Incremental Stock Time Series Data Delivery and Visualization

Tak-chung Fu[1,2,*], Fu-lai Chung[1], Pui-ying Tang[1], Robert Luk[1] and Chak-man Ng[2]

[1]Dept. of Computing, Hong Kong Polytechnic University, Hunghom, Hong Kong.
[2]Dept. of Computing and Information Management, HKIVE (Chai Wan), Hong Kong.

ABSTRACT

SB-Tree is a binary tree data structure proposed to represent time series according to the importance of data points. Its use in stock data management is distinguished by preserving the critical data points' attribute values, retrieving time series data according to the importance of data points and facilitating multi-resolution time series retrieval. As new stock data are available continuously, an effective updating mechanism for SB-Tree is needed. In this paper, a comparative study of different updating approaches is reported. Three families of updating methods are proposed. They are periodic rebuild, batch updating and point-by-point updating. Their efficiency, effectiveness and characteristics are compared and reported through various experiments.

General Terms

Algorithms, Performance, Theory.

Keywords

Time series representation, Binary tree data structure, Incremental data delivery, Multi-resolution visualization, Time series data management.

1. Introduction

A time series is a collection of observations made chronologically. Time series data can be easily obtained from scientific and financial applications, for examples, daily temperatures, daily sale totals and prices of mutual funds. The natures of time series data include large in data size and high dimensionality. Indeed, a large set of time series data is from the stock market. Stock time series has its own characteristics over other time series data, e.g. electrocardiogram (ECG). Stock time series is typically characterized by a few critical points and multi-resolution consideration is always necessary for long-term and short-term analyses. It is always updated frequently and continuously. In addition, technical analysis is usually used to identify patterns of market behavior, which have high probability to repeat themselves. These patterns are similar in the overall shape but with different amplitudes and/or durations.

Time series is typically represented by techniques like discrete Fourier transform (DFT) [1], discrete Wavelet transform (DWT) [2], and singular value decomposition (SVD) [3]. These approaches are based on global transformation that the time series is transformed to the frequency domain. The problem of this global nature of the transformation process is that adding a new data point to the time series requires re-computation for the entire time series. It makes incremental updating impossible. Moreover, these three approaches will smooth out the critical points when representing the time series in compressed form. Another approach, piecewise aggregate approximation (PAA), represents the time series as a sequence of box basis functions [4]. It supports incremental updating by accumulating every *n* new data points. However, if visualizing the represented time series directly, it looks ascetically unpleasing and same as DFT, the critical points in the time series are smoothed out. Such a shortcoming is particularly critical in stock applications, where the extreme price values are essential to characterize the stock patterns and investment decisions.

In view of the need to manipulate stock time series data effectively and efficiently, a stock time series representation scheme called specialized binary tree (SB-Tree) is proposed in [5] with its applications reported in [6-9]. In this paper, the incremental updating mechanisms for this representation are studied and reported. The paper is organized into five sections. A brief review of SB-Tree is given in section 2. Section 3 introduces three categories of updating methods for SB-Tree, namely, periodic rebuild, batch updating and point-by-point updating. Five different updating methods are proposed and the simulation results are reported in Section 4. The final section makes the conclusion of the paper.

2. A Specialized Binary Tree (SB-Tree) for Stock Time Series Representation

In this section, the SB-Tree time series representation scheme is briefly reviewed. It is based on determining the data point importance in the time series. Instead of storing the time series data according to time or transforming it into other domains (e.g. frequency domain), data points of a time series are stored according to their importance.

2.1 Data Point Importance

In view of the importance of extreme points in stock time series, the identification of *perceptually important points* (PIP) is first introduced in [10] and used for pattern matching of technical (analysis) patterns in financial applications. Similar idea can be found in independent works [11-14].

The frequently used stock patterns are typically characterized by a few critical points. For example, the head-and-shoulder pattern

* Corresponding Author: cstcfu@comp.polyu.edu.hk

consists of a head point, two shoulder points and a pair of neck points. These points are perceptually important in the human identification process and should be considered as having higher importance. The proposed scheme follows this idea by reordering the sequence *P* based on PIP identification, where the data point identified in an earlier stage is considered being more important than those points identified afterwards. The perceptual importance is generally defined by the vertical distance (VD) between the point of interest to its current pair of adjacent PIPs.

2.2 SB-Tree

After introducing the concept of data point importance, a specialized binary tree (SB-tree) structure was suggested to store the time series data. Its creation is based on the PIP identification process. As exemplified in Fig.1, the first and the last data points in the sequence, being considered as the first and second PIPs, are assigned as the root and the first left child of the tree respectively. The third PIP identified, i.e. point 5 which makes the largest vertical distance to points 1 & 10, becomes the right child of node 1. It is typically considered as the real root of the tree. The subsequent tree building process can be described by the following recursive statements. Starting with the current node pointer *ptr* and a new node *node* for the new PIP identified,

- If *node->x* is less than *ptr->x* then goto the left arc of *ptr*
  - If *ptr->left* is empty, add *node* to this position
  - Else *ptr* = *ptr->left* and start the next iteration
- Else goto the right arc of *ptr*
  - If *ptr->right* is empty, add *node* to this position
  - Else *ptr* = *ptr->right* and start the next iteration

By applying the above recursive procedures, a SB-Tree can be created for a given time series.

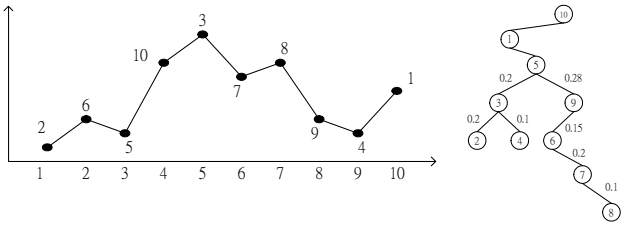

Fig.1. Sample time series and the SB-Tree built

2.3 Accessing SB-Tree

Given the required number of data points *n*, the SB-Tree is accessed recursively. By starting from the root node, the time series data points are retrieved according to their importance.

- Starting from the root of the SB-Tree, it is the first point to be retrieved and will be marked as USED
- The SB-Tree is accessed from the root and each accessible node in each path of the tree will be reached. An accessible node is defined as the first node in a path that is not marked as USED
- By comparing the distances among all the accessible nodes, the one with maximum distance is selected as the next PIP. Again, this node will be marked as USED when retrieved
- This process continues until the required number of data points is retrieved (i.e. marked as USED).

Fig.2 shows the visual effect of retrieving different number of data points from the SB-Tree created in Fig.1. Thus, multi-resolution visualization of time series data can be facilitated. As demonstrated in Fig.3, the Hong Kong Hang Seng Index (HSI) can be displayed in different resolutions to suit different application environments, e.g. mobile devices with limited screen size/resolution.

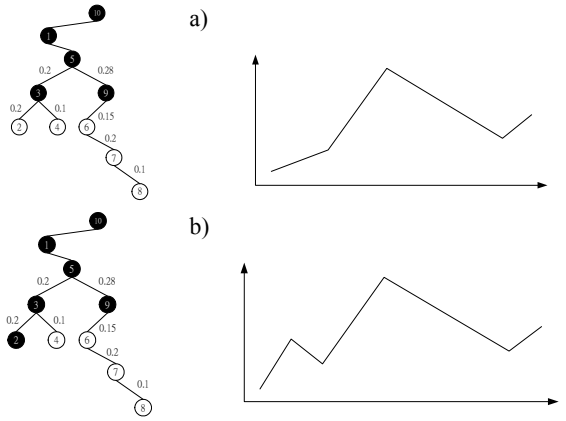

Fig.2. Retrieving different number of data points from the SB-Tree : (a) 5 data points and (b) 6 data points

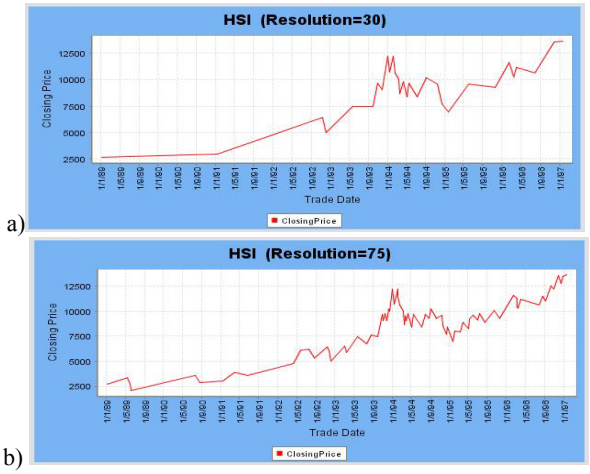

Fig.3. Hang Seng Index time series (with 2000 data points) in different resolutions: (a) 30, (b) 75

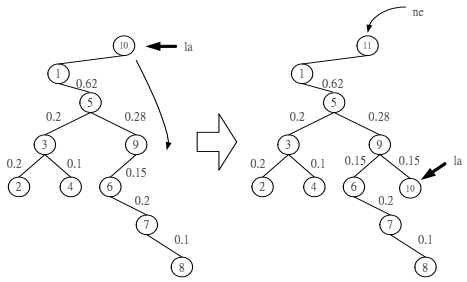

Fig.4. Appending a new data point to the SB-Tree

3. Incremental Updating of the SB-Tree

New time series data is available frequently and continuously. On the other hand, build/rebuilding a SB-Tree for the newly arrived data points is very time consuming. Therefore, an efficient updating mechanism for the SB-Tree is necessary. In this section, three categories of updating methods, namely, periodic rebuild, batch update and point-by-point update, are proposed.

3.1 Periodic Rebuild

Rebuilding the SB-Tree for every new data point is time consumption or unnecessary because there might not have

significant change in the SB-Tree structure. Therefore, periodic rebuild might be one possible solution. In our SB-Tree, the last data point of the current time series is the root node. When there is a new data point *ne*, this point will become the last data point in the updated time series. Therefore, it should be added as the root node of the SB-Tree and the last data point before updating *la* (the root node before update) will be relocated to the right-hand-side of the rightmost leaf node as shown in Fig.4.

According to the SB-Tree building process, the vertical distances (VDs) for the first two PIPs are always zero. That means the VD of the last data point before updating *la,* in any circumstance, is zero. Since the accessing order of the nodes is based on their VDs, the VD of *la* should be computed and it should be set as the VD between *la* and the line connected the parent node of *la* and the new data point *ne*. Otherwise, it will not be traversed before all nodes with VD larger than zero have been accessed (because its VD is equal to zero).

After a number of points are added, the whole tree is required to be rebuilt in order to guarantee the SB-Tree structure. However, the problem is how to determine the appropriate time to rebuild the whole SB-Tree. The precision of the points retrieved and the process time are directly proportional to the frequency of rebuilding the tree. The more frequent rebuilding is, the higher precision and the longer processing time are.

The simplest way is to rebuild the tree after updating a certain number of data points. However, this rebuild indicator is too general which ignores the characteristic of SB-Tree and its PIP identification process. A SB-Tree is effective in displaying a time series with a few critical points which are the relative maxima or minima of a segment of the time series. PIP identification process is used to select the most critical points from these relative maxima and minima. If there is a new coming data point, the most important issue is to determine whether there is any change of the current PIP sequence after adding the point. The change of PIP sequence would result in rebuilding the SB-Tree. In general, there is no significant change in the SB-Tree structure if the new coming data point follows closely the current rising or dropping trend. In other words, changing of PIP sequence usually happens when there is a trend reversal or any extreme points such as a peak or a trough appear. Thus, a function, which could be able to indicate the trend reversal, is chosen as the rebuilding indicator. Moving averages (MA) of stock prices is a calculated effort to eliminate or minimize the fluctuation of the numerical value of the phenomenon being observed so that an underlying trend could be recognized when a sequential series of that phenomenon is reviewed. One of the most widely used MA is Simple Moving Average (SMA). SMA gives equal weighting to each day's price of the number of days chosen. This is simply calculated by getting the average of a set of values, say the past 20 days of values.

By using two SMAs, not only the fluctuations of time series are smoothed twice thereby minimizing misleading whipsaws, but also the trend reversal will be indicated shortly after it has taken place. For example, the intersection between a rising short-term SMA and a longer term SMA could signal the start of an uptrend, or vice versa. Thus, the trend reversal signal is given when two SMAs are intersected. Since an intersection of two SMAs can signify trends, it is adopted as the indicator for rebuilding an SB-Tree. Fig.5 shows the algorithm of the periodic rebuild approach (PRA).

```
Function Periodic_Rebuild(root, P)
   Input:   SBTree root
            sequence P[1..m+1]
   Output:  SBTree root // Updated
Begin
   Calculate SMA1 and SMA2

   If SMA1 and SMA2 crossover Then
      Create_SB_Tree(P[1..m+1])
   Else
      Create new_node of the p[m+1]
      new_node->left = root->left
      new_node->right = NULL

      ptr = new_node->left
      Repeat Until (ptr->right = NULL)
      Begin
         ptr = ptr->right
      End
      ptr->right = root

      root->left = NULL
      root->right = NULL
   End

   root = new_node
   Return root
End
```

Fig.5. Pseudo code of the periodic rebuild approach (PRA)

3.2 Batch Update

In this subsection, an updating method called sub-tree updating (STU) is proposed for appending a batch of data points to the original SB-Tree. Intuitively, it may be more efficient to accumulate a few data points and update them in batch instead of appending one after another. The idea here is to materialize the original SB-Tree and consider the newly arrived data points as a new tree. Therefore, the two segments of the time series can be joined together by adding the new tree built by a batch of new data points as a sub-tree to the original materialized SB-Tree. Moreover, the size of the sub-tree is the number of new data points to be updated for each process. The sub-tree size can be customized and various sub-tree sizes would perform differently in terms of processing time and precision of data point retrieval.

Fig.6 shows an example of joining a sub-tree to the materialized SB-Tree. The first node of the sub-tree will be changed as the root node of the SB-Tree and the third node of the sub-tree will then be added to the rightmost node (under the root node) of the SB-Tree. In this example, the sub-tree size is 5. That means there are 5 new data points (from 11 to 15) for updating. We propose to create a 7-node sub-tree for the data points from 9 to 15, instead of starting from 11 to 15. Building the sub-tree starting from 2 data points before the first new data point is for pre-computing the VD of the node which joins the two tree (node 10 in this case). Fig.7 shows the algorithm of the proposed sub-tree updating (STU) method.

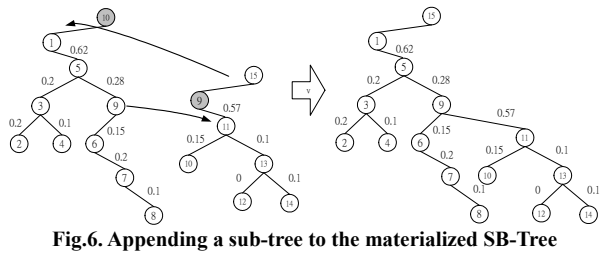

Fig.6. Appending a sub-tree to the materialized SB-Tree

```
Function Subtree_Updating(root, subtree)
   Input:   SBTree root
            SBTree subtree
   Output:  SBTree root // Updated
Begin
   ptr = root->left
   Repeat Until (ptr->right = NULL)
   Begin
      ptr = ptr->right
   End

   ptr->right = subtree->left->right
   subtree->left = root->left
   root = subtree

   Return root
End
```

Fig.7. Pseudo code of sub-tree updating (STU) method

In STU, sub-trees are only appended to the rightmost part of the original SB-Tree without changing its structure. As a result, the tree will be biased to the right and retrieving the most important PIPs might have problems. For this reason, rebuilding the whole SB-Tree should be applied after adding a certain amount of new data points to eliminate the derivation between the sub-tree updating and the exact building of the tree. We term this approach sub-tree updating rebuild (STUR). Similarly, the SMA indicator described in previous section can be employed to trigger the rebuilds.

3.3 Point-by-Point Update

Although rebuilding the whole SB-Tree periodically could reduce the time of the updating process, the structure of the SB-Tree cannot be guaranteed after each update process. Moreover, the time consumed to rebuild the whole tree may be very long since its processing time increases exponentially as number of points increases. In this subsection, two point-by-point updating methods are proposed.

3.3.1 Change of PIP Detection

A necessary step for the point-by-point updating methods is to determine whether there is any change of the PIPs identified after adding a new data point. If the same PIPs are identified, it shows that there is no change in this iteration and the same checking can be carried out along the right-hand-side of the SB-Tree. It is because the new data point is appended to the rightmost of the time series, the potentially affected segment is only located on the right-hand side. In other words, if there is a change of the PIP at the root node, it will be the same as creating a brand new SB-Tree (see Fig.8a and Fig.1). On the other hand, if there is no change on the leaf nodes of the rightmost path, the new data point is simply added to the right-hand-side of the rightmost leaf node (see Fig.8b). With this behavior, the updating process can be formulated as follows:

- Starting from the real root of the SB-Tree, *ptr*
- If its PIP still makes the maximum VD, there will be no change on *ptr* and the left sub-tree.
  - *ptr = ptr->right,* and repeat the checking again (for the right sub-tree).
- Else reconstruct the sub-tree under this node by the tree construction mechanism introduced in section 2.2.

Now, the problem is how to determine whether there is any change of PIP identified after adding a new data point. Two approaches are proposed.

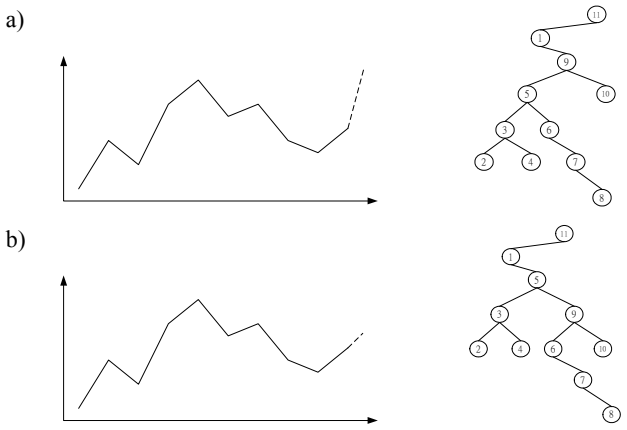

Fig.8. Different behaviors after adding a new data point

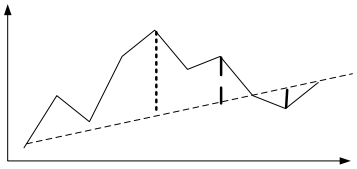

Fig.9. Distances stored for determining if there is any change of PIP identified after adding a new data point

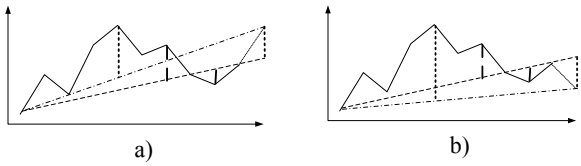

Fig.10. Determining if there is any change of PIP identified after adding a new data point

3.3.2 Guarantee Updating Approach (GUA)

If prior information is provided, it is possible to detect the change of PIPs in an efficient manner. First, more information has to be stored in each node. They are the VD of the data points with the second largest distance on the same side of the current PIP identified and the VD of the data point with maximum distance on the opposite side of the evaluating segment. The VD of the alternate PIP on the same side of the identification process is defined as *au* while the one on the opposite side is defined as *al* as shown in Fig.9.

With this information, the change of PIP can be detected by checking if the new VD of the original PIP identified matches with the following situations: (i) if the new data point is on the same side of the previous PIP identified, then, it is the sum of the alternate VD on the opposite side and the vertical derivation between the previous slope and the new slope after added the new point *de'* (see Fig.10(a), i.e., *al*+*de'* in this case), (ii) if the new data point is on the opposite side of the previous PIP identified, it is the sum of the alternate VD on the same side and the vertical derivation between the previous slope and the new slope after added the new point *de'* (see Fig.10(b), i.e., *au*+*de'* in this case). If the new VD of the original PIP identified still has the larger distance, the same PIP is identified. Otherwise, there is a potential change of PIP. Therefore, to guarantee the tree built by the updating process is the same as rebuilding the whole tree, the segment under this node is suggested to be rebuilt. Fig.11 shows the guarantee updating approach (GUA).

```
Function Point_by_Point_Updating(root, P)
   Input:   SBTree root
            sequence P[1..m+1]
   Output:  SBTree root // Updated
Begin
   Create new_node of the p[m+1]
   new_node->left = root->left
   new_node->right = NULL

   position=Find_Rebuild_Position(root->left, P)

   ptr = new_node->left
   If (position = NULL) Then
      Repeat Until (ptr->right = NULL)
      Begin
         ptr = ptr->right
      End
      ptr->right = root
      root->left = NULL
      root->right = NULL
   Else
       Repeat Until (ptr->x = position) OR
      (ptr->right = NULL)
      Begin
         prev_ptr = ptr
         ptr = ptr->right
      End
      st = prev_ptr->x

      prev_ptr->right =
      Create_SB_Tree(P[st..m+1])
   End
   root = new_node
   Return root
End

Function Find_Rebuild_Position(node,sequence)
   Input:   SBTree root
            sequence P[st..m+1]
   Output:  coord-x position
Begin
   Calculate VD of p[m+1] according to p[st]
   and p[m] and assign to delta
   Calculate the new VD of node->x and assign
   to node->dist

   If P[m+1] located on the same side as node->x
   according to the line joined p[1] and p[m]
   Then
      If (node->al + delta > node->dist) Then
         Return node->x
      Else
         node->al = node->al + delta
      End
   Else
      If (node->au + delta > node->dist) Then
         Return node->x
      Else
         node->au = node->au + delta
      End
   End

   If (node->right = NULL) Then
      Return NULL
   Else
      position = Find_Rebuild_Position
       (node->right, P[node->x..m+1])
   End

   Return position
End
```

Fig.11. Pseudo code of guarantee updating approach (GUA)

The GUA can be exemplified using the example in Fig.1. As shown in Fig.12, new data points have been appended to the original time series. Given the data point 11, as the shape of the overall time series has been significantly changed, when evaluating the root node with the new entry, data point 9 (the point with maximum distance on the opposite side before) should replace the original PIP and becomes the new PIP identified. Therefore, rebuilding the whole tree is needed (Fig.13b). The same case happens again when adding data point 12 (Fig.13c). In this example, rebuilding of the whole tree frequently occurred because of the short length of the time series and the great fluctuation of the data points. In most of the cases, rebuilding always happens in a small sub-tree such as adding data points 13 to 15. Data point 12 is only needed to be appended to the SB-Tree in Fig.13d and rebuilding the sub-tree under data point 12 is needed when data point 14 has been added (Fig.13e). It is obvious that a SB-Tree is based to grow to the right-hand-side of the tree until the tree is greatly unbalanced and the whole tree is needed to be rebuilt (i.e. the shape of the time series is greatly influenced once a considerable number of data points are accumulated). The frequency and the size of rebuilding a sub-tree directly reflect the fluctuation of the time series.

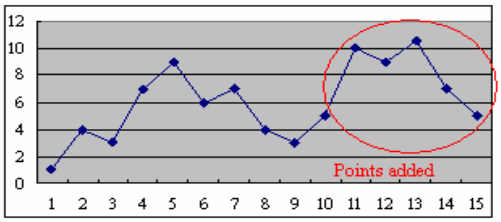

Fig.12. Five data points are added to the sample time series

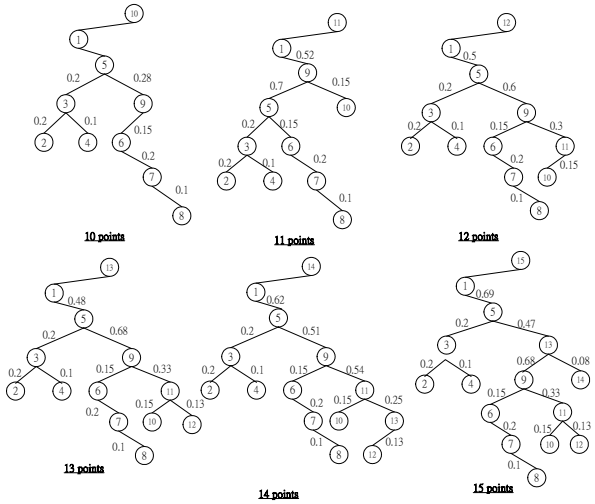

Fig.13. The updating process of the SB-Tree by GUA

3.3.3 Approximate Updating Approach (AUA)

Although the above change of PIP detection method can guarantee that the structure of the SB-Tree updated is the same as that by rebuild the whole tree, it leads to frequent rebuilding of the SB-Tree. Therefore, an approximate updating process is also proposed. Instead of guaranteeing the VD of previous PIP identified being larger than the possible VDs of all the data points in the time series, the VDs of the (i) previous PIP identified *vd'*, (ii) the data point with the second largest distance on the same side of the current PIP identified au', (iii) the data point with maximum distance to the opposite side of the evaluating segment *al'* and (iv) the last data point before updating a new data point *la* are compared to estimate if there is any change of PIP identified (see Fig.14). The modified function for finding the rebuilt

position is shown in Fig.15. Instead of storing the information of the alternate distances measured during tree building, the locations of the alternate points should be stored in the nodes for approximate updating. The tree updated by this method may not ensure that it is the same as to rebuild the tree because the possible PIP identified is not only limited to these three data points, but it is true in most of the cases. Moreover, the whole SB-Tree can be rebuilt after a certain amount of PIPs accumulated to eliminate the derivation between the approximate updating and the exact building of the tree (see section 3.1).

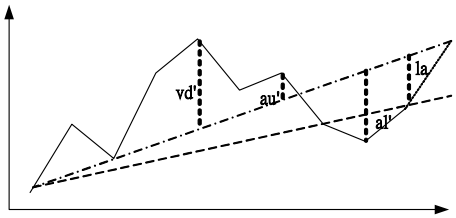

Fig.14. Determine the change of PIP identified after adding a new data point by vertical distance measurement among (i) previous PIP identified (vd'), (ii) the data point with the second largest distance in the same side of the current PIP identified (au'), (iii) the data point with maximum distance in the opposite side of the evaluating segment (al') and (iv) the last data point before updating a new data point (la)

```
Function Find_Rebuild_Position
(node, sequence)
   Input:  SBTree root
           sequence P[st..m+1]
   Output: coord-x position
Begin
   Calculate VD of p[m+1] according to p[st]
   and p[m] and assign to delta

   Calculate the new VD of node->x and assign
   to node->dist

   Calculate the new VD of node->al and
   assign to node->al
   If (node->al > node->dist) Then
      Return node->x
   End

   Calculate the new VD of node->au and
   assign to node->al
   If (node->au > node->dist) Then
      Return node->x
   End

   If (node->right = NULL) Then
      return NULL
   Else
      position = Find_ Rebuild_Position
       (node->right, P[node->x..m+1])
   End
   Return position
End
```

Fig.15. Pseudo code of the Find_Rebuild_Position function for approximate updating approach (AUA)

4. Experimental Results

The SB-Tree for time series representation was implemented in Java 2 Platform, Standard Edition (J2SE). The environment for running the experiments is: Intel Pentium(R) M Processor 1.4Ghz with 512M RAM. The performance of different updating approaches is evaluated in terms of precision and speed. Time series with 2500 data points captured from the past ten years of Hong Kong Hang Seng Index (HSI) is used for simulation and demonstration.

After a series of preliminary experiments, the SMA pair of 10-day and 30-day SMAs was selected to determine the time for rebuilding the SB-Tree in both periodic rebuild approach (PRA) and sub-tree update rebuild (STUR). Also, the size of the sub-tree is set to 10 for batch updating. The short forms (label) used to represent different updating approaches are summarized below.

Table 1. Labels used to represent different updating approaches

Label	Approach
Build	Rebuild the whole SB-Tree
PRA	Periodic Rebuild Approach
Batch Updating	
STU	Sub-Tree Updating Approach
STUR	Sub-Tree Updating Approach with Rebuild
Point-by-point Updating Approach	
GUA	Guarantee Updating Approach
AUA	Approximate Updating Approach

4.1 Precision

The SB-Tree updated to have *m* data points will be compared with a newly created SB-Tree for the *m* new data points. Precision is evaluated by calculating the percentage of correct PIP retrieved. Starting from the 10 data points of the time series, comparison is carried out by adding each point until the remaining 2490 data points were added. By retrieving 40% of the PIPs from the updated SB-Tree and the newly created SB-Tree according to the access method mentioned in section 2.3, the percentage of correct PIPs retrieved is calculated by counting the number of PIPs retrieved from the newly created SB-Tree could also be found in PIPs retrieved from the updated SB-Tree. To simplify the calculation, if the third PIP retrieved from the updated SB-Tree, for example, is the same as the third PIP retrieved from newly created SB-Tree, it is considered as correct.

In Fig.16, the percentage of correct data points retrieved by using the periodic updating approach (PRA) is shown. The line is very fluctuant since the precision decreases as points keep adding to the tree, but it increases to 100% once tree rebuild occurred. Fig.17 shows the precision achieved by sub-tree updating (STU). The precision of the point retrieved keeps decreasing as more points are added to the tree. As expected, the precision would not increase unless rebuild tree mechanism is introduced. This updating method will suffer in multi-resolution visualization since it does not preserve the shape of the time series. The precision of the approximate updating approach (AUA) is shown in Fig.18. Although this approach could not guarantee the SB-Tree structure, it still achieved pretty high precision. On average, 99% data points can be correctly retrieved.

The guarantee updating approach could guarantee the structure of the updated SB-Tree being the same as building a new SB-Tree. Therefore, both the percentage of correct points retrieved and the percentage of correct sequence is 100% throughout the updating process. On the other hand, the reason of getting error by all the other approaches (i.e. PRA, STU (STUR) and AUA) is due to the VD of some nodes in the SB-Tree is not updated. After adding a new data point, the VD of the nodes has been changed. However, those nodes not evaluated during the updating process will not be updated in these approaches. Moreover, PRA, STU (STUR) and AUA may not guarantee the structure of the updated SB-Tree being the same as rebuilding the whole tree. PRA simply adds the last data point to the right-hand-side of the rightmost node. STU only attaches the sub-tree to the original tree and AUA may not be able to detect the change of PIP as PIP identified is not only limited to those PIP candidates.

Fig.19 shows the average percentage of PIPs correctly retrieved for time series updated from 10 data points to 2500 data point by using different updating methods. Moreover, since the order of PIPs retrieved is important for multi-resolution visualization, the sequence of PIPs retrieved from the updated SB-Tree is also compared with that from the newly created SB-Tree. GUA and AUA could provide 100% or nearly 100% correct PIPs, while only GUA could also guarantee the retrieval sequence.

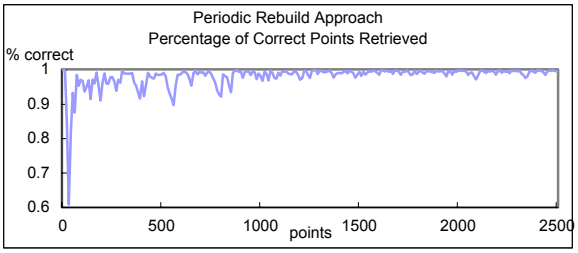

Fig.16. Precision of PRA method

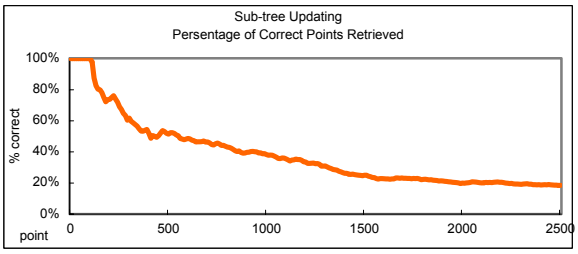

Fig.17. Precision of STU method

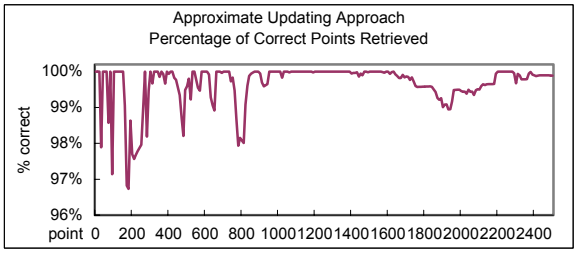

Fig.18. Precision of AUA method

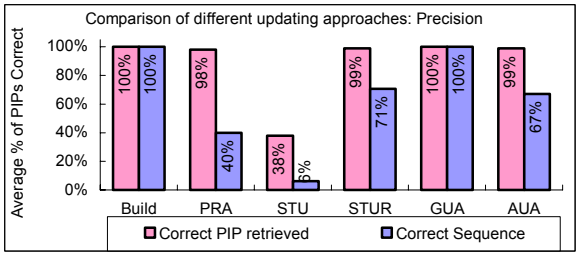

Fig.19. Percentage of correct PIP retrieved and PIP in correct sequence for different updating methods

4.2 Speed

In our second experiment, the processing time of different updating methods were compared. The total time for updating the time series data point from 10 to 2500 required nearly 30 minutes if we build a new SB-Tree for every new data point. For the periodic rebuild approach, the total time for updating required about 3 minutes. As shown in Fig.20. On average, the whole tree will be rebuilt after adding 11.5 data points. The time for rebuilding the whole tree depends on the tree size, where time increases exponentially as the number of points increases.

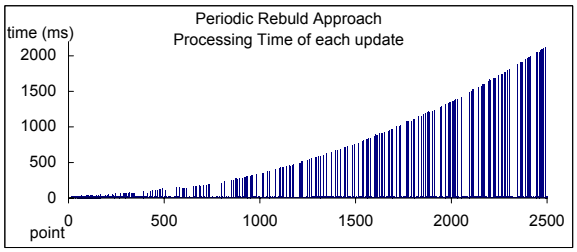

Fig.20. Processing time of updating the SB-Tree using periodic rebuild approach (PRA)

For the sub-tree updating approach, the total time for updating is about 3 seconds. On average, each updating process required only 0.013s to append a sub-tree to the SB-Tree as shown Fig.21 below.

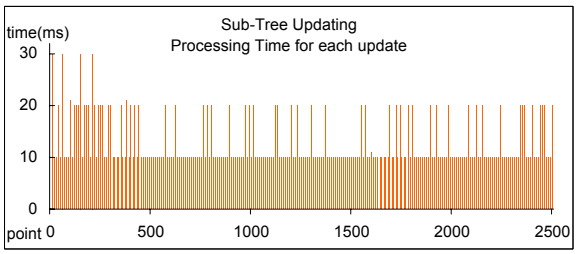

Fig.21. Processing time of updating the SB-Tree using sub-tree updating approach (STU)

For the approximate updating approach, the total time for updating is about 40 seconds. On average, about 0.016s is required for adding a data point. Moreover, about 36 data points are required to be rebuilt by adding each data point. Fig.31 compares the processing time of the updating process using AUA and the rebuild process. When updating data points 1231 and 2182, the process time of AUA is the same as rebuilding a new tree. This is because a change of PIP was detected at the real root node of the SB-Tree.

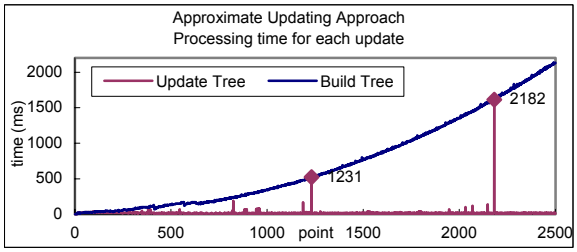

Fig.22. Processing time of updating the SB-Tree using approximate updating approach (AUA)

For the guarantee updating approach, the total time for updating is about 17.38 minutes. Fig.23 shows that the time required is over half of the time required to rebuild a new SB-Tree for every newly arrived data point. Although this updating approach could guarantee the structure of the SB-Tree, the updating process is too slow to be an efficient updating method.

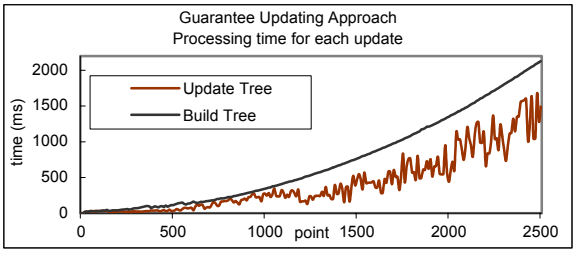

Fig.23. Processing Time of updating the SB-Tree using guarantee updating approach (GUA)

Finally, Fig.24 shows the average processing time of updating a data point for the process of updating the time series from 10 data points to 2500 data points using different updating methods. STU would update a point within the shortest time since it could add a number of points at a time. However, the advantage of STU could not be preserved when rebuild mechanism was introduced, i.e. STUR. It is because large amount of time will be consumed by rebuilding the whole tree. For the point-by-point updating methods, the performance of AUA is satisfactory and only 0.016 second was required to update a point.

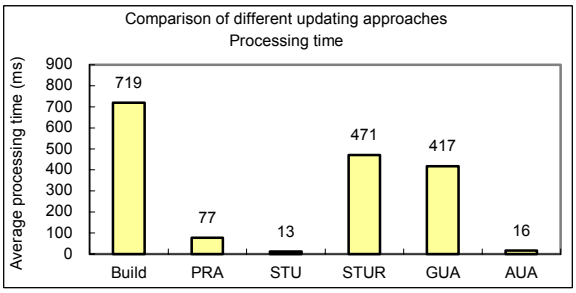

Fig.24. Processing Time of updating the SB-Tree using different updating approaches

5. Conclusions

In this paper, five updating methods, which can be grouped into three categories, i.e., periodic rebuild, batch updating and point-by-point updating, for incremental updating of SB-Tree with newly arrived data points are proposed. By applying the updating methods, the SB-Tree can be updated continuously and this is a necessary feature for representing stock time series because one of the natures of stock time series data is frequent and continuous update.

According to the experimental results, the GUA method is the preferred solution if the data point retrieval order is critical. GUA could guarantee the SB-Tree structure, so that data points could be retrieved according to their importance. It should be an attractive solution to the multi-resolution data visualization applications, e.g. progressive display of stock data on mobile devices [6]. However, GUA requires large amount of time for each updating. For those online applications which require fast update, AUA may be one of the appropriate choices. Although it could not guarantee the order of retrieval is according to their importance, its percentage of correct retrieval is nearly 100%. This means that it is capable to retrieve most of the important points of the time series. AUA is suitable for applications which require displaying most of the points of the time series and updating frequently.

If new data points always arrive in batch and those points are required to be updated to the time series within a short period of time, STU may be a better choice than the point-by-point updating approaches. Besides, STU and STUR are also suitable for applications which always display the most recently data points of the time series, since time is not wasted to maintain the "old" data.

References

[1] R.Agrawal, C.Faloutsos, and A.Swami, "Efficient similarity search in sequence databases," *Proc. of the 4th Conf. on Foundations of Data Organization and Algorithms*, 1993.

[2] K.P. Chan and A.C. Fu, "Efficient time series matching by Wavelets," *Proc. of the 15th ICDE*, pp.126-133, 1999.

[3] K.V. Kanth, D. Agrawal, A. Singh, "Dimensionality reduction for similarity searching in dynamic databases," *Proc. Of the ACM SIGMOD Conf.*, pp. 166-176, 1998.

[4] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality reduction for fast similarity search in large time series databases," *Knowledge & Information Systems*, 2000.

[5] T.C. Fu, F.L. Chung, R. Luk and C.M. Ng, "A specialized binary tree for financial time series representation," *In 10th ACM SIGKDD Workshop on Temporal Data Mining*, pp. 96-104, 2004.

[6] T.C. Fu, F.L. Chung, R. Luk and C.M. Ng, "Progressive time series visualization in a mobile environment," *In Proc. of the 9th IEEE ISCC*, vol. 2, pp. 662-667, 2004.

[7] T.C. Fu, F.L. Chung, C.F. Lam, R. Luk and C.M. Ng, "Adaptive data delivery framework for financial time Series visualization," *Accepted by the 4th ICMB*, 2005.

[8] T.C. Fu, F.L. Chung, R. Luk and C.M. Ng, "SBT-Forest, An indexing approach for specialized binary tree," *accepted by ICITA*, 2005.

[9] T.C. Fu, F.L. Chung, R. Luk and C.M. Ng, "Financial time series indexing based on low resolution clustering," *In the 4th IEEE ICDM Workshop on Temporal Data Mining: Algorithms, Theory and Applications*, pp. 5-14, 2004.

[10] F.L. Chung, T.C. Fu, R. Luk and V. Ng, "Flexible time series pattern matching based on perceptually important points," *IJCAI Workshop on Learning from Temporal and Spatial Data*, pp.1-7, 2001.

[11] D. Douglas and T. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *The Canadian Cartographer*, vol.10, no.2, pp.112-122, 1973.

[12] C.S. Perng, H. Wang, R. Zhang and D. Parker, "Landmarks: A new model for similarity-based pattern querying in time series databases," *In Proc. of the 16th ICDE*, pp.33-42, 2000.

[13] B. Pratt and E. Fink, "Search for patterns in compressed time series," *Image and Graphics*, vol.2, no.1, pp.89-106, 2002.

[14] E. Fink and B. Pratt, "Indexing of compressed time series," *Data Mining in Time Series Databases*, pp.51-78, 2003.