Fast Window Correlations Over Uncooperative Time Series

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 [29, 34] showed how to compute Pearson correlation using Fast Fourier Transforms and Wavelet transforms, but such techniques don't work for time series where the energy is not concentrated in only a few frequency components, e.g. white noise, non-stationary etc. 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. Experiments verify the asymptotic arguments.

2. NOTE TO REVIEWERS

This paper was submitted to Sigmod 2005 and was rejected. The rejection was based on stylistic and clarity grounds that have been rectified here. This paper is not summer beach reading, but we think you will find its results and algorithms interesting.

3. MOTIVATION

Many applications generate multiple data streams. For example,

- The earth observing system data project consists of 64,000 time series covering the entire earth [5] though the satellite covers the earth in swaths, so the time series have gaps.
- There are about 50,000 securities traded in the United States, and every second up to 100,000 quotes and trades (ticks) are generated.

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.

Particularly significant are multi-stream statistics, because they permit the fusion of information from multiple sources. 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.* We have used related techniques to solve financial problems such as the online discovery of a few securities that represent a stock index consisting of hundred of securities. Thus these techniques help solve not only this problem, but also many related ones, even when the time series are uncooperative.

We contrast this work with the considerable recent body of work on massive data streams [9, 25, 14] where the assumption is that data can be read once and is not stored. In our applications, 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.

4. 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_{start}, 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 $[t_{start} \cdots t_{start+w-1}]$, $[t_{start+1} \cdots t_{start+w}]$, $\cdots$ where t_{start} is some start time.)
- (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.

4.1 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 depend on the kind of time series they are.

- **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. [3]. This technique has been improved and generalized by [11, 24, 28]. Wavelet Transforms (DWT) [6, 12, 27, 32], Singular Value Decompositions (SVD) [22], and Piecewise Constant Approximations [21, 19, 26, 33] have also been proposed for similarity search. Keogh has pioneered many of the recent ideas in the indexing of dynamic time warping databases [18, 31]. The performance of these techniques varies depending on the characteristics of the datasets [29].
- **Category 2 ("uncooperative"):** In the general case, such regularities are absent. However, sketch-based approaches [2, 15] 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). Thus, the guarantees given by the Johnson-Lindenstrauss lemma [17] hold. In time series data mining, sketch-based approaches have been used to identify representative trends [8, 16], maintain histograms [30] and to compute approximate wavelet coefficients [12], for example.

5. CONTRIBUTIONS OF THIS PAPER

Previous work [34, 29] showed how to solve the windowed correlation problem in the cooperative setting using high quality digests obtained based on 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 correlation 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 proceed as follows:

1. We adopt a sketch-based approach.
2. 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).
3. 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 [23].
4. 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.

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; 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.

6. ALGORITHMIC IDEAS

Following [29, 34], our approach begins by distinguishing among three time periods from smallest to largest.

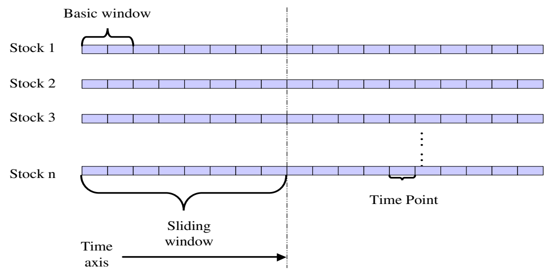


Figure 1: Sliding windows and basic windows.

- 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 [29, 34],

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 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.

For cooperative time series, the strategy is just this: given a digest consisting of the first few, say k, Fourier coefficients for a sliding window up to basic window number n, recompute the digest after basic window $n+1$ ends as follows. Compute a digest of basic window $n+1$ and then update the sliding window digest's Fourier coefficients. This update takes constant time per coefficient.

Fourier Transforms do an excellent job of summarizing many time series, as pointed out by [3] and subsequently used in [11, 13, 24, 28]. Empirical studies [34] using both random walk and stock market price data show that the cost savings compared to a naive approach are significant and the accuracy is excellent (no false negatives and few false positives). Unfortunately, as pointed out above, the Fourier Transform behaves very poorly for white noise style data. For such data, we use sketches.

[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.

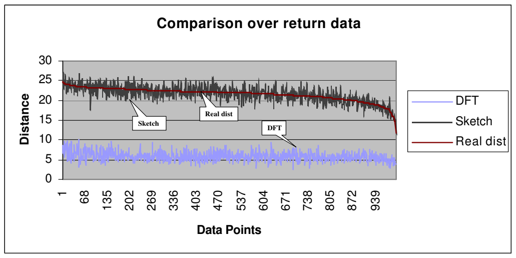


Figure 2: Discrete Fourier Transform and Sketch estimates for return data.

6.1 The Sketch Approach

The sketch approach, as developed by Kushikvitz et al. [23], Indyk et al. [15], and Achlioptas [2], 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 distance. On the other hand, the Fourier approximation is essentially never correct.

For each random vector $\mathbf{r}$ of length equal to the sliding window length $sw = nb \times bw$, we compute the dot product with each successive length sw portion of the stream (successive portions being one timepoint apart and bw being the length of a basic window). As noted by Indyk [16], convolutions (computed via Fast Fourier Transforms) can perform this efficiently off-line. The difficulty is how to do this efficiently online.

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 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 k-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

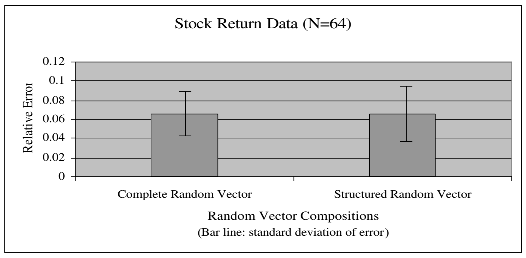


Figure 3: Real pairwise distance, estimated sketch distances for 64 random vectors, and estimated sketch distances for 64 structured random vectors.

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.)

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 $O(\log(sw/bw))$ additions per datum. For the problem sizes we consider in this paper, the structured random vector approach is faster, though in principle it must be weighed against the small loss in accuracy.

6.2 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 to those random vectors). Multi-dimensional search structures don't work well for more than 4 dimensions in practice [29]. 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 candidates of correlated windows.

Note that we use correlation and distance more or less interchangeably 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 $\hat{x} = \frac{x - avg(x)}{\sigma_x}$, here $\sigma_x = \sqrt{\sum_{i=1}^{n}(x_i - avg(x))^2}$.

6.3 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. c, d and f are defined in next paragraph.

6.4 Combinatorial Design

This 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 50 times the necessary 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

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 1,560.

(1,560 in this example). Instead, we use combinatorial design.

Combinatorial design is effectively a disciplined sampling approach with some guarantees [7]. 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.

In our example, a two-factor combinatorial design would reduce the number of experiments from 1,560 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 $\geq$ 0.99 and standard deviation for recall$\leq$ 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 neighborhood 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.

Table 3 shows that the two approaches achieve very similar results. The table also shows the relationship between precision and the number of pairs whose distances are slightly greater than the target distance. Population Ratio 1 is the number of pairs with distance value 1.1 $\times$ desired distance divided by the number of pairs within the desired distance. Population Ratio 2 is the ratio based on 1.2 $\times$ the desired distance. When these ratios are high, the precisions for both combinatorial design and exhaustive approaches are low.

6.5 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 [10].

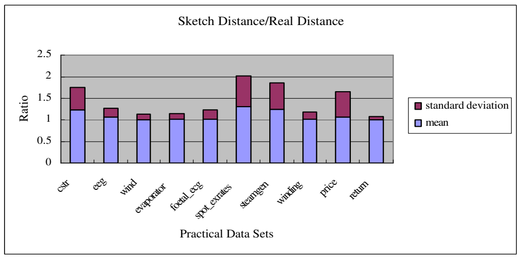
The goal of bootstrapping is to test the robustness of a conclusion on a sample data set by creating new samples from the initial sample with replacement. In our case, the conclusion to test is whether a given parameter setting with respect to recall and precision shows robust good behavior. To be concrete, 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).[2]

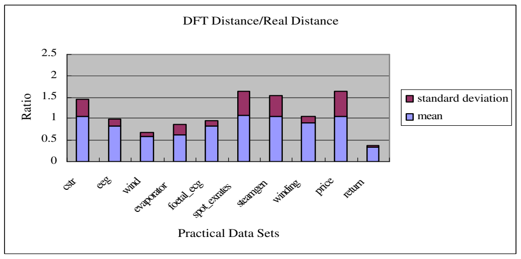
[2] An alternative, which avoids a parametric normality assumption, is to sort the calculated recalls over all the bootstraps and take the value that is x% from the lowest where x could be typically 1, 5 or 25. We have done this for our data (results not shown) and recalls of the 1% from lowest

Data title	$prec_{cd}^{mean}$	$prec_{cd}^{std}$	$prec_{ex}^{mean}$	$prec_{ex}^{std}$	Population Ratio 1	Population Ratio 2
spot_exrates	0.20	0.02	0.22	0.04	1.19	1.64
cstr	0.18	0.01	0.19	0.03	1.3	2.05
foetal_ecg	0.23	0.04	0.26	0.06	1.12	1.43
evaporator	0.007	0.0001	0.007	0.0001	15.81	102.26
steamgen	0.35	0.01	0.36	0.02	1.13	1.44
wind	0.002	0.001	0.002	0.001	14.8	564.87
winding	0.07	0.02	0.07	0.02	1.21	2.19
eeg	0.15	0.04	0.15	0.03	1.18	1.74
price	0.13	0.02	0.15	0.02	1.32	2.18
return	0.008	0.001	0.009	0.001	18.1	117.75

Table 3: Combinatorial design then refinement vs. exhaustive search over the dense parameter space. Population ratios indicate the number of pairs at 1.1 and 1.2 times the target distance divided by the number at the desired distance.



(a) Sketch



(b) DFT

Figure 4: DFT distance versus sketch distance over empirical data

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

7. EXPERIMENTS

Our approach has many moving 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.

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.

Fortunately, we have several data sets from stock market data and from the UC Riverside repository [20] that afford us an empirical test of the method.[3]

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

7.1 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.

7.2 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. Table 4 shows that for disjoint samples, the recall still meets the 99% threshold and the precision is as good or often higher than in the bootstrapping experiments on the test sample (table 2). So, using the parameters derived from a training sample (and confirmed by using bootstrapping) of a data set works well across that entire data set.

7.3 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

[3]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 [1]. All the other empirical data sets came from the UCR Time Series Data Mining Archive [20] maintained by Eamonn Keogh.

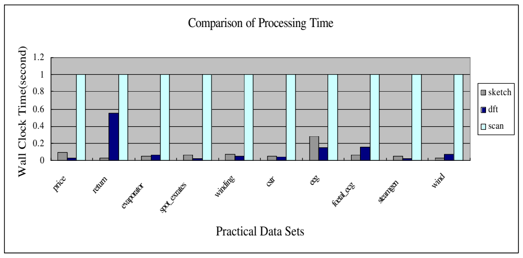


Figure 5: System performance over a variety of datasets.

Data title	rec^{mean}	rec^{std}	$prec^{mean}$	$prec^{std}$
spot_exrates	0.99	0.01	0.25	0.02
cstr	0.99	0.003	0.2	0.01
foetal_ecg	0.99	0.001	0.26	0.02
evaporator	1	0	0.007	0.0003
steamgen	1	0	0.33	0.01
wind	1	0	0.002	0.001
winding	0.99	0.007	0.1	0.01
eeg	1	0	0.13	0.02
price	0.99	0.001	0.18	0.02
return	1	0	0.008	0.002

Table 4: Recall and Precision of disjoint sample sets

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 unlike the Fourier-based one.

7.4 Stability of Parameter Settings Across Applications

Our general approach is to find the optimal parameter settings for each data set by sampling and bootstrapping. An interesting question is how similar these parameter settings are. If very similar, there might be a single good default to choose. This would eliminate the need for the sample-

Data title	rec^{mean}	rec^{std}	$prec^{mean}$	$prec^{std}$
spot_exrates	0.98	0.007	0.47	0.06
cstr	0.99	0.01	0.26	0.02
foetal_ecg	1	0	0.26	0.06
evaporator	1	0	0.007	0.0001
steamgen	0.98	0.006	0.61	0.04
wind	0.98	0.007	0.05	0.005
winding	0.98	0.06	0.13	0.04
eeg	0.99	0.006	0.14	0.001
price	0.99	0.01	0.13	0.03
return	0.98	0.01	0.008	0.001

Table 5: The recall and precision of empirical data sets with the optimal parameters for price data. Recall decreases as compared with choosing the optimal parameters for each data set on its own.

bootstrap step of our framework. In the following experiment, we took the best parameters for stock market prices and applied them to other data sets. The following table 5 shows the recall and precision obtained. Recall is better for the optimized-per-data-set parameters compared with the single-parameter-for-all approach. So seeking a set of optimized parameters for each data set is a better idea.

8. THEORETICAL PROBABILISTIC GUARANTEES FOR RECALL

Whereas any approximation technique can give poor precision if most distances are near the target distance, recall can be bounded using Chernoff and Chebyshev bounds. The analysis gives a bound on the probability of a false dismissal, i.e., the sketch estimate of the distance between a pair is larger than the real distance between the pair. Intuitvely, Chebyshev's inequality gives a probability of false dismissal for a group. Chernoff bounds show us how to combine the sometimes erroneous returns from the groups to get a less erroneous final result.

For concreteness, we assume that we want a recall of 99% and no more than 200 sketches. (In our experiments, we never exceeded 70.) Here is the analysis.

Let v be a length 1 vector whose length is being approximated (So $\|v\|$ is the L2 distance between two time series windows in our case.). We obtain a bound on the probablity that $\|v\|$ will be overestimated.

Consider a group of $\{r_1, r_2, \cdots, r_g\}$ of g random vectors.

Lemma 1 $E[\|(v * r_1, v * r_2, \cdots, v * r_g)\|^2] = g$.

Lemma 2 Let $f(v) = \|(v * r_1, v * r_2, \cdots, v * r_g)\|^2$ Then the variance $E[(f(v) - E[f(v)])^2] \leq 2g$.

Chebyshev's inquality states that $Pr[\|X - E[X]\| \geq t\sigma_X] \leq 1/(t^2)$, where σ_X is the square root of the variance.

We apply this with $X = f(v)$ and $t = 2$ say (we set several constants in this analysis to reasonable but ultimately arbitrary values; this is the first).

Then $Pr[X \geq E[X] + 2\sigma_X] \leq 1/4$. Substituting g for $E[X]$ and $2g$ for the square of σ_X, we obtain $Pr[X \geq g(1 + 2\sqrt{2}/\sqrt{g})] \leq 1/4$.

Setting the group size g to 4 (arbitrary setting again), we get a c of $1 + \sqrt{2}$ or about 2.4.

Let the third arbitrary setting be that f, the fraction of groups that must report that the pair is within the threshold 2.4 g, is 1/2. Now we look at the influence of having multiple groups.

Lemma (Chernoff) Let $Y_1, Y_2, \cdots, Y_k$ be independent random trials with $Pr[Y_i = 1] = p_i$ and $Pr[Y_i = 0] = 1 - p_i$. Let $Y = \sum_{i=1}^{k} Y_i$ and let $\mu = E[Y] = \sum_{i=1}^{k} p_i$.
Then, for any $\delta > 0$, $Pr[Y > (1+\delta)\mu] < ((e^{\delta})/((1+\delta)^{1+\delta}))^{\mu}$.

For our problem, we let X_i be the estimated length for v in the ith group and Y_i is defined by

$$Y_i = \begin{cases} +1 & \text{if } X_i \geq c \times g; \\ 0 & \text{otherwise} \end{cases}$$

When $Y_i = 1$, a group has overestimated the distance. The Chernoff bounds allow us to bound how likely it is that this will happen for $(1 - f)$ of the k groups, where k will be chosen to achieve a recall of 99%. Because we had set t to 2 above, each group will be wrong with probability $p_i \leq 1/4$. The expected value of the sum is therefore $k/4$. Because we have set f to be 1/2, a bad case would arise when $Y > k/2$ (that is more than half the groups declare the distance to be too high). Therefore $\delta = 1$. So
$Pr[Y > k/2] < ((e^{\delta})/((1 + \delta)^{1+\delta}))^{\mu} = (e/4)^{k/4}$.

This false dismissal error probability is less than 1% (equivalently, recall $\geq$ 99%) only when $(k/4) \geq 12$, so the number of groups has to be 48. Because each group contains four sketches, this gives a total of $48 \times 4 = 192$ random vectors, more than three times what we need in practice.

Notice that there are other ways to get a recall $\geq$ 99%. For example if f is 1/4, $\delta = 2$ (more than 3/4 of the groups must be wrong). Therefore $Pr[Y > 3k/4] < (e^2/27)^{k/4}$. This requires only fifteen groups or 60 random vectors.

Of course there are many other possibilities given $c = 2.4$ and $g = 4$. One interesting compromise is to set $f = 0.35$. In this case, the theoretical analysis indicates that the number of groups is 21 (84 sketches).

The empirical study indicates that this theoretical analysis is conservative for our data sets. In our empirical data sets, with these settings a recall of 99% could be achieved with 16 or fewer sketches (4 groups).

9. SUMMARY AND FUTURE WORK

Correlation is an indication 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 [4] and mutual information. Our plan is to use Pearson correlation and the techniques developed here as a rapid filter to see which pairs of time series windows are likely to have any given such co-trending measure.

10. ACKNOWLEDGMENT

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

11. REFERENCES

[1] Wharton research data services(wrds). http://wrds.wharton.upenn.edu/.

[2] D. Achlioptas. Database-friendly random projections. Santa Barbara, CA, May 2001. ACM SIGMOD-PODS.

[3] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity searching in sequence databases. In *Proceedings of the 4th International Conference of Foundations of Data organization and Algorithms (FODO)*, pages 69–84, Chicago, Illinois, MN, 1993. Springer Verlag.

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

[5] A. Braverman. *Personal Communication*. 2003.

[6] K.-P. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. ICDE, 1999.

[7] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton. The combinatorial design approach to automatic test generation. *IEEE Software*, 1996.

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

[9] E. Drinea, P. Drineas, and P. Huggins. A randomized singular value decomposition algorithm for image processing. Panhellenic Conference on Informatics (PCI), 2001.

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

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

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

[13] D. Q. Goldin and P. C. Kanellakis. On similarity queries for time-series data: constraint specification and implementation. The 1st Int'l Conference on the Principles and Practice of Constraint Programming, 1995.

[14] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. SIGMOD, 2001.

[15] P. Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, page 189. IEEE Computer Society, 2000.

[16] P. Indyk, N. Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. VLDB, 2000.

[17] W. Johnson and J. Lindenstrauss. Extensions of lipschitz mapping into hilbert space. *Contemporary Mathematics*, 26, 1984.

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

[19] E. Keogh, K. Chakrabarti, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. SIGMOD, 2001.

[20] E. Keogh and T. Folias. The ucr time series data mining archive. Riverside CA. University of California - Computer Science & Engineering Department, 2002. http://www.cs.ucr.edu/∼eamonn/TSDMA/index.html.

[21] E. Keogh, M. P. K. Chakrabarti, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 2000.

[22] F. Korn, H. Jagadish, and C. Faloutsos. Efficiently supporting ad hoc queries in large datasets of time sequences. SIGMOD, 1997.

[23] E. Kushikvitz, R. Ostrovsky, and Y. Ranbani. Efficient search for approximate nearest neighbors in high dimensional spaces. STOC, 1998.

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

[25] G. Manku, S. Rajagopalan, and B. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. SIGMOD, 1999.

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

[27] I. Popivanov and R. Miller. Similarity search over time series data using wavelets. ICDE, 2002.

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

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

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

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

[32] 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.

[33] B.-K. Yi and C. Faloutsos. Fast time sequence indexing for arbitrary lp forms. VLDB, 2000.

[34] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. Hong Kong, China, August 2002. VLDB.

12. APPENDIX: USING STRUCTURED RANDOM VECTORS TO COMPUTE SLIDING SKETCHES

Given a sliding window length sw and a basic window length bw, we show how to compute the sketches for the sliding windows starting at each timepoint. Naively, this can be done by computing the inner product of the length sw window ending at each data point with each of the random vetors. This involves much redundant computation, however, so is extremely inefficient. Instead, we will use a far more efficient approach based on the convolution between a "structured random vector" and a data vector of length sw.

Recall that sliding windows consist of an integral number of basic windows, so $sw = nb{*}bw$, where $nb = \frac{bw}{sw}$ is the number of basic windows within a sliding window. A random vector $\mathbf{r}_{bw}$ is constructed as follows.

$$\mathbf{r}_{bw} = (r_0, r_1, \cdots, r_{bw})$$

where

$$r_i = \begin{cases} +1 & \text{with probability } \frac{1}{2}; \\ -1 & \text{with probability } \frac{1}{2}. \end{cases}$$

To form a random vector of length sw, another random vector $\mathbf{b}$ is constructed of length nb $(= sw/bw)$. We call that second vector the *control vector*.

That is,

$$\mathbf{b} = (b_0, b_1, \cdots, b_{nb})$$

where

$$b_i = \begin{cases} +1 & \text{with probability } \frac{1}{2}; \\ -1 & \text{with probability } \frac{1}{2}. \end{cases}$$

The random vector $\mathbf{r}$ for a sliding window is then built as follows.

$$\mathbf{r} = (\mathbf{r}_{bw} \cdot b_0, \mathbf{r}_{bw} \cdot b_1, \cdots, \mathbf{r}_{bw} \cdot b_{nb})$$

Please note that the fully random unit here is of size bw and the structured random vector is made up of a concatenation of nb instances of the random unit or its negation. This reduction of randomness does not noticeably diminish the accuracy of the sketch estimation as we showed in the body of the paper.

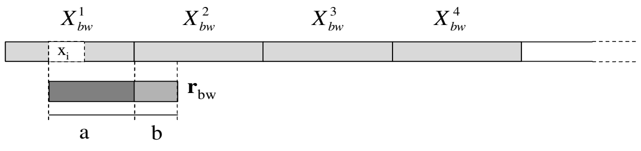
The choice of sw and bw depends on the application under consideration, because bw is the delay before results are reported. For instance, a trader may ask which pairs of stock returns have been correlated with a value of over 0.9 for the last three hours and want the correlation information reported every 30 seconds. Here the sliding window size is 3 hours and the basic window size 30 seconds.

Example:

We are given a time series $X = (x_1, x_2, \cdots)$, sliding window size $sw = 12$, and a basic window size $bw = 4$.

If the random vector within a basic window is $\mathbf{r}_{bw} = (1, 1, -1, 1)$, the control vector $\mathbf{b} = (1, -1, 1)$, the random vector for a sliding window will be $\mathbf{r}_{sw} = (1, 1, -1, 1, -1, -1, 1, -1, 1, 1, -1, 1)$.

End of example



Compute:
1. The partial dot product of $\mathbf{r}_{bw}$ with X over interval **a**
2. The partial dot product of $\mathbf{r}_{bw}$ with X over interval **b**

Figure 6: Dot products with two basic windows

To obtain the sketch, we will not compute the inner product between the random vector $\mathbf{r}_{sw}$ and a given data sliding window. Instead we use a "structured convolution". To see why this might help, let's continue the example from above.

Example

$X_{sw}^1 = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12})$

$X_{sw}^5 = (x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}, x_{12}, x_{13}, x_{14}, x_{15}, x_{16})$

A term in the sketch vector for the length sw window beginning with x_1 is given by:

$X_{sk}^1 = \mathbf{r}_{sw} \cdot X_{sw}^1 = b_1 \cdot \mathbf{r}_{bw} \cdot (x_1, x_2, x_3, x_4) + b_2 \cdot \mathbf{r}_{bw} \cdot (x_5, x_6, x_7, x_8) + b_3 \cdot \mathbf{r}_{bw} \cdot (x_9, x_{10}, x_{11}, x_{12})$

$= \mathbf{b} \cdot (\mathbf{r}_{bw} \cdot (x_1, x_2, x_3, x_4), \mathbf{r}_{bw} \cdot (x_5, x_6, x_7, x_8), \mathbf{r}_{bw} \cdot (x_9, x_{10}, x_{11}, x_{12}))$

The analogous term for the window beginning at x_5 is given by:

$X_{sk}^5 = \mathbf{r}_{sw} \cdot X_{sw}^5 = b_1 \cdot \mathbf{r}_{bw} \cdot (x_5, x_6, x_7, x_8) + b_2 \cdot \mathbf{r}_{bw} \cdot (x_9, x_{10}, x_{11}, x_{12}) + b_3 \cdot \mathbf{r}_{bw} \cdot (x_{13}, x_{14}, x_{15}, x_{16})$

$= \mathbf{b} \cdot (\mathbf{r}_{bw} \cdot (x_5, x_6, x_7, x_8), \mathbf{r}_{bw} \cdot (x_9, x_{10}, x_{11}, x_{12}), \mathbf{r}_{bw} \cdot (x_{13}, x_{14}, x_{15}, x_{16}))$

The computation of $\mathbf{r}_{bw} \cdot (x_5, x_6, x_7, x_8)$ occurs many times. That repeated computation is eliminated by the structured convolution.

End of Example

As we have seen, we want to take the dot product of $\mathbf{r}_{bw}$ with each length bw window of X. For efficiency, we will be computing convolutions with successive disjoint windows of X of length bw. This yield dot products of initial and final portions of $\mathbf{r}_{bw}$ with corresponding pieces of these disjoint windows, which we call partial dot products. Appropriate pairs of partial dot products then need to be added together to construct a sketch.

Figure 6 illustrates the partial dot products computed over basic window X_{bw}^1 and X_{bw}^2. In generaly $nb + 1$ such computations are needed to form a dot product of length sw.

Figure 7 shows the convolution of a basic window of data

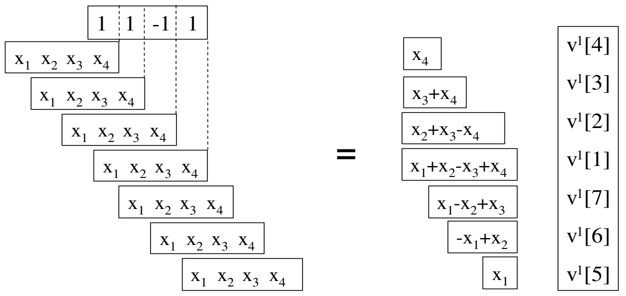


Figure 7: Structured Convolution

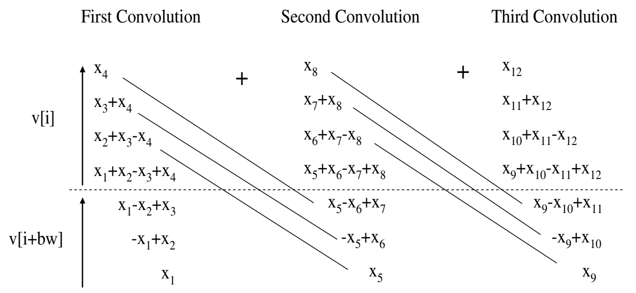


Figure 8: Sum up the corresponding pairs

Given time series $X = (x_1, x_2, \cdots, x_n)$ random vector $\mathbf{r}_{bw} = (r_1, r_2, \cdots, r_{bw})$, and control vector $\mathbf{b} = (b_1, b_2, \cdots, b_{nb})$, proceeds as follows:

1. Convolve $\mathbf{r}_{bw}$ with $(x_1, x_2, ..., x_{bw})$ without wraparound. This yields a vector $\mathbf{v^1}$ of length $2 * bw - 1$:

$$\begin{cases} v^1[i] = \sum_{j=i}^{bw} x_j \cdot r_{j-i+1} & 1 \le i \le bw; \\ v^1[i+bw] = \sum_{j=1}^{i-1} x_j \cdot r_{bw+j-i+1} & 2 \le i \le bw. \end{cases}$$

2. Repeat Step 1 nb times over data chunks of length bw starting from locations $bw+1, \cdots, nb*bw+1$ to obtain $v^{bw+1}, v^{2bw+1}, \cdots, v^{nb*bw+1}$
3. Compute

$$u^{h*bw+1}[i] = v^{h*bw+1}[i] + v^{(h+1)bw+1}[i+bw]$$

for $1 \le i \le bw$ and $h \ge 0$

Then

$$u^{h*bw+1}[i] = (x_{h*bw+i} \cdots x_{(h+1)bw+i-1}) \cdot (r_1, \cdots, r_{bw})$$

4. Compute the inner product between control vector $\mathbf{b}$ and each of

$$(u^{h*bw+1}[i], u^{(h+1)*bw+1}[i], \cdots, u^{(h+nb-1)*bw+1}[i])$$

for $1 \le i \le bw$ and $h \ge 0$

Figure 9: Structured Convolution Procedure

with the random vector $\mathbf{r}_{bw}$.

Figure 8 shows the addition of corresponding pairs of partial dot products. Each slanted line represents a sum, e.g. $x_4 + (x_5 - x_6 + x_7)$.

The final step is to compute the dot product with $\mathbf{b}$ of the results of the dot products with $\mathbf{r}_{bw}$.

The full procedure is shown in Figure 9.

12.1 Algorithmic Analysis

The structured random vector procedure eliminates the redundant computations inherent in doing a full dot product for the window of length sw starting at every timepoint. Each basic window of data is convolved with a random vector basic window only once, which will eliminate the redundancy existing in the direct convolution.

Here is the performance analysis.

1. Naive algorithm:

   For each random vector and each timepoint, the dot product requires $O(sw)$ integer additions.
2. New algorithm:

   The structured random vector algorithm divides the cost into two parts.

   (a) $O(sw/bw)$ integer additions;

   (b) $O(\log\ bw)$ floating point operations (using the Fast Fourier Transform in computing the convolution)