Paper No: 146

Adaptively Detecting Aggregation Bursts in Data Streams

Shouke Qin, Weining Qian, Aoying Zhou
Dept. of Computer Science and Engineering
Fudan University
220 Handan RD, Shanghai, 200433, China
Tel: +86-(0)21-6564-3024
Fax: +86-(0)21-6564-3503
email: skqin,wnqian,ayzhou@fudan.edu.cn

Abstract. Finding bursts in data streams is attracting much attention in research community due to its broad applications. Existing burst detection methods suffer the problems that 1) the parameters of window size and absolute burst threshold, which are hard to be determined a priori, should be given in advance. 2) Only one side bursts, i.e. either increasing or decreasing bursts, can be detected. 3) Bumps, which are changes of aggregation data caused by noises, are often reported as bursts. The disturbance of bumps causes much effort in subsequent exploration of mining results. In this paper, a general burst model is introduced for overcoming above three problems. We develop an efficient algorithm for detecting adaptive aggregation bursts in a data stream given a burst ratio. With the help of a novel inverted histogram, the statistical summary is compressed to be fit in limited main memory, so that bursts on windows of any length can be detected accurately and efficiently on-line. Theoretical analysis show the space and time complexity bound of this method is relatively good, while experimental results depict the applicability and efficiency of our algorithm in different application settings.

– **Key Word 1**: Data Streams
– **Key Word 2**: Data Mining and Knowledge Discovery
– **Add_Keyword**: Burst Detection
– **Author List**: Shouke Qin, Weining Qian and Aoying Zhou

Adaptively Detecting Aggregation Bursts in Data Streams

Shouke Qin, Weining Qian, Aoying Zhou

Department of Computer Science and Engineering
Fudan University, 220 Handan Rd, Shanghai, China
{skqin,wnqian,ayzhou}@fudan.edu.cn

Abstract. Finding bursts in data streams is attracting much attention in research community due to its broad applications. Existing burst detection methods suffer the problems that 1) the parameters of window size and absolute burst threshold, which are hard to be determined a priori, should be given in advance. 2) Only one side bursts, i.e. either increasing or decreasing bursts, can be detected. 3) Bumps, which are changes of aggregation data caused by noises, are often reported as bursts. The disturbance of bumps causes much effort in subsequent exploration of mining results. In this paper, a general burst model is introduced for overcoming above three problems. We develop an efficient algorithm for detecting adaptive aggregation bursts in a data stream given a burst ratio. With the help of a novel inverted histogram, the statistical summary is compressed to be fit in limited main memory, so that bursts on windows of any length can be detected accurately and efficiently on-line. Theoretical analysis show the space and time complexity bound of this method is relatively good, while experimental results depict the applicability and efficiency of our algorithm in different application settings.

1 Introduction

Detecting bursts robustly and efficiently poses a challenge in many applications of online monitoring for data streams, such as telecommunication networks, traffic management, trend-related analysis, web-click streams analysis, intrusion detection, and sensor networks. Many methods have been proposed to detect bursts or changes in a time period, called a window, in E-mail [10], Gamma ray [13] and networks traffic [3, 11] data streams. However, these methods are not adaptive enough for many real-life applications, since they need fixed parameter setting for window size and absolute threshold of bursting. Furthermore, only one-side bursts can be detected by these methods.

We argue that ratio threshold for bursting measurement and adaptive window size is more suitable for data stream analysis applications. In network traffic monitoring, for example, when attaches occur, the workload of package routing to an IP address or requests to a server within a certain time period may increase remarkably compared with last certain time period. An absolute threshold may cause false report of bursts in a rush time and missing of attacks when the

workload is low. Furthermore, the lasting time varies in different kinds of attacks, which are all interested by monitors. Fig.1 shows the double-side bursts with various peak height and window size.

Though, a natural solution for adaptively detecting aggregation bursts is to applying existing algorithms on different window sizes, peak heights, and both increasing and decreasing side simultaneously, apparently it may consume much storage and computation resources. Furthermore, as any other data stream mining tasks, it is required that burst detection algorithm should be accurate, while use limited storage space and scan a data stream sequentially only once. We present a novel method for adaptively detecting aggregation bursts in data streams. It relies on the efficient algorithms for construction and maintenance of a compact summary data structure, called *inverted histogram*, or IH. We show that the problem of burst detection can be transformed to linearly scan of the histogram. Furthermore, it is proved that the bucket error of the histogram is bounded, while the space and computation complexity for maintaining the histogram is low.

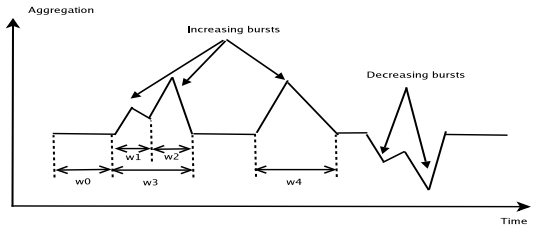


Fig. 1. Bursts with various peak height and window size. In the stream, data in window w_0 are not burst, while data in w_1 are a small burst, compared with those in w_0. Data in w_2 are another burst compared with those in w_1. Furthermore, w_3 and w_4 depict another two bursts, whose window sizes are larger than bursts w_1 and w_2.

Bursts found by existing approaches at large time scales are not necessarily reflected at smaller time scales. That is because those bursts at large time scales are composed of many consecutive *bumps* which are those positions where the values are high but not high enough to be bursts. In this paper, a general burst model is introduced for overcoming the problem.

1.1 Related Work

Monitoring and mining data stream has attracted considerable attention recently [10, 13, 3, 11, 2]. An efficient work for statically detecting bursts in data stream with normal distribution based on sliding window is presented in [13]. Kleinberg focuses on modelling and extracting structure from text stream in [10]. There is also work in finding changes on data stream [3, 11, 2]. Although, they have different objectives and different application backgrounds, the same thing is that

they can only monitor one window with a given size. The method in [13] can monitor multiple windows with different sizes. But, the maximum window size and window number is limited.

Some of the above approaches detect bursty behaviors of aggregate results [13, 3, 11]. The amount of streaming data is too large to maintain in main memory. This prompts a need to maintain a synopsis of the stream to all the methods. The work in [3, 11] is based on sketch. Haar wavelets can also be used to compute aggregates [7], and their error are bounded in [5]. The burst detecting method in [13] is based on wavelets. However, it can only monitor the monotonic aggregates with respect to the window size, such as *sum*. The non-monotonic ones, such as *average*, cannot be monitored with it.

Histograms have been used widely to capture data distribution, to present the data by a small number of buckets. They are usually used in selectivity estimation. An excellent survey of the history of histograms can be found in [9]. V-optimal histograms can be maintained with dynamic programming algorithms which running in quadratic time and linear space. Over data stream model, the approximate V-optimal histograms can support the point queries, aggregate queries and range sum queries [8, 6, 12]. If the stream is ordered, namely the values of data elements are non-negative, the work of [8] has the cheapest space $O(\frac{k^2 \log nR^2}{\epsilon})$ and time $O(\frac{nk^2 \log nR^2}{\epsilon})$. It provides an $(1+\epsilon)$ approximation for the V-optimal histogram with k levels, n is the length of stream and R is the maximum value of data elements.

1.2 Our Contributions

Our contributions can be summarized as follows:

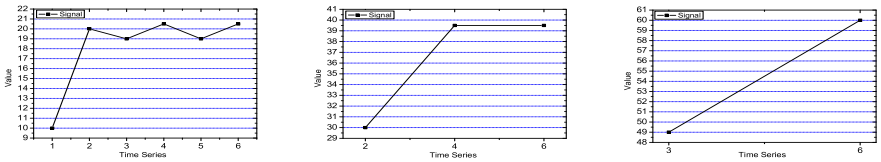
- First, we put forward a novel definition of burst. To the best of our knowledge, this is the first work considering such adaptive burst model on data stream. This model is more general and fit for the real world applications.
- Second, we design both false positive and false negative algorithms for finding bursts accurately in high speed data streams. They can detect bursts dynamically with double side alarm domain and avoid being disturbed by bumps on overall stream.
- Third, we propose a novel histogram—IH, with relative error guaranteed in each bucket, which can answer burst queries accurately with very cheap cost of space and time. We note that IH is an interesting data structure in its own right and might find applications in other domains.
- Fourth, we complement our analytical results with an extensive experimental study. Our results indicate that the new method can indeed detect the bursts accurately within an economic space and time cost.

The remainder of this paper is organized as follows. Section 2 presents the problem and definition of the paper. Section 3 presents our algorithms for detecting bursts. Section 4 presents a novel histogram which is the base of burst detecting algorithms. Section 5 show the results and analysis of experiments. Section 6 concludes the paper.

2 Problem Statements

In this section, we present the model and definition of the problem for clarifying the objectives in this paper.

As described in the introduction, bumps can induce bursts at large time scales. We can show it through a small example. In Fig.2.(a), the current stream length is 6. x_6 is a new comer and its value is 20.5. The relative threshold is denoted by β, which is a positive ratio. Here, aggregate function F is *sum* and $\beta = 1.1$. From Fig.2.(a), x_4 and x_6 are all bumps. They cannot induce burst on the 1-length window, for $x_4 = 20.5 < \beta x_3 = 1.1 * 19 = 20.9$ and $x_6 = 20.5 < \beta x_5 = 1.1 * 19 = 20.9$. But they induce bursts on 2-length and 3-length window respectively, such as Fig.2.(b) and Fig.2.(c), for $(x_3 + x_4) > \beta(x_1 + x_2)$ and $(x_4 + x_5 + x_6) > \beta(x_1 + x_2 + x_3)$. In that, if burst does not occur in window



(a) Window Length = 1 (b) Window Length = 2 (c) Window Length = 3

Fig. 2. Bumps Can Induce Bursts at Large Time Scales

L(window length), window L+1 should not be detected. Because remnant bursts, if had, are all induced by bumps.

A data stream can be considered as a sequence of points $x_1, ..., x_n$ in increasing order. Each element in stream can be a value in the range $[0..R]$. To detect bursts on overall data stream needs only caring about the latest two consecutive subsequences with the same length when each new value x_i comes. The formal definition of adaptively detecting bursts on data stream model is shown as follows.

Definition 1. *F is an aggregate function. s'_i and s_i are the latest two consecutive subsequences of a stream with the same length of i, $1 \leq i \leq \frac{n}{2}$. An increasing burst occurs on i-length window when x_n comes, if $\forall j \in 1..i$, $\beta > 1$, $F(s_j) \geq \beta F(s'_j)$. An decreasing burst occurs on i-length window when x_n comes, if $\forall j \in 1..i$, $0 < \beta < 1$, $F(s_j) \leq \beta F(s'_j)$.*

In the example of bumps above, as Definition 1, $s'_1 = x_5$, $s_1 = x_6$, $s'_2 = x_3, x_4$, $s_2 = x_5, x_6$ and $s'_3 = x_1, x_2, x_3$, $s_3 = x_4, x_5, x_6$.

3 Algorithms for Detecting Bursts

We begin to present our algorithms for detecting bursts in this section. These algorithms can be based on the approximate V-optimal histogram in [8](AVOH in short) or IH. IH is a novel histogram proposed by us whose details are presented in section 4. Due to limited space allowed to detect bursts on data stream, approximate burst detecting can take two possible approaches, namely, false positive oriented and false negative oriented. The former includes some unreal bursts in the final result, whereas the latter misses some real bursts. In the real world, different application requirements need false positive oriented algorithm and false negative oriented algorithm respectively.

3.1 False Positive Algorithm

The aggregate result w_i of the latest i-length window ($1 \leq i \leq n$) can be got from AVOH or IH. In fact, $F(s_i) = w_i = f_{ss}(i)$ and $F(s_i') = w_{2i} - w_i = f_{ss}(2i) - f_{ss}(i)$. $f_{ss}(i) = \Sigma_{j=i}^{n} x_j$, it is the suffix sum which is the sum of the last $n-i+1$ values of stream x when x_n comes. Therefore, the criterions of increasing burst and decreasing burst in Definition 1 can be depicted by $w_i \geq \beta(w_{2i} - w_i)$ ($\beta > 1$) and $w_i \leq \beta(w_{2i} - w_i)$ ($0 < \beta < 1$) respectively. They can also be transformed to

$$w_{2i} \geq (\beta + 1)(w_{2i} - w_i), (\beta > 1) \tag{1}$$

and

$$w_{2i} \leq (\beta + 1)(w_{2i} - w_i), (0 < \beta < 1) \tag{2}$$

Detecting bursts false positively is achieved by augmenting the left side of inequality (1) and the right side of inequality (2) relatively. This is fulfilled by two functions. One is IH.getLargerValue(i) which returns a value larger than real w_i from IH. The other is IH.getSmallerValue(i) which returns a value smaller than real w_i from IH. It can be seen that the accuracy of our method is only affected by these two functions. Their details and bounds are introduced in section 4.

When x_n comes and is inserted into histogram, this algorithm is called for detecting bursts induced by x_n. The algorithm has no input parameter. Its output is the alarm information. Here, we just return the number of bursts. The algorithm behaves level wise detection from level 1 to level $\frac{n}{2}$. Each level is a pair of the latest two consecutive windows with the same length on stream x. In level i, the algorithm detects burst between such two windows with length i. That is accomplished by two steps. The first step is to compute w_i and w_{2i} with the two functions above, just as the statements in Line 4 and Line 5. The second step is to detect the occurrence of a burst by inequality (1) as the statement in Line 6. The level wise detection is broken at Line 9 when inequality (1) is not met, for remnant bursts are all induced by bumps. Because of different applications oriented, there are at least three algorithms can be shown. First one is to detect increasing bursts only. Second one is to detect decreasing bursts only. Third one is to detect both two kinds of bursts. For the limitation of space, we show only the first algorithm to which the others are similar.

Algorithm 1 detectBurstsFalsePositively

```
1: burstNum ← 0;
2: winSize ← 1;
3: while winSize ≤ n/2 do
4:    temp1=IH.getLargerValue(winSize);
5:    temp2=IH.getSmallerValue(2 ∗ winSize);
6:    if temp2 ≥ (β + 1)(temp2-temp1) then
7:       increase burstNum and winSize by 1;
8:    else
9:       break;
10:   end if
11: end while
12: return burstNum;
```

Algorithm 1 can catch all bursts induced by x_n on overall stream in $O(k)$ time, $k(0 \leq k \leq \frac{n}{2})$ is the number of bursts found by it. Therefore, it takes time $O(nk)$ for detecting all bursts in a stream with length n. If we just want to know whether one burst occurs or not when x_n comes instead of on which window burst occurs or how many bursts are induced on overall stream, the time cost can be $O(1)$. We can claim the following theorem by denoting the space cost of a B-bucket histogram with B and its updating cost for each x_i with T_B.

Theorem 1. *Algorithm 1 can detect bursts false positively on data stream in* $O(n(T_B + k))$ *time and* $O(B)$ *space.*

3.2 False Negative Algorithm

Similar to the analysis in the above section, detecting bursts false negatively is achieved by abating the left side of inequality (1) and the right side of inequality (2) relatively. Algorithm 1 can be capable of detecting bursts false negatively when given minor modification. We can just put the statements, temp1=IH.getSmallerValue($winSize$) and temp2=IH.getLargerValue($2*winSize$), instead of Line 4 and Line 5 in Algorithm 1. The analysis of its cost is same as that of Algorithm 1. We can claim the following theorem.

Theorem 2. *We can detect bursts false negatively on data stream in* $O(n(T_B + k))$ *time and* $O(B)$ *space.*

It is clear that in addition to *sum*, the two algorithms above can monitor not only monotonic aggregates with respect to the window size for example *max*, *min*, *count* and *spread*, but also non-monotonic ones such as *average*.

4 Buckets Order Inverted Histogram—IH

In this section, we begin to introduce a novel histogram, called Inverted Histogram (i.e., IH in brief). It is the base of burst detecting algorithms presented

in section 3. At the beginning, a simple histogram is introduced. It is better than the existing approximate V-optimal histograms for its cheap space and time when being used to answer burst queries. But both the existing approximate V-optimal histograms and the simple histogram are facing the great challenge that the absolute error in buckets created recently are getting larger and larger. The everlasting increasing of absolute error in buckets will decay the accuracy of burst detection rapidly. Later, we introduce the enhanced histogram—IH, which not only has cheap space and time, but also answers burst queries precisely.

4.1 A Simple Histogram

Each point of stream $x'_1, ..., x'_n$ we read can be thought of the prefix sum of $x_1, ..., x_n$ with $x'_i = f_{ps}(i)$. $f_{ps}(i)$ is the prefix sum of stream x when x_i comes, $f_{ps}(i) = \Sigma_{j=1}^{i} x_j$. Details of our idea are as follows. What we want is to partition stream x' into B intervals(buckets), $(b_1^a, b_1^b), ..., (b_B^a, b_B^b)$. b_i is the i-th bucket, $1 \leq i \leq B$. b_i^a and b_i^b are the minimum and maximum value within bucket b_i respectively. It's possible that $b_i^a = b_i^b$. We also want to bound the relative error δ in each bucket. The maximum relative error in b_i is $\delta = \frac{b_i^b - b_i^a}{b_i^a}$. One bucket should maintain b_i^a, b_i^b and the number of values in it, namely, its width. Furthermore, the buckets are disjoint and cover $x'_1..x'_n$. Therefore, $b_1^a = x'_1$ and $b_B^b = x'_n$. During the processing, $b_i^b \leq (1+\delta)b_i^a$ is maintained. The algorithm on seeing the n'st value x_n, will compute $x'_n = f_{ps}(n) = b_B^b + x_n$. We just have to update the last bucket, either by setting $b_B^b = x'_n$ when $x'_n \leq (1+\delta)b_B^a$, or creating a new bucket when $x'_n > (1+\delta)b_B^a$, with $b_{B+1}^a = b_{B+1}^b = x'_n$. The algorithm is quite simple. It will not be shown here. The proof of Theorem 3 is in Appendix A.

Theorem 3. *We can build the simple histogram with* $O(\frac{\log n + \log R}{\log(1+\delta)})$ *space in* $O(n)$ *time. The relative error in each bucket is at most* δ.

Although the simple histogram can be built with cheap cost of time and space, both the approximate V-optimal histogram and the simple histogram have same defect in detecting bursts. That is the absolute error and size of the last bucket are getting larger and larger as the increasing of stream length. Thus, the errors induced by using these buckets to compute aggregates are also increasing. When a new value x_n comes, we use the bucket from the last one to the first one to estimate w_i from 1-length window to $\frac{n}{2}$-length window. Therefore, the errors of aggregates on small windows are much larger than those on large windows. According to Definition 1, window $L+1$ is not detected, if there is no burst occurs in window L. Provided that the aggregate of window L is computed falsely, the bursts of windows whose sizes are larger than L may be neglected. In consequence, the greater the stream length is, the more error the detecting methods will suffer. In the later of this paper, the analysis can be validated from experiments. We present IH in next section which overcomes this defect.

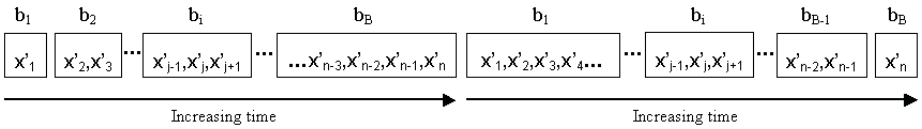
Algorithm 2 updateIH(x_n)

```
1: increase bucket number B by 1;
2: j ← B;
3: create a new bucket and put x_n into it;
4: for i = B − 1 to 1 do
5:    add x_n to both b_i^a and b_i^b;
6:    if b_i^b ≤ (1 + δ)b_j^a then
7:       b_j^b ← b_i^b;
8:       add width of b_i to width of b_j;
9:       delete b_i;
10:      decrease bucket number B by 1;
11:   end if
12:   decrease j by 1;
13: end for
```

4.2 Enhanced Histogram—IH

Let us reconsider the size of the latest bucket in the above two histograms. They are getting large when stream length increases. However, we hope the recent bucket has higher precision, in other words, the recent bucket has smaller width.

Our idea is to invert the buckets order which uses the smaller bucket to store the newer points, such as in Fig.3.(b). The oldest point x'_1 and the latest point x'_n of stream x' are in b_1 and b_B respectively. In Fig.3.(a), $x'_i = f_{ps}(i)$. In Fig.3.(b), $x'_i = f_{ss}(i)$. $f_{ss}(i)$ is the suffix sum, $f_{ss}(i) = \Sigma_{j=i}^{n} x_j$. Fig.3.(a) is the bucket series of our simple histogram and approximate V-optimal histogram. The size of the last bucket in Fig.3.(a) is getting larger and larger as time goes on.



(a) Normal Buckets Order (b) Inverted Buckets Order

Fig. 3. Buckets Orders of Approximate V-optimal Histogram and IH

Same as the simple histogram, we want to minimize the bucket number with relative error guaranteed in each bucket. The only difference is the stream we read can be thought of as $x'_i = f_{ss}(i)$. Details of our idea is similar to that of the simple histogram. IH can be built with following algorithm.

Algorithm 2 has an input parameter x_i and no output. On seeing a new value x_n, it has to update all buckets, namely, $O(\frac{\log n + \log R}{\log(1+\delta)})$ buckets by executing the statements from Line 4 to 13. First, at Line 3, it creates a new bucket for x_n and puts the bucket on the last position of buckets series. Then, at Line 5, it updates the maximum and minimum value of all the buckets by adding x_n to

them from the new created one to the oldest one. In the process of updating, from Line 6 to 11, the algorithm merges consecutive buckets b_i and b_j when $b_i^b \leq (1+\delta)b_j^a$, with $b_j^b = b_i^b$. In fact, IH can also be constructed by the simple histogram algorithm feeded with inverted data stream $x_n, .., x_1$. That is a very nice situation, the maximum relative error in each bucket of IH can be bounded and the space and time cost are still cheap. We claim the following theorem to guarantee these. The proof of it is in Appendix B.

Theorem 4. *Algorithm 2 can build an IH with* $O(\frac{\log n+\log R}{\log(1+\delta)})$ *space in* $O(\frac{n(\log n+\log R)}{\log(1+\delta)})$ *time. The relative error in each bucket is at most* δ*.*

As described in section 3, two functions are called for getting w_i. Provided that the real value of w_i is within bucket b_j, IH.getLargerValue(i) returns b_j^b and IH.getSmallerValue(i) returns b_j^a. The relative error of their return values is bounded by δ. The precision of them can be improved by considering the values within a bucket are equidistant. To guarantee false positive or false negative detection, we need to maintain $maxD$ and $minD$ within each bucket. $maxD$ is the maximum distance between two consecutive point within the bucket. $minD$ is the minimum distance between two consecutive point within the bucket. Provided that the real value of w_i is within bucket b_j, IH.getLargerValue(i) returns the value of $min(b_j^b, b_j^a + maxD(i - \Sigma_{k=1}^{i-1} Wid(b_k) - 1))$ and IH.getSmallerValue(i) returns $b_j^a + minD(i - \Sigma_{k=1}^{i-1} Wid(b_k) - 1)$. Based on Theorem 1, Theorem 2 and Theorem 4, we can get the following corollary.

Corollary 1. *We can detect bursts false positively or false negatively on data stream in* $O(n(\frac{\log n+\log R}{\log(1+\delta)} + k))$ *time and* $O(\frac{\log n+\log R}{\log(1+\delta)})$ *space.*

5 Performance Evaluation

In this section, our empirical studies show the adaptive method in this paper can efficiently give accurate alarms to bursts with just a small cost of space.

5.1 Experimental Setup

All the histogram constructing algorithms and burst detecting algorithms are implemented by using Microsoft Visual C++ Version 6.0. We conducted all testing on a 2.4GHz CPU Dell PC with 512MB main memory running Windows 2000. Our algorithms are tested on a variety of data sets. Due to the limitation of space, we report the results for only two representative data sets here:

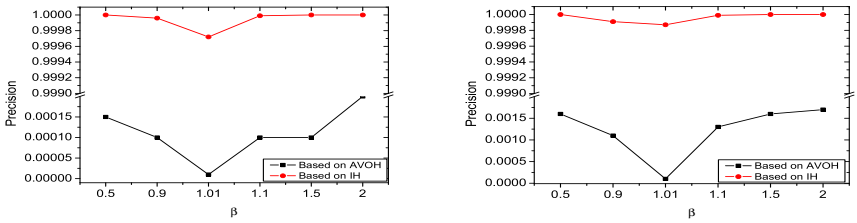
– Web Site Requests (Real): This data set is obtained from the Internet Traffic Archive [1, 3]. It consists of all the requests made to the 1998 World Cup Web site between April 26, 1998 and July 26, 1998. During this 92 days period of time the site received 1,352,804,107 requests. Our basic window, namely, an item x_i is the requests number in one second. So, the stream length is $n = 92 * 24 * 3600 = 7,948,800s$. It is denoted by D1.

– Network Traffic (Synthetic): This data set is generated with the burst arrival of data stream using a pareto[4] distribution, which is often used to simulate network traffic where packets are sent according to ON OFF periods. The density function of pareto distribution is $P(x) = \frac{ab^a}{x^{a+1}}$, where $b \geq x$ and a is the shape parameter. The expected burst count, $E(x)$, is $\frac{ab}{a-1}$. The tuple arrive rate λ_1 is driven by an exponential distribution and the interval λ_2 between signals is also generated using exponential distribution. In this data set, we set expect value $E(\lambda_1) = 400 tuples/s, E(\lambda_2) = 500 tuples, a = 1.5, b = 1$. The size of this time series data set is $n = 10,000,000s$. It is denoted by D2.

In the experiments, two accuracy metrics are measured: recall and precision. Recall is the ratio of true alarms raised to the total true alarms should be raised. Precision is the ratio of true alarms raised to the total alarms raised. Our aggregate function F is sum.

5.2 Performance Study

Precision of burst detection based on IH and AVOH We study the precision of detecting bursts false positively based on IH and AVOH on D1 and D2 respectively. In this experiment, we set $\delta = 0.01$. It can be confirmed from

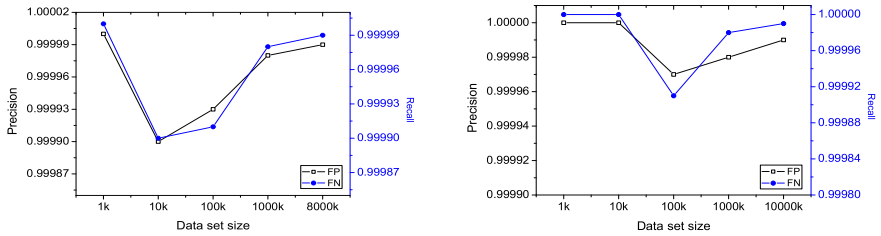


(a) varying β on D1 (b) varying β on D2

Fig. 4. Precision of burst detection based on IH and AVOH varying β ($\delta = 0.01$)

Fig.4 that the burst detecting method based on IH is far more accurate than that based on AVOH at any setting of threshold β.

Precision and Recall of Adaptively Burst Detecting We test the precision and recall of both false positive algorithm(FP in short) and false negative algorithm(FN in short) on D1 and D2 respectively. In this experiment, we set $\beta = 1.1, \delta = 0.01$. It can be seen from Fig.5 that on both data sets the precision of FP are at least 99.99% and with recall guaranteed 1, and the recall of FN are also at least 99.99% and with precision guaranteed 1. In Fig.5.(a), we see

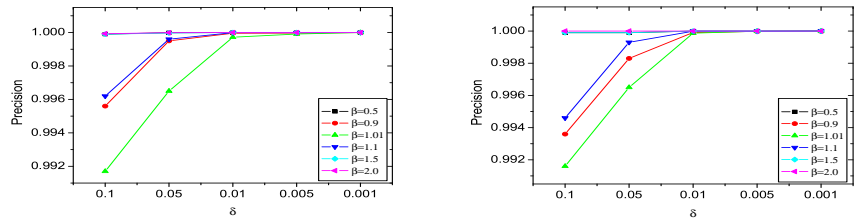


(a) varying data set size on D1 (b) varying data set size on D2

Fig. 5. Precision and Recall of FP and FN varying data set size ($\beta = 1.1, \delta = 0.01$)

the precision and recall of FP and FN are all 1 when stream length is 1k. When stream length is 10k, precision and recall are all 99.99% for occurring an error. Because the accuracy of burst detection does not decay with the increasing of stream length, the precision and recall are always above 99.99% and getting better as time goes on. The same result can be got from Fig.5.(b). Therefore, our method can give highly accurate answers to burst detection over streaming data.

How to Set an Appropriate δ to a β We discuss the setting of β and δ. From that, we can know the most appropriate setting of δ, if β have been set. It means with that setting of δ we can use the most economic space and time to find bursts accurately under relative threshold β. It can be seen from Fig.6 that

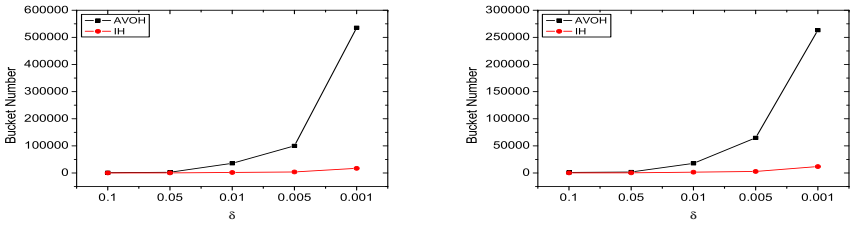


(a) varying δ on D1 (b) varying δ on D2

Fig. 6. Precision of detecting bursts false positively (varying δ and β, on D1 and D2)

the most adaptable setting of δ on each data set is $max(\frac{\beta-1}{10}, 0.01)$. The same heuristic result is also got from other experiments we have made.

Compare Space Cost of IH and AVOH We test the space cost of IH and AVOH on D1 and D2 respectively on condition that both of them have the same maximum relative error in their buckets. Here, we set $k = 1$ for AVOH and vary the maximum relative error δ in each bucket of both IH and AVOH from 0.1 to 0.001. In this experiment, β has no influence on results. It can be seen from

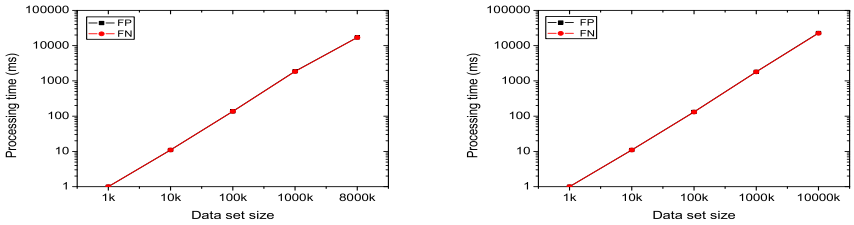


(a) varying δ on D1 (b) varying δ on D2

Fig. 7. Space cost of IH and AVOH (varying δ on D1 and D2)

Fig.7 that IH consumes less memory than AVOH on any condition of δ. The space saved by IH is getting larger and larger as the decreasing of δ. Therefore, our histogram is more adaptable to be used to detect bursts over streaming data.

Processing Time By varying size of data sets, we study the time cost of our algorithms on D1 and D2 respectively. In this experiment, we set $\beta = 1.1, \delta = 0.01$. Our method is very efficient. This is confirmed in Fig.8, where the processing



(a) varying data set size on D1 (b) varying data set size on D2

Fig. 8. Processing Time of FP and FN varying data set size ($\beta = 1.1, \delta = 0.01$)

time of FP and FN are same. The method of us can processing $400 * 10^7$ tuples

in 20 seconds. This means that it is easily capable of processing traffic rates on 100Mbs links, and with some work then 1Gbps and higher are within reach.

6 Conclusions and Future Work

In this paper, we studied the problem of detecting bursts in data streams. A novel concept of adaptive aggregation burst is introduced for precisely modelling real-life data streams. This burst model empowers us to detect double-side relative bursts dynamically without disturbed by bumps. We propose the effective algorithms for accurately detecting such bursts under false positive and false negative constraints. The algorithms are developed based on a space and time efficient histogram, IH, which can be maintained with low overhead while being more suitable for burst detection than other popular histograms. Intensive experiments on both synthetic and real-life data sets show that our method is quite efficient for analyzing high speed data streams. Furthermore, it is shown that IH based algorithm cost less space and time than using approximate V-optimal histogram. Future work includes the study of extension of current IH-based burst detection method on multiple correlated data streams and burst forecasting.

References

1. Internet traffic archive. http://ita.ee.lbl.gov/.
2. S. Ben-David, J. Gehrke, and D. Kifer. Detecting change in data streams. In *Proc. of VLDB*, 2004.
3. G. Cormode and S. Muthukrishnan. Whats new: Finding significant differences in network data streams. In *Proc. of INFOCOM*, 2004.
4. M. E. Crovella, M. S. Taqqu, and A. Bestavros. Heavy-tailed probability distributions in the world wide web. *A practical guide to heavy tails: STATISTICAL TECHNIQUES AND APPLICATIONS*, pages 3–26, 1998.
5. M. Garofalakis and P. B. Gibbons. Wavelet synopses with error guarantees. In *Proc. of SIGMOD*, 2003.
6. A. C. Gilbert and et al. Fast, small-space algorithms for approximate histogram maintenance. In *Proc. of STOC*, 2002.
7. A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proc. of VLDB*, 2001.
8. S. Guha, N. Koudas, and K. Shim. Datastreams and histograms. In *Proc. of STOC*, 2001.
9. Y. Ioannidis. The history of histograms. In *Proc. of VLDB*, 2003.
10. J. Kleinberg. Bursty and hierarchical structure in streams. In *Proc. of SIGKDD*, 2002.
11. B. Krishnamurthy, S. Sen, Y. Zhang, and Y. Chen. Sketch-based change detection: Methods, evaluation, and applications. In *Proc. of IMC*, 2003.
12. S. Muthukrishnan and M. Strauss. Rangesum histograms. In *Proc. of SODA*, 2003.
13. Y. Zhu and D. Shasha. Efficient elastic burst detection in data streams. In *Proc. of SIGKDD*, 2003.

A The Proof of Theorem 3

The proof of Theorem 3 is as follows.

Proof. Let us assume that the simple histogram needs to maintain at most B buckets for stream $x_1, .., x_n$. $x'_i = f_{ps}(i)$. Therefore, we have $x'_1(1+\delta)^{B-1} < x'_n$. Then, $B < \log_{1+\delta}(x'_n/x'_1)+1 = (\log x'_n - \log x'_1)/\log(1+\delta)+1$. x'_n can be at most nR where R is the maximum value of x_i. So, $B < (\log n + \log R - \log x'_1)/\log(1+\delta) + 1$. Furthermore, clearly, the algorithm takes an incremental $O(1)$ time per new value. These prove the theorem.

B The Proof of Theorem 4

The proof of Theorem 4 is as follows.

Proof. In fact here, we require at most $(\frac{2(\log n+\log R)}{\log(1+\delta)})$ buckets for stream $x(x_1, .., x_n)$. Because the process of merging consecutive buckets cannot minimize bucket number as the simple histogram. We assume a buckets series $b_1, .., b_m$, which is constructed by the simple histogram feeded with inverted data stream $x_n, .., x_1$. b_1 is the oldest bucket with $b_1^a = x_n$. b_m is the new created bucket with $b_m^b = \Sigma_{j=n}^1 x_j$. We assume another buckets series $c_1, .., c_l$ is constructed by Algorithm 2 feeded with data stream $x_1, .., x_n$. Then, the latter buckets series can be think of as a division and remerging of the former and $l \geq m$. A bucket b_i can be divided into three consecutive buckets at most in $c_1, .., c_l$. Furthermore, the first bucket and the third bucket must have been merged with other buckets. Namely, there is at most one bucket c_j in $c_1, .., c_l$ that $c_j \subseteq b_i$. Now, we assume there are two buckets $c_j, c_k (j < k)$ in $c_1..c_l$ that $c_j \subseteq b_i$ and $c_k \subseteq b_i$. Then, we have $c_j^b \leq c_k^a$, because c_j^b and c_k^a are all come from same bucket b_i. However, Algorithm 2 must have merged them, if they exist. Therefore, there is at most one bucket c_j in $c_1, .., c_l$ that $c_j \subseteq b_i$. Then, $b_1, .., b_m$ can be at most divided into $2m-1$ buckets, namely, $l \leq 2m-1$. This proves the theorem.