Better Burst Detection (TR2005-876)

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

October 29, 2005

Abstract

A burst is a large number of events occurring within a certain time window. As an unusual activity, it's a noteworthy phenomenon in many natural and social processes. 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 [24] 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 new algorithmic 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

A burst is an unexpectedly large number of events occurring within a certain time window. As an unusual activity, it's a noteworthy phenomenon in many natural and social processes.

- In stock trading, trading volume is an important indicator of the price trend. A burst of volume often indicates a strong buy/sell interest, thus leads a price movement. [1]

- In astronomy, astrophysicists are interested in high-energy photons activities in the universe. When a burst of photon activity is observed, a gamma ray burst occurs which may reflect the occurrence of a supernova.
- In telecommunication, a large number of access requests within a short period of time might indicate a Distributed Denial of Service (DDoS) attack, worth closely monitoring.

To efficiently detect bursts is of critical importance under some circumstances. For example, to detect unusually high tsunami activity as early as possible could save thousands of lives.

If the length of the time period when a burst occurs is known a priori, the detection can easily be done in linear time by keeping a running count of the number of events. However, in many situations, the window size is unknown a priori. For example, interesting gamma ray bursts could last several seconds, several minutes or even several days. The size itself may be an interesting subject to be discovered. Furthermore, many data applications require detection of bursts across a variety of window sizes. For example, traders use multiple Volume Moving Averages (VMA) at different time scales from minutes to years to infer the market trend.

1.1 Elastic burst detection and the Shifted Binary Tree

The *elastic burst detection* problem [24] 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 k window sizes in a sequence of length N naively requires $O(kN)$ time. This is unacceptable in a high-speed data stream environment.

In [24], the authors show that a simple data structure called the *Shifted Binary Tree* 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 1 aggregates two adjacent nodes at level 0. In general, a node at level $i+1$ aggregates two nodes at level i, thus includes 2^{i+1} time points. There are only $log_2 N + 1$ levels where N is the maximum window size. 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.

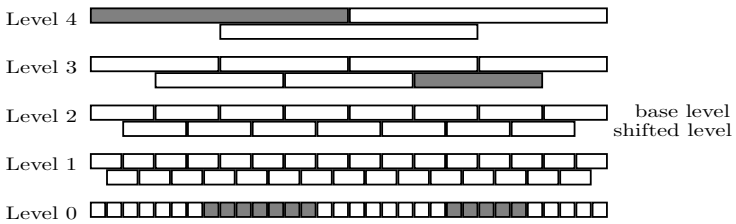


Figure 1: An example of a Shifted Binary Tree. The two shaded sequences in level 0 are included in the shaded nodes in level 4 and level 3 respectively.

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, i.e. $A[x_t \cdots x_{t+w-1}] \leq A[x_t \cdots x_{t+w+c}]$, 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 $f(2+2^{i-1})$. Whenever 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 and many non-burst windows are filtered away without requiring 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.

1.2 Contributions

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.

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 as a Host Data Structure

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 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.
- 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.
- The top level has 1 cell, storing the aggregate over the whole time window.

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. The value (starting time, ending time, length/size) of a cell is the aggregate (starting time, ending time, length/size) of its corresponding shadow window. Figure 2 shows an aggregation pyramid built on a sliding window of size 8.

By construction, an aggregation pyramid has the following properties as shown in Figure 3.

- 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.

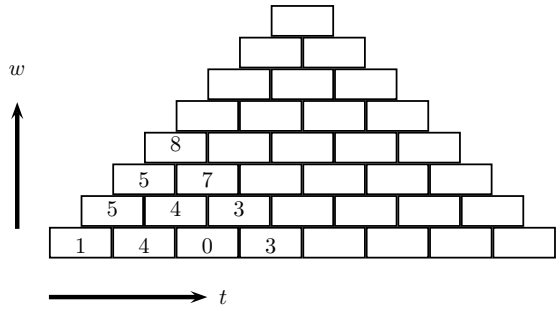


Figure 2: An aggregation pyramid on a window of size 8

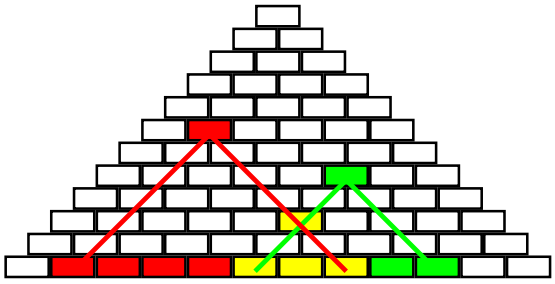


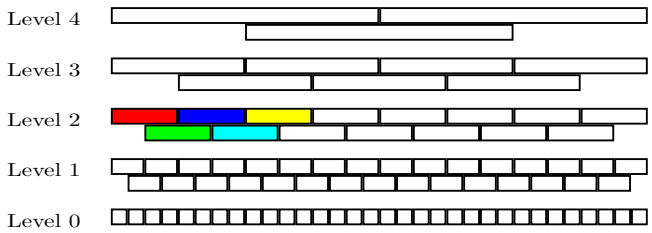
Figure 3: Shadow and Overlap in an Aggregation Pyramid. The red and yellow subsequence between two red diagonals is the shadow of the red cell, the yellow and green subsequence between two green diagonals is the shadow of the green cell. The yellow subsequence is the overlap of the red cell and the green cell.

- 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.

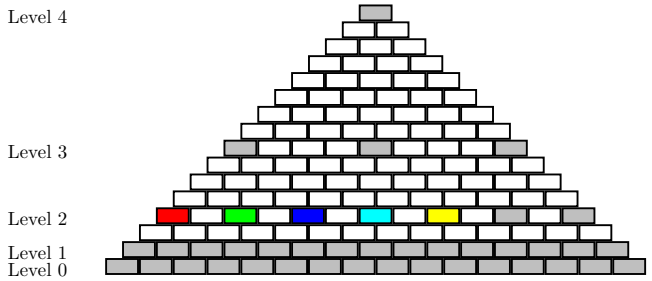
When a new data item arrives at time t, the aggregation pyramid can easily be updated by recursively applying the follow formula from $h = 0$ to the top level.

$$cell(h, t) = cell(h - 1, t - 1) + cell(1, t)$$

If $cell(h, t)$ exceeds the threshold for a window of size $h+1$, i.e., exceeds $f(h+1)$, a burst ending at time t has occurred.



(a) Shifted Binary Tree



(b) Embed Shifted Binary Tree in Aggregation Pyramid

Figure 4: Embed a Shifted Binary Tree (SBT) in an Aggregation Pyramid (AP). Each grayed/colored cell in the AP corresponds to a node in the SBT. The different colors in level 2 show the one-to-one correspondence.

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. So, 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 4 shows how. The colored/grayed cells in the aggregation pyramid correspond to the nodes in the Shifted Binary Tree. Notice that level i in a Shifted Binary Tree corresponds to level 2^i in the aggregation pyramid.

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 intuitively in Figure 5.

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

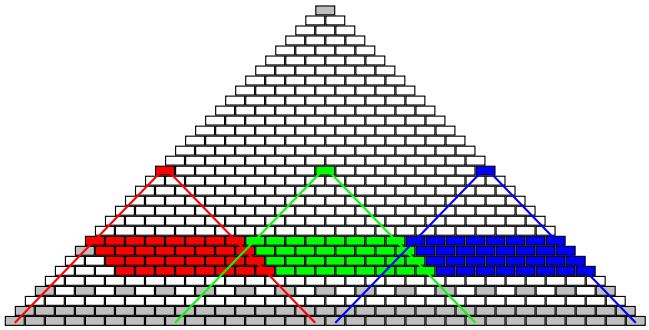


Figure 5: The shadow property and the detailed search region in a Shifted Binary Tree. The quadrilateral-shaped region of a specific color is the detailed search region for the corresponding node having the same color.

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 5.

Obviously, there are many other possible embeddings into 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, because the shadow of the top-level cell includes everything. Clearly, it's very likely for the top-level cell to exceed the threshold of window size 1. In that case, it will raise an alarm every time vastly increasing the need to search.

The Shifted Binary Tree structure reduces the alarm probability by half-overlapping two consecutive nodes at the same level. So the trigger for a cell of window size 2^{i+1} to do a detailed search is the threshold for more than a quarter that size. Thus, the probability of raising an alarm is dramatically reduced and more cells filtered out in the first stage.

Furthurmore, by using different embedding structures on different data inputs, we can adjust the probability of raising an alarm and the cost of maintaining the structure. The optimal performance can be achieved by trading off structure maintenance against filtering selectivity.

3 Shifted Aggregation Tree

3.1 Shifted Aggregation Tree Generalizes Shifted Binary 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 series. 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

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

	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	$\geq w_i$

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 are two neighboring nodes at the same level. We call 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 one 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 6 shows some examples of Shifted Aggregation Trees.

3.2 Shifted Aggregation Tree Shadows and Detection

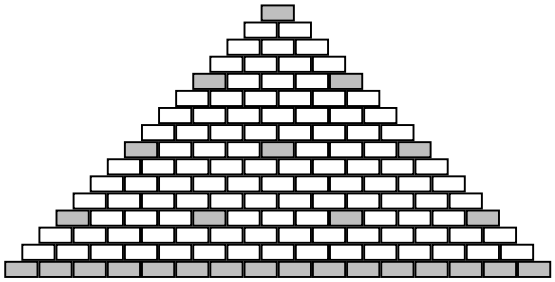
A Shifted Aggregation Tree shares an important property with a Shifted Binary Tree:

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

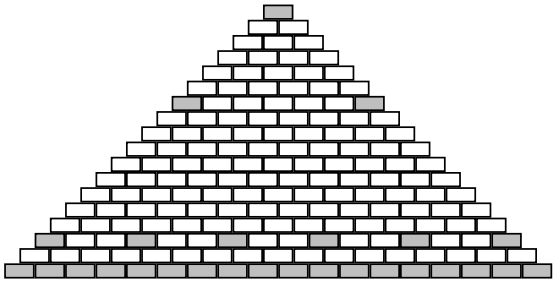
Where h_i is the corresponding window size of level i, and s_i is the shift of level i. Figure 7 illustrates this property in the aggregation pyramid. Because $h_i - s_i$ is the length of the overlapping shadow between two neighboring nodes at level i, the thresholds of all windows of lengths up to $h_i - s_i + 1$ have to be shaded 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 $i-1$.

The Shifted Aggregation Tree detection algorithm is similar to that of the Shifted Binary Tree, as shown in Figure 8.

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. Part of the detailed search region can be further filtered away, by binarily checking the aggregate in a node at level i 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 $f(h) \leq node(i,t) < f(h+1)$, no burst will present in any window of size greater than h.



(a) a Shifted Aggregation Tree of size 16



(b) a Shifted Aggregation Tree of size 18

Figure 6: Examples of Shifted Aggregation Trees

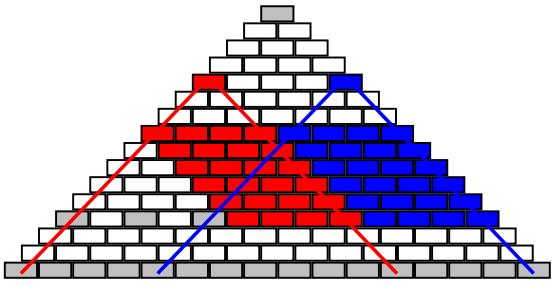


Figure 7: Illustration of the shadow property and the detailed search region in a Shifted Aggregation Tree

```
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) ≤ node(i, t) < f(h + 1),
            where h_{i-1} − s_{i-1} + 2 ≤ h ≤ h_i − s_i + 1
        then search the portion with sizes w, w ≤ h,
            in the detailed search region DSR(i, t) for real bursts
        endif
        ++i;
    end
end
```

Figure 8: Shifted Aggregation Tree detection algorithm

The detailed search is performed by checking each cell one by one. Notice two neighboring cells overlap, to avoid duplicate computation, we start from one "seed" cell, then by adding/subtracting the difference between two neighboring cells, we can get the aggregate for the neighboring cells. This process is repeated until the whole DSR is populated.

Because of the properties of a SAT, it's guaranteed to find such a "seed" in or near each DSR without the need to aggregate a long sequence of the original time series. Recall in a SAT, the shift at level i is a multiple of the shift at level $i-1$, i.e. $s_{i-1} \leq s_i$, and the time span for the $DSR(i,t)$ is s_i, there has to be a node at level $i-1$ whose shadow window ends between the interval $t - s_i + 1$ and t, call it S. And in a SAT, the overlap of two neighboring nodes at level i has to cover any node at level $i-1$, i.e $h_{i-1} \leq h_i - s_i + 1$. If $s_{i-1} > 1$, then $h_{i-1} - s_{i-1} + 2 \leq h_{i-1}$, i.e. level $i-1$ is between $h_{i-1} - s_{i-1} + 2$ and $h_i - s_i + 1$, thus S lies within the $DSR(i,t)$. If $s_{i-1} = 1$, then S lies one level lower than the $DSR(i,t)$.

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 it exceeds its minimum threshold. 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 a fruitless search and the cost of searches will both be reduced. Adversely, a sparser structure costs less time to update, but may take more time to do detailed searches. 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

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.

4.1 State-space Algorithm

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[16].

- 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 onto the top of SAT B, we can get another SAT A, we say state B can be transformed to state A. Recall there are some constraints the top level of SAT A has to satisfy. Each node at the top level has to aggregate several children in the lower levels of SAT B. The shadow of all the nodes of the top level has to cover the whole SAT B. The shift for the new level has to be an integral multiple of the shift of the level below in order to speed up detailed search.

  The transformation rule defines how to grow a complicated SAT from the first simple SAT.

- 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

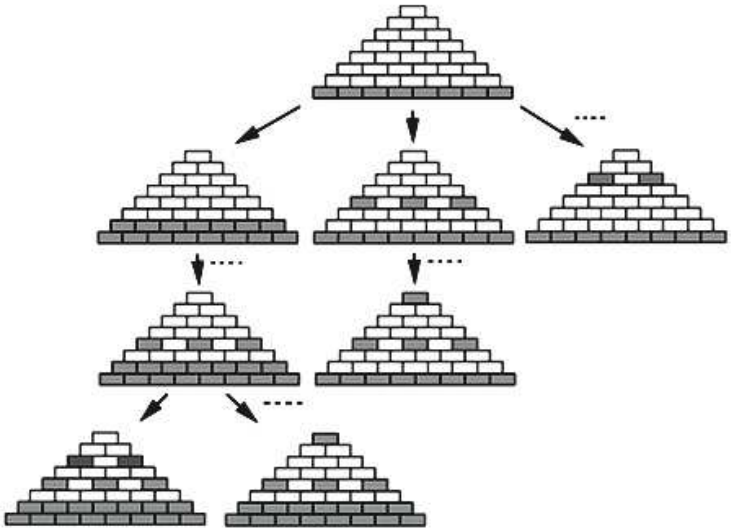


Figure 9: State space growth

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.

In summary, the algorithm starts with a Shifted Aggregation Tree having level 0 only, then keeps growing the candidate set of SATs, until a set of final SATs are reached. Figure 9 illustrates how the state space grows.

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. Such combinatorial considerations show that there are an exponential number of ways to grow a Shifted Aggregation Tree. 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 window 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. 22).

We also restrict the number of states having the same shadow size and the number of final states. For example, if we have visited 500 states whose maximum shadow is of size 100, we don't explore any new such states. And if we have visited say 10000 final states, the algoritm stops.

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 measure this cost empirically by running this Shifted Aggregatioon Tree on a small set of sample data. Another method is to use the expected number of operations in a theoretical cost model to model the CPU running time. 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 only need to consider 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 update phase, the filtering 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 filtering 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. The number of comparison operations is

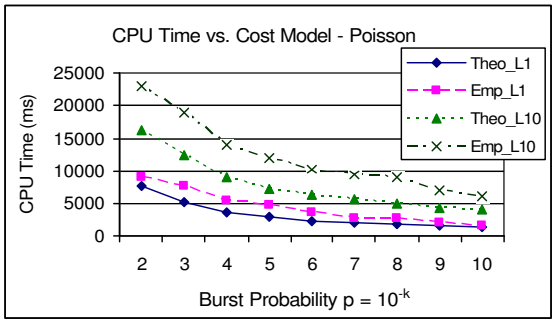
$$\sum_i (\log_2(h_i - s_i - h_{i-1} + s_{i-1} - 1) + 1)$$

- Cost in the detailed 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, s_i be the shift at level i, the expected number of cell to be checked is

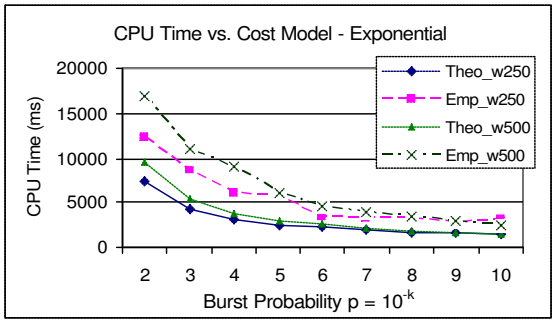
$$\sum_i \sum_w (P(w|h_i) \cdot s_i)$$

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

The advantage of the theoretical cost model is that it doesn't subject to the fluctuation of the CPU usage in the empirical model when testing on the sample data. In the early stage of the state-space algorithm, the fluctuation of the CPU usage could cause inaccurate cost of a state, so that some worse state and its descendants get explored first due to the best-first strategy. As stated



(a) Two Poisson distributions with $\lambda = 1, 10$ respectively (L1:$\lambda = 1$,L10:$\lambda = 10$)



(b) Two exponential distributions with maximum window sizes 250 (w250) and 500 (w500) respectively

Figure 10: Comparison of the theoretical cost model and the empirical cost model on Poisson data and exponential data

above, because we limit the number of states having the same window size and the number of final states in order to prune the exponential state space, the actual better state and its descendants may be pruned in the later stage, thus a better solution is missed in this case. Another advantage is that the theoretical model is much faster than the empirical model, usually thousands of times faster depending on the amount of training data.

Our experiment (Fig. 10) shows for different settings, i.e. different burst probabilities, different maximum window sizes of interest and different distribution parameters, the theoretical model performs better than the empirical model. The theoretical cost model models the actual CPU running time well for Poisson and exponential distributions. The data setup is explained in the next section.

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 desired Shifted Aggregation Trees, which in turn affect the alarm probability. Later we test our algorithms on two real world data sets: stock data and website traffic 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. 15).

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. The CPU time in each test is the wall clock time spent on each testing data set (5 million data points in synthetic data, about 31 million points in the SDSS SkyServer traffic data and about 23 million points in the IBM stock data).

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

$$D = \frac{Number\ of\ nodes\ in\ the\ SAT\ in\ one\ cycle}{Number\ of\ cells\ in\ the\ pyramid\ in\ one\ cycle}$$

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 when embedding in the aggregation pyramid.

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

A set of synthetic data was generated using a random number generator. 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.

- Poisson distribution
  Many real world phenomena can be modeled as a Poisson process, such as customers arriving at a service station, emissions from radioactive material, etc. It's well known in a Poisson process the number of events happening within the time interval $[0, t]$ follows the Poisson distribution. Also the normal distribution is the limit distribution of the Poisson distribution.

- Exponential distribution
  One class of data application that doesn't follow the Poisson distribution [26] [10] but is characterized by behaviors such as network traffic is self-similar or fractal data. In a fractal process, following the "80/20" law" for example i.e, say 80% of the time there is no activity, 20% of the time there is some activity; within the 20% of the time, 80% of that time has little activity and 20% of that time there is high activity; and so on. In such a case, the number of activities within one unit time follows the exponential distribution.

For each distribution, we synthesized a set of data with different distribution parameters in a broad range. Each testing 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 use the normal distribution in the following analysis of 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

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

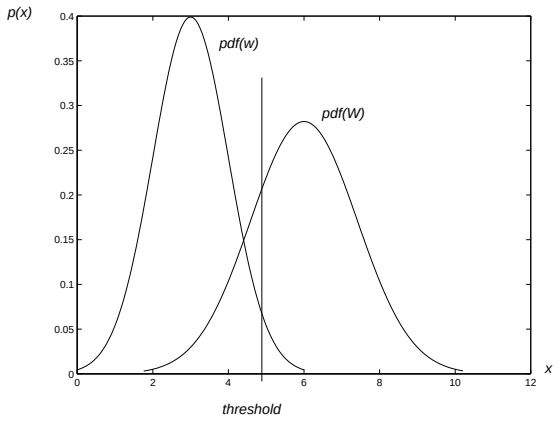


Figure 11: Illustration of the alarm probability for a window of size W to exceed the threshold for size w, i.e. the portion under the pdf of size W and to the right of the threshold line for size w.

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

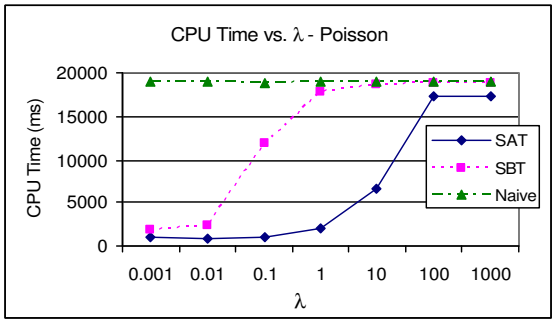
The alarm probability P_a for an aggregate of window size W to exceed the threshold for size w, is $Pr[S_o(W) \geq f(w)]$. Therefore,

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

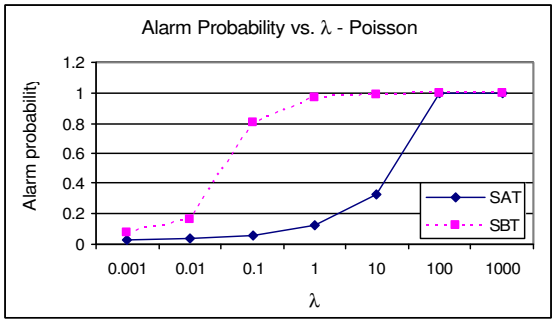
where $T = W/w$, denotes the *bounding ratio*. The smaller T is, the tighter the bounding, and vice versa.

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.

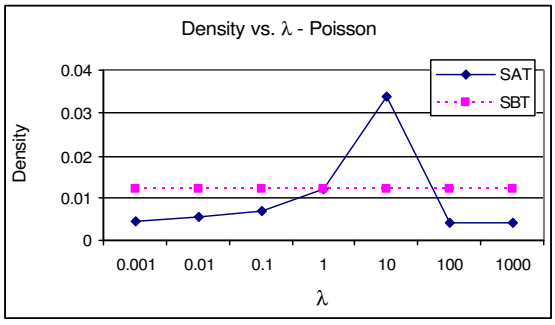
- The larger the ratio $\frac{\mu}{\sigma}$ is, the larger the alarm probability P_a.
  This is illustrated intuitively in Figure 11. Figure 11 shows the probability density functions (pdf) for two normal random variables, one for the number of events in a window of size w which has mean $w\mu$ and standard deviation $\sqrt{w}\sigma$, another similar one for a window of size W. The threshold line shows where $f(w)$ lies. When a distribution realization for size W appears to the right of the threshold line, the aggregate is greater than the threshold for size w, an alarm is raised. So the value of P_a is the area below the pdf of size W but to the right of the threshold line.



(a) CPU time

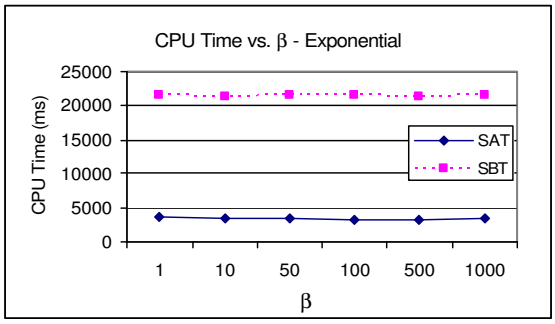


(b) Alarm Probability

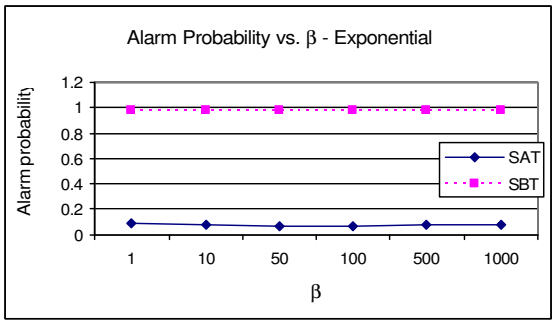


(c) Density (i.e. the ratio between the number of cells to be updated and the number of cells to be filtered or detailed searched)

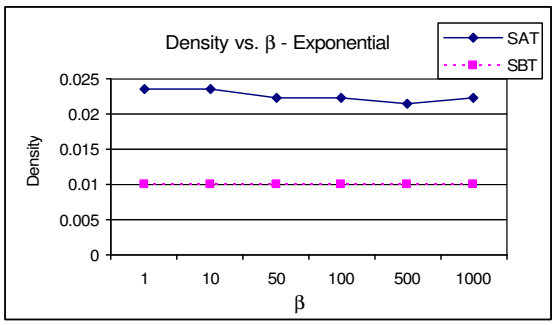
Figure 12: The effect of λ in the Poisson distribution



(a) CPU time



(b) Alarm Probability



(c) Density

Figure 13: The effect of β in the exponential distribution

As μ increases, both curve peaks and the threshold line move to the right along the x axis, but the gap between the two peaks increases. There are more portions to the right of the threshold line under the pdf of size W. There are more chances to raise an alarm. As σ increases, both curves stretch along the x axis, the threshold line moves to the right. The portion to the right of the threshold line under the pdf of size W decreases. There are less chances to raise an alarm.

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. In this test, the burst probability is set to be 10^{-6}. Figure 12 shows the CPU time, 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 anyway. So the Shifted Aggregation Tree becomes sparse again to reduce the updating time, but is essentially useless.

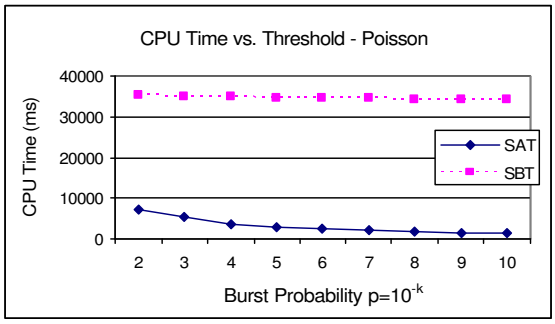
For an exponential distribution with scale parameter β, both μ and λ are β, so the ratio is the constant 1. This means that changing β should have no effect on the alarm probability. Figure 13 shows the effect of different β. The experiments show that there is no noticeable effect of β.

- The smaller the burst probability p, the larger the threshold, the smaller P_a.
  This essentially moves the threshold line of size w to the right in Figure 11. So P_a decreases.

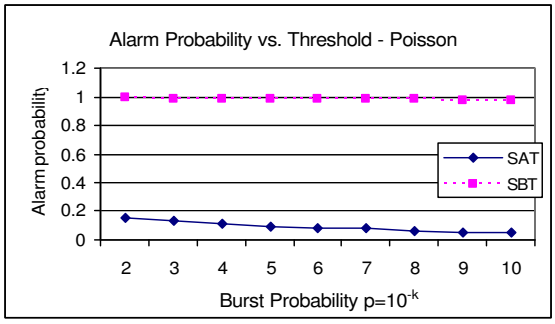
  Figure 14 and 15 show the effect of different thresholds for the Poisson distribution and the exponential distribution respectively. The burst probabilities range from 10^{-2} to 10^{-10}. As the burst probabilities go down, both the alarm probabilities and the densities decrease, because there are fewer bursts to worry about, so speed depends on reducing the updating time.

- As the 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 16.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.

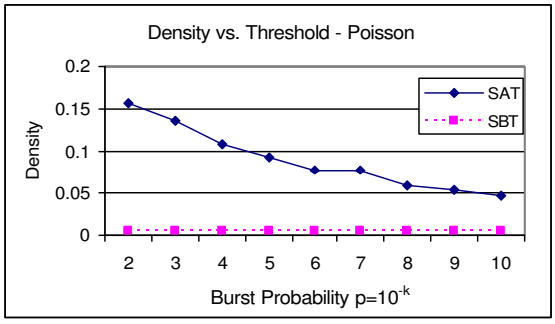
- As the size w increases, so does P_a.
  Figure 16.b shows the alarm probabilities at different levels in a Shifted Binary Tree and a Shifted Aggregation Tree. The Shifted Binary Tree always has a high alarm probability at the high levels, while in a Shifted



(a) CPU time

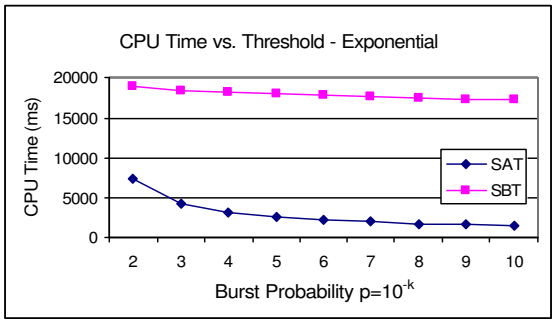


(b) Alarm Probability

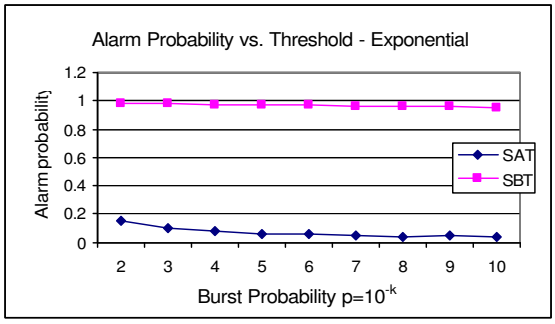


(c) Density

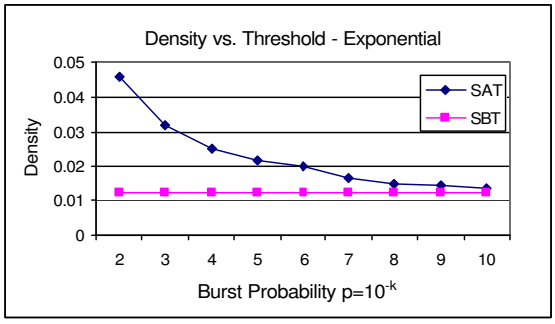
Figure 14: The effect of burst probability in the Poisson distribution



(a) CPU time

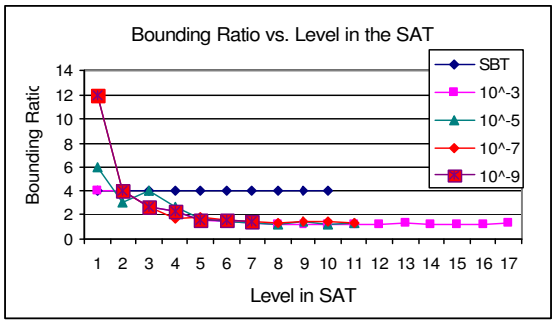


(b) Alarm Probability

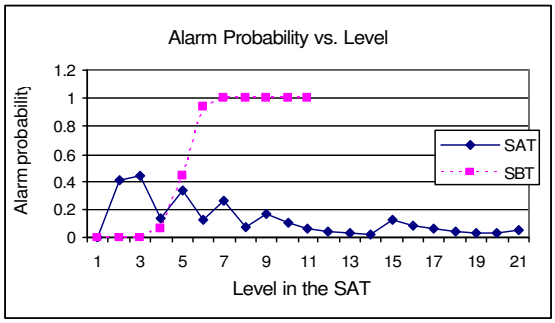


(c) Density

Figure 15: The effect of the burst probability in the exponential distribution



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



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

Figure 16: How the bounding ratio in a Shifted Aggregation Tree adjusts as a function of window size and the burst probability in order to reduce the alarm probability.

Table 2: Statistics for the SDSS SkyServer traffic data and the IBM stock data

	SDSS	IBM
Size	31,536,000	23,085,000
Mean	120.95	287.06
Standard deviation	64.87	2796.05
Min	0	0
Max	576	2806500

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.

In summary, because the Shifted Aggregation Tree can adjust its structure to reduce the alarm probability, it achieves far better running time than the Shifted Binary Tree (Fig. 15).

5.3 Real World Data

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

- The Sloan Digital Sky Survey (SDSS) SkyServer traffic data
  This data set records all the access traffic to the SDSS SkyServer 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 preprocessing aggregates all the records for each second and places a zero in the time point entry if there is no activity within that second. The training data consists of seven days of second-by-second data.

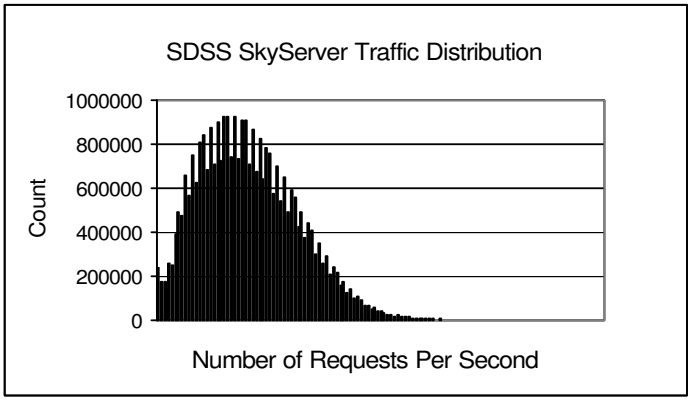
- The NYSE TAQ stock data
  This data set includes tick-by-tick trading activities of the IBM stock between Jan. 1st, 2001 to May 31st, 2004. There are a total of 6,134,362 ticks, and each record contains the time precise to the second, as well as each trade's price and volume. The preprocessing aggregates all the trading volumes within the same second and pads the second with a 0 if there is no activity within this second. A week's (5 day) worth of data is used as the training data.

Table 2 gives the basic statistics about these two data sets. Figure 17 shows the histograms of these two data sets. The histograms show the SDSS SkyServer traffic data follows the poisson distribution, while the IBM stock data is closer to the exponential distribution.

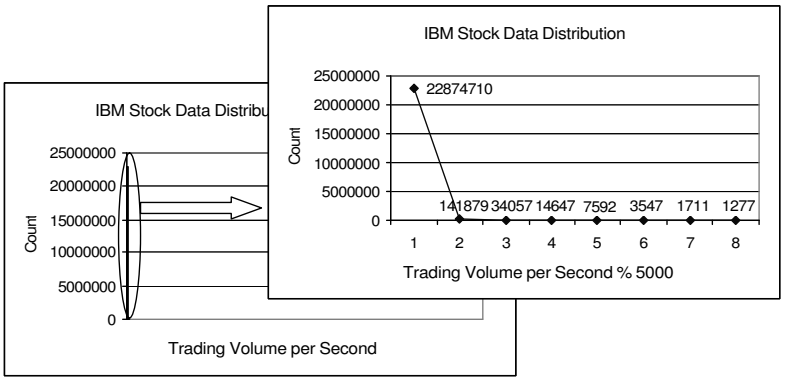
5.3.1 Performance Tests

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^{-2} to

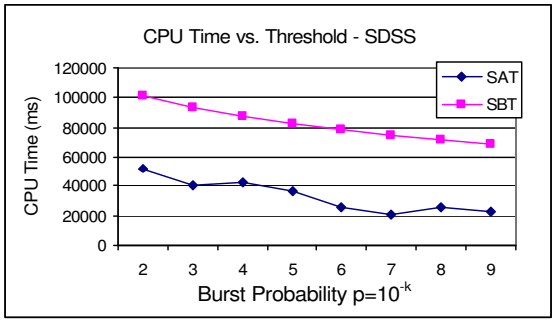


(a) SDSS

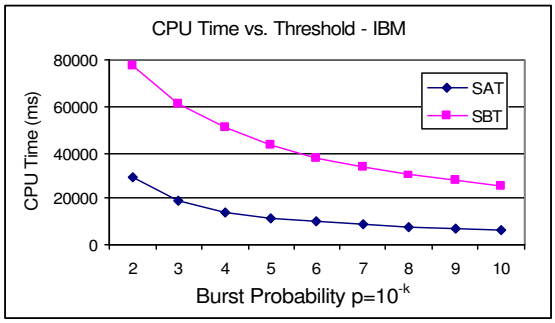


(b) IBM

Figure 17: The histogram distributions of the Sloan Digital Sky Survey (SDSS) SkyServer traffic data and the IBM stock data

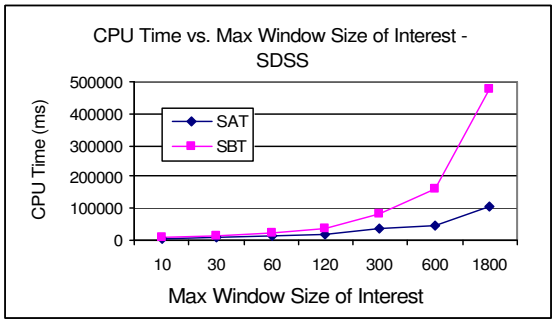


(a) SDSS

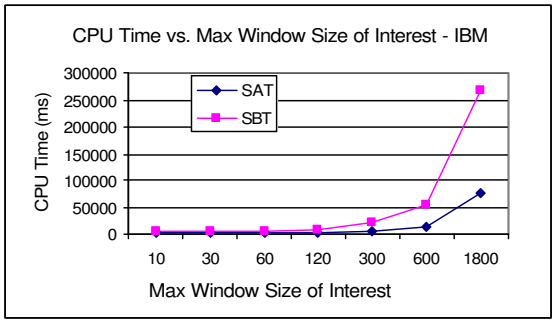


(b) IBM

Figure 18: The effect of the thresholds in the Sloan Digital Sky Survey (SDSS) data and the IBM data



(a) SDSS



(b) IBM

Figure 19: The effect of the maximum window size of interest in the Sloan Digital Sky Survey (SDSS) data and the IBM data

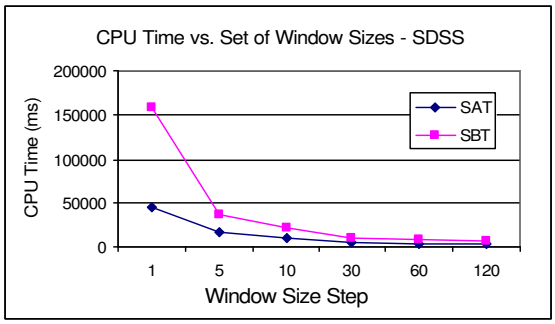
10^{-9}. The maximum window size is set to 300 for SDSS, 500 for IBM. Bursts at every window size are detected.

Figure 18 shows the results for both data sets. As the burst probability decreases, the CPU time for the Shifted Aggregation Tree decreases quickly.

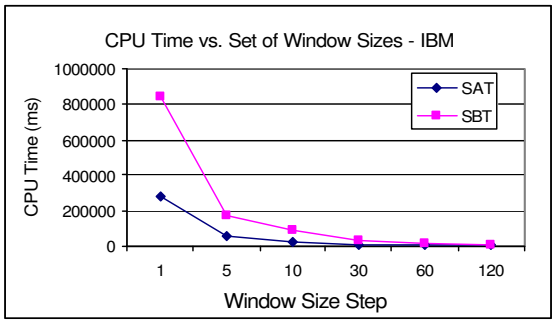
- Different maximum window sizes of interest
  The maximum window sizes are set from 10 seconds up to 1800 seconds. The burst probability is set to 10^{-6}. Bursts at every window size are detected.

  Figure 19.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 of interest
  Instead of detecting bursts at every window size, we want to see how the



(a) SDSS



(b) IBM

Figure 20: The effect of different sets of window sizes of interest in the Sloan Digital Sky Survey (SDSS) data and the IBM data

Table 3: Statistics for testing data sets for robust test

	Mean	StdDev		Mean	StdDev
IBM_w12	376.21	2147.10	SDSS_w10	73.92188	27.078788
IBM_w20	279.20	5430.01	SDSS_w18	98.841951	33.240921
IBM_w29	401.35	2307.30	SDSS_w25	121.711748	39.165514
IBM_w70	250.81	1716.69	SDSS_w40	168.854438	50.386027
IBM_w100	306.83	1760.76	SDSS_w45	183.680347	55.054213
IBM_w150	163.15	1184.95	SDSS_w50	197.57913	57.572911

Table 4: Testing parameters for robust test

	Max Window Size	Burst Probability	Window Size Step
IBM_setting1	250	10^{-3}	1
IBM_setting2	500	10^{-6}	5
IBM_setting3	750	10^{-7}	10
IBM_setting4	1000	10^{-8}	20
SDSS_setting1	200	10^{-4}	1
SDSS_setting2	400	10^{-5}	5
SDSS_setting3	600	10^{-6}	10
SDSS_setting4	800	10^{-8}	20

Shifted Aggregation Tree performs with different sets of window sizes. The test is performed to detect bursts for a set of window sizes $n, 2*n, 3*n, ...$, where 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 20 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.

In summary, the Shifted Aggregation Tree yields about 2 to 5 times speedup over the Shifted Binary Tree on these two data sets. Also it shows even the data points come every millisecond, it takes far less than one millisecond on average to process a new datum under these settings. Thus the Shifted Aggregation Tree is suitable for the real data stream environment.

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 SkyServer traffic 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 training set contains 3 pieces of training data, each piece

Table 5: Statistics for testing data sets for search parameters

	Mean	StdDev	Max Size	Size Step	Burst Probability
IBM_w20	242.94	2183.56	250	10	10^{-3}
IBM_w49	192.90	5599.56	100	5	10^{-5}
IBM_w54	385.55	4041.39	500	1	10^{-6}
IBM_w87	418.73	2485.27	750	15	10^{-7}
IBM_w138	228.03	1102.72	1000	25	10^{-4}
SDSS_w10	70.16	26.36	100	5	10^{-4}
SDSS_w25	117.02	37.73	250	1	10^{-6}
SDSS_w30	135.41	42.87	500	10	10^{-5}
SDSS_w40	166.19	49.97	750	15	10^{-7}
SDSS_w50	195.27	57.05	1000	25	10^{-8}

contains one week's record (7 days for SDSS and 5 days for IBM). Figure 21 shows the CPU times for four different testing scenarios on each training set.

Table 3 shows the statistics for the training data. In the table, the first three sets are taken from the data to be detected, the second three sets are taken outside the data to be detected. Table 4 shows other parameters in these tests.

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.

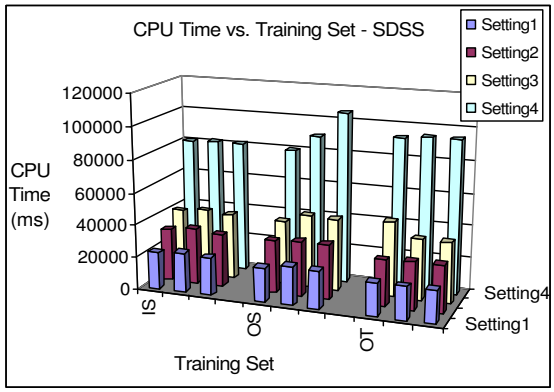
A structure based on a different data type can perform quite poorly. For example, a structure based on SDSS data performs by a factor of 2 to 3 times slower for IBM data than a structure based on out-of-sample IBM data.

5.3.3 Search parameter in the state-space algorithm

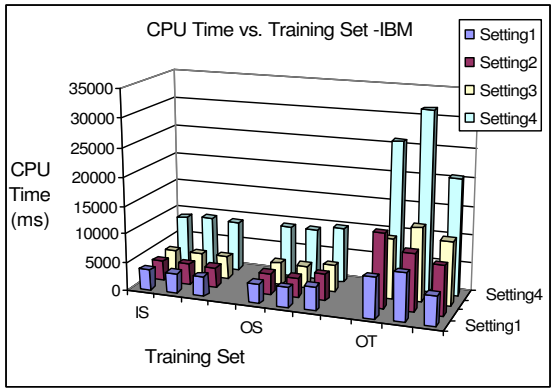
We want to study how different search parameters affect the desired Shifted Aggregation Tree structures in the state-space algorithm. We tested on different data settings with different numbers of final states and different number of states with the same maximum window size, to see when there are diminishing returns to broaden the search.

The number of states with the same maximum window size, and the number of final states are set to be 10, 25, 50, 100, 250, 500, 750, 1000 respectively. Other parameters are chosen randomly to cover a broad parameter range. Table 5 summarizes the parameter settings. 5 pieces of training data are picked for the SDSS data and the IBM data respectively. Figure 22 shows the CPU running times for the Shifted Aggregation Trees found using these parameters. It also shows the CPU running time for the Shifted Binary Tree as a reference.

Both experiments show that even with small values of these parameters, the Shifted Aggregation Trees discovered are close to those discovered with much larger values of the parameters. The best-first search strategy works well in this

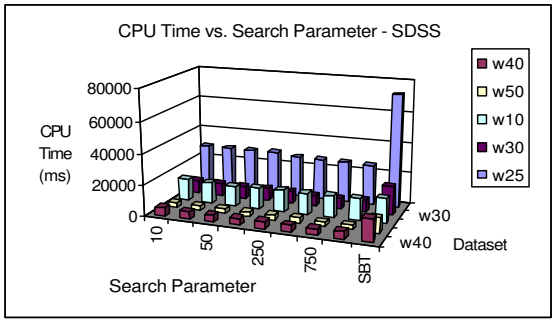


(a) SDSS

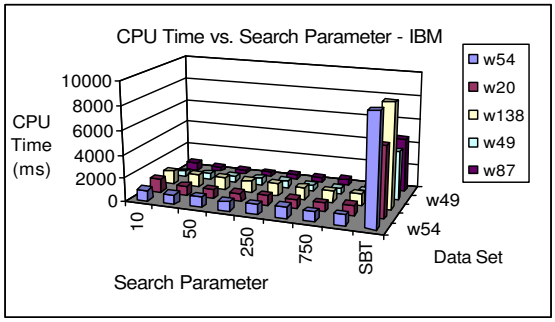


(b) IBM

Figure 21: Robustness Test on the Sloan Digital Survey (SDSS) traffic data and the IBM stock data (IS: in-sample, OS: out-of-sample, OT: out-of-type)



(a) SDSS



(b) IBM

Figure 22: CPU time for Shifted Aggregation Trees found using different search parameters

Table 6: Some highly-correlated stocks at different resolutions

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

situation. In practice, we believe that setting both parameters 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 6 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.

6 Related Work

6.1 Novelty, Anomaly, Surprise and Outlier Detection

As a task to detect unusual high numbers of events, burst detection belongs to a broader category of detection tasks: novelty/anomaly/outlier/surprise detection. The novelty/anomaly/outlier/surprise detection has been widely used in

fraud detection, network intrusion detection, financial analysis, health monitoring, etc.

Although intuitively novelty/anomaly/outlier/surprise/burst are straightforward concepts, attempted formal definitions are often vague and domain dependent. Following the classification for the outlier detection methods, the literature broadly falls into the following categories: depth-based [8], distribution-based [3, 9], distance-based [12, 11, 4, 2, 13], density-based [5, 21], and example-based [27, 20].

The depth-based method is based on computational geometry and computing layers of the k-d convex hull. Objects in the outer layer are identified as outliers.

In the distribution-based method, the data set is fit with a standard distribution, or a model/data structure associated with probabilities [9]. Outliers/surprises [9] are those points with small probability under this distribution model. In [10], a burst of high frequency word is defined as the frequency of the word usage is substantially higher than others, thus can be seen as a distribution-based method.

The distance-based method treats outliers [11, 13, 2, 4] as those points whose distances to their neighbors exceed some threshold. In [23], the surprise is defined as a large difference between two consecutive averages, which can be seen as a distance-based method. Spatial indexing techniques are usually used to speed up the distance computation.

The distance-based method can capture only the "global" outliers, since the threshold is usually determined globally. There is another type of outlier which is relative to its neighbors. The density-based method was proposed to overcome this shortcoming by defining the outliers as those points whose local densities are significantly different from their neighbors.

If we see the detection of outlier/novelty/anomaly as a classification problem, the classification technique in machine learning can be used to identify outliers/novelty. The Support Vector Machine (SVM) classifier has attracted more and more interest [15, 14, 22, 27]. In [27], Zhu et al. approach the outlier detection problem by learning how a user defines an outlier. The user manually identifies a small set of outliers based on their subjective criteria. A SVM classifier is learnt from the small amount of user feedback.

It has been recognized that instead of classifying a point/pattern as either an outlier/surprise or a normal point, it is better to associate some fuzzy degree to the outlier/surprise. This fuzzy degree describes how confident the classification is and how likely it's an outlier/surprise [5, 21, 14].

Our definition of burst is simply a large number of events exceeding some certain threshold, which is widely used in many real world applications.

6.2 Burst Modeling and Detection

Among the many topics in time series and data streams, burst modeling and detection is attracting increasing interest. There are several papers to study bursts under different settings.

Wang et al. [26] use a one-parameter model, b-model, to model the bursty behaviors in self-similar time series and synthesize realistic trace data. This type of time series includes a large number of real world data applications, such as Ethernet, file system, web, video and disk traffic, etc. Different from those

which are usually modeled by a Poisson process, these series are self-similar over different time scales and exhibit significant burstiness. They follow the "80/20 law" in databases: 80% of the queries access 20% of the data. The bias parameter b is used to model the bias percentage of the activities, i.e. 80% or 60%. The bias is applied recursively to a segment of series, (starting from a uniformed distribution) to synthesize a trace, i.e. b% of the segment has more activities than the rest of the segment. The entropy is used to describe the burstiness and to fit the model into the training data. The synthetic traces generated from this model are very realistic compared to real data.

Kleinberg [10] studies the bursty and hierarchical structure in temporal text streams. The interest is to find how high frequency words change over time. The word usage in many text streams, such as email, news articles and research publications, usually exhibits some bursty and hierarchical behaviors. During a certain duration, some words appear more frequently than others and the frequencies change over time. He assumed the gaps between two consecutive messages follow an exponential distribution, and used infinite-state automaton to model the different levels of burstiness in different time scales. The burstiness of words is defined as those words with significantly higher frequency than others.

While Wang et al. [26] and Kleinberg [10] focus the bursty behaviors and modeling, our focus is a high-performance algorithm to detect bursts thus complementing their work. Once the bursty structure is modeled and the criteria to identify a burst is determined, our framework can adapt to the input to achieve high-performance detection.

Neill et al. [18, 19, 17] study the problem of detecting significant spatial clusters in multidimensional space. The significant spatial clusters are defined as a square region (extended to a rectangular region in the later papers) with the highest density. They consider a general density function which is a function of the count and the underlying population. The density function could be non-monotonic. They are only interested in the region with the highest density, while our work detects all regions exceeding some threshold. A top-down, branch-and-bound method is used together with the so-called overlap-kd tree, to prune impossible regions. An overlap-kd tree is a hierarchical space-partition data structure where adjacent regions partially overlap. It's similar to the Shifted Binary Tree in that both data structures have adjacent windows/regions overlapping and the overlapping patterns are fixed and independent of the input data. Our technique could be applied to their data structure, an area meriting further investigation.

Vlachos et al. [25] 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. [6, 7] 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.

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] Market volume analysis. http://marketvolume.com/.

[2] 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.

[3] V. Barnett and T. Lewis. Outliers in statistical data. 1994.

[4] 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.

[5] 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.

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

[7] 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.

[8] T. Johnson, I. Kwok, and R. Ng. Fast computation of 2-dimensional depth contour. Proceedings of the 4th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 1998.

[9] Eamonn Keogh, Stefano Lonardi, and William 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. In *KDD '02: Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 91–101, New York, NY, USA, 2002. ACM Press.

[11] Edwin M. Knorr, Raymond T. Ng, , and Vladimir 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. Algorithms for mining distance-based outliers in large datasets. Proceedings of 24th International Conference on Very Large Data Bases, 1998.

[13] 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.

[14] Junshui Ma and Simon Perkins. Online novelty detection on temporal sequences. KDD, 2003.

[15] Junshui Ma and Simon Perkins. Time series novelty detection using one-class support vector machines. IJCNN, 2003.

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

[17] 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.

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

[19] Daniel B. Neill, Andrew W. Moore, Francisco Pereira, and Tom Mitchell. Detecting significant multidimensional spatial clusters. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 969–976, Cambridge, MA, 2005. MIT Press.

[20] Spiros Papadimitriou and Christos Faloutsos. Cross-outlier detection. 8th International Symposium on Spatial and Temporal Databases, 2003.

[21] Spiros Papadimitriou, Hiroyuki Kitagawa, Phillip B. Gibbons, and Christos Faloutsos. Loci: Fast outlier detection using the local correlation integral. Proceedings of the 19th International Conference on Data Engineering, 2003.

[22] Bernhard Scholkopf, Robert Williamson, Alex smola, John Shawe-Tayleor, and John Platt. Support vector method for novelty detection. Advances in Neural Information processing Systems 12, 1999.

[23] Cyrus Shahabi, Xiaoming Tian, and Wugang Zhao. Tsa-tree: A wavelet-based approach to improve the efficiency of multi-level surprise and trend queries on time-series data. 12th International Conference on Scientific and Statistical Database Management(SSDBM), 2000.

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

[25] 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.

[26] Mengzhi Wang, Tara Madhyastha, Ngai Hang Chan, Spiros Papadimitriou, and Christos Faloutos. Data mining meets performance evaluation: Fast algorithms for modeling bursty traffic. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering (ICDE'02)*, pages 507–516, Washington, DC, USA, 2002. IEEE Computer Society.

[27] Cui Zhu, Hiroyuki Kitagawa, Spiros Papadimitri, and Christos Faloutsos. Obe: Outlier by example. PAKDD, 2004.