Journal of Zhejiang University SCIENCE A
ISSN 1673-565X (Print); ISSN 1862-1775 (Online)
www.zju.edu.cn/jzus; www.springerlink.com
E-mail: jzus@zju.edu.cn

# Monitoring correlative financial data streams by local pattern similarity*

Tao JIANG[†1], Yu-cai FENG[†1], Bin ZHANG[2], Zhong-sheng CAO[1], Ge FU[1], Jie SHI[1]

([1]*College of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*)

([2]*Department of Computer Science, Hengyang Normal University, Hengyang 421008, China*)

[†]E-mail: jiangtao_albert@yahoo.cn; fyc@dameng.com

Received June 12, 2008; Revision accepted Nov. 25, 2008; Crosschecked May 8, 2009

**Abstract:** Developing tools for monitoring the correlations among thousands of financial data streams in an online fashion can be interesting and useful work. We aimed to find highly correlative financial data streams in local patterns. A novel distance metric function slope duration distance (SDD) is proposed, which is compatible with the characteristics of actual financial data streams. Moreover, a model monitoring correlations among local patterns (MCALP) is presented, which dramatically decreases the computational cost using an algorithm quickly online segmenting and pruning (QONSP) with $O(1)$ time cost at each time tick $t$, and our proposed new grid structure. Experimental results showed that MCALP provides an improvement of several orders of magnitude in performance relative to traditional naive linear scan techniques and maintains high precision. Furthermore, the model is incremental, parallelizable, and has a quick response time.

**Key words:** Data mining, Model, Data streams, Correlation, Local pattern, Pattern similarity
**doi:**10.1631/jzus.A0820445 **Document code:** A **CLC number:** TP391

## INTRODUCTION

In many domains, including financial markets and sensor networks, applications consist of data streams. It is very important to monitor correlative data streams for special applications but it is not easy to process such data in an environment of high speed data streams. This is because data stream time series have their own special characteristics compared with traditional archived data. First, in stream time series, data is frequently updated. Thus, previous methods applied to traditional archived data may not work in this scenario. Second, owing to the frequent updates, it is very difficult to store all the data in memory or on disk, therefore, efficient and one-pass algorithms are very important for achieving a real time response.

In this study, we deal with an important scenario in stream applications where incoming data are from a set of continuous financial stream time series. At each timestamp $t$, a new data item is appended to the corresponding financial stream time series. We hope to quickly find all the similar stream pairs up to the current time, that have local pattern distances which do not exceed a user-specified threshold $\varepsilon$, i.e., locally correlated stream pairs. Fig.1 illustrates the problem. It can be seen from Fig.1 that $A$ and $B$ are locally correlated from $t(187)$ to $t(200)$.

Previous studies have approached the problem as sub-sequence similarity matching with distance function $L_p$-norms (Agrawal *et al*., 1993) or dynamic time warping (DTW) (Berndt and Clifford, 1996). However, $L_p$-norms requires two sub-sequences to keep the same length, and DTW has a lower efficiency by a direct implementation. In addition, many methods in these studies focused on detecting a single static pattern over multiple stream time series data, or checking which pattern (from multiple static patterns) was close to a single stream time series up to the

* Project (Nos. 2006AA01Z430 and 2007AA01Z309) supported by the National Hi-Tech Research and Development Program (863) of China

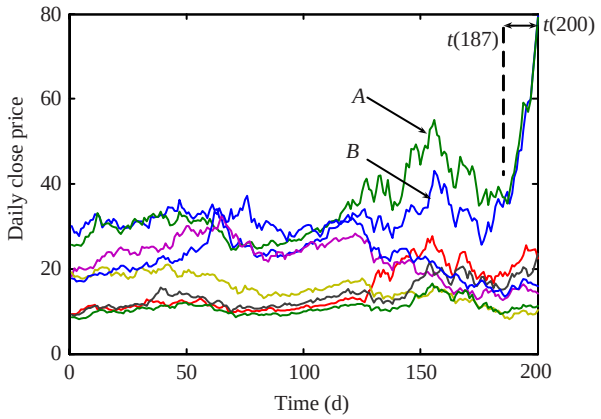


**Fig.1 Eight real stock price curves to illustrate locally correlated financial data streams**

current timestamp. This differs from our goal in that we hope to automatically find all correlative data stream pairs in real time, but not report some correlations with some given patterns or sub-sequences. So, these technologies are not applicable in our scenario.

Some existing models, for example StatStream (Zhu and Shasha, 2002) and BRAID (Sakurai *et al*., 2005), can be used to monitor correlations among financial data streams. StatStream was the first unified scheme to monitor correlations among thousands of data streams in real time. BRAID is also a uniform model which focuses on detecting lag correlations between data streams. Zhu and Shasha (2002) used a sliding window with length $w$ which starts at the timestamp $t-w+1$ and ends at the current timepoint $t$ as temporal span, and discrete fourier transform (DFT) (Agrawal *et al*., 1993) to reduce dimensions. They also proved the theoretical relationship between the correlation coefficient and the normalized Euclidean distance. The main ideas of Sakurai *et al*.(2005) included incremental computing based on algebra and multi-resolution computing that made their approach dramatically faster than the traditional methods.

However, using these two models to monitor the correlation of financial data streams will not be effective because they give little consideration to financial domain features. Firstly, it is difficult to decide the appropriate size of the sliding window. Formally, given two sliding window sub-sequences $S$ and $R$ with a length $w$ ($w\geq 3$), if $S[1{:}k]$ and $R[1{:}k]$ are very similar and $S[k+1{:}w]$ and $R[k+1{:}w]$ are the least similar, $1<k\leq w-1$ ($k\in\mathbb{N}$), then how should we decide the size of $w$? Obviously, we should set up the size of the sliding window $k$. Secondly, using the sliding window will increase the storage space, especially for the lag correlation (Sakurai *et al*., 2005).

In our study, we make use of the algorithm quickly online segmenting and pruning (QONSP) to produce local patterns of financial data streams. We use the slope and duration features to measure each local pattern, and local pattern similarity is based on our defined novel metric function slope duration distance (SDD). We also use a grid structure modified from Zhu and Shasha (2002), to quickly capture the correlations of large amounts of financial data streams. Our method can efficiently monitor all correlative local pattern pairs with small errors.

## RELATED WORK

There has been an increasing interest in data streams in recent years. Many methods for stream pattern discovery have been proposed. Wu *et al*.(2004) defined a new distance metric function to search for similar sub-sequences and the piecewise linear approximation (PLA) (Chen Q., *et al*., 2007) technique was used to segment financial stream time series. Lian *et al*. (2007) used a multi-resolution approach to match a data stream with a given query pattern, using a sliding window. To monitor a given stream pattern, an improved DTW algorithm was developed (Sakurai *et al*., 2007). The spatial assembling distance (SpADe) model (Chen Y.G., *et al*., 2007) can be used to find the patterns based on the shape of whole time series or sub-sequences. However, these studies were focused mainly on pattern similarity matching based on a given pattern (or some fixed patterns), using $L_p$-norms or DTW to measure the similarity between patterns.

It is more difficult to monitor correlative stream pairs than to monitor special patterns over large amounts of data streams because correlative stream pairs need $n(n-1)/2$ comparisons when there are $n$ existing data streams, using the linear scan compared method. The system StatStream (Zhu and Shasha, 2002) was the first unified model to monitor correlations among any pair of data streams in real time within a sliding window, in which DFT was used as a summary of the data. Guha *et al*.(2003) described an improved method for finding correlations by first carrying out dimensionality reduction with random projections and then periodically computing the

singular value decomposition (SVD) (Korn *et al*., 1997). BRAID (Sakurai *et al*., 2005) is implemented to monitor the lag correlation over any pair of data streams. The proposed streaming pattern discovery in multiple time series (SPIRIT) method (Papadimitriou *et al*., 2005) used principle component analysis (PCA) technology and addressed the problem of capturing correlations and finding hidden variables corresponding to trends in collections of data streams. Subsequently, they introduced a method to find optimal local patterns that concisely described the main trend in a time series (Papadimitriou and Yu, 2006), and an approach to track the local correlative time series streams (Papadimitriou *et al*., 2006). Recently, a Boolean representation based on the data-adaptive method (Zhang *et al*., 2007) was proposed for correlation analysis among a large number of time series streams.

However, these approaches gave little consideration to the characteristics of financial data streams except that of Wu *et al*.(2004). Euclidean distance or DTW is used for measuring the distance between two time sequences or patterns in most of these approaches. But it is well-known that Euclidean measure is very sensitive to distortion and noise (Keogh, 2002). Although DTW can handle local time shifting and scaling, it is still sensitive to amplitude shifting. Our method will overcome these problems. To our knowledge, little work has been done using a correlative approach for monitoring financial data streams. Although Zhu and Shasha (2002) and Sakurai *et al*.(2005) proposed a unified model to monitor correlations between data streams, their models do not suit our scenario as described in the Section "INTRODUCTION".

Our work differs from previous studies in some respects. Firstly, we propose a novel distance metric function which adequately considers the features of financial domains. Secondly, we present a novel model monitoring correlations among local patterns (MCALP) which uses a different partitioning and searching policy in its grid structure, and can effectively and efficiently monitor the correlations of financial stream time series.

## SDD

According to Wu *et al*.(2004), financial data streams can be represented by an up-down-up-down repetitive pattern, that is a zigzag shape. So, we can divide a financial data stream into a sequence of linear segments. Generally, the data points of their two ends are called End Points. Thus, a local pattern is the current linear segment, which starts from the last End Point and ends at the current data item.

**Definition 1** (End Point) Given a sliding window $W$ ($=\{s_1, s_2, \ldots, s_w\}$) which consists of $w$ data points from stream time series $S$, $m \le w \le n$, and a threshold $\mu$ ($\mu \in \mathbb{N}$ and $\mu < w$), if $s_i \in W$ ($1 < i \le w$) satisfies the following conditions: (1) $s_i$ is the maximum of $W$; (2) $s_i$ is also the maximum of set $\{s_{w+1}, s_{w+2}, \ldots, s_{w+\mu}\}$; (3) $s_i$ is the last one satisfying the above two conditions, we call $E_i=(s_i, t_i)$ the upper end point where $t_i$ is the corresponding timestamp of $s_i$. Similarly, we can define the lower end point by a symmetric method. An upper or lower end point is called an End Point.

In the above definition, the parameter $\mu$ denotes a delay time which is used to decide the local extreme point, i.e., End Point, and will affect the length of segments during the procedure of segmenting.

A financial market is inconstant owing to the influence of all kinds of factors. However, for a sub-sequence from financial data streams, there are only three possible local trends: an ascend trend, a descend trend or no trend. If the trend continues for a long time, we call the sub-sequence a local pattern. A local pattern *lp* from financial time series $S$ can be described as *lp*=(*trend*, *slope*, *duration*), which includes the local trend of *lp* in $S$, the slope of the linear curve to approximate the original sub-sequence (for example, we can use PLA to approximate the sub-sequence) and the time duration of *lp*, respectively.

**Definition 2** (Local pattern) For a sub-sequence *subS* from a financial stream time series $S$, if it satisfies the following conditions: (1) starting from the last identified End Point $E_{\text{last}}$ and ending at the current data point ($s_{\text{cur}}$, $t_{\text{cur}}$); (2) satisfying the inequality $Err_{\text{PLA}}(subS) / \sum_{i=1}^{n} s_i^2 \le \delta$, where $Err_{\text{PLA}}$ is the approximation error when using PLA technology to approximate *subS* and $\delta$ is a user-defined threshold, for example, $\delta$=0.02; (3) $t_{\text{cur}} - E_{\text{last}}.t > SLen_{\min}$, where $SLen_{\min}$ is the minimum length of *subS*, we call *subS* a local pattern.

In Definition 2, if the current data point ($s_{\text{cur}}$, $t_{\text{cur}}$) is an End Point, the local pattern is called a whole

local pattern; otherwise, it is called a semi-local pattern. Following the above definition, we can now define two local pattern distances, the slope duration distance (Definition 3) and the local pattern similarity (Definition 4).

**Definition 3** (Slope duration distance) Given any two local pattern objects, $lp_1$ and $lp_2$, whose local trends are the same, $SDD(lp_1, lp_2)=\lambda_1|k_1-k_2|/(k_1+k_2)+\lambda_2|t_1-t_2|/(t_1+t_2)$, where $k_i$ denotes the absolute value of the slope of $lp_i$ when using PLA to approximate $lp_i$, and $t_i$ ($t_i\in\mathbb{N}$) denotes the number of data items of $lp_i$, that is the time duration of $lp_i$, $\lambda_1$ and $\lambda_2$ ($0<\lambda_1, \lambda_2<1$ and $\lambda_1+\lambda_2=1$) are user-defined parameters. $f_i$ ($=\{k_i, t_i\}$) are called feature points of $lp_i$. We also define $SDD(f_i, f_j)=SDD(lp_i, lp_j)$.

**Definition 4** (Local pattern similarity) For two local pattern objects, $lp_1$ and $lp_2$, if their distance satisfies the inequality $SDD(lp_1, lp_2)\leq\varepsilon$, $lp_1$ is similar to $lp_2$, $\varepsilon$ is a user-defined threshold, i.e., $\varepsilon$=0.1.

Now, we will explain Definitions 1~4 using Examples 1 and 2, as follows:

**Example 1** Consider stream $R$ in Fig.2. According to the definition of an End Point, points $E_{11}$ (or $r_1$), $E_{12}$ (or $r_7$), and $E_{13}$ (or $r_{16}$) are End Points, but points $r_4$ and $r_6$ are not End Points because $r_4$ is not the maximum of $\{r_1, r_2, r_3, r_4\}$ and $r_6$ does not satisfy the third condition of Definition 1. Here, note that the starting point and last point must be End Points, i.e., $r_1$ and $r_{21}$. Because point $r_7$ is an End Point, the local pattern $lp_{17}=\{r_1, r_2, \ldots, r_7\}$ is a whole local pattern. However, the local pattern $lp_{16}=\{r_1, r_2, \ldots, r_6\}$ is a semi-local pattern.

**Example 2** Consider two local patterns, that is, $lp_{17}^R$ from stream $R$ and $lp_{17}^S$ from stream $S$ in Fig.2 with $\lambda_1$=0.65, $\lambda_2$=0.35 and $\varepsilon$=0.1. According to the definition of $SDD$, $SDD(lp_{17}^R, lp_{17}^S)=0.65|1.705-1.6|/(1.705+1.6)+0.35|7-7|/(7+7)=0.0207$. $f_{17}^R$ ($=\{1.705, 7\}$) and $f_{17}^S$ ($=\{1.6, 7\}$) are feature points of $lp_{17}^R$ and $lp_{17}^S$. Obviously, $lp_{17}^R$ is similar to $lp_{17}^S$ on the condition of $SDD(lp_{17}^R, lp_{17}^S)\leq 0.1$.

Next we will show that our distance function $SDD$ is a metric function by Theorem 1 and we can use metric distance indexing methods for a faster search. First, we introduce the following lemmas.

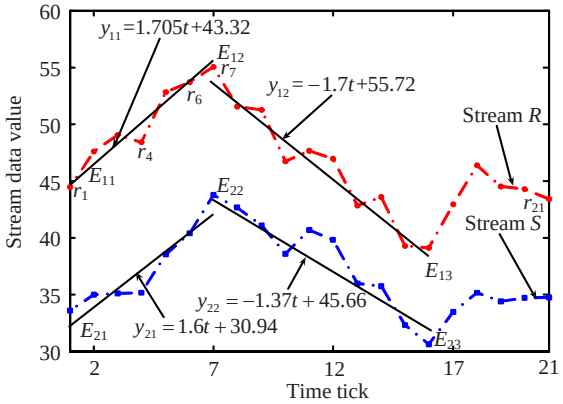


**Fig.2 An example to illustrate Definitions 1~4 with $SLen_{min}$=5, $\mu$=5 and $\delta$=0.02**

**Lemma 1** If $a, b, c\geq 0$, then the following inequality is true:

$$\frac{|a-c|}{a+c}\leq\frac{|a-b|}{a+b}+\frac{|b-c|}{b+c}. \tag{1}$$

**Proof** We prove Lemma 1 by induction. For three non-negative numbers $a$, $b$ and $c$, there are six possible arrangements, that is, $a\geq b\geq c$, $a\geq c\geq b$, $b\geq a\geq c$, $b\geq c\geq a$, $c\geq a\geq b$ and $c\geq b\geq a$. Firstly, we consider the case $a\geq b\geq c$.

Assume inequality Eq.(1) is correct, then $a\geq b\geq c\Rightarrow(a-c)/(a+c)\leq(a-b)/(a+b)+(b-c)/(b+c)$. If $a=0$ or $b=0$ or $c=0$, obviously the above inequality is true. So, we need only to consider the situation of $a>b>c>0$. Thus, we have $(a-c)/(a+c)\leq\{(a-b)(b+c)+(a+b)(b-c)\}/\{(a+b)(b+c)\}$, that is $(a-c)/(a+c)\leq 2(ab-bc)/\{(a+b)(b+c)\}$. Multiplying $(a+c)(a+b)(b+c)$ on both sides of the above inequality, we have $(a-c)(a+b)(b+c)\leq 2(a+c)(ab-bc)$, that is, $ac+b^2\leq ab+bc\Rightarrow b(b-c)\leq a(b-c)$. Because $b-c>0$, we have $b\leq a$. So Lemma 1 is correct in the first case.

The other cases can be proved using a procedure similar to that used in the first case. Here, we omit them due to space limitations. So, Lemma 1 is correct.

**Lemma 2** If $a, b, X_1, X_2, Y_1, Y_2\geq 0$, $X_1\leq X_2$ and $Y_1\leq Y_2$, then $a\cdot X_1+b\cdot Y_1\leq a\cdot X_2+b\cdot Y_2$.

**Theorem 1** Given two local pattern objects $lp_1$ and $lp_2$, their distance $SDD(lp_1, lp_2)$ is metric.

**Proof** To prove $SDD(lp_1, lp_2)$ is metric, we need to prove that it is symmetric and reflexive, and that it

satisfies the triangle inequality. It is obvious that $SDD(lp_1, lp_2)=SDD(lp_2, lp_1)\geq 0$ and $SDD(lp_1, lp_2)=0$, so $SDD(lp_1, lp_2)$ is symmetric and reflexive. Next, we will prove that $SDD(lp_1, lp_2)$ satisfies the triangle inequality, i.e.,

$$SDD(lp_1, lp_3)\leq SDD(lp_1, lp_2)+SDD(lp_2, lp_3). \quad (2)$$

By the definition of the distance function *SDD*, it is obvious that the distances of two local patterns are the summation of a slope component and a duration component. If we prove the triangle equality for both components, it will certainly be true for *SDD* by Lemma 2. Through Lemma 1, we have the following two equalities, that is, $|k_1-k_3|/(k_1+k_3)\leq|k_1-k_2|/(k_1+k_2)+|k_2-k_3|/(k_2+k_3)$ and $|t_1-t_3|/(t_1+t_3)\leq|t_1-t_2|/(t_1+t_2)+|t_2-t_3|/(t_2+t_3)$. In other words, inequality Eq.(2) is correct in each component of Theorem 1. So, Theorem 1 is true.

## OUR PROPOSED MCALP MODEL

### Segmenting and pruning of stream time series

Compared with DFT (Zhu and Shasha, 2002), PLA provides greater fidelity. So, we use PLA to approximate each local pattern (*lp*) of financial data streams. Formally, for a given local pattern, $lp=(s_1, s_2, \ldots, s_n)$ of length $n$, PLA can use one line segment, $y_t=a\cdot t+b$ ($t\in[1, n]$), to approximate *lp*, where $a$ and $b$ are two coefficients in a linear function such that the approximation error, $Err_{\mathrm{PLA}}(lp)$, of *lp* is minimized. $Err_{\mathrm{PLA}}(lp)$ is defined in the following Eq.(3) by the squared Euclidean distance between the approximated and actual time series segments:

$$Err_{\mathrm{PLA}}=\sum_{t=1}^{n}(s_t-(a\cdot t+b))^2. \quad (3)$$

In particular, $a$ and $b$ must satisfy two conditions:

(1) $\dfrac{\partial Err_{\mathrm{PLA}}(lp)}{\partial a}=0$, and (2) $\dfrac{\partial Err_{\mathrm{PLA}}(lp)}{\partial b}=0$.

Through solving the two equations, we can obtain the coefficients of $a$ and $b$ using the following equations:

$$a=12\sum_{t=1}^{n}\frac{t-(n+1)/2}{n(n+1)(n-1)}s_t, \quad (4)$$

$$b=6\sum_{t=1}^{n}\frac{(2n+1)/3-t}{n(n-1)}s_t. \quad (5)$$

When segmenting stream time series, $Err_{\mathrm{PLA}}$ will also be computed. If $Err_{\mathrm{PLA}}(lp)\leq\sigma$ is satisfied, we approximate local pattern *lp* with PLA, where $\sigma$ is a user-defined threshold, i.e., $\sigma=10$; otherwise, we throw away the local pattern. In fact, the above condition is not precise because different streams may have a bigger difference value on their data points. So, we define the relative approximation error as $RErr_{\mathrm{PLA}}$, that is, the improved condition:

$$RErr_{\mathrm{PLA}}(lp)=Err_{\mathrm{PLA}}\Big/\sum_{i=1}^{n}s_i^2\leq\delta, \quad (6)$$

where $\delta$ is a user-defined threshold, i.e., $\delta=0.02$.

To decrease the impact of shorter-time random oscillation, we use moving average technology, which is widely used in financial markets, to smooth financial data streams. Formally, given a data stream *S*, the value of the current timestamp $i$ is $S(i)$, and the $r$-period moving average is $MA_r(i)=\frac{1}{r}\sum_{j=i-r+1}^{i}S(j)$, where $S(j)$ is the previous value of timestamp $i$ ($i-r+1\geq 1$), including the current timestamp, and $r$ denotes the number of aggregations over the time interval of the moving average (i.e., one day, one week or one month) which assigns equal weight to every point in the averaging interval. In our scheme, we generally set the parameter value of $r$ to 3 or 5.

### QONSP algorithm

The QONSP algorithm uses the sliding window method and a data-driven mechanism to segment multiple data streams in real time. It receives all newly arrived data $dv_S$ from stream $S\in SS$, where *SS* is the set of all data streams. Then, it indexes $dv_S$ into a $B^+$-tree index, $index_{lp}$ (subsection "Update of grid structure and index"). Next, it invokes the algorithm Calculating_Err to compute $RErr_{\mathrm{PLA}}$. Finally, it makes use of two policies to finish segmentation: (1) judging if the current new data $dv_S$ is the extreme value (maximum or minimum) of $lp_S$; (2) using $\mu$ to confirm if *Exmum* is the local extreme value. In the QONSP algorithm, there exist some temporary variables. Among them, $Off_{\mathrm{DS}}$, $Off_{\mathrm{Seg}}$, and $Off_{\mu}$ represent

the offset of the data stream, the offset of the segment, and the offset of $dv_S$ relative to the extreme value, *Exmum*, respectively, and *posExmum* denotes the offset of *Exmum* relative to left End Point.

The detailed algorithm is shown below as Algorithm 1. In Algorithm 1, lines 7~22 divide financial time series into line segments, according to $RErr_{\mathrm{PLA}}(lp)$ computed by the Calculating_Err algorithm (line 7). Pruning is implemented along with the segmentation process. At line 8, QONSP judges if the current data $dv_S$ is the extreme value of the current segment *lp*. When it is true, *Exmum* and *posExmum* are used to store its value and its position (line 10), respectively. At the same time, segmentation begins to move to the next point (line 11). When it is false, QONSP adds 1 to $Off_\mu$ (line 12). If $Off_\mu$ passes the threshold $\mu$, we output the new segment $Seg_{\mathrm{new}}$ (line 14), and move to the next segment (line 16). Line 17 makes segmentation move to the next point. In fact, a new segment will also come into being when $RErr_{\mathrm{PLA}}\geq\delta$. Here, it includes two cases: (1) *Exmum* is a local extreme value of *lp*, which is denoted by the condition $Off_\mu>0$ at line 20; (2) *Exmum* is not a local extreme value of *lp*, indicated by $Off_\mu==0$. For the former, QONSP increases the offset of $Off_{\mathrm{DS}}$ with *posExmum*−1 (lines 15, 20). However, for the latter, QONSP adds offset $Off_{\mathrm{Seg}}-2$ (remove current point) to $Off_{\mathrm{DS}}$ at line 21. In QONSP, the new segment $Seg_{\mathrm{new}}$ begins at the current position of $Off_{\mathrm{DS}}$ and ends at the next position of $Off_{\mathrm{DS}}$, i.e., $Seg_{\mathrm{new}}=S(Off_{\mathrm{DS}}[S]:Off_{\mathrm{DS}}[S]+posExmum-1)$.

**Algorithm 1** QONSP ($SS$, $\mu$, $\delta$, $SLen_{\min}$)

Input: *SS*—the set of streams,
$\mu$—a threshold of delay time,
$\delta$—the maximal relative approximation error,
$SLen_{\min}$—the shortest length of local pattern.

Output: whole local patterns.

```
1  Initialize arrays Off_DS[], Off_Seg[], Off_μ[] with 1, 2, 0, and
   index_lp with first data from all streams T_i∈SS;
2  while (segmentation is not finished)
3    Receive a new data dv_S from stream S∈SS;
4    Update index_lp with dv_S;
5    if (Off_Seg[S]<SLen_min) {Off_Seg[S]++;
       continue;}
6    Get local pattern lp of S from index_lp;
7    if (Calculating_Err(lp)<δ)
8      if (dv_S is the extreme value of lp) {
9        Off_μ=1; // Off_μ—the offset vs Exmum
10         Exmum=dv_S; posExmum=Off_Seg;
11       Off_Seg[S]=Off_Seg[S]+1; continue;}
12     if (1≤Off_μ<μ) Off_μ++;
13     if (Off_μ==μ) {
14       Off_μ=0; Output new segment Seg_new;
15       Off_DS[S]=Off_DS[S]+posExmum−1;
16       posExmum=Off_DS[S]=2;} // to next segment
17       Off_Seg[S]=Off_Seg[S]+1; // to next point
18     else {
19       Output new segment Seg_new;
20       if (Off_μ>0) Off_DS[S]= Off_DS[S]+posExmum−1;
21       if (Off_μ==0) Off_DS[S]=Off_DS[S]+ Off_Seg[S]−2;
22         Off_μ=0; posExmum=Off_Seg[S]=2;}
```

Now, we will give an example to illustrate the processing of segmentation as follows:

**Example 3** Consider the stream time series *R* in Fig.3. In the beginning, $Off_{\mathrm{DS}}[R]$, $Off_{\mathrm{Seg}}[R]$ and $Off_\mu[R]$ are initialized with 1, 2 and 0. The first End Point is $E_1$ (or $r_1$), which is used to initialize $index_{lp}$. Then, the data $r_1$ is received at tick $t$=2 and inserted into $index_{lp}$. At present, *lp* is $\{r_1, r_2\}$. When line 5 of the QONSP algorithm is finished, $Off_{\mathrm{Seg}}[R]$ increases to 5 and *lp* turns into $\{r_1, r_2, \ldots, r_5\}$. Because $RErr_{\mathrm{PLA}}(lp)<0.03$, QONSP enters line 8 and starts to judge if $r_5$ is the extreme value of *lp*. In fact, $r_5$=1.634 ($<r_1$=1.636) is the minimum of *lp*. So, the algorithm now sets up $Off_\mu$, *Exmum* and *posExmum* with 1, $r_5$ and 5, respectively, and will be ready for the next data $r_6$ ($Off_{\mathrm{Seg}}[R]$=6). Because $r_5<r_6$, $r_5<r_7$, we have $Off_\mu$=1+1+1=3 by line 12. But, when $Off_{\mathrm{Seg}}[R]$=8, $r_8$=1.628 is less than $r_5$, then $Off_\mu$, *Exmum* and *posExmum* change into 1, $r_8$ and 8, respectively. Repeating the process until $Off_{\mathrm{Seg}}[R]$=22, $Off_\mu$, *Exmum* and *posExmum* change into 1, $r_{22}$ and 22, respectively. Obviously, some shorter segments, i.e., $\{r_{11}, r_{12}\}$ and $\{r_{14}, r_{15}, r_{16}\}$, are pruned during the process. Because $r_{22}<r_{23}$, $r_{24}$, $r_{25}$, $r_{26}$, $r_{27}$, and the corresponding $RErr_{\mathrm{PLA}}(lp)<\delta$ (i.e., $RErr_{\mathrm{PLA}}(\{r_1, r_2, \ldots, r_{27}\})<\delta$), then $Off_\mu$ arrives at $\mu$ (=6). Therefore, QONSP outputs the new segment $Seg_{\mathrm{new}}=\{r_1, r_2, \ldots, r_{22}\}$ and will begin to process the next segment by lines 14~16. Now, the new End Point is $E_2$ (or $r_{22}$). By a similar analysis, the next End Points will be $E_3$, $E_4$ and $E_5$, and the next segments are $\{r_{22}, r_{23}, \ldots, r_{31}\}$, $\{r_{31}, r_{32}, \ldots, r_{36}\}$, $\{r_{36}, r_{37}, \ldots, r_{50}\}$.

In the algorithm Calculating_Err, efficiency is very important over high speed data streams. To decrease the cost of computing, we use a buffer $BufMA_r$ to store the last $r$ raw data item value $s_i$ ($i$ is

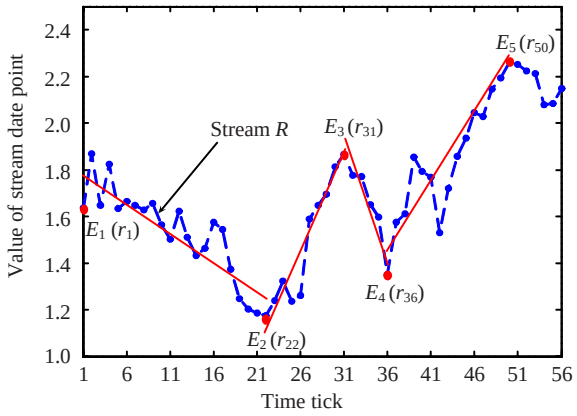


**Fig.3 An example to illustrate the QONSP algorithm with parameters $SLen_{min}$=5, $\delta$=0.03 and $\mu$=6**

timestamp) and a variable $lastMA_r$ to store the last $r$-period moving average when computing the current $r$-period moving average $curMA_r$ at timestamp $t$. Thus, we can compute $curMA_r$ by the equation $curMA_r=(lastMA_r-BufMA_r(t-r+1)+s_i)/r$. When computing the coefficients $a$, $b$ and $Err_{PLA}$ (lines 2~15 in Algorithm 2), we make use of an accumulative method which only needs three variables, $PreSum_{ts}$, $PreSum_s$ and $PreSum_{ss}$, to store the last corresponding aggregation values, and we can obtain the current aggregation values only by a addition operation. We transform $a$, $b$, $Err_{PLA}$ and $RErr_{PLA}$ into incremental computing forms by rewriting Eqs.(4), (5), (3), and (6), respectively. Then, they can be quickly computed by the following Eqs.(7)~(10):

$$a=\frac{12(PreSum_{ts}-PreSum_s\cdot(t+1)/2)}{t(t+1)(t-1)}, \quad (7)$$

$$b=\frac{6(PreSum_s\cdot(2t+1)/3-PreSum_{ts})}{t(t-1)}, \quad (8)$$

$$Err_{PLA}=PreSum_{ss}+a^2t(t+1)(2t+1)/6+abt(t+1)+b^2t-2aPreSum_{ts}-2bPreSum_s, \quad (9)$$

$$RErr_{PLA}=Err_{PLA}/PreSum_{ss}, \quad (10)$$

where $1\le t\le n$. Of course, there is still a need of a circle to implement when the length of pattern segment is equal to $SLen_{min}$ (lines 4~8). In fact, Calculating_Err is only invoked at that time. The detailed algorithm is as Algorithm 2.

**Algorithm 2** Calculating_Err ($PreSum_{ts}$, $PreSum_s$, $PreSum_{ss}$, $s_t$, $t$)

Input: $PreSum_{ts}$—the last sum of previous $Off_{Seg}$ data $s_t\cdot t$, $PreSum_s$—the last sum of previous $Off_{Seg}$ data $s_t$, $PreSum_{ss}$—the last sum of previous $Off_{Seg}$ data $s_t\cdot s_t$, $Seg_{cur}$—current data of current segment, $t$—length of $Seg_{cur}$.

Output: relative approximation error $RErr_{PLA}$.

```
1  Compute moving average A_r(S) for current data s_t;
2  if (Off_Seg==SLen_min)
3    PreSum_ts=PreSum_s=PreSum_ss=0;
4  for i=1 to Off_Seg do
5    PreSum_ts=PreSum_ts+i·s_i;
6    PreSum_s=PreSum_s+s_i;
7    PreSum_ss=PreSum_ss+ s_i·s_i;
8  end for
9  else
10   PreSum_ts=PreSum_ts+t·s_t;
11   PreSum_s=PreSum_s+s_t;
12   PreSum_ss=PreSum_ss+s_t·s_t;
13 end if
14 Compute coefficients a and b by Eqs.(7) and (8);
15 Compute Err_PLA and RErr_PLA by Eqs.(9) and (10);
```

Specifically, the efficiency of QONSP can be described by Theorem 2:

**Theorem 2** Given $m$ data streams with length $n$, the QONSP algorithm is a one-pass algorithm and can finish segmentation in linear time $O(mn)$.

**Proof** Owing to the adoption of a sliding window, QONSP is clearly a one-pass algorithm. Firstly, we will prove that the Calculating_Err algorithm only needs constant time $O(1)$ to compute $RErr_{PLA}$ at each time tick $t$, irrespective of the length of segment $Len_{Seg}$. By the previous analysis, we know that Calculating_Err algorithm adopts an accumulative method to obtain the corresponding aggregation values, only by an addition operation, and then can compute $RErr_{PLA}$ using Eqs.(7)~(10). So, the above proposition is correct. Meanwhile, we also know that the total data points are $m\times n$. Thus, the total time of segmentation will be $m\times n\times O(1)=O(mn)$ for $m$ streams with length $n$.

**Monitoring correlations among data streams**

Monitoring correlations among thousands of data streams and then efficiently reporting the most correlative data stream pairs is not a trivial task since these patterns usually reside in large amounts of high dimensional and noisy data. The data distribution and underlying clustering structure may also change whereby a previously uncovered pattern may become

obsolete as time goes by. It is no different from the sub-sequence similarity matching technology in which the query patterns are known beforehand.

So finding an efficient approach to solve this problem is the key to success. One straightforward way is by indexing each local pattern into an $R^*$-tree. However, the method is inefficient, in terms of both space and search speed. This is because almost every local pattern will correspond to a point in the ($k$, $t$) feature space. Moreover, the search performance will also suffer because the $R^*$-tree will become tall and slow. In addition, our system needs to update frequently. So, it will not be competent for our task, owing to the index cost being too high. Due to the characteristics of high speed and frequent updating, the clustering method also cannot complete the task efficiently.

1. Grid structure partition

Through the analysis of the previous section, we know that the indexing and clustering methods are not well suited to monitoring the correlations among large amounts of data streams. However, a grid structure (Bentley *et al.*, 1980) provides a good scheme to report the most correlative local patterns. The improved performance of the scheme depends on the following factors: (a) the data distribution of the local pattern points; (b) how the grid structure is partitioned; (c) how the grid structure is updated; (d) having a perfect searching algorithm.

Firstly, let us consider the point distribution of local patterns. In fact, we can obtain the statistical information from historical data. We need only to get the maximum for the duration and slope of each local pattern, respectively, because the minimal duration is certainly 1, and the minimal slope is 0 when considering the local pattern trend. Generally, we use the previously defined threshold $SLen_{\min}$ for the minimal duration, because it is not important to report correlative local patterns that are very short.

After finding the distribution of points, we can divide the grid structure into grid units. The partitions are obtained by dividing the domain of each local pattern attribute into intervals. The critical problem is how to decide the length of intervals and the number of intervals. One simple method is to use equal intervals and then to keep a suitable number of intervals in the direction of each axis, that is, $Num_k$ and $Num_t$, which play an important role in the number of grid units $Num_g$ ($=Num_k \times Num_t$). A smaller $Num_g$ will reduce the number of empty units but result in poor searching performance; a larger $Num_g$ will increase the space. So we should keep a balance between them. A good approach is to keep a few empty units. In our scheme, we use the straightforward method to partition the grid structure with the maximal $t$=99, the maximal $k$=2.9 (that is tan 71°). The intervals of slope and duration are 3 and 0.3, respectively.

2. Grid searching algorithm (GSA)

In our model, the definition of correlation is different from that of (Zhu and Shasha, 2002) who used the standard correlation coefficient equation, namely the Pearson correlation equation. We consider not only the similarity, but also the number of similar local patterns. In other words, we consider only some special types of correlations, for example, the minimal number correlation (denoted as *minCorr*) which includes the least similar local patterns and the maximal number correlation (denoted as *maxCorr*) which contains the most similar local patterns. This idea is derived from the following observations:

**Observation 1** In financial markets, many factors will affect the trend of a financial data stream. It is not generally a coincidental phenomenon, if two local patterns $lp_1$ and $lp_2$ are highly correlative. It may be caused by an underlying event, for example, the national economic policy or an incident within corporations. If the causal factor is the economic policy, generally most financial data streams will be affected and they will maintain the same movement trend. So many local patterns will remain correlative. However, if a specific incident occurs within corporations, there will be few correlative local patterns.

**Observation 2** Finding the *maxCorr* is less important than finding the *minCorr* because it is easier to obtain information on national economic policy than information within corporations.

**Observation 3** A local pattern point *lpPoint* can be treated as a random variable. When mapping all *lpPoints* from all data streams to a grid structure, there are some empty grid units, a few grid units with a few *lpPoints* and a few grid units containing many *lpPoints*.

Based on the above three observations, our model will focus on finding *minCorr*. Of course, other situations can be monitored according to need, for example, finding *maxCorr* or all correlative local

patterns. We denote the GSA as Algorithm 3 which uses a data-driven mechanism reflected in lines 2~9. GSA can selectively monitor special types of correlations, for example *minCorr*, by line 4, and thereby decrease the computation cost. In line 5, GSA invokes the subroutine Correlative_Search (as Algorithm 4).

**Algorithm 3** *GSA* ($\varepsilon$, $p$)
Input: $\varepsilon$—the threshold of local pattern similarity,
$p$—the number of searching grid units.
Output: the correlative local pattern pairs.
1 Construct the grid structure according to the data distribution of local pattern points and the corresponding partition method;
2 While (a new data item ($s_t$, $t$) arrives)
3 Insert the corresponding semi-local pattern point grid structure by the hashed method;
4 Sort the number of local pattern points contained in each grid unit in ascending order;
5 Choose the first $p$ grid units and invoke *Correlative_Search* to report the correlative local patterns;
6 If (the data item ($s_t$, $t$) is an End Point)
7 Clear all local pattern points of the End Point from the corresponding grid units;
8 End if
9 End While

The detailed algorithm of Correlative_Search is as follows:

**Algorithm 4** Correlative_Search (*pArray*, $\varepsilon$)
Input: *pArray*—an array storing the number of feature points of the first $p$ grid units,
$\varepsilon$—local pattern similarity threshold.
Output: the correlative local pattern pairs.
1 for ($i$=1 to $p$)
2 for ($j$=1 to *pArray*[$i$])
3 Obtain the searching rectangle $Rec_j$ of current local pattern $lp_j$ by Theorem 3;
4 Get all adjacent cells of $lp_j$, $Cell_{\text{adj}}^{j}$, by $Rec_j$;
5 for (each local pattern $lp_k \in Cell_{\text{adj}}^{j}$)
6 Skip having compared pairs or the pairs of $j$=$k$ or self-correlative pairs;
7 if ($SDD(lp_j, lp_k)\le\varepsilon$)
8 Report the local pattern pair ($lp_j$, $lp_k$);
9 end if
10 end for
11 end for
12 end for

Correlative_Search is quite straightforward and its theoretical foundation is in the following theorem:

**Theorem 3** Given the distance threshold $\varepsilon$ of *SDD* function and a feature point $A=(k_a, t_a)$, if the grid structure is partitioned by the equal interval of the features of $k$ and $t$, for any point $P$ of the rectangle which is the region formed by extending $A$ leftward and downward $2\varepsilon/(1+\varepsilon)$, rightward and upward $2\varepsilon/(1-\varepsilon)$, respectively, $SDD(P, A)\le\varepsilon$ is true.

**Proof** Firstly, consider the case $k_p\le k_a$. By the condition, we have $k_a-k_p\le k_a\cdot 2\varepsilon/(1+\varepsilon)\Rightarrow k_a-k_p\le(k_a+k_p)\varepsilon$, so we have $|k_a-k_p|/(k_a+k_p)\le\varepsilon$. Next considering the case: $k_a\le k_p$, by the condition, we have $k_p-k_a\le k_a\cdot 2\varepsilon/(1-\varepsilon)\Rightarrow k_p-k_a\le(k_a+k_p)\varepsilon$, so we have $|k_a-k_p|/(k_a+k_p)\le\varepsilon$. Synthesize the case $k_p\le k_a$ and $k_p\ge k_a$ $\Rightarrow|k_a-k_p|/(k_a+k_p)\le\varepsilon$. Multiplying $\lambda_1$ at its two sides, we have the following inequality:

$$\lambda_1 \,|\, k_a - k_p \,|\, /(k_a + k_p) \le \lambda_1\varepsilon. \tag{11}$$

By a similar procedure, we have $|t_a-t_p|/(t_a+t_p)\le\varepsilon$. Multiplying $\lambda_2$ at its two sides, we have the following inequality:

$$\lambda_2 \,|\, t_a - t_p \,|\, /(t_a + t_p) \le \lambda_2\varepsilon. \tag{12}$$

From Eqs.(11) and (12), we can obtain $\lambda_1|k_a-k_p|/(k_a+k_p)+\lambda_2|t_a-t_p|/(t_a+t_p)\le\lambda_1\varepsilon+\lambda_2\varepsilon$. Due to $\lambda_1+\lambda_2=1$, we have $\lambda_1|k_a-k_p|/(k_a+k_p)+\lambda_2|t_a-t_p|/(t_a+t_p)\le\varepsilon$, i.e., $SDD(P, A)\le\varepsilon$.

A visual representation of the searching region *Rec* of Theorem 3 is shown in Fig.4. From Fig.4, we can observe that the searching range of the feature point $A$, which is located in grid unit *cell* (4, 4), is $[k_a-k_a\cdot 2\varepsilon/(1+\varepsilon),\ k_a+k_a\cdot 2\varepsilon/(1-\varepsilon)]$ and $[t_a-t_a\cdot 2\varepsilon/(1+\varepsilon),\ t_a+t_a\cdot 2\varepsilon/(1-\varepsilon)]$. In $Rec_a$, feature points $B$, $C$, $D$, $E$, $F$ and $G$ are the candidate items.

The following example illustrates the process of Algorithm 3 and Algorithm 4:

**Example 4** Consider the three real streams $R$, $S$ and $T$ in Fig.5 with $\lambda_1$=0.65, $\lambda_2$=0.35, and $\varepsilon$=0.1, where $f_{xn}$ denotes the feature point formed by the first $n$ data points of stream $X$, i.e., $f_{r7}$. After the time tick $t$=5, three feature points $f_{r5}$={1.76, 5}, $f_{s5}$={1.0, 5}, and $f_{t5}$={2.64, 5} will be inserted into grid units *cell* (1, 3), *cell* (1, 1) and *cell* (1, 6) respectively, where *cell* ($m$, $n$) corresponds to the grid unit of the $m$th interval at

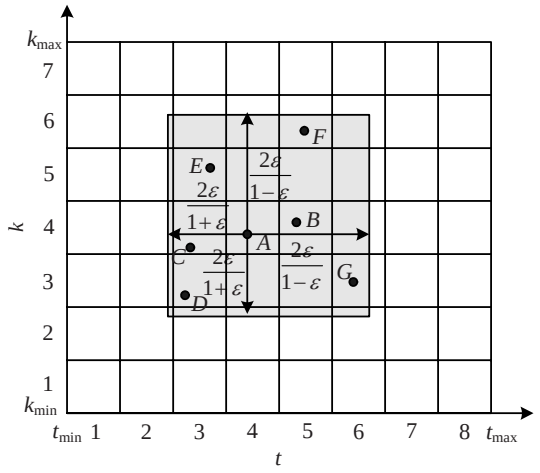


**Fig.4 Grid structure used to illustrate the searching range of the feature point *A***

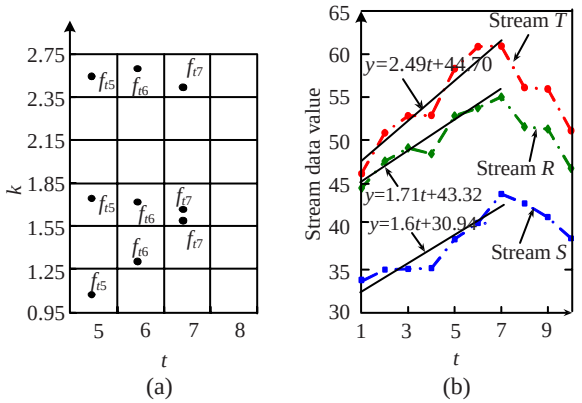


**Fig.5 A visual representation of Algorithms 3 and 4 with parameters $\varepsilon$=0.1, $p$=2. (a) Grid structure; (b) Three stream sequences *R*, *S* and *T***

*t*-axis and the *n*th interval at *k*-axis. Then, GSA invokes Correlative_Search to report correlative local pattern pairs. According to Theorem 3, the search range of $f_{r5}$ at *k*-axis should be $[k_5^R - k_5^R \cdot 2\varepsilon/(1+\varepsilon), k_5^R + k_5^R \cdot 2\varepsilon/(1-\varepsilon)]$, that is, [1.76−1.76×2×0.1/(1+0.1), 1.76+1.76×2×0.1/(1−0.1)]=[1.44, 2.15]. Clearly, $f_{s5}$ and $f_{t5}$ are not included in the range, so no correlative local pattern pairs are reported. At $t$=6, $f_{r6}$={1.75, 6}, $f_{s6}$={1.27, 6}, and $f_{t6}$={2.74, 6} are inserted into *cell* (2, 2), *cell* (2, 3), and *cell* (2, 6), respectively. But, by a similar computation, GSA does not find correlative pairs. However, when $t$=7, $f_{r7}$={1.71, 7}, $f_{s7}$={1.6, 7}, and $f_{t7}$={2.49, 7}. Now, the ranges of $f_{r7}$ at *t*- and *k*-axis are [1.71−1.71×2×0.1/(1+0.1), 1.71+1.71×2×0.1/(1−0.1)]=[1.39, 2.09], [7−7×2×0.1/(1+0.1), 7+7×2×0.1/(1−0.1)]=[5.72, 8.56], respectively. We can see clearly that $f_{r6}$ and $f_{s7}$ are contained in the above range. However, because $f_{r6}$ and $f_{r7}$ are a self-correlative pair, we consider only local pattern pair, $f_{r7}$ and $f_{s7}$. In fact, $SDD(f_{r7}, f_{s7})$=0.65×|1.71−1.6|/(1.71+1.6)+0.35×|7−7|/(7+7)=0.0216≤$\varepsilon$, so GSA outputs $\{r_1, r_2, \ldots, r_7\}$ and $\{s_1, s_2, \ldots, s_7\}$ as correlative pairs.

3. Update of grid structure and index

The system includes a grid index $index_{grid}$ and a local pattern index $index_{lp}$. The $index_{grid}$ is used to maintain the grid structure. The $index_{lp}$ is a $B^+$-tree index, which is used for segmenting and pruning, as well as updating grid structure. The structure of $index_{grid}$ is shown in Fig.6 and consists of a set of lists corresponding to grid units.

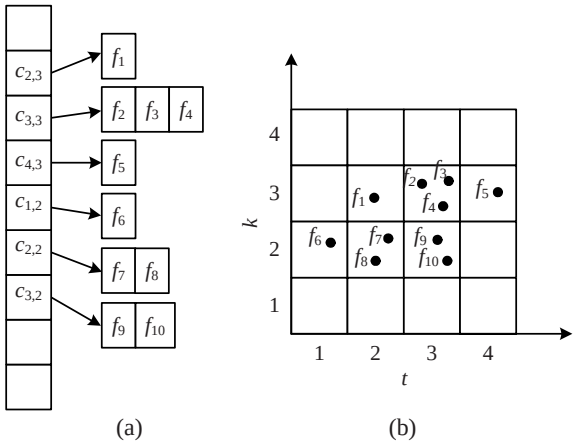


**Fig.6 An example to illustrate the grid index structure. (a) Index based on grid structure; (b) Grid structure**

The $index_{grid}$ is a hashed index. Each grid unit $c(i, j)$ corresponds to a fixed position of $index_{grid}$, and the position has a pointer pointing at the list $l(i, j)$ which contains all the information of $c(i, j)$ and each node of $l(i, j)$ is a triangular tuple (*StreamID*, *k*, *t*).

The $index_{lp}$ indexes two types of data. The first type contains all data items (=$\{s_1, s_2, \ldots, s_{cur}\}$) of the current local pattern *lp* which begins from the last identified End Point $EP_{last}$ and ends right before the current data point ($s_{cur}$, $t_{cur}$) from all data streams, so as to find the next new End Point $EP_{new}$. Whenever identifying $EP_{new}$, we clear the space occupied by these data items from $EP_{last}$ to the data item before $EP_{new}$ so that the future data items can use them. The second type is hashed address data from the first semi-local pattern $lp_{first}$ to the current semi-local pattern $lp_{cur}$ from all data streams, for updating the grid

structure when a whole local pattern is identified. A data structure *lpDataNode* is used to save all the data of the current local pattern and is defined as follows:

```
struct lpDataNode {
    double dataItem[lpMaxLen];
    int hashAddr[lpMaxLen];
  }
```

where *lpMaxLen* denotes the maximal length of the local pattern. In the leaf node of $B^+$-tree, we keep a pointer to point at *lpDataNode*. Considering that all *lpDataNodes* take up only a small volume of main memory, we load them into the main memory.

Next, we will introduce how to update the grid structure. We use a dynamic update policy to maintain the grid structure. At the end of each whole local pattern *lp*, all corresponding feature points $\{(k_{\text{first}}, t_{\text{first}}), (k_{\text{second}}, t_{\text{second}}), \ldots, (k_{\text{last}}, t_{\text{last}})\}$ of *lp* are cleared from the corresponding grid unit by searching the $B^+$-tree to obtain the hashed address list *hashAddr* of *lp*. Our system can monitor two types of correlations, that is, synchronized correlations which occur between two current local patterns $lp_{\text{cur1}}$ and $lp_{\text{cur2}}$, and lagged correlations in which there is a previous local pattern $lp_{\text{pre}}$ and a current local pattern $lp_{\text{cur}}$. Suppose that the local pattern *lp* is hashed to cell *c*, then *lp* will be compared to any local patterns that have been hashed to the neighborhood of *c*.

**Time and space complexity**

Reporting correlative streams is generally computationally expensive, which in the worst case needs $O(N^2)$ time, where $N$ is the size of dataset. In this subsection, we will show that our GSA algorithm requires only linear space and time.

Let us firstly discuss space complexity. Assume that there are $N$ financial data streams, and $m$ is the average length of all local patterns during their life. The total space will be $N\times(m-SLen_{\min})$ double bytes and $N\times(m-SLen_{\min})$ int bytes with a little extra index main memory because each local pattern is mapped into a feature point $(k, t)$, and there are $m-SLen_{\min}$ semi-local patterns during the period of $m$ time ticks. By a similar analysis, the space of $B^+$-tree is $N\times m$ double bytes and $N\times(m-SLen_{\min})$ int bytes, in addition to the extra index space. Because $m<<N$, when $N$ is larger, the actual space is linear and much less than the original size of the dataset.

The most time-consuming part of our algorithm is the Correlative_Search algorithm. Its time complexity is $O(p\times FPNum_{\text{gu}})$, which is linear, where $FPNum_{\text{gu}}$ $(=N\times(m-SLen_{\min})/(Num_k\times Num_t))$ denotes the average number of feature points contained in each grid unit, $Num_k$ and $Num_t$ are the interval numbers on the $k$- and $t$-axis, respectively, and $p$ represents the number of search grid units.

**Parallel implementation**

As noted by Zhu and Shasha (2002), our model can also use a parallel implementation by a straightforward decomposition. Consider a network of $M$ servers to monitor $N_s$ streams. Assuming that these servers have similar computing resources, we complete the work to monitor the streams in two stages. The first stage is online segmenting and pruning of the data streams, mapping streams into a grid structure. The $N_s$ streams are equally divided into $M$ groups. The server $i$ ($i$=1, 2, …, $M$) will read those streams in the $i$th group, segment and prune the $i$th group, index the $i$th group, and map the $i$th group into a grid structure. The second stage is reporting highly correlated stream pairs based on the grid structure. The grid structure is also geometrically and evenly partitioned into $M$ parts. A server $X$ will read in its part, a set of cells $S_X$. Server $X$ will also read a set of cells $\tilde{S}_X$ including cells adjacent to the boundary cells in $S_X$ according to Theorem 3. Each server invokes the procedure of Correlative_Search to report the correlated stream pairs.

## EXPERIMENTAL EVALUATION

**Experimental setup**

To evaluate the effectiveness and efficiency of our proposed approaches, we carried out extensive experiments with real stock data. In experiments, we used more than 1 820 000 data points from 1700 real stocks in China from http://finance.yahoo.com/ (2008-04-16). We conducted experiments on a Pentium III PC with 512 MB memory.

The main parameters were set up as follows. We used the close of day prices as financial data streams. In a real time environment, we used the average quotes as the close price over a fixed time interval.

We set up an $r$-period moving average with parameter $r$=3 or 5. The size of the sliding window was changed from $m$=6 to 99 for the time interval over days, and the threshold $\mu$ of delay time was 6 in the procedure for segmenting and pruning. The shortest length of pattern segment, $SLen_{\min}$, was 5 in the interval over days, and the parameter $\delta$ denoting the maximal relative approximation error was 0.03 on identifying a pattern segment. In fact, *SDD* indicates a summation of relative percent difference between the $k$ component and $t$ component based on weight. Consider that $k$ component exerts a larger impact than the $t$ component. We then set up the parameters $\varepsilon$, $\lambda_1$ and $\lambda_2$ with 0.1, 0.65 and 0.35, in pattern similarity, respectively.

## Effectiveness of MCALP

In this subsection, firstly, we will show that *SDD* distance is an appropriate measurement of local patterns in a financial domain. Then, we evaluate the effectiveness of the GSA algorithm.

Let us first compare *SDD* distance and Euclidean distance for three real segments $L_r$, $L_s$ and $L_t$ formed by the first seven points from streams $R$, $S$ and $T$ in Fig.5, respectively. Table 1 shows the values of their data points, where $t_i$ denotes the time tick. We define $dist(L_1, L_2)$ as the Euclidean distance of two segments $L_1$ and $L_2$. By computation, $dist(L_r, L_s)$, $dist(L_r, L_t)$ and $dist(L_s, L_t)$ are 34.03, 12.76, and 46.29, respectively. This indicates that $L_r$ and $L_t$ are more similar than the other two segments, $L_r$ and $L_s$, $L_s$ and $L_t$. We can draw the same conclusion even if we normalize the values in Table 1. However, $L_r$ and $L_s$ are more similar than the other two segments when using our *SDD* measurement. This is shown clearly in Fig.5. So, *SDD* is more effective than Euclidean distance, considering that the users pay more attention to the trends of local patterns.

**Table 1 Values of three local patterns with seven data points**

| Segment | Values of data point | | | | | | |
|---|---|---|---|---|---|---|---|
| | $t_1$ | $t_2$ | $t_3$ | $t_4$ | $t_5$ | $t_6$ | $t_7$ |
| $L_r$ | 44.44 | 47.57 | 49.04 | 48.40 | 52.80 | 53.71 | 55.01 |
| $L_s$ | 33.56 | 34.97 | 35.07 | 35.13 | 38.49 | 40.37 | 43.75 |
| $L_t$ | 46.12 | 50.82 | 52.81 | 52.91 | 58.29 | 60.78 | 60.90 |

Next, we carried out an experiment using the parameters of the previous subsection, with 1600 stream time series randomly chosen from 1700 instances. Because there were so many streams, we only show partial results in Fig.7, which includes eight representative stocks. It can be seen clearly from Fig.7 that A and B were correlative during many periods, for example, $t(111)$~$t(129)$ of A ($k_a$=1.73) and $t(111)$~$t(128)$ of B ($k_b$=1.62), $t(129)$~$t(149)$ of A ($k_a$=0.80) and $t(128)$~$t(149)$ of $B$ ($k_b$=0.76), etc.

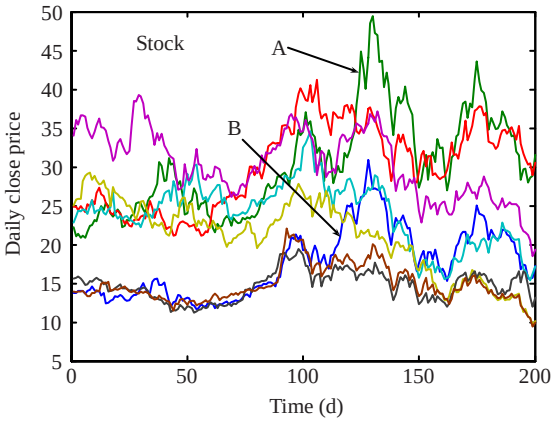


**Fig.7 An example to illustrate two correlative stream time series**

## Efficiency of MCACP

In this section, we first report the efficiency of the Calculating_Err algorithm, and then evaluate the time spent in monitoring the correlative stream pairs when changing the number of streams or the threshold value of $p$. Here, we assume that $\varepsilon$ is 0.1 in all experiments. Note that, methods like StatStream (Zhu and Shasha, 2002) are designed for the same length local patterns and cannot handle generic cases with local patterns of different lengths. Thus, we did make comparisons with StatStream in our experiments.

In the first test, we compared the performance of the algorithm Calculating_Err relative to the corresponding naive algorithm (Naive) which uses three circles to compute coefficients $a$ and $b$, $RErr_{\mathrm{PLA}}$, respectively. We defined the relative CPU cost as the proportion of Naive spending time relative to the average computing time of Calculating_Err for each invoking. It is clear from Fig.8 that Calculating_Err was faster than Naive. In fact, the CPU running time of the Calculating_Err method is almost constant and its time complexity is $O(1)$. However, the CPU running time of Naive gradually increases with the increase of the length of local patterns, and the time

complexity of the Naive method is $O(n)$. Calculating_Err was 32.3 times faster than Naive in CPU time when the local pattern length was 75. Even with the shortest length 5, Calculating_Err was 3.51 times faster than Naive in CPU time.

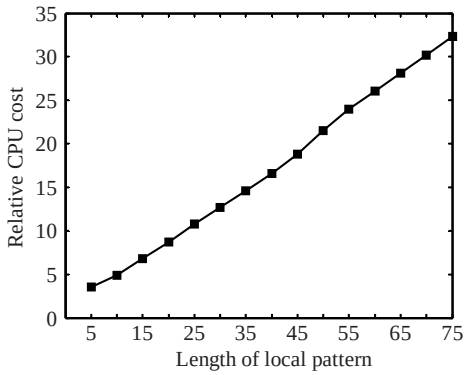


**Fig.8 Efficiency of the Calculating_Err algorithm**

The next test compared the wall clock time for different numbers of streams when monitoring all correlative streams pairs. In the experiment, we used two methods, that is, the Naive method which does not use grid structure and needs a sequence scan to select the most correlative local patterns, and the MCALP method. Using the Naive method, the time taken to compute the correlations among $N_s$ streams $T_0$ is proportional to $N_s^2$. With the MCALP method, the work to monitor correlations has two parts: (1) updating the index takes time $T_1$ which is proportional to $N_s$; (2) detecting correlations based on the grid takes time $T_2$ that is proportional to $N_{Rec}^2$ which denotes the average number of feature points contained in the rectangular searching region *Rec*. Because $T_2$ is the main time of the MCALP method and $N_{Rec}<<N_s$, using the MCALP method is much faster than using the Naive method (Fig.9). At the same time, we also found that the time spent became longer with the increase in the number of streams. Note that the unit of measure of spending time is different between Figs.9a and 9b.

The last test studied the impact of different $p$. In the experiment, we specially observed two situations, namely the number of streams $N_s$=800 (the corresponding curve is denoted as $C_{800}$) and the number of streams $N_s$=1600 (the corresponding curve is denoted as $C_{1600}$). From Fig.10, the time spent of $C_{800}$ and $C_{1600}$ both gradually increased with the increase of $p$. At the same time, we also observed that $C_{1600}$ increased more quickly than $C_{800}$. It is obvious that the responding time was sufficient to adapt to real-time situations. Although the granularity of time was day in the above experiments, in fact, our model MCALP can handle smaller granularity of time, i.e., intervals of 2, 5, 10 min, etc.

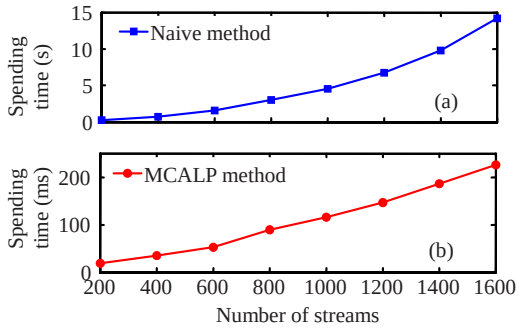


**Fig.9 Comparison of spending time for different numbers of streams with $\varepsilon$=0.1. (a) Naive method; (b) MCALP method**

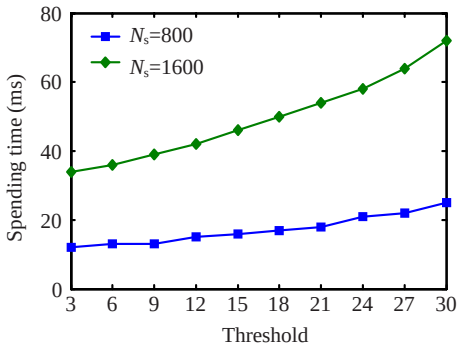


**Fig.10 Comparison of spending time for different $p$ with $\varepsilon$=0.1**

## Impact of parameters

In this subsection, we report mainly the impact of parameters $\varepsilon$ and $\delta$. These two parameters are very important and affect the precision of local pattern approximation. In general, the smaller the two parameters are, the better the precision should be. We define the summation of $Err_{PLA}$ of all segments as $TErr_{PLA}$, where $TErr_{PLA}=\sum_{i=1}^{N} Err_{PLA}(Seg_i)$, and $N$ represents the number of segments. To eliminate the impact of amplitude offset, all streams in the experiments were normalized with zero mean and unit standard deviation.

Fig.11 shows the change of average $TErr_{PLA}$ with $\delta$ from 0.01 to 0.10 on 800 streams with 512 data points. In the beginning, the error quickly increased with the increment of $\delta$ from 0.01 to 0.03, and then it slowly increased in $\delta$=[0.04, 0.10]. By the analysis of previous discussion, we know that $\delta$ controls the degree of oscillation for all data points of $L$ near $y$. So, a bigger $\delta$ brings a bigger error. But, $\mu$ is invariable and it maintains the number of segments with little change, which results in a slow increase of $Err_{PLA}$. At the same time, we can see that $\delta$=0.03 was a reasonable threshold with $\mu$=5.

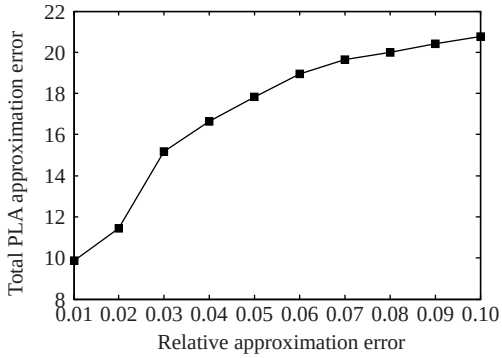


**Fig.11 Total PLA approximation error with respect to $\delta$ with $\mu$=5**

Fig.12 depicts the change of average $TErr_{PLA}$ with respect to $\delta$ on 1600 streams with 512 data points. From Fig.12, we can observe that the cumulative $TErr_{PLA}$ increased with the increase of $\mu$ when $\delta$ was kept invariable. This phenomenon has a similar explanation to that mentioned above in Fig.11. In the experiment, a better $\mu$ was 6 and $\mu$=10 should be excluded because it caused a bigger increment in $TErr_{PLA}$.

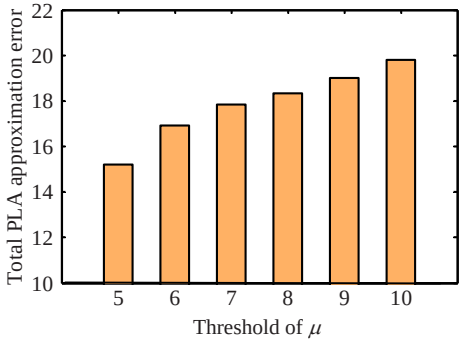


**Fig.12 Total PLA approximation error with respect to $\mu$ with $\delta$=0.03**

As for parameter $\varepsilon$, obviously, $0 \le \varepsilon < 1$, and reflects the degree of relative deviation of two local patterns. So, a bigger $\varepsilon$ corresponds to a larger $\mu$ and a larger $\delta$; otherwise, the smaller $\mu$ and $\delta$. By experiment, we found that $\varepsilon$=[0.05, 0.15] gave better precision.

## CONCLUSION

Monitoring multi-streams and reporting any correlative stream pairs is a significant challenge. Our study solves the problem with a guaranteed response time and high accuracy. We make use of PLA and SDD to metric the similarity of two local patterns measured with their two local features, that is, their slope and duration. At the same time, we map all local patterns to a grid structure so as to find the correlative local pattern pairs. Experiments conducted using real stock data showed that our model MCALP is effective and efficient. This work focused on correlations based on single local similar patterns. One interesting future direction would be to explore the correlations between financial data streams with multiple continuous local similar patterns.

## References

Agrawal, R., Faloutsos, C., Swami, A., 1993. Efficient Similarity Search in Sequence Databases. Proc. Int. Conf. on Foundations of Data Organization and Algorithms, Chicago, Illinois. Springer-Verlag, Germany, p.69-74.

Bentley, J.L., Weide, B.W., Yao, A.C., 1980. Optimal expected-time algorithms for closest point problems. *ACM Trans. Mathem. Software (TOMS)*, **6**(4):563-580. [doi:10.1145/355921.355927]

Berndt, D.J., Clifford, J., 1996. Finding Patterns in Time Series: A Dynamic Programming Approach. Proc. Advances in Knowledge Discovery and Data Mining, AAAI/MIT Press, Menlo Park, CA, USA, p.229-248.

Chen, Q., Chen, L., Lian, X., Liu, Y., Jeffrey, X.Y., 2007. Indexable PLA for Efficient Similarity Search. Proc. VLDB Conf., Vienna, Austria. VLDB Endowment, USA, p.435-446.

Chen, Y.G., Nascimento, M.A., Ooi, B.C., Tung, A.K.H., 2007. Spade: On Shape-based Pattern Detection in Streaming Time Series. Proc. IEEE ICDE, Istanbul, Turkey. IEEE, USA, p.786-795. [doi:10.1109/ICDE.2007.367924]

Guha, S., Gunopulos, D., Koudas, N., 2003. Correlating Synchronous and Asynchronous Data Streams. Proc. ACM SIGKDD, Washington, D.C., USA. ACM, USA, p.529-534. [doi:10.1145/956750.956814]

Keogh, E., 2002. Exact Indexing of Dynamic Time Warping. Proc. VLDB Conf., Hong Kong, China. Morgan Kaufmann, USA, p.406-417.

Korn, F., Jagadish, H.V., Faloutsos, C., 1997. Efficiently Supporting Ad Hoc Queries in Large Datasets of Time Sequences. Proc. SIGMOD Conf., Birmingham, UK, p.289-300. [doi:10.1145/253260.253332]

Lian, X., Chen, L., Yu, J.X., Wang, G.R., Yu, G., 2007. Similarity Match over High Speed Time Series Streams. Proc. IEEE ICDE Conf., Istanbul, Turkey. IEEE, USA, p.1086-1095. [doi:10.1109/ICDE.2007.368967]

Papadimitriou, S., Yu, P.S., 2006. Optimal Multi-scale Patterns in Time Series Streams. Proc. ACM SIGMOD, Chicago, Illinois. ACM, USA, p.647-658. [doi:10.1145/1142473.1142545]

Papadimitriou, S., Sun, J., Faloutsos, C., 2005. Streaming Pattern Discovery in Multiple Time-series. Proc. VLDB Conf., Trondheim, Norway. ACM, USA, p.697-708.

Papadimitriou, S., Sun, J., Yu, P.S., 2006. Local Correlation Tracking in Time Series. Proc. IEEE ICDM, Hong Kong, China. IEEE, USA, p.456-465. [doi:10.1109/ICDM.2006.99]

Sakurai, Y., Papadimitriou, S., Faloutsos, C., 2005. Braid: Stream Mining through Group Lag Correlations. Proc. ACM SIGMOD, Baltimore, Maryland. ACM, USA, p.599-610. [doi:10.1145/1066157.1066226]

Sakurai, Y., Faloutsos, C., Yamamuro, M., 2007. Stream Monitoring under the Time Warping Distance. Proc. IEEE ICDE, Istanbul, Turkey. IEEE, USA, p.1046-1055. [doi:10.1109/ICDE.2007.368963]

Wu, H., Salzberg, B., Zhang, D., 2004. Online Event-driven Subsequence Matching over Financial Data Streams. Proc. ACM SIGMOD, Paris, France. ACM, USA, p.23-34. [doi:10.1145/1007568.1007574]

Zhang, T.C., Yue, D.J., Gu, Y., Yu, G., 2007. Boolean Representation Based Data-adaptive Correlation Analysis over Time Series Streams. Proc. ACM CIKM Conf., Lisboa, Portugal. ACM, USA, p.203-212. [doi:10.1145/1321440.1321471]

Zhu, Y., Shasha, D., 2002. Statstream: Statistical Monitoring of Thousands of Data Streams in Real Time. Proc. VLDB Conf., Hong Kong, China. Morgan Kaufmann, USA, p.358-369.