Similarity Range Queries in Streaming Time Series

Maria Kontaki, Apostolos N. Papadopoulos, Yannis Manolopoulos

Department of Informatics, Aristotle University
Thessaloniki, 54124, Greece

Abstract. Similarity search in time series databases is an important research direction. Several methods have been proposed in order to provide algorithms for efficient query processing in the case of static time series of fixed length. In streaming time series the similarity problem is more complex, since the dynamic nature of streaming data make these methods inappropriate. In this paper, we propose a new method to evaluate similarity range queries in streaming time series. The method is based on the use of a multidimensional access method, the R*-tree, which is used to store features of the time series extracted by means of the DFT (Discrete Fourier Transform). We take advantage of the incremental computation of the DFT and equip the R*-tree with a deferred update policy in order to improve maintenance costs. The experimental evaluation based on synthetic random walk time series and on real stock market data shows that significant performance improvement is achieved in comparison to the sequential scanning of the database.

1 Introduction

Nowadays, a significant number of applications require the manipulation of data streams [12, 2, 4, 7]. Examples of these applications are online stock analysis, computer network monitoring, network traffic management, earthquake prediction. The major common characteristic of the above applications is that they are all time-critical. Therefore, the DBMS must be equipped by effective and efficient tools for data stream processing.

An important query type that has been studied thoroughly in database literature is the similarity query. Given a query object Q the similarity query asks for all objects O_x that are similar to Q to a sufficient degree. Similarity queries have been studied for multidimensional objects, images, video, time series and other non-traditional data types. In data streams the problem is more challenging since the query object, the data or both may change over time. The similarity between two objects is expressed by means of a distance metric (e.g., Euclidean, Manhattan).

Basically, there are three similarity query types: 1) similarity range query, 2) similarity nearest-neighbor query and 3) similarity join query. In this paper, we study similarity range queries in streaming time sequences where both the query

Kontaki M., N. Papadopoulos A. and Manolopoulos Y. (2004).
Similarity Range Queries in Streaming Time Series.
In *Proceedings of the 4th International Workshop on Pattern Recognition in Information Systems*, pages 69-79
DOI: 10.5220/0002675700690079

sequence and the data sequences change over time. In this query type, given a query object Q and a distance e, the systems determines the data objects O_x that are within distance e from Q. These queries can be used on their own, or can be part of complex data mining tasks for clustering and classification [13].

The length of a streaming time series can be very large, since new values are appended. Therefore, the similarity of two time series is expressed by means of the last values of each sequence (e.g. 128, 256, 1024). Each sequence can be defined as a vector in a high-dimensional space. Dimensionality reduction techniques (e.g., DFT, KLT) can be used in order to reduce the number of dimensions, allowing efficient multidimensional access methods to be utilized. However, each vector changes over time since new values are continuously appended. The naive procedure is to delete the old vector by updating the access method, to re-apply the dimensionality reduction technique to the new vector, and to store the resulting vector in the access method. This process is very time consuming both in CPU time and disk accesses and therefore is inappropriate in our case.

In this work we use the R*-tree access method [3] to index the vectors corresponding to the time series. The dimensionality reduction technique applied to the original time series is based on an incremental computation of the DFT which avoids recomputation. Moreover, the R*-tree is equipped by a deferred update policy in order to avoid index adjustments every time a new value for a stream is available. Experiments performed on synthetic random walk time series and on real stock market data have shown that the proposed approach outperforms the sequential scanning of the database significantly.

The rest of the paper is organized as follows. In the next section we briefly discuss related work on similarity-based queries both in streams and traditional databases. In Section 3 the proposed method is presented in detail, giving emphasis to the incremental DFT computation and the deferred R*-tree update policy. The performance evaluation results are offered in Section 4. Finally, Section 5 concludes the work and raises some issues for future research in the topic.

2 Related Work

During the last years, data streams have attracted the interest of researchers. In [2, 4] a system architecture for continuous queries has been proposed and some very important issues on data streams are addressed.

Similarity queries in streaming time series have been studied in [9] where whole-match queries are investigated by using the Euclidean distance as the similarity measure. A prediction-based approach is used for query processing. The distances between the query and each data stream are calculated using the predicted values. When the actual values of the query are available, the upper and lower bound of the prediction error are calculated and the candidate set is formed using the predicted distances. The same authors have proposed another approach which uses pre-fetching [10].

Both the aforementioned research efforts examine the case of whole-match queries, where the data is static time series and the query is dynamic (changes over time). In [12] the authors present a method for query processing in streaming time series where both the query object and the data are dynamic. The Va-stream and Va$^+$-stream access methods have been proposed, which are variants of the Va-file [15]. These structures are able to generate a summarization of the data and enable the incremental update of the structure every time a new value arrives. The performance of this approach is highly dependent on the number of bits associated with each dimension.

One of the first studies in similarity queries for time sequences databases has been performed in [1]. DFT is used as the feature extraction method, and the Euclidean distance is used as the similarity measure. The DFT coefficients are stored in an R*-tree. The Euclidean distance is the most common similarity measure [8, 6, 11, 9] due to its simplicity. Transformations other than DFT have been used as well: Haar wavelet transform [6], piecewise linear representation [11], categorization [14], segmented means [16].

The contribution of our work is three-fold: a) the exploitation of the DFT transform which has been successfully applied in time series databases, b) the use of an index structure to store the transformed vectors, c) the application of a deferred update policy in order to avoid high reorganization costs for index adjustments. With this approach, multidimensional access methods can be used to handle similarity range queries in streaming time series in an effective and efficient way.

3 Proposed Method

We begin with some definitions. Stream is an infinite sequence of numbers. Streaming time series is called a finite sequence of numbers with fixed length that changes over time. For example assume that the last five prices of a stock at time $t = 0$ are: 20.32, 25.70, 28.32, 23.87, 24.27. At time $t = 1$ the new value 25.33 is arrived. The new streaming time series is: 25.70, 28.32, 23.87, 24.27, 25.33.

We assume that streams are sampled at random time intervals. Therefore the query and data streams do not have the same length. A stream is denoted by the symbol S_x and finite time series by the symbol $S_x[i : j]$, where i is the first time instance of the time series and j is the last. The number of values of a time series is therefore j-i and corresponds to a window w. $S_x(i)$ is the i-th value of the time series.

In our study, the Euclidean distance between two finite time series is used as the similarity measure. The distance between two streaming time series S_x and S_y is defined by the Euclidean distance between the last w values of S_x and S_y.

3.1 Incremental DFT Computation

The DFT is used as the feature extraction method. Real-life time series often concentrate the energy in the first few components of the DFT. Therefore we

need less information in order to capture the characteristics of the original vector. Another important feature of the DFT is that the Euclidean distance in the original and the Euclidean distance in the frequency domain are equal. By taking the first coefficients of the DFT vectors, the resulting distance between two vectors is reduced, and therefore no false dismissals occur during range query processing [1, 8].

Normally, every time a new value for a stream arrives, the DFT vector must be recalculated by using the last w values of the stream. This may lead to high costs since the recomputation of the DFT is quite expensive. However, as the following proposition explains, the computation of the DFT can be performed incrementally avoiding recomputation.

Proposition 1. *Let S be a streaming time series with values $S(0)$, $S(1)$, ..., $S(N-1)$ and length N. If a new value for this stream arrives, we get the sequence $T(1)$, $T(2)$, ..., $T(N)$. The DFT coefficients of T can be computed by the DFT coefficients of S according to the following equation:*

$$T(n) = \frac{1}{\sqrt{N}} \cdot (\sqrt{N} \cdot S(n) - S(0) + T(N)) \cdot e^{j2\pi n/N} \tag{1}$$

Proof. Note that $S(i) = T(i)$ for $1 \leq i \leq N-1$. The n-th coefficient of S is given by:

$$S(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} S(k) \cdot e^{-j2\pi kn/N} \tag{2}$$

Similarly, the n-th coefficient of T is given by:

$$T(n) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} T(k+1) \cdot e^{-j2\pi kn/N} \tag{3}$$

We begin with (1) and substitute the values of $S(n)$ as follows:

$$T(n) = \frac{1}{\sqrt{N}}(S(0) + S(1)e^{-j2\pi n/N} + ...+$$
$$+S(N-1)e^{-j2\pi(N-1)n/N} - S(0) + T(N))e^{j2\pi n/N}$$

By algebraic manipulations in the above equation and taking into consideration that $S(i) = T(i)$ for $1 \leq i \leq N-1$, and that $e^{j2\pi n/N} = e^{-j2\pi(N-1)n/N}$ we get:

$$T(n) = \frac{1}{\sqrt{N}}(T(1) + T(2)e^{-j2\pi n/N} + ...+$$
$$+T(N-1)e^{-j2\pi(N-2)n/N} + T(N)e^{-j2\pi(N-1)n/N})$$

which is exactly (3).

The above proposition can be used to incrementally compute the new DFT vector of a streaming time series, taking into account the previous one, and therefore, avoiding the recomputation.

3.2 Deferred R*-tree Update

Since the number of streams may be quite large, the use of an index structure is desirable in order to avoid the computation of the distance between the query and all the time series. We use the R*-tree as an index structure for the DFT coefficients of the streaming data series.

In our case the problem is that the DFT of data time series must be updated when a new value arrives. If we update the index every time a new value becomes available, the overhead may be prohibitive due to additional page accesses. In order to avoid continuous deletions and insertions in the R*-tree, we use a deferred update policy. A parameter D is used to control the updates. If the distance between the new and the old DFT vector exceeds the value of parameter D, then the R*-tree is updated. Otherwise, no update is performed. This technique leads to considerable saving in CPU and I/O time. The last recorder DFT vector is stored in the last disk page of every streaming time series, in order to become available when a new value arrives.

The example that is follow will clarify the method. Assume that we have a stream S and the last N values form a time series $S[k-N:k]$, where k is the position of last value of the stream. When a new value arrives, a new time series is formed $S[k+1-N:k+1]$. Let S_k and S_{k+1} be the DFT vectors of series $S[k-N:k]$ and $S[k+1-N:k+1]$ respectively. If $Euclidean(Sk+1, Sk) < D$ then S_{k+1} is stored as the most recent DFT but it is not inserted into the R*-tree. Assume that another value for the same stream arrives. $S[k+2-N:k+2]$ is the new time series and S_{k+2} is the DFT of this sequence. Assume that $Euclidean(Sk+2, Sk) < D$. Then, S_{k+2} replaces S_{k+1}, and S_{k+1} is discarded. S_{k+2} is not inserted in the R*-tree. Notice that the comparison is always between the new DFT and the DFT that has been last recorded in the R*-tree. If $D(Sk+2, Sk) \geq D$ then S_{k+2} replaces S_k in the R*-tree.

In summary, we need both the last recorded DFT vector and the previously calculated DFT vector. The first is used to decide whether an update will occur or not, and the second is used for the incremental computation of the new DFT vector.

The update of the R*-tree is performed as follows: When a new DFT is produced, an exact match query is performed in order to locate the tree leaf that where the DFT vector is stored. The new DFT replaces the old one in this leaf. Then, the MBRs of the corresponding path from the leaf to the root are adjusted accordingly. Since the coefficients of consecutive DFT vectors are similar, the overlap enlargement is not significant.

3.3 Similarity Queries

The range query algorithm has two basic steps. In the first step the R*-tree is traversed. The distances in the frequency domain between MBRs and query are computed. If the traversed node is leaf, the distances are computed using the streaming time series that falls into this node. During the traversal, MBRs or streaming time series that their distance is more than $e + D$, are pruned. Recall

that a new DFT vector replaces the old one in the R*-tree only if the Euclidean distance between the two vectors exceeds D. The candidate set is formed by the remaining streaming time series. In the second step, using the candidate set, the real streaming time series are retrieved from the disk. The distances between them and the query are computed. The streams with distances more than e are discarded. Figure 1 outlines the algorithmic form of the range search for the R*-tree.

```
Algorithm RangeSearch(Node, Query, e, D, Candidates)
1.  if Node.type == LEAF
2.      for i=1 to Node.ReservedEntries
3.          dist = DFTEuclideanDIST(Query, Node.entries[i])
4.          if dist < (e + D)
5.              Candidates = Candidates ⋃ Node.entries[i]
6.          endif
7.      endfor
8.  else
9.      for i=1 to Node.ReservedEntries
10.         dist = DFTEuclideanDIST (Query, Node.entries[i])
11.         if dist < (e + D)
12.             RangeSearch(Node.entries[i].ptr, Query, e, Candidates)
13.         endif
14.     endfor
15. endif
16. end
```

Fig. 1. Range search algorithm.

4 Performance Evaluation

In this section, we present the experimental results in order to evaluate the performance of the proposed technique. The algorithms have been implemented in C++ and the experiments have been conducted on a Pentium IV Workstation with 512MB memory, running Windows 2000.

The database is composed of 50,000 streams, and the length of each stream is set to 20,000 values. The data are produced by means of a random-walk process. Each stream is generated as $S_x(i) = 100 * (sin(0.1 * RW(i)) + 1 + i/20000)$, $0 \leq i \leq 19999$, where RW is a random walk series of 20,000 values. We assume a page size of 4KB both for data and the R*-tree index. For each diagram we use the name **DFT** for the proposed method and the **SS** for the sequential scanning of the database. In addition, we performed experiments with 50,000 real time sequences representing stock variations obtained from http://finance.yahoo.com.

We study the performance of a query with respect to e (maximum similarity distance), D (minimum update distance), window size w, number of DFT coefficients and number of streams. Due to space limitation only the most representative results are presented.

In the first experiment we study the performance of similarity range query processing with respect to the parameter e. The values of the other parameters are as follows: a) the window size is $w = 256$, b) only the first two coefficients of the DFT are considered, c) the minimum update distance is $D = 100$, d) the user specified distance e is 1550 (for the other experiments) and e) the number of streams is 50000. This holds for all experiments unless other values are specified explicitly. The parameter e ranges between 1500 and 2200. We select this range of values, because for e=1500 only a few time series are selected, whereas for e=2200 almost half of the number of streams are contained in the answer set.

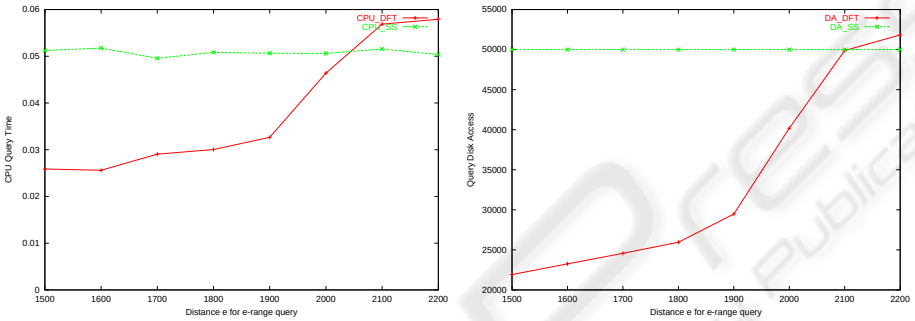
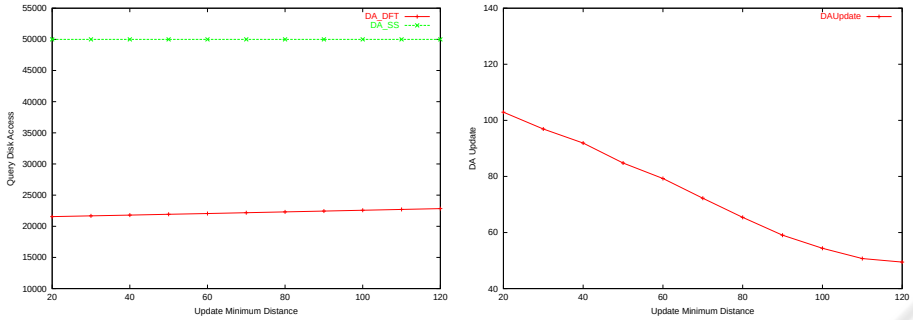


Fig. 2. CPU time and number of disk accesses vs query range e.

The results of the first experiment are depicted in Fig. 2. The proposed method clearly outperforms the sequential scanning. We note that a user usually is interested in relatively small values of e in order to obtain the most similar time series. It is observed that when e is equal to 2050 (almost half of the dataset is contained in the answer) or greater, sequential scanning shows better performance. This was anticipated, since all indexed-based methods perform better for small selectivities due to the time overhead to search the index pages.

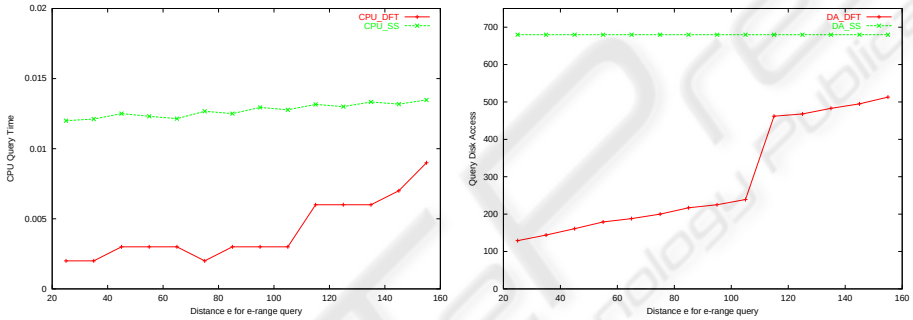
In the second experiment (Fig. 3) we examine the performance of a query with respect to the minimum update distance D. It is evident that parameter D does not affect the performance of range queries significantly. On the other hand, the D parameter affects the cost per update as Fig. 3 shows. If D is small, more updates are performed, and consequently the R*-tree quality is decreased. This means that upcoming updates require more disk accesses in order to locate the entries in the leaf level. On the other hand, large values of D introduce less updates and therefore the R*-tree quality is satisfactory. For $D = 100$ we obtain a good compromise between the number of updates and the tree quality.

By using real data sets the performance improvement is even more significant, since the DFT manages to describe more concretely the time sequences. In Fig. 4 we present the number of disk accesses and the number of candidate time sequences that are examined for different values of e. It is evident that the proposed method outperforms by factors the brute-force search method.



(a) disk accesses per query (b) disk accesses per update

Fig. 3. Disk accesses vs D.



(a) CPU time (b) disk accesses

Fig. 4. CPU time and disk accesses vs e (real dataset).

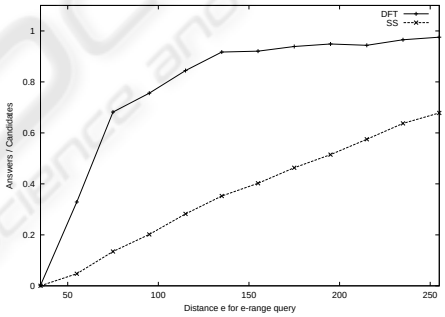
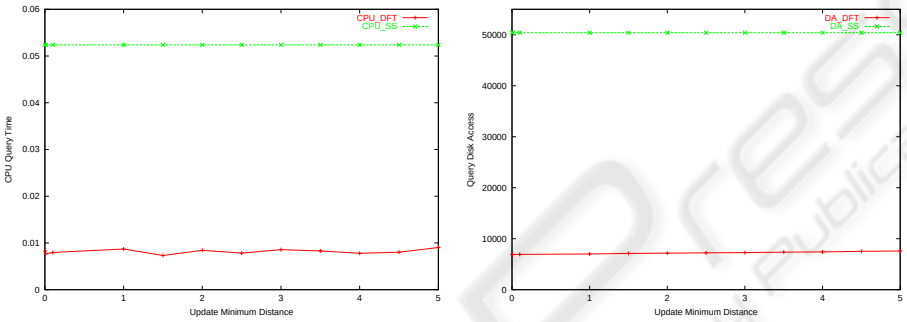


Fig. 5. Ratio of answers over candidates vs e (real dataset).

A very important performance measure is the ratio of the number of answers over the number of candidates. This ratio gives the efficiency of the method regarding the false alarms, and it is depicted in Fig. 5 for the real dataset by

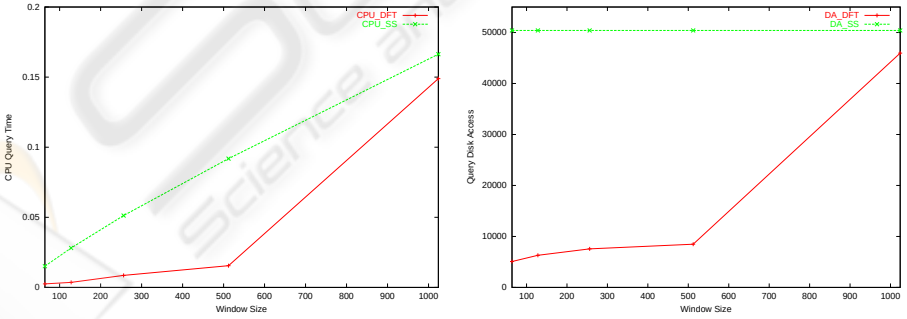
varying the query range e. It is evident that the DFT-based method manages to prune irrelevant streams and and to reduce significantly the number of false alarms.

Figure 6 shows the performance of the method when the minimum update distance (D) is varied and Fig. 7 the impact of the sliding window size. In all cases the DFT-based method performs very well and outperforms sequential scanning considerably.



(a) CPU time (b) disk accesses

Fig. 6. CPU time and disk accesses vs D (real dataset).



(a) CPU time (b) disk accesses

Fig. 7. CPU time and disk accesses vs w (real dataset).

5 Concluding Remarks

Streaming time series are used in many modern applications in order to capture the changes of a value with respect to time. In this paper we have studied the problem of streaming time series indexing in order to provide a mechanism for similarity range query processing. The proposed method uses the R*-tree as the underlying access method, equipped by an incremental computation of the DFT and a deferred update technique. The experimental results have shown that the proposed method outperforms the sequential scanning of the dataset.

Currently we study the application of the proposed method for k-nearest-neighbor queries, in order to perform comparisons with other approaches studied in [12]. Future research in the area may include: a)the selection of the minimum update distance D by means of an analytical formula in order to select the appropriate value according to the database size, the window size w and the data distribution, b)the study of the buffer impact on the performance of the methods, and c)the application of the proposed approach for similarity join queries in streaming time series.

References

1. R. Agrawal, C. Faloutsos, and A. Swami: "Efficient Similarity Search In Sequence Databases", *Proc. FODO*, pp. 69-84, Evanston, Illinois, USA, October 1993.
2. B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom: "Models and Issues in Data Stream Systems", *Proc. ACM PODS*, pp. 1-16, Madison, Wisconsin, June 2002.
3. N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger: "The R*-tree: an Efficient and Robust Access Method for Points and Rectangles", *Proc. ACM SIGMOD*, pp. 322-331, Atlantic City, NJ, May 1990.
4. S. Babu, and J. Widom: "Continuous Queries over Data Streams", *SIGMOD Record*, Vol. 30, No. 3, pp. 109-120, September 2001.
5. T. Bozkaya, N. Yazdani, and M. Ozsoyoglu: "Matching and Indexing Sequences of Different Lengths", *Proc. CIKM*, Las Vegas, NV, USA, 1997.
6. K. Chan, and A. W. Fu: "Efficient Time Series Matching by Wavelets", *Proc. IEEE ICDE*, pp. 126-133, 1999.
7. S. Chandrasekaran, and M. J. Franklin: "Streaming Queries over Streaming Data", *Proc. VLDB*, Hong Kong, China, August 2002.
8. C. Faloutsos, M. Ranganathan, and Y. Manolopoulos: "Fast Subsequence Matching in Time-Series Databases", *Proc. ACM SIGMOD*, pp. 419-429, Minneapolis, Minnesota, USA, May 1994.
9. L. Gao, and X. S. Wang: "Continually Evaluating Similarity-Based Pattern Queries on a Streaming Time Series", *Proc. ACM SIGMOD*, Madison, Wisconsin 2002.
10. L. Gao, Z. Yao, and X. S. Wang: "Evaluating Continuous Nearest Neighbor Queries for Streaming Time Series via Pre-fetching", *Proc. VLDB*, Hong Kong, China, August 2002.
11. E. J. Keogh, and M. J. Pazzani: "An Indexing Scheme for Fast Similarity Search in Large Time Series Databases", *Proc. SSDBM*, Clevelant, Ohio, 1999.
12. X. Liu, and H. Ferhatosmanoglu: "Efficient k-NN Search on Streaming Data Series". *Proc. SSTD*, Santorini, Greece, July 2003.

13. A. Nanopoulos, Y. Theodoridis and Y. Manolopoulos: "C^2P: Clustering based on Closest-Pairs", *Proc. VLDB*, pp.331-340, 2001.
14. S. Park, W. W. Chu, J. Yoon, and C. Hsu: "Efficient Searches for Similar Subsequences of Different Lengths in Sequence Databases", *Proc. IEEE ICDE*, 2000.
15. R. Weber, H.-J. Schek and S. Blott: "A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces", *Proc. VLDB*, pp.194-205, New York City, New York, August 1998.
16. B.-K. Yi, and C. Faloutsos: "Fast Time Sequence Indexing for Arbitrary Lp Norms", *Proc. VLDB*, Cairo, Egypt, 2000.
17. B. Yi, H V. Jagadish, and C. Faloutsos: "Efficient Retrieval of Similar Time Sequences Under Time Wraping", *Proc IEEE ICDE*, pp. 201-208, Orlando, Florida, February 1998.