INTRODUCTION

In many application domains, data can be represented as a series of values (time series). Examples include stocks, seismic signals, audio, and many more. 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. Research in this field has focused on the development of effective transformation techniques, the application of dimensionality reduction methods, and the design of efficient indexing schemes. These tools enable the process of similarity queries in time series databases. In the case where time series are continuously updated with new values (streaming time series), the similarity problem becomes even more difficult to solve, since we must take into consideration the new values of the series. The dynamic nature of streaming time series makes the methods proposed for the static case inappropriate. To attack the problem, significant research has been performed towards the development of effective and efficient methods for streaming time series processing. In this paper, we introduce the most important issues concerning similarity search in static and streaming time series databases, presenting fundamental concepts and techniques that have been proposed by the research community.

BACKGROUND

Time series are used in a broad range of applications, modeling data that change over time. For example, stock price changes, audio signals, seismic signals, and electrocardiograms, can be represented as time series data. In fact, any measurement that changes over time can be represented as a time series. Two simple time series examples are depicted in Figure 1.

We differentiate between two types of time series, namely: (1) static time series and (2) streaming time series. In the first case, we assume that the time series is composed of a finite number of sample values, whereas in the second case, the size of the series is increasing since new values are appended. For example, if the data correspond to stock prices for the year 2004, then we can use static time series to capture the stock prices of the time period of interest. On the other hand, if there is a need for continuous stock monitoring as time progresses, then streaming time series are more appropriate.

Streaming time series is a special case of streaming data, which nowadays are considered very important, and there is an increasing research interest in the area. Traditional database methods cannot be applied directly to data streams. Therefore, new techniques and algorithms are required in order to guarantee efficient and effective query processing in terms of the CPU time and the number of queries processed.
of disk accesses. The most important difficulty that these techniques must address is the continuous change, which poses serious restrictions.

The purpose of a time series database is to organize the time series in such a way that user queries can be answered efficiently. Although user queries may vary according to the application, there are some fundamental query types that are supported:

- **whole-match queries**, where all time series have the same length; and
- **subsequence-match queries**, where the user’s time series is smaller than the time series in the database, and therefore, we are interested in time series which contain the user’s time series.

In contrast to traditional database systems, time series databases may contain erroneous or noisy data. This means that the probability that two time series have exactly the same values in the same time instances is very small. In such a case, *exact search* is not very useful, and therefore, *similarity search* is more appropriate. There are three basic types of similarity queries:

- **Similarity range query**: given a user time series \( Q \) and a distance \( e \), this query retrieves all time series that are within distance \( e \) from \( Q \).
- **Similarity nearest-neighbor query**: given a user time series \( Q \) and an integer \( k \), this query retrieves the \( k \) series that are closer to \( Q \).
- **Similarity join query**: given two sets of time series \( U \), \( V \) and a distance \( e \), this query retrieves all pairs \( (u,v) \) \( u \in U \) and \( v \in V \) such that the distance between \( u \) and \( v \) is less or equal to \( e \).

It is evident from the above definitions that in order to express similarity between two time series objects, a distance measure \( D \) is required. This distance measure usually ranges between 0 and 1. If two time series \( u \) and \( v \) are similar, then the value \( D(u,v) \) should be close to 1, whereas if they are dissimilar, then \( D(u,v) \) should be close to 0. Similarity search can be applied for whole-match queries and subsequence-match queries as well, for static or streaming time series.

**SIMILARITY SEARCH IN TIME SERIES**

We begin our study with methods proposed for static time series. Streaming time series are considered later in this section. The efficient processing of similarity queries requires the addressing of the following important issues:

- the definition of a meaningful distance measure \( D \) in order to express the similarity between two time series objects,
- the efficient representation of time series data, and
- the application of an appropriate indexing scheme in order to quickly discard database objects that cannot contribute to the final answer.

Assuming that each time series has a length of \( m \), then it is natural to think that each time series is represented as a vector in the \( m \)-dimensional space. In such a case, the similarity between two time series \( u \) and \( v \) can be expressed as the Euclidean distance:

\[
D(u,v) = \sqrt{\sum_{i=1}^{m} (u[i] - v[i])^2}
\]

where \( u[i], v[i] \) is the value of \( u \) and \( v \) for the \( i \)-th time instance. The Euclidean distance has been widely used as a similarity measure in time series literature (Agrawal,
Related Content

Issues in Transaction-Time Temporal Object Database Systems
[www.igi-global.com/article/issues-transaction-time-temporal-object/3271?camid=4v1a](www.igi-global.com/article/issues-transaction-time-temporal-object/3271?camid=4v1a)

An Efficient Concurrency Control Algorithm for High-Dimensional Index Structures
Seok Il Song and Jae Soo Yoo (2004). *Journal of Database Management* (pp. 60-72).
[www.igi-global.com/article/efficient-concurrency-control-algorithm-high/3316?camid=4v1a](www.igi-global.com/article/efficient-concurrency-control-algorithm-high/3316?camid=4v1a)

On the Adaptation of an Agile Information Systems Development Method
[www.igi-global.com/article/adaptation-agile-information-systems-development/3340?camid=4v1a](www.igi-global.com/article/adaptation-agile-information-systems-development/3340?camid=4v1a)

Engineering Information Modeling in Databases
[www.igi-global.com/chapter/engineering-information-modeling-databases/11149?camid=4v1a](www.igi-global.com/chapter/engineering-information-modeling-databases/11149?camid=4v1a)