Time Series Indexing Taking Advantage of the Generalized Suffix Tree

Daniel Yoshinobu Takada Chino, Felipe Alves da Louza, Agma Juci Machado Traina,
Cristina Dutra de Aguiar Ciferri, Caetano Traina Júnior

Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo
{chinodyt, louza, agma, cdac, caetano}@icmc.usp.br

Abstract. A time series is a collection of observations sequentially taken over time. Time series appear in several application areas such as finance, marketing, agriculture, weather, industrial and scientific data gathering. Similarity searching on time series databases is an important asset to extract knowledge from them. In this article, we propose Telesto, a novel indexing approach aimed at performing similarity search over time series, which is based on discretized time series and generalized suffix trees. Initially, Telesto discretizes time series and represents them as strings, using as a basis the Symbolic Aggregate Approximation (SAX) technique. Thereafter, these strings are indexed using a generalized suffix tree. To provide both range and nearest neighbor query operations among discretized time series, Telesto extends the suffix tree substring search algorithm by calculating distances between the discretized time series. Performance tests showed that Telesto is scalable in response to increasing sizes of databases and queries, in addition to be very efficient in similarity queries over large real-world time series databases.

Categories and Subject Descriptors: Core Database Foundations and Technology [**Access methods and indexing**]: Databases

Keywords: generalized suffix tree, indexing, similarity search, time series

1. INTRODUCTION

A time series is a collection of observations sequentially obtained over time, usually in the real domain. Time series are found in several application areas, such as finance, marketing, agriculture, weather, industrial and scientific data gathering. Due to the recent technological advances and the increasing storage capacity of computers, the volume of data related to time series collected, stored and available for analysis has increased constantly and substantially. As a result, it has been increasingly important to perform search and retrieval operations efficiently over time series databases.

Similarity searching on time series databases is an important tool to extract knowledge from these databases. It is widely used as a subroutine in applications based on clustering [Kalpakis et al. 2001], classification [Geurts 2001] and mining association rules [Luo et al. 2004]. Through similarity searches, it is possible to find locations with similar behavior. For instance, through the analysis of agrometeorological data, we can find sugar-cane producing regions and regions similar to them to learn what features should be employed and nurtured to strengthen the performance of other regions [Romani et al. 2010].

The problem of performing similarity queries over time series can be divided into two classes: whole matching and subsequence matching [Das et al. 1998]. While whole matching compares complete

This work was supported by the following foment agencies: FAPESP, Microsoft Research, CNPq and CAPES. The authors also thank the CEPAGRI-UNICAMP for the set of images AVHRR / NOAA.

series with same sizes, subsequence matching compares series with different sizes. In this article, we focus on the second class of similarity queries, the subsequence matching, since it is more general and time series usually have different sizes in real applications.

Time series in the domain of real numbers can be discretized to decrease data complexity and local dispersion. In order to do so, the real domain data is converted into a discrete and low cardinality domain. Many techniques have been proposed in the literature, such as the *Discrete Fourier Transform* (DFT) [Faloutsos et al. 1994], the *Discrete Wavelet Transform* (DWT) [Chan and Fu 1999], the *Singular Value Decomposition* (SVD) [Keogh et al. 2001b], the *Adaptive Piecewise Constant Approximation* (APCA) [Geurts 2001] [Keogh et al. 2001b], the *Piecewise Aggregate Approximation* (PAA) [Keogh et al. 2001b] and the *Symbolic Aggregate Approximation* (SAX) [Lin et al. 2003], which is the most commonly used technique and, therefore, is used as a basis for our proposal.

The SAX technique is based on PAA and assumes normality of the resulting aggregated values. SAX employs an approximate distance function that lower bounds the Euclidean distance [Lin et al. 2007]. When using SAX, initially the time series must be transformed into the PAA representation and then it is discretized into a sequence of discrete symbols (i.e. string). Thus, the original problem of similarity comparison between time series can be transformed into a string comparison problem. This type of comparison can be efficiently solved using the generalized suffix tree, which extends the suffix tree to simultaneously index more than one string [Gusfield 1997].

In this article, we propose Telesto, a novel indexing technique aimed at performing similarity queries over time series. Telesto is based on discretized time series and generalized suffix trees. Initially, Telesto discretizes time series and represents them as strings using SAX. Then, these strings are indexed using a generalized suffix tree. Finally, Telesto extends the suffix tree substring search algorithm by calculating distances between the discretized time series. This extension enables the creation of range and nearest neighbor query operators that can be applied over discretized time series.

This article is organized as follows. Section 2 reviews related work that employ suffix trees for analyzing time series, while Section 3 summarizes the main concepts used as the basis for developing our work. Section 4 describes the proposed Telesto method, and Section 5 discusses performance tests. Section 6 concludes de article and also highlights future work.

2. RELATED WORK

In the literature, there are few approaches that employ suffix trees for analyzing time series. Lin et al. (2005) introduced the VizTree, a time series pattern discovery and visualization system based on suffix trees. The proposed visualization approach works by transforming the time series into strings using SAX, and encoding these strings into a modified suffix tree with properties of patterns mapped onto colors and others visual properties.

In [Rasheed et al. 2011], it was proposed a noise resilient algorithm for periodicity detection using suffix trees as the basic data structure. The algorithm calculates symbols and segment periodicity and detects the partial periodicity in time series. The presented experiments have shown that their proposal performs more efficiently when compared to other algorithms in presence of noise. That is, when noise was added, mixed or even withdrawn, the algorithm did not suffer from such conditions.

However, the aforementioned approaches differ from our work on their purpose and on the characteristics of the employed suffix tree. In detail, they focus on time series visual mining and time series behavior forecasting, whereas our objective is to find time series similar to a given one. Furthermore, they do not employ multi-scale data structures such as generalized suffix trees, making it too costly to analyze simultaneously more than one string. On the contrary, we use generalized suffix trees as a basis for our proposal. These differences motivated the development of a new access method that allows indexing time series using generalized suffix trees.

3. BACKGROUND

3.1 Time Series Similarity

Let $T = \{T_1, T_2..., T_n\}$ be a database with n time series and T_q be a time series center of a query. Given a distance function $d(T_i, T_q)$ that denotes the dissimilarity between the time series T_i and T_q, similarity queries in time series databases can be divided into two types [Das et al. 1998]:

(1) **Whole matching**, in which time series of the same size are compared (i.e. the size of T_q is equal to the time series in database $T_1, T_2..., T_n$), the distances $d(T_i, T_q)$ between the series T_q and $T_i \in T$ are calculated, and the nearest series are retrieved.
(2) **Subsequence matching**, in which time series with different sizes are compared, i.e. the size of T_q is equal or smaller than the time series in database $T_1, T_2..., T_n$. Subsequence matching can be transformed into a whole matching by using a sliding window whose size is equal to the size of the query T_q in $T_i \in T$.

The main unary similarity queries most frequently used over time series databases are the range and the nearest neighbor queries. The range query is defined as follows. Given r_q as a range distance, a range query retrieves the time series $T_i \in T$ such as $d(T_i, T_q) \leq r_q$. On the other hand, the nearest neighbor (NN) query retrieves the closest time series to T_q. The NN query is important to study similarity measures, since the effectiveness of similarities measure is directly reflected on a NN classifier [Ding et al. 2008]. In this work, we focus on the range and the NN queries using subsequence matching in time series databases.

3.2 Simbolic Aggregate Aproximation (SAX)

The SAX representation allows time series of size l to be represented by strings of arbitrary size w ($w < l$) [Lin et al. 2003]. For a given time series, SAX consists of the following steps. Firstly, the time series is normalized using z-score, so that the data have standard normal distribution [Goldin et al. 1995]. Next, the normalized time series is converted to the Piecewise Aggregate Approximation (PAA) representation, decreasing the time series dimensionality [Keogh et al. 2001a]. Lastly, the PAA representation is discretized into a string with an alphabet of size $\sigma > 2$. The size of the alphabet refers to the SAX discretization level.

Let a vector on the real numbers' domain $T_a = t_1, ..., t_l$ be a normalized time series of size l. The PAA representation of T_a can be described as the vector $\bar{T_a} = \bar{t}_1, ..., \bar{t}_w$ of size w. The i^{th} element $\bar{t}_i$ of $\bar{T_a}$ can be calculated by Equation 1.

$$\bar{t}_i = \frac{w}{l} \sum_{j=\frac{l}{w}(i-1)+1}^{\frac{l}{w}i} t_j \tag{1}$$

Figure 1 shows the PAA dimensionality reduction of the time series T_a of size l into a time series of size w, which is performed as follows. The time series is divided into w windows of equal sizes. Then, the mean value of the data in each window is calculated. Finally, the time series is represented by a vector containing the mean values calculated using Equation 1.

The PAA representation $\bar{T_a}$ is converted into a sequence of equiprobable symbols in an alphabet $\Sigma = \{\alpha_1, ..., \alpha_\sigma\}$ of cardinality σ. Normalized time series usually have Gaussian distribution, thus, we can define a list of sorted breakpoints $\beta_1, ..., \beta_{\sigma-1}$ such that the area under a $N(0, 1)$ Gaussian curve from β_i to β_{i+1} is equal to $1/\sigma$ (β_0 and β_σ are defined as $-\infty$ e ∞, respectively). Then, the time series $\bar{T_a}$ is discretized into a string $S_a = s_1 s_2 \dots s_w$. The i^{th} element s_i is defined using Equation 2.

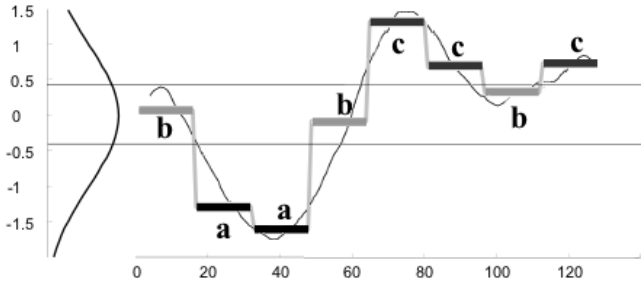


Fig. 1. A discretized time series using SAX ($l = 120$, $w = 8$ e $\sigma = 3$), adapted from [Lin et al. 2003].

$$s_i = \alpha_j, \quad \text{if} \quad \beta_{j-1} < \bar{t}_i < \beta_j \tag{2}$$

Given two discretized time series $S_a = s_1 s_2 \dots s_w$ and $S_q = s_{q_1} s_{q_2} \dots s_{q_w}$, they can be compared using the MINDIST function [Lin et al. 2003], which is an adaptation of the PAA distance [Keogh et al. 2001a] and is described in Equation 3. The $dist(\alpha_i, \alpha_j)$ function represents the distance between two symbols and can be calculated using Equation 4.

$$MINDIST(S_a, S_q) = \sqrt{\frac{l}{w}} \sqrt{\sum_{i=1}^{w} (dist(s_i, s_{q_i}))^2} \tag{3}$$

$$dist(\alpha_i, \alpha_j) = \begin{cases} 0, & \text{if } |i-j| \leq 1 \\ \beta_{max(i,j)-1} - \beta_{min(i,j)}, & \text{otherwise} \end{cases} \tag{4}$$

Since the PAA distance lowerbounds the Euclidean distance [Keogh et al. 2001a], and the MINDIST lowerbounds the PAA distance [Lin et al. 2007], by transitivity, the MINDIST lowerbounds the Euclidean distance. This proof has appeared in [Lin et al. 2007]. Therefore, we can perform similarity queries in discretized time series using the MINDIST function warranting that no false dismissals occur [Faloutsos et al. 1994].

3.3 Generalized Suffix Tree

Let $\Sigma = \{\alpha_1, \dots, \alpha_\sigma\}$ be an alphabet with σ characters. Let Σ^* be the set of all strings generated from Σ and let $S \in \Sigma^*$ be a string with m characters ($|S| = m$). Let $S[i : j]$ ($1 \leq i \leq j \leq m$) be a substring between (and including) the i^{th} and j^{th} characters of S. Let all substrings $S[1 : i]$ be prefixes of S and let all substrings $S[i : m]$ be suffixes of S, denoted by $S[i]$. Let $\$ \notin \Sigma$ be a terminal character that represents the end of a string. Finally, let $T = \{S_1, S_2..., S_n\}$ be a set with n strings.

A generalized suffix tree for the set T indexes all the strings $S_i \in T$ in a single tree with the following properties [Gusfield 1997]: (i) Each internal node, except the root, has at least two children. (ii) Each edge represents, by its label, a substring of $S_i \in T$. (iii) Two edges leaving from the same node cannot represent substrings with a common prefix. (iv) The path to a node (internal or leaf) X, denoted by $path(X)$, represents a substring formed by the concatenation of the labels on the path from the root to the node X. (v) For each leaf node L, labeled with k pairs $[X_a, Y_b]$, $path(L)$ is composed exactly of the suffixes $S_{X_a}[Y_b]$.

Figure 2 shows an example of a generalized suffix tree for the set $T = \{S_1, S_2\}$, where $S_1 = abca\$$ and $S_2 = bcaba\$$. The leaf node highlighted in red has the pairs $[1, 4]$ and $[2, 5]$, which represent the

suffixes $S_1[4] = a\$$ and $S_2[5] = a\$$, also highlighted in red, respectively (property (v)). For example, the path to the node X is equal to the substring "bca" (property (iv)).

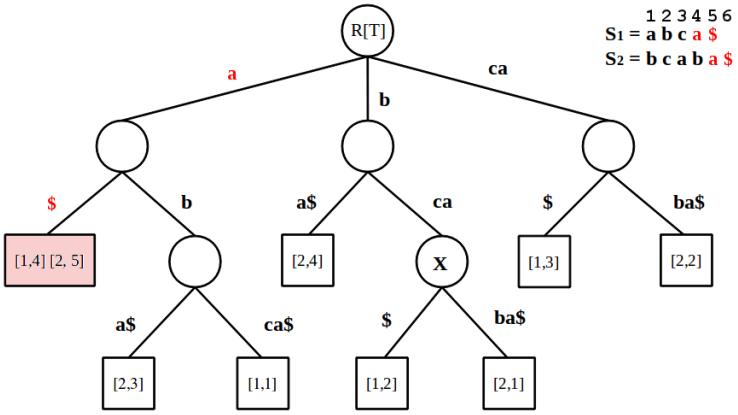


Fig. 2. Example of a generalized Suffix Tree

The trivial algorithm for the generalized suffix tree construction is described as follows [Gusfield 1997]. Initially, it creates a root node $R[T]$. Then, for each string $S_i \in T = \{S_1, S_2, \ldots S_n\}$, it adds up all the suffixes $S_i[k]$ of S_i in the tree. In order to preserve the properties of the tree, in each iteration, the suffix $S_i[k]$ is inserted into the $path(Y)$, where $path(Y)$ has the longest common prefix with $S_i[k]$, and if $path(Y) = S_i[k]$ a new leaf is inserted into the node Y, otherwise the node Y is divided. For example, the step-by-step execution of the trivial algorithm to construct the suffix tree of Figure 2 is described as follows. First, the root node $R[T]$ is created. Then the suffixes $S_1[1] = abca\$$, $S_1[2] = bca\$$ and $S_1[3] = ca\$$ are inserted into $R[T]$, since there is no path in the tree with the corresponding prefixes. Next, the suffix $S_1[4] = a\$$ is inserted into the node Y, where $path(Y) = abca\$$, since they share the same prefix "a". For this, the node Y is divided and the suffix $S_1[4] = a\$$ is inserted. The remainder suffixes are inserted in the suffix tree in the same way.

Regarding the substring search in a generalized suffix tree, it is done deterministically in linear time [Gusfield 1997]. Starting from the root node, it performs a substrings matching between the query string S_q and the substrings in the tree, finding the $path(Y) = S_q$. Then, all leaf nodes below the node Y have all occurrences of S_q in $T = \{S_1, S_2, \ldots S_n\}$. For example, the substring searching of $S_q = a\$$ in the suffix tree of Figure 2 is described as follows. The search starts from the root node $R[T]$ and finds the path $path(Y) = a$. Next, the search finds the $path(Y) = a\$$, reaching a leaf node which indicates that S_q occurs at the suffixes $S_1[4]$ and $S_2[5]$.

4. THE PROPOSED TELESTO METHOD

In this section, we describe Telesto (acronym for Time Series Generalized Suffix Tree), a novel indexing approach aimed at time series similarity queries, which is based on discretized time series and generalized suffix trees. To build and perform range and NN queries, Telesto is divided into three stages: pre-processing, construction and querying.

Figure 3 shows the pre-processing and the construction stages of the Telesto method. In the **pre-processing stage**, the time series are discretized as follows. Let $T = \{T_1, T_2, \ldots, T_n\}$ be a database with n time series. Initially, each series $T_i \in T$ is retrieved from disk (Figure 3(a)) and mapped into a string S_i using SAX (Figure 3(b)). The result of this stage is a set of discretized time series $S = \{S_1, S_2, \ldots, S_n\}$. Then, the **construction stage** executes the discretized time series indexing

using a generalized suffix tree (Figure 3(c)). The generalized suffix tree is constructed according to the trivial construction algorithm described in Section 3.3.

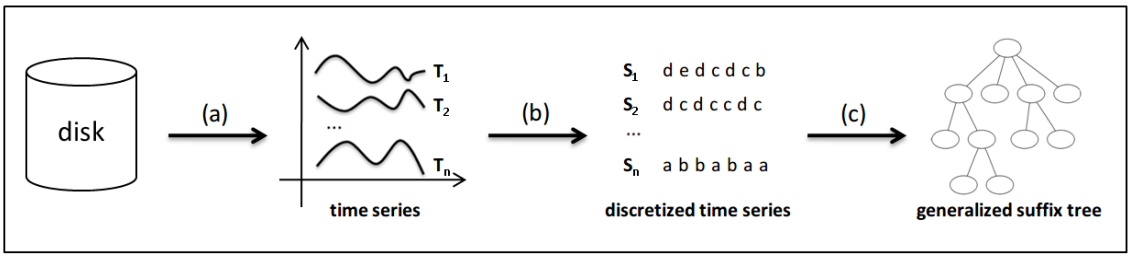


Fig. 3. Telesto: pre-processing and construction stage.

The **querying stage**, depicted in Figure 4, is responsible for performing range and NN queries between discretized time series through substrings matching. Let $T_q = t_1, t_2, \ldots, t_m$ be a time series query. In the first step, the time series T_q is mapped using SAX into a string $S_q = s_1 s_2 \ldots s_m$ (Figure 4 (a)); note that $|T_q| = |S_q| = m$. Then, Telesto performs the similarity query over the generalized suffix tree (Figure 4(b)). To execute a range query, Telesto retrieves all series $t_i \in T$ that are at a distance $d(T_q, T_i) < r_q$ (Figure 4(c)), where r_q is the query radius. In the NN query, Telesto retrieves the series closest to T_q.

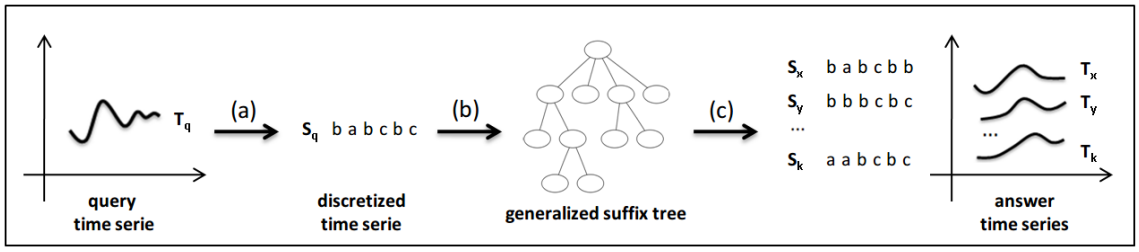


Fig. 4. Telesto: querying stage.

In order to compare time series in SAX representation, i.e. to calculate the distance $d(T_i, T_j)$ between these series, Telesto uses the *MINDIST*(S_i, S_j) distance function described in Section 3.2. The range query operation proposed extends the substring search in generalized suffix tree as follows. Let $S_q = s_1 s_2 \ldots s_m$ be a discretized time series query. The range query search for S_q finds all substrings of size m in the generalized suffix tree. During each search, for each internal node X traversed in the tree, there is a corresponding substring $path(X) = S_i[1:k]$. In the case that MINDIST$(S_i[1:k], S_q[1:k]) > r_q$, the search is stopped, i.e. a pruning occurs and $S_i[1:k]$ is discarded. In the case when the search finds a substring $S_i[1:m]$ with m characters and MINDIST$(S_i[1:m], S_q[1:m]) \leq r_q$, $S_i[1:m]$ is inserted into the answer set. The NN query operation is based on the same basic range query operation, however the query radius r_q is variable from infinity and reduces to the distance value of the string closest to S_q, as the search procedure traverses the generalized suffix tree.

Another important feature of Telesto's querying stage is that it acts as a filtering phase, since it uses the MINDIST distance, which guarantees the absence of false negatives (Section 3.2). On the other hand, false positives can occur, which creates the need for a later refinement stage. Therefore, Telesto generates a set of possible candidates. Consequently, each candidate must be tested in the original domain.

5. PERFORMANCE EVALUATION

The benefits of the Telesto method were investigated through performance tests executed using a real-world time series database. These data were extracted from a series of satellite images captured by AVHRR/ NOAA satellite (Advanced Very High Resolution Radiometer/National Oceanic and Atmospheric Administration). These series correspond to monthly measures of the Normalized Difference Vegetation Index (NDVI), which indicates the soil vegetative vigor represented in the pixels of the images [Rouse et al. 1973]. In the pre-processing stage, we used the SAX representation with discretization level equals to 5, i.e. the time series were discretized in strings of an alphabet of size 5. Each string has 108 characters and each character corresponds to a month in the extracted series.

In the performance tests, we used databases with 50, 100, 150, 200 and 250 thousand time series and 10 query groups. Each group has randomly generated queries of 5 different sizes (12, 24, 36, 48 and 60 characters), such that the queries consist of time series corresponding to 1, 2, 3, 4 and 5 years, respectively. We performed both range and NN queries. For range queries, we used the error $\epsilon = 0.005$ to calculate the query radius r_q, such that $\epsilon = r_q/|S_q|$ and $|S_q|$ is the query size.

The experiments were conducted on a computer with an Intel Core i7 2.67 GHz processor, 12 GB of main memory, 2 SATA 1 TB hard disks and Linux Ubuntu system (32 bits). To compare the range and NN queries operations, we implemented a method that performs sequential scan on the databases. Both methods, Telesto and sequential scan, were implemented using the C++ programming language (main memory only), considering the filtering stage. All tests were performed 5 times, so the values presented and discussed in this section correspond to the average of these 5 executions.

Figure 5 shows the time spent (in seconds) by Telesto during the **construction stage** for distinct database sizes. Telesto showed a linear growth in response to increasing data volumes (dotted line). Therefore, we can conclude that the Telesto's construction stage is scalable, since the increase in volume did not impair its performance. Furthermore, the time spent to construct the index of the biggest database was approximately 70 seconds, which can be considered small, since, as shown in the next test, a single sequential scan for range query on that database spent on average 9 seconds. Therefore for multiple queries the time spent in construction stage pays off the sequential scan.

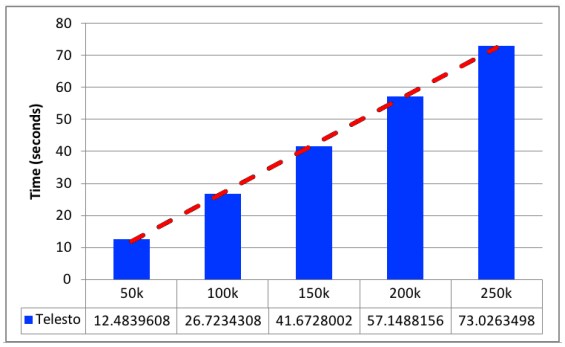
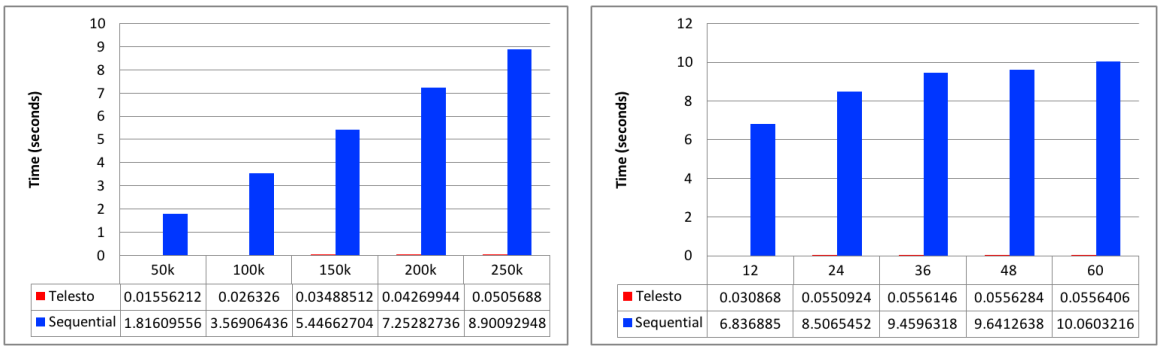


Fig. 5. Telesto: time spent during the construction stage.

The second set of experiments evaluated the **querying stage**. First, we show in Figure 6 the time spent to process range queries. The results depicted in Figure 6(a) evaluate the impact of the database size increase and in Figure 6(b) evaluate the impact of the query size increase using a database of size 250K. As can be observed, in both cases Telesto's performance was far superior to sequential scan. In fact, Telesto provided a performance gain that ranged from 116 to 176 times considering the impact of database size increase, and ranged from approximately 180 to 220 times considering the impact of query size increase. Another positive feature is that Telesto showed a linear growth in query

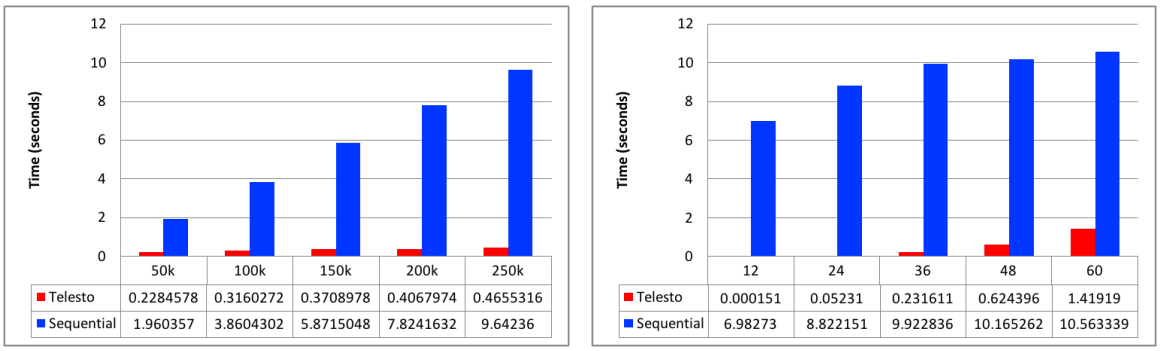
cost in response to increased database sizes (Figure 6). Moreover, Telesto's time spent during range query processing was almost indifferent to the query size, except for the query size of 12 symbols, as presented in Figure 6(b).



(a) Increasing database size (b) Increasing query size

Fig. 6. Time comparison of range query processing.

Figure 7 depicts the performance results for NN queries. While the results shown in Figure 7(a) evaluate the impact of the database size increase, the results shown in Figure 7(b) evaluate the impact of a query size increase using the database of size 250K. Telesto provided a performance improvement of up to 20 times considering the impact of the database size increase, and up to almost 168 times considering the impact of the query size increase. Similar to the range query performance results, Telesto also showed a linear growth in NN query costs in response to increased database sizes (Figure 7(a)). However, the time spent during NN query processing was not indifferent to the query size, because of the query range used is not fixed, as in range queries. We can conclude that Telesto is scalable in the size of the query, and it is very efficient.



(a) Increasing database size (b) Increasing query size

Fig. 7. Time comparison of NN query processing.

6. CONCLUSIONS AND FUTURE WORK

In this article, we proposed Telesto, a novel indexing approach aimed at time series similarity search. Telesto was described in terms of its main features: (i) pre-processing stage, which discretizes time series into strings; (ii) construction stage, which indexes these strings using a generalized suffix tree; and (iii) querying stage, which performs range and NN queries over the generalized suffix tree.

Telesto was validated through performance tests using real-world time series databases. The results showed that Telesto was very efficient in the processing of range and NN queries. Compared with the sequential scan, Telesto provided a performance gain of up to 220 times in the range query processing and of up to 168 times in the NN query processing. The results also showed that the time spent by Telesto to process range queries was almost indifferent to the query size. Furthermore, Telesto showed a linear growth in construction and query costs in response to increased database sizes. Therefore, the increase in volume did not impair the performance of Telesto's construction and querying stages.

We are currently investigating the persistent storage of Telesto, since it currently remains entirely in the main memory. We also plan to investigate how the discretization levels of the SAX representation (i.e. the size of its alphabet) influences the efficiency of Telesto, since in this article we only used the SAX representation with discretization level equals to 5 in the pre-processing stage.

REFERENCES

Chan, K.-P. and Fu, A. W.-c. Efficient time series matching by wavelets. In *Proceedings of the 15th International Conference on Data Engineering*. Washington, DC, USA, pp. 126–134, 1999.

Das, G., Lin, K.-I., Mannila, H., Renganathan, G., and Smyth, P. Rule discovery from time series. In *Knowledge Discovery and Data Mining*. New York, USA, pp. 16–22, 1998.

Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. J. Querying and mining of time series data: experimental comparison of representations and distance measures. *PVLDB* 1 (2): 1542–1552, 2008.

Faloutsos, C., Ranganathan, M., Manolopoulos, Y., and Manolopoulos, Y. Fast subsequence matching in time-series databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Minneapolis, USA, pp. 419–429, 1994.

Geurts, P. Pattern extraction for time series classification. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*. Freiburg, Germany, pp. 115–127, 2001.

Goldin, D. Q., Kanellakis, P. C., and Kanellakis, P. C. On similarity queries for time-series data: Constraint specification and implementation. In *Proceedings of the 1st International Conference on Principles and Practice of Constraint Programming*. Cassis, France, pp. 137–153, 1995.

Gusfield, D. *Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology*. Cambridge University Press, 1997.

Kalpakis, K., Gada, D., and Puttagunta, V. Distance measures for effective clustering of ARIMA time-series. In *Proceedings of the IEEE International Conference on Data Mining*. San Jose, USA, pp. 273–280, 2001.

Keogh, E., Chakrabarti, K., Pazzani, M., and Mehrotra, S. Dimensionality reduction for fast similarity search in large time series databases. *Knowledge and Information Systems* 3 (3): 263–286, 2001a.

Keogh, E., Chakrabarti, K., Pazzani, M., and Mehrotra, S. Locally adaptive dimensionality reduction for indexing large time series databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Santa Barbara, USA, pp. 151–162, 2001b.

Lin, J., Keogh, E. J., and Lonardi, S. Visualizing and discovering non-trivial patterns in large time series databases. *Information Visualization* 4 (2): 61–82, 2005.

Lin, J., Keogh, E. J., Lonardi, S., and Chiu, B. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*. San Diego, USA, pp. 2–11, 2003.

Lin, J., Keogh, E. J., Wei, L., and Lonardi, S. Experiencing SAX: a novel symbolic representation of time series. *Data Mining and Knowledge Discovery* 15 (2): 107–144, 2007.

Luo, K., Wang, J., and Sun, J.-G. Rules discovery from cross-sectional short-length time series. In *Proceedings of the 8th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*. Sydney, Australia, pp. 604–614, 2004.

Rasheed, F., Al-Shalalfa, M., and Alhajj, R. Efficient periodicity mining in time series databases using suffix trees. *IEEE Transactions on Knowledge and Data Engineering* 23 (1): 79–94, 2011.

Romani, L. A. S., Goncalves, R. R. V., Zullo-Jr., J., Traina-Jr., C., and Traina, A. J. M. New DTW-based method to similarity search in sugar cane regions represented by climate and remote sensing time series. In *Proceedings of the IEEE International Geoscience & Remote Sensing Symposium*. Honolulu, USA, pp. 355–358, 2010.

Rouse, J. W., Haas, R. H., Schell, J. A., and Deering, D. W. Monitoring vegetation systems in the great plains with ERTS. In *Proceedings of the Third ERTS Symposium*. Washington, DC, USA, pp. 309–317, 1973.