

Using Triangle Inequality to Efficiently Process Continuous Queries on High-Dimensional Streaming Time Series

Zhengrong Yao, Like Gao, X. Sean Wang

Dept. of Information and Software Eng., George Mason University, Fairfax, Virginia
{zyao, lgao, xywang}@gmu.edu

Abstract

In many applications, it is important to quickly find, from a database of patterns, the nearest neighbors of high-dimensional query points that come into the system in a streaming form. Treating each query point as a separate one is inefficient. Consecutive query points are often neighbors in the high-dimensional space, and intermediate results in the processing of one query should help the processing of the next. This paper extends the KD tree with triangle inequality to deal with high-dimensional streaming time series. More specifically, the distances calculated for earlier query points (to patterns) are used to filter out patterns that are not possible to be the nearest neighbor of the current one. Experiments show that this extension works well.

1 Introduction

Many applications deal with high-dimensional data such as hyperspectral data, histograms of images or data sampled by sensor arrays. The data are generally coming from an outside environment, and the system needs to “reason about” the incoming data. A common technology is to use a database of patterns that are of the same structure as the incoming data. The system finds the pattern from the database that matches most closely to the incoming data, and thus derives useful information for quick responses.

When the high-dimensional data come to the query processing system continuously in forms of streams, which we call *high-dimensional streaming time series*, the challenge to process these queries in real time is two-fold. First, it is costly, in terms of the CPU time and I/O access, to find the nearest neighbor from a large pattern database. Secondly, the data in the stream may arrive fast and the time allowed to process each query is relatively short. Thus, how to efficiently find the nearest neighbor in a stream is the key issue to achieve the goal of processing on the fly.

Commonly used approaches for the nearest neighbor search with high dimensional data are partitioning algorithms. In these methods, each pattern is treated as a point in the high-dimensional space and all patterns are partitioned

into hierarchical structures. Among them, KD tree has been seen as an optimal choice for memory based data [4, 8]. Another type of partitioning algorithms that are based on R-tree, SR-tree [6], e.g., are commonly used for the disk based data. A common problem of these index based approaches is that when the data dimensionality is higher, such as greater than 20, the performance drops so quickly that these methods become worse than the sequential scan.

In many cases, the consecutive query points in the stream may somewhat related. In each dimension, the values should not “jump” too big and too often in a gradually changing environment. Consecutive query points are neighbors not only in time, but also in space. We call this the “*continuity*” of streaming time series. In this paper, we propose a solution which combines the traditional algorithms with the “triangle inequality” to exploit the continuity for better query performance.

Intuitively, once a distance between a pattern and an earlier query point is calculated, we can save this distance for reuse when the next query point is processed. If the two query points are close enough, a distance lower-bound between the second query point and the pattern can be given by using “triangle inequality”. Therefore, some patterns can be filtered out quickly and the times of the real distance evaluation can be reduced a lot.

In this paper, we focus on the memory based nearest neighbor query. Extension of the investigated methods to disk based queries should be interesting but beyond the scope of this paper. We choose the KD tree and sequential scan methods as benchmark algorithms. By enhancing the KD tree with the triangle inequality strategy, we obtain a new algorithm for streaming time series, which we call the KD+TR algorithm. Our experiments show that this new algorithm can provide very good improvement over the traditional methods for streaming time series with strong continuity.

This paper deals with the continuous queries on high-dimensional streaming time series and thus falls into the area of data stream processing and management. Recently, there are special interest in this area due to the emerging new applications. Generally, the studies on data streams are classified into four categories: (a) traditional database

operations on data streams [2], (b) query plan optimization [9, 7, 9], (c) data stream mining [11, 5] and (d) data stream architectures and framework [3, 10]. More detail about related issues can be found in the tutorial [1].

The rest of this paper is organized as follows. Section 2 defines the related concepts and terms of streaming time series. Section 3 describes our algorithm. Section 4 shows the results of the experiments, and finally Section 5 concludes the paper and addresses our future work.

2 Problem Definition

Definition Given an integer $d > 0$, a sequence of d real values is called a *high-dimensional point* with d being the *dimensionality* of this point.

We call the high-dimensional points stored in a *database* \mathcal{P} as *patterns*, and we use P_i to denote the i^{th} pattern in the database. In this paper, we assume that all patterns in \mathcal{P} have the same dimensionality of d .

Definition A time sequence of high-dimensional points is called a *high-dimensional streaming time series*.

We assume that the data points in the stream arrive at times t_0, t_1, t_2, \dots . We use S_i to denote the high-dimensional point that arrives at time t_i and thus $S_i = \langle S_i^1, S_i^2, \dots, S_i^d \rangle$, where each S_i^j is a real value. We also assume that all S_i has the same dimensionality of d as the patterns in the database \mathcal{P} . The high-dimensional streaming time series is denoted as S and $S = \langle S_0, S_1, \dots, S_i, \dots \rangle$.

Definition Given a pattern database \mathcal{P} , a distance measure \overline{xy} and a high-dimensional streaming time series $S = \langle S_0, S_1, \dots, S_i, \dots \rangle$, the *nearest neighbors* of S is an infinite sequence $N = \langle P_{n_0}, P_{n_1}, \dots, P_{n_i}, \dots \rangle$, where P_{n_i} is in \mathcal{P} with the condition that $\overline{P_{n_i} S_i} < \overline{P_k S_i}$ for all $k \neq n_i$. Each S_i is thus also called a *query point*.

Therefore, the nearest neighbors of a high-dimensional streaming time series $S = \langle S_0, S_1, \dots \rangle$ are the nearest neighbors in the pattern database \mathcal{P} of all S_i . Here, we assume there are no ties for the nearest neighbor of each S_i . Otherwise, one of the tied patterns will be randomly chosen.

Definition The *continuous query* studied in this paper is to find the nearest neighbors of a high-dimensional streaming time series from a pattern database.

In general, the distance function in the above definition can be any that is meaningful for an application. However, in order to use the triangle inequality strategy of this paper, the distance function d must satisfy the following triangle inequality condition: $\overline{a_i a_j} \leq \overline{a_i a_k} + \overline{a_k a_j}$ for all high-dimensional points a_i, a_j and a_k . In this paper, we will

exclusively use the Euclidean distance which satisfies the triangle inequality condition.

we define the *continuity* as follows.

Definition For a pattern database \mathcal{P} and a streaming time series S , the *continuity* of S on database \mathcal{P} , within a time period, is the percentage of cases when two successive query points share the same nearest neighbor.

Note the continuity of a streaming time series is not a constant, rather it will change from time to time. However, the continuity value is always between 0 and 1, inclusively. In our experiments, since we are dealing with finite streams, we will use the averaged continuity over the entire (finite) stream.

3 Algorithms

A straightforward use of traditional algorithms for our continuous queries is to process each query point independently. Since KD tree can be used for its good performance in relatively low-dimensional data for memory based queries, we choose it as the benchmark index method in this paper. In very high-dimensional cases, sequential scan method has better performance and thus can be used as another benchmark method. More detail about the sequential scan and KD tree can be found in [4].

The basic idea of using the continuity is as follows. Suppose when query point S_k was processed, the distance between S_k and a pattern P was calculated. When a subsequent query point S_i arrives and if S_i is quite close to S_k , then we can use the “triangle inequality” and the already calculated distance between S_k and P to check whether P may be the nearest neighbor of S_i or not. The distance $\overline{PS_k}$ is known when S_i comes. The triangular property of the Euclidean distance determines

$$\overline{PS_i} \geq \overline{PS_k} - \overline{S_k S_i}. \quad (1)$$

If $\overline{PS_k} - \overline{S_k S_i} > h$, where h is the current minimum distance to S_i , then $\overline{PS_i} \geq h$ and hence P cannot be the nearest neighbor of S_i . In high dimensional space, distance calculation can be very costly and this triangle inequality can save us from the trouble of calculating many distances by using previously calculated distance $\overline{PS_k}$. Note that $\overline{S_k S_i}$ is not related to patterns, and when query point S_i comes, it is relatively less costly to calculate $\overline{S_k S_i}$ for a few selected S_k . Also note that if only partial distance between S_k and P was calculated, we can still use the above strategy.

We extend the KD tree with triangle inequality. We call it the KD+TR algorithm. Figure 1 shows the data structure of terminal nodes for the KD+TR algorithm. We associate with the node the latest query point that requires a scan on the node and the distance lower bound of this query point to all the patterns in the node. For each pattern in the node,

we associate its distance to a query point (which is the last one whose distance to this pattern is calculated), and the query point ID. Before the patterns in a terminal node are examined, we use triangle inequality as follows. First, the query point and associated distance lower bound are used to determine if any of the patterns in the node needs to be examined with respect to the current minimum distance. If the answer is positive, we go into the node and examine each pattern. Again, triangle inequality is used. The new distance calculations will replace older ones, and the query point associated with the node will be replaced by S_i . The smallest distance found in this scan is stored as the distance lower bound of S_i to all the patterns in the node.

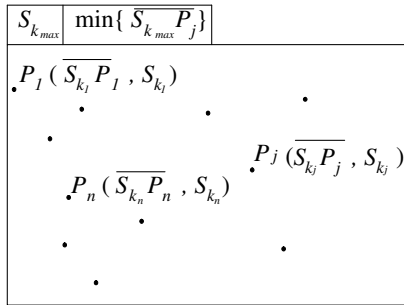


Figure 1. Terminal node of KD+TR algorithms

In summary, the major steps of KD+TR algorithm are given in Figure 2.

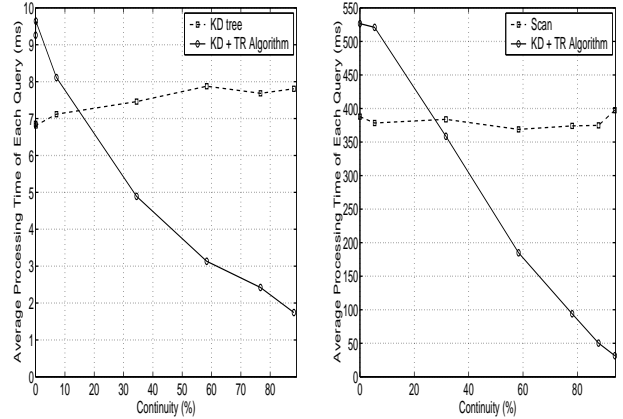
1. If the node is a terminal node:
 - 1.1 Use triangle inequality at the terminal node.
 - 1.2 If terminal node satisfies triangle inequality, use it at pattern level and find the candidate patterns in the node.
 - 1.3 Examine all candidate patterns P_i , record their distances and the query IDs
 - 1.4 Find the nearest neighbor of the query point inside the node, record its distance to terminal node. If this distance is less than the current minimum one, update the current nearest neighbor and the minimum distance.
2. if the node is a nonterminal node: same as traditional KD tree.

Figure 2. The KD+TR algorithm

4 Experimental Results

To assess the performance of KD+TR algorithm, we performed the experiments with a synthetic data set and a real

one. The patterns and query points in the synthetic data set were generated by a random process. We generated pattern sets of 4, 10, 20 and 80 dimensions respectively. The real data set came from the Digital Spectral Library (1993) of U.S. Geological Survey which are of 436 dimensions. For each pair of successive query points in the initial query stream, we generated a middle query point by using the averaged value on each dimension. After inserting this middle query point into the stream, we generated a new streaming time series with different continuity. We create several groups stream queries whose continuity from 0% to about 90%.



(a) Dimensionality=10 (b) Dimensionality=80

Figure 3. Effect of continuity (synthetic data, number of patterns=100,000)

The experimental results with the synthetic data are given in Figure 3. Figure 3.(a) compares the performances of the KD tree and KD+TR algorithm. Figure 3.(b) shows the comparison of the KD+TR algorithm with the sequential scan in the 80-dimensional case. As can be seen, the cost of the KD tree algorithm (or sequential scan) does not change very much. But the cost of KD+TR algorithm decreases very quickly as the continuity increases. Since the KD+TR algorithm uses full distance calculation, there is a cross-over point at which the KD+TR algorithm performs well over the KD tree algorithm (or sequential scan). In the database with 100,000 10-dimensional patterns, the KD tree algorithm needs to verify about 10% of the patterns. But once we introduced triangle inequality, the verification drops from 8% (continuity=0) to 0.8% (continuity=88%). This basically is where the performance improvement of the KD+TR algorithm comes from. Continuity also affects query processing when dimensionality is rather high.

Similar results are seen in the experiments with the real data set, as shown in Figure 4. Since the real data have a

very high dimensionality of 436, we need to pre-process the data in order to use KD tree and KD+TR methods. With some mathematical transforms, i.e., Fourier Transform, we can concentrate the most significant values of the high dimensional points in a few dimensions where the distances among points in the transformed space are preserved. Therefore, even the data dimensionality is high, the KD tree and KD+TR algorithms can still work better than the sequential scan method. In Figure 4, again, the KD+TR algorithm outperforms the KD tree method as continuity increases.

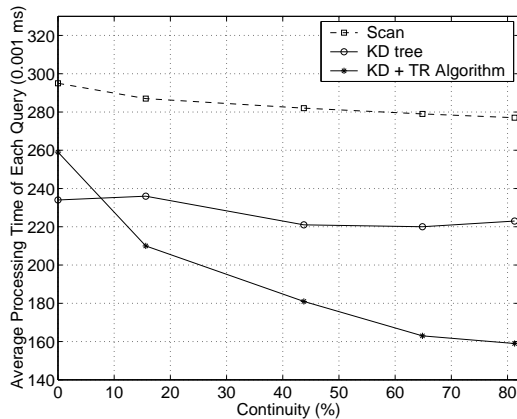


Figure 4. Effect of continuity (real data, data set size=480, dimensionality=436)

5 Conclusion

The continuity of streaming time series makes the neighboring query points share a lot of information. Triangle inequality with such information can reduce the distance evaluation time, which improves the performance of continuous queries. In this paper, based on the traditional algorithm and triangle inequality, we proposed the KD+TR algorithm for continuous queries. We use experiments to compare the new algorithm with the traditional one.

Although we only implemented the KD+TR algorithm for memory based query to show how to combine triangle inequality with traditional algorithm like KD tree algorithm, it should be interesting to extend triangle inequality with other index structures, such as R-tree and X-tree. In addition, in some applications, there may not have strong continuity among the incoming query points. For such cases, how to improve the query performance needs further investigation.

References

- [1] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and issues in data stream systems. In *Proc. of PODS*, 2002.
- [2] S. Babu and J. Widom. Continuous queries over data streams. In *SIGMOD Record*, Sept., 2001.
- [3] J. Chen, D. J. DeWitt, F. Tian, and Y. Wang. NiagaraCQ: a scalable continuous query system for Internet databases. In *Proc. of SIGMOD*, 2000.
- [4] Jerome H. Friedman, J. Bentely, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3), 1977.
- [5] L. Gao and X. S. Wang. Continually evaluating similarity-based pattern queries on a streaming time series. In *Proc. of SIGMOD*, 2002.
- [6] Gisli R. Hjaltason and Hanan Samet. Ranking in spatial databases. In *Symposium on Large Spatial Databases*, pages 83–95, 1995.
- [7] Samuel Madden, Mehul Shah, Joseph M. Hellerstein, and Vijayshankar Raman. Continuously adaptive continuous queries over streams. In *Proc. of SIGMOD*, 2002.
- [8] V. Ramasubramanian and K.K. Paliwal. Fast k-dimensional tree algorithms for nearest neighbor search with applications to vector quantization encoding. *IEEE Transactions on Signal Processing*, 40(3), 1995.
- [9] S. D. Viglas and Jeffrey F. Naughton. Rate-based query optimization for streaming information sources. In *Proc. of SIGMOD*, 2002.
- [10] Stan Zdonik, Ugur Cetintemel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Donald Carney. Monitoring streams - a new class of data management applications. In *Proc. of VLDB*, 2002.
- [11] Yunyue Zhu and Dennis Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *Proc. of VLDB*, 2002.