Efficient Elastic Burst Detection in Data Streams *

Yunyue Zhu
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
yunyue@cs.nyu.edu

Dennis Shasha
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
shasha@cs.nyu.edu

ABSTRACT

Burst detection is the activity of finding abnormal aggregates in data streams. Such aggregates are based on sliding windows over data streams. In some applications, we want to monitor many sliding window sizes simultaneously and to report those windows with aggregates significantly different from other periods. We will present a general data structure for detecting interesting aggregates over such elastic windows in near linear time. We present applications of the algorithm for detecting Gamma Ray bursts in large-scale astrophysical data. Detection of periods with high volumes of trading activities and high stock price volatility is also demonstrated using real time Trade and Quote (TAQ) data from the New York Stock Exchange (NYSE). Our algorithm beats the direct computation approach by several orders of magnitude.

1. INTRODUCTION

Consider the following application that motivates this research. An astronomical telescope, Milagro[1] was built in New Mexico by a group of prestigious astrophysicists from the Los Alamos National Laboratory and many universities. This telescope is actually an array of light-sensitive detectors covering a large pool of water about the size of a football field. It is used to constantly observe high-energy photons from the universe. When many photons observed, the scientists assert the existence of a Gamma Ray Burst. The scientists hope to discover primordial black holes or completely new phenomena by the detection of Gamma Ray Bursts. The occurrences of Gamma Ray Bursts are highly variable, flaring on timescale of minutes to days. Once such a burst happens, it should be reported immediately. Other telescopes could then point to that portion of sky to confirm the new astrophysical event. The data rate of the observation is extremely high. Hundreds of photons can be recorded within a second from a tiny spot in the sky[8, 13].

*Work supported in part by U.S. NSF grants IIS-9988636 and N2010-0115586.

There are also many applications in data stream mining and monitoring when people are interested in discovering time intervals with unusually high numbers of events. For example:

- In telecommunication, a network anomaly might be indicated if the number of packages lost within a certain time period exceeds some threshold.
- In finance, stocks with unusually high trading volumes would attract the notice of the traders (or regulators). Also stocks with unusually high price fluctuations within a short time period provide more opportunity of speculation. Therefore they would be watched more closely.

Intuitively, given an aggregate function F (such as sum or count), the problem of interest is to discover subsequences s of a time series stream such that $F(s) >> F(s')$ for most subsequences s' of size $|s|$. In the case of burst detection, the aggregate is sum. If we know the duration of the time interval, we can maintain the sum over sliding windows of a known window size and sound an alarm when the moving sum is above a threshold. Unfortunately, in many cases, we cannot predict the length of the burst period. In fact, discovering that length is part of the problem to be solved. In the above example of Gamma Ray Burst detection, a burst of photons associated with a special event might last for a few milliseconds, a few hours, or even a few days. There are different thresholds associated with different durations. A burst of 10 events within 1 second could be very interesting. At the same time, a burst that lasts longer but with lesser density of events, say 50 events within 10 seconds, could be of interest too.

Suppose that we want to detect bursts for a time series of size n and we are interested in all the n sliding window sizes. A brute-force search has to examine all the sliding window sizes and starting positions. Because there are $O(n^2)$ windows, the lower bound of the time complexity is $O(n^2)$. This is very slow for many applications. Fortunately, because we are interested only in those few windows that experience bursts, it is possible to design a nearly linear time algorithm. In this paper we present a burst detection algorithm with time complexity approximately proportional to the size of the input plus the size of the output, i.e. the number of windows with bursts.

1.1 Problem Statement

There are two categories of time series data stream monitoring: *point monitoring* and *aggregate monitoring*. In point monitoring, the latest data item in the stream is of interest. When the latest item falls in some predefined domain, an alarm would be sounded. For example, a stock trader who places a limited sell order on Enron informs the stock price stream monitor to raise an alarm (or automatically sell the stock) once the price of stock fall under $10 to avoid further losses. Since only the latest data in the stream need to be considered, point monitoring can be implemented without much effort.

Aggregate monitoring is much more challenging. Aggregates of time series are computed based on certain time intervals (windows). There are three well-known window models that are the subjects of many research projects [9, 10, 11, 21].

1. **Landmark windows:** Aggregates are computed based on the values between a specific time point called the landmark and the present. For example, the average stock price of IBM from Jan 1st, 2002 to today is based on a landmark window.
2. **Sliding windows:** In a sliding window model, aggregates are computed based on the last w values in the data stream. The size of a sliding window w is predefined. For example, the running maximum stock price of IBM during the previous 5 days is based on sliding windows.
3. **Damped window:** In a damped window model the weights of data decrease exponentially into the past. For example, the damping moving average avg_{new} after a new data item x is inserted can be updated as follows:

$$avg_{new} = avg_{old} * p + x * (1 - p), 0 < p < 1$$

The sliding window model is the most widely used in data stream monitoring. Motivated by the Gamma Ray example, we have generalized this to the *elastic window model*. In an elastic window model, the user needs to specify only the range of the sliding window sizes, the aggregate function and alarms domains, and will be notified of all those window sizes in the range with aggregates falling in the corresponding alarm domains.

Here we give the formal definition of the problem of monitoring data stream over elastic windows.

PROBLEM 1. *For a time series $x_1, x_2, ..., x_n$, given a set of window sizes $w_1, w_2, ..., w_m$, an aggregate function F and threshold associated with each window size, $f(w_j), j = 1, 2, ..., m$, monitoring elastics window aggregates of the time series is to find all the subsequences of all the window sizes such that the aggregate applied to the subsequences cross their window sizes' thresholds, i.e.*

$$\forall i \in 1..n, \forall j \in 1..m, s.t.\ F(x[i\ ..\ i + w_j - 1]) \geq f(w_j)$$

The threshold above can be estimated from the historical data or the model of the time series. Elastic burst detection is a special case of monitoring data streams on elastic windows. In elastic burst detection, the alarm domain is $[f(w_j), \infty)$. Note that it is also possible for the alarm domain to be $(-\infty, f(w_j)]$.

1.2 Our Contributions

The contributions of the paper are as follows.

- We introduce the concept of monitoring data streams on elastic windows and show several important applications of this model.
- We design an innovated data structure, called the Shifted Wavelet Tree, for efficient elastic burst monitoring. This data structure is applied to general aggregate monitoring and burst detection in higher dimensions.
- We apply our algorithm to real world data including the Milagro Gamma Ray data stream, NYSE real time tick data and text data. Our method is up to several magnitudes faster than a direct computation, which means that a multi-day computation can be done in a matter of minutes.

2. DATA STRUCTURE AND ALGORITHM

In this section, we first give some background on the wavelet data structure. In section 2.2 we discuss the Shifted Wavelet Tree and the algorithm for efficient elastic burst detection in an offline setting. This is extended to a streaming algorithm in section 2.3. Our algorithm will also be generalized to other problems in data stream monitoring over elastic windows in section 2.4 and to higher dimensions in section 2.5.

2.1 Wavelet Data Structure

In wavelet analysis, the wavelet coefficients of a time series are maintained in a hierarchical structure. Let us consider the simplest wavelet, the Haar wavelet. For simplicity of notation, suppose that the size of time series n is a power of two. This would not affect the generality of the results. The original time series makes up of level 0 in a wavelet tree. The pair wise (normalized) averages and differences of the adjacent data items at level 0 produce the wavelet coefficients at level 1. The process is repeated for the averages at level i to get the averages and differences at level $i + 1$ until there is only one average and difference at the top level. Table 1 shows the process in computing the Haar wavelet decomposition. The Haar wavelet coefficients include the average in the highest level and the differences in each level. From these wavelet coefficients, the original time series can be constructed without loss of information. Usually a few wavelet coefficients can represent the trend of the time series, and they are selected as a compact representation of the original time series.

The wavelet coefficients above can also be viewed as the aggregates of the time series at different time intervals. Figure 1-a shows the time interval hierarchy in the Haar wavelet decomposition. At level i, there are $n2^{-i}$ consecutive windows with size 2^i. All the windows at the same level are disjoint. The aggregates that the Haar wavelet maintains are the (normalized) averages and differences. In our discussion of burst detection, the aggregate of interest is the

Table 1: Haar Wavelet decomposition

Level 3	$\frac{a_1+a_2+a_3+a_4+a_5+a_6+a_7+a_8}{2\sqrt{2}}$				$\frac{(a_1+a_2+a_3+a_4)-(a_5+a_6+a_7+a_8)}{2\sqrt{2}}$			
Level 2	$\frac{a_1+a_2+a_3+a_4}{2}$		$\frac{a_5+a_6+a_7+a_8}{2}$		$\frac{(a_1+a_2)-(a_3+a_4)}{2}$		$\frac{(a_5+a_6)-(a_7+a_8)}{2}$	
Level 1	$\frac{a_1+a_2}{\sqrt{2}}$	$\frac{a_3+a_4}{\sqrt{2}}$	$\frac{a_5+a_6}{\sqrt{2}}$	$\frac{a_7+a_8}{\sqrt{2}}$	$\frac{a_1-a_2}{\sqrt{2}}$	$\frac{a_3-a_4}{\sqrt{2}}$	$\frac{a_5-a_6}{\sqrt{2}}$	$\frac{a_7-a_8}{\sqrt{2}}$
Level 0	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8

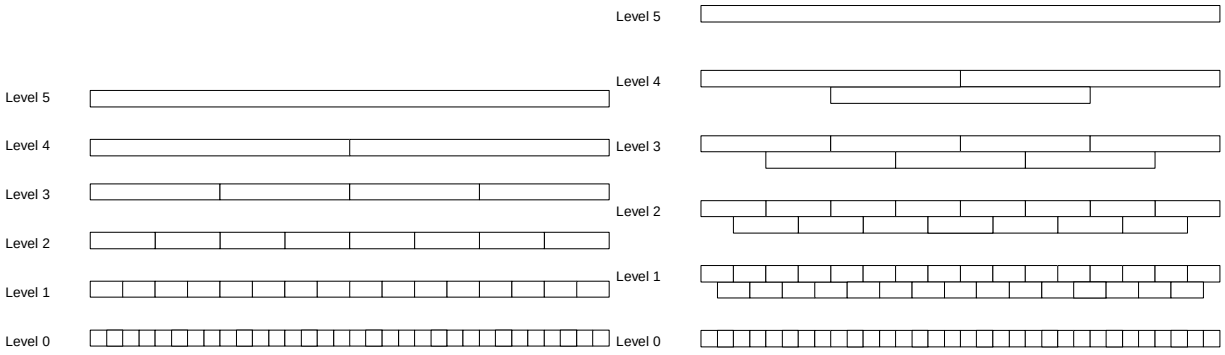

Figure 1: (a)Wavelet Tree (left) and (b)Shifted Wavelet Tree(right)

sum. Obviously, such a wavelet tree can be constructed in $O(n)$ time.

The first few top levels of a wavelet tree yield concise multi-resolution information of the time series. This gives the wavelet lots of applications. However, for our purpose of burst detection, such a data structure has a serious problem. Because the windows at the same level are non-overlapping, a window of arbitrary start position and arbitrary size might not be included in any window in the wavelet tree, except the window at the highest level that includes everything. For example, the window consisting of three time points in the middle, $(n/2-1, n/2+1, n/2+2)$, is not contained in any window in the wavelet tree except the largest one. This makes wavelets inconvenient for the discovery of properties of arbitrary subsequences.

2.2 Shifted Wavelet Tree

In a *shifted wavelet tree* (SWT) (in figure 1-b), the adjacent windows of the same level are half overlapping. In figure 1, we can see that the size of a SWT is approximately double that of a wavelet tree, because at each level, there is an additional "line" of windows. These additional windows provide valuable overlapping information for the time series. They can be maintained either explicitly or implicitly. If we keep only the aggregates for a traditional wavelet data structure, the aggregates of the overlapping windows at level i can be computed from the aggregates at level $i-1$ of the wavelet data structure.

To build a SWT, we start from the original time series and compute the pair wise aggregate (sum) for each two consecutive data items. This will produce the aggregates at level 1. A downsampling on this level will produce the input for the higher level in the SWT. Downsampling is simply sampling every second item in the series of aggregates. In figure 1-b, downsampling will retain the upper consecutive non-overlapping windows in each level. This process is repeated

```
Given :  x[1..n],n=2^a
Return:  shifted wavelet tree SWT[1..a][1..]

b=x;
FOR i = 1 TO a //remember a = log2 n
  //merge consecutive windows and form
  //level i of the shifted wavelet tree
  FOR j = 1 TO size(b)-1 STEP 2
    SWT[i][j]=b[j]+b[j+1];
  ENDFOR
  //downsample, retain a non-overlapping cover
  FOR j = 1 TO size(SWT[i])/2
    b[j]=SWT[i][2*j-1];
  ENDFOR
ENDFOR
```

Figure 2: Algorithm to construct shifted wavelet tree

until we reach the level where a single window includes every data point. Figure 2 gives a pseudo-code to build a SWT. Like regular wavelet trees, SWT can also be constructed in $O(n)$ time.

For a subsequence starting and ending at arbitrary positions, there is always a window in the SWT that tightly includes the subsequence as figure 3 shows and the following lemma proves.

LEMMA 1. *Given a time series of length n and its shifted wavelet tree, any subsequence of length $w, w \leq 2^i$ is included in one of the windows at level $i+1$ of the shifted wavelet tree.*

PROOF. The windows at level $i+1$ of the shifted wavelet

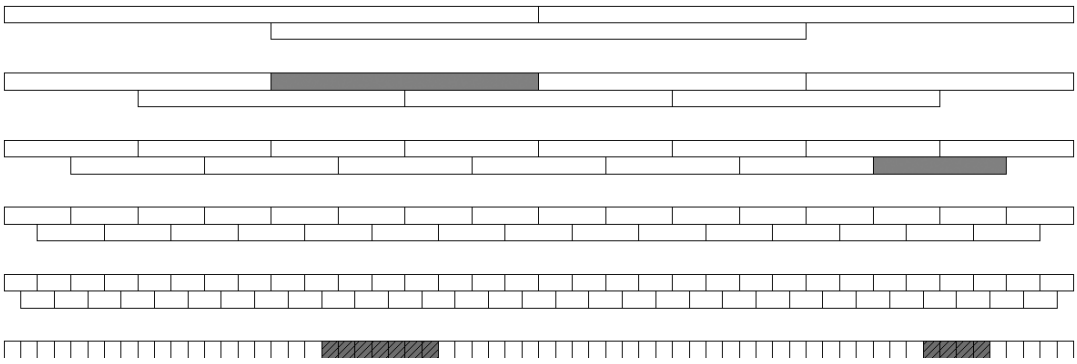

Figure 3: Examples of the windows that include subsequences in the shifted wavelet tree

tree are:

$$[(j-1)2^i+1 \,..\, (j+1)2^i], j = 1, 2, ..., \frac{n}{2^i} - 1.$$

A subsequence with length 2^i starting from an arbitrary position c will be included in at least one of the above windows, because

$$[c \,..\, c+2^i-1] \subseteq [(j-1)2^i+1 \,..\, (j+1)2^i], j = \lfloor \frac{c}{2^i} \rfloor + 1.$$

Any subsequence with length $w, w \leq 2^i$ is included in some subsequence(s) with length 2^i, and therefore is included in one of the windows at level $i+1$. We say that windows with size $w, 2^{i-1} < w \leq 2^i$, are monitored by level $i+1$ of the SWT. □

Because for time series of non-negative numbers the aggregate sum is monotonically increasing, the sum of the time series within a sliding window of any size is bounded by the sum of its including window in the shifted wavelet tree. This fact can be used as a filter to eliminate those subsequences whose sums are far below their thresholds.

Figure 4 gives the pseudo-code for spotting potential subsequences of size $w, w \leq 2^i$, with sums above its threshold $f(w)$. The algorithm will search for burst in two stages. First, the potential burst is detected at the level $i+1$ in the SWT, which corresponds to the subsequence $x[(j-1)2^i + 1 \,..\, (j+1)2^i]$. In the second stage, those subsequence of size 2^i within $x[(j-1)2^i+1 \,..\, (j+1)2^i]$ with sum less than $f(w)$ are further eliminated. The moving sums of sliding window size 2^i can be reused for burst detection of other window size $w' \neq w, w' \leq 2^i$. The detail search of burst on the surviving subsequences is then performed. A detail search in a subsequence is to compute the moving sums with window size w in the subsequence and to verify if there are bursts there.

In the spirit of the original work of [2] that uses lower bound technique for fast time series similarity search, we have the following lemma that guarantees the correctness of our algorithm.

LEMMA 2. *The above algorithm can guarantee no false negatives in elastic burst detection from a time series of non-negative numbers.*

```
Given :  time series x[1..n], n = 2^a,
  shifted wavelet tree SWT[1..a][1..],
  window size w, threshold f(w)
Return:  Subsequence of x with burst

i = ⌈log2 w⌉;
FOR j = 1 TO size(SWT[i+1])
  IF (SWT[i+1][j]>f[w])
  //possible burst in subsequence x[(j−1)2^i+1 .. (j+1)2^i],
  //first we compute the moving sums with
  //window size 2^i within this subsequence.
    FOR c = (j − 1)2^i + 1 TO j2^i
      y=sum(x[c .. c−1+2^i]);
      IF y>f[w]
        detailed search in x[c .. c−1+2^i]
      ENDIF
    ENDFOR
  ENDIF
ENDFOR
```

Figure 4: Algorithm to search for burst

PROOF. From lemma 1, any subsequence of length $w, w \leq 2^i$ is contained within a window in the SWT:

$$[c \,..\, c+w-1] \subseteq [c \,..\, c+2^i-1] \subseteq [(j-1)2^i+1 \,..\, (j+1)2^i]$$

Because the sum of the time series of non-negative numbers is monotonic increasing, we have

$$\sum(x[c \,..\, c+w-1]) \leq \sum(x[c \,..\, c+2^i-1]) \leq \sum(x[(j-1)2^i+1 \,..\, (j+1)2^i]).$$

By eliminating sequences with lengths larger than w but with sums less than $f(w)$, we do not introduce false negatives because

$$\sum(x[(j-1)2^i+1 \,..\, (j+1)2^i]) < f(w) \Rightarrow \sum(x[c \,..\, c+w-1]) < f(w).$$

□

In most applications, the algorithm will perform detail search seldom and then usually only when there is a burst of interest. For example, suppose that the moving sum of a time series is a random variable from a normal distribution. Let

the sum within sliding window w be $S_o(w)$ and its expectation be $S_e(w)$, We assume that

$$\frac{S_o(w) - S_e(w)}{S_e(w)} \sim Norm(0,1).$$

We set the threshold of burst $f(w)$ for window size w such that the probability that the observed sums exceed the threshold is less that p, i.e., $Pr(S_o(w) \geq f(w)) \leq p$. Let $\Phi(x)$ be the normal cumulative distribution function, we have

$$f(w) = S_e(w)(1 - \Phi^{-1}(p)).$$

Because our algorithm monitors the burst based on windows with size $W = Tw, 1 \leq T < 2$, a real burst will always result in an alarm. However, it is possible that an alarm will be raised because there are more than $f(w)$ events in a window of W. We call this probability the false alarm rate p_f. Suppose that $S_e(W) = TS_e(w)$, we have

$$\frac{S_o(W) - S_e(W)}{S_e(W)} \sim Norm(0,1).$$

$$\begin{aligned} p_f &= Pr(S_o(W) \geq f(w)) = Pr\ \frac{S_o(W) - S_e(W)}{S_e(W)} \geq \frac{f(w) - S_e(W)}{S_e(W)} \\ &= \Phi\ -\frac{f(w) - S_e(W)}{S_e(W)}\ = \Phi\ 1 - \frac{f(w)}{TS_e(w)} \\ &= \Phi\ 1 - \frac{1 - \Phi^{-1}(p)}{T} \end{aligned}$$

The false alarm rate is very small for small p, the primary case of interest. For example, let $p = 10^{-6}, T = 1.5$, p_f is 0.006. In this model, the upper bound of false alarm rate is guaranteed.

The time for a detailed search in a subsequence of size W is $O(W)$. The total time for all detailed searches is linear in the number of false alarms and true alarms(the output size k). The number of false alarm depends on the data and the setting of thresholds, and it is approximately proportional to the output size k. So the total time for detail searches is bounded by $O(k)$. To build the SWT takes time $O(n)$, thus the total time complexity of our algorithm is approximately $O(n+k)$, which is linear in the total of the input and output size.

2.3 Streaming Algorithm

The SWT data structure in the previous section can also be used to support a streaming algorithm for elastic burst detection. Suppose that the set of window sizes in the elastic window model are $2^L < w_1 < w_2 < ... < w_m \leq 2^U$. For simplicity of explanation, assume that new data become available at every time unit.

Without the use of SWT, a naive implementation of elastic burst detection has to maintain the m sums over the sliding windows. When a new data item becomes available, for each sliding window, the new data item is added to the sum and the corresponding expiring data item of the sliding window is subtracted from the sum. The running sums are then checked against the monitoring thresholds. This takes time $O(m)$ for each insertion of new data. The response time is one time unit if enough computing resources are available.

By comparison, the streaming algorithms based on SWT data structure will be much more efficient. For the set of window size $2^L < w_1 < w_2 < ... < w_m \leq 2^U$, we need to maintained the levels from $L+2$ to $U+1$ of the SWT that monitor those windows. There are two methods that provide tradeoffs between throughput and response time.

- **Online Algorithm:**The online algorithm will have a response time of one time unit. In the SWT data structure, each data item is covered by two windows in each level. Whenever a new data item becomes available, we will update those $2(U-L)$ aggregates of the windows in the SWT immediately. Associated with each level, there is a minimum threshold. For level i, the minimum threshold δ_i is the min of the thresholds of all the windows monitored by level i, that is, $\delta_i = \min f(w_j), 2^{i-2} < w_j \leq 2^{i-1}$. If the sum in the last windows at level i exceeds δ_i, it is possible that the sum of one of the window sizes monitored by level i exceeds its threshold. We will perform a detailed search on those time intervals. Otherwise, the data stream monitor awaits insertions into the data stream. This online algorithm provides a response time of one time unit, and each insertion of the data stream requires $2(U-L)$ updates plus possible detailed searching.
- **Batch Algorithm:** The batch algorithm will be lazy in updating the SWT. Remember that the aggregates at level i can be computed from the aggregates at level $i-1$. If we maintain aggregates at an extra level of consecutive windows with size 2^{L+1}, the aggregates at levels from $L+2$ to $U+1$ can be computed in batch. The aggregate in the last window of the extra level is updated every time unit. An aggregate of a window at the upper levels in the SWT will not be computed until all the data in that window are available. Once an aggregate at a certain upper level is updated, we also check alarms for time intervals monitored by that level. A batch algorithm gives higher throughput though longer response time (with guaranteed bound of about the window size) than an online algorithm as the following lemmas state.

LEMMA 3. *The amortized processing time per insertion into the data stream for a batch algorithm is* 2.

PROOF. At the level $i, L+2 \leq i \leq U+1$, of the SWT, every 2^{i-1} time unit there is a window in which all the data are available. The aggregates at that window can be computed in time $O(1)$ based on the aggregates at level $i-1$. Therefore the amortized update time for level i is $\frac{1}{2^{i-1}}$. The total amortized update time for all levels (including the extra level) is

$$1 + \sum_{i=L+2}^{U+1} \frac{1}{2^{i-1}} \leq 2.$$

□

LEMMA 4. *The burst activity of a window with size w will be reported with a delay less than $2^{\lceil \log_2 w \rceil}$.*

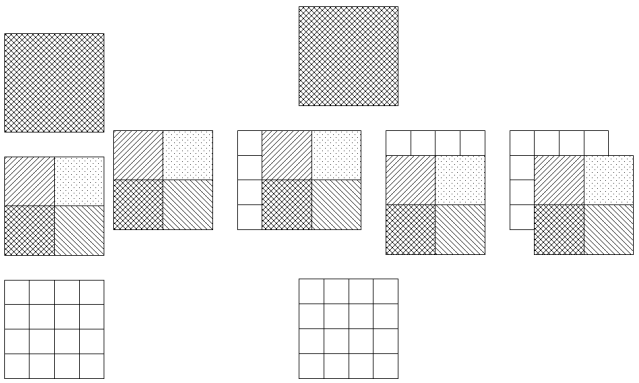

Figure 5: (a)Wavelet Tree (left) and (b)Shifted Wavelet Tree(right)

PROOF. A window with size $w, 2^{i-1} < w \leq 2^i$, is monitored by level $i+1$ of the SWT. The aggregates of windows at level $i+1$ are updated every 2^i time units. When the aggregates of windows at level $i+1$ are updated, the burst activity of window with size w can be checked. So the response time is less than $2^i = 2^{\lceil \log_2 w \rceil}$. □

2.4 Other Aggregates

It should be clear that in addition to sum, the monitoring of many other aggregates based on elastic window could benefit from our data structure, as long as the following holds.

1. The aggregate F is monotonically increasing or decreasing with respect to the window, i.e., if window $[a_1..b_1]$ is contained in window $[a_2..b_2]$, then $F(x[a_1..b_1]) \leq F(x[a_2..b_2])$ (or $F(x[a_1..b_1]) \geq F(x[a_2..b_2])$) always holds.
2. The alarm domain is one sided, that is, $[threshold, \infty)$ for monotonic increasing aggregates and $(-\infty, threshold]$ for monotonic decreasing aggregates.

The most important and widely used aggregates are all monotonic: *Max, Count* are monotonically increasing and *Min* is monotonically decreasing. Another monotonic aggregate is *Spread*. Spread measures the volatility or surprising level of time series. Spread of a time series $\vec{x}$ is

$$Spread(\vec{x}) = Max(\vec{x}) - Min(\vec{x}).$$

Spread is monotonically increasing. The spread within a small time interval is less than or equal to that within a larger time interval. A large spread within a small time interval is of interest in many applications in data stream because it indicates that the time series has experienced large movement.

2.5 Extension to Two Dimensions

The one-dimensional shifted wavelet tree for time series can naturally be extended to higher dimensions. In this section we consider the problem of discovering elastic spatial bursts using a two-dimensional shifted wavelet tree. Given an image of scattering dots, we want to find the regions of the image with unexpectedly high density. In an image of the sky with many dots representing the stars, such regions might indicate galaxies or supernovas. As for the case for elastic bursts in time series, the problem is to report the positions of (spatial) sliding windows having different sizes, within which the density exceeds some predefined threshold.

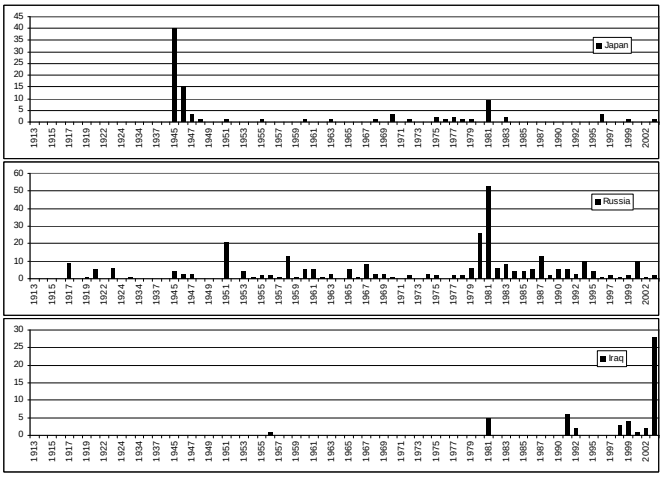

Figure 6: Bursts of the number of times that countries were mentioned in the presidential speech of the state of the union

The two-dimensional shifted wavelet tree is based on the two-dimensional wavelet structure. The basic wavelet structure separates a two-dimensional space into a hierarchy of windows as shown in figure 5-a. Aggregate information will be computed recursively based on those windows to get a compact representation of the data. Our two-dimensional shifted wavelet tree will extend the wavelet tree in a similar fashion as in the one-dimensional case. This is demonstrated in figure 5-b. At the same level of the wavelet tree, in addition to the group of disjoint windows that are the same as in the wavelet tree, there are another three groups of disjoint windows. One group of windows offsets the original group in the horizontal direction, one in the vertical direction and the third one in both directions.

Any square spatial sliding window with size $w \times w$ is included in one window of the two-dimensional SWT. The size of such a window is at most $2w \times 2w$. Using the techniques of section 2.2, burst detection based on SWT-2D can report all the high density regions efficiently.

3. EMPIRICAL RESULTS

Our empirical study will first demonstrate the desirability of elastic burst detection for some applications. We also study the performance of our algorithm by comparing our algorithm with the brute force searching algorithm in section 3.2.

3.1 Effectiveness Study

As an emotive example, we monitor bursts of interest in countries from the presidential of the State of the Union addresses from 1913 to 2003. In figure 6 we show the number of times that some countries were mentioned in the speeches.

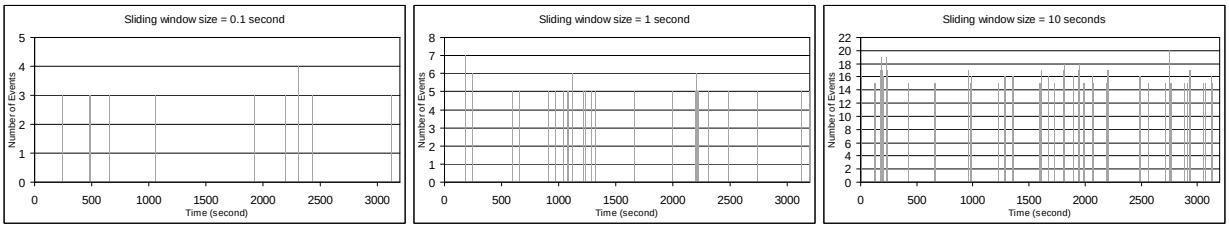

Figure 7: Bursts in Gamma Ray data for different sliding window size

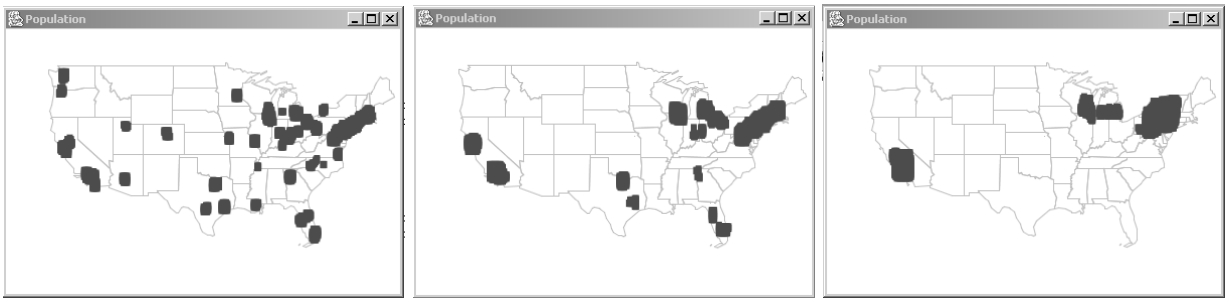

Figure 8: Bursts in population distribution data for different spatial sliding window size

There are clearly bursts of interest in certain countries. An interesting observation here is that these bursts have different durations, varying from years to decades.

The rationale behind elastic burst detection is that a predefined sliding window size for data stream aggregate monitoring is insufficient in many applications. The same data aggregated in different time scales will give very different pictures as we can see in figure 7. In figure 7 we show the moving sums of the number of events for about an hour's worth of Gamma Ray data. The sizes of sliding windows are 0.1, 1 and 10 seconds respectively. For better visualization, we only show those positions with bursts. Naturally, bursts at small time scales that are extremely high will produce bursts in larger time scales too. More interestingly, bursts at large time scales are not necessarily reflected at smaller time scales, because those bursts at large time scales might be composed of many consecutive "bumps". Bumps are those positions where the numbers of events are high but not high enough to be "bursts". Therefore, by looking at different time scales at the same time, elastic burst detection will give more insight into the data stream.

We also show in figure 8 an example of spatial elastic bursts. We use the 1990 census data of the population in the continental United State. The population in the map are aggregated in a grid of $0.2^\circ \times 0.2^\circ$ in Latitude/Longitude. We compute the total population within sliding spatial windows with sizes $1^\circ \times 1^\circ$, $2^\circ \times 2^\circ$ and $5^\circ \times 5^\circ$. Those regions with population above the 98 percentile in different scales are highlighted. We can see that the different sizes of sliding windows give the distribution of high population regions at different scales.

3.2 Performance Study

Our experiments were performed on a 1.5GHz Pentium 4 PC with 512 MB of main memory running Windows 2000. We tested the algorithm with two different types of data sets:

- The Gamma Ray data set : This data set includes 12 hours of data from a small region of the sky, where Gamma Ray bursts were actually reported during that time. The data are time series of the number of photons observed (events) every 0.1 second. There are totally 19,015 events in this time series of length 432,000.
- The NYSE TAQ Stock data set : This data set includes four years of tick-by-tick trading activities of the IBM stock between July 1st, 1998 and July 1st, 2002. There are 5,331,145 trading records (ticks) in total. Each record contains trading time (precise to the second), trading price and trading volume.

In the following experiments, we set the thresholds of different window sizes as follows. We use the first few hours of Gamma Ray data and the first year of Stock data as training data respectively. For a window of size w, we compute the aggregates on the training data with sliding window of size w. This gives another time series $\vec{y}$. The thresholds are set to be $f(w) = avg(\vec{y}) + \xi std(\vec{y})$, where $avg(\vec{y})$ and $std(\vec{y})$ are the average and standard deviation respectively. The factor of threshold ξ is set to 8. The list of window sizes is $5, 10, ..., 5 * N_w$ time units, where N_w is the number of windows. N_w varies from 5 to 50. The time unit is 0.1 seconds for the Gamma Ray data and 1 minute for the stock data.

First we compare the wall clock processing time of elastic burst detection from the Gamma Ray data in figure 9. Our algorithm based on the SWT data structure is more than ten

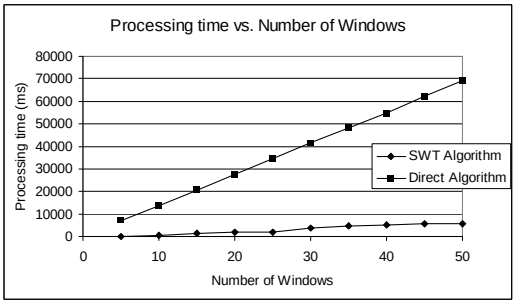

Figure 9: The processing time of elastic burst detection on Gamma Ray data for different number of windows

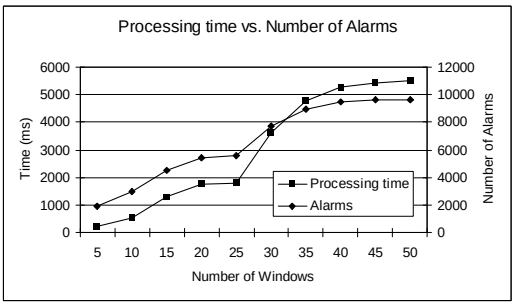

Figure 10: The processing time of elastic burst detection on Gamma Ray data for different output sizes

times faster than the direct algorithm. The advantage of using our data structure becomes more obvious as we examine more window sizes. The processing time of our algorithm is output-dependent. This is confirmed in figure 10, where we examine the relationship between the processing time using our algorithm and the number of alarms. Naturally the number of alarms increases as we examine more window sizes. We also observed that the processing time follows the number of alarms well. Recall that the processing time of the SWT algorithm includes two parts: building the SWT and the detailed searching of those potential portions of burst. Building the SWT takes only 200 milliseconds for the data set, which is negligible when compared to the time to do the detailed search. Also for demonstration purposes, we intentionally, to our disadvantage, set the thresholds lower and therefore got many more alarms than what physicists are interested in. If the alarms are scarce, as is the case for Gamma Ray burst detection, our algorithm will be even faster. In figure 11 we fix the number of windows to be 25 and change the factor of threshold ξ. The larger ξ is, the higher the thresholds are, and therefore the fewer alarms will be sounded. Because our algorithm is dependent on the output sizes, the higher the thresholds are, the faster the algorithm runs. In contrast, the processing time of the direct algorithm does not change accordingly.

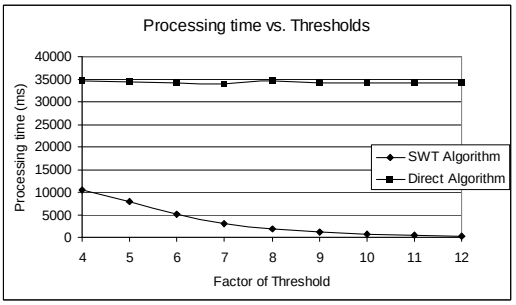

Figure 11: The processing time of elastic burst detection on Gamma Ray data for different thresholds

For the next experiments, we test the elastic burst detection algorithm on the IBM Stock trading volume data. Figure 12 shows that our algorithm is up to 100 times faster than a brute force method. We also zoom in to show the processing time for different output sizes in figure 13.

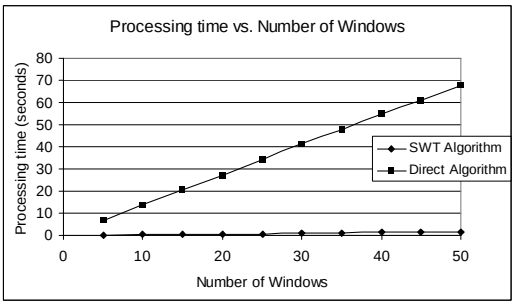

Figure 12: The processing time of elastic burst detection on Stock data for different number of windows

In addition to elastic burst detection, our SWT data structure works for other elastic aggregates monitoring too. In the following experiments, we search for big spreads on the IBM Stock data. Figure 14 and 15 confirms the performance advantages of our algorithm. Note that for the aggregates of Min and Max, and thus Spread, there is no known deterministic algorithm to update the aggregates over sliding windows incrementally in constant time. The filtering property of SWT data structure gains more by avoiding unnecessary detailed searching. So in this case our algorithm is up to 1,000 times faster than the direct method, reflecting the advantage of a near linear algorithm as compared with a quadratic one.

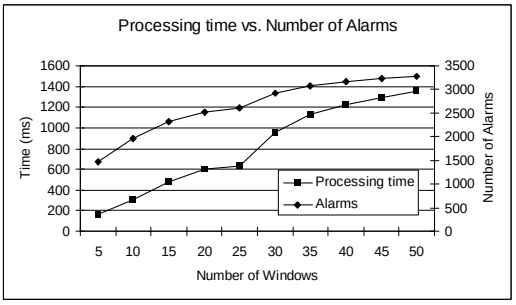

Figure 13: The processing time of elastic burst detection on Stock data for different output sizes

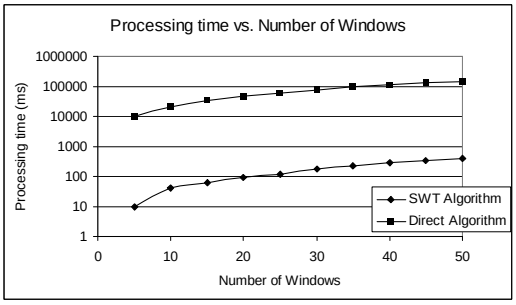

Figure 14: The processing time of elastic spread detection on Stock data for different number of windows

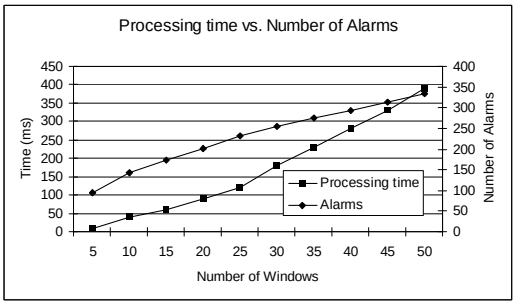

Figure 15: The processing time of elastic spread detection on Stock data for different output sizes

4. RELATED WORK

There is much recent interest in data stream mining and monitoring. An excellent survey of models and issues in data stream can be found in [3]. The sliding window is recognized as an important model for data stream. Based on the sliding window model, previous research studies the computation of different aggregates of data stream, for example, correlated aggregates [10], count and other aggregates[7], frequent itemsets and clusters[9], and correlation[21]. The work [12] studies the problem of learning models from time-changing streams without explicitly applying the sliding window model. The Aurora project[4] considers the systems aspect of monitoring data streams. Also the algorithm issues in time series stream statistics monitoring are addressed in StatStream[21]. In this paper we extend the sliding window model to the elastic sliding window model, making the choice of sliding window size more automatically.

Wavelets are heavily used in the context of data management and data mining, including selectivity estimation[17], approximate query processing[19, 5], dimensionality reduction[6] and streaming data analysis[11]. However, its use in elastic burst detection is innovative.

Data mining on bursty behavior has attracted more attention recently. Wang et al. [20] study fast algorithms using self-similarity to model bursty time series. Such models can generate more realistic time series streams for testing data mining algorithm. Kleinberg[16] also discusses the problem of burst detection in data streams. The focus of his work is in modeling and extracting structure from text streams. The speeches of the State of the Union are also used as an example for discovery of burst structure. Our work is different in that we focus on the algorithmic issue of counting over different sliding windows.

There is also work in finding surprising patterns in time series data. However the definition of surprise is application dependent and it is up to the domain experts to choose the appropriate one for their application. Jagadish et al.[14] use optimal histograms to mine deviants in time series. In their work deviants are defined as points with values that differ greatly from that of surrounding points. Shahabi et al.[18] also use wavelet-based structure(TSA-Tree) to find both trends and surprises in large time series dataset, where surprises are defined as large difference between two consecutive averages of a time series. In very recent work, Keogh et al.[15] propose an algorithm based on suffix tree structures to find surprising patterns in time series database. They try to learn a model from the previously observed time series and declare surprising for those patterns with small chance of occurrence. By comparison, an advantage of our definition of surprise based on spread is that it is simple, intuitive and scalable to massive and streaming data.

5. CONCLUSIONS AND FUTURE WORK

This paper introduces the concept of monitoring data streams based on an elastic window model and demonstrates the desirability of the new model. The beauty of the model is that the sliding window size is left for the system to discover in data stream monitoring. We also propose a novel data structure for efficient detection of elastic bursts and other aggregates. Experiments on real data sets show that our algorithm is faster than a brute force algorithm by several orders of magnitude. We are currently collaborating with physicists to deploy our algorithm for online Gamma Ray burst detection. The monitoring of non-monotonic aggregates is a topic for future work.

6. ACKNOWLEDGMENTS

We are grateful to Prof. Allen I. Mincer of the Milagro Project, for giving us the preliminary tutorial of astrophysics. We also thank the Milagro collaboration for making the Gamma Ray data available to us.

7. REFERENCES

[1] http://www.lanl.gov/milagro/, 2002.

[2] R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient Similarity Search In Sequence Databases. In D. Lomet, editor, *Proceedings of the 4th International Conference of Foundations of Data Organization and Algorithms (FODO)*, pages 69–84, Chicago, Illinois, 1993. Springer Verlag.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, pages 55–68. ACM, 2002.

[4] D. Carney, U. Cetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and

S. B. Zdonik. Monitoring streams - a new class of data management applications. In *VLDB 2002,Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, 2002.

[5] K. Chakrabarti, M. N. Garofalakis, R. Rastogi, and K. Shim. Approximate query processing using wavelets. In *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, pages 111–122, 2000.

[6] K.-P. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering, Sydney, Australia*, pages 126–133, 1999.

[7] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. In *Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 6-8, 2002, San Francisco, CA, USA. ACM/SIAM, 2002*, pages 635–644, 2002.

[8] R. A. et. al. (The Milagro Collaboration). Evidence for TeV emission from GRB 970417a. In *Ap.J. Lett. 533, L119*, 2000.

[9] V. Ganti, J. Gehrke, and R. Ramakrishnan. Demon: Data evolution and monitoring. In *Proceedings of the 16th International Conference on Data Engineering, San Diego, California*, 2000.

[10] J. Gehrke, F. Korn, and D. Srivastava. On computing correlated aggregates over continual data streams. In *Proc. ACM SIGMOD International Conf. on Management of Data*, 2001.

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

[12] G. Hulten, L. Spencer, and P. Domingos. Mining time-changing data streams. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 97–106. ACM Press, 2001.

[13] A. j Smith for the Milagro Collaboration. A search for bursts of tev gamma rays with milagro. In *Proceedings of the 27th International Cosmic Ray Conference(ICRC 2001), 07-15 August 2001, Hamburg, Germany*, 2001.

[14] H. V. Jagadish, N. Koudas, and S. Muthukrishnan. Mining deviants in a time series database. In M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland,UK*, pages 102–113. Morgan Kaufmann, 1999.

[15] E. Keogh, S. Lonardi, and B. Y. Chiu. Finding surprising patterns in a time series database in linear time and space. In *the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,July 23 - 26, 2002. Edmonton, Alberta, Canada*, pages 550–556, 2002.

[16] J. Kleinberg. Bursty and hierarchical structure in streams. In *the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining,July 23 - 26, 2002. Edmonton, Alberta, Canada*, pages 91–101, 2002.

[17] Y. Matias, J. S. Vitter, and M. Wang. Wavelet-based histograms for selectivity estimation. In L. M. Haas and A. Tiwary, editors, *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 448–459, 1998.

[18] C. Shahabi, X. Tian, and W. Zhao. Tsa-tree: A wavelet-based approach to improve the efficiency of multi-level surprise and trend queries on time-series data. In *12th International Conference on Scientific and Statistical Database Management (SSDBM'00),July 26 - 28, 2000,Berlin, Germany*, pages 55–68, 2000.

[19] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadephia, Pennsylvania, USA*, pages 193–204, 1999.

[20] M. Wang, T. M. Madhyastha, N. H. Chan, S. Papadimitriou, and C. Faloutsos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In *ICDE 2002, 18th International Conference on Data Engineering, February 26-March 1, 2002, San Jose, California*, 2002.

[21] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB 2002,Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 358–369, 2002.