Online Event-driven Subsequence Matching over Financial Data Streams

Huanmei Wu* Betty Salzberg† Donghui Zhang
College of Computer and Information Science
Northeastern University
Boston, MA 02215
{maggiewu, salzberg, donghui}@ccs.neu.edu

ABSTRACT

Subsequence similarity matching in time series databases is an important research area for many applications. This paper presents a new approximate approach for automatic online subsequence similarity matching over massive data streams. With a simultaneous online segmentation and pruning algorithm over the incoming stream, the resulting piecewise linear representation of the data stream features high sensitivity and accuracy. The similarity definition is based on a permutation followed by a metric distance function, which provides the similarity search with flexibility, sensitivity and scalability. Also, the metric-based indexing methods can be applied for speed-up. To reduce the system burden, the event-driven similarity search is performed only when there is a potential event. The query sequence is the most recent subsequence of piecewise data representation of the incoming stream which is automatically generated by the system. The retrieved results can be analyzed in different ways according to the requirements of specific applications. This paper discusses an application for future data movement prediction based on statistical information. Experiments on real stock data are performed. The correctness of trend predictions is used to evaluate the performance of subsequence similarity matching.

1. INTRODUCTION

Many applications generate data streams and there is an increasing need to maintain statistical information online. Stream databases are distinguished from conventional databases in several aspects. Raw data is too large to be stored in a traditional database for efficient data management. Querying on the stream database is difficult in set-oriented data management systems. Because the data is changing constantly, a single-pass search over the stream is mandatory since it is infeasible or impossible to rewind the stream. The answers of the query usually are approximate and partial answers. Examples of stream databases can be found in stock market quotes, sensor data, telecommunication systems, and network management.

Subsequence matching in time series databases tries to find subsequences from the large data sequences in the database that are similar to a given query sequence. It is important in data mining and is used for pattern matching, future movement prediction, new pattern identification, rule discovery and computer aided diagnosis. Stream data are naturally ordered in time. Some streams are ordered in a fixed time interval and can be treated as stream time series directly. Some streams come in irregularly and special procedures are needed in order to apply time series techniques. For example, there are thousands of stock transactions every second, which may be carried out at any time and there are different numbers of transactions at different times.

Existing techniques on time series subsequence matching mainly focus on discovering the similarity between an online querying subsequence and a traditional database. The queried data are static and are accessed using an index. Research in time series *data streams* is in its preliminary stage. Only some basic statistical measures such as moving averages and standard derivation have been addressed. There is recent research [17, 18, 29, 38] on similarity matching over data streams. The papers [38] treat pair-wise correlated statistics in an online fashion, focusing on similarity for whole data streams, not on subsequence similarity. The papers [17, 18] treat similarity-based continuous pattern queries with prediction, which can be extended to answer nearest k neighbors on a stream time series. Last, [29] uses an index structure for K-NN search on data streams. In contrast, we are investigating application-guided subsequence matching over online financial data streams and online query subsequences. Our database is a dynamic stream database which stores recent financial data. It will be automatically updated as new stream data comes in. So our database includes the most recent historical data. The query subsequence is automatically generated based on the current state of the data stream. And our new similarity measure satisfies the special requirements of financial data analysis.

Subsequence similarity of financial data streams has its unique properties. First, according to Elliott Wave Theory [15], the movement of the stock market can be predicted by observing and identifying a repetitive pattern of waves. Based on this wave theory, the online piecewise linear representation of the stream data should be in an up-down-up-down repetitive pattern (the *zigzag* shape). Keogh et al. [26] summarized four well-known algorithms for time series segmentation. None of them has addressed the zigzag requirement. The result of the compression algorithm in [14] is in $zigzag$ shape, but the algorithm does not satisfy the real time re-

*This work is part of CenSSIS, the Center for Subsurface Sensing and Imaging Systems, under the Engineering Research Centers Program of the National Science Foundation (Award # EEC-9986821).

†This work is partially supported by NSF grant IIS-0073063.

SIGMOD 2004 June 13-18, 2004, Paris, France.

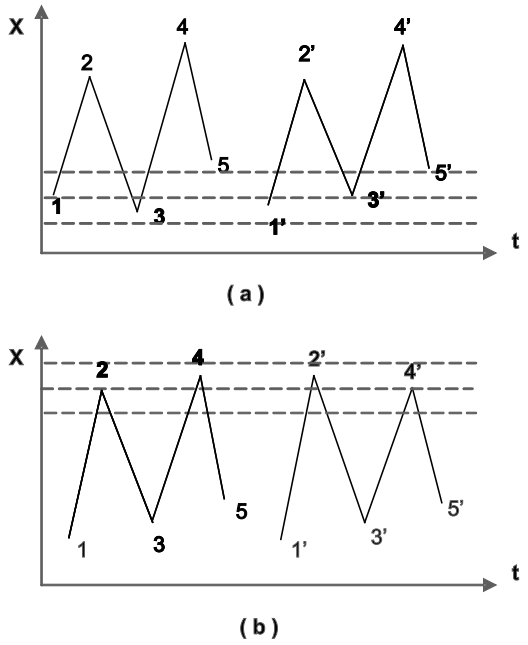

Figure 1: Subsequence similarity with different relative positions: (a) Two subsequences differ in the relative positions of the lower points; (b) Two subsequences differ in the relative positions of the upper points.

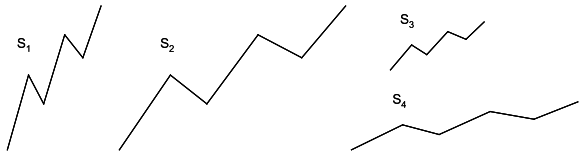

Figure 2: Subsequence similarity with time scaling and amplitude rescaling.

quirements for online stock data analysis because it has a longer delay to identify an extreme point when the extrema ratio is large. It is necessary to have a new online segmentation algorithm that can quickly and accurately identify potentially important points. Second, the relative position of the upper and lower end points plays an important role in subsequence similarity. Figure 1 shows two examples of two pairs of subsequences which would be considered similar using existing subsequence similarity measures. But technical analysis of financial data is also concerned with the relative position of the upper end points as well as the relative position of the lower end points. The two pairs of subsequences in Figure 1 would not be considered similar by financial data analysts. Third, subsequence similarity should be flexible with regard to time shifting and scaling, price shifting and amplitude rescaling ($Amplitude$ is the value difference of two adjacent end points). Financial data technical analysis assumes that the amplitude difference is more important than the time difference. For example, in Figure 2, all four subsequences are derived from a sequence with time scaling, or amplitude rescaling, or both. The pairs S_1 and S_2, S_3 and S_4 have the same amplitude changes, but different time changes, and the pairs S_1 and S_3, S_2 and S_4 have the same time changes but different amplitude changes. According to financial analysts, S_1 and S_2, S_3 and S_4 are more similar while S_1 and S_3, S_2 and S_4 are less similar. A new subsequence similarity definition that allows amplitude rescaling (but with limitations) is required.

Our new online event-driven subsequence similarity matching takes into account and gracefully handles the special properties of financial data analysis. We make the following main contributions:

1. We propose a 3-tier online simultaneous segmentation and pruning algorithm. It takes a raw financial data stream as input and produces a stream of piecewise linear representation end points. The end points are in an upper-lower-upper-lower repetitive pattern (the *zigzag* shape). This tiered segmentation and pruning algorithm provides the piecewise linear representation with high sensitivity and accuracy. The algorithm runs in linear time and with constant memory.

2. We explore an alternative similarity measure for subsequence matching, where a metric distance function is defined based on a permutation of the subsequence. The permutation ensures two subsequences have the same relative positions. The distance function controls the extent of amplitude rescaling. The new definition provides subsequence similarity search with sensitivity, flexibility and scalability. Any existing metric-based indexing technology can be employed for search speed-up.

3. We perform event-driven subsequence similarity matching over an up-to-date database using the end points of the piecewise linear representation. The query will be carried out only when there is a new end point. The automatically generated query subsequence is the most recent subsequence of the end points, which reflects the most recent information of the raw financial data stream. A new mechanism that can turn on or off the search engine is enabled.

4. We apply a new definition of *trend* for financial data streams using the results of subsequence similarity search to predict future data movement. Our definition of trend does not place any restrictions on the characteristics of the stock streams on which it is applied. The market can be a bull market, a bear market or a no-trend market. Our event-driven subsequence similarity search is more accurate in seizing critical points for a trend period than algorithms which search at all time instances. In addition, our approach is 30 times faster than searching at all time instances.

The rest of the paper is organized as follows. Section 2 briefly discusses related work on subsequence matching and data stream processing. Section 3 describes our strategy for data processing over incoming streams. The subsequence similarity matching of the resulting piecewise linear representation is explained in detail in Section 4. One application of our similarity search for trend prediction is discussed in Section 5. Section 6 presents our experiment results and Section 7 concludes this paper and provides some future research directions.

2. RELATED WORK

Similarity search in time series is useful for many data mining applications. Agrawal et al. [2] has first introduced whole sequence similarity matching. Faloutsos et al. [13] generalized it to subsequence similarity matching. The basic idea is to transform the sequence into the frequency domain using a Discrete Fourier Transformation (DFT). Then the first few features are extracted and the

Euclidean distance is used as the similarity distance function. Multidimensional indexing methods such as the R*-tree [5] can be applied for fast search. In subsequence matching, the R*-tree stores only Minimum Bounding Rectangles (MBR). New research based on subsequence search has grown in several aspects. New methods in constructing MBRs reduce false negatives [30]. Keogh et al. proposed Piecewise Aggregate Approximation (PAA) to reduce the dimensionality and to support fast sequence matching using R-trees. Other feature extraction functions, such as the Discrete Wavelet Transformation (DWT) [8, 20], Adaptive Piecewise Constant Approximation (APAC) [24], and Single Value Decomposition (SVD) [28] have been proposed to reduce the dimensionality of time series data. New distance functions such as Dynamic Time Warping [32, 35] and Longest Common Subsequences [11] have been explored to overcome the brittleness of the Euclidean distance measure or its variations [2, 13, 33].

Data streams have attracted more research interest recently [1, 3, 4, 12, 17, 18, 19, 21, 22, 29, 31, 38]. Babu et al. [3] showed how to define and evaluate continuous queries over data streams. Some basic statistics over data streams have been studied. Datar et al. [12] studied single stream statistics using sliding windows. Gehrke et al. [19] studied statistics for correlated aggregates over multiple data streams using histograms. Gao et al. [17, 18] introduce a new strategy of continuous queries with prediction on a stream time series. Liu et al. [29] treat KNN search over data streams using index structures. Zhu et al. [38] proposed a new method for statistics over thousands of data streams. Their research focuses on pair-wise correlation using a grid-based data structure. Data stream clustering algorithms include STREAM [22, 31], Fractal Clustering [4], and CluStream [1]. STREAM aims to provide guaranteed performance of data stream clustering and CluStream is developed for clustering large evolving data streams.

Stock data analysis has attracted researchers for years. Autoregressive and moving average are long used techniques [23] for stock market prediction. In the field of data mining, intensive research has been done on the application of neural networks to stock market prediction [27]. Stock trends can be also predicted based on the association of trends with news articles [16]. Fink and Pratt applied subsequence similarity matching in compressed time series by identifying major extrema [14]. The previous work does not concern the real time requirements of online financial data analysis. For instance, the t-test based piecewise segmentation in [16] works on static historical time series in the training phase. The compression algorithm in [14] runs in an online fashion. But it will take longer delay time to identify the previous extremum, which is not practical for stock trading where early detection of a potential end point is critical.

Our work differs from previous research in several aspects. The problem addressed here is online subsequence search over financial data steams and we have addressed the special requirements of financial data technical analysis. Our distance measure for subsequence similarity is a metric distance function based on a permutation. The subsequence matching process is triggered by new online events. Our database maintains up-to-date information with newly arrived data, not previously obtained data.

3. ONLINE DATA STREAM PROCESSING

Translating massive data streams into manageable data for the database, which can be queried and indexed upon is an important step for data stream subsequence similarity matching. This section discusses the data preprocessing steps before similarity search which result in piecewise linear representation of incoming streams. The process of data stream aggregation, segmentation and pruning is explained in more detail below.

3.1 Aggregation and Smoothing

Piecewise linear representation of the data streams requires the data streams to have one fixed value for each time interval. The incoming data streams may arrive at any time. Aggregation over raw data streams is both necessary and important for practical applications. A stream may acquire different aggregate values for different purposes. For example, in stock market analysis, the open, high (MAX), low (MIN), close, and volume (SUM) values of one quote over a time interval (minutes, hours, days, months or years) are very important information.

Aggregation makes sure there is a unique value for each time instance over a fixed time interval. If we draw the data movement with time, we can see a lot of shorter-time random oscillation over a longer-term trend. We need to filter out the noise before further data processing. We use the standard moving average which is widely used in the financial market [6] to smooth the data:

$$MA_p(i) = \frac{1}{p} \sum_{j=i-p+1}^{i} X(j)$$

where *X(i)* is the value for i = 1, 2, ..., n and n is the number of periods. $MA_p(i)$ calculates the p-interval moving average time series which assigns equal weight to every point in the averaging interval. By smoothing through the moving average, shorter-term noise will be filtered out while a clean trend signal is generated.

3.2 Piecewise linear representation

Piecewise Linear Representation uses line segments to approximate a time series [14, 26]. Our approach is new because we adopt a tiered online segmentation and pruning strategy. We do not segment over the price stream directly, instead we segment over one financial indicator, ***Bollinger Band Percent (%b)*** [6], to be the first base input for line segmentation. Then we prune over the end points of the %b line segments based on some criteria of %b. The final line segments over the raw data stream are obtained by pruning on the previous line segments with criteria based on the raw price stream. We will explain in detail why we choose %b to do the segmentation and how the tiered structure provides high sensitivity and accuracy in the online segmentation.

3.2.1 %b indicator

Bollinger Bands [6] are widely used financial indicators which provide relative definitions of high and low values for time series. The bands are curves drawn above and below a moving average by a measure of standard derivation. An example of time series and Bollinger bands is shown in Figure 3b. The three curves are defined as follows:

middle_band = p-period moving average
upper_band = *middle_band* + 2 · p-period standard deviation
lower_band = *middle_band* - 2 · p-period standard deviation

%b, shown in Figure 3c, is another popular indicator derived from Bollinger Bands. %b tells us the current state within the bands. The formula for %b is the following:

$$\%b = \frac{close\ price - lower\_band}{upper\_band - lower\_band}$$

%b is chosen to be the first base for linear segmentation because of the following. First, %b has a smoothed moving trend similar to the price movement. If the price moves in an up trend, %b is also in an up trend. And if the price is in a down trend, %b is also in a down

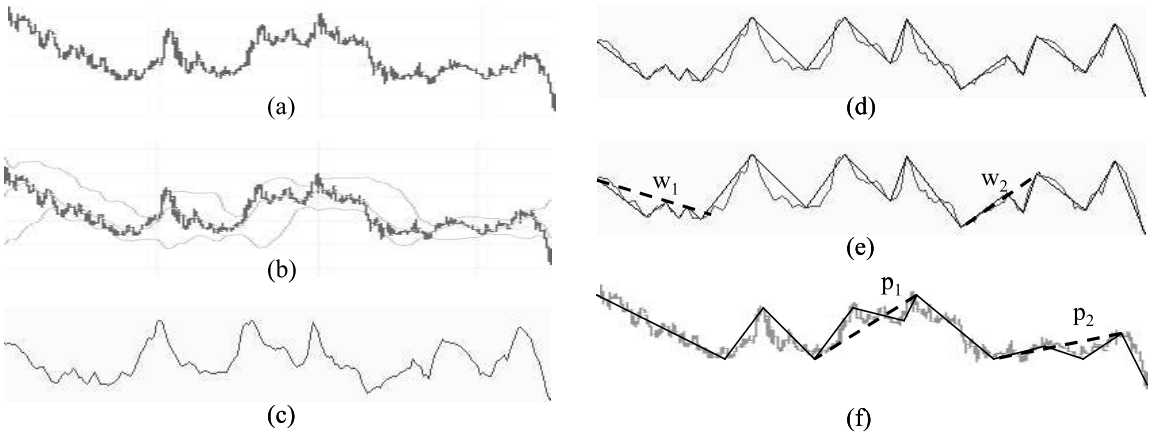

Figure 3: Piecewise linear representation (PLR). (a) Raw financial price stream data; (b) Raw stream data with Bollinger Bands; (c) The corresponding stream of %b values; (d) PLR of %b without pruning; (e) PLR of %b with pruning only on %b during segmentation; (f) PLR of raw stream data with pruning on %b and raw data during segmentation.

trend. The upper and lower end points of %b correspond to the upper and lower end points of the raw price data. Second, %b is a normalized value of the real price. Most %b values are between -1 and 2 no matter what real price values are. So we can set a uniform segmentation threshold for %b which we could not do over the real price. For example, if the average price of a stock is $1.00, a change of $0.20 may be considered as a big movement. But to a stock with an average price about $100.00, $0.20 difference can only be considered as noise. Third, %b is very sensitive to the price change. It will manifest the price change accurately without any delay. So segmentation over %b is more suitable than segmentation directly over the raw price data stream.

3.2.2 Segmentation

Segmentation is based on the %b values. For each time instance, there is a corresponding %b value. Segmentation over %b finds optimized upper and lower end points of the piecewise linear representation for %b. Figure 3d shows the segmentation results of Figure 3c. Our segmentation algorithm is different from others not only because of a different definition of upper and lower end points but also the resulting end points of our segmentation are in the zigzag shape which is not the case in other algorithms.

Our segmentation algorithm uses a *sliding window* with varying size. The sliding window can only contain at most *m* points, beginning after the last identified end point and ending right before the current point, as shown in Figure 4. If there are more than *m* points between the last end point and the current point, only the last *m* points are contained in the sliding window. The segmentation tries to find a possible upper or lower point only in the current sliding window. An upper point is defined as follows (the definition of a lower point is symmetric and thus is omitted here):

Suppose the current point is $\mathrm{P}_j(X_j, t_j)$. The upper point $P_i(X_i, t_i)$ is a point in the current sliding window that satisfies:

1. X_i = *max(X values of current sliding window)*;
2. $X_i > X_j + \delta$ *(where* δ *is the given error threshold)* ;
3. $P_i(X_i, t_i)$ *is the last one satisfying the above two conditions.*

Figure 4 shows an example of an upper point. Here, $P_j = P_{13}$ is the current point. The previous identified end point is P_3, so the

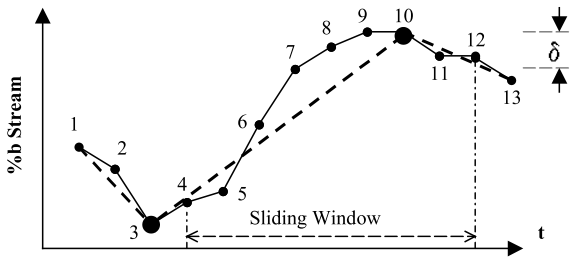

Figure 4: A sliding window which finds an upper point. Suppose that *m* = 10, δ = 1.0, P_3 is the last identified end point, P_{13} is the current point. The actual sliding window size *m* is 8. P_{10} is a new upper point.

sliding window currently contains *m* = 9 points starting from P_4. Both P_9 and P_{10} have the maximum value, but only P_{10} is found as a upper point because it is the last one with the maximal value in the sliding window. Another thing needed to be mentioned here is the *delay time*, which is the time difference between the actually time of an end point and the time when it is identifies as an end point. Although the upper point is at t_{10}, it is only identified at t_{13}. The delay time for identifying t_{10} is $t_{13} - t_{10}$. The threshold δ plays an important role in the delay and the number of line segments. A smaller δ will reduce the delay time but result in a larger number of short line segments, some of which may still be noise. A larger δ will decrease the number of line segments but with longer delay. If δ is too large, some useful information will be filtered out. There is a tradeoff between the delay time and more accurate piecewise linear representation. We propose an optimized algorithm for simultaneous online segmentation and pruning. The new algorithm will reduce the delay time yet will give more accurate piecewise linear representation.

3.2.3 Pruning

Before going into detail for our online segmentation and pruning algorithm, we first introduce the rationale and approach for pruning. To the best of our knowledge, no other published algorithm does pruning. Pruning is the process to remove noise-like line seg-

ments along with the segmentation process. Segmentation tries to find potential end points using a smaller threshold δ_s, so new end points can be identified with shorter delay time. Pruning is smoothing over recently identified end points. Noise introduced by small δ_s will be filtered out by the pruning process and more accurate line segments are generated. This segmentation and pruning mechanism helps to quickly identify a new end point yet with accurate piecewise linear representation. The shorter delay time is very important for real time applications such as stock data analysis. The end points are generally critical points for stock transactions. The earlier such points are identified, the better the chances are for profitable stock trading.

The pruning process itself is a two-step process. First, %b is used in the filter step. But when mapping %b pruned end points onto raw data, the piecewise linear representation on raw data may still have some noise. It is possible that the %b data values change considerably while the raw data values change very little. So we need a refinement step. Pruning on the raw data stream not only removes the oscillations of a trend, but also enforces the zigzag shape. Under rare conditions, the end points mapped directly from %b end points may not be in the zigzag shape. Figure 3e and 3f shows the pruning results on both %b and on the raw stream data. The thick dotted line segments are new line segments generated by the pruning process. The corresponding filled line segments covered by dotted lines are removed.

The actual technique for pruning is following. If the absolute %b or raw data values of two adjacent end points (called the *amplitude*) differs by less than a certain value, that line segment should be removed. Note there may be different values for pruning on %b from those used in pruning the raw data stream. The tricky part is we must keep the zigzag shape of the end points, so we must remove two adjacent end points at the same time. This creates a problem as shown in Figure 5. Here, the line segment $\overrightarrow{cd}$ is under the pruning threshold, so pruning is needed. There are several ways to remove $\overrightarrow{cd}$.

In online segmentation and pruning, at each new end point, we check the previous line segment for pruning. For example, in Figure 5, at the time when end point *e* is identified, line segment $\overrightarrow{cd}$ is tested for pruning. First we check the need for pruning on %b. If needed, pruning is carried out. Then the system waits for next stream data to come in and no pruning on raw data is done. If no pruning on %b is needed, the same line segment is checked for pruning on raw data. So there is at most one pruning at each end point. The pruning algorithm is the same for pruning on both %b and raw data.

We compare the last end point with the third last end point to see which one gives a better piecewise linear representation. If the two points are upper points, the one with the larger value will be kept. Otherwise, if both lower points, the one with the smaller value will be kept. Figure 5 gives an example for pruning with the last end point as a lower point. End points *e* and *c* are compared. If e has smaller value, end points *c* and *d* will be removed from the end points stream, and a new line segment $\overrightarrow{be}$ is generated (Figure 5b). If *c* has smaller value, end points *d* and *e* will be removed. Line segment $\overrightarrow{bc}$ will remain (Figure 5b).

3.3 Online segmentation and pruning

Our online subsequence similarity matching is based on the similarity between two subsequences of end points. A single-pass for online segmentation and pruning is mandatory. To reduce the time delay in identifying end points and improve piecewise linear representation, we use different thresholds for segmentation and pruning: a smaller threshold δ_s for segmentation over %b, a larger threshold δ_p^b for pruning over %b, and a separate δ_p^d for pruning over raw stream data. A smaller threshold for segmentation will ensure the sensitivity and reduce delay. A larger pruning threshold will filter out noise. Our experiments show that $\delta_s \simeq 0.02$, $\delta_p^b \simeq 1.5$ are suitable for most stock prices. The value of δ_p^d is flexible and varies according to different users. Experiments have shown that10% to 20% of the price change over the trading period has reasonable results. For instance, for intra-day trading, if a quote's average daily price change is \$1.50, δ_p^d between \$0.15 to \$0.30 all can achieve pretty good results.

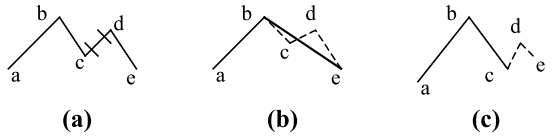

Figure 5: Two possible ways for pruning line segment cd.

The online segmentation and pruning are running simultaneously. Whenever an upper/lower point is identified by the segmentation process, the previous line segment is checked for pruning as mentioned in Section 3.2.3. To better explain the online segmentation and pruning algorithm, an animation of the process is illustrated in Figure 6. Suppose now we are after the time when t_0 is identified as an upper point (Figure 6a). As time goes on, P(t_1) is identified to be a potential lower point (Figure 6b). A temporary line segment $\overrightarrow{t_0t_1}$ is generated. The line segment immediately before t_0 is checked for pruning. Since the amplitude of the line segment on %b is larger than δ_p^b, and that of raw stream is larger than δ_p^d, neither pruning on %b nor on raw stream is needed. Similarly, end points P(t_2) and P(t_3) are identified as potential end points without pruning (Figure 6c).

A pruning is encountered when P(t_4) is identified as a potential upper point (Figure 6d). The line segment $\overrightarrow{t_2t_3}$ is checked for pruning. Since the amplitude of $\overrightarrow{t_2t_3}$ is less than δ_p^b, a pruning process is required. The last end point P(t_4) and the third last end point P(t_2) are compared for a better Piecewise Linear Representation on %b. Since both points are upper points, the one with the larger value will be kept. Here, the value at t_4 is larger, end points P(t_2) and P(t_3) are removed, and line segments $\overrightarrow{t_1t_2}$ $\overrightarrow{t_2t_3}$ $\overrightarrow{t_3t_4}$ on both %b and the raw stream are removed. A new line segment $\overrightarrow{t_1t_4}$ is created.

Continuing the segmentation and pruning process to time t_5, a new potential lower end point is identified without pruning. Another pruning process is encountered at time t_6 when a new potential upper point is identified (Figure 6e). The amplitude for the previous line segment $\overrightarrow{t_4t_5}$ on %b is larger than δ_p^b, so no pruning on %b is required. But the amplitude of $\overrightarrow{t_4t_5}$ is less than threshold δ_p^d, pruning on raw data stream is required. By comparing the raw price values at t_4 and t_6, The end point at t_6 is kept while end points t_4 and t_5 are removed.

As a summary for Figure 6, for time t_0 to t_6, two end points on the raw data stream are identified, i.e., the end points at t_1 and t_6. All other potential end points are removed by pruning on either %b or the raw stream. The end points of %b are only a temporary tool and will not be kept in the final piecewise linear representation of the raw data stream. Also we have the following observations:

- If an end point has one following line segment whose amplitude is larger than the pruning threshold on both %b and raw

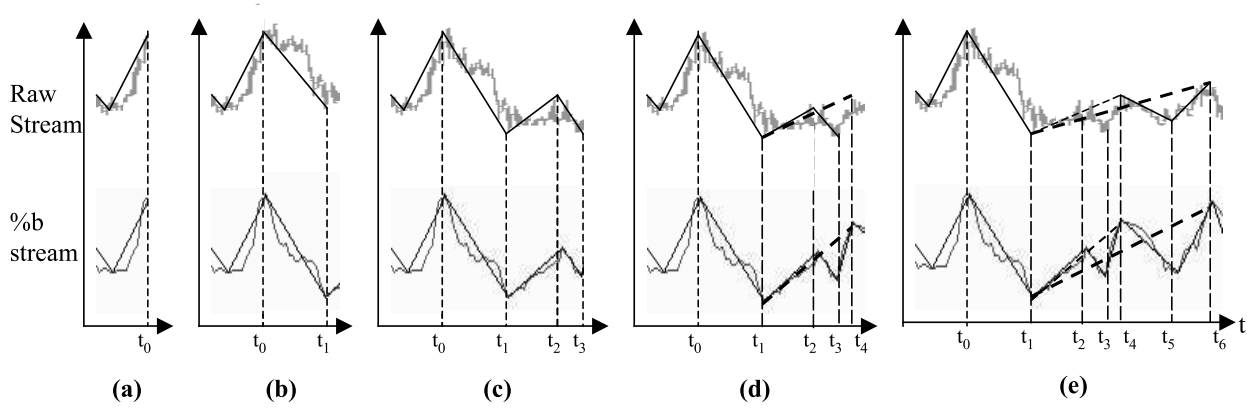

Figure 6: Illustration of the online segmentation and pruning.

data stream, that end point is fixed, i.e., it can not be removed by further segmentation and pruning process.

- After pruning, if an end point has two following line segments, that end point is fixed.
- When a new potential end point is identified but no pruning is needed, a new fixed end point will be produced.
- The online segmentation and pruning algorithm will only affect the last three end points.

Combining the above observations, it is easy to understand that the online segmentation and pruning can be done with varying-length sliding windows, starting from the last fixed end point to the current data of the stream. And there are at most three end points that need to be kept for the following segmentation and pruning procedure. All the fixed end points are updated into the database in real time, so the database has up-to-date information.

3.4 Dynamic Adjustment

Occasionally a stock quote will have a dramatic change in price caused by a stock split or stock merge. Upon stock merge (or split), the current stock price will have a sharp increase (or decrease). A dynamic adjustment is needed to correct the historical data, which is adjusted with the same ratio for the change. In the case of a merge, we only need to increase the historical data values according to the merge ratio. But in the case of stock split, not only we need to decrease the historical data values, but also we need to prune on the historical PLR, since some line segments are under the threshold δ_p^d. Thus a recursive pruning on the historical data is carried out.

Another optimization is to approximate the stream at different granularities by constructing hierarchical PLR end points. The database stores the base end points, which is obtained by PLR on the raw streams with base δ_s, δ_p^b and δ_p^d. It can be used for similarity matching of PLR query subsequence over 1 minute raw data and with the same thresholds. When a query subsequence is with other time granularities (such as 20 minutes) or with different thresholds (larger than the base thresholds), a dynamic process to construct the PLR with the same conditions as the query subsequence is performed on the historical base PLR end points.

4. SUBSEQUENCE SIMILARITY MATCHING

Our online event-driven subsequence similarity matching over data streams is based on the piecewise linear representation of the stream. In this section, we first provide details about our new definition of subsequence similarity. Then we introduce event-driven online subsequence similarity matching.

4.1 Subsequence similarity

The subsequences in our application are subsequences of end points. The subsequence similarity matching in our application finds the subsequences of end points that are similar to the query subsequence. For simplicity, the retrieved subsequence and the query sequence have the same number of end points.

There have been many research efforts for efficient similarity search based on Euclidean distance or its variations [8, 13, 25, 30, 33], DTW distance [26, 32, 35], or LCS distance [11]. However, they do not address the special requirements of financial data anylysis. For example, the distance functions do not concern the relative position of corresponding end points. We hereby propose a new subsequence similarity definition which is more appropriate for financial data analysis.

Our similarity distance function is based on the relative positions (the permutations) of the upper and lower end points in the subsequence. The permutation of a sequence S with *n* elements is a permutation of 1, 2, ..., *n*. It is calculated through the following steps. Consider a stream of end points:

$$S = \{(X_1, t_1), (X_2, t_2), ..., (X_n, t_n)\}$$

First, we divide the end points into two subsets by putting all the upper points into one subset and all lower end points into another. In each subset, the end points are still in the order of time. Without loss of generality, suppose that (X_1, t_1) is an upper point and *n* is even, we will get a new sequence of the n points as

$$S' = \{\ [\ (X_1, t_1),\ (X_3, t_3), ...,\ (X_{n-1}, t_{n-1})\ ],$$

$$[\ (X_2, t_2),\ (X_4, t_4), ...,\ (X_n, t_n)\ ]\ \}$$

Next we sort the X values of each subset. We will get another sequence

$$S'' = \{[(X_{i_1}, X_{i_3}, ..., X_{i_{n-1}}], [(X_{i_2}, X_{i_4}, ..., X_{i_n}]\}$$

where $X_{i_1} \le X_{i_3} \le ... \le X_{i_{n-1}}$, $X_{i_2} \le X_{i_4} \le ... \le X_{i_n}$, $i_1, i_3, ..., i_{n-1}$ is a permutation of 1, 3, ... , *n-1* and $i_2, i_4, ..., i_n$ is a permutation of 2, 4, ... , *n*.

$\{i_1, i_3, ..., i_{n-1}, i_2, i_4, ..., i_n\}$ is called the **permutation** of S. It represents the relative positions of the upper end points and the lower end points. With the permutation of a subsequence, we can define subsequence similarity as following:

DEFINITION 1. *Given two subsequences S and S':*

$$S = \{(X_1, t_1), (X_2, t_2), ..., (X_n, t_n)\}$$

$$S' = \{(X'_1, t'_1), (X'_2, t'_2), ..., (X'_n, t'_n)\}$$

S and S' are similar if they satisfy the following two conditions:

- *S and S' have the same* **permutation**.
- $d(S, S') < \gamma$ *where*

$$d(S, S') = \frac{1}{n-1}(\alpha \cdot \sum_{i=1}^{n-1} ||X_{i+1} - X_i| - |X'_{i+1} - X'_i|| + \beta \cdot \sum_{i=1}^{n-1} |(t_{i+1} - t_i) - (t'_{i+1} - t'_i)|)$$

and α, β and $\gamma \geq 0$ and are user-defined parameters.

The γ value is dependent on the raw data pruning threshold δ_p^d, and special applications. Our experiments show that the optimal value of γ is around $\frac{1}{3} \cdot \delta_p^d$, when $\alpha = 1$ and $\beta = 0$.

In all of our experiments on financial data we use $\alpha = 1$ and $\beta = 0$. We have made our definition of similarity more general because its metric properties can be proved in the more general case and therefore it may prove useful for non-financial data as well.

The subsequence similarity definition seems brittle in the special case of Figure 7. The values at P_2 and P_4 only differ a tiny bit in the two sequences, but the permutations of the two sequences are different. They will not be considered similar using our similarity definition. We can still handle the special case by changing the search algorithm still using the similarity definition. In our search algorithm, the permutations of the query subsequence and the retrieved subsequences are compared first, if the same permutation, the distances are calculated. If a query subsequence has any pairs of upper points (or lower points) with distance under a certain predefined threshold, we consider the query subsequence to have two permutations. Subsequences of the two possible permutations are both searched. In the worst case, the distances between the query subsequence and all possible subsequences will be computed. Since after piecewise linear representation to reduce the dimensions, the subsequence lengths of the PLR end points are below 10, the possible permutations are limited and the special cases are uncommon, so the query performance is still reasonable.

The two parts of the similarity definition are both necessary and complementary which can be illustrated in Figure 2. The permutation is concerned only with relative positions of the end points and not with the differences of actual prices. The permutation alone provides our similarity search with the flexibility of time scaling and amplitude rescaling. The amplitude distance function is more sensitive to the change of amplitudes. The two parts together give our similarity search with flexibility, sensitivity and scalability.

Next we want to show that our distance function is a metric function and we can use metric distance indexing methods for faster search. First we introduce the following lemma.

LEMMA 1. *if a, b, c ≥ 0, $|a - b| \leq |a - c| + |c - b|$.*

LEMMA 2. *if a, b, c, X_1, X_2 Y_1, $Y_2 \geq 0$, $X_1 \leq X_2$ and $Y_1 \leq Y_2$, then $a(bX_1 + cY_1) \leq a(bX_2 + cY_2)$.*

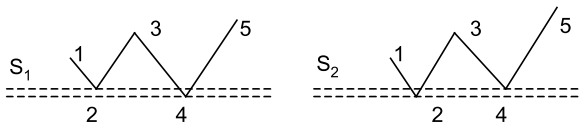

Figure 7: Special cases that are brittle under our similarity definition yet can be handled using our query engine.

Lemma 1 can be proved easily by listing all the possible combinations of a, b and c. Lemma 2 can be proved by the properties of inequality.

THEOREM 1. *For sequences S, S' (with the same length), the distance d(S, S') is metric.*

PROOF. To prove $d(S, S')$ is metric, we need to prove it is symmetric and reflexive, and it satisfies the triangle inequality. Obviously $d(S, S') = d(S', S) \geq 0$ and $d(S, S) = 0$, so $d(S, S')$ is symmetric and reflexive. Next, we need to prove that $d(S, S')$ satisfies the triangle inequality, i.e., $d(S, S') \leq d(S, S'') + d(S'', S')$.

Using the definition of the distance function d as a sum of an amplitude component and a time component, if we prove the triangle equality for both components, it will certainly be true for d by Lemma 2. We thus show:

$$\sum_{i=1}^{n-1}(|\Delta X_i - \Delta X'_i|) \leq \sum_{i=1}^{n-1}(|\Delta X_i - \Delta X''_i| + |\Delta X''_i - \Delta X'_i|)$$

and

$$\sum_{i=1}^{n-1}(|\Delta t_i - \Delta t'_i|) \leq \sum_{i=1}^{n-1}(|\Delta t_i - \Delta t''_i| + |\Delta t''_i - \Delta t'_i|)$$

where

$$\Delta X_i = |X_{i+1} - X_i|, \ \Delta t_i = t_{i+1} - t_i$$

ΔX_i, $\Delta X'_i$, and $\Delta X''_i$ are positive since they are absolute values. Δt_i, $\Delta t'_i$, and $\Delta t''_i$ are positive since $t_{i+1} \geq t_i$ according the properties of time series data. And we know that α and β are non-negative. The proof completes due to Lemma 1. □

4.2 Event-driven subsequence match

Stream data comes in continuously. Performing similarity search upon all incoming data is not efficient for massive stream data management, especially not for real time applications such as stock market analysis. Another possible option is to do similarity search after a fixed time period (for example every 20 minutes). This will reduce the computation burden but it is insensitive to the changes between two query times and may lose some potentially important information. The event-driven similarity search proposed here will reduce the huge computation burden over the system as well as maintain sensitivity to changes.

An ***event*** means a new potential end point is being identified and no pruning is need. For example, in Figure 6, when $P(t_1)$, $P(t_2)$, $P(t_3)$, $P(t_5)$ are identified as potential end points, they are called events; while no event occurs when $P(t_4), P(t_6)$ are identified. The event-driven subsequence similarity matching performs automatic subsequence similarity search only at the time when there is a new event. The similarity search is totally automatic. The search requests are automatically generated by the online segmentation and pruning algorithm. The automatically generated query subsequence is the most recent *n* fixed and potential end points (including the newly identified potential end point at the event).

The automatic event-driven subsequence similarity has a trigger, which separates the online similarity search (the query engine) from the online segmentation and pruning process (the data engine). The data engine is for data acquisition and database updating, which runs all the time and processes each incoming stream data. The query engine can be turned on or off without affecting the data engine. This is a very friendly feature for application users. For some time periods, an application user may not want to trade, so the query engine is off while the data acquisition engine is still on. When the user returns to trade, the query engine is turned on with an up-to-date database.

5. TREND PREDICTION

The results of our online event-driven subsequence similarity matching can be analyzed using different analytical or statistical approaches for different applications. Practical utilizations of our subsequence similarity matching include trend prediction, new pattern recognition, and dynamic clustering of multiple data streams based on subsequence similarity. As a sample application, trend prediction is discussed in details as follows.

Each historical end point has a trend. A *trend* of an end point is the tendency of the raw stream after a given number (k) of end points from the current end point. The trend of one end point may be different for different durations of time. We define the trend of an end point based on the number of end points. *Trend-K* is the overall trend from the current end point to the next k^{th} end point. For simplicity, we define four trends: **UP, DOWN, NOTREND, UNDEFINED**. Given an end point E and its next k^{th} end point E_k, the trend of E is defined as follows (where ϵ is a user defined parameter):

If $E_k.X \geq E.X + \epsilon$, E.trend = UP;
If $E_k.X \leq E.X - \epsilon$, E.trend = DOWN;
If $E.X - \epsilon < E_k.X < E.X + \epsilon$, E.trend = NOTREND;
If E_k does not exist, E.trend = UNDEFINED.

According to the above definition, the most recent *k* end points have trend of UNDEFINED. All other historical end points have fixed trends of UP, DOWN or NOTREND. *k* is important in determining the trend of an event. Figure 8 gives an simple example of how the value of *k* affects the trend. For example, when *k* = 1, *b* has a DOWN trend (the price at *c* is lower than that at *b* by ϵ or more); but when *k* = 2, *b* has an UP trend (the price at *d* is higher than that at *b* by ϵ or more); and when *k* = 3, *b* has an NOTREND trend (the price between *e* and *b* is less than ϵ). Our experiments show that, if we choose the value of ϵ to be 10% to 20% of the average price change over a period, it is optimal for short-term trading, such as intra-day trading. Long-term trading favors a larger ϵ value.

Subsequence similarity search returns a list of end points. Each is the last end point of one retrieved subsequence. Simple statistical information are carried out on trends of retrieval end points, and the statistical results is used to predict the trend at the query event. Our statistical approach is simply to count how many UP, DOWN, NOTREND end points. Then we calculate the percentage of each trend *D* using the following formula:

$$F(D) = \frac{\#\ of\ retrieved\ end\ points\ with\ trend\ D}{total\ \#\ of\ all\ retrieved\ end\ points} \cdot 100\%$$

If there is a large number of similar subsequences at an event, its trend can be predicted based on F(D) value of each trend. We propose the following scheme:

if $|F(UP) - F(DOWN)| < F(NOTREND) + \lambda$,
predict **NOTREND**;

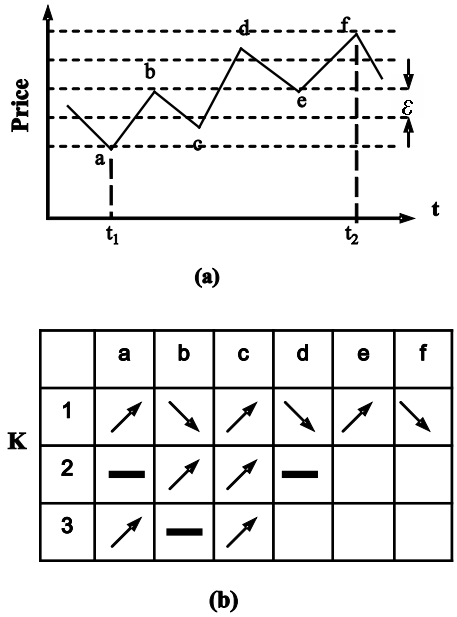

Figure 8: Trends of end points.

otherwise,
if $F(UP) > F(DOWN)$, predict **UP**;
else, predict **DOWN**.

Here λ is a user-defined threshold, e.g. 5%. For instance, if the similarity search retrieved 1000 similar subsequences from history, and the statistics show historical trends with 70% UP, 10% DOWN, 20%, the future moving trend of the query event can be predicted as UP. This is because F(UP)-F(DOWN)=60%, which is larger than F(NOTREND)+λ (=25%). As another example, if the historical trends are 51% UP and 49% DOWN, even though
F(UP) > F(DOWN),
we really should predict NOTREND since they are close.

6. PERFORMANCE EVALUATION

6.1 Experimental setup

We have evaluated the performance of our online event-driven subsequence similarity search based on the correctness of trend predictions. Real stock data are used in our experiments. For each incoming data stream, aggregated values per minute have been accumulated. More than 3,000,000 data points from 20 different stocks were used in experiments. First, about 3,000,000 historical data points are used as a test bed to set up all parameters and build the initial database. Another 500,000 new data points are tested for online similarity search followed by trend prediction. For simplicity, for one query, the query is performed on a single stream, which is the same as the query subsequence.

All our experiments are conducted on a DELL OPTIPLEX GX 260 with Pentium(R) 4 processor, 2.66GHz CPU, 1GB RAM. Series of experiments have been carried out on how to choose each parameter. For example, there is a series of experiments for *p*-interval moving average with *p* = 8, 10, 12, 15, 18, 20, 22, 25. Another series is for segmentation with δ_p^b = 0.05, 0.1, 0.12, 0.15, 0.2, 0.3 and δ_p^d = 0.1, 0.15, 0.2, 0.3. For the experiments discussed below, the following parameter setting is constant with moving average

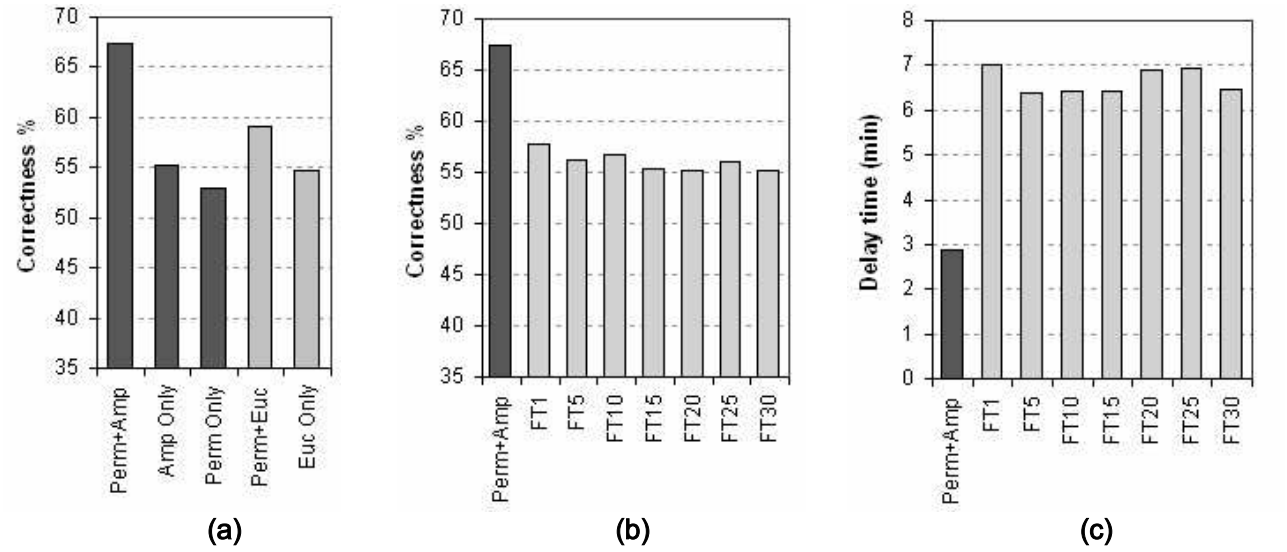

Figure 9: Correctness of trend predictions. (a) With different similarity measures; (b) By different query mechanisms; (c) Average delay time to identify an end point.

p = 20, segmentation sliding window size m = 10, segmentation threshold δ_s = 0.02, pruning threshold δ_p^b = 0.1. Other parameters depend on the properties of raw data streams. δ_p^d is about 10% - 20% of the average daily price change of a raw data stream. The similarity threshold γ of our distance function is $\frac{1}{3} \cdot \delta_p^d$. Trend duration K, trend range ϵ are user-specified as is the subsequence length parameter n. Typically, n is between 3 and 8.

Although we defined similarity using a distance function which includes time as well as amplitude, we do not use time here (β is zero and α is one). Our new similarity measure is thus called *Perm + Amp*, denoting that it is based only on the permutation and the amplitude.

Our experiments demonstrated that raw streams can be grossly grouped according to their average price changes over a fixed time period. If the average daily price changes are almost the same, the best parameter setting on one stream is almost the best setting on another stream. But the same setting may have quite different performance for two streams with different average price changes. The following discusses the performance of one group of streams whose average daily price changes are between \$1.00 to \$3.00. Other groups have displayed the same pattern with different parameter setting.

6.2 Experiments on similarity definition

Prediction accuracy using our new similarity measures is compared with similarity measures based on Euclidean distance. The Euclidean distance mentioned here allows subsequence similarity with price shifts and time scaling. Each experimental set has more than 1,600 queries and 500 predictions. The comparison is based on the accuracy in trend prediction. Figure 9a displays the percentage of correct predictions in 5 experimental sets. Accuracy based on our new similarity measure (*Perm+Amp*), which uses an amplitude distance function over permutation, achieved superior results. The correctness of prediction based on the Euclidean distance function (*Perm+Euc*) is 8% less than that based on our new similarity measure (*Perm+Amp*). Trend predictions based on permutation only (*Perm only*) and amplitude distance function only (*Amp only*) are also summarized in Figure 9a. Their performance is much less accurate than the combined one. These results also prove that the two parts of our similarity definition are complementary to each other and both are important. The percentage of correct trend predictions decreases more than 10% if using similarity measure based on permutation only or the distance function only. The same conclusion can also be summarized with Euclidean distance by comparing the performance among permutations only (*Perm only*), Euclidean distance only (*Euc only*), and the combination of permutations and Euclidean distance (*Perm+Euc*).

6.3 Experiments on event-driven matching

Our query engine uses an event-driven similarity search mechanism instead of querying for a fixed period. The correctness of trend predictions is summarized in Figure 9b. A series of experiments have been performed over different fixed time periods, ranging from 1 minute (*FT1*) to 30 minutes (*FT30*). *FTi* means query every *i* minutes. It is clearly shown that event driven subsequence similarity search (*Perm+Amp*) has outperformed search over any fixed period *FTi*. The different fixed time period searches have almost the same average correctness. This is because we use the most recent end points to search the database and predict the movement for the query time. The query sequence does not concern how far away the query point to the last identified potential end point (the delay time). It is easy to understand that the closer the query point to the last end point, the shorter delay time, and thus the better prediction accuracy. The fixed time period queries have almost the same prediction correctness because the correctness is the average accuracy in prediction.

Figure 9c shows average delay time for different query mechanisms. The delay time for event-driven similarity was 3min, while the delays for all the fixed periods were all around 7min. This delay time explains the lower correctness in trend prediction with fixed time periods. Each line segment covers about 30 raw data points. Although search over a fixed period could gives better prediction when the query point is close to the last end points, there is more changes the query points with fixed period would be far away from the last end point.

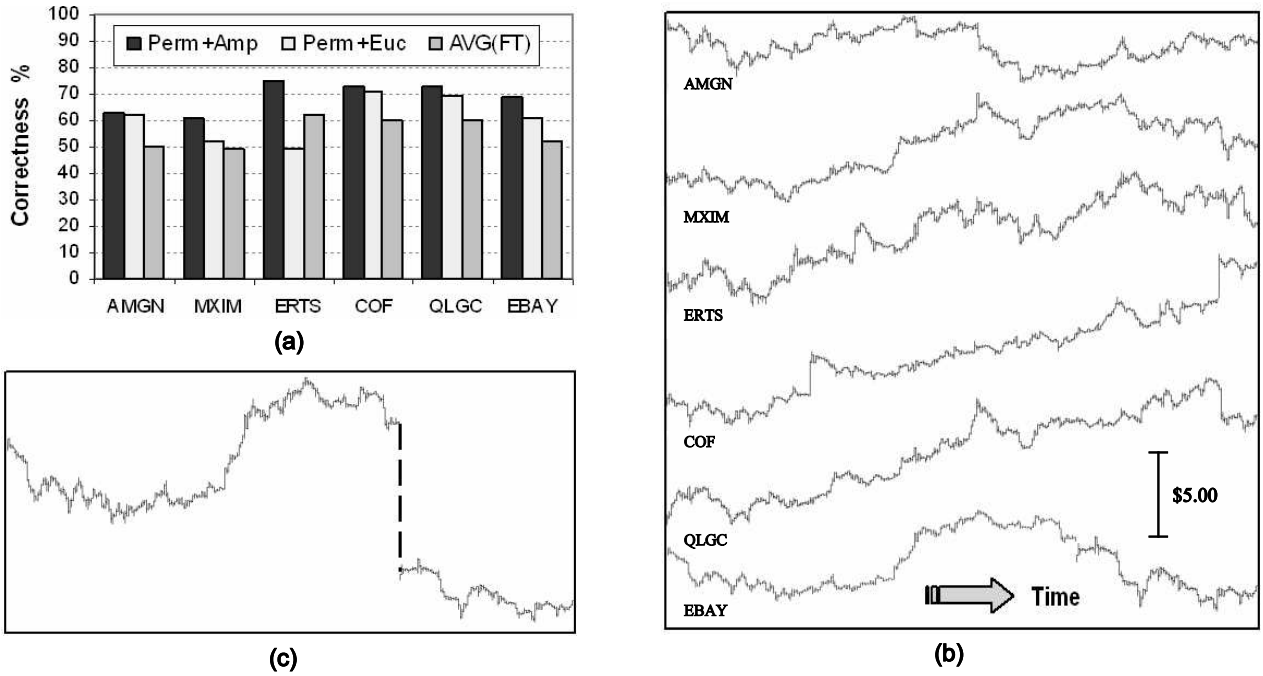

Figure 10: Subsequence similarity matching over differenct data streams. (a) Correctness of trend predictions; (b)The corresponding data streams; (c) The unadjusted raw EBAY stream.

6.4 Experiments on data streams

Experiments on the correctness of trend prediction over different streams have been performed to test the effects by stream properties. Figure 10a compares the correctness of trend predictions using different similarity measures ($Perm + Amp$ and $Perm + Euc$) and different search mechanisms over different data streams. $Perm+Amp$ and $Perm+Euc$ use event-driven similarity matching. $AVG(FT)$ is the average correctness of prediction with fixed time interval from 1min to 30min. It can be seen that the event-driven similarity search using our new similarity measure has better performance in all the streams. Figure 10b shows the raw data streams we used in our experiments whose results are in Figure 10a. From looking at these raw data streams, we can see that our method works well for a wide variety of data streams. The correctness of trend predictions are more than 60% for all the streams. It works better when the market is in an overall bull/bear market (*ERTS, COF* and *QLGC*), where the trend predictions are more than 70% correct. Even when the market is a no-trend market ($AMGN$ and $MXIM$), our prediction scheme still works well, with prediction correctness more than 60%. On the contrary, the correctness of predictions based on Euclidean distance or with fixed time interval varies according to the characteristics of different streams, sometimes only achieving totally random predictions (50% correctness).

The stream of EBAY is of great interest because it requires dynamic adjustment as described in Section 3.4. Figure 10b shows the adjusted stream of EBAY and Figure 10c is the raw EBAY stream without adjustments. Figure 10c starts with a bear market. Then it changes to a bull market, followed by a no-trend market (before the dashed line). At the time of the dashed line, the stock has a share split of 2:1 (one share to two shares split) and the price has a sharp drop from $110.00 to $55.00. Our event-driven subsequence similarity matching dynamically adjusts this special situation gracefully, with 70% correct predictions. The corresponding predictions with Euclidean distance are only 60% correct. The correctness of predictions based on fixed time intervals is much worse (almost random — 51% correct).

6.5 Experiments on query subsequence length and trend duration

A series of experiments has been performed to test the effect of subsequence lengths and trend durations. For financial data analysis, the lengths of subsequences of the resulting PLR range from 3 to 8. Our experimental results show that when the average lengths between two adjacent PLR end points is between 30 to 40, our method will have better prediction results. When the subsequence has 3 points, there are 2 intervals (line segments). Thus the corresponding subsequence of raw streams contains 60 to 80 data points. When the subsequence has 8 points, there are 7 intervals. Thus the corresponding subsequence of raw streams contain 210 to 280 data points.

For each of the experiments shown in Figure 11, we use the same streams and query periods using our method. We only vary the query subsequence length. For longer query subsequences, there are more permutations. Hence there are fewer similar subsequences available for distance comparison. In spite of this, Figure 11a shows that longer query subsequences result in better prediction correctness. Figure 11b illustrates the relative number of similar subsequences (with same permutation) based on length. In addition, because there are fewer similar subsequences, the number of times a prediction can be made(i.e., "UP" or "DOWM", but not "NOTREND") is smaller with longer query subsequences and this is also illustrated in Figure 11b.

Figure 11c shows the correctness of trend predictions with different trend duration k. We can see that the closer the prediction events to the query events, the more accurate the predictions are.

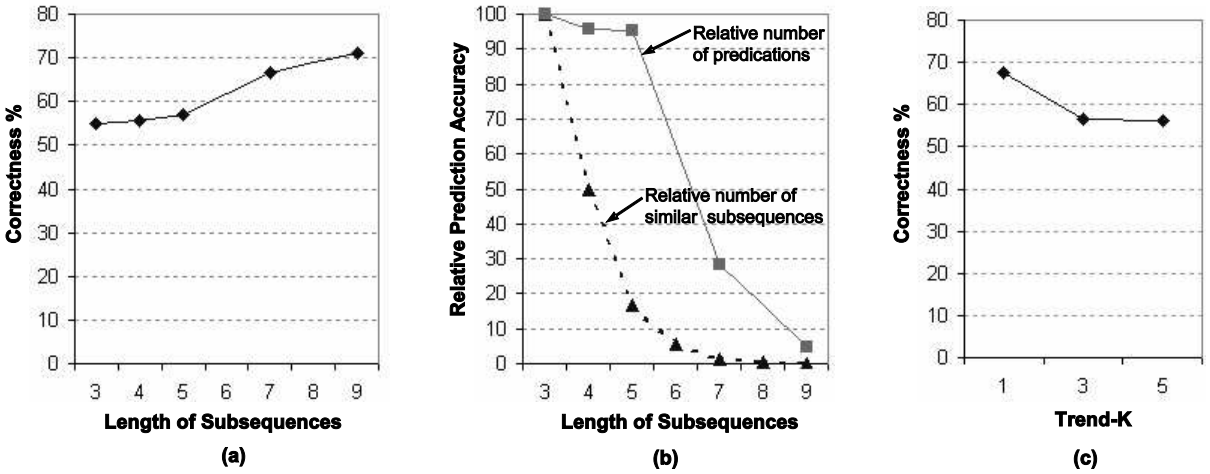

Figure 11: Trend predictions with different subsequence lengths and trend durations. (a) Correctness of trend predictions with different subsequence lengths; (b) Relative prediction accuracy with different subsequence lengths; (c) Correctness of trend predictions with different trend duration k.

6.6 Experiments on CPU cost and query time

We define the *CPU cost* as the average computation time for subsequence similarity matching by sequential scan. The *relative CPU cost* is measured relative to the CPU cost of FT1(one minute time periods). As shown in Figure 12, it is easy to understand that the CPU cost decreases as the fixed query time intervals increase. It also shows CPU cost of event-driven subsequence similarity matching is about the same as FT30 (30-minute time periods). The CPU cost of similarity matching based on permutations and a distance function (*Amp+Perm* and *Euc+Perm*) is only a little higher based on permutations (*Perm Only*). This is because we store the permutations of historical data and thus save run time computations. We only compute the distance between the historical subsequences whose permutations is the same as that of the current query subsequence. It also explains why similarity matching methods based on distance functions only (*Amp Only* and *Euc Only*) have higher CPU cost. They do not have the pre-computed permutations as a filter, so they need to compute the distance between each historical subsequence and the query subsequence. So our event-driven similarity matching mechanism has greatly reduced the CPU cost.

Our event-driven similarity matching runs in real time. Using similarity based on permutations and the amplitude distance function, the average response time for a single query is only 130 milliseconds for subsequences with length 7 and queries on the database with 100,000 PLR end points, which corresponds to 3,000,000 raw stream data points. The raw financial data streams come in with irregular time intervals, and we aggregated the raw data with a fixed time interval, which is 1 minute in our case. A 130-millisecond responding time is fast enough for real-time predictions.

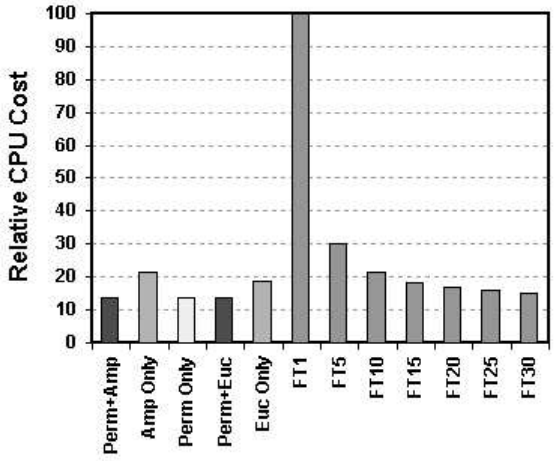

Figure 12: Relative CPU cost to evaluate query with different similarity measures and matching mechanisms.

7. CONCLUSIONS & FUTURE WORK

In this paper, we introduced a new approach for event-driven subsequence similarity matching based on a newly defined subsequence similarity measure over financial data streams. Upon studying the special requirements and real-time application needs of financial data analysis, we proposed a new simultaneous online segmentation and pruning algorithm for piecewise linear representation of raw financial data streams. The new algorithm used tiered processes for incremental segmentation. It features quick identification of new end points yet maintains accurate segmentation. We also defined a new subsequence similarity measure for subsequence matching. The new similarity measure is composed of two parts, a permutation and a distance function. Experimental results showed it has better performance than subsequence similarity measures based on Euclidean distance. An event-driven online subsequence similarity search approach is proposed, in which automatic online queries are generated only at a time when a line segment is generated. The new search mechanism had about 30 times less computational burden than the scheme to query at each time instance. Performance experiments demonstrated that event-driven search outperformed the searches with any fixed time period. Using the similarity search results as a guidance, we have achieved promising trend prediction correctness (average 68%). Our approach works well for a wide variety of streams. The query response time is fast for real time applications.

Future research can proceed in several directions. Our immediate plans include incorporating indexing in the search algorithm. Since we have shown that our distance function is metric, a number of indexes [7, 9, 10, 34, 36, 37] may be applicable. Another possibility is to explore a weighted statistical function to improve trend prediction. The weighted statistics will consider the effects of similarity distance and time difference. Still another problem is finding an algorithm for dynamic clustering of multiple streams. Last, scheduling and concurrency control of multiple queries over massive data streams in real-time applications is also of great research interest.

8. REFERENCES

[1] C. C. Aggarwal, J. Han, J. Wang, and P. S. Yu. A Framework for Clustering Evolving Data Streams. *VLDB*, pages 81–92, 2003.

[2] R. Agrawal, C. Faloutsos, and R. Swami, A. Efficient Similarity Search in Sequence Database. *FODO*, pages 69–84, 1993.

[3] S. Babu and J. Widom. Continuous Queries Over Data Stream. *SIGMOD Record*, 30(3):109–120, 2001.

[4] D. Barbara and P. Chen. Using the Fractal Dimension to Cluster Datasets. *SIGKDD*, pages 260–264, 2000.

[5] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In *ACM SIGMOD*, pages 322–331, 1990.

[6] J. A. Bollinger. *Bollinger on Bollinger Bands*. McGraw-Hill, first edition, 2001.

[7] T. Bozkaya and M. Ozsoyoglu. Distance-Based Indexing for High-Dimentional Metric Spaces. *SIGMOD*, pages 357–368, 1997.

[8] K.-P. Chan and A.-C. Fu. Efficient Time Series Matching by Wavelets. *ICDE*, pages 126–133, 1999.

[9] T. Chiuch. Content Based Image indexing. *VLDB*, pages 582–593, 1994.

[10] T. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Space. *VLDB*, pages 426–435, 1997.

[11] G. Das, D. Gunopulos, and H. Mannila. Finding Similar Time Series. *PKDD*, pages 88–100, 1997.

[12] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining Stream Statistics over Sliding Windows. *SODA*, pages 635–644, 2002.

[13] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast Subsequence Matching in Time-Series Database. In *SIGMOD*, pages 419–429, 1994.

[14] E. Fink and K. B. Pratt. Indexing of compressed time series.

[15] A. J. Frost and R. R. Prechter. *Elliott Wave Principle*. New Classics Library, first edition, 1998.

[16] G. P. C. Fung, J. X. Yu, and W. Lam. News Sensitive Stock Trend Prediction. *PAKDD*, pages 481–493, 2002.

[17] L. Gao and X. S. Wang. Continually Evaluating Similarity-Based Pattern Queries on a Streaming Time Series. In *SIGMOD*, pages 370–381, 2002.

[18] L. Gao, Z. Yao, and X. S. Wang. Evaluating Continuous Nearest Neighbor Queries for Streaming Time Series via Pre-fetching. In *CIKM*, pages 485–492, 2002.

[19] J. Gehrke, F. Korn, and D. Srivastava. On Computing Correlated Aggregates over Continual Data Streams. *SIGMOD*, pages 126–133, 2001.

[20] A. C. Gilbert, Y. Kotidis, and S. Muthukrishnan. Surfing Wavelets on Streams: One-pass Summaries for Approximate Aggregate Queries. *VLDB*, pages 79–88, 2001.

[21] L. Golab and M. T. Ozsu. Issues in Data Stream Management. *SIGMOD Record*, 32(2):5–14, 2003.

[22] S. Guha, N. Rastogi, R. Motwani, and L. O'Callahan. Clustering Data Stream . *IEEE FOCS Conference*, pages 359–366, 2000.

[23] T. Hellstrm and K. Holmstrm. "Predicting the Stock Market". 1998.

[24] E. J. Keogh, K. Chakrabarti, S. Mehrotra, and M. J. Pazzani. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. In *SIGMOD*, pages 151–162, 2001.

[25] E. J. Keogh, K. Chakrabarti, M. J. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems*, 3(3):263–286, 2001.

[26] E. J. Keogh, S. Chu, D. Hart, and M. J. Pazzani. An Online Algorithm for Segmenting Time Series. In *ICDM*, pages 289–296, 2001.

[27] D. Komo, C. Chang, and H. Ko. "Neural Network Technology for Stock Market Index Prediction". *ISSIPNN*, pages 543–546, 1994.

[28] F. Korn, H. V. Jagadish, and C. Faloutsos. Efficiently Supporting ad hoc Queries in Large Datasets of Time Sequences. In *SIGMOD*, pages 289–300, 1997.

[29] X. Liu and H. Ferhatosmanoglu. Efficient k-NN Search on Streaming Data Series. In *SSTD*, pages 83–101, 2003.

[30] Y.-S. Moon, K.-Y. Whang, and W.-S. Han. General Match: a Subsequence Matching Method in Time-series Databased Based on Generalized Windows. In *SIGMOD*, pages 382–393, 2002.

[31] L. O'Callaghan, A. Meyerson, R. Motwani, N. Mishra, and S. Guha. Streaming-Data Algorithms for High-Quality Clustering. *ICDE*, pages 685–, 2002.

[32] S. Park, S.-W. Kim, and W. W. Chu. Segment-Based Approach for Subsequence Searches in Sequence Databases. *SAC*, pages 248–252, 2001.

[33] D. Rafiei and A. Mendelzon. Similariy-Based Queries for Time-series data. *SIGMOD*, pages 13–24, 1997.

[34] J. Uhlmann. Satifying General Proximity Similarity Queries with Metric Trees. *IPL*, 4:175–179, 1991.

[35] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient Retrieval of Similar Time Serquences under Time Warping. In *ICDE*, pages 201–208, 1998.

[36] P. N. Yianilos. Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces. *Proc. ACM-SIAM Symposium on Discrete Algorithms*, pages 311–321, 1993.

[37] C. Yu, B. Ooi, K. Tan, and H. Jagadish. Indexing the Distance, an Efficient Method to KNN Processing. *VLDB*, pages 421–430, 2001.

[38] Y. Zhu and D. Shasha. StatStream: Statistical Monitoring of Thousands of Data Streams in Real Time. *VLDB*, pages 358–369, 2002.