Better Burst Detection

Xin Zhang   Dennis Shasha
Department of Computer Science
Courant Institute of Mathematical Sciences
New York University
{xinzhang,shasha}@cs.nyu.edu

Abstract

A burst is a large number of events occurring within a certain time window. Many data stream applications require the detection of bursts across a variety of window sizes. For example, stock traders may be interested in bursts having to do with institutional purchases or sales that are spread out over minutes or hours. Detecting a burst over any of k window sizes, a problem we call elastic burst detection, *in a stream of length N naively requires $O(kN)$ time. Previous work [18] showed that a simple Shifted Binary Tree structure can reduce this time substantially (in very favorable cases near to $O(N)$) by filtering away obvious non-bursts. Unfortunately, for certain data distributions, the filter marks many windows of events as possible bursts, even though a detailed check shows them to be non-bursts.*

In this paper, we present a better framework for elastic burst detection: a family of data structures that generalizes the Shifted Binary Tree. We then present a heuristic search algorithm to find an efficient structure among the many offered by the framework, given the input. We study how different inputs affect the desired structures. Experiments on both synthetic and real world data show a factor of up to 35 times improvement compared with the Shifted Binary Tree over a wide variety of inputs, depending on the data distribution. We show an example application that identifies interesting correlations between bursts of activity in different stocks.

1. Introduction

Detecting bursts is useful in many applications:

- A burst of trading volume in some stock might indicate insider trading.
- A burst of gamma rays may reflect the occurrence of a supernova.
- A burst of methane may anticipate a coming volcanic eruption.

If the length w of the time window when a burst occurs is known in advance, detecting a burst can easily be done in linear time by keeping a running count of the number of events in the last w time units. However, in many situations, the window length is unknown *a priori*. For example, interesting gamma ray bursts could last several seconds, several minutes or even several days.

The *elastic burst detection* problem [18] is to detect bursts across multiple window sizes. Formally:

Problem 1 *Given a data source producing non-negative data elements $x_1, x_2, ...,$ a set of window sizes $W = w_1, w_2, ..., w_m$, a monotonic, associative aggregation function A (such as "sum" or "maximum") that maps a consecutive sequence of data elements to a number (it is monotonic in the sense that $A[x_t \cdots x_{t+w-1}] \leq A[x_t \cdots x_{t+w}]$, for all w), and thresholds associated with each window size, $f(w_j)$, for $j = 1, 2, ..., m$, the elastic burst detection is the problem of finding all pairs (t, w) such that t is a time point and w is a window size in W and $A[x_t \cdots x_{t+w-1}] \geq f(w)$.*

A naive algorithm is to check each window size of interest one at a time. To detect bursts over the k window sizes would then require $O(kN)$ time. This is unacceptable in a high-speed data stream environment.

In [18], the authors show that a simple data structure called the *Shifted Binary Tree (SBT)* could be the basis of a filter that would detect all bursts, and perform in time independent of the number of windows when the probability of bursts is very low.

A Shifted Binary Tree is a hierarchical data structure inspired by the Haar wavelet tree. The leaf nodes of this tree (denoted level 0) correspond to the time points of the incoming data; a node at level $i+1$ aggregates two nodes at level i, so includes 2^{i+1} time points. The Shifted Binary Tree includes a shifted sublevel to each level above level 0. In the shifted sublevel i, the corresponding windows are still of length 2^i but those windows are shifted by 2^{i-1} from the base sublevel. Figure 1 shows an example of a Shifted Binary Tree.

The overlap between the base sublevels and the shifted sublevels guarantees that all the windows of length w, $w \leq 1 + 2^i$, are included in one of the windows at level $i + 1$. Because the aggregation function A is monotonically increasing, $A[x_t \cdots x_{t+w-1}] \leq A[x_t \cdots x_{t+w+c}]$, for all w and c. So if $A[x_t \cdots x_{t+w+c}] \leq f(w)$, then surely $A[x_t \cdots x_{t+w-1}] \leq f(w)$. The Shifted Binary Tree takes advantage of this monotonic property as follows: each node at level $i + 1$ is associated with the threshold value $f(2 + 2^{i-1})$. If more than $f(2 + 2^{i-1})$ events are found in a window of size 2^{i+1}, then a detailed search must be performed to check if some subwindow of size w, $2 + 2^{i-1} \leq w \leq 1+2^i$, has $f(w)$ events. All bursts are guaranteed to be reported, while many non-burst windows are filtered away without need of a detailed check when the burst probability is very low.

However, some detailed searches will turn out to be fruitless (i.e. there is no burst at all). For example, assume the threshold for window size 4 is 100, for 5 is 120, and for 8 is 150. Because each node at level 8 covers window size 4 and 5, if there are 101 events within a level 8 window, a detailed search has to be performed. But there may not be any window of size 4 exceeding the threshold 100. In this case, the detailed search turns out to be fruitless.

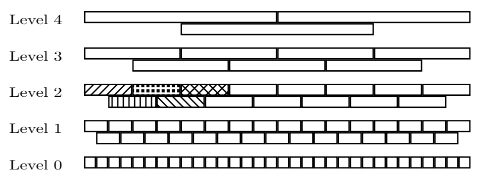
After applying the Shifted Binary Tree in several settings, we have observed two difficulties:

1. When bursts are rare but not very rare, the number of fruitless detailed searches grows, suggesting that we may want more levels than the Shifted Binary Tree provides.

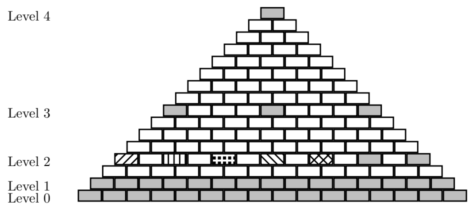
2. Conversely, when bursts are exceedingly rare we may need fewer levels than the Shifted Binary Tree provides.

In other words we want a structure that adapts to the input.

In this paper, we present a family of multiresolution overlapping data structures, called *Shifted Aggregation Trees*, which generalizes the Shifted Binary Tree and includes many other structures. We present a heuristic search algorithm to find an efficient Shifted Aggregation Tree given the input time series and the window thresholds. We theoretically analyze and empirically study how different data distributions and different window thresholds affect the desired structures and the probability to trigger a detailed search. Experiments on both synthetic data and real world data show that the Shifted Aggregation Tree outperforms the Shifted Binary Tree over a variety of inputs, yielding up to a factor of 35 times speedup in some cases.



(a) Shifted Binary Tree



(b) Embed a Shifted Binary Tree in an aggregation pyramid

Figure 1. A Shifted Binary Tree embedded in an aggregation pyramid. Each shaded/grayed cell in the aggregation pyramid corresponds to a node in the Shifted Binary Tree.

The paper is organized as follows. Section 2 introduces the concept of aggregation pyramid, which acts as a host data structure in which all Shifted Aggregation Trees are embedded. Section 3 introduces the Shifted Aggregation Tree and a generalized detection algorithm. Section 4 describes a heuristic state-space search algorithm to find an efficient Shifted Aggregation Tree given the inputs. Section 5 studies how different inputs affect the desired structures and presents experiments and results tested on both synthetic and real world data. Section 6 reviews related work. Section 7 concludes our work.

2. Aggregation Pyramid

2.1. Aggregation Pyramid

Our generalized framework is based on a dense data structure called the *aggregation pyramid (AP)*. All data structures in our framework contain a small subset of the cells of an aggregation pyramid.

An aggregation pyramid (AP) is an N-level isosceles triangular-shaped data structure built over a time window of size N.

- Level 0 has N cells and is in one-to-one correspondence with the original time series.

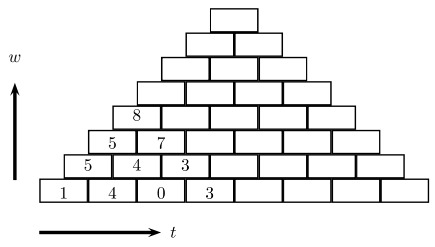


Figure 2. An aggregation pyramid on a window of size 8

- Level 1 has $N-1$ cells, the first cell stores the aggregate of the first two data items (say, data items 1 and 2) in the original time series, the second cell stores the aggregate of the second two data items (data items 2 and 3), and so on.
- Level h has $N-h$ cells, the i^{th} cell stores the aggregate of the $h+1$ consecutive data in the original time series starting at time i.

In all, an aggregation pyramid stores the original time series and all the aggregates for every window size starting at every time point within this sliding window. Each cell corresponds to one window, called the *shadow* of the cell. Figure 2 shows an aggregation pyramid built on a sliding window of size 8.

By construction, an aggregation pyramid has the following properties.

- All the cells along the 45^o diagonal have the same starting time. All the cells along the 135^o diagonal have the same ending time.
- A cell ending at time t at level h, denoted by *cell*(h,t), stores the aggregate for the length $h+1$ window starting at time $t-h$ and ending at time t.
- The shadow window of any cell c in the subpyramid rooted at cell r is covered by the shadow of cell r. We say c is *shaded by* r. By monotonicity, the aggregate in cell c is guaranteed to be bounded by the aggregate in cell r.
- The overlap of two cells is a cell c at the intersection of the 135^o diagonal touching the earlier cell c_1 and the 45^o diagonal touching the later cell c_2. The shadow window for cell c is the intersection of the shadows of cells c_1 and c_2.

2.2. Embedding the Shifted Binary Tree into the Aggregation Pyramid

Recall that in a Shifted Binary Tree, level 0 stores the original time series, and level i stores the aggregates of window size 2^i. Each node in a Shifted Binary Tree has a corresponding cell in the aggregation pyramid. Thus the Shifted Binary Tree can be embedded in the aggregation pyramid. Figure 1 shows how. The grayed cells in the aggregation pyramid correspond to the nodes in the Shifted Binary Tree.

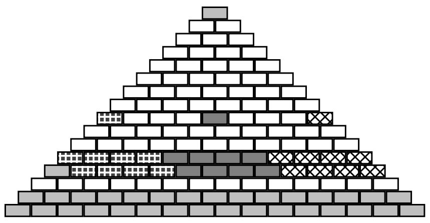


Figure 3. The shadow property and the detailed search region in a Shifted Binary Tree. The quadrilateral-shaped region of a specific pattern is the detailed search region for the corresponding node having the same pattern.

An important property of a Shifted Binary Tree is that a window of length w, $w \leq 1+2^i$, is contained in one of the windows at level $i+1$. This is illustrated in Figure 3. By induction, a window of length w, $w \leq 1+2^{i-1}$ is contained in one of the windows at level i. Thus, after a node at level $i+1$ is updated, if it exceeds the threshold for size $2+2^{i-1}$, i.e. $f(2+2^{i-1})$, then the detailed search has to be performed for all the cells having sizes between $2+2^{i-1}$ and $1+2^i$. Also when a node at level $i+1$ is updated at time t, we need to search only the cells ending after time $t-2^i$, because the cells ending at or before time $t-2^i$ have been covered by the preceding node at level $i+1$. We call this quadrilateral-shaped region — bounded by the window size range $[2+2^{i-1}, 1+2^i]$ and the time range $[t-2^i+1, t]$ — the *detailed search region (DSR)*, please see Figure 3.

Obviously, there are many other possible subsets of the aggregation pyramid. As long as a subset includes the level 0 cells and the top-level cell, it can be used together with this update-search framework to detect bursts. By using different structures on different data inputs, we can achieve optimal performance by trading off structure maintenance against filtering selectivity.

3. Shifted Aggregation Tree

3.1. Shifted Aggregation Tree

Like a Shifted Binary Tree, a *Shifted Aggregation Tree (SAT)* is a hierarchical tree structure defined on a subset of the cells of an aggregation pyramid. It has several levels, each of which contains several nodes. The nodes at level 0 are in one-to-one correspondence with the original time se-

	SBT	SAT
Number of children	2	≥ 2
Levels of children for level $i+1$	i	$\leq i$
Shift at level $i+1$: S_{i+1}	$2 * S_i$	$k * S_i, k \geq 1$
Overlapping window size at level $i+1$: O_{i+1}	window size at level i: w_i	$\geq w_i$

Table 1. Comparing the Shifted Aggregation Tree (SAT) with the Shifted Binary Tree (SBT)

ries. Any node at level i is computed by aggregating some nodes below level i. Two consecutive nodes at the same level overlap in time.

A Shifted Aggregation Tree is different from a Shifted Binary Tree in two ways:

- The parent-child structure
  This defines the topological relationship between a node and its children, i.e. how many children it has and their placements.
- The shifting pattern
  This defines how many time points apart two neighboring nodes at the same level are. We called this distance the *shift*.

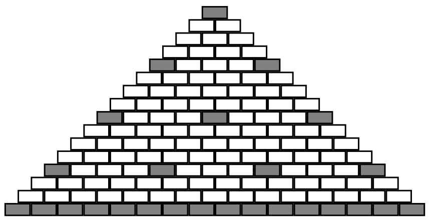
In a Shifted Binary Tree (SBT), the parent-child structure for each node is always the same: one node aggregates two nodes at the level lower. The shifting pattern is also fixed: two neighboring nodes in the same level always half-overlap. In a Shifted Aggregation Tree (SAT), a node could have 3 children and be 2 time points away from its preceding neighbor, or could have 64 children and be 128 time points away from its preceding one. Table 3.1 gives a side-by-side comparison of the difference between a SAT and a SBT. Clearly, a SBT is a special case of a SAT. Figure 4 shows some examples of Shifted Aggregation Trees.

3.2. Shifted Aggregation Tree Shadows and Detection

A Shifted Aggregation Tree has similar properties to a Shifted Binary Tree:

Any window of size w, $w \leq h_i - s_i + 1$, is shaded by a node at level i.

Where h_i is the corresponding window size of level i, and s_i is the shift of level i. Because $h_i - s_i$ is the length of the overlapping shadow between two neighboring nodes at level i, the thresholds of all windows of lengths up to $h_i - s_i + 1$ have to be handled by one of the nodes at level i. By induction, all levels up to $h_{i-1} - s_{i-1} + 1$ have to be shaded by one of the nodes at level $i-1$.


(a) a Shifted Aggregation Tree of size 16

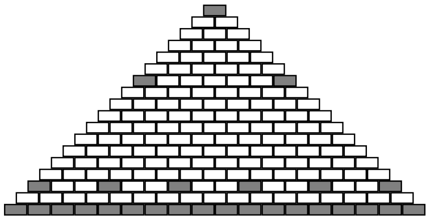

(b) a Shifted Aggregation Tree of size 18

Figure 4. Examples of Shifted Aggregation Trees

The Shifted Aggregation Tree detection algorithm is similar to that of the Shifted Binary Tree, as shown in Figure 5.

The detailed search region $DSR(i,t)$ in a Shifted Aggregation Tree is bounded by the window size range $[h_{i-1} - s_{i-1} + 2, h_i - s_i + 1]$ and the time span $[t - s_i + 1, t]$. This generalizes the detailed search region in a Shifted Binary Tree. By binarily checking against the thresholds for sizes between $h_{i-1} - s_{i-1} + 2$ and $h_i - s_i + 1$, we can find an h, such that $f(h) \leq node(i,t) < f(h+1)$. The cells with sizes greater than h in the $DSR(i,t)$ can be further filtered away, because no burst will present in any window of size greater than h. The detailed search is performed by checking each cell one by one.

Because the shift for each level is fixed, at every s_i time points, a node at level i is updated and its detailed search region is checked if a threshold may have been exceeded. Once a node at the top level is updated, all possible bursts will have been checked. Therefore, a burst is reported no later than s_{top} time points after it occurs, where s_{top} is the shift for the top level.

The total running time of the detection algorithm is the sum of the update time and the comparison/search time. Intuitively, if a Shifted Aggregation Tree has more levels and smaller shifts, i.e. a denser structure, it will take a longer time to maintain this structure, but the probability of an un-

```
for every time point $t$ starting from 1
    $i$ = 1;
    while (a window at level $i$ ends at the current time $t$)
        update $node(i,t)$ by aggregating its children
        if $f(h) \leq node$(i,t) $< f(h+1)$, where
            $h_{i-1} - s_{i-1} + 2 \leq h \leq h_i - s_i + 1$
        then search the portion with sizes $w$, $w \leq h$
            in the detailed search region $DSR(i,t)$
            for real bursts
        endif
        $++i$;
    end
end
```

Figure 5. Shifted Aggregation Tree detection algorithm

necessary search and the cost of searches will both be reduced. A good Shifted Aggregation Tree should balance the update time against the comparison/search time to obtain the optimal performance. In the next section, we present a heuristic state-space algorithm to find an efficient Shifted Aggregation Tree given a sample of the input.

4. Heuristic state-space algorithm to search an efficient Shifted Aggregation Tree

4.1. State-space Algorithm

Given the input series and the window thresholds, the optimization goal is to minimize the time spent both updating the structure and checking for real bursts.

Finding an efficient Shifted Aggregation Tree (SAT) naturally fits into a state-space algorithm framework if we see a Shifted Aggregation Tree as a state and see the growth from one SAT to another as a transformation. In a state-space algorithm, the problem to be solved is represented by a set of states and a set of transformation rules mapping states to states. The solutions to the problem are represented by final states which satisfy some conditions and have no outgoing transformations. The search algorithm starts from one initial state, then repeatedly applies the transformation rules to the set of states currently being explored to generate new states. When at least one final state is reached, the algorithm stops. There are different strategies to choose the order to traverse the state space. Depth-first search, breadth-first search, best-first search, and A^* search are commonly used ones[13].

- Initial state
  Since every Shifted Aggregation Tree has to include the original time series, the starting point is the SAT containing only level 0.
- Transformation rule
  If by adding a level to SAT B, we can get another SAT A, we say state B can be transformed to state A.
- Final states
  Final states are those Shifted Aggregation Trees which can detect bursts in all windows of interest. Since a SAT having top window size h and shift s can cover window sizes up to $h - s + 1$, it's a final state if $h - s + 1 \geq N$, where N is the maximum window size of interest.
- Traversing strategy
  In order to find an efficient structure, we use the best-first strategy to explore the state space. Each state is associated with a cost which will be discussed in 4.2. Since different Shifted Aggregation Trees (SATs) cover different maximum window sizes and have different top-level shifts, the costs are normalized in order for these SATs to be comparable, i.e. divided by the product of the maximum window size and the top-level shift. The state with the minimum cost is picked as the next state to be explored.
- The final Shifted Aggregation Tree with the minimum cost is picked as the desired structure.

Given a Shifted Aggregation Tree, there are many ways it can grow. The next candidate level could aggregate multiple nodes from multiple different levels, and have different shifts. For example, for a Shifted Aggregation Tree containing only level 0, the next possible level could have size 2 and shift 1 or 2; alternatively, it could have size 100 and shift 1, 2 ... 99, and so on. Therefore, we introduce some complexity-reducing constraints to avoid an exhaustive breadth first search strategy.

Let the maximum window size of all the explored states be L. Assume S is the current state to be explored. Instead of generating all possible next states for S at once, we generate only states whose maximum window sizes don't exceed $2L$. Then we put S in a list which stores all the states not yet fully explored. Whenever a new state with a larger window size W is generated, L is updated with the new value W. Then we go through each state in the list of partially-explored states and generate new states for them having maximum window sizes up to the new $2L$.

This avoids growing many highly unlikely Shifted Aggregation Trees at the early stage (saying with a very large size 10000 and shift 5000), but it allows us to gradually grow the intermediate structures and explore the more reasonable ones first. Note that this doesn't prune the search space, but controls the order of traversal of the search space. Our experiments show that the best-first strategy works well. (Fig. 12).

4.2. Cost model

The cost associated with each state is used to indicate which structure to choose in term of running time. One can use an empirical cost, i.e. the actual CPU running time when running this Shifted Aggregation Tree on a small set of sample data. Another method is to use the expected number of operations in a theoretical cost model. The advantage of the theoretical cost model is that the cost of a state can be evaluated much faster, usually hundreds to thousands of times faster depending on the amount of training data. Our model is a simple RAM model: all operations (updates and comparisons) take constant time.

Let s_{top} be the shift at the top level; recall that every s_{top} time points, a node at the top level is updated and bursts below are covered. Thus, we need to consider only the number of operations every s_{top} time points, namely in one update-search *cycle*. The expected number of operations in one cycle is the sum of the number of operations in the updating phase, the comparison phase (to decide if a detail search is needed) and the detailed search phase, respectively.

- Cost in the update phase
  The number of updating operations is just the number of nodes that exist every s_{top} timepoints in the Shifted Aggregation Tree.
- Cost in the comparison phase
  For a node at level i, we need to find out h, $h_{i-1} - s_{i-1} + 2 \leq h \leq h_i - s_i + 1$, such that $f(h) \leq node(i,t) < f(h+1)$. This can be done using binary search. Thus the number of operations is

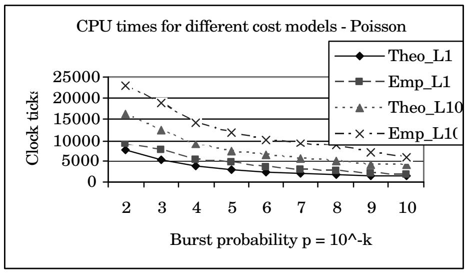
$$\sum_i (\log_2(h_i - s_i - h_{i-1} + s_{i-1} - 1) + 1)$$

- Cost in the search phase
  The number of operations is the expected number of cell accesses in the detailed search region. Let $P(w|h_i)$ be the probability to check a cell of size w given a node at level i with window size h_i, the expected number of cell to be checked is

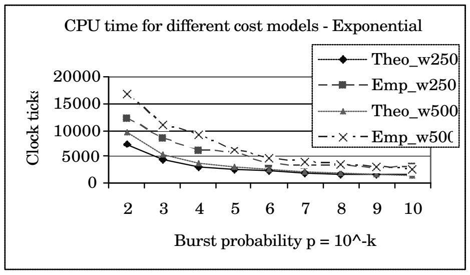
$$\sum_i \sum_w (P(w|h_i))$$

  $P(w|h_i)$ can be estimated from the statistics in the sample data.

Our experiment (Fig. 6) shows that the theoretical cost model models the actual CPU running time well. Because it doesn't suffer the sensitiveness to the fluctuation of the CPU usage in the empirical model, the theoretical cost model usually produces better results than the empirical model, though not perfectly everywhere. The experiment setup is explained in the next section. As a result, we can check many different possible SATs very fast.



(a) Two Poisson distributions with $\lambda = 1, 10$ respectively



(b) Two exponential distributions with maximum window sizes 250 and 500 respectively

Figure 6. Comparison of the theoretical cost model and the empirical cost model on the Poisson data and the exponential data

5. Empirical Results

In this section, we study how Shifted Aggregation Trees perform under different data distributions and different window thresholds. We first test on a set of synthetic data drawn from two classes of distributions common in the real world: the Poisson distribution and the exponential distribution. We analyze the alarm probability, then demonstrate empirically how different distributions and different window thresholds affect the Shifted Aggregation Tree structures, which in turn affect the alarm probability. Later we test our algorithms on two real world data sets: stock data and weblog data. The experiments show that the Shifted Aggregation Tree-based detection always outperforms the Shifted Binary Tree-based detection, sometimes by a multiplicative factor of 35 (Fig. 8).

All the experiments were performed on a 2Ghz Pentium 4 PC having 512 megabytes of main memory. The operating system is Windows XP and the program is implemented in C++. The theoretical cost model (i.e. the expected number of operations) is used in the experiments.

5.1. Shifted Aggregation Tree Density and Alarm Probability

In order to see how the input affects the desired structure, we first define two variables to describe the characteristics of a Shifted Aggregation Tree: *density* and *alarm probability*.

Let s_{top} be the shift at the top level. As noted above, every s_{top} time points, an update-search cycle is finished. The *density* D is defined as

$$D = \frac{Number\ of\ nodes\ in\ the\ SAT\ every\ s_{top}\ points}{Number\ of\ cells\ in\ the\ AP\ every\ s_{top}\ points}$$

Intuitively, the density describes the ratio between the number of cells to be updated in the updating phase and the number of cells to be filtered or searched in the detailed search phase. As the name suggests, it describes how dense a Shifted Aggregation Tree structure is.

While the density characterizes a static structural property of a Shifted Aggregation Tree, the alarm probability describes the dynamic statistical property of a Shifted Aggregation Tree running on a data set. Recall that if a node exceeds the minimum threshold within its detailed search region, it will raise an alarm and start a detailed search. The *alarm probability* P_a^i at level i is defined as

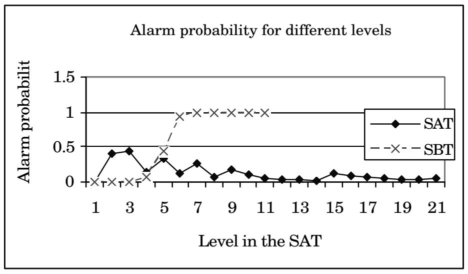
$$P_a^i = \frac{Number\ of\ nodes\ raising\ alarms\ at\ level\ i}{Number\ of\ nodes\ updated\ at\ level\ i}$$

Since the actual CPU cost is positively related both to alarm probability and to the size of the detailed search region, we define the alarm probability of a Shifted Aggregation Tree as the weighted sum of the alarm probability for each level multiplied by the number of cells in their detailed search regions. Intuitively, the larger the alarm probability, the more detailed searches are performed requiring more CPU time. This gives a dynamic statistical description of how a Shifted Aggregation Tree performs on a data set.

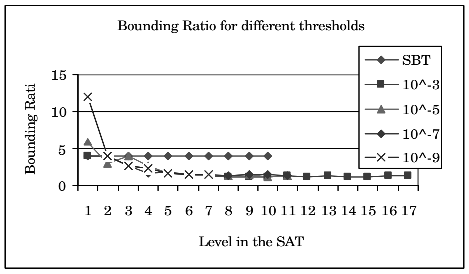
5.2. Synthetic Data

Two classes of probabilistic distributions which have been widely used to model many real world applications were chosen to generate the synthetic data: the Poisson distribution and the exponential distribution. For each distribution, we synthesized a set of data with different parameters in a broad range. Each data set includes 5 million data points. The first 20,000 data points are used as the training data in the state-space algorithm to find a desired structure. To make our task challenging, in these tests, we want to find bursts for every window size between 1 and 250.

Because the Central Limit Theorem says that the sum of N independent random variables with any i.i.d distributions follows the normal distribution when N is large, we will use the normal distribution in the following analysis of the alarm probability.



(a) Alarm probability as a function of window size in the Shifted Binary Tree vs. the Shifted Aggregation Tree



(b) The bounding ratio for different levels in a Shifted Binary Tree and Shifted Aggregation Trees for different burst probabilities

Figure 9. How the bounding ratio in a Shifted Aggregation Tree adjusts as a function of window size and the burst probability to reduce the alarm probability

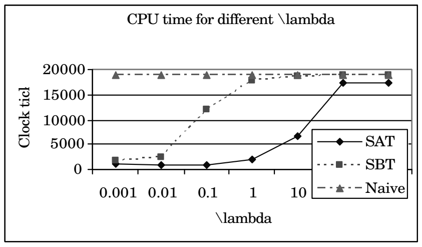
Assume that each point in the input time series has a number of events characterized by a mean μ and a standard deviation σ. Then a sliding window of the time series of size w has mean $w\mu$ and standard deviation $\sqrt{w}\sigma$. Assume that for each window size, the probability to exceed the threshold should be some value p. We can characterize this by saying that $Pr[S_o(w) \geq f(w)] \leq p$, where $S_o(w)$ is the observed number of events for window size w and $f(w)$ is the threshold for window size w.

Let $\Phi(x)$ be the normal cumulative distribution function, for a normal random variable X,

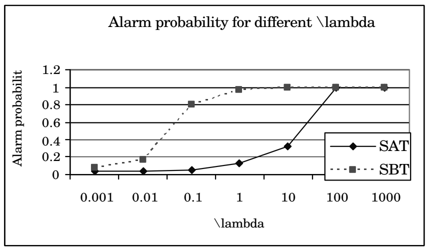
$$Pr[X \geq -\Phi^{-1}(p)] \leq p$$

We have

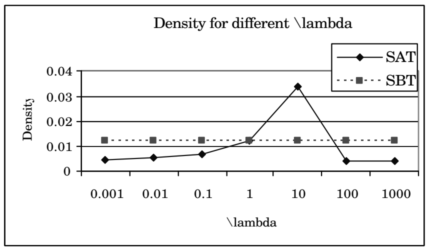
$$Pr[\frac{S_o(w) - w\mu}{\sqrt{w}\sigma} \geq -\Phi^{-1}(p)] \leq p$$



(a) CPU time

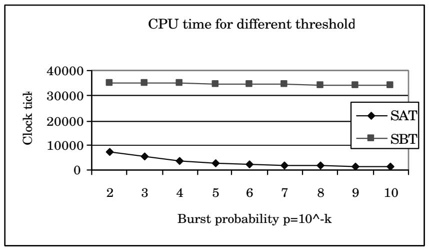


(b) Alarm Probability

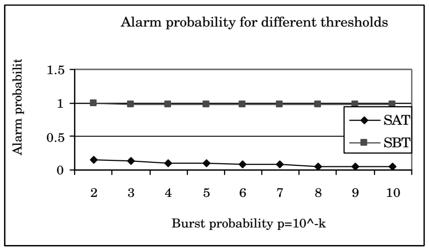


(c) Density (i.e. the ratio between the number of cells to be updated and the number of cells to be filtered or detailed searched)

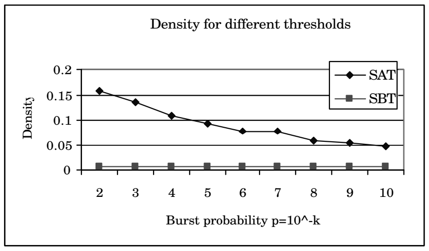
Figure 7. The effect of λ in the Poisson distribution



(a) CPU time



(b) Alarm Probability



(c) Density

Figure 8. The effect of burst probability in the Poisson distribution

Therefore, $f(w)$ should set to be $w\mu - \sqrt{w}\sigma\Phi^{-1}(p)$.

The alarm probability P_a for a window size W is $Pr[S_o(W) \geq f(w)]$. Therefore,

$$\begin{aligned} P_a &= Pr[S_o(W) \geq f(w)] \\ &= Pr[\frac{S_o(W) - W\mu}{\sqrt{W}\sigma} \geq \frac{f(w) - W\mu}{\sqrt{W}\sigma}] \\ &= \Phi(-\frac{f(w) - W\mu}{\sqrt{W}\sigma}) \\ &= \Phi(\frac{(W-w)\mu}{\sqrt{W}\sigma} + \frac{\sqrt{w}\sigma\Phi^{-1}(p)}{\sqrt{W}\sigma}) \\ &= \Phi((\sqrt{T} - \frac{1}{\sqrt{T}})\sqrt{w}\frac{\mu}{\sigma} + \frac{\Phi^{-1}(p)}{\sqrt{T}}) \end{aligned}$$

where $T = W/w$, denoted the *bounding ratio*. The smaller T is, the tighter the bounding, and vice versa.

So P_a is determined by the distribution parameters μ and σ, the threshold parameter p, the bounding ratio T and the level w in the underlying aggregation pyramid. We can draw the following conclusions from the formula above.

- The larger the ratio $\frac{\mu}{\sigma}$ is, the larger the alarm probability P_a.

  For a Poisson distribution with shape parameter λ, the mean μ is λ and the standard deviation σ is $\sqrt{\lambda}$, so the ratio is $\sqrt{\lambda}$. Different λ ranging from 10^{-3} to 10^3 were tested on the synthetic data. In this test, the burst probability is set to be 10^{-6}. Figure 7 shows the CPU times, the alarm probabilities and the densities for different λ.

  As λ, i.e. $(\frac{\mu}{\sigma})^2$, increases, P_a increases. More detailed searches are performed so the CPU time increases. To mitigate this, the Shifted Aggregation Tree must become denser in order to bring down the alarm probability. When λ becomes very large, the alarm probability is close to 1. So the Shifted Aggregation Tree becomes sparse again to reduce the updating time, but is essentially useless.

  For an exponential distribution with scale parameter β, both μ and λ are β, so the ratio is the constant 1. This means that changing β should have no effect on the alarm probability. Due to space limitations, the figures for the exponential data are not shown here. Please refer to [21] for the figures. The experiments show also that there is no noticeable effect of β.

- The smaller the burst probability p, the larger the threshold, the smaller P_a.
  Figure 8 shows the effect of different thresholds. The burst probabilities range from 10^{-2} to 10^{-10}. As the burst probabilities go down, both the alarm probabilities and the densities decrease, because there are fewer bursts to worry about, so speed depends on reducing the updating time.
- As the size w increases, so does P_a.
  Figure 9.b shows the alarm probabilities at different levels in a Shifted Binary Tree and a Shifted Aggregation Tree. The Shifted Binary Tree always has a high alarm probability at the high levels, while in a Shifted Aggregation Tree, by using a small bounding ratio T, the alarm probability remains low. Thus the Shifted Aggregation Tree has more filtering power than the Shifted Binary Tree.
- As the bounding ratio T decreases, so does P_a.
  In a Shifted Aggregation Tree, T could be very close to 1, e.g. $W = w + 1$, whereas T in a Shifted Binary Tree is designed to be about 4. Figure 9.a shows the bounding ratios at different levels of a Shifted Aggregation Tree and a Shifted Binary Tree under different burst probabilities. Notice how the bounding ratio changes in a Shifted Aggregation Tree: it is high at the lower levels where the window size w is small, while low at the higher levels where the window size w is large, in order to bring down the alarm probability. As the burst probability becomes smaller, there are fewer bursts. Thus, the bounding ratio becomes a little larger, and the Shifted Aggregation Tree becomes sparser.

In summary, because the Shifted Aggregation Tree can adjust its structure to reduce the alarm probability, it achieves far better running time thatn the Shifted Binary Tree (Fig. 8).

5.3. Real World Data

We have used two real world data sets to test the proposed framework.

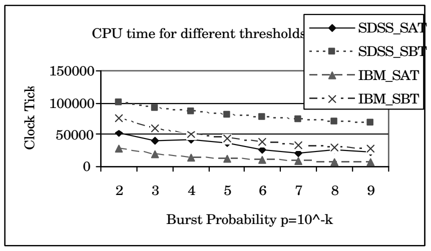
- The Sloan Digital Sky Survey (SDSS) Weblog data
  This data set records all the web access requests to the SDSS website from Jan. 1st, 2003 to Dec. 31st, 2003. Each record includes the request time precise to the second, the source IP address and the target URL. The data set has 17,432,468 records. The distribution follows the Poisson distribution. The training data consists of seven days of second-by-second data.
- The NYSE TAQ Stock Data
  This data set includes tick-by-tick trading activities of the IBM stock between Jan. 1st, 2001 to May 31st, 2004. There are a total of 6,134,362 ticks, and each record contains the time precise to the second, as well as each trade's price and volume. The distribution is close to the exponential distribution. A week's (5 day) worth of data is used as the training data.

5.3.1. Performance Tests We are interested in comparing the Shifted Aggregation Tree with the Shifted Binary Tree under different settings.

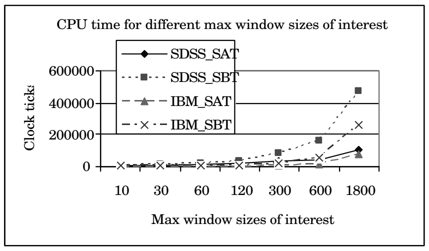
- Different thresholds
  The thresholds are set to reflect a burst probability ranging from 10^{-3} to 10^{-10}. The maximum window size is set to 300 for SDSS, 500 for IBM. Bursts at every window size are detected. Figure 10.a shows the results for both data sets. As the burst probability decreases, the CPU time for the Shifted Aggregation Tree decreases quickly.
- Different maximum window sizes of interest
  The maximum window sizes are set from 10 seconds up to 1800 seconds. The burst probability is set to 10^{-6}. Bursts at every window size are detected. Figure 10.b shows the results. As the maximum window size increases, there are more possible levels to adjust the bounding ratio, thus the speedup for the Shifted Aggregation Tree over the Shifted Binary Tree increases.
- Different sets of window sizes
  Instead of detecting bursts at every window size, we want to see how the Shifted Aggregation Tree performs with different sets of window sizes, n. Suppose n is set to be 1, 5, 10, 30, 60, 120 respectively. The burst probability is set to be 10^{-6} and the maximum window size is set to be 600 for SDSS, 3600 for IBM. Figure 10.c shows that as the set of window sizes becomes sparser, there are fewer bursts to worry about, thus both the Shifted Binary Tree and the Shifted Aggregation Tree take less time.

5.3.2. Robustness test Since the structure of a Shifted Aggregation Tree depends on the input used to train it, we are interested in how sensitive the structure is to whether training on one portion of the data gives good results when tested on another portion.

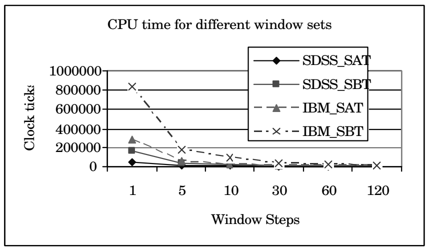
We constructed three training sets for the SDSS data and the IBM data. One set is taken from the testing data to be detected. The second is taken from the same type of data, but outside the testing data. For IBM, it's taken from the trading activities in 2000; for SDSS, it's taken from the weblog of 2004. The third set is taken from the other type of data, i.e, we use the IBM data to train a Shifted Aggregation Tree, then use it to detect the SDSS data, and vice versa. Each



(a) Thresholds

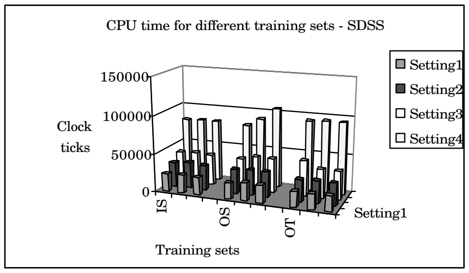


(b) Maximum window size of interest

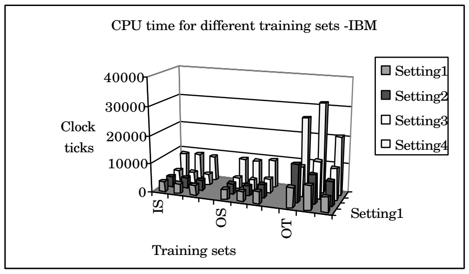


(c) Different set of window sizes of interest

Figure 10. Performance test: CPU time comparison for the Sloan Digital Sky Survey (SDSS) weblog data and the IBM stock data



(a) SDSS



(b) IBM

Figure 11. Robustness Test on the Sloan Digital Survey Weblog Data (SDSS) and the IBM stock data (IS: in-sample, OS: out-of-sample, OT: out-of-type)

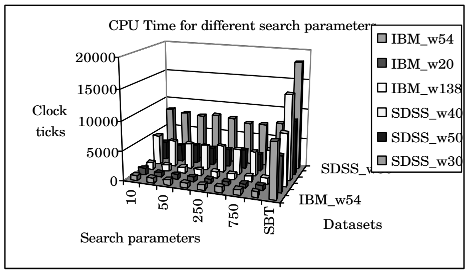


Figure 12. Search parameter Test

training set contains 3 pieces of training data, each piece contains one week's record (7 days for SDSS and 5 days for IBM). Figure 11 shows the CPU times for four different testing scenarios on each training set.

When testing the structure created based on data from the same data type but distinct from the test data, the performance on the IBM data is about the same as using a structure based on the testing data itself. The reason is that the out-of-sample training set has similar statistics to the in-sample training set. By contrast in the SDSS data, the statistics in the out-of-sample training set are different from those in the in-sample training set. Thus the structure based on out-of-sample data costs about 20 percent more time than one based on in-sample data.

A structure based on a different data type can perform quite poorly. For example, a structure based on SDSS data performs by a factor of 2 to 3 times slower for IBM data than a structure based on out-of-sample IBM data.

5.3.3. Search parameter in the state-space algorithm We want to study how different search parameters affect the desired Shifted Aggregation Tree structures in the state-space algorithm. We tested on different data settings with different numbers of final states to see when there are diminishing returns to broadening the search. The number of final states is set to be 10, 25, 50, 100, 250, 500, 750, 1000 respectively. Three pieces of training data are picked for the SDSS data and the IBM data respectively. Due to space limitations, details not shown here can be found in [21].

Figure 12 shows the CPU running times for the Shifted Aggregation Trees using these parameters. It also shows the CPU running time for the Shifted Binary Tree as a reference. The experiments show that even with small numbers of final states, the Shifted Aggregation Trees discovered are close to those discovered when the number of final states is much larger. The best-first search strategy works well in this situation. In practice, we believe that setting the number of final states to be 500 works well.

5.4. Sample Data Mining Application

We believe that high-performance burst detection could be a preliminary primitive for further knowledge discovery and data mining process. As an example, we look at the correlation of bursts in stock data.

We collected the tick-by-tick TAQ stock data in 2003 for the Standard & Poor's 100 stocks. We want to discover which stocks share similar volume characteristics, i.e. when there is a burst of trading in one stock, which other stocks also exhibit a burst? Because trading bursts can happen across different time resolutions, we monitor the correlation at multiple time scales and set the window sizes of interest to be 10, 30, 60, and 300 seconds. The burst probability is set to 10^{-9}.

Bursts are detected using a Shifted Aggregation Tree, tuned as described above. The bursts detected are converted to a 0-1 string where 0 means no burst and 1 means a burst. The correlation is computed over these 0-1 strings.

These bursts tell an interesting story. First, stocks within the same sector are correlated strongly e.g. Microsoft (MSFT), Oracle(ORCL) and Cisco(CSCO). Surprisingly strong correlations of bursty behaviors can be found across different industries also however. For example, Pfizer Inc. (PFE, health care, Drugs, major Pharmaceuticals), Pepsico Inc. (PEP, Beverage), Procter & Gamble Co. (PG, Non-Durables Household Products) are highly correlated. Table 2 shows some highly correlated stocks at different window sizes. We are not claiming these still anecdotal observations as a major result of our paper, but just as a suggestive example of how burst detection can feed into data mining applications.

Resolution	Highly-correlated stocks
10s	C/GE/XOM, CSCO/MSFT/ORCL
30s	C/GE/XOM, CSCO/MSFT/ORCL, PEP/PFE/PG
60s	C/GE/XOM/PEP/PFE/PG/GE, CSCO/MSFT/ORCL
300s	C/GE/XOM/PEP/PFE/PG/GE, CSCO/MSFT/ORCL, WFC/XOM/WMT, KO/USB/VZ

Table 2. Some highly-correlated stocks at different resolutions

6. Related Work

There has been a lot of research in monitoring and mining data streams as surveyed in [2, 7, 6]. Recently, monitoring and mining bursty behaviors is attracting more and more interests.

Wang et al. [20] model the bursty behavior in self-similar time series to synthesize more realistic traces. Kleinberg [10] studies the bursty and hierarchical structure in temporal text stream. His interest is to discover how high frequency words change over time. Our focus is a high performance algorithm to detect bursts, thus complementing that work.

Neill et al. [15, 16, 14] study the problem of detecting significant spatial clusters in multidimensional space. They consider a general density function which could be non-monotonic, but are only interested in the region with the highest density, while our work detects all regions exceeding some threshold. They use an overlapping data structure called the overlap-kd tree with a fixed overlapping structure. Thus their structure, like the Shifted Binary Tree, is fixed regardless of the characteristics of the input data. Our techniques could be applied to their data structure, an area meriting further investigation.

Vlachos et al. [19] mine the bursty behavior in the query logs of the MSN search engine. They use moving averages to detect time regions having high numbers of queries. Only two window sizes are considered, short term and long term. The detected bursts are further compacted and stored in a database to support burst-based queries. We share the view that burst detection should be a preliminary primitive for further knowledge mining process, but we deal with many more window sizes.

Datar et al. and Gibbons et al. [5, 8] study a related problem: estimating the number of 1's in a 0-1 stream and the sum of bounded integers in an integer stream in the last N elements. They use synopsis structures called Exponential Histograms and Waves respectively. Like our Shifted Aggregation Tree, these are multiresolution aggregation structures, though with coarser aggregation levels for the past and finer aggregation levels for recent data.

Burst detection belongs to a broader category of detection tasks: change/novelty/anomaly/outlier/surprise detection. Due to space limitations, we won't compare them here. Please refer to [9, 3, 12, 11, 1, 4, 17], etc, for related work in this area.

7. Conclusion

In this paper, we propose a framework for adaptive and therefore better elastic burst detection. We present a family of data structures that generalizes the Shifted Binary Tree and many others. We present a heuristic search algorithm to find an efficient structure given the input time series and the window thresholds. Experiments on both synthetic and real world data show an improvement factor of up to 35 times depending on the input.

Besides its immediate practical benefits, this framework – aggregation pyramid along with a simple adaptive search methodology – can be extended to spatial burst detection and other applications. Applying this framework to time-evolving time series is also the topic of future work. Further, given rapid burst detection, new real-time data mining applications may become possible.

References

[1] Charu C. Aggarwal and Philip S. Yu. Outlier detection for high-dimensional data. Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, 2001.

[2] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. Madison and Wisconsin, 2002. ACM SIGMOD-PODS.

[3] Stephen D. Bay and Mark Schwabacher. Mining distance-based outliers in near linear time with randomization and a simple pruning rule. Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2003.

[4] Markus M. Breunig, Hans-Peter Krigel, Raymond T. Ng, and Jorg Sander. Lof: Identifying density-based local outliers. Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, 2000.

[5] Mayur Datar, Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Maintaining stream statistics over sliding windows. *SIAM*, 31(6), September 2002.

[6] Christos Faloutsos. Indexing and mining streams tutorial. Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data, 2004.

[7] M. Garofalakis, J. Gehrke, and R. Rastogi. Querying and mining data streams: You only get one look. Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, 2002.

[8] Phillip B. Gibbons and Srikanta Tirthapura. Distributed stream algorithms for sliding windows. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 63–72, 2002.

[9] E. Keogh, S. Lonardi, and W. Chiu. Finding surprising patterns in a time series database in linear time and space. Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002.

[10] Jon Kleinberg. Bursty and hierarchical structure in streams. pages 91–101. Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2002.

[11] E. M. Knorr, R. T. Ng, , and V. Tucakov. Distance-based outliers: algorithms and applications. Proceedings of 26th International Conference on Very Large Data Bases, 2000.

[12] Edwin M. Knorr and Raymond T. Ng. Finding intensional knowledge of distance-based outliers. Proceedings of 25th International Conference on Very Large Data Bases, 1999.

[13] Zbigniew Michalewicz and David B. Fogel. *How To Solve It: Modern Heuristics*. Springer, 2002.

[14] Daniel Neill and Andrew Moore. Rapid detection of significant spatial clusters. Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2004.

[15] Daniel B. Neill and Andrew W. Moore. A fast multi-resolution method for detection of significant spatial disease clusters. In *Advances in Neural Information Processing Systems 16*, Cambridge, MA, 2004. MIT Press.

[16] Daniel B. Neill, Andrew W. Moore, Francisco Pereira, and Tom Mitchell. Detecting significant multidimensional spatial clusters. In *Advances in Neural Information Processing Systems 17*, pages 969–976, Cambridge, MA, 2005. MIT Press.

[17] S. Papadimitriou, H, Kitagawa, P.B.gibbons, and C.Faloutsos. Loci: Fast outlier detection using the local correlation integral. Proceedings of the 19th International Conference on Data Engineering, 2003.

[18] Dennis Shasha and Yunyue Zhu. *High Performance Discovery in Time Series: Techniques and Case Studies*. Springer, 2003.

[19] Michail Vlachos, Christopher Meek, Zografoula Vagena, and Dimitrios Gunopulos. Identifying similarities, periodicities and bursts for online search queries. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 131–142, New York, NY, USA, 2004. ACM Press.

[20] Mengzhi Wang, Tara madhyastha, Ngai Hang Chan, Spiros Papadimitriou, and Christos Faloutos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. Proceedings of the 18th International Conference on Data Engineering, 2002.

[21] Xin Zhang and Dennis Shasha. High performance burst detection. *Technical Report, New York University*, 2005.