; 32, 40, \ldots\}$$

The enhanced algorithm for incremental computation is shown in Figures 6 and 7. If $b = 1$, these figures are absolutely equal to Figures 3 and 4 in Section 3.3.1.

5. THEORETICAL ANALYSIS

In this section we give a theoretical analysis to show the accuracy and complexity of BRAID. Again, we focus on two sequences X and Y. To simplify the discussion, and without loss of generality, we assume that the given sequences X and Y are normalized to zero mean and unit standard deviation (i.e., $\bar{x} = \bar{y} = 0$ and $\sigma(x) = \sigma(y) = 1$).

5.1 Accuracy

Our upcoming experiments show that we can closely estimate lag correlations despite our two approximations, the geometric probing and the smoothing. It turns out that this can be explained. In fact, if X and Y are "smooth" enough, then the errors introduced by smoothing and by probing are small. In fact, the probing error can even be zero, if

```
Algorithm BRAID
  input: new values at $t$ for $k$ sequences $X_1, \ldots, X_k$
  output: lag correlations for sequence pairs if any
  for each sequence $X$ do
    // Update the sums and sum squares of $X$
    SumKeeping($X$);
  for each pair of sequence $X$ and $Y$ do
    // Update the products of $X$ and $Y$
    ProductKeeping($X$, $Y$);
    if output is required at $t$ then
      // Spot a lag correlation if any
      LagDetecting($X$, $Y$);
      report a lag correlation if any;
    endif
  endfor

Algorithm LagDetecting($X$, $Y$)
  // Compute the correlation coefficient of $X$ and $Y$
  for $i = 0$ to $b-1$ do
    compute $R_0(i,t)$;
    delete $R_0(i,t-1)$;
  endfor
  // Compute the correlation coefficient for the level $h$
  for $h = 0$ to $\lceil \log(m/b) \rceil$ do
    if $t$ mod $2^h = 0$ then
      for $i = b$ to $2b-1$ do
        compute $R_h(i,t_h)$;
        delete $R_h(i,t_h-1)$;
      endfor
    else
      break;
    endif
  endfor
  // Fit splines on the graph $R_h(l/2^h)$ vs $l$. The series of $l$ is
  // a mixture of geometric and arithmetic progressions.
  $\hat{R}_h$ := $Spline(R_0(0), R_0(1/2^0), \ldots, R_h(l/2^h), \ldots)$;
  extract the optimal lag from $\hat{R}_h$ (if any);
```

Figure 6: Enhanced algorithm for detecting lag correlations.

the input sequences each has a Nyquist frequency. Next we elaborate on each type of error.

5.1.1 *Smoothing*

Informally, the intuition is the following:

OBSERVATION 4. *For sequences with low frequencies, smoothing introduces only small error.*

Let x_t be the values of the original sequence X at time t $(t = 1, \ldots, n)$, and $\hat{x}_{t_h}$ be the smooth version of x_t $(t_h = \lfloor t/2^h \rfloor)$. Given the Haar wavelet coefficients of X, w_i $(i = 1, \ldots, n)$, then the error in the approximation of Equation (11) depends on the energy in the high frequencies.

$$\sum_{t=1}^{n} (x_t - \hat{x}_{t_h})^2 = \sum_{i > n/2^h} w_i^2 \qquad (12)$$

Since very few of the wavelet coefficients of real data sets are often significant and a majority are small [18], Equation (12) shows that the error is limited to a very small value for most of practical data.

THEOREM 1. *Let R be the correlation coefficient between sequences X and Y, then*

$$R = 2^h \sum_{t_h} \hat{x}_{t_h} \hat{y}_{t_h} + \Delta C \qquad (13)$$

```
Algorithm SumKeeping($X$)
  // Compute sum and sum square
  compute $Sx_0(1,t)$, $Sxx_0(1,t)$;
  delete $Sx_0(1,t-2b)$, $Sxx_0(1,t-2b)$;
  for $i = 2$ to $2b$ do
    compute $Sx_0(i,t)$, $Sxx_0(i,t)$;
    delete $Sx_0(i,t-1)$, $Sxx_0(i,t-1)$;
  endfor
  for $h = 1$ to $\lfloor \log n \rfloor$ do
    if $t$ mod $2^h = 0$ then
      // Compute window average for $h$
      compute $Ax_h(t_h)$;
      delete $Ax_{h-1}(t_{h-1}-2b-1)$, $Ax_{h-1}(t_{h-1}-2b)$;
      // Compute sums and sum squares for $h$
      compute $Sx_h(1,t_h)$, $Sxx_h(1,t_h)$;
      delete $Sx_h(1,t_h-2b)$, $Sxx_h(1,t_h-2b)$;
      for $i = b+1$ to $2b$ do
        compute $Sx_h(i,t_h)$, $Sxx_h(i,t_h)$;
        delete $Sx_h(i,t_h-1)$, $Sxx_h(i,t_h-1)$;
      endfor
    else
      break;
    endif
  endfor

Algorithm ProductKeeping($X$, $Y$)
  // Compute inner-products
  for $i = 0$ to $2b-1$ do
    compute $Sxy_0(i,t)$;
    delete $Sxy_0(i,t-1)$;
  endfor
  for $h = 1$ to $\lfloor \log n \rfloor$ do
    if $t$ mod $2^h = 0$ then
      // Compute inner products for $h$
      for $i = b$ to $2b-1$ do
        compute $Sxy_h(i,t_h)$;
        delete $Sxy_h(i,t_h-1)$;
      endfor
    else
      break;
    endif
  endfor
```

Figure 7: Enhanced algorithm for updating sufficient statistics.

where

$$\Delta C = \sum_t (\Delta x_t \hat{y}_{t_h} + \Delta y_t \hat{x}_{t_h} + \Delta x_t \Delta y_t)$$
$$\Delta x_t = x_t - \hat{x}_{t_h}$$

PROOF. Since $x_t = \hat{x}_{t_h} + \Delta x_t$, the covariance is

$$C = \sum_t (\hat{x}_{t_h}\hat{y}_{t_h} + \Delta x_t \hat{y}_{t_h} + \Delta y_t \hat{x}_{t_h} + \Delta x_t \Delta y_t)$$

If X and Y are normalized, $\sigma(x) = \sigma(y) = 1$. Thus, we have

$$R = \frac{\sum_t x_t y_t}{\sigma(x) * \sigma(y)} = 2^h \sum_{t_h} \hat{x}_{t_h}\hat{y}_{t_h} + \Delta C$$

□

LEMMA 1. *Let R_h be the correlation coefficient between $\hat{x}$ and $\hat{y}$, and let $\hat{\sigma}^2(x) = \sum_{t_h} \hat{x}_{t_h}^2$, then*

$$R \approx R_h = \frac{\sum_{t_h} \hat{x}_{t_h}\hat{y}_{t_h}}{\hat{\sigma}(x) * \hat{\sigma}(y)} \quad (14)$$

PROOF. By Equation (12), we can see $x_t >> \Delta x_t$, then $R >> \Delta C$, and $\sigma^2(x) \approx \hat{\sigma}^2(x)$. Therefore, we have $R \approx R_h$. □

5.1.2 Lag probing

The second source of error is the fact that BRAID probes a small subset of the actual values of the lag l. It turns out that, for smooth enough sequences, this introduces little, or even *no* error.

OBSERVATION 5. *Let f_X and f_Y be the Nyquist frequencies of X and Y, and let l be the lag length for X and Y. Then BRAID will find the lag correlations perfectly, if $0 \le l < l_R$ where*

$$l_R = \frac{2b}{f_R} \qquad f_R = \min(f_X, f_Y)$$

BRAID uses coarse sequences to find lag correlations efficiently. To avoid missing lag correlations, we also need a reliable criterion upon which to check the accuracy. Before showing the criterion, we need to describe some theorems.

THEOREM 2 (SAMPLING THEOREM). *If a continuous function contains no frequencies higher than f_{high}, it is completely determined by its value at a series of points less than $1/2f_{high}$ apart.*

PROOF. See [27]. □

In the theorem, the minimum sampling frequency, $f_{Nq} = 2f_{high}$, is called the *Nyquist* frequency.

We next show the relationship between CCF and the Fourier transform, known as the cross-correlation theorem.

THEOREM 3 (CROSS-CORRELATION THEOREM). *Let $\mathcal{F}$ be the Fourier transform, and $R(l)$ be the CCF of X and Y, then we have*

$$\mathcal{F}_R = \mathcal{F}_X \mathcal{F}_Y^* \quad (15)$$

where '' denotes the complex conjugate.*

PROOF. See [27]. □

Based on Theorems 2 and 3, we can derive the following theorem.

THEOREM 4. *Let f_X and f_Y be the Nyquist frequencies of X and Y, and let $f_R = \min(f_X, f_Y)$. Then $R(l)$ is perfectly reconstructed from its samples taken uniformly if the sampling (i.e., 'probing') frequency is at least f_R.*

PROOF. For X and Y, we have

$$\mathcal{F}_X(f) = 0 \quad (f > f_X/2)$$
$$\mathcal{F}_Y(f) = 0 \quad (f > f_Y/2)$$

then

$$\mathcal{F}_R(f) = 0 \quad (f > f_R/2,\ f_R = \min(f_X, f_Y))$$

Therefore, f_R is the Nyquist frequency of $R(l)$, i.e., $R(l)$ can be reconstructed perfectly if the sampling frequency is f_R. □

Since probing by BRAID is not uniform, we need a new lemma. As always, we assume that the stream rate is one measurement per time unit.

LEMMA 2. *Given b coefficients for each level, BRAID will have no error in the estimation of a lag correlation of l if*

$$2^h \le \frac{1}{f_R} \quad (16)$$

where h is the level that covers the lag correlation of l:

$$h = \begin{cases} \lfloor \log_2 \frac{l}{b} \rfloor & (l \ge b) \\ 0 & (l < b) \end{cases} \quad (17)$$

PROOF. Theorem 4 shows that no information is lost if a signal is sampled at f_R. Therefore, $R(l)$ can be determined if the interval, 2^h, is less than or equal to $1/f_R$. □

LEMMA 3. *BRAID spots lag correlations of* $0 \leq l < l_R$*:*

$$l_R = \frac{2b}{f_R} \tag{18}$$

PROOF. Lag correlations can be detected if the following condition is satisfied:

$$\begin{array}{lll} & l < 2b & \& \quad 1 \leq 1/f_R, \\ or & l < 4b & \& \quad 2 \leq 1/f_R, \\ & \dots & \\ or & l < 2^{h+1}b & \& \quad 2^h \leq 1/f_R. \end{array}$$

$R(l)$ can be determined if $0 \leq l < 2^{h+1}b \leq 2b/f_R$. □

5.2 Complexity

In this section, we discuss the complexity of BRAID and show that BRAID can efficiently estimate lag correlations.

Let m be the maximum lag we want to capture. Suppose that m is proportional to the sequence length n.

LEMMA 4. *For every pair of sequences, the "Naive Solution" requires space* $O(n)$ *and time* $O(n)$*.*

PROOF. When x_n and y_n arrive, it computes the correlation coefficient $R(l, n)$ for each lag l (l = 0, 1, ... , m). To compute $R(l, n)$, it keeps y_t (t=$n - m$, ... ,n) and x_n. Thus, it needs to store $2m + 3$ values and update the $m + 1$ values. Since m is proportional to n, it requires space $O(n)$ and time $O(n)$. □

LEMMA 5. *For each pair of sequences, the proposed BRAID requires* $O(\log n)$ *space.*

PROOF. BRAID needs to maintain the ($\log m$+2) correlation coefficients since the bottom level has double the number of coefficients. To compute the correlation coefficients, BRAID keeps the $\log m + 1$ values for X and ($\log m + 2$) values for Y. This requires space $O(\log n)$. While BRAID needs to maintain sufficient statistics, they are the same complexity. Therefore, the space complexity of BRAID is $O(\log n)$. □

LEMMA 6. *The proposed BRAID requires* $O(1)$ *amortized time per time-tick for updating sufficient statistics.*

PROOF. The window average needs to be computed every 2^h time ticks (h=1, ... ,$\log m$). On average, BRAID computes one value for each incoming data point, because $\sum_{h=1}^{\log m} 1/2^h \approx 1$. Similarly, the sufficient statistics can be updated in $O(1)$ time. Therefore, the amortized time complexity of BRAID is $O(1)$. □

When output is required, BRAID employs interpolation to closely estimate lag correlations.

LEMMA 7. *The proposed BRAID requires* $O(\log n)$ *time for interpolating.*

PROOF. The cubic spline for the geometric probing requires $O(\log n)$ time. See [14]. □

BRAID can retain more than one correlation coefficient for each level, that is $b \geq 1$. Since b is a small constant (i.e., $b \ll n$), the space required is still $O(\log n)$, and the time complexity is $O(1)$.

6. EXPERIMENTS

To evaluate the effectiveness of BRAID, we performed experiments on real and synthetic datasets. We compared BRAID with the naive implementation. As mentioned in Section 4, we can have "enhanced BRAID", where $b > 1$. That is, we keep more than one window-average for each level. We performed our experiments with $b = 16$, on an Intel Xeon 2.8GHz with 1GB of memory, running Linux.

The experiments were designed to answer the following questions:

1. How well does BRAID estimate the correlation coefficients for periodic and/or bursty datasets?
2. How successful is it in spotting lag correlations?
3. How does it scale with the sequence lengths n in terms of the computation time?

6.1 Datasets

We performed experiments on the following real and synthetic data sets. For synthetic data sets, we used the following:

- *Sines*: the data set consists of two sequences of length $n = 32,768$. Each sequence is a mixture of sine waves of different frequencies. We chose this setting because it resembles real data, such as network and automobile traffic (daily, weekly, and yearly periodicities), product sales (umbrellas, flu medicine - yearly periodicities), etc.
- *SpikeTrains*: a pair of periodic pulse trains with white noise. The period of these sequences is 6500, and the length is 100,000. This data set is also realistic: for example, disk access traffic is bursty [32], with daily periodicity; the famous sunspot dataset has spikes, with a 9-11 year periodicity [30].

For Real data sets, we used the following:

- *Humidity*, *Light*, *Temperature*: humidity, illuminance, and temperature readings, from 55 sensors within several buildings. Each sensor gives a reading every 30 seconds. We chose two sequences for each of the data sets, *Humidity* and *Light*, and used all sequences for *Temperature*.
- *Kursk*: Seismic recordings from multiple sensors, showing the explosion of the Russian submarine "Kursk" [25]. Each sequence has a single burst. We extracted two subsequences of length n=70,000.
- *Sunspots*: Number of sunspots per day. The dataset has a period of approximately 11 years [1]. We chose two intervals from the dataset, each length n=25,900, and treated them as if they were two different time sequences.

6.2 Detecting lag correlations

Figures 8 and 9 show the estimation of BRAID for all data sets. In these figures, "Naive" denotes the exact correlation coefficients computed by the naive implementation. "Approximation" means the correlation coefficients computed from the window averages. BRAID interpolates the missing values between these correlation coefficients.

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

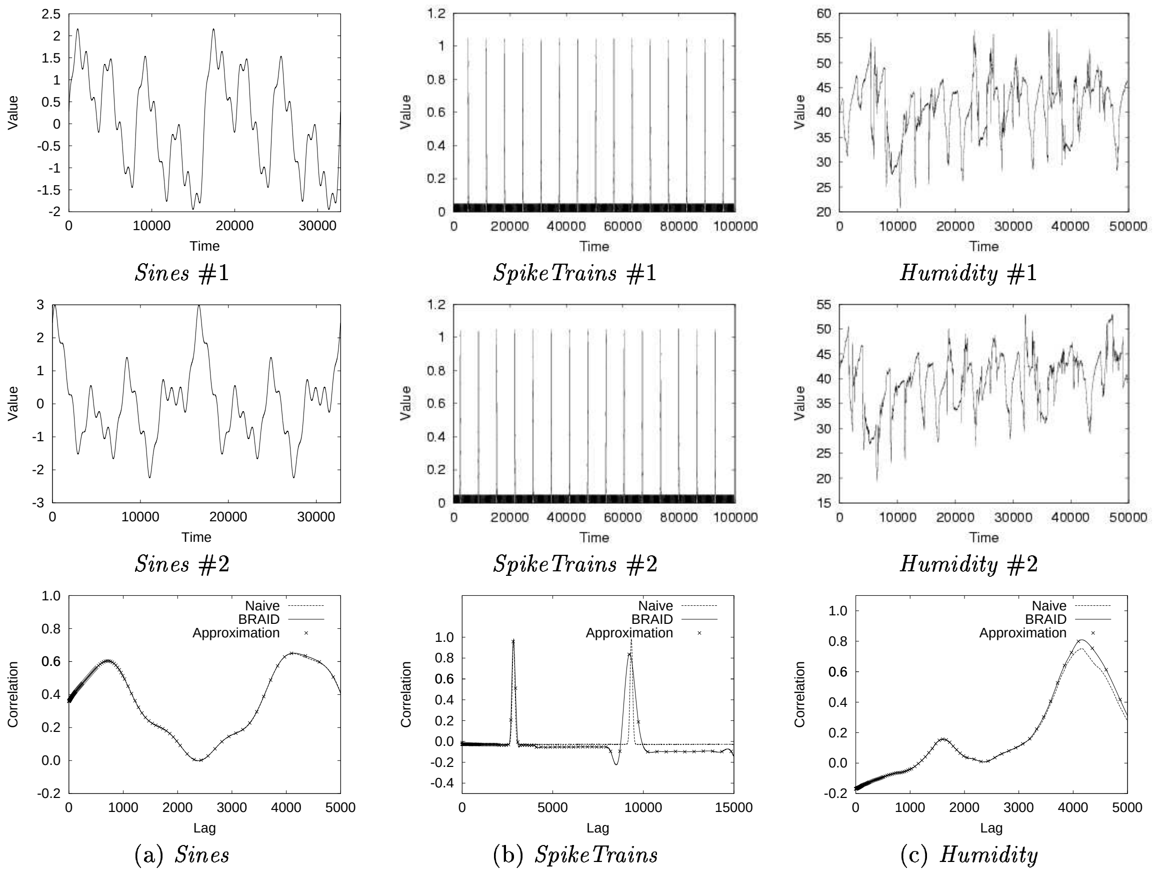


Figure 8: Estimation of the correlation coefficients (CCF) for *Sines*, *SpikeTrains*, and *Humidity*. The bottom row shows the CCF; "Naive", and BRAID, using dashed and solid lines, respectively.

Figure 8 (a) shows that BRAID perfectly approximates the correlation coefficients of the sinusoidal wave. Figure 8 (b) also indicates that BRAID closely estimates the correlation coefficients using interpolation and captures the lag correlations of the spike. Similar trends are shown in Figure 9. While the data values fluctuate dramatically with time ticks in real datasets, BRAID successfully captures the periodic lag correlations.

Table 2 shows the estimation error of the captured lag correlations. As shown in Definition 1, the most important lag correlation is the earliest local maximum. The experiments clearly demonstrate that BRAID detects the correct lag perfectly, most of the time. The largest relative error was about 1%.

Datasets	Lag correlation		Estimation error (%)
	Naive	BRAID	
Sines	716	716	0.000
SpikeTrains	2841	2830	0.387
Humidity	3842	3855	0.338
Light	567	570	0.529
Kursk	1463	1472	0.615
Sunspots	1156	1168	1.038

Table 2: Estimation error of lag correlations.

6.3 Performance

We theoretically discussed the complexity of BRAID in Section 5.2. However, BRAID needs not only the computation of correlation coefficients, but also overheads including the ones for the interpolation and extraction of the lag correlation from the spline curve. Therefore, we did an empirical study of the computation time.

Figure 10 compares BRAID and the naive implementation in terms of wall clock time under varying sequence lengths n. We used *Sines* for this experiment. The wall clock time is the average of the processing time to update sufficient statistics for each time tick and detect the lag correlations.

For stream data, since the sequence length continues to grow, the computation time also increases. Instead of the $O(n)$ that the naive implementation requires, BRAID can achieve a dramatic reduction in computation time. This trend shown in the figure corresponds to our theoretical discussion in Section 5.2. Theoretically, BRAID requires time $O(1)$ for updating the sufficient statistics; the computation time does not depend on n. In this experiment, the wall clock time increases slightly as n grows. This increase is caused by interpolation, because we need to interpolate through a larger, $O(\log n)$ number of points. Specifically, BRAID is up to about 40,000 times faster than the naive implementation.

6.4 Effect of probing

In Section 5.1.2 we provided that says when exactly BRAID can estimate the lag without error. (see Lemma 3) In this

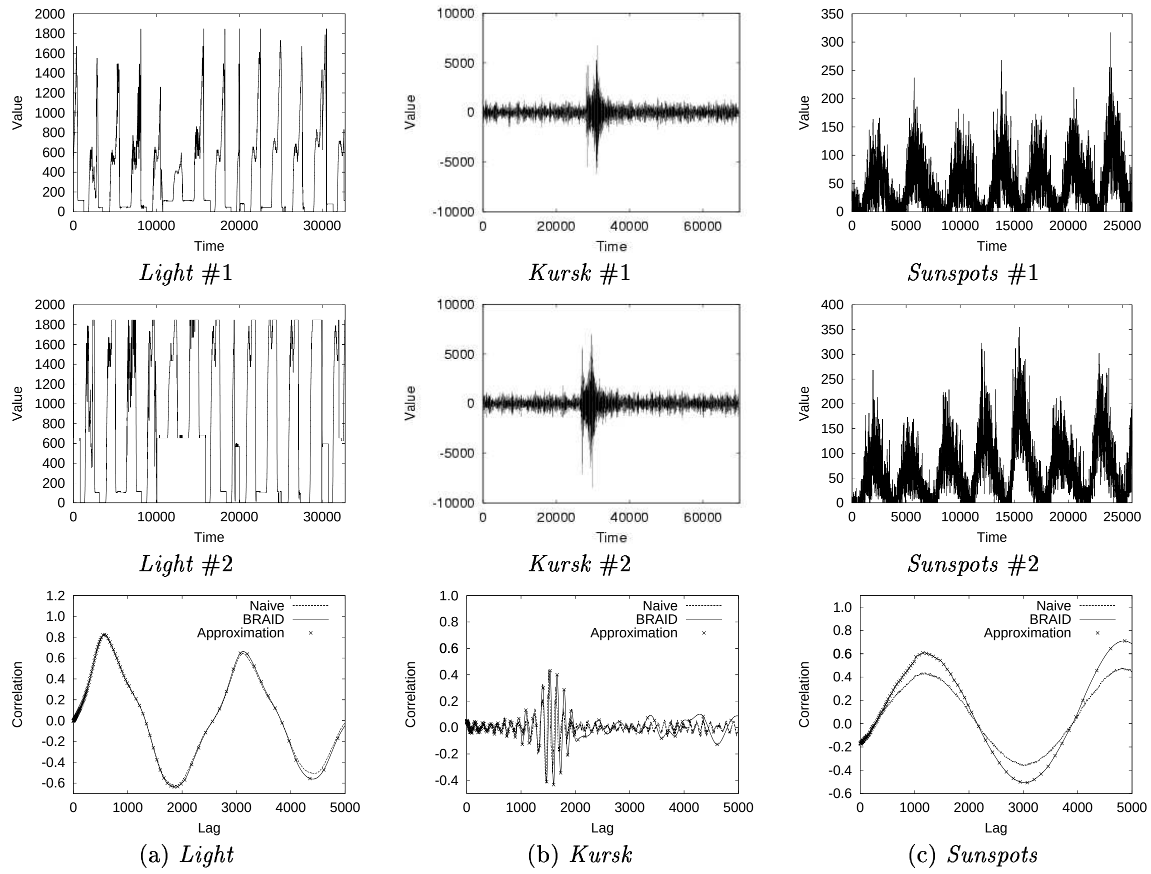


Figure 9: Estimation of correlation coefficients (cross-correlation function CCF) for *Light*, *Kursk*, and *Sunspots*. The bottom row shows the CCF; "Naive", and BRAID, using dashed and solid lines, respectively.

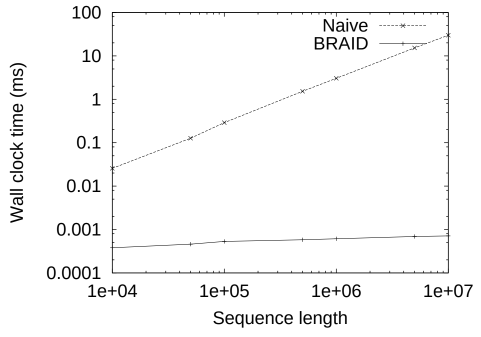


Figure 10: Wall clock time as a function of sequence length. BRAID can be up to 40,000 times faster.

section, we will show examples of the criterion and discuss the effect of probing by exploiting Lemma 3 to reinforce our theoretical analysis.

Figure 11 (a) shows the power spectrum of *Sines* in the frequency domain. Note that *Sines* #1 and #2 have the same power spectrum. The power spectrum is computed from the normalized sequences. That is, $\sum \mathcal{F}^2 = 1$ and $\mathcal{F}^2(f_0) = 0$. Intuitively, f on the y-axis means the first f Fourier coefficients. We can observe that in this figure, the energy is distributed in the range of $0 \le f \le 32$. Thus, the Nyquist frequency is $f_R = 64/n$. According to Lemma 3, we have $l_R = 1024b$. Figure 11 (b) presents an estimation of BRAID for *Sines*. We used one coefficient (i.e., $b = 1$) for each level in Figure 11 (b). We saw how BRAID can closely estimate the lag correlation with $b = 1$, because the lag is less than $l_R = 1024$.

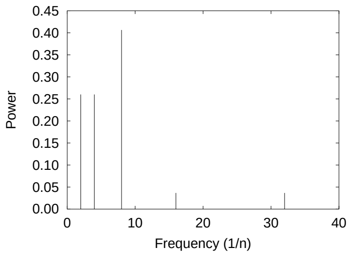
We next discuss the criterion for real data. Real data often include high frequencies of very small energy. The Nyquist frequency could be extremely high for such data. Since the frequency limit is widely understood in various areas (e.g., audio processing, network analysis, and electrical engineering), we will focus on the frequency components that are larger than a very small value, say ϵ. We set $\epsilon = 0.01$ in this experiment.

Figures 12 (a) and (b) present the power spectra of *Light*, and Figure 12 (c) shows the estimation for $b = 1$. The power of the frequency components larger than ϵ is distributed in the frequency range of $0 \le f \le 52$. Therefore, we have $l_R \approx 630$. As shown in Figure 12 (c), we can spot the lag correlation, because the local maximum is near l_R.

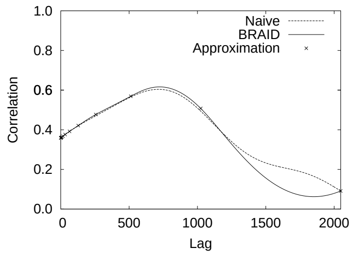
In Lemma 3, we expect the frequency of each sequence to obtain l_R. Incremental algorithms have been proposed to compute the frequency in the streaming sense (e.g., [17]). BRAID can utilize any and all of these solutions to compute the frequency efficiently. However, this research topic is beyond the scope of this paper.

6.5 Detecting group lag correlations

We applied BRAID to detecting group lag correlations

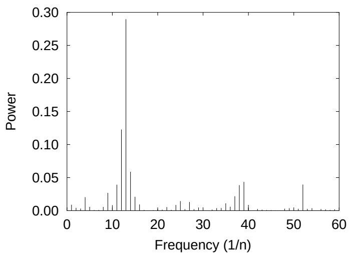


(a) Power spectrum of *Sines*

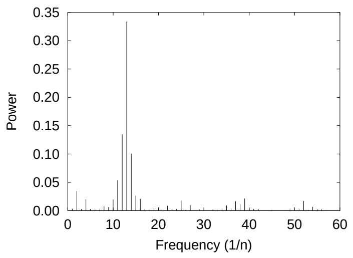


(b) Estimation of CCF

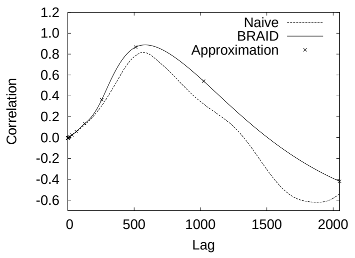
Figure 11: Estimation of CCF for $b = 1$ (*Sines*).



(a) Power spectrum of *Light* #1



(b) Power spectrum of *Light* #2



(c) Estimation of CCF

Figure 12: Estimation of CCF for $b = 1$ (*Light*).

for the 55 *Temperature* sequences. We chose two correlated sequence pairs and show them in Figure 13. The bottom row in the figure shows the CCF estimation by BRAID. The negative lags refer to the symmetric case. For examples, in Figure 13 (e), *Temperature* #16 is delayed if the two sequences are correlated with a positive lag; inversely, the negative lags mean a case when *Temperature* #19 is delayed.

Each sensor is located in a different place. Figures 13 (e)(f) show that each pair has its lag. Similar to these figures, BRAID has successfully detected the lag correlations for all other correlated sequence pairs.

7. CONCLUSIONS

We have introduced the problem of automatic lag correlation detection on streaming data and proposed BRAID to address this problem, using careful approximations and smoothing. The resulting method has all the desired characteristics:

1. *It is 'any-time':* it can give a response at any time, pinpointing the lag if it exists, or declaring that there is no lag correlation.
2. *Low resource consumption:* it needs $O(\log n)$ space, and $O(1)$ time to update the statistics.
3. *High accuracy:* it detects the correct lag within 1% relative error or less

Our experiments on real and realistic data show that BRAID works as expected, estimating the unknown lags with excellent accuracy and high speed. Specifically, BRAID can be up to 40,000 times faster than the naive implementation while the largest relative error was 1%.

8. REFERENCES

[1] D. J. Abadi, D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, M. Stonebraker, N. Tatbul, and S. B. Zdonik. Aurora: a new model and architecture for data stream management. *VLDB Journal*, 12(2):120–139, 2003.

[2] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. *FODO*, pages 69–84, Oct. 1993.

[3] B. Babcock, S. Babu, M. Datar, and R. Motwani. Chain : Operator scheduling for memory minimization in data stream systems. *ACM SIGMOD*, pages 253–264, June 2003.

[4] G. E. Box, G. M. Jenkins, and G. C. Reinsel. *Time Series Analysis: Forecasting and Control.* Prentice Hall, Englewood Cliffs, NJ, 3rd edition, 1994.

[5] R. P. Brent. *Algorithm for Minimization without Derivatives.* Dover Publications, 2002.

[6] D. Carney, U. Cetintemel, A. Rasin, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Operator scheduling in a data stream manager. *VLDB*, pages 838–849, Sept. 2003.

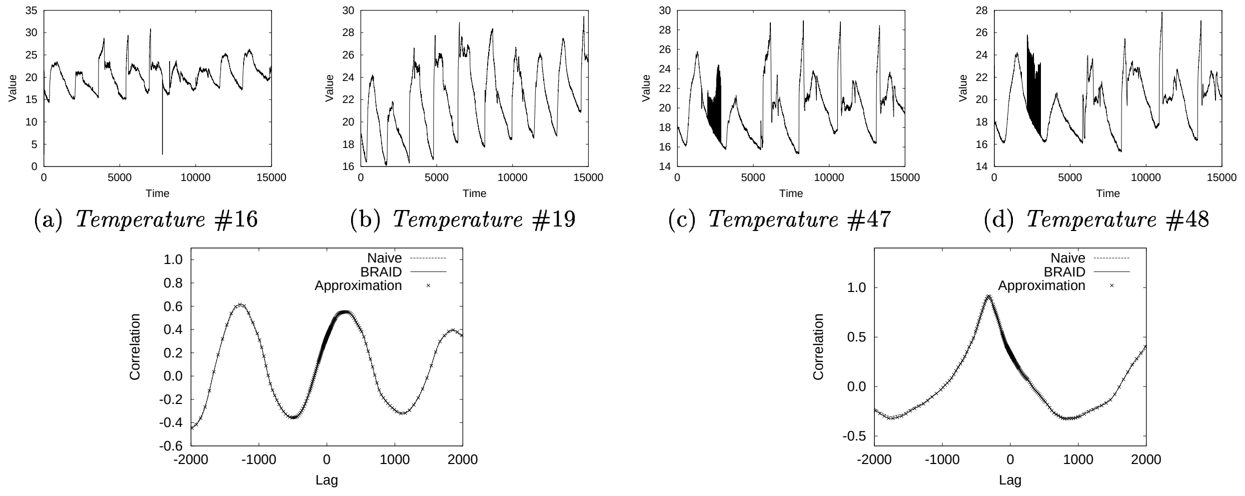
[7] S. Chandrasekaran, O. Cooper, A. Deshpande, M. J. Franklin, J. M. Hellerstein, W. Hong, S. Krishnamurthy, S. Madden, V. Raman, F. Reiss, and M. A. Shah. Telegraphcq: Continuous dataflow processing for an uncertain world. *CIDR*, Jan. 2003.

[8] S. Chandrasekaran and M. J. Franklin. Remembrance of streams past: Overload-sensitive management of archived streams. *VLDB*, pages 348–359, August-September 2004.

[9] C. D. Cranor, T. Johnson, O. Spatscheck, and V. Shkapenyuk. Gigascope: A stream database for network applications. *ACM SIGMOD*, pages 647–651, June 2003.

[10] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. *ACM SIGMOD*, pages 40–51, June 2003.

[11] A. Dobra, M. N. Garofalakis, J. Gehrke, and



(a) *Temperature* #16 (b) *Temperature* #19 (c) *Temperature* #47 (d) *Temperature* #48

(e) Estimation of CCF of #16 and #19 (f) Estimation of CCF of #47 and #48

Figure 13: Example of group lag correlations.

R. Rastogi. Processing complex aggregate queries over data streams. *ACM SIGMOD*, pages 61–72, June 2002.

[12] P. Domingos and G. Hulten. Mining high-speed data streams. *KDD*, pages 71–80, 2000.

[13] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proc. ACM SIGMOD*, pages 419–429, Minneapolis, MN, May 25-27 1994.

[14] G. E. Forsythe. *Computer Methods for Mathematical Computations.* Prentice-Hall, 1977.

[15] S. Ganguly, M. N. Garofalakis, and R. Rastogi. Processing set expressions over continuous update streams. *ACM SIGMOD*, pages 265–276, June 2003.

[16] V. Ganti, J. Gehrke, and R. Ramakrishnan. Mining data streams under block evolution. *SIGKDD Explorations*, 3(2):1–10, 2002.

[17] A. C. Gilbert, S. Guha, P. Indyk, S. Muthukrishnan, and M. Strauss. Near-optimal sparse fourier representations via sampling. *STOC*, pages 152–161, 2002.

[18] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. *VLDB*, pages 79–88, Sept. 2001.

[19] S. Guha, C. Kim, and K. Shim. Xwave: Approximate extended wavelets for streaming data. *VLDB*, pages 288–299, August-September 2004.

[20] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE TKDE*, 15(3):515–528, 2003.

[21] J. Han and M. Kamber. *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2000.

[22] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. *KDD*, pages 97–106, 2001.

[23] E. J. Keogh. Exact indexing of dynamic time warping. *VLDB*, pages 406–417, Aug. 2002.

[24] E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. J. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. *ACM SIGMOD*, pages 151–162, May 2001.

[25] K. Koper, T. Wallace, S. Taylor, and H. Hartse. Forensic seismology and the sinking of the kursk. *EOS Trans., AGU, 82*, pages 37,45–46, 2001.

[26] N. Koudas, B. C. Ooi, K.-L. Tan, and R. Zhang. Approximate nn queries on streams with guaranteed error/performance bounds. *VLDB*, pages 804–815, August-September 2004.

[27] B. P. Lathi. *Signal Processing and Linear Systems.* Oxford University Press, 1998.

[28] S. Madden, M. A. Shah, J. M. Hellerstein, and V. Raman. Continuously adaptive continuous queries over streams. *ACM SIGMOD*, pages 49–60, June 2002.

[29] R. Motwani, J. Widom, A. Arasu, B. Babcock, S. Babu, M. Datar, G. S. Manku, C. Olston, J. Rosenstein, and R. Varma. Query processing, approximation, and resource management in a data stream management system. *CIDR*, Jan. 2003.

[30] S. Papadimitriou, A. Brockwell, and C. Faloutsos. Adaptive, hands-off stream mining. *VLDB*, pages 560–571, Sept. 2003.

[31] N. Tatbul, U. Cetintemel, S. B. Zdonik, M. Cherniack, and M. Stonebraker. Load shedding in a data stream manager. *VLDB*, pages 309–320, Sept. 2003.

[32] M. Wang, T. Madhyastha, N. H. Chang, S. Papadimitriou, and C. Faloutsos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. *ICDE*, Feb. 2002.

[33] B.-K. Yi, N. Sidiropoulos, T. Johnson, H. Jagadish, C. Faloutsos, and A. Biliris. Online data mining for co-evolving time sequences. *ICDE*, pages 13–22, 2000.

[34] Y. Zhu and D. Shasha. Statistical monitoring of thousands of data streams in real time. *VLDB*, pages 358–369, Aug. 2002.

[35] Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. *KDD*, pages 336–345, 2003.