Efficient Similarity Search in Streaming Time Sequences

M. Kontaki and A.N. Papadopoulos

Data Engineering Lab.
Department of Informatics
Aristotle University
54124 Thessaloniki, GREECE
{kontaki,apostol}@delab.csd.auth.gr

Abstract

Query processing in data streams is a very important research direction. The challenge in a database of data streams is to provide efficient algorithms and access methods for query processing, taking into consideration the fact that the database changes continuously as new data arrive. Traditional access methods that continuously update the data are considered inefficient, due to the significant update costs. In this paper we present IDC-Index, an efficient technique for similarity query processing in streaming time sequences, which is based on a multidimensional access method enhanced with a deferred update policy and an incremental computation of the Discrete Fourier Transform (DFT), which is used as a feature extraction method. The method manages to reduce the number of false alarms examined and therefore achieves high answers/candidates ratio. Moreover, an extensive performance evaluation based on synthetic random walk and real time sequences have shown that the proposed technique outperforms significantly existing approaches for similarity range query processing.

1 Introduction

Nowadays, a significant number of applications require the manipulation of data streams [BW01, BBD02, CF02, LF03, CDIM03, GMM+03]. 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 that have been extensively used in the literature:

- similarity range query,
- similarity nearest-neighbor query and
- similarity join query.

In this paper, we study similarity range queries in streaming time sequences, where both the query 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 [NTM01].

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 propose the IDC-Index (Incremental DFT Computation) which is based on the R*-tree access method [BKSS90] in order to index the vectors corresponding to the underlying time sequences. 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 time series data have shown that the proposed approach outperforms significantly the sequential scanning (SS) of the database and a recently proposed method called VA$^+$-stream [LF03], which is based on the vector approximation access method [WSB98].

The rest of the paper is organized as follows. In the next section we briefly discuss related work on similarity-based queries both in streaming and non-streaming time series databases. Moreover, we give a brief description of the VA$^+$-stream method for similarity search in streaming time sequences, and present our motivation and contribution. Our method is presented in detail in Section 3. The performance evaluation results are offered in Section 4. Finally, Section 5 concludes the work and raises some issues for future research in the area.

2 Background

2.1 Related Work

One of the first studies in similarity queries for time sequences databases has been performed in [AFS93]. 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 [FRM94, CF99, KP99, GW02] due to its simplicity. Transformations other than DFT have been used as well: Haar wavelet transform [CF99], piecewise linear representation [KP99], categorization [PCYH00, PLC99], segmented means [YF00].

During the last years, data streams have attracted the interest of researchers. In [BW01, BBD02, GKMS03] 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 [GW02] 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. Then, false alarms are discarded. The same authors have proposed two different approaches, based on pre-fetching [GYW02, GW02b].

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 [LF03] 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 variations of the VA-file [WSB98]. 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.

2.2 The VA$^+$-stream Method

In this section we briefly describe the VA$^+$-stream access method which has been proposed in [LF03] as a similarity search method in streaming time series. The VA$^+$-stream is based on VA$^+$-file [WSB98], a structure that has been proposed as an index method to overcome the dimensionality curse and to support efficient similarity search for non-uniform data.

Since a streaming time sequence contains a large number of values, similarity is expressed with respect to the W last value of the streams. This is applied to the query time sequence as well. Therefore, if $W = 256$ then each sequence is considered a point in the 256-D space, as it is illustrated in Figure 1.

The VA$^+$-stream divides the data space into 2^b cells, where b is a user-specified parameter. The VA$^+$-stream allocates different number of bits for each dimension. The sum of these bits is equal to b. Each cell is an approximation of the data points that fall into

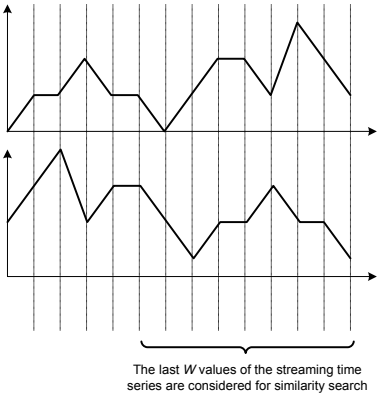
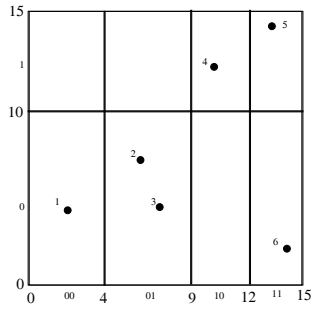


Figure 1: Example of streaming time sequences.

this cell and is represented by a bitstring of length b. An example of six time sequences in the 2-D space (W=2) is given in Figure 2.



	Data Point	Cell Vector
1	(2,4)	00 0
2	(6,6)	01 0
3	(7,4)	01 0
4	(10,12)	10 1
5	(13,14)	11 1
6	(14,2)	11 0

Figure 2: An example of VA$^+$-stream.

As new values arrive, new dimensions are created and therefore the structure must be adjusted accordingly to reflect the changes. Since the dimensionality remains constant, the left-most dimension must be eliminated, whereas a new dimension is formed, composed by the newly arrived values of the time sequences. Therefore, the data stream values considered are obtained by a sliding window which always contains the last W stream values. In order to adjust the structure due to the newly arrived values, a bit reallocation method is applied. Each dimension is quantized independently with its assigned bits, in order to achieve the least reproduction error.

The VA$^+$-stream access method can answer similarity range and nearest-neighbor queries. We focus on similarity range queries and describe shortly the query processing mechanism. A similarity range query is defined by a streaming time sequence Q and a radius e expressing the region of interest. The query is processed by using the following phases:

- In the first phase, the VA$^+$-stream is searched and the distance between the cell vectors and the query point is computed. The distance between the query point and the data point that falls into a cell cannot be smaller than the distance between the query point and the cell. Therefore the real distance is lower-bounded. If the distance of a cell is greater than e then the data points that fall into this cell are rejected. The candidate set is formed by the remaining points.
- In the second phase the candidate set is scanned and the real distance is computed, in order to discard false alarms.

2.3 Motivation and Contribution

Let us examine some important drawbacks of the VA$^+$-stream method developed in [LF03] and described in the previous section. In order for the method to be applicable, all stream values must be available at a given time instance. However, there are cases where new values are available for a portion of the data streams, whereas other data streams maintain their previous value. In such a case, the VA$^+$-stream access method can not be applied.

Another worth mentioning issue is that the VA$^+$-stream access method has very high space requirements (as we demonstrate in the performance evaluation section). A more compact structure is more appropriate, since it would be very advantageous to maintain the structure in main memory, in order to improve similarity query performance.

Finally, according to the discussion in [LF03], the number of time sequences must be known in advance, in order to calculate the number of bits per dimension. If new time sequences are inserted, the bit allocation may not reflect the data distribution adequately, requiring a possible reorganization of the structure.

The proposed method avoids the aforementioned problems. 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 which have been implemented in many

commercial DBMSs, can be used to handle similarity range queries in streaming time series in an effective and efficient way, avoiding the implementation of complex access methods. Moreover, the proposed approach can be applied to other multidimensional access methods as well (e.g., SS-tree [WJ96], SR-tree [KS97], X-tree [BKK96], TV-tree [LJF95]).

3 Proposed Method

A stream is denoted by the symbol S_x and a 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. Generally, the query and data streams do not have the same length.

Let us define the problem formally. Let us assume n streaming time series, each updated over time. To find similar streaming time series, use only the last N values of each one and update these values when a value becomes available. Given a query, find the streaming time series that are within distance e.

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, which preserves the Euclidean distance between two sequences. 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 [AFS93, FRM94].

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
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 $T(1), T(2), ..., T(N)$, where $S(i)$=$T(i)$ and $T(N)$ is the new value. The DFT coefficients of T can be computed by the DFT coefficients of S according to the following equation:

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

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

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

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

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

We begin with Equation 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 Equation 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 Update Policy

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 recorded 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 $Euclidean(S_{k+1}, S_k) < D$ 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 $Euclidean(S_{k+2}, S_k) < D$. 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 $D(S_{k+2}, S_k) \geq D$ 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 weather an update will occur or not, and the second is used for the incremental computation of the new DFT vector.

We used statistical analysis on the initial data for the choice of D. Therefore D is a static parameter. Due to space limitations we cannot explain the whole analysis. We only refer that it is based on the estimation ratio of the DFT to the real data (e.g $a = D_{DFT}/D_{real}$), where D is the Euclidean distance). It is obvious that a more sophisticated analysis can be used to have a better choice of D. For example, it is possible to change D according to the data changes over time. However, this is an issue for future work.

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.

By combining the R*-tree with the incremental DFT computation and the deferred update technique we obtain the IDC-Index.

3.3 Similarity Queries

Recall that a new DFT vector replaces the old one in the IDC-Index only if the Euclidean distance between the two vectors exceeds D. In order for similarity range queries to produce the correct results, the user-defined distance e must be expanded. This way, we guarantee that there will be no false dismissals. Therefore, the range query algorithm proceeds as follows:

1. Compute the DFT of the query time series Q,
2. Perform a range query with center Q and radius $e + D$, by using the IDC-Index,
3. Retrieve the candidate data time series, and
4. Refine the results in order to discard falls alarms.

4 Performance Evaluation

In order to evaluate the performance of the IDC-Index approach in comparison to SS and VA$^+$-stream, all methods have been implemented in C++[1]. The experiments have been conducted on a Pentium IV Workstation with 1GBytes of memory, running Windows 2000. A number of experimental series have been conducted on synthetic and real-life datasets. The datasets used are the following:

[1]We are grateful to X. Liu and H. Ferhatosmanoglu for providing us with the MATLAB code of the VA$^+$-stream.

- **SYNTHETIC**: this dataset 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.

- **STOCKS**: is the daily stock prices obtained from http://finance.yahoo.com. In order to have an adequate number of streams, we have generated some by transposing values of the real streams. The data set consists of 50,000 time sequences, and the maximum length of each one is set to 1,500.

- **TAO**: this dataset (Tropical Atmosphere Ocean) contains the sea surface temperature of 65 sites on Pacific and Atlantic Ocean since 1974, obtained from the Pacific Marine Environmental Laboratory (http://www.pmal.noaa.gov/tao). We have used the highest data resolution (e.g. the sampling time interval) that was available. About 105,000 streams form the data set, and the maximum length of each one is set to 1,200.

We study the performance of a query with respect to e (maximum similarity distance), D (minimum update distance), sliding window size W, and number of streams. We measure the CPU time and the number of disk accesses for the processing of similarity range queries. Moreover, we study the cost per update by varying the parameter D. Finally, we compare IDC-Index and VA$^+$-stream with respect to the number of candidates produced by the filter step. Due to space limitations, only the most representative results are illustrated. The parameter values used (if not otherwise specified) are: W=256, D=5 (real datasets), D=100 (synthetic dataset). In all the experimental series the first two DFT coefficients are used. CPU time is measured in seconds.

In order to evaluate the performance of the various approaches in a realistic setting, a workload was generated composed of queries intermixed with update operations. In this workload 60% of the operations are queries whereas 40% are arrivals of new values for the time sequences. These arrivals are uniformly distributed among the query operations.

Before presenting the experimental results we discuss an implementation detail regarding the use of the VA$^+$-stream access method. The VA$^+$-stream divides the data space into 2^b cells. As it is mentioned in [LF03], assigning 3 bits per dimension is a good choice. Assume that we assign 3 bits at each dimension and the window size is 256. Then, b = 3*256 = 768 and the number of produced cells is 2^{768}. However, not all of these cells are useful, since many of them do not contain any data points. In order to overcome this problem, a structure called CSET is used. This structure stores for each data point the corresponding cell. Thus, only the useful cells are used. The drawback is the size of the structure. In order to define a cell, d integers are required (see Figure 3). For each d-dimensional data point a cell is stored and therefore the CSET has n*d integers where n is the number of the real data points. To compare the size of the CSET, consider that the real data is n*d real numbers. We used the CSET structure, which was also used in [LF03] for the experimental evaluation.

	Cells	CSET
1	00 0	0 0
2	00 1	0 1
3	01 0	1 0
4	01 1	1 1
5	10 0	2 0
6	10 1	2 1
7	11 0	3 0
8	11 1	3 1

Figure 3: The CSET structure used by VA+-stream.

In the first experiment we give the performance of the methods for various values of e, which denotes the radius of the range query. Figures 4 and 5 illustrate the results for the SYNTHETIC and TAO datasets. Both the CPU time and the number of disk accesses are given. It is evident that the performance of IDC-Index is significantly better than that of SS and VA$^+$-stream. In this experiment we assumed that both IDC-Index and VA$^+$-stream access methods reside on disk. This is the reason for the large number of disk accesses posed by VA$^+$-stream. In the sequel we will assume that both the IDC-Index and the VA$^+$-stream are maintained in main memory, although this approach is more advantageous for the VA$^+$-stream because (as it is illustrated in the sequel) it requires considerably more space.

Figure 6 illustrates the space requirements of IDC-Index and VA$^+$-stream in MBytes for STOCKS and TAO datasets. It is evident that IDC-Index requires considerably less space and therefore it is more feasible to maintain the structure in main memory. By

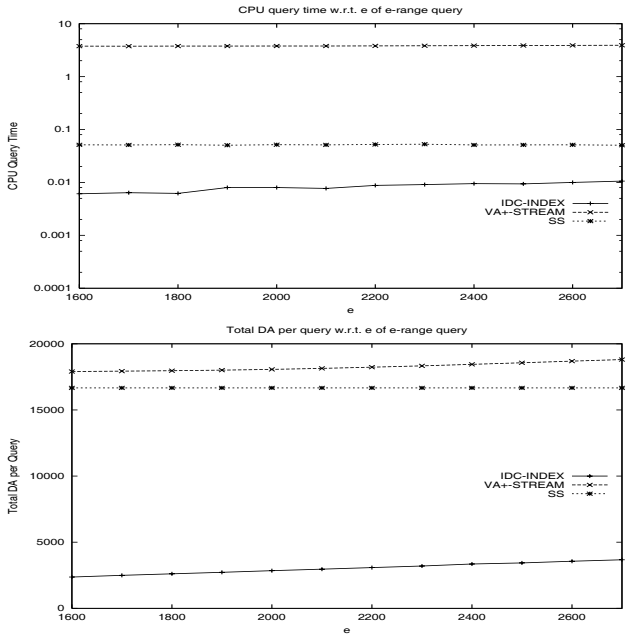


Figure 4: CPU time and number of disk accesses vs query range e for SYNTHETIC dataset.

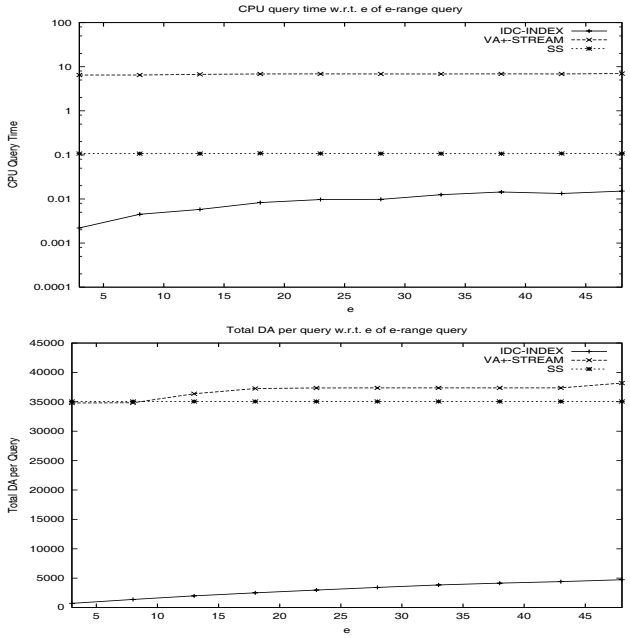


Figure 5: CPU time and number of disk accesses vs query range e for TAO dataset.

increasing the space dimensionality (sliding window length) the space requirements of IDC-Index are constant, due to the dimensionality reduction performed by the DFT. On the other hand, the space requirements of VA$^+$-stream increase linearly with respect to the number of dimensions. Similar results have been observed for the SYNTHETIC dataset.

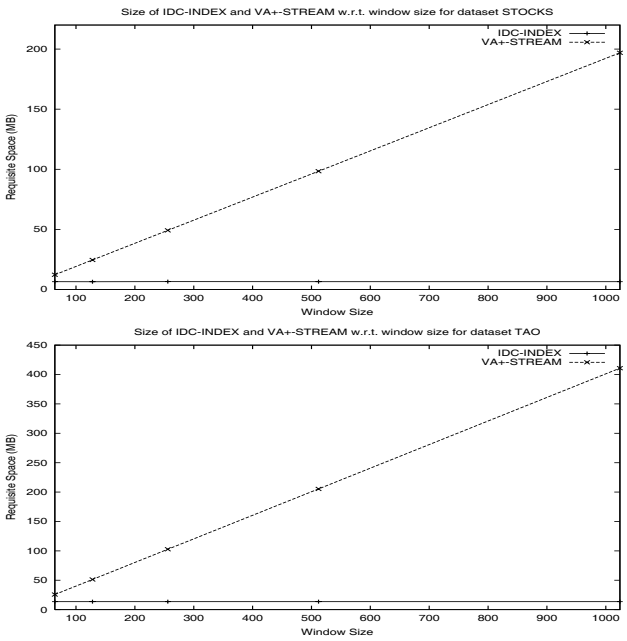


Figure 6: Index space requirements vs W for STOCKS and TAO datasets.

An important performance measure is the number of candidates that each method reports after the completion of the filter step. The number of candidates is proportional to the expected cost of the refinement step. IDC-Index is expected to retrieve more candidates by increasing the minimum update distance D. This is reasonable, since as D increases, updates in the IDC-Index are more rare, leading to small update costs, but sacrificing the accuracy of the structure. Moreover, the radius $e+D$ increases, leading to a larger search range which increases the number of candidates reported in the filter step. Figure 7 illustrates similarity search performance for IDC-Index and VA$^+$-stream by varying D. We note that the use of the parameter D is very important, since it can be used as a *tuning* parameter. If new values for the streaming time sequences arrive at a rapid rate, we can increase the value of D in order to reduce update costs. On the other hand, if the arrival rate is low, we can decrease

D in order to keep the structure updated.

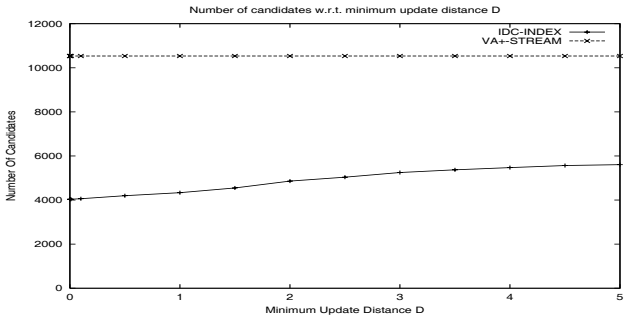


Figure 7: Number of candidates vs D for TAO dataset.

Another important measure is the ratio between the number of answers and the number of candidates. If this ratio is high, then the percentage of false alarms is low and vice-versa. Figure 8 depicts this ratio for STOCKS and TAO datasets by varying the range query radius e. It is evident that IDC-Index achieves a higher hit ratio.

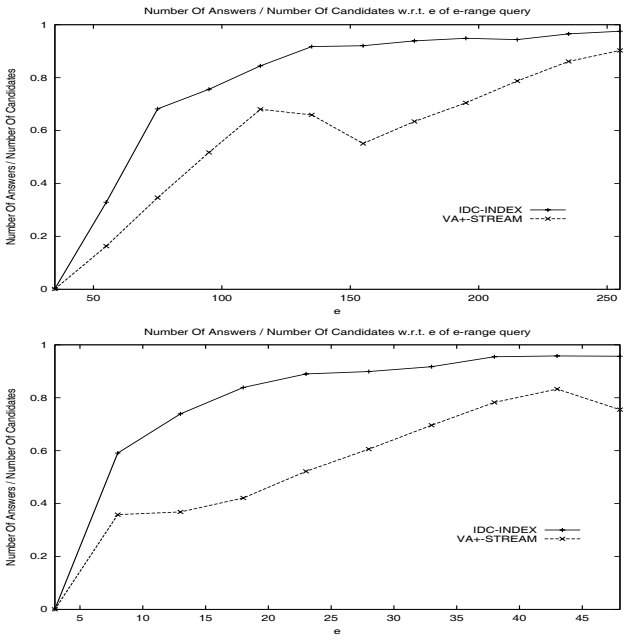
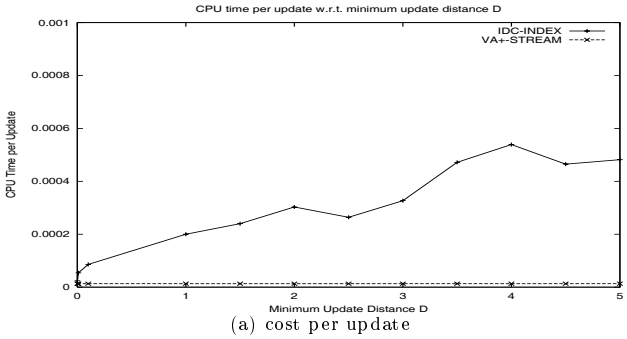
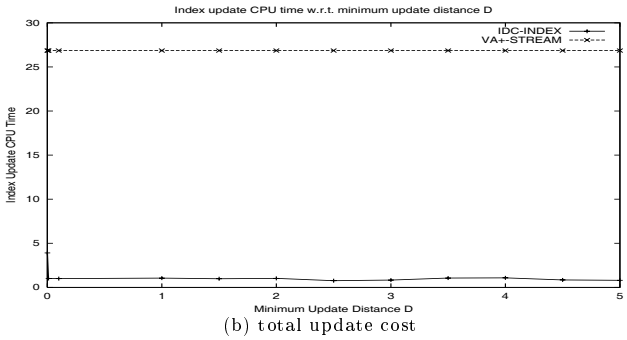


Figure 8: Hit ratio vs e for STOCKS and TAO datasets.

Another important issue is the update costs of each method. Figure 9(a) illustrates the cost for a single update. It is evident that the VA$^+$-stream method requires less time to perform a single update than IDC-Index. However, update operations are more frequent for the VA$^+$-stream than for the IDC-Index. Therefore, the total update cost for the workload is significantly low for the IDC-Index as it is illustrated in Figure 9(b).



(a) cost per update



(b) total update cost

Figure 9: Update costs for STOCKS datasets.

In Figure 10 we study the performance of the methods with respect to the sliding window size W. Recall that W affects the dimensionality of the VA$^+$-stream, and therefore it is expected that by increasing W its performance will degrade. On the other hand, IDC-Index is more immune to the increase of W, because of the dimensionality reduction performed via DFT. The CPU time and the number of disk accesses with respect to W are illustrated in Figure 10.

In the last experiment we study the performance of the methods with respect to different workloads using the TAO dataset. Each experiment consists of 1000 operations (queries and updates). Figure 11 illustrates the total CPU time and the total I/O operations for the various workloads. The IDC-Index outperforms the VA$^+$-Stream even when the update rate is high. For SS the early-stop idea is used [KP99]. Evidently the CPU time for SS is less than IDC-Index when the update ratio is more than 45% since the sequen-

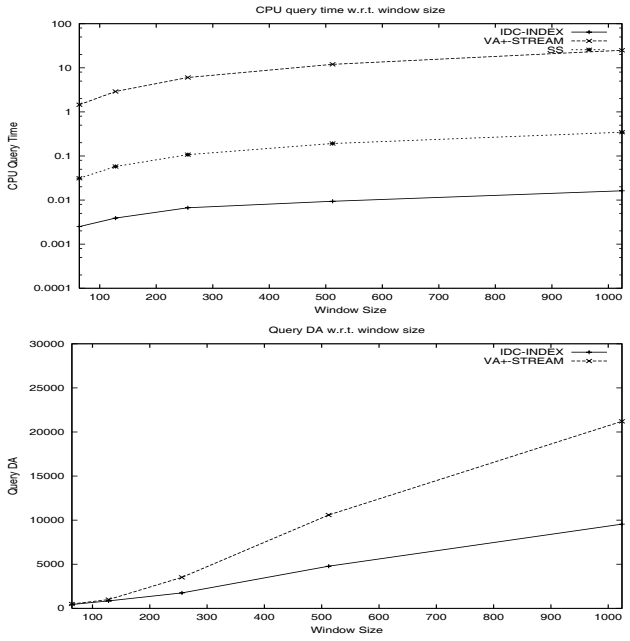


Figure 10: CPU time and disk accesses vs W for TAO dataset.

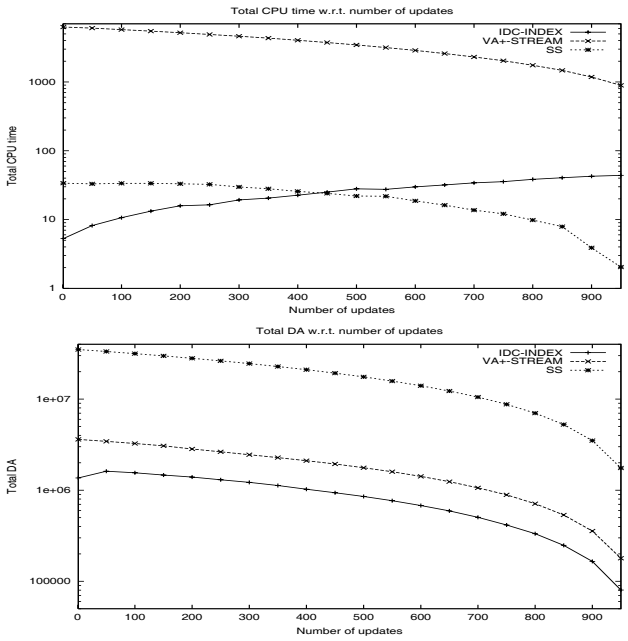


Figure 11: CPU time and disk accesses vs number of updates for TAO dataset.

tial scanning does not rely on an index, and therefore on index updates are performed. On the other hand, the number of I/O operations of IDC-Index is much smaller than SS and therefore the overall performance of IDC-Index is better.

5 Concluding Remarks

Similarity search on streaming time sequences is being considered very important due to the changing nature of data. In order to answer similarity queries efficient access methods must be employed, taking into consideration that update costs should be reduced and query response time must be satisfactory.

In this paper a new organization scheme (IDC-Index) for streaming time sequences has been proposed and studied. IDC-Index is equipped by a) an incremental computation of DFT which is used as a feature extraction method and b) a deferred update policy which is controllable and affects update costs and index performance. The performance evaluation results based on synthetic and real-life datasets have shown that considerable improvement is achieved in query response time and space requirements in comparison to sequential scanning and VA$^+$-stream. The proposed method could be easily applied to other multidimensional methods as well.

Currently we are extending this work towards nearest-neighbor query processing, whereas some interesting issues for further research are:

- the development of an analytical formula for the choice of D,
- the consideration of the incremental computation of other feature extraction methods,
- the processing of multiple continuous queries over streaming time sequences,
- the consideration of multidimensional streaming time sequences.

Acknowledgement

We are very grateful to the people at TAO Project Office for making their data available. The TAO dataset is accessible from the following URL: *http://www.pmal.noaa.gov/tao*.

References

[AFS93] R. Agrawal, C. Faloutsos, and A. Swami: "Efficient Similarity Search In Sequence Databases", *Proc. FODO*, pp. 69-84, Evanston, Illinois, USA, October 1993.

[ALSS95] R. Agrawal, K.-I. Lin, H. S. Sawhney, and K. Swim: "Fast Similarity Search in the Presence of Noise, Scaling, and Translation in Time-Series Databases", *Proc. VLDB*, Zurich, Switzerland, September 1995.

[BBD02] 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.

[BKK96] S. Berchtold, D. Keim and H.-P. Kriegel: "The X-tree: An Index Structure for High-Dimensional Data", *Proc. VLDB*, Bombay, India, 1996.

[BKSS90] 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.

[BW01] S. Babu, and J. Widom: "Continuous Queries over Data Streams", *SIGMOD Record*, Vol. 30, No. 3, pp. 109-120, September 2001.

[BYO97] T. Bozkaya, N. Yazdani, and M. Ozsoyoglu: "Matching and Indexing Sequences of Different Lengths", *Proc. CIKM*, Las Vegas, NV, USA, 1997.

[CF99] K. Chan, and A. W. Fu: "Efficient Time Series Matching by Wavelets", *Proc. IEEE ICDE*, pp. 126-133, 1999.

[CF02] S. Chandrasekaran, and M. J. Franklin: "Streaming Queries over Streaming Data", *Proc. VLDB*, Hong Kong, China, August 2002.

[CDIM03] G. Cormode, M. Datar, P. Indyk, and S. Muthukrishnan: "Comparing Data Streams Using Hamming Norms (How to Zero In)", *IEEE TKDE*, Vol. 15, No. 3, pp. 529-540, May/June 2003.

[FRM94] 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.

[GW02] L. Gao, and X. S. Wang: "Continually Evaluating Similarity-Based Pattern Queries on a Streaming Time Series", *Proc. ACM SIGMOD*, Madison, Wisconsin 2002.

[GW02b] L. Gao, and X. S. Wang: "Improving the performance of continuous queries on fast data streams: Time series case", *Proc. SIGMOD/DMKD Workshop*, Madison, Wisconsin, June 2002.

[GYW02] 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.

[GKMS03] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. J. Strauss: "One-Pass Wavelet Decompositions of Data Streams", *IEEE TKDE*, Vol. 15, No. 3, pp. 541-554, May/June 2003.

[GMM+03] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. OCallaghan: "Clustering Data Streams: Theory and Practice", *IEEE TKDE*, Vol. 15, No. 3, pp. 515-528, May/June 2003.

[KS97] N. Katayama and S. Satoh: "The SR-tree: An Index Structure for High-Dimensional Nearest Neighbor Queries", *Proc. ACM SIGMO*, pp.369-380, Tucson, AZ, 1997.

[KP99] E. J. Keogh, and M. J. Pazzani: "An Indexing Scheme for Fast Similarity Search in Large Time Series Databases", *Proc. SSDBM*, Clevelant, Ohio, 1999.

[LJF95] K. Lin, H.V. Jagadish and C. Faloutsos: "The TV-tree: An Index Structure for High Dimensional Data", *The VLDB Journal*, Vol.3, pp.517-542, 1995.

[LF03] X. Liu, and H. Ferhatosmanoglu: "Efficient k-NN Search on Streaming Data Series". *Proc. SSTD*, Santorini, Greece, July 2003.

[NTM01] A. Nanopoulos, Y. Theodoridis and Y. Manolopoulos: "C^2P: Clustering based on Closest-Pairs", *Proc. VLDB*, pp.331-340, 2001.

[PCYH00] 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.

[PLC99] S. Park, D. Lee, and W.W. Chu. "Fast Retrieval of Similar Subsequences in Long Sequence Databases", *Proc. IEEE KDEX Workshop*, Evanston, Illinois, 1999.

[WJ96] D. A. White and R. Jain,"Similarity Indexing with the SStree", *Proc. IEEE ICDE*, New Orleans, USA, pp.516-523, 1996.

[WSB98] 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.

[YF00] B.-K. Yi, and C. Faloutsos: "Fast Time Sequence Indexing for Arbitrary Lp Norms", *Proc. VLDB*, Cairo, Egypt, 2000.

[YJF98] 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.