World Wide Web (2015) 18:1419–1441
DOI 10.1007/s11280-014-0296-y

Detecting anomaly in data streams by fractal model

Rong Zhang · Minqi Zhou · Xueqing Gong · Xiaofeng He · Weining Qian · Shouke Qin · Aoying Zhou

Received: 29 July 2013 / Revised: 30 April 2014 /
Accepted: 16 May 2014 / Published online: 10 June 2014

Abstract Detecting anomaly in data streams attracts great attention in both academic and industry communities due to its wide range application in venture analysis, network monitoring, trend analysis and so on. However, existing methods on anomaly detection suffer three problems. 1) A large number of false positive results are generated. 2) Training data are needed to build the detection model, and an appropriate time window size along with corresponding threshold has to be set empirically. 3) Both time and space overhead is usually very high. To address these limitations. We propose a fractal-model-based approach to detection of anomalies that change underlying data distribution in this paper. Both a history-based algorithm and a parameter-free algorithm are introduced. We show that the later method consumes only limited memory and does not involve any training process. Theoretical analyses of the algorithm are presented. The experimental results on real life

R. Zhang · M. Zhou (✉) · X. Gong · X. He · W. Qian · A. Zhou
Center for Cloud Computing and Big Data, Software Engineering Institute,
Shanghai Key Laboratory of Trustworthy Computing, East China Normal University,
Shanghai, China 200062
e-mail: mqzhou@sei.ecnu.edu.cn

R. Zhang
e-mail: rzhang@sei.ecnu.edu.cn

X. Gong
e-mail: xqgong@sei.ecnu.edu.cn

X. He
e-mail: xfhe@sei.ecnu.edu.cn

W. Qian
e-mail: wnqian@sei.ecnu.edu.cn

A. Zhou
e-mail: ayzhou@sei.ecnu.edu.cn

S. Qin
Baidu Inc., Beijing, China 100085
e-mail: qinshouke@baidu.com

data sets indicate that, compared with existing anomaly detection methods, our algorithm can achieve higher precision with less space and time complexity.

Keywords Data streams · Anomaly detection · Fractal model

1 Introduction

Real-time monitoring and online data mining over data streams attracts great research attention in research community because of its wide variety of real world applications. For instance, it is very important to monitor the data streams for telecommunication network performance tuning, highway traffic management, trend-related prediction analysis, web-click streams analysis, information system intrusion detection, and sensor networking, etc. Some of these tasks are mission critical which require fast response. Traditional techniques initially proposed to manage static data collections cannot be applied directly to monitor or mine dynamic data streams. Robust and efficient anomaly detection is needed to process huge amount of information in constrained time period in order to monitor the data stream online.

However, constrained computing resources such as limited memory availability, as well as functional requirements including sequentially scanning and online accurate result report pose great challenge for data stream processing. For instance, almost all of the aforementioned monitoring tasks require monitoring data streams with multilevel window sizes/granularities. It is also assumed in these applications that no limit on the maximal number of monitored levels and the width of the largest monitored level is imposed. Algorithms consuming limited system resources therefore are much desired, which is also the case for anomaly detection in data stream.

In general, anomaly detection methods detect abnormality in the data stream based on a model that is derived from a normal historical behavior of a data stream. They try to discover the significant differences over short-term or long-term behaviors that are inconsistent with the derived model. However, the detected results largely depend on the data distribution of the underlying data stream. For the data stream whose distribution changes constantly, the existing methods will result in a large volume of false positives. On the other hand, due to the application-dependent definition of abnormal behavior, each existing method can only detect a particular type of abnormal behaviors. In real situation, however, different types of anomalies do co-exist and relate to each other. Therefore, it is meaningful to define abnormal behavior in a more general sense, and is highly desirable to have an efficient, accurate, and scalable mechanism for detecting abnormal behaviors over dynamic data streams.

One approach for anomaly detection is via detecting the change of self-similarity. Self-similarity is a property, which means an object is exactly or approximately similar to a part of itself. Self-similarity is ubiquitous both in natural phenomena (e.g., blood pressure, breathing, heart rate) and in social events (e.g., network traffic measurement, stock price change, TCP connection occurrence). In a statistical sense, self-similarity retains similar behavior, functionality, organization or property of a data set over a span of space and time. For instance, self-similarity leads to natural clusters of the data set, the bursty phenomenon in network traffic over long range of time span, etc. It represents the intrinsic nature of the data. Abnormal behavior defined based on the change of self-similarity naturally captures the essence of the event. Fractal, a geometric object generated by a repeating pattern recursively, consists of different parts, each of which are self-similar. As a consequence, fractals

which are built upon the statistical self-similarity, have been found with wide variety of applications [2].

Due to its generality, fractal model is introduced into anomaly detection in this paper. More specifically, we focus on modeling data with fractals based on recurrent iterated function systems (RIFS) and detect the anomaly based on new fractal creation. Intuitively, fractals defined by RIFS are descriptions of data by using data themselves. Thus, it is capable of capturing characteristics of underlying data distributions. Our data stream algorithm show that, by maintaining fractal models online, anomalies that change the underlying data distributions could be found, when new fractal pieces appear.

We made three major contributions.

– We propose a general-purpose and adaptable definition for abnormal behavior based on the change of self-similarity. Self-similarity captures the intrinsic property of the data streams, therefore a broader concept of definition of anomaly detection can be defined than those application-dependent definitions.
– We introduce the fractal analysis based on self-similarity into anomaly detection of the data stream. The piecewise fractal model is presented, which, to the best of our knowledge, is the first time that the piecewise fractal model is introduced to process data stream with guaranteed space cost and error. The fractal model appropriately describes the bursty behaviors of the data streams without giving the size of window being monitored in advance.
– We prove theoretically that the proposed parameter-free algorithm can return answers more accurately than existing methods. Extensive experiments are performed. Both the theoretical analyses and experimental results illustrate that the proposed method can monitor data streams accurately and efficiently with limited resource consumption.

2 Related work

Most existing data stream management systems (DSMS) are motivated by monitoring applications [1, 10, 11, 27]. Algorithms are proposed for specific monitoring tasks over data streams, such as change detection [6], deviation monitoring [23], incident alarming [9], monitoring correlated financial streams [33] , detecting topic bursts in documents [18, 28], querying and mining aggregation bursts over data streams [8, 25, 32, 34]. Monitoring abnormal behaviors in the network traffic systems has been studied in [19] and [13]. The detection of trends and surprise in time series database is studied in [26]. In [12], the exception of trend is defined and detected.

The bursty behaviors are the appearance of the underlying changes of the self-similarity in the data. For example, when a burst occurs on the aggregate result over a segment of a data stream, it usually implies that the distribution over the segment of the data stream is undergoing a change, or the trend of the data stream segment is probably an exception. Existing methods have some limitations to deal with bursty data. First, the application-dependent definition of an abnormal behavior limits each method to detect only one specific type of abnormal behaviors. Second, most existing methods are designed for querying over windows with given length. Since users usually do not know what the most appropriate size of the monitoring window is, they tend to monitor a large number of windows of various sizes simultaneously. This choice results in unnecessary storage requirement and computation. Third, the basic procedure for detecting anomalies over a data stream is to derive a model for normal behaviors from the history of the data stream, then check whether there are any significant differences in short-term or long-term behavior that are inconsistent with the model.

However, if the distribution of a data stream changes frequently, the direct application of the existing methods will lead to a large number of false positive results.

Fractal geometry has been known to be a powerful tool to describe complex shapes in nature [20]. Since they are far more natural than the traditional Euclidean geometry to model the natural phenomena, fractal techniques have been successfully applied to modeling and compressing the data sets [2, 16, 21, 31]. However, it is difficult to apply existing fractal techniques directly in the data stream scenario, since constructing the fractal model is in polynomial time complexity, while multi-pass scan of the whole data is required. O'Rourke [24] proposes to fit straight lines between data ranges, segments each arc into straight parts in linear time and aims to solve a linear inequality system. In this paper, we propose a novel approach to detect anomaly by applying fractal model to the data stream problem. The change of self-similarity in the data is captured by fractal model, hence detect the anomaly in data stream.

The remainder of this paper is organized as follows. The preliminary knowledge of fractal and recurrent iterated function systems (RIFS) are introduced in Section 3. In Section 4, the data stream and the related anomaly detection are discussed. The problem we aim to address in this paper is presented. In Section 5, a novel piecewise fractal model for data streams is proposed. The space and error bound of this model are analyzed in detail. A parameter-free anomaly detection algorithm with the detailed theoretical analyses is described in Section 6. We present the experimental setup and the result analyses in Section 7. Finally, Section 8 concludes the paper.

3 Preliminaries of fractals

Self-similarity is widespread in both nature phenomena and social events. Fractals, as a consequence of self-similarity, have been found wide-ranging applications in numerous cases [2]. In this paper, we will focus on modeling data with fractals based on recurrent iterated function systems (RIFS).

The piecewise fractal model proposed in this paper, which is for describing data streams, can be tracked back to its mathematical roots of recurrent iterated function systems. Therefore, at the very beginning of our discussion, some preliminary knowledge about fractal techniques based on recurrent iterated function systems is introduced in this section.

3.1 Power law scaling relationship

It is observed that the pieces of a fractal object, when enlarged, are similar to larger pieces of the whole [20]. If these pieces are identically re-scaled replica of the others, then the fractal is called an exact one. If, in a statistical sense, the similarity holds only on the given feature of different granularity, then the fractal is a statistical one. Deterministic fractals such as the real von Koch curve and Cantor set are both exact in that every part is strictly self-similar, representing an exact (or scaled) copy of the whole; however, most natural objects are statistical fractals.

If a quantitative property, q, is measured on a time scale s, then the value of q depends on s according to the following scaling relationship [20]:

$$q = ps^r. \tag{1}$$

This is called power law scaling relationship of q with s. p is a factor of proportionality and r is the scaling exponent. The value of r can be easily determined by the slope of the linear least squares fitting to the graph of data pairs ($\log s$ $\log q$):

$$\log q = \log p + r \log s. \tag{2}$$

Data points of exact fractals are located exactly on the line of the regression slope, whereas those of statistical fractals scatter around the regression slope, for the two sides of (2) are equal only in a distribution sense.

For discrete time series data, (1) can be transformed to

$$q(s,t) = s^r q(t), \tag{3}$$

where t is the basic time unit and $s (s \in \mathbb{R})$ is the scaling measurement [7]. r can be determined by

$$r = \frac{\log(q(s,t)/q(t))}{\log s}. \tag{4}$$

The power-law scaling relationship is also retained when q is a first order aggregate function (e.g. sum, $count$) or some second order statistic variables (e.g. $variance$).

3.2 Iterated function system (IFS)

An iterated function system [4] is a scheme for describing and manipulating complex fractal attractors using simple mathematical models. More precisely, they are a finite collection of contraction maps M_i for $i = 1, 2, 3, ..., m$ which map a compact metric space onto itself. In particular, a linear contraction map has the form

$$M_i \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11}^i & a_{12}^i \\ a_{21}^i & a_{22}^i \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b_1^i \\ b_2^i \end{bmatrix} \tag{5}$$

The attractor of the IFS is the fixed point of $x = M(x)$. Assume $\mathbf{A}$ is the attractor, then

$$\mathbf{A} = \bigcup_{i=1}^{m} M_i(\mathbf{A}). \tag{6}$$

By modelling the real life data with an IFS, we can use the attractor of the IFS to approximate the real life data.

When the map parameters of IFS have been determined, the attractor can be constructed with one of two algorithms. The first one is a deterministic algorithm. It is based on the contractivity of the M mapping and begins by selecting any compact set $A_0 \subset R^T$, where T is the topographical dimension of the data set. After successively computing $A_n = M^n(A)$, the sequence $\{A_j\}_{j=1}^{\infty}$ will converge to the attractor of the IFS. The second one is a randomized iterative algorithm. It uses the properties of the dynamical system associated with the M_i's, and it proceeds with initially $x_0 \in R^T$. Then it chooses recursively and independently with $x_n \in \{M_1(x_{n-1}), M_2(x_{n-1}), ..., M_m(x_{n-1})\}$ for $n = 1, 2, 3, ...$ The M_i is selected with a given probability. The sequence $\{x_n\}_{n=1}^{\infty}$ will converge to the attractor of the IFS.

3.3 Recurrent iterated function system (RIFS)

Recurrent iterated function systems were first introduced in [5] as a generalization of the original finite map IFS. Different from IFS which is defined for certain metric space, RIFS can be generalized for arbitrary metric spaces. Therefore, RIFS is capable of encoding a

wider range of measures. RIFS is always used to produce complicated objects with only a few maps. RIFS theory has been successfully applied to image processing and compressing [15, 22].

In RIFS, given the contraction maps, the attractor $A \subset R^d$ is $\bigcup_{i=1}^{N} A_i$ where $A_i \subset R^d$ $(i = 1, ..., N)$ are possibly overlapping partitions. Furthermore, $M_j(\mathbf{A}) = A_j = \bigcup_{<i,j>\in G} M_j(A_i)$. Here G is a directed graph, and $\mathbf{A} = (A_1, ..., A_N) \in (R^d)^N$.

Each edge $< i, j >\in G$ indicates that the mapping composition $M_j \diamond M_i$ is allowed. The attractor can also be represented by $A = \mathbf{M}(A)$, in which $\mathbf{M} : (R^d)^N \rightarrow (R^d)^N$ is defined as $(M_1, M_2, ...M_N)$. Thus, given a set vector $\mathbf{D} = (D, D, ..., D)$ with $D \subset R^d$, $\mathbf{A} = \lim_{i\rightarrow\infty} \mathbf{M}^i(\mathbf{D})$. Similar to IFS, attractor A can be used to approximate the data. When the graph G is complete, the RIFS degrades to an IFS [16].

IFS and RIFS are often used to summarize and compress fractal data. The problem lies in the determination of the attractor. Since the attractor can be computed given the mapping, the computation of the parameters of the mapping is usually called the inverse problem of IFS or RIFS [16].

Given the mapping $\mathbf{M}$, $\mathbf{D}' = \mathbf{M}(\mathbf{D})$, we consider the condition that $\mathbf{D}' \subset \mathbf{D}$ in this paper. Under this condition, the *open set* property holds, so that the *Collage Theorem* can be applied to guarantee the precision of approximating the original data using the attractor [2, 5]. This theorem provides a way to measure the goodness of fit of the attractor associated with an RIFS and a given function.

Collage theorem Let (X, d) be a complete metric space where d is a distance measure. Let $\mathbf{D}$ be a given function and $\epsilon > 0$ be a given real number. Choose an RIFS, with contractility factor $s = max\{s_i : i = 1, 2, 3, ..., N\}$ so that $d(\mathbf{D}, \bigcup_{i=1}^{N} M_i(\mathbf{D})) \leq \epsilon$. Then $d(\mathbf{D}, A) \leq \frac{\epsilon}{1-s}$ where $\mathbf{A}$ is the attractor of the RIFS.

Since the contraction property of the mapping holds and $\mathbf{D}' \subset \mathbf{D}$, the size of $\mathbf{D}'$ must be smaller than that of $\mathbf{D}$.

RIFS is more suitable for data stream processing than IFS. Intuitively, at any time, only a small piece of the stream can be obtained. Thus, the whole attractor may not be precisely estimated.

4 Problem description

A data stream X is a sequence of points $x_1, ..., x_n$. Function F being monitored can be not only the monotonic aggregates with respect to the window size, for example sum and $count$, but also the non-monotonic ones such as $average$.

In the algorithms introduced in this paper, F can be extended to some second order statistics, such as $variance$. In real world applications, with different definitions of abnormal behavior, individual detecting methods can monitor the results of different functions. For example, burst detection monitors the results of $sum, count, average$, and outlier and deviant detecting algorithms monitor the results of $variance$. One common property of the above functions is that they all retain the power-law scaling relationship.

Following we show that there exists such a property for aggregate function F and some second order statistics in different kinds of real life applications.

Here, the experimental results on two real life data sets $\mathbf{D}1$ and $\mathbf{D}2$ are chosen as examples (the details of these data sets are described in Section 6.1). The monitored quantitative

property q on time scale s is determined as follows. For a window of size w_i, we compute the aggregates on testing data stream X. This gives another time series Y, y_1, ..., y_k.

$$y_k = \sum_{j=k}^{k+w_i-1} x_j, k = 1, 2, ..., n - w_i + 1$$

The *average* and *sample variance* of $y_1, y_2..., y_k$ are denoted by $avg(y(w_i))$ and $var(y(w_i))$, respectively. Let q_1 represent $avg(y(w_i))$, q_2 represent $var(y(w_i))$, and s be w_i.

$$q_1 = avg(y(w_i)) = \frac{1}{k}\sum_{j=1}^{k} y_j$$

$$q_2 = var(y(w_i)) = \frac{1}{k}\sum_{j=1}^{k}(y_j - q_1)^2$$

After plotting the data points $(\log s, \log q)$ (q is either the average or the variance), with base equals to 10, on the graph, we draw a line of the linear regression. Note that from

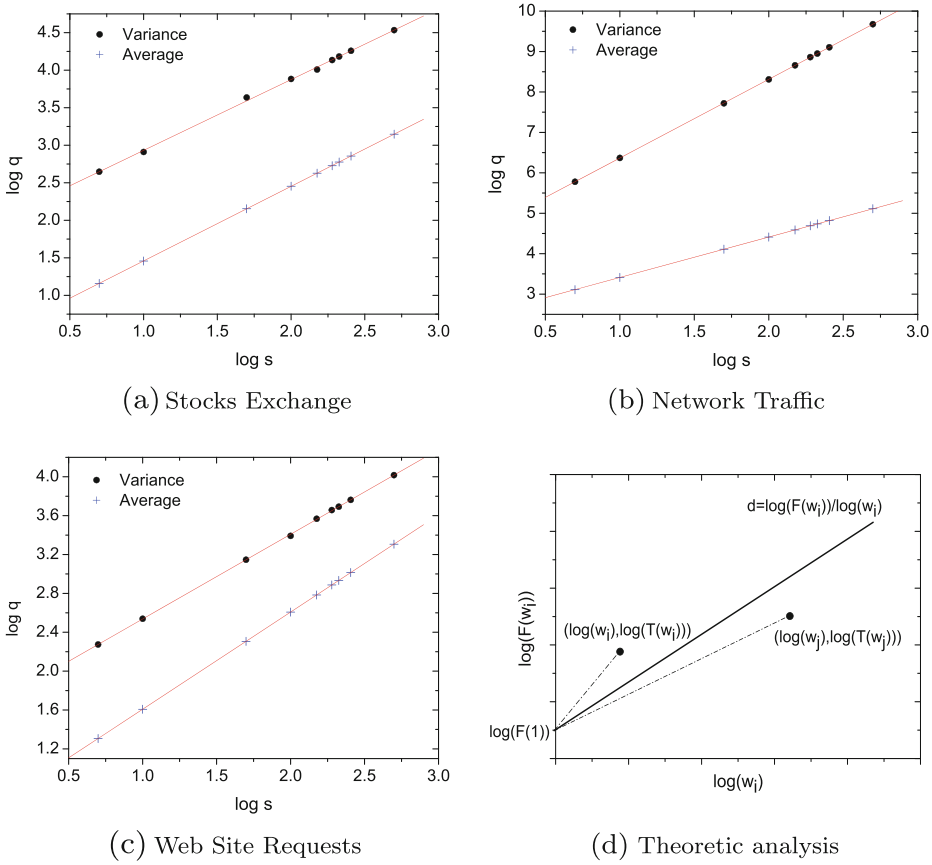


Figure 1 Power law scaling of aggregate function F on real life data sets

Table 1 List of notations and definitions

x_i	i-th data point in stream X
m	number of pieces/contraction maps
A_i	distance
B_i	distance
α	error constraint of piecewise fractal model
M_i	i-th contraction map
$\mathbf{D}_i$	i-th partition of whole metric space $\mathbf{D}$
$\mathbf{D}'_i$	the partition mapped from $\mathbf{D}_i$, $\mathbf{D}'_i \subset \mathbf{D}_i$
P_i	i-th piece of fractal model
P'_i	the piece mapped from P_i by M_i, $P'_i \subset P_i$

Figure 1, the regression line connects all the data points, which means that the rule of power-law scaling relationship of F is obeyed.

Monitoring anomalies in data streams is to detect the change of power-law scaling relationship, i.e., self-similarity, on the sequences with continuous incoming new value x_i. Under such a data stream model, the problem about anomaly detection can be described as follows.

Problem statement A data stream X is a sequence of points $x_1, ..., x_n$. An anomaly is detected if the self-similarity of F over X changes (i.e., the historic one is violated) when new value x_n comes.

In this definition, we use the change of self-similarity to describe the occurrence of abnormal behavior in data streams. This definition is a broader concept than those specific ones defined in previous methods. Note that the definition of anomalies is highly application-dependent. The aim of this paper is not to provide a general approach that can detect anomalies for all applications, but a general-purpose method that detect anomalies that change underlying data distributions which can be characterized by fractal models. Different applications can be built based on anomaly detection results.

In the next section, we describe how to measure the change of the self-similarity for current data stream (Table 1).

5 Piecewise fractal model for data streams

The recurrent iterated function system (RIFS) can be used to generate deterministic fractals to approximate real world data. The inverse problem deals with how to set the parameters of an RIFS when modeling a given application. Constructing a piecewise fractal model for a data stream is equivalent to solving an inverse problem in data stream scenario. In this section, we investigate two approaches to solve the problem. One is L_2 error optimal method, whose complexity is $O(n^2)$ in time and $O(n)$ in space. The other is an approximate method with at most $(1 + \epsilon)$ times the error for the optimal solution. It uses $O(n)$ time and $O(m)$ space, m is the number of required pieces, $m < \log n$.

5.1 Inverse problem

Since both iterated function system (IFS) and recurrent iterated function system (RIFS) can produce complicated objects with only a few maps, the associated inverse problem has been

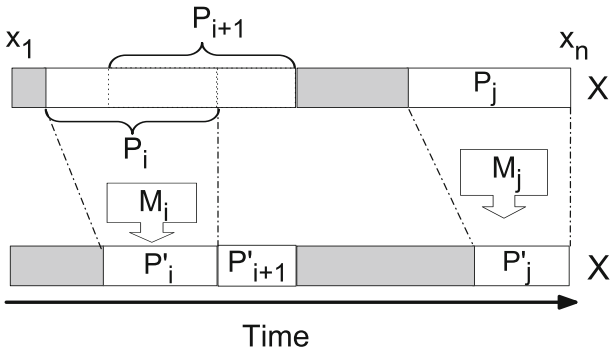


Figure 2 Contraction mapping on stream $\boldsymbol{X}$ in our piecewise fractal model

attracting much attention [14, 15, 22, 29]. The inverse problem is to determine the map parameters needed by an IFS or RIFS to model a given data set.

The RIFS is a general form of the IFS, which is capable of modeling a wide range of objects. If the inverse problem can be addressed, then RIFS theory can be applied to model arbitrary functions while achieving large-scale data compression. Hence we focus on the inverse problem of RIFS to model the data stream in this paper.

From the preliminary introduction on RIFS, we known that an RIFS is a finite collection of contraction maps M_i ($i = 1, 2, 3, ..., m$). Each linear contraction map has the form:

$$M_i \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a^i_{11} & 0 \\ a^i_{21} & a^i_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} b^i_1 \\ b^i_2 \end{bmatrix}. \tag{7}$$

The mapping (7) is often called shear transformation [2]: vertical lines are mapped to vertical lines by the factor a^i_{22}. In (7), the parameter is called the contraction factor for map M_i. a^i_{22} is constrained to be a real number with $|a^i_{22}| < 1$. In general, it is not required that the contraction mapping fractal model have any zero entries in the maps. However, the zero entry in the coefficient matrix of (7) ensures that the resulting linear mapping is single valued [3]. The RIFS discussed in this paper consists of such contraction maps.

RIFS can map a partition D_i of whole metric space D to another partition D'_j of D under two constraints. Each M_i maps an interval $[x_{(i,0)}..x_{(i,1)}]$ of X, denoted by D_i, to a subinterval $[x'_{(i,0)}..x'_{(i,1)}]$ of D_i itself, denoted by D'_i. $x_{(i,0)}$ and $x_{(i,1)}$ are the begin time stamp and end time stamp of interval D_i, respectively. Correspondingly, $x'_{(i,0)}$ and $x'_{(i,1)}$ are the begin time stamp and end time stamp of interval D'_i, respectively. Then the first constraint follows

$$D'_i = M_i(D_i), D'_i \subset D_i. \tag{8}$$

The second constraint which needs to be maintained in RIFS is that every contraction mapping maps the two endpoints $(x_{(i,0)}, y_{(i,0)})$ and $(x_{(i,1)}, y_{(i,1)})$ of D_i to two endpoints $(x'_{(i,0)}, y'_{(i,0)})$ and $(x'_{(i,1)}, y'_{(i,1)})$ of D'_i. It is illustrated in Figure 2. M_i transforms a larger interval D_i of stream X to a smaller interval D'_i of the same stream.

Thus, the endpoints of D'_i are also the endpoints of the attractor of M_i, that is,

$$M_i \begin{bmatrix} x'_{(i,0)} \\ y'_{(i,0)} \end{bmatrix} = \begin{bmatrix} x_{(i,0)} \\ y_{(i,0)} \end{bmatrix}, M_i \begin{bmatrix} x'_{(i,1)} \\ y'_{(i,1)} \end{bmatrix} = \begin{bmatrix} x_{(i,1)} \\ y_{(i,1)} \end{bmatrix}. \tag{9}$$

Once the contraction factor a_{22}^i for each mapping is determined, the remaining parameters can be obtained using the endpoint constraint (9). These parameters are computed by the following equations.

$$a_{11}^i = \frac{x'_{(i,1)} - x'_{(i,0)}}{x_{(i,1)} - x_{(i,0)}} \tag{10}$$

$$b_1^i = \frac{x_{(i,1)}x'_{(i,0)} - x_{(i,0)}x'_{(i,1)}}{x_{(i,1)} - x_{(i,0)}} \tag{11}$$

$$a_{21}^i = \frac{y'_{(i,1)} - y'_{(i,0)}}{x_{(i,1)} - x_{(i,0)}} - a_{22}^i \frac{y_{(i,1)} - y_{(i,0)}}{x_{(i,1)} - x_{(i,0)}} \tag{12}$$

$$b_2^i = \frac{x_{(i,1)}y'_{(i,0)} - x_{(i,0)}y'_{(i,1)}}{x_{(i,1)} - x_{(i,0)}} - a_{22}^i \frac{x_{(i,1)}y_{(i,0)} - x_{(i,0)}y_{(i,1)}}{x_{(i,1)} - x_{(i,0)}} \tag{13}$$

Now the problem of constructing the piecewise fractal model for data streams is reduced to determining the appropriate value of contraction factor a_{22}^i.

5.2 Piecewise fractal model

In a piecewise fractal model, the goal of the global optimal solution is to minimize the sum of error of each piece. It also requires that the model to be maintained incrementally for evolving data streams with small overhead on time and space. This global optimization problem is NP-hard [17]. To ease the problem, we first construct the piecewise fractal model to solve a local optimization problem. This approach intends to minimize the error of individual pieces. Then an approximate local optimal algorithm for building the piecewise fractal model is suggested, which needs only one-pass sequential scan of the data stream and $O(m)$ memory consumption. m is the number of contraction maps M_i, i.e., the number of pieces we try to model on a stream. The error of the approximate method is two times smaller than that of the local optimal method, which we will prove later.

Below we first introduce the optimal fractal model, then the approximate model is presented.

5.2.1 Optimal model

Assume the data point in a data stream X is a $(timestamp : i, value : y_i)$ pair. Let the start point of a piece P of stream X be (s, y_s), and the end point of P be (e, y_e), $e > s$. Since the data points in a stream are discrete, a contraction mapping M can be defined as $M(i, y_i) = (int(i \cdot a_{11} + b_1), i \cdot a_{21} + y_i a_{22} + b_2)$. The data points in a larger piece of a data stream are mapped by M from $P\{(s, y_s), (e, y_e)\}$ to a smaller piece of the same stream $P'\{(s', y_{s'}), (e', y_{e'})\}$, where $P' \subset P$ and $e - s > e' - s'$. Every two adjacent P's meet only at the endpoints and do not overlap. For all the pieces of the whole data stream, however, there might be some overlaps, that is, $\bigcup P_i = \bigcup P'_i = X$. Suppose the L_2 error between the data points in mapping $M(P)$ and those in the original piece P' is $E(M) = (M(P) - P')^2$, then the error of a mapping M is

$$E(M) = \sum_{i=s}^{e} ((i \cdot a_{21} + y_i a_{22} + b_2) - y_j)^2,$$

where $j = int(i \cdot a_{11} + b_1)$. Given P and P', the optimal mapping M_{opt} is the one such that the minimum value of $E(M)$ is reached.

On data stream X, it is ensured that each contraction mapping M maps the start point and end point of piece P to corresponding ones of piece P', namely, $(s', y_{s'}) = M(s, y_s)$ and $(e', y_{e'}) = M(e, y_e)$. a_{21} and b_2 can be obtained by (12,13). Therefore, we have

$$E(M) = \sum_{i=s}^{e} (A_i a_{22} - B_i)^2, \tag{14}$$

$$\begin{aligned} A_i &= y_i - \left(\frac{e-i}{e-s} y_s + \frac{i-s}{e-s} y_e\right) \\ B_i &= y_j - \left(\frac{e-i}{e-s} y_{s'} + \frac{i-s}{e-s} y_{e'}\right). \end{aligned}$$

A_i and B_i have their own geometric interpretations. The term A_i denotes vertical distance between the point y_i belonging to piece P and the straight line connecting end points of piece P. The term B_i denotes vertical distance between the point y_j belonging to piece P' and the straight line connecting end points of piece P', where $j = int(i \cdot a_{11} + b_1)$.

The coefficient a_{22} of the optimal mapping M_{opt} can be computed using the following equation.

$$\frac{\partial E(M)}{\partial a_{22}} = 2 \sum_{i=s}^{e} (A_i a_{22} - B_i) A_i = 0$$

Then

$$a_{22} = \sum_{i=s}^{e} A_i B_i / \sum_{i=s}^{e} A_i^2.$$

Consequently, the optimal contraction mapping M_{opt} can be obtained by maintaining $\Sigma_{i=s}^{e} A_i B_i$ and $\Sigma_{i=s}^{e} A_i^2$ over the data streams. Algorithm 1 illustrates the method for constructing the optimal piecewise fractal model for a data stream.

Algorithm 1 OptimalModel(x_n)

```
1: compute Σ_{i=s}^{e} A_i B_i;
2: compute Σ_{i=s}^{e} A_i^2 incrementally;
3: if Σ_{i=s}^{e} A_i B_i / Σ_{i=s}^{e} A_i^2 < 1 then
4:     add x_n to current piece;
5: else
6:     create a new piece for x_n;
7:     pieceNumber++;
8:     if pieceNumber > m then
9:         delete the oldest piece;
10:    end if
11: end if
12: add x_n to all pieces;
```

Algorithm 1 takes x_n as the input. The only parameter m sets the maximal number of the maintained pieces. Computing $\Sigma_{i=s}^{e} A_i B_i$ in Line 1 take $O(n)$ time and $O(n)$ space. Line 2 computes $\Sigma_{i=s}^{e} A_i^2$ incrementally. If the contraction factor is less than 1, then x_n will be put

into the current piece, otherwise it will be put into a new piece. If the number of pieces is more than the pre-defined threshold m, then Line 9 deletes the oldest piece.

Assume the length of a data stream is n. Algorithm 1 maintains optimal piecewise fractal model on a data stream with $O(n + m)$ space in $O(n^2)$ time with m as the number of contraction maps.

5.2.2 Approximate model

For data stream processing, it is required that we do one-pass sequential scan on the data stream only, for it is impossible to store the whole stream in memory. Therefore, the value of $\Sigma_{i=s}^{e} A_i B_i$ cannot be maintained in the data stream scenario, as Algorithm 1 requires. To address this problem, we construct an approximate optimal piecewise fractal model to compute the approximate value of $\Sigma_{i=s}^{e} A_i B_i$.

Denote the approximate value of a_{22} to be a'_{22}. Similarly we have

$$
\begin{aligned}
C_i &= y_i - \left(\frac{e' - i}{e' - s'} y'_s + \frac{i - s'}{e' - s'} y'_e \right) \\
\sum_{i=s}^{e} B_i^2 &= \frac{e - s}{e' - s'} \sum_{i=s'}^{e'} C_i^2 \\
(a'_{22})^2 &= \frac{\sum_{i=s}^{e} B_i^2}{\sum_{i=s}^{e} A_i^2}
\end{aligned}
$$

Hence we only need to maintain $\Sigma_{i=s'}^{e'} C_i^2$ and $\Sigma_{i=s}^{e} A_i^2$ incrementally for current pair of P and P' on the data stream. In this way, the a'_{22} and corresponding approximate optimal results M_{app} can be computed.

The approximate piecewise fractal model consists of m pieces which can be used to reconstruct the original data stream. The error of reconstruction is bounded by Collage Theorem [2, 5].

Each piece P_i corresponds to a contraction mapping M_i and a pair of partition P and P'. One value needed to be stored for P_i is the suffix sum $F(n - si')$ starting from $(si', y_{si'})$, where a suffix sum of data stream is defined as $F(w_i) = \Sigma_{j=n-wi+1}^{n} x_j$. The other value needs to be stored for P_i is the number of points included in this suffix sum. We first try to add each data point of the evolving stream into the current piece. If failed, then a new piece is created for this point. Because the end point of piece P'_i is the start point of piece P'_{i+1}, it is not necessary to store the end point. We do not consider the reconstruction of original data stream in this paper, so the contraction factor does not need to be stored either. If the model is used to detect anomalies only, one piece is enough.

The method of constructing an approximate optimal piecewise fractal model for a data stream is described in Algorithm 2.

Under the condition of $\Sigma_{i=s}^{e} B_i^2 / \Sigma_{i=s}^{e} A_i^2 < 1$, the incoming x_n is put into the partitions P and P' corresponding to the current piece P_i. The newly added x_n serves as the end point of current piece. When $\Sigma_{i=s}^{e} B_i^2 / \Sigma_{i=s}^{e} A_i^2 \geq 1$, in Line 6, a new piece P_{i+1} is created. The piece P'_i serves as the new P_{i+1} and a new P'_{i+1} is created. x_{n-1} becomes the start point of P'_{i+1}. x_n becomes the end point for both P'_{i+1} and P_{i+1}. Therefore, it always follows that

$$P_{i+1} = P'_i \bigcup P'_{i+1}$$

Algorithm 2 ApproximateModel(x_n)

```
compute Σ_{i=s}^{e} B_i^2 incrementally;
compute Σ_{i=s}^{e} A_i^2 incrementally;
if Σ_{i=s}^{e} B_i^2 / Σ_{i=s}^{e} A_i^2 < 1 then
    add x_n to current piece;
else
    create a new piece for x_n;
    pieceNumber++;
    if pieceNumber > m then
        delete the oldest piece;
    end if
end if
add x_n to all pieces;
```

When the existing number of pieces exceeds the threshold m, the oldest piece is deleted. Only the latest m pieces for the data stream are kept. Figure 2 illustrates the piecewise fractal model for data stream X with maps M_i and M_j.

The upper bound of the time and space complexity to maintain the approximate optimal piecewise fractal model for a data stream is $O(n)$ and $O(m)$ respectively.

5.3 Error bound

Algorithm 2 is designed to approximate the local optimal solution. The error bound of the resulting approximate solution is shown in Theorem 1.

Theorem 1 *Assume Algorithm 2 produces an approximate optimal solution M_{app} which maps the partition P to P' with error $E(M_{app})$, and the optimal method maps the partition P to P' with error $E(M_{opt})$ then $E(M_{app}) \leq 2E(M_{opt})$ always holds.*

Proof Expand $E(M_{opt})$ and $E(M_{app})$ by (14) and

$$\begin{aligned} E(M_{opt}) &= \sum_{i=s}^{e} (A_i a_{22} - B_i)^2 \\ &= a_{22}^2 \sum_{i=s}^{e} A_i^2 - 2a_{22} \sum_{i=s}^{e} A_i B_i + \sum_{i=s}^{e} B_i^2 \end{aligned}$$

and

$$\begin{aligned} E(M_{app}) &= \sum_{i=s}^{e} (A_i a'_{22} - B_i)^2 \\ &= (a'_{22})^2 \sum_{i=s}^{e} A_i^2 - 2a'_{22} \sum_{i=s}^{e} A_i B_i + \sum_{i=s}^{e} B_i^2. \end{aligned}$$

divide $E(M_{app})$ by $E(M_{opt})$, we get

$$\frac{E(M_{app})}{E(M_{opt})} = \frac{(a'_{22})^2 \sum_{i=s}^{e} A_i^2 - 2a'_{22} \sum_{i=s}^{e} A_i B_i + \sum_{i=s}^{e} B_i^2}{a_{22}^2 \Sigma_{i=s}^{e} A_i^2 - 2a_{22} \Sigma_{i=s}^{e} A_i B_i + \Sigma_{i=s}^{e} B_i^2}. \tag{15}$$

Recall that:

$$a_{22} = \sum_{i=s}^{e} A_i B_i / \sum_{i=s}^{e} A_i^2$$

and

$$(a'_{22})^2 = \sum_{i=s}^{e} B_i^2 / \sum_{i=s}^{e} A_i^2,$$

divided by $\Sigma_{i=s}^{e} A_i^2$, (15) can be transformed to

$$\frac{E(M_{app})}{E(M_{opt})} = \frac{2(a'_{22})^2 - 2a_{22}a'_{22}}{(a'_{22})^2 - (a_{22})^2}.$$

Let $\delta = a_{22}/a'_{22}$. Divide both sides of the above equation by $(a'_{22})^2$, we get

$$\frac{E(M_{app})}{E(M_{opt})} = \frac{2(1-\delta)}{1-\delta^2} = \frac{2}{1+\delta} \leq 2,$$

where we use the fact that $\delta \geq 0$.

$$\delta = \frac{a_{22}}{a'_{22}} = \sqrt{\frac{(\sum_{i=s}^{e} A_i B_i)^2}{\sum_{i=s}^{e} A_i^2 \sum_{i=s}^{e} B_i^2}} \leq 1$$

Then $\delta = a_{22}/a'_{22} \leq 1$. Therefore,

$$\frac{E(M_{app})}{E(M_{opt})} \leq 2.$$

This finishes the proof of $E(M_{app}) \leq 2E(M_{opt})$. □

From Theorem 1, the relative error of approximate solution is bounded above. For some applications in which the bounded absolute error is wanted, we provide a parameter α to control the error. Then the error of mapping M_{app} on partitions P and P' can be guaranteed as follows.

$$E(M_{app}) \leq \alpha \cdot E(P'_{LSF}),$$

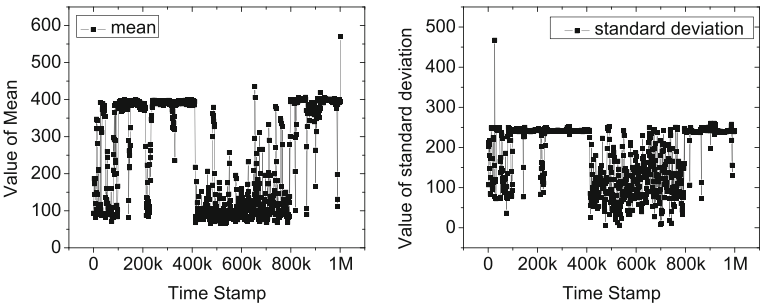
where $E(P'_{LSF})$ is the Least Linear Fitting error on partition P'. Therefore, the error produced by approximate piecewise fractal model on partition P' is less than the smaller value of $\alpha E(P'_{LSF})$ and $2E(M_{opt})$, which is

$$E(M_{app}) \leq min\{\alpha E(P'_{LSF}), 2E(M_{opt})\}$$

To control the parameter α, we only need to check whether $E(M_{app}) \leq \alpha E(P'_{LSF})$ holds or not when a new x_n comes. If $E(M_{app}) \leq \alpha E(P'_{LSF})$ holds, x_n is put into the current partition; otherwise, a new partition is created for x_n. The rest is the same as that of Algorithm 2.

6 Anomaly detection algorithms

In this section, we first introduce a history-based method, which takes an approach similar to existing burst detection methods. Then, a parameter free mechanism for anomaly detection is presented. Compared with the history-based method, it is more adaptable to real world applications since it frees users from tedious parameter selection procedure.



(a) *mean* (b) *standard deviation*

Figure 3 The distribution is changing constantly on D2

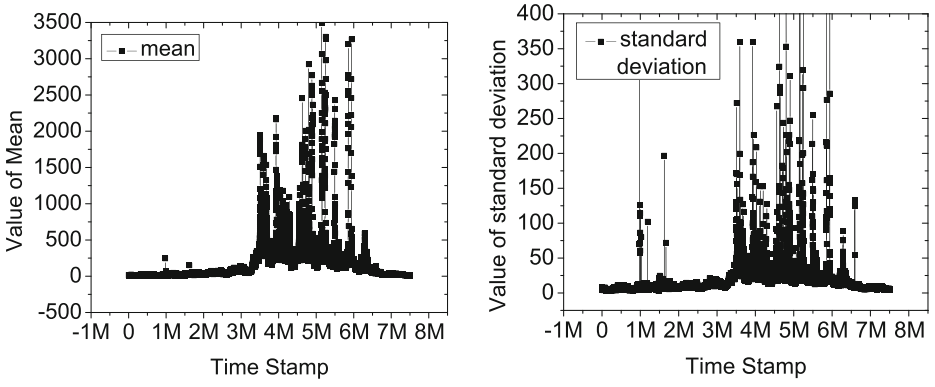
6.1 History-based method

Existing methods detect data stream based on the model derived from a normal behavior in the data stream history. For instance, the threshold of bursty behaviors of aggregate function F on the i-th window is set as follows. At first, the moving $F(w_i)$ is computed over some training data which could be the foremost 1 % of the data set. The training data forms another time series data set, called Y. Then the threshold of anomaly is set to be

$$T(w_i) = \mu_y + \lambda\sigma_y$$

where, μ_y and σ_y are the mean and standard deviation, respectively. The threshold can be tuned by varying the coefficient λ of standard deviation.

These methods then look for the significant differences in short-term or long-term behaviors which are inconsistent with the model. This ultimately depends on the distribution of the data over the streams. For those real-life data streams whose distribution change constantly, existing methods will result in a large volume of false positives. This is illustrated in Figures 3 and 4.



(a) *mean* (b) *standard deviation*

Figure 4 The distribution is changing constantly on D3

Algorithm 3 AnomalyDetection(x_n)

```
1: read x_n;
2: ApproximateModel(x_n);
3: if createNewPiece() then
4:       alarm anomaly;
5: end if
```

In these two experiments, for every fixed length of interval, say, 1000 data points, we compute the *mean* and *standard deviation*. The values of this interval and the time stamp of right side end point are plotted on the graphs.

It can be seen from Figures 3 and 4 that the *mean* and *standard deviation* fluctuate dramatically. In this case, the performance of history-based model will degenerate rapidly due to the large variation between consecutive windows. The result is that a large volume of false positives are returned by the existing methods. We will show the detailed experimental results later.

Another drawback of the history-based model is that it cannot be trained online to adapt to the evolving data streams. It is because we do not know exactly how long the data segment from a distribution will be, therefore difficult to set an appropriate interval size for the training data. It is not a trivial job to determine the threshold of change, even if we can measure the change of distribution [6].

6.2 Parameter-free method

Above discussion prompts us to investigate the parameter-free mechanism for detecting abnormal behaviors in the data streams. We use the following equation to represent the data derived from the data stream.

$$Y = X + e,$$

where Y is either an aggregate function (e.g., sum, $count$, $average$), measurement of trend and deviant (say, $variance$), or distance of two kinds of distribution on the data stream. X is the result of Y in normal condition. e, the effect caused by abnormal behaviors on Y, can be seen as a noise. We assume that X and e have individual distributions and the uncertainty noise e is white Gaussian.

In the scenario of discrete data points, Gaussian white noise can only be modelled and processed in time series data. The constraints imposed on the data stream processing include limited memory consumption, sequentially linear scanning, and on-line accurate result reporting over evolving data streams. Therefore, we build piecewise fractal model on the data streams to model the change of self-similarity induced by Gaussian white noise.

The occurrence of abnormal behaviors is rooted in the change of self-similarity in the data stream. The piecewise fractal model can find the pieces with self-similarity in a data stream. That a new piece is created means there is a change on current self-similarity.

Algorithm 3 is a parameter-free algorithm to detect anomaly based on the change of self-similarity. It alarms at the occurrence of an anomaly when a new piece is created for incoming x_n.

Note that the parameter m in Algorithm 1 is in essence a parameter on limitation of memory consumption. Considering the compactness for storing a contraction map, m could be set sufficiently large in the environment with commodity servers. Thus, our method is a parameter-free one from this point-of-view.

With this model, we can evaluate both the signal and noise in the data stream. Since the anomaly detection only focuses on the prominent noise which causes the change of self-similarity, the evaluation of anomaly on e is not affected by the evolving of normal signal X.

7 Performance evaluation

We present the empirical studies in this section. It shows that the fractal-based method can efficiently generate accurate alarms to abnormal behaviors with small cost in space and time.

7.1 Experimental setup

The algorithms are implemented under Microsoft Visual C++ version 6.0. The experiments are conducted on a platform with a 2.4GHz CPU, 512MB main memory, and the operating system is Windows 2000. We applied our algorithms to a variety of data sets. Due to the space limitation, only the testing results of three real data sets are reported here.

- Stocks Exchange: This data set contains one year tick-by-tick trading records in 1998. These records are from tens of corporations on Hongkong main board. We randomly pick up the records of a corporation to form a data stream. It has 485,984 unique records, each of which contains trading time (precise to the second) and trading price. It is called data set D1.
- Network Traffic: This data set BC-Oct89Ext4 is a network traffic tracing data set obtained from the Internet Traffic Archive [13, 30]. It contains a million packet arrivals seen on an ethernet at the Bellcore Morristown Research and Engineering facility. It is called data set D2 here.
- Web Site Requests: This data set is also obtained from the Internet Traffic Archive [13, 30]. 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-day period, the site received 1,352,804,107 requests. The basic window, i.e., an item x_i, is the number of requests issued in one second. So the stream length is $n = 92 * 24 * 3600 = 7,948,800s$. It is data set D3.

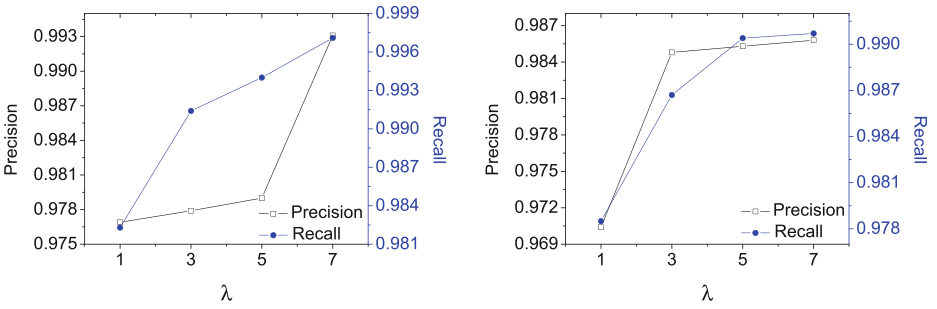
7.2 Accuracy metrics

In this subsection, we present the results of burst detection [8, 25, 34]. The comparison tests will be presented later. Note that our methods can also be applied to re-construct data and detect anomaly defined by existing methods.

Burst detection is defined as follows. Assume that F is an aggregate function, and w_i is the latest sliding window of a stream with length l_i ($1 \leq l_i \leq n$). $T(w_i)$ is the threshold on $F(w_i)$ specified by users. When x_n comes, a burst is reported on window w_i if $F(w_i) \geq T(w_i)$.

In our burst detection test setting, sum aggregate is chosen as F. Two accuracy metrics , recall and precision, are considered. Recall is the ratio of true alarms raised to the total true alarms which should be raised; precision is the ratio of true alarms raised to the total alarms raised.

To set the threshold for $F(w_i)$ on the i-th window, we compute moving $F(w_i)$ over some training data. The training data is the foremost 1 % of each data set. It forms another time



(a) Varying λ on D1 (b) Varying λ on D3

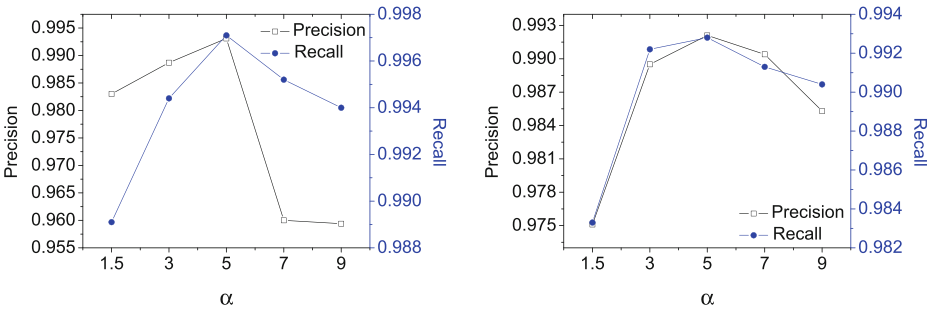
Figure 5 The accuracy when varying burst threshold λ. (**a**) accuracy when varying λ on D1, (**b**) accuracy when varying λ on D3

series data set, called Y. The absolute thresholds are set to be $T(w_i) = \mu_y + \lambda\sigma_y$. The length of windows are 4, 8, ..., $4 * NW$ time units, where NW is the number of windows. NW varies from 50 to 1100.

7.3 Experimental results

We first evaluate the relationship between accuracy and threshold of detecting bursts. In this experiment, we set the number of monitored windows $NW = 70$ and the maximum number of pieces $m = 8$ in piecewise fractal model. Note that the number of windows NW is for comparison purpose only. It's not used in our algorithm. The error parameter α of piecewise fractal model is set to 7 on D1 and 9 on D3. It can be seen from Figure 5 that under any setting of λ the precision and recall of our method are always above 97 %. The accuracy are even above 99 % when $\lambda = 7$, as shown in Figures 5a and b. We conclude that, with the increasing value of λ, our method can guarantee better accuracy for detecting bursts.

To study the relationship between accuracy of burst detection and error bound of piecewise fractal model, we set number of monitored windows $NW = 70$ and the maximal number of pieces $m = 8$ in piecewise fractal model. The threshold parameter λ of piecewise fractal model is set to 7 on D1 and 5 on D3. It can be seen from Figure 6 that when the value



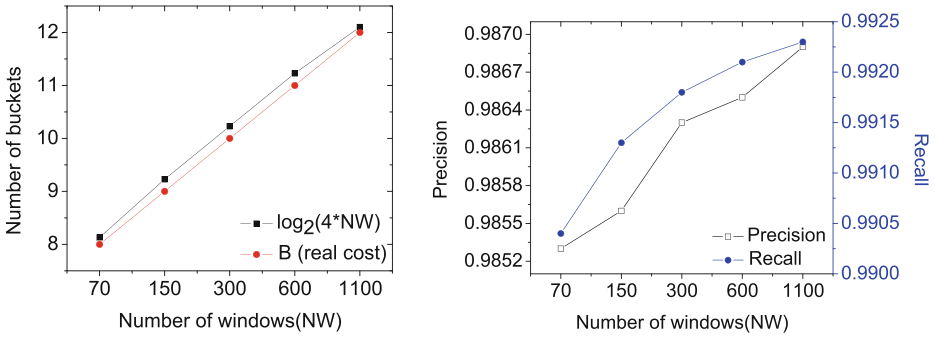
(a) Varying α on D1 (b) Varying α on D3

Figure 6 The accuracy when varying error bound parameter α. (**a**) accuracy when varying α on D1, (**b**) accuracy when varying α on D3

of α increases, the precision and recall also increase in certain domain. When α exceeds certain value, both the precision and recall decrease. α cannot be too small, because smaller α means a piece can only store less data points, hence the length of the overlapping pieces for the piecewise fractal models becomes shorter when the number of pieces m is fixed. Thus the estimated aggregate value of larger windows is more error prone, and the error of burst detection increases. On the other hand, it is easy to understand that larger α result in longer pieces, therefore larger reconstruction error.

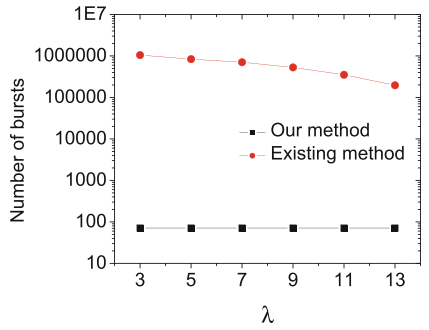
To investigate the relationships between accuracy and number of monitored windows, and the relationship between space cost and number of monitored windows, we set $\lambda = 5$ and $\alpha = 9$. Due to the space limitation, we only show the results on D3. With the increase of number of monitored windows, the number of used pieces also increases. But the number of consumed pieces can still be guaranteed by $m < \log_2 4 * NW$. It can be seen from Figure 7 that with only a few pieces the precision and recall are still high. The accuracy is getting better and better when NW increases. This is because the average size of monitored windows becomes larger with the increase of NW. The fractal approximation is more accurate over large time scales. So the error decreases with the increasing of NW. This illustrate our algorithm's ability to monitor large amounts of windows with high accuracy over streaming data.

In the experiment, we also compare our method with existing burst detection methods [8, 25, 34]. Since these methods have the same definition on burst, we choose [34] to represent the rest. We implement the SWT based algorithm of [34]. When detecting anomaly on data streams, existing methods will return large number of false positive results. Because it is difficult to set the appropriate window size, they have to monitor multiple windows of different sizes. In the experiment, we set $NW = 100$. Since we do not consider the reconstruction of original data stream in this paper, we set $m = 1$ and $\alpha = +\infty$, which means we do not use α to control the error bound of piecewise fractal model. It can be seen from Figure 8 that SWT returns large number of false positive results in every case. For example, in Figure 8b, 10^8 alarms are returned, which is definitely not informative to the users. Moreover, it is difficult to set the threshold of bursts. A slightly smaller or larger λ can either boost false alarms or false positive, respectively. Therefore, our parameter-free method is more adaptable for providing accurate alarms in anomaly detection.

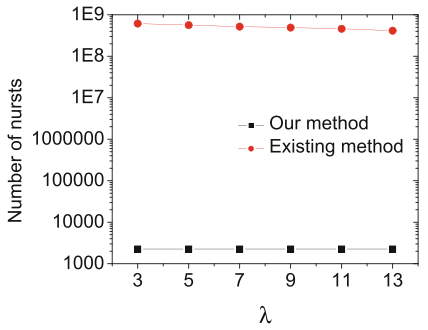


(a) Space cost (b) Accuracy

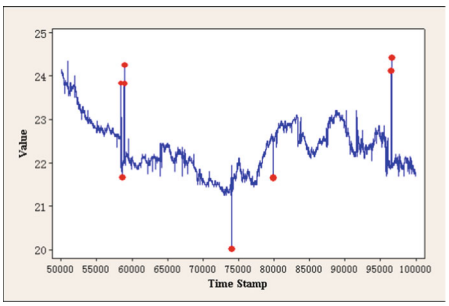
Figure 7 The efficiency of burst detection on D3 when varying number of windows. (**a**) space cost when varying NW (**b**) accuracy when varying NW



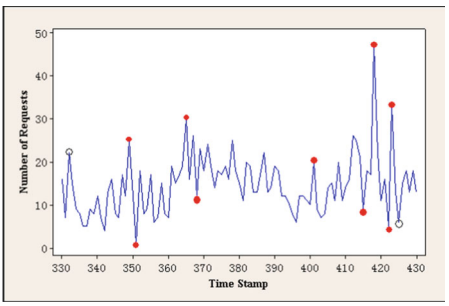
(a) Results Number on D1



(b) Results Number on D3



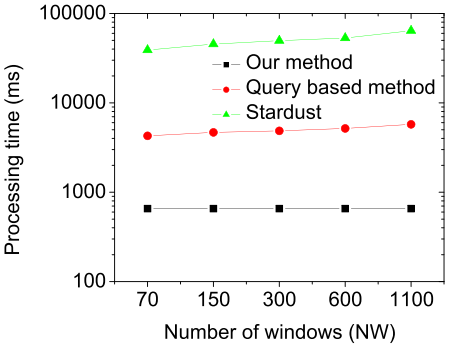
(c) Anomalies on D1



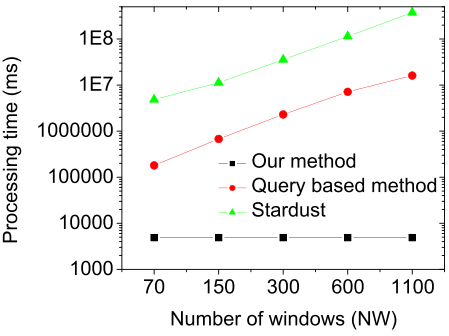
(d) Anomalies on D3

Figure 8 Comparing of number of results, with $NW = 100$, $\lambda = 7$

In Figure 8c and d, solid points are the anomalies identified by our algorithm on two pieces of data streams of data set D1 and D3. It is shown that the result is consistent with the intuition. It is worth noting that the anomalies identified is insensitive to the setting of λ, as it is indicated in Figure 8. Furthermore, the number of anomalies found by the method introduced in [34] is too large (Figure 8a and b), so that those data items cannot be illustrated in Figure 8c and d.

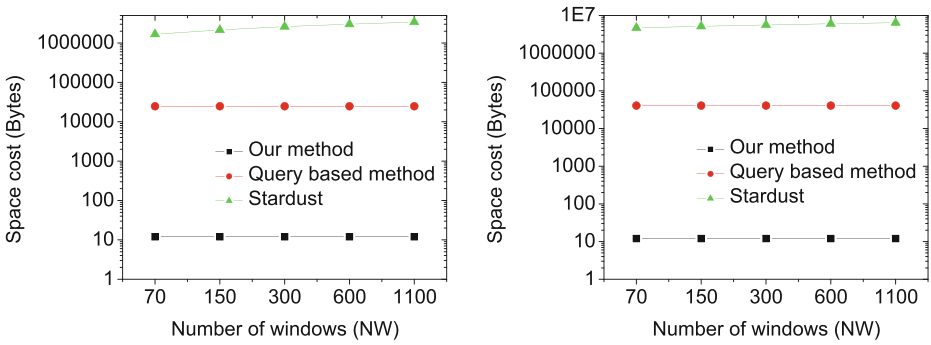


(a) Time complexity on D2



(b) Time complexity on D3

Figure 9 Time complexity comparison on different data sets by varying the number of monitored windows



(a) Space complexity on D2 (b) Space complexity on D3

Figure 10 Space complexity comparison on different data sets by varying the number of monitored windows

There are two data items denoted by hollow points in Figure 8d that having relative high variance but are not identified by our algorithm. This is because the result illustrated is only a small piece in the whole stream. The un-illustrated data items before and after them have similar values to those two items.

To demonstrate the time and space efficiency of our algorithm, we compare with *Stardust* [8] and the query-based method [25]. It can be observed from Figures 9 and 10 that our method uses much smaller memory, runs more than tens of times faster than other methods. We use the code provided by the author of [8] and [25] to do the comparing test. The rest of setting is the same as before, except for the special setting for Stardust with box capacity $c = 2$.

Our method is also space efficient. This is confirmed in Figure 10. The space cost of the method in this paper is the space cost of piecewise fractal model which is only affected by the maintained pieces. In this test, m is set to 1. However, Stardust has to maintain all the monitored data points and the index structure at the same time. And the query-based method has to maintain a inverted histogram (IH) in memory. It can be seen that the space saved by piecewise fractal model is getting larger and larger as the increasing of NW and stream length. Thus, our method is more suitable to detect anomalies over streaming data.

8 Concluding remarks

In this paper, we propose a general-purpose and flexible definition for abnormal behavior. The definition is based on the change of self-similarity of the data streams. We incorporate the fractal analysis technique into anomaly detection. Based on the piecewise fractal model, we propose a novel method for detecting anomalies over a data stream. Fractal analysis is employed to model the self-similar pieces in the original data stream. Piecewise fractal model is proved to provide accurate monitoring results, while consuming only limited storage space and computation time. Fractal techniques focus on the most appropriate window size, not the past data stream history. It frees the users from monitoring a number of windows of different sizes. Both theoretical and empirical results show the high accuracy and efficiency of the proposed method. Furthermore, this approach can also be used to reconstruct the original data stream with the error guaranteed.

Acknowledgements This work is partially supported by National High-tech R&D Program (863 Program) under grant number 2012AA011003, and National Science Foundation of China under the grant number 61232002, and 61170086.

References

1. Arasu, A., Babcock, B., Babu, S., Datar, M., Ito, K., Nishizawa, I., Rosenstein, J., Widom, J.: Stream: The stanford stream data manager. IEEE Data Eng. Bull. **26**(1), 19–26 (2003)
2. Barnsley, M.: Fractals Everywhere. Academic Press, New York (1988)
3. Barnsley, M.F.: Fractal functions and interpolation. Constr. Approx. **2**, 303–329 (1986)
4. Barnsley, M.F., Demko, S.G.: Iterated function schemes and the global construction of fractals. In: Proceedings of the Royal Society (1985)
5. Barnsley, M.F., Elton, J.H., Hardin, D.P.: Recurrent iterated function systems. In: Constructive Approximation (1989)
6. Ben-David, S., Gehrke, J., Kifer, D.: Detecting change in data streams. In: Proceedings of VLDB (2004)
7. Borgnat, P., Flandrin, P., Amblard, P.O.: Stochastic discrete scale invariance. IEEE Signal Process. Lett. **9**(6), 181–184 (2002)
8. Bulut, A., Singh, A.K.: A unified framework for monitoring data streams in real time. In: Proceedings of ICDE (2005)
9. Cai, Y.D., Clutter, D., Pape, G., Han, J., Welge, M., Auvil, L.: Maids: Mining alarming incidents from data streams. In: Proceedings of SIGMOD (2004)
10. Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N., Zdonik, S.B.: Monitoring streams - a new class of data management applications. In: Proceedings of VLDB (2002)
11. Chandrasekaran, S., Deshpande, A., Franklin, M., Hellerstein, J., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M.: Telegraphcq: Continuous dataflow processing for an uncertain world. In: Proceedings of CIDR (2003)
12. Chen, Y., Dong, G., Han, J., Wah, B.W., Wang, J.: Multi-dimensional regression analysis of time-series data streams. In: Proceedings of VLDB (2002)
13. Cormode, G., Muthukrishnan, S.: What's new: Finding significant differences in network data streams. In: Proceedings of INFOCOM (2004)
14. Forte, B., Vrscay, E.R.: Solving the inverse problem for measures using iterated function systems: A new approach. Adv. Appl. Probab. **27**(3), 800–820 (1995)
15. Hart, J.C.: Fractal image compression and the inverse problem of recurrent iterated function systems. In: Proceedings of SIGGRAPH (1994)
16. Hart, J.C.: Fractal image compression and recurrent iterated function systems. IEEE Comput. Graph. Appl. **16**(4), 25–33 (1996)
17. Hartenstein, H., Ruhl, M., Saupe, D., Vrscay, E.R.: Book chapters: On the inverse problem of fractal compression. Ergodic Theory Anal. Efficient Simul. Dyn. Syst., 617–647 (2001)
18. Kleinberg, J.: Bursty and hierarchical structure in streams. In: Proceedings of SIGKDD (2002)
19. Krishnamurthy, B., Sen, S., Zhang, Y., Chen, Y.: Sketch-based change detection: Methods, evaluation and applications. In: Proceedings of IMC (2003)
20. Mandlebrot, B.B.: The Fractal Geometry of Nature. Freeman, New York (1982)
21. Mazel, D.S., Hayes, M.H.: Using iterated function systems to model discrete sequences. IEEE Trans. Signal Process. **40**(7), 1724–1734 (1992)
22. Mitra, S.K., Murthy, C.A., Kundu, M.K., Bhattacharya, B.B.: Fractal image compression using iterated function system with probabilities. In: IEEE International Conference on Information Technology: Coding and Computing (2001)
23. Muthukrishnan, S., Shah, R., Vitter, J.S.: Mining deviants in time series data streams. In: Proceedings of SSDBM (2004)
24. O'Rourke, J.: An on-line algorithm for fitting straight lines between data ranges. Commun. ACM **24**(9), 574–578 (1981). doi:10.1145/358746.358758
25. Qin, S., Qian, W., Zhou, A.: Approximately processing multi-granularity aggregate queries over data streams. In: Proceedings of ICDE (2006)
26. Shahabi, C., Tian, X., Zhao, W.: Tsa-tree: A wavelet-based approach to improve the efficiency of multi-level surprise and tend queries on time-series data. In: Proceedings of SSDBM (2000)
27. Sullivan, M., Heybey, A.: Tribeca: A system for mananging large database of network traffic. In: USENIX (1998)

28. Vlachos, M., Meek, C., Vagena, Z., Gunopulos, D.: Identifying similarities periodicities and bursts for online search queries. In: Proceedings of SIGMOD (2004)
29. Wadstromer, N.: An automatization of barnsley's algorithm for the inverse problem of iterated function systems. IEEE Trans. Image Process. **12**(11), 1388–1397 (2003)
30. Website: Internet traffic archive. http://ita.ee.lbl.gov/ (1989)
31. Wu, X., Barbara, D.: Using fractals to compress real data sets: Is it feasible? In: Proceedings of SIGKDD (2003)
32. Zhou, A., Qin, S., Qian, W.: Adaptively detecting aggregation bursts in data streams. In: Proceedings of DASFAA (2005)
33. Zhu, Y., Shasha, D.: Statstream: Statistical monitoring of thousands of data streams in real time. In: Proceedings of VLDB (2002)
34. Zhu, Y., Shasha, D.: Efficient elastic burst detection in data streams. In: Proceedings of SIGKDD (2003)