Fast Window Correlations Over Uncooperative Time Series*

Richard Cole Dennis Shasha Xiaojian Zhao
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
{cole,shasha,xiaojian}@cs.nyu.edu

1. ABSTRACT

Data arriving in time order (a data stream) arises in fields including physics, finance, medicine, and music, to name a few. Often the data comes from sensors (in physics and medicine for example) whose data rates continue to improve dramatically as sensor technology improves. Further, the number of sensors is increasing, so correlating data between sensors becomes ever more critical in order to distill knowlege from the data. In many applications such as finance, recent correlations are of far more interest than long-term correlation, so correlation over sliding windows (*windowed correlation*) is the desired operation. Fast response is desirable in many applications (e.g., to aim a telescope at an activity of interest or to perform a stock trade). These three factors – data size, windowed correlation, and fast response – motivate this work.

Previous work [10, 14] showed how to compute Pearson correlation using Fast Fourier Transforms and Wavelet transforms, but such techniques don't work for time series in which the energy is spread over many frequency components, thus resembling white noise. For such "uncooperative" time series, this paper shows how to combine several simple techniques – sketches (random projections), convolution, structured random vectors, grid structures, and combinatorial design – to achieve high performance windowed Pearson correlation over a variety of data sets.

2. MOTIVATION

Many applications, from space sensors to finance, generate multiple data streams. Such applications share the following characteristics:

- Updates come in the form of insertions of new elements rather than modifications of existing data.
- Data arrives continuously.
- One pass algorithms to filter the data are essential because the data is vast. However, if the filter does its job properly, there should be few enough candidates that even expensive detailed analysis per candidate will have only a modest impact on the overall running time. "Few enough" does not imply extremely high precision. In our experiments a precision of even 1% can still reduce computation times compared to a naive method by factors of 50 or more because the order of magnitude is linear instead of quadratic.

In this paper we describe one of several problems having to do with the relationship of many different time series: the discovery of time series windows having high Pearson correlations. We call this problem *windowed correlation*. In contrast to the considerable recent body of work on massive data streams [31, 32, 33] where the assumption is that data can be read once and is not stored, we assume an initial filtering step must be completed in one pass, but a second pass may search for data in a well-organized and potentially growing data structure.

3. PROBLEM STATEMENT

A data stream, for our purposes, is a potentially unending sequence of data in time order. For specificity, we consider data streams that produce one data item each time unit.

Correlation over windows from the same or different streams has many variants. This paper focusses on synchronous and asynchronous (a.k.a. lagged) variations, defined as follows.

- (Synchronous) Given N_s streams, a start time t_s, and a window size w, find, for each time window W of size w, all pairs of streams S_1 and S_2 such that S_1 during time window W is highly correlated (over 0.95 typically) with S_2 during the same time window. (Possible time windows are $[t_s \cdots t_s + w - 1]$, $[t_s + 1 \cdots t_s + w]$, $\cdots$)
- (Asynchronous correlation) Allow shifts in time. That is, given N_s streams and a window size w, find all time windows W_1 and W_2 where $|W_1| = |W_2| = w$ and all pairs of streams S_1 and S_2 such that S_1 during W_1 is highly correlated with S_2 during W_2.

3.1 What Makes a Time Series Cooperative?

Given N_s streams and a window of size $winsize$, computing all pairwise correlations naively requires $O(winsize \times (N_s)^2)$ time. Fortunately, extremely effective optimizations are possible, though the optimizations vary according to the type of time series.

*This work has been partly supported by the U.S. National Science Foundation under grants NSF IIS-9988345, N2010-0115586, MCB-0209754 and CCR-0105678.

KDD'05, August 21–24, 2005, Chicago, Illinois, USA.

- **Category 1 ("cooperative"):** The time series often exhibit a fundamental degree of regularity, at least over the short term, allowing long time series to be compressed to a few coefficients with little loss of information using data reduction techniques such as Fast Fourier Transforms and Wavelet Transforms. Using Fourier Transforms to compress time series data was originally proposed by Agrawal et al. [34]. This technique has been improved and generalized by [35, 30, 29]. Wavelet Transforms (DWT) [28, 15, 27, 26], Singular Value Decompositions (SVD) [25], and Piecewise Constant Approximations [36, 24, 23, 22] have also been proposed for similarity search. Keogh has pioneered many of the recent ideas in the indexing of dynamic time warping databases [21, 20]. The performance of these techniques varies depending on the characteristics of the datasets [10].

- **Category 2 ("uncooperative"):** In the general case, such regularities are absent. However, sketch-based approaches [11, 12] can still give a substantial data reduction. These are based on the idea of taking the inner product of each time series window, considered as a vector, with a set of random vectors (or equivalently, this can be regarded as a collection of projections of the time series windows onto the random vectors). Then the guarantees given by the Johnson-Lindenstrauss lemma [19] hold. In time series data mining, sketch-based approaches have been used to identify representative trends [18, 17], maintain histograms [16], and to compute approximate wavelet coefficients [15], for example.

4. CONTRIBUTIONS OF THIS PAPER

Previous work [14, 10] showed how to solve the windowed correlation problem in the cooperative setting using high quality digests obtained via Fourier transforms. Unfortunately, many applications generate uncooperative time series. Stock market returns (change in price from one time period (e.g., day, hour, or second) to the next divided by initial price, symbolically $(p_{t+1} - p_t)/p_t$) for example are "white noise-like." That is, there is almost no relation from one time point to the next.

For collections of time series that don't concentrate power in the first few Fourier/Wavelet coefficients, which we have termed *uncooperative*, we adopt a sketch-based approach. There are several difficulties to overcome:

1. Unfortunately, computing sketches directly for each neighboring window is very expensive. For each new datum, for each random vector, it costs $O(sw)$ time where sw is the size of the sliding window. (We will be using 25 to 60 random vectors.) To reduce this expense, we combine two ideas: convolutions and "structured random vectors" to reduce the time complexity to $O(sw/bw)$ integer additions and $O(\log\ bw)$ floating point operations per datum and random vetor. The length bw is the time delay before a correlation is reported (e.g., if sw were an hour then bw might be a minute).

2. Even with this, we obtain sketch vectors of too high a dimensionality for effective use of multi-dimensional data structures. We combat this well-known "curse of dimensionality" by using groups of sketches and combining the results as in the scheme due to [13].

3. There are four parameters to be set (two of which we introduce later). Optimizing these parameters to achieve good recall and precision requires a search through a large parameter space. For this we use combinatorial design. We validate both the use of combinatorial design and the stability of the parameter choices experimentally through bootstrapping.

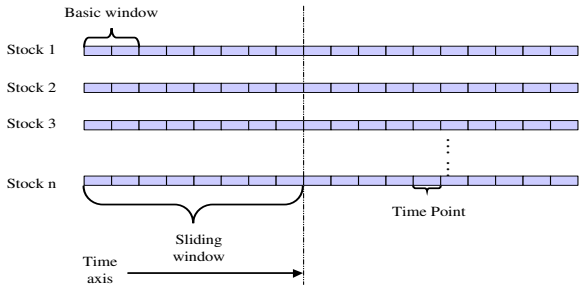


Figure 1: Sliding windows and basic windows.

The end result is a system architecture that, given the initial portions of a collection of time series streams, will determine (i) whether the time series are cooperative or not; (ii) if so, it will use Fourier or Wavelet methods (because they are faster by a constant); and (iii) if not, it will discover the proper parameter settings and apply them to compute sketches of the evolving data streams.

Thus, our contributions are of two kinds: (1) a greatly improved and more general solution to the on-line correlation problem; and (2) a synthesis of techniques – sketches, structured random vectors, combinatorial design with neighborhood search, and bootstrapping – that may be useful for many other problems.

5. ALGORITHMIC IDEAS

Following [10, 14], our approach begins by distinguishing among three time periods from smallest to largest.

- timepoint – the smallest unit of time over which the system collects data, e.g., a second.

- basic window – a consecutive subsequence of timepoints over which the system maintains a *digest* (i.e., a compressed representation) e.g., two minutes.

- sliding window – a user-defined consecutive subsequence of basic windows over which the user wants statistics, e.g., an hour. The user might ask, "which pairs of streams were correlated with a value of over 0.9 for the last hour?"

Figure 1 shows the relationship between sliding windows and basic windows.

The use of the intermediate time interval called the basic window yields two advantages [10, 14],

1. (Near online response rates) Results of user queries need not be delayed more than the basic window time. In this example, the user will be told about correlations for the 2PM to 3PM window by 3:02 PM and

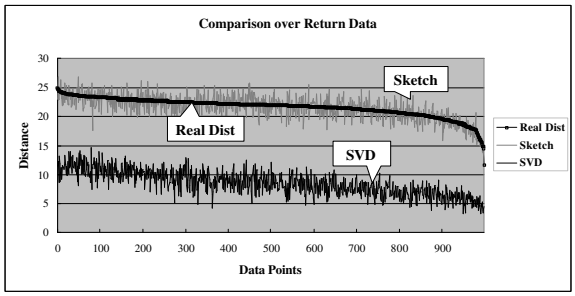


Figure 2: The sketch approach is superior to the Singular Value Decomposition, Wavelet, and Discrete Fourier Transform approaches for uncooperative time series. Of those three, Singular Value Decomposition is the best so is the one to which sketches are compared.

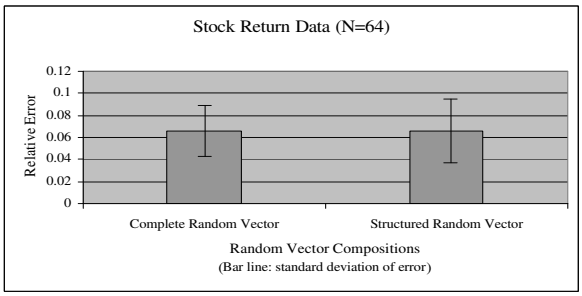


Figure 3: Real pairwise distance, estimated sketch distances for 64 random vectors, and estimated sketch distances for 64 structured random vectors.

correlations for the 2:02 PM - 3:02 PM window by 3:04 PM.[1]

2. (Free choice of window size) Maintaining stream digests based on the basic window allows the computation of correlations over windows of arbitrary size (chosen up front) with high accuracy.

5.1 The Sketch Approach

The sketch approach, as developed by Kushilevitz et al. [13], Indyk et al. [12], and Achlioptas [11], provides a very nice guarantee: with high probability a random mapping taking points in R^m to points in $(R^d)^{2b+1}$ (the (2b+1)-fold cross-product of R^d with itself) approximately preserves distances (with higher fidelity the larger b is).

Quantitatively, given a point $\mathbf{x} \in R^m$, we compute its dot product with d random vectors $\mathbf{r}_i \in \{1,-1\}^m$. The first random projection of $\mathbf{x}$ is given by $\mathbf{y}_1 = (\mathbf{x} * \mathbf{r_1}, \mathbf{x} * \mathbf{r_2}, ..., \mathbf{x} * \mathbf{r_d})$. We compute $2b$ more such random projections $\mathbf{y_1}, ..., \mathbf{y_{2b+1}}$. If $\mathbf{w}$ is another point in R^m and $\mathbf{z_1}, ..., \mathbf{z_{2b+1}}$ are its projections using dot products with the same random vectors then the median of $\|\mathbf{y_1} - \mathbf{z_1}\|, \|\mathbf{y_2} - \mathbf{z_2}\|, ...\|\mathbf{y_{2b+1}} - \mathbf{z_{2b+1}}\|$ is a good estimate of $\|\mathbf{x} - \mathbf{w}\|$. It lies within a $\theta(1/d)$ factor of $\|\mathbf{x} - \mathbf{w}\|$ with probability $1 - (1/2)^b$.

Sketches work much better than Fourier methods for uncooperative data. Figure 2 compares the distances of the Fourier and sketch approximations for 1,000 pairs of 256 timepoint windows having a basic window size of length 32. As you can see, the sketch distances are close to the real distances. On the other hand, the Fourier, Wavelet, and even SVD approximations work very poorly for uncooperative data, because the information needed to capture uncooperative time series is spread out over all their coefficients.

Our approach is to use a "structured" random vector. The apparently oxymoronic idea is to form each structured random vector $\mathbf{r}$ from the concatenation of $nb = sw/nb$ random vectors: $\mathbf{r} = \mathbf{s_1}, ..., \mathbf{s_{nb}}$, where each $\mathbf{s_i}$ has length bw. Further each $\mathbf{s_i}$ is either $\mathbf{u}$ or $-\mathbf{u}$, and $\mathbf{u}$ is a random vector in $\{1,-1\}^{bw}$. This choice is determined by a random binary nb-vector $\mathbf{b}$: if b_i=1, $\mathbf{s_i}$=$\mathbf{u}$ and if b_i=0, $\mathbf{s_i}$=$-\mathbf{u}$. The structured approach leads to an asymptotic performance of $O(nb)$ integer additions and $O(\log bw)$ floating point operations per datum and per random vector. In our applications, we see 30 to 40 factor improvements over the naive method.

In order to compute the dot products with structured random vectors, we first compute dot products with the random vector $\mathbf{u}$. We perform this computation by convolution once every bw timesteps. Then each dot product with $\mathbf{r}$ is simply a sum of nb already computed dot products. (We explain this in more detail in the appendix of [4].)

The use of structured random vectors reduces the randomness, but experiments show that this does not appreciably diminish the accuracy of the sketch approximation, as we can see from Figure 3.

Though structured random vectors enjoy good performance, as we will see, please note that a clever use of unstructured (that is, standard) random vectors together with convolutions can lead to an asymptotic cost of $O(\log\ sw \log(sw/bw))$ floating point multiplications per datum. Structured random vector approaches use $O(\log bw)$ multiplications and $O(sw/bw)$ additions per datum. For the problem sizes we consider in this paper, the structured random vector approach is faster, though in principle it needs to be weighed against the small loss in accuracy.

5.2 Partitioning Sketch Vectors

In many applications, sketch vectors are of length up to 60. (In such a case, there are 60 random vectors to which each window is compared and the sketch vector is the vector of the dot products with those random vectors). Multidimensional search structures don't work well for more than 4 dimensions in practice [10]. Comparing each sketch vector with every other one destroys scalability though because it introduces a term proportional to the square of the number of windows under consideration.

For this reason, we adopt an algorithmic framework that partitions each sketch vector into subvectors and builds data structures for the subvectors. For example, if each sketch vector is of length 40, we might partition each one into ten groups of size four. This would yield ten data structures. We then combine the closeness results of pairs from each data structure to determine an overall set of candidate correlated windows [2].

[1] One may wonder whether the basic window and therefore the delay can be reduced. The tradeoff is with computation time. Reducing the size of the basic window reduces the compression achieved and increases the frequency and hence expense of correlation calculations.

[2] Note that we use correlation and distance more or less in-

5.3 Algorithmic Framework

Given the idea of partitioning sketch vectors, we have to say how to combine the results of the different partitions. This introduces four parameters, as we will see. Suppose we are seeking points within some distance d in the original timeseries space.

- Partition each sketch vector s of size N into groups of some size g.
- The ith group of each sketch vector s is placed in the ith grid structure (of dimension g).
- If two sketch vectors $s1$ and $s2$ are within distance $c \times d$ in more than a fraction f of the groups, then the corresponding windows are candidate highly correlated windows and should be checked exactly. The remaining issue is to choose good values for c, f and g

5.4 Combinatorial Design

The above framework eliminates the curse of dimensionality by making the groups small enough that multi-dimensional search structures (even grid structures) can be used. The framework also introduces the challenge of optimizing the settings of four parameters: the length N of the sketch vector, the size g of each group, the distance multiplier c, and the fraction f.

Our optimization goal is to achieve extremely high recall (above 0.95) and reasonable precision (above 0.02). We are satisfied with a fairly low precision because examining even 50 times the number of the winning pairs on the raw data is much much better than examining all pairs, as we show later in our experiments.

Increasing the size of the sketch vector improves the accuracy of the distance estimate but increases the search time. In our experiments, accuracy improved noticeably as the sizes increased to about 60; beyond that, accuracy did not improve much. Larger group sizes also improve accuracy, but increase the search time. A typical set of possible parameter values therefore would be:

Size of Sketch (N): 30, 36, 48, 60
Group Size (g): 1, 2, 3, 4
Distance Multiplier (c): 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1, 1.1, 1.2, 1.3
Fraction (f): 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1

As we will see, every possible selection of parameter values requires a test on many pairs of windows (typically a few million) in order to get a robust set of parameters. For this reason, we would like to avoid testing all possible settings (2,080 in this example). Instead, we use combinatorial design.

Combinatorial design is effectively a disciplined sampling approach with some guarantees [9]. The key idea of n-factor combinatorial design is that the tests will cover all n-way combinations of parameters. For concreteness, two factor combinatorial design requires that for every pair of parameters (a.k.a. factors) $p1$ and $p2$ and for every value $v1$ from $p1$ and $v2$ from $p2$, some experiment will test $p1.v1$ and $p2.v2$ together. This property is not the same as exhaustive search, of course. For example, if there were 4 binary variables, one possible two factor combinatorial design would be the one found in table 1.

a1	a2	a3	a4
0	0	0	0
0	1	0	1
1	0	1	0
1	1	1	1
0	0	1	1
1	1	0	0

Table 1: Example of Two-Factor Combinatorial Design.

Data title	$prec_{cd}^{mean}$	$prec_{cd}^{std}$	$prec_{ex}^{mean}$	$prec_{ex}^{std}$
spot_exrates	0.18	0.02	0.2	0.03
cstr	0.16	0.02	0.18	0.03
foetal_ecg	0.22	0.01	0.25	0.008
evaporator	0.007	0.0001	0.007	0.0001
steamgen	0.32	0.02	0.34	0.01
wind	0.001	0.001	0.001	0.0001
winding	0.05	0.02	0.06	0.02
eeg	0.12	0.03	0.14	0.07
price	0.11	0.04	0.14	0.03
return	0.008	0.002	0.009	0.001

Table 2: Combinatorial design vs. exhaustive search over a parameter search space of size 2,080.

In our example, a two-factor combinatorial design would reduce the number of experiments from 2,080 to only 130.

When faced with a sampling proposal like combinatorial design, one must ask whether some global optimum is missed through sampling. This could be a particularly significant issue if small changes in parameter values could yield large changes in time or quality of result. We call such a situation *parameter discontinuity* and the hoped-for opposite *parameter continuity*.

Fortunately, across a wide variety of data sets, our framework appears to enjoy parameter continuity. So the best value found by combinatorial design is close to that returned by exhaustive search. Table 2 illustrates this. In the table, we have listed the precision of the best parameters for each data set after doing the bootstapping tests. Here "best" is defined as those having average recall ≥ 0.99 and standard deviation for recall≤ 0.001 as well as reasonably high precision and low standard deviation for precision.

In fact, we use parameter continuity in a second way: the c and f values may take any real value. For the purposes of sampling them with combinatorial design, however, we make them discrete. Once we find a good set of discrete values, we may want to find better values by exploring a local neighborhood around that good set. For example, if the optimal set has $c = 0.7$, then we will search $0.63, 0.64, 0.65, 0.66, 0.67, \cdots, 0.74,\ 0.75, 0.76, 0.77$. We call this local neighbor-

terchangeably because one can be computed from the other once the data is normalized. Specifically, Pearson correlation is related to Euclidean distance as follows:

$$D^2(\hat{x}, \hat{y}) = 2(1 - corr(x, y))$$

Here $\hat{x}$ and $\hat{y}$ are obtained from the raw time series by computing $\hat{x} = \frac{x - avg(x)}{\sigma_x}$, where $\sigma_x = \sqrt{\sum_{i=1}^{n}(x_i - avg(x))^2}$.

hood search *refinement*. To see whether separating the refinement step from the initial parameter search works well, we tested whether an exhaustive search on a dense parameter space (c values having two digits of precision in our case) would have yielded a substantially different result from a combinatorial design followed by refinement approach. Using combinatorial design gave the same recall as exhaustive search and a precision at least 86% as good as exhaustive search across the variety of data sets from the UC Riverside collection[4, 7].

5.5 Bootstrapping To Determine Parameter Robustness

Optimizing parameter settings for one data sample may not yield good parameter settings for others. For example, suppose that we find the optimal parameter settings for stock return data over the first month. Will those settings still work well for a later month? Without further assumptions we cannot answer this, but we can estimate out-of-sample variability by using bootstrapping [8].

The goal of bootstrapping is to test the robustness of a conclusion on a sample data set by creating new samples from the initial sample with replacement. In our case, the conclusion to test is whether a given parameter setting with respect to recall and precision shows robust good behavior. To be concrete, suppose we take a sample S of one million pairs of windows. A bootstrapped sample would consist of one million pairs drawn from S with replacement. Thus the newness of a bootstrapped sample comes from the duplicates.

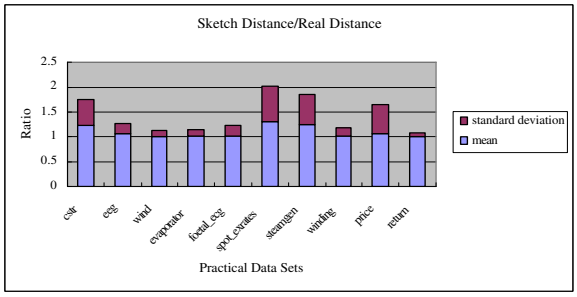
We use bootstrapping to test the stability of a choice of parameters. After constructing each bootstrapped sample, we check the recall and precision of that sample given our chosen parameter settings. Provided the mean recall over all bootstrapped samples less the standard deviation of the recall is greater than our threshold (say 0.95) and the standard deviation for precision is low, then the parameter setting is considered to be good. This admittedly heuristic criterion for goodness reflects the idea that the parameter setting is "usually good" (under certain normality assumptions roughly 3/4 of the time).[3]

Otherwise, we take a bigger sample, perform combinatorial design, optimize, bootstrap, and do the standard deviation test again.

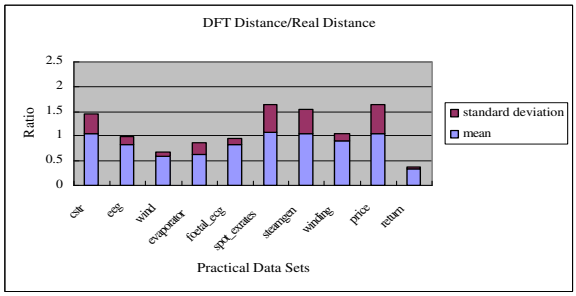
6. EXPERIMENTS

Our approach has many interacting parts. We use sketches, partition them into groups, and then combine the results from the groups. We use an optimization approach based on sampling (two-factor combinatorial design) of the parameter space and of the data space. None of this can be well-justified theoretically without some rather onerous assumptions.

[3] An alternative, which avoids a parametric normality assumption, is to sort the calculated recalls over all the bootstraps and take the value that is x% from the lowest where x could be typically 1, 5 or 25. We have done this for our data (results not shown) and recalls of the 1% from lowest bootstrap value are very close to the mean whenever the normality assumptions lead us to a "usually good" conclusion. Precisions can vary by a factor of two however. In summary, both parametric and non-parametric tests lead to the same conclusions on these data sets.



(a) Sketch



(b) DFT

Figure 4: DFT distance versus sketch distance over empirical data

Fortunately, we have several data sets from stock market data and from the UC Riverside repository [7] that afford us an empirical test of the method.[4]

The Hardware is a 1.6G, 512M RAM PC running RedHat 8.0. The language is K (www.kx.com).

6.1 Experiment: how common is the uncooperative case?

In this experiment, we took a window size of 256 ($sw = 256$ and $bw = 32$) across 10 data sets and tested the accuracy of the Fourier coefficients as an approximation to distance compared with structured random vector-based sketches. Figure 4 shows that the Discrete Fourier Transform-based distance perform badly in some data types while our sketch based distance works stably across all the data sets. On the other hand, when the time series closely resemble a random walk, as for stock price data, the Fourier series approach gives significantly better precision levels at the same recall as compared with the sketch method. Database people will appreciate the following analogy to data structures: sketches are like B-trees (the default choice) and Fourier Transform approaches are like bit vectors (better in some cases).

[4]The stock data in the experiments are end-of-day prices from 7,861 stocks from the Center for Research in Security Prices (CRSP) at Wharton Research Data Services (WRDS) of the University of Pennsylvania [6]. All the other empirical data sets came from the UC Riverside Time Series Data Mining Archive [7] maintained by Eamonn Keogh. The number of time series in each of those data sets ranges from 1,365 to 13,736.

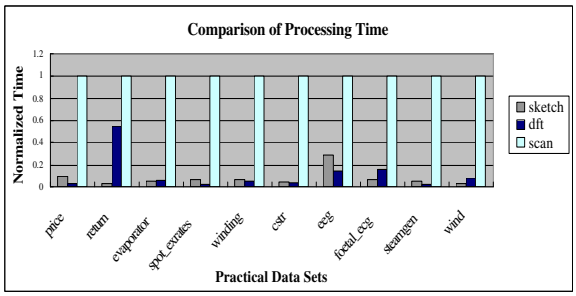


Figure 5: System performance over a variety of datasets. Minimum recall for approximation method is 99%

6.2 Experiment: How good is bootstrapping?

The operational claim of bootstrapping is to simulate samples across a whole data set by repeated samples from a single initial sample. In our case, we want the optimal parameters found on one sample (with bootstrapping) to meet the recall and precision thresholds on completely disjoint samples. As shown in our technical report [4], using the parameters derived from a training sample (and confirmed by bootstrapping) of a data set works well across that entire data set. We also show there that different data sets should have different sets of parameters.

6.3 Performance Tests

The previous subsection shows that the sketch framework gives a sufficiently high recall and precision. The next question is what is the performance gain of using (i) our sketch framework as a filter followed by verification on the raw data from individual windows compared with (ii) simply comparing all window pairs. Because the different applications have different numbers of windows, we take a sample from each application, yielding the same number of windows.

To make the comparison concrete, we should specify our software architecture a bit more. The multi-dimensional search structure we use is in fact a grid structure. The reason we have rejected more sophisticated structures is that we are asking a radius query: which windows (represented as points) are within a certain distance of a given point? A multi-scale structure such as a quadtree or R-tree would not help in this case. Moreover, the grid structure can be stored densely in a hash table so empty cells take up no space.

Figure 5 compares the results from our system, a Fourier-based approach, and a linear scan over several data sets. To perform the comparison we normalize the results of the linear scan to 1. The figure shows that both the sketch-based approach described here and the Fourier-based approach are much faster than the linear scan. Neither is consistently faster than the other. However as already noted, the sketch-based approach produces consistently accurate results though the Fourier-based one wins when the data resembles a random walk.

7. SUMMARY AND FUTURE WORK

Correlation can indicate that two time series exhibit similar trends. Windowed correlation is useful because it indicates that the two time series trend together over a certain duration of time. We call this a *co-trending measure*. Many applications such as finance privilege the recent past (e.g., the last two hours) over the distant past (e.g., yesterday), so an important special case has to do with windowed correlation over the most recent windows.

This paper has proposed an efficient algorithm for rapidly discovering highly correlated windows synchronously or with lags. Experiments validate the usefulness of our algorithm for many data sets, both synthetic and real.

Our measure of correlation is Pearson correlation which is closely related to Euclidean distance over a normalized vector space. Other co-trending measures have of course been invented such as cointegration [5], mutual information [3], and matching pursuit [2]. Preliminary indications are that techniques similar to the ones we have studied may be used for those problems. The long-term goal is to find fast algorithms for data-rich co-trending problems. The present paper is a step in that direction.

8. ACKNOWLEDGMENT

Warm thanks to Eamonn Keogh of University of California at Riverside for his data sets.

9. REFERENCES

[1] A. Gionis, P. Indyk and R. Motwani, Similarity Search in High Dimensions via Hashing, VLDB, 518-529, 1999

[2] S. Mallat and Z. Zhang, Matching Pursuit With Time-Frequency Dictionaries, IEEE Transactions on Signal Processing, 1993

[3] T. M. Cover and J. A. Thomas, Elements of Information Theory, New York, Wiley, 1991

[4] R. Cole, D. Shasha and X. J. Zhao, Fast Window Correlations Over Uncooperative Time Series, Department of Computer Science, New York University, New York, NY, Technical Report, 2005

[5] C. Alexander, Market Models: A Guide to Financial Data Analysis, John Wiley & Sons, 2001

[6] Wharton Research Data Services(WRDS), http://wrds.wharton.upenn.edu/

[7] E. Keogh and T. Folias, The UCR Time Series Data Mining Archive. Riverside CA. University of California - Computer Science & Engineering Department, http://www.cs.ucr.edu/∼eamonn/TSDMA/index.html, 2002

[8] B. Efron and R. J. Tibshirani, An Introduction to the Bootstrap, Chapman & Hall/CRC, 1994

[9] D. M. Cohen, S. R. Dalal, J. Parelius and G. C. Patton, The Combinatorial Design Approach to Automatic Test Generation, IEEE Software, 13, 83-87, 1996

[10] D. Shasha and Y. Zhu, High Performance Discovery in Time Series: Techniques and Case Studies, Springer, 2003

[11] D. Achlioptas, Database-friendly Random Projections, ACM SIGMOD-PODS, May, Santa Barbara, CA, 2001

[12] P. Indyk, Stable distributions, pseudorandom generators, embeddings and data stream computation, Proceedings of the 41st Annual Symposium on Foundations of Computer Science, 0-7695-0850-2, 189, IEEE Computer Society, 2000

[13] E. Kushilevitz, R. Ostrovsky and Y. Ranbani, Efficient Search for Approximate Nearest Neighbors in High Dimensional Spaces, STOC, 1998

[14] Y. Zhu and D. Shasha, StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time, VLDB, Hong Kong, China, August, 2002

[15] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan and M. Strauss, Surfing wavelets on streams: One-pass summaries for approximate aggregate queries, VLDB, 2001

[16] N. Thaper, S. Guha, P. Indyk and N. Koudas, Dynamic multidimensional histograms, SIGMOD, Madison, Wisconsin, 2002

[17] P. Indyk, N. Koudas and S. Muthukrishnan, Identifying Representative Trends in Massive Time Series Data Sets using Sketches, VLDB, 2000

[18] G. Cormode, P. Indyk, N. Koudas and S. Muthukrishnan, Fast mining of massive tabular data via approximate distance computations, ICDE, 2002

[19] W. Johnson and J. Lindenstrauss, Extensions of Lipschitz mapping into hilbert space, Contemporary Mathematics, 26, 189-206, 1984

[20] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos and E. Keogh, Indexing multi-dimensional time-series with support for multiple distance measures, SIGKDD, 2003

[21] E. Keogh, Exact indexing of dynamic time warping, VLDB, 2002

[22] B. K. Yi and C. Faloutsos, Fast time sequence indexing for arbitrary Lp forms, VLDB, 2000

[23] T. Palpanas, M. Vlachos, E. Keogh, D. Gunopulos and W. Truppel, Online amnesic approximation of streaming time series, ICDE, 2004

[24] E. Keogh, K. Chakrabarti, S. Mehrotra and M. Pazzani, Locally Adaptive Dimensionality Reduction for Indexing large Time Series Databases, SIGMOD, 2001

[25] F. Korn, H. V. Jagadish and C. Faloutsos, Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences, SIGMOD, 1997

[26] Y. L. Wu, D. Agrawal and A. E. Abbadi, A comparison of dft and dwt based similarity search in time-series databases, The 9th ACM CIKM Int'l Conference on Information and Knowledge Management, 2000

[27] I. Popivanov and R. Miller, Similarity Search Over Time Series Data Using Wavelets, ICDE, 2002

[28] K. Chan and A. W. Fu, Efficient Time Series Matching by Wavelets, ICDE, 1999

[29] D. Rafier and A. Mendelzon, Similarity-based queries for time series data, SIGMOD, 1997

[30] C. S. Li, P. S. Yu and V. Castelli, Hierarchyscan: A hierarchical similarity search algorithm for databases of long sequences, ICDE, 1996

[31] E. Drinea, P. Drineas and P. Huggins, A Randomized Singular Value Decomposition Algorithm for Image Processing, Panhellenic Conference on Informatics (PCI), 2001

[32] G. Manku, S. Rajagopalan and B. Lindsay, Random Sampling Techniques for Space Efficient Online Computation of order Statistics of Large Datasets, SIGMOD, 1999

[33] M. Greenwald and S. Khanna, Space-Efficient Online Computation of Quantile Summaries, SIGMOD, 2001

[34] R. Agrawal, C. Faloutsos and A. Swami, Efficient Similarity Searching In Sequence Databases, Proceedings of the 4th International Conference of Foundations of Data organization and Algorithms (FODO), Springer Verlag, Chicago, Illinois, MN, 69-84, 1993

[35] C. Faloutsos, M. Ranganathan and Y. Manolopoulos, Fast subsequence matching in time-series databases, SIGMOD, Minneapolis, MN, May, 1994

[36] E. Keogh, K. Chakrabarti, M. Pazzani and S. Mehrotra, Dimensionality Reduction for fast similarity search in large time series databases, Knowledge and Information Systems, 3, 263-286, 2000