Big Data Research 15 (2019) 12–28

Contents lists available at ScienceDirect

Big Data Research

www.elsevier.com/locate/bdr

Visual Exploration of Geolocated Time Series with Hybrid Indexing ☆

Georgios Chatzigeorgakidis [a,b,*], Kostas Patroumpas [b], Dimitrios Skoutas [b], Spiros Athanasiou [b], Spiros Skiadopoulos [a]

[a] Department of Informatics and Telecommunications, University of the Peloponnese, Tripolis, Greece
[b] Information Management Systems Institute, ATHENA R.C., Athens, Greece

ARTICLE INFO

Article history:
Received 14 September 2018
Received in revised form 20 December 2018
Accepted 6 February 2019
Available online 7 February 2019

Keywords:
Time series
Spatial indices
Visual exploration
Summarization

ABSTRACT

Geolocated time series are time series that correspond to specific locations. They can represent, for example, visitor check-ins at certain venues or readings of sensors installed at various places. The amount and significance of such time series have increased in many domains over the last years. However, although several works exist for time series visualization and visual analytics in general, there is a lack of efficient techniques for visual exploration and analysis of geolocated time series in particular. In this paper, we present two approaches that rely on hybrid spatial-time series indices to allow for efficient map-based visual exploration and summarization of geolocated time series data. In particular, we use the BTSR-tree index and we introduce a new variant of the *i*SAX index, called geo-*i*SAX. The former is a spatial-first hybrid index that extends the R-tree by maintaining bounds for the time series indexed at each node. Following a similar rationale, geo-*i*SAX is a time series-first hybrid index that maintains spatial MBRs of the geolocated time series indexed in each node. We describe the structure of these indices and show how they can be directly exploited to produce map-based visualizations of geolocated time series at different levels of granularity. We empirically validate our approach using two real-world datasets, as well as a synthetic one that is used to test the scalability of our methods.

1. Introduction

Time series are generated and stored at a vastly increasing rate in many industrial and research applications, including the Web and the Internet of Things, public utilities, finance, astronomy, biology, and many more. A significant portion concerns *geolocated* time series, i.e., those generated at, or otherwise associated with, specific locations. While indexing, mining and exploring time series data has attracted a lot of interest from the database and data mining communities [1–3], studying of geolocated time series is still largely overlooked.

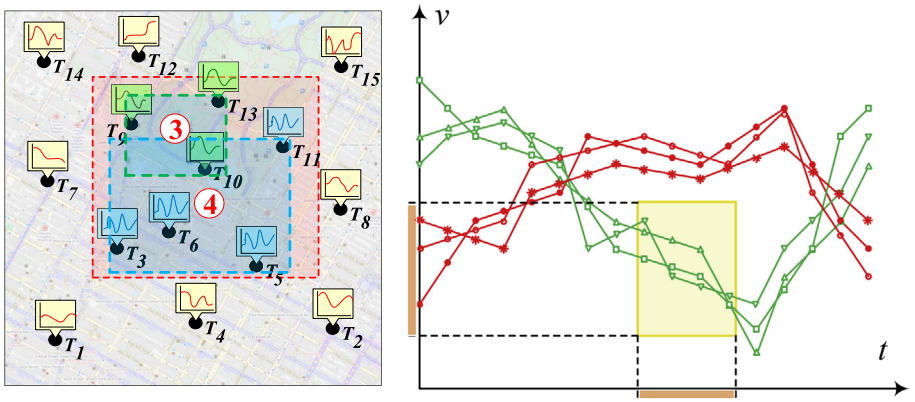
Geolocated time series can be found in various domains and applications. For instance, time series can be used to represent, analyze and forecast water consumption measured by smart meters installed in urban households [4]. Analyzing such time series can provide valuable insights regarding trends and patterns of consumer behavior in daily life. These results can then be used to forecast and balance water demand, as well as to plan and prioritize interventions that can guide consumers towards more sensible water use. Similar use cases can be found in other domains, such as in geomarketing or mobile advertisement, where geolocated time series may represent the number of visitors or the revenue generated at a certain location across time. Extracting insights, trends and patterns can be significantly facilitated by *map-based visualizations* of *summarized* time series data. Such visualizations can reveal, for instance, which type of consumption patterns are most frequently observed among consumers in a certain area or what the spatial distribution of sales for a certain product looks like.

However, time series is an inherently complex data type, and such datasets can reach extremely large volumes, both *horizontally* (i.e., very long series of values across time) and *vertically* (i.e., time series generated by countless sources). Consequently, management, analysis and exploration of big time series data is a task requiring efficient and scalable algorithms. In particular, visual exploration of geolocated time series needs to process the required information efficiently, while the user interacts with the application. For example, whenever the user zooms in or scrolls the map, visual analytics and aggregates should be computed on-the-fly, e.g., identifying the predominant patterns in the time series and their spatial distribution within the actual map area.

☆ This article belongs to VSI: Big Data Exploration.

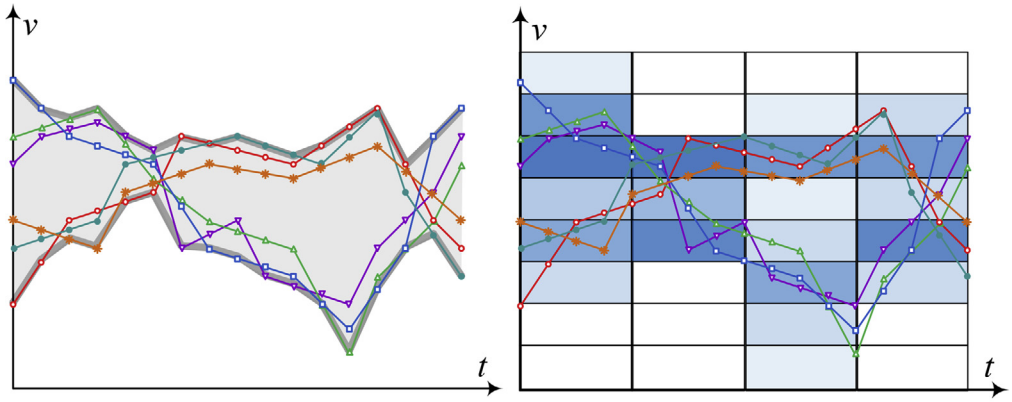
* Corresponding author at: Department of Informatics and Telecommunications, University of the Peloponnese, Tripolis, Greece.
E-mail addresses: chgeorgakidis@uop.gr (G. Chatzigeorgakidis), kpatro@imis.athena-innovation.gr (K. Patroumpas), dskoutas@imis.athena-innovation.gr (D. Skoutas), spathan@imis.athena-innovation.gr (S. Athanasiou), spiros@uop.gr (S. Skiadopoulos).

https://doi.org/10.1016/j.bdr.2019.02.001
2214-5796/



(a) Exploring time series patterns in a *spatial region*. (b) Exploring time series in a *timebox* of interest.

Fig. 1. Visual exploration examples over geolocated time series. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)



(a) Bundle summary. (b) Tile map summary.

Fig. 2. Examples of computed summaries on the time series domain.

Consider the example illustrated in Fig. 1(a). When the user zooms the map into the red rectangle, the visualization application should identify, summarize and present the two patterns (shown in blue and green color) appearing therein. For such requests that inherently combine spatial filters with time series analysis, it is inefficient to evaluate each predicate separately, e.g., apply a spatial filter on the time series of a large dataset and then calculate summaries of the candidates, or vice versa. The same stands for the case of exploration on the time series domain, as depicted in Fig. 1(b). Consider a user drawing a *timebox* (i.e., a rectangle in the time series domain) or zooming in the yellow part. The application should identify the time series that are fully contained within that filter area, i.e., their values along the specified time range fall within the value range (both ranges shown in orange in Fig. 1(b)), and then provide an informative summary comprising aggregate spatial information to avoid cluttering the map.

Efficient filtering and retrieval over large datasets of geolocated time series can be enabled by *indexing*. Several approaches have been proposed that efficiently index large amounts of plain time series data. They either rely on *Discrete Wavelet Transform* to reduce the dimensionality of time series [5], or make use of a family of indices based on *Symbolic Aggregate Approximation* (SAX) [3,6,1,7]. However, all aforementioned techniques index the data solely on the time series domain, not taking the spatial dimension into account. If each analyzed time series is inherently associated with a spatial attribute (e.g., locations of smart meters), such indexing is not sufficient for queries and visualizations that additionally involve spatial filters.

In this paper, we propose two geolocated time series summarization approaches for visual exploration, named *bundle* and *tile map summary*. These are supported and driven by two appropriate *hybrid* indices that speed up the result computation, providing efficient exploration of geolocated time series data. They consist of a spatial and a time series summary that jointly facilitate knowledge extraction and insight gaining. The spatial summary is similar for both and consists of *Minimum Bounding Rectangles* (MBRs) of geolocated time series, according to a specific predicate (i.e., spatial proximity, or time series similarity). Each MBR is associated with a counter denoting the number of time series it contains. A visualization example of the spatial summary is depicted in Fig. 1(a), where the geolocated time series are organized in two groups (i.e., green and blue colored) according to their similarity. Each group is depicted along with a number that indicates the amount of time series that it contains (i.e., three geolocated time series for the first group and four for the second).

The main difference among the two methods lies in the time series part of the summary. The bundle summary consists of sets of *Minimum Bounding Time Series* (MBTS) [8], an example of which is depicted in Fig. 2(a). An MBTS is a band with upper and lower bounds that encloses all time series of a set, providing with a notion of a range of the time series values throughout the time axis. On the other hand, the tile map summary (Fig. 2(b)) of a set of time series indicates (using a corresponding shading), the density of the time series points at each tile of a partitioning of the domain, obtained by discretizing the time and value axes. This way, it avoids overplotting that would be caused by outputting a large

number of resulting time series and provides a notion of how the values of the time series are distributed across time.

The bundle summary is driven by our recently proposed *BTSR-tree* index [8]. This is a *spatial-first* hybrid index, in the sense that it is primarily built on the spatial attribute of the data. More specifically, it is an extension of the R-tree spatial index [9], offering efficient support for similarity search over geolocated time series. The idea behind the BTSR-tree index is to combine both spatial proximity and time series similarity. To that end, in addition to the standard MBR denoting the spatial bound of its contents, each node is augmented with an MBTS of all the time series contained in its subtree. Maintaining both kinds of bounds per node enables pruning the search space simultaneously in the spatial and the time series domains while traversing the index. To increase pruning effectiveness, the time series indexed in a given node are further distinguished into *bundles* on the basis of their similarity, hence achieving tighter bounds in the MBTS of these bundles. For providing prompt visualizations of summaries over geolocated time series data and minimizing latency when drawing the relevant graphic elements, we need early access to both spatial and time series information while traversing the index. For this purpose, we adapt the BTSR-tree index so as to also include *aggregates* per node, i.e., the number of time series pertaining to each bundle. Subsequently, we introduce a new traversal algorithm for efficient retrieval of a given number of bundles that are the most representative in the map area.

The tile map summary is driven by geo-*i*SAX, a hybrid index we introduce in this paper. This is a *time series-first* index, i.e., it is primarily built in the time series domain. More specifically, it constitutes a hybrid variant of the *i*SAX index [3,6,1], augmented with spatial attributes of its nodes' children, to combine spatial and time series information. In each node, besides the SAX word that describes all its children time series, geo-*i*SAX keeps also the MBR that they form. To minimize the size and overlap of the MBRs, we propose a spatial splitting policy, that instead of choosing the splitting dimension in a round-robin fashion (as in *i*SAX), it does so by selecting the dimension that produces the smallest overlap and overall size of the two generated MBRs. We introduce a traversal algorithm for applying timebox search on large (both vertically and horizontally) geolocated time series datasets. The traversal algorithm is applied on our geo-*i*SAX index and returns a tile map-like summary of the qualifying geolocated time series, by taking advantage of the SAX representation's properties.

To the best of our knowledge, this is the first work that considers visual exploration and summarization of geolocated time series. Specifically, we propose two summarization methods enabling efficient map-based exploration driven by suitable hybrid indices. The work in this paper builds upon and extends our effort in [10], which showed the benefits of using BTSR-tree for visual exploration of geolocated time series. Here, we introduce another novel summarization approach that enables a different exploration method, by employing a time series-first hybrid index. In brief, our main contributions are as follows:

- We suggest an adapted variant of the BTSR-tree index, as well as a novel algorithm for its traversal in order to quickly retrieve summaries (a.k.a. bundles) of geolocated time series within a given spatial area.
- We propose a hybrid variant of the *i*SAX index, called geo-*i*SAX, which combines time series with spatial information within its nodes. Based on that, we describe a novel traversal algorithm for geo-*i*SAX that enables fast timebox search by performing efficient pruning, while avoiding false negatives.
- We exemplify the proposed visualization methods with two use cases based on real-world datasets. In addition, we empirically evaluate the performance of our summarization methods, confirming their low execution time against a large synthetic dataset of geolocated time series.

The remainder of this paper is organized as follows. Section 2 reviews related work. Section 3 outlines basic concepts and formulates the problem. Sections 4 and 5 introduce our methods for efficient visual exploration of geolocated time series by harnessing the potential of the BTSR-tree and geo-*i*SAX indices, respectively. Section 6 presents indicative use cases with map visualizations and also reports performance results from our empirical study. Finally, Section 7 concludes the paper and outlines future research directions.

2. Related work

In the following, we review existing approaches regarding indexing and visual exploration of time series.

Indexing of time series Earlier approaches towards indexing of time series data were often based on leveraging multi-resolution representations. For instance, the Discrete Wavelet Transform [11] is used in [5] to gradually reduce the dimensionality of time series data via the Haar wavelet and generate an index using the coefficients of the transformed sequences. In [12], it is further observed that, other than orthonormal wavelets, bi-orthonormal ones can also be used for efficient similarity search over wavelet-indexed time series data, demonstrating several such wavelets that outperform the Haar wavelet in terms of precision and performance. In addition, an alternative approach regarding k-nearest neighbor search over time series data is introduced in [13]. The proposed method accesses the coefficients of Haar-wavelet-transformed time series through a sequential scan over step-wise increasing resolutions.

State-of-the-art approaches for time series indexing comprise methods based on the *Symbolic Aggregate Approximation* (SAX) representation [14]. This is derived from the *Piecewise Aggregate Approximation* (PAA) representation of a time series [15,16], by quantizing the segments of its PAA representation on the y-axis. The first attempt to leverage the potential of the SAX representation was presented in [3], introducing the indexable Symbolic Aggregate Approximation (*i*SAX), capable of a multi-resolution representation for time series. The *i*SAX index was further extended to *i*SAX 2.0 [6] by enabling bulk loading of time series data. Its next version is the *i*SAX 2+ index [1], which handles better the expensive I/O operations caused by the aggressive node splitting while building the index. Finally, the ADS+ index [7] is another extension of *i*SAX, which overcomes the still significantly expensive index build time by adaptively building the index while processing the workload of queries issued by the user. A comprehensive overview of time series indexing approaches based on SAX representation is presented in [17].

Unfortunately, none of the abovementioned access methods can inherently support geolocated time series, i.e., time series inextricably associated with a location. To the best of our knowledge, the only index in the literature that supports such time series is the BTSR-tree index [8]. This hybrid index follows a similar rationale set by *spatio-textual indices* [18] that have been proposed to speed up evaluation of queries combining location-based predicates with keyword search. Essentially, this paradigm implies combining a spatial index structure (e.g., R-tree, Quadtree, Space-Filling Curve) with a textual index (e.g., inverted file, signature file). Depending on their structure, these variants can be characterized either as *spatial-first* or *textual-first* indices [19]. In a similar spirit, our BTSR-tree is a spatial-first index based on the R-tree that can additionally abstract similarity of time series instead of a textual one. As a result, it can offer analogous improvements when searching

against geolocated time series data, as we discuss in more detail in Section 4.1.

Visual exploration of time series Numerous approaches attempt to leverage the potential of summarizing or aggregating the information of large time series data to facilitate visual exploration and knowledge extraction. An early approach is [20], where the authors use tile maps and box plots to discover ten-year trends in air pollution data. In [21], the authors introduce a pixel-oriented visualization to detect recursive patterns, where each data value is represented by one pixel. The authors demonstrate the potential of their method using a stock market dataset. An extension of this work is presented in [22], where several time granularities are combined in a single visualization to enhance the knowledge extraction potential of recursive patterns.

Of particular interest are visualization approaches that attempt to leverage the potential of declarative SQL-like languages and DBMSs to enable exploratory queries. Such an approach is suggested in M4 [23], where the authors introduce an aggregation-based dimensionality reduction scheme for visualizing horizontally large time series using line charts. Their approach operates on top of an RDBMS and supports various SQL queries that select and visualize particular parts of time series. ForeCache [24] leverages two prefetching mechanisms to facilitate exploration of large geospatial, multidimensional and time series data stored in a DBMS. By predicting the user's behavior, it fetches the necessary data as the user interacts with the application. Another declarative language-based visualization is suggested in [25], where relational algebra queries are used to represent the visualization, leveraging the potential of traditional and visualization-specific optimizations. In contrast, a recent tutorial [26] advocates the use of example-based methods in exploration of large relational, textual, and graph datasets. Such a *query-by-example* approach has been applied in [27] so as to explore relevance feedback for retrieval from time series databases. Instead of returning the top matching time series, this technique incorporates diversity into the results, which are presented to the user for feedback and refined in several rounds.

RINSE [28] is a Recursive Interactive Series Explorer specifically designed for exploration of data series. Built on top of ADS+ [7], a special adaptive index structure for data series, it can progressively build parts of the index on demand at query time, concerning only those chunks of the data involved in users' queries. In terms of visualization, users can get those series qualifying to range or nearest-neighbor queries interactively drawn on screen, as well as monitor various statistics regarding the index footprint (e.g., RAM and disk usage) as it gets updated. In contrast, ATLAS [29] is a visual analytics tool specifically geared towards interactivity when ad hoc filters, arbitrary aggregations, and trend exploration are applied against massive time series data. This client-server architecture employs a column store as its backend equipped with indexing, and preemptively caches data that may be required in queries so as to reduce latency when *panning*, *scrolling*, and *zooming* over time series. Recently, the ONEX paradigm [30] concerns online exploration of time series. It first constructs compact similarity groups over time series for specific lengths based on Euclidean distance, and then can efficiently support exploration of these groups with the Dynamic Time-Warping (DTW) method over their representatives of different lengths and alignments. *Smoothing* can be applied to streaming time series to remove noise in visualizations while preserving large-scale deviations [31]. To highlight important phenomena without harming representation quality from oversmoothing, this approach introduces quantitative metrics involving variance of first differences and kurtosis to automatically calibrate smoothing parameters.

The ability to zoom in to specific parts of interest of a large time series can significantly enhance the exploratory potential of a visualization. Stack zooming [32] provides such a functionality, by building hierarchies of line chart visualizations for user-defined intervals on large time series data. Each selected interval is zoomed and stacked beneath the initial time series. A similar approach is KronoMiner [33], which employs a radial-based visualization to enable zooming functionality for specific time intervals. The interface is visually refined through an iterative design procedure involving expert user feedback. ChronoLenses [34] introduces a domain-independent visualization that offers the ability to perform on-the-fly transformations (e.g., Fourier transform, auto-correlation) of the selected interval using *lenses*.

Zooming in regions of interest in time series can be performed via *timeboxes*, which essentially consist of rectangular regions on the time series domain thus specifying intervals in both the time and value axis. The procedure retrieves the time series whose values in the given region are fully contained in the rectangle. Hochheister et al. introduced timeboxes [35] along with *TimeSearcher*, an application for visual exploration of time series datasets that implements timebox queries. The user is able to draw rectangles on the time series domain and the results are separately displayed on-screen. Keogh et al. [36] extended the timeboxes, introducing the *Variable Time Timeboxes*, which allowed a degree of uncertainty in the time axis. Later versions of TimeSearcher (such as [37]) provided enhanced functionality, allowing the visual exploration of longer time series (>10,000 time points) and offering forecasting functionality.

Contrary to our approach, none of the aforementioned methods and systems provides map-based visual exploration of *geolocated* time series. In our recent work [10], we have introduced a summary construction method for geolocated time series, that utilizes our spatial-first BTSR-tree, to enable spatial-domain map-based exploratory visualizations. In this paper, we augment this work by introducing a *time series-first* hybrid index to facilitate timebox search on both horizontally and vertically large time series datasets. We enable efficient exploration on geolocated time series datasets, by timely executing user-defined timebox search, enabling the exploration also in the time series domain.

3. Problem formulation

A *time series* is a time-ordered sequence of values $T = \{v_1, \ldots, v_n\}$, where v_i is the value at the i-th time point and n is the length of the series. In this work, we specifically deal with *geolocated time series* [8], i.e., time series that are additionally characterized by a *location*, denoted by $T.loc$. Assuming a 2-dimensional space, we further use the notation $T.loc_x$, $T.loc_y$ to refer to the (x, y) coordinates of T's location. In the rest of the paper, when it is clear from the context, we also refer to such geolocated time series as *objects* for brevity.

To quantify the proximity of two geolocated time series in the *spatial domain*, we measure the distance of their respective locations. More specifically:

Definition 1 *(Spatial Distance).* The *spatial distance* between two geolocated time series T and T' is calculated using the Euclidean distance of their respective locations. Furthermore, we normalize this distance with $maxDist_{sp}$, i.e., the maximum spatial distance of any pair of objects in the dataset, to obtain a measure in the interval $[0, 1]$. Thus:

$$dist_{sp}(T, T') = \frac{\sqrt{(T.loc_x - T'.loc_x)^2 + (T.loc_y - T'.loc_y)^2}}{maxDist_{sp}}. \quad \square \tag{1}$$

Moreover, we measure similarity in the *time series domain*. Similarly to other prior works (e.g., [3]), we apply the Euclidean distance in this domain. In future work, we plan to make use of more complex distance measures [38]. More specifically:

Definition 2 *(Time Series Distance).* The *time series distance* between two time series T and T' of equal length n is calculated as:

$$dist_{ts}(T, T') = \frac{\sqrt{\sum_{i=1}^{n}(T.v_i - T'.v_i)^2}}{maxDist_{ts}} \quad (2)$$

where $maxDist_{ts}$ denotes the maximum distance on the time series domain of any pair of objects in the dataset and is used for normalization, as above. □

Our objective in this paper is to compactly represent a large number of geolocated time series by some form of summaries so as to support and facilitate their visual exploration. Intuitively, given a set of geolocated time series, we want to provide summaries that express both their pattern across time as well as their corresponding spatial extent. These summaries may be constructed over the whole dataset, e.g., to provide an initial quick overview of the whole data, or over the results of a previous query, e.g., over those time series located inside a bounding box drawn by the user on the map. Specifically, we consider two types of summaries, called *bundle summaries* and *tile map summaries*, respectively, which we describe next.

3.1. Bundle summaries

This type of summary is composed of a set of k *"bundles"*, where each bundle comprises the following information:

- a cluster of similar time series in the temporal domain
- a set of Minimum Bounding Rectangles (MBRs) summarizing in the spatial domain the respective locations of those time series
- an integer indicating the number of objects located within each of the above MBRs.

To derive such bundles, we use the notion of *Minimum Bounding Time Series* (MBTS) introduced in [8]. An MBTS bundles together a set of time series $\mathcal{T}$ using a pair of bounds that fully contain all of them. Fig. 3 depicts an example of two MBTSs for two disjoint sets of time series. Formally:

Definition 3 *(Minimum Bounding Time Series (MBTS)).* Given a set of time series $\mathcal{T}$, its MBTS consists of an *upper bounding time series* $T^{\sqcap}$ and a *lower bounding time series* $T^{\sqcup}$, constructed by respectively selecting the maximum and minimum of values at each time point among all time series in set $\mathcal{T}$ as follows:

$$\begin{aligned} T^{\sqcap} &= \{\max_{T\in\mathcal{T}} T.v_1, \ldots, \max_{T\in\mathcal{T}} T.v_n\} \\ T^{\sqcup} &= \{\min_{T\in\mathcal{T}} T.v_1, \ldots, \min_{T\in\mathcal{T}} T.v_n\}. \end{aligned} \quad (3)$$

□

Hence, we can formulate the problem of summarizing a set of geolocated time series by means of a bundle summary as follows:

Problem 1 *(Bundle Summary).* Given a set $\mathcal{T}$ of geolocated time series, a spatial area q of interest, a number k of desired bundles, and a number of l MBRs per bundle, the problem is to efficiently compute a *bundle summary* that consists of a list of k tuples over the subset $\{T \in \mathcal{T} : within(T.loc, q)\}$ of time series located within area q. Each such tuple in the bundle summary has the following structure:

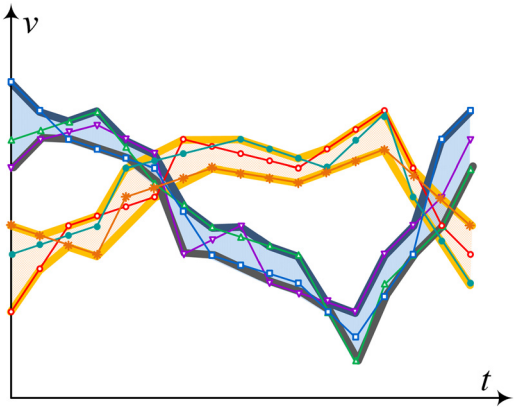


Fig. 3. Example illustrating the resulting bundles for two sets of time series.

$$R_b = \{\langle mbts, \{\langle mbr, cnt\rangle\}\rangle\} \quad (4)$$

where *mbts* is a time series summary in the form of MBTS and is associated with a list of l MBRs; the count *cnt* of objects within each such *mbr* is also available. □

In Section 4, we show how the BTSR-tree index can be used to address this problem, i.e., to efficiently compute bundle summaries.

3.2. Tile map summaries

The bundles in the summary type introduced above are formed in a *data-driven* manner, as objects belonging to the same bundle should be similar (e.g., based on clustering). An alternative way to visually highlight spatio-temporal patterns in a large set of geolocated time series is through summaries that rely on a *fixed partitioning* of the time series domain. More specifically, the entire domain may be subdivided into adjacent, non-overlapping *tiles* (Fig. 2(b)), so that each tile captures the portion of a time series falling within this tile. Simple aggregates (e.g., counts) of time series per tile can easily convey the distribution of values in the dataset. As shown in the example, the higher the concentration of data points within a tile, the darker its shade. More formally:

Definition 4 *(Tile Map).* Let the *time domain* $[t_{min}, t_{max})$ be divided into successive intervals of equal size τ, resulting into a subdivision $\{[t_{min}, t_{min}+\tau), [t_{min}+\tau, t_{min}+2\tau), \ldots, [t_{max}-\tau, t_{max})\}$. Subdivision of the *value domain* $[v_{min}, v_{max})$ is carried out using a finite number of *breakpoints* $\{v_1, v_2, \ldots, v_h\}$ where $v_1 = v_{min}$, and $v_h = v_{max}$, whereas the rest may be arbitrary values provided that $v_1 < v_2 < \cdots < v_h$. This subdivision yields disjoint, consecutive segments $\{[v_{min}, v_1), [v_2, v_3), \ldots, [v_h, v_{max})\}$ in the value axis. The resulting *tile map* is a matrix of tiles over both domains, so that tile (i, j) corresponds to time interval $[t_{min}+i\cdot\tau, t_{min}+(i+1)\cdot\tau)$, $i \in 0, \ldots, \left\lceil \frac{t_{max}-t_{min}}{\tau} \right\rceil$, and to value segment $[v_j, v_{j+1})$, $j \in 0, \ldots, h-1$.

Essentially, such a tile map has a similar effect as *space-driven* partitionings like quadtrees or grid subdivisions [39]. As in the case of spatial objects, a time series can be checked for containment within each tile. More specifically, a time series data point v_t at time t is contained in tile (i, j) if $t_{min}+i\cdot\tau \le t < t_{min}+(i+1)\cdot\tau$ and $v_j \le v_t < v_{j+1}$. Once all time series are mapped into tiles, this matrix offers a summary of the entire dataset.

However, we may inspect specific portions of a tile map, by checking a group of neighboring tiles. This can be abstracted as a *timebox* [40] applied over both domains. Intuitively, such a timebox is a rectangle in the time series domain that fully contains a set of time series in the time and value range that it represents. Fig. 1(b) depicts with green color the time series that are contained within a timebox, among a set of time series. More specifically:

Definition 5 *(Timebox).* A *timebox* p specifies a time interval $[t, t')$ and a value range $[v, v')$ in order to identify any qualifying time series. We denote as $timebox(T, p)$ once a time series T qualifies to this timebox p if $\forall t_i \in [t, t'), v \leq T.v_i < v'$.

Overall, given a timebox p and a spatial area q of interest over a set of geolocated time series, we are interested in identifying:

- The tiles that summarize objects located within q and are also fully included within p, i.e., no data point of the time series falls outside p in the respective time interval.
- In addition, the summary provides also a set of k MBRs covering all qualifying time series; a counter measures the time series contained within each such MBR.

We can now formulate the problem of computing *tile map summaries* as follows:

Problem 2 *(Tile Map Summary).* Given a set $\mathcal{T}$ of geolocated time series, a timebox p and a spatial area q of interest, and a number k of desired MBRs, a *tile map summary* provides a list of k tuples over the subset $\{T \in \mathcal{T} : within(T.loc, q) \wedge timebox(T, p)\}$ of time series located within area q and qualifying to timebox p. Each such tuple in the tile map summary has the following structure:

$$R_t = \{\langle tmap, \{\langle mbr, cnt\rangle\}\rangle\} \quad (5)$$

where *tmap* represents the tile map constructed over the qualifying time series and is associated with a list of k MBRs that outline their spatial extent; the count *cnt* of objects within each such *mbr* is also available. □

In Section 5, we propose a technique that can efficiently construct tile map summaries by employing an extended variant of the *i*SAX index.

4. Computing bundle summaries

Intuitively, the first visualization method displays the bundle summaries for a spatial area of interest, as defined in Section 3.1. This may concern the currently visible area on a map, so a set of time series patterns and their respective spatial extents are computed and visualized. Using this process, a user can select the bundle of her preference and the proper spatial summary will appear on the map after acquiring the necessary MBRs from the BTSR-tree index. Whenever the user zooms in/out or pans around the map, the BTSR-tree is traversed, and the corresponding bundles, MBRs, and object counts are obtained to drive the visualization. In each case, the rectangle corresponding to the visible part of the map is used to feed a traversal algorithm that efficiently gathers the results. Next, we first outline the structure of the BTSR-tree index, and then we introduce a novel algorithm for its traversal in order to compute the bundle summaries for a given area of interest.

4.1. The BTSR-tree index

To generate efficient visualizations of geolocated time series data, we need early access to both spatial and time series related information while traversing the index, in order to maintain low latency levels when drawing the required graphic elements. However, none of the approaches presented in Section 2 supports geolocated time series indexing. To the best of our knowledge, the recently proposed BTSR-tree index [8] is the only one that provides the desired functionality.

The BTSR-tree is based on the R-tree [9] for the spatial indexing part. The R-tree organizes a hierarchy of nested d-dimensional rectangles. Each node corresponds to a disk page and represents the MBR of its children or, for leaf nodes, the MBR of its contained geometries. The number of entries per node (excluding the root) is between a lower bound m and a maximum capacity M. Query execution in R-trees starts from the root. MBRs in any visited node are tested for intersection against the search region. Qualifying entries are recursively visited until the leaf level or until no further overlaps are found. Several paths may be probed, as multiple sibling entries could overlap with the search region. The BTSR-tree extends the information stored within each node of the R-tree with bundles of MBTS, as illustrated with the colored strips per node in Fig. 4. This allows to efficiently prune the search space when evaluating hybrid queries combining time series similarity with spatial proximity.

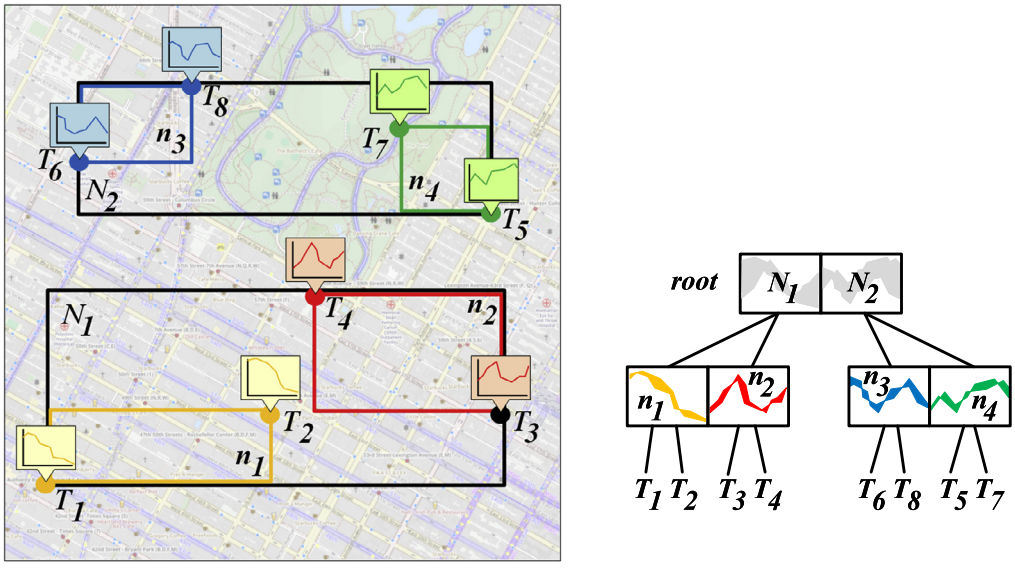
A BTSR-tree is constructed exactly like an R-tree [9] with respect to the spatial contents of a geolocated time series dataset $\mathcal{T}$, as depicted in Fig. 4(a). As in R-trees, each node of the BTSR-tree has at least m and at most M entries and stores the MBRs of its children. Additionally, for each child, a node stores a pre-specified number of MBTS, shown as colored strips per node in Fig. 4(b), each one enclosing all the time series indexed in its subtree. Each MBTS is calculated according to Eq. 3. Construction and maintenance of the BTSR-tree follow the procedures of the R-tree for data insertion, deletion and node splitting. Objects (i.e., geolocated time series) are inserted into leaf nodes and any resulting changes are propagated upwards.

Once the nodes have been populated, the MBTS of each node are calculated bottom-up. First, in each leaf node, the contained time series are clustered into k bundles using *k-means clustering* according to their Euclidean distance in the time series domain. Then, the MBTS of each bundle is computed and stored in the node. The example in Fig. 3 depicts the $k = 2$ MBTS (as two bands with a thick outline) obtained for a set of time series (shown as thin polylines). As a next step, each parent node receives all the MBTS of its children and computes its own k bundles and respective set of MBTS by clustering them. The process continues upwards, until reaching the root of the tree. Optionally, *Piecewise Aggregate Approximation* [15,16] can be applied over the time series. As detailed in [8], this allows a trade off between the number of bundles per node and the MBTS resolution, thus permitting a larger number ($> k$) of bundles in nodes at higher levels in the tree hierarchy.

4.2. Deriving bundle summaries from the BTSR-tree index

To support the summaries required by the visualization method, we further extend the information stored in each node of the BTSR-tree index with the *count* of geolocated time series that are fully contained within each bundle. This is done bottom-up during insertion, while the index is traversed to calculate the bundles. At each leaf node, after the clustering, we propagate the number of members of each cluster to its parent, which in turn calculates its clusters and aggregates the counts it has received for each bundle's members. This procedure continues up to the root of the tree. We stress that this process concerns the building of the index and is carried out once geolocated time series are being inserted.

We now present our summarization technique for producing map-based visualizations of geolocated time series. The process is outlined in Algorithm 1. It takes as input the *input rectangle* q, i.e.,



(a) Sample dataset with MBRs. (b) Hybrid BTSR-tree index.

Fig. 4. Sample dataset with MBRs as maintained by the BTSR-tree index.

the spatial area of interest for which the visualization is produced, the number k of bundles and the number l of MBRs per bundle to be generated. The process comprises three distinct steps. Initially, the BTSR-tree index is traversed to obtain the MBRs contained in the input rectangle, along with their bundles and the number of objects per bundle (Line 1). Next, k-means clustering is applied using the average time series per bundle as centroids (Line 2). Finally, the new bundles are calculated and the proper MBRs and corresponding object counts are assigned to each bundle (Line 3). Next, we describe each step in more detail.

Step 1: BTSR-tree Traversal. During this step, the BTSR-tree index is traversed, with the target being the fast provision of a predefined number k of geolocated time series bundles contained within the given area q, along with l MBRs where these bundles can be found and the total number of geolocated time series that reside within each MBR. All required information is stored within the nodes of the BTSR-tree, thus, when a node that is contained within the input rectangle is found, the relevant information is retrieved and added to the intermediate results, without any need to continue searching in its sub-tree. The output of this step is passed to the next phase of bundle clustering.

In more detail, the traversal is performed as follows. After initializing a queue with the root's children (Line 7), we iterate over it (Line 8) until it is empty. For each inner node's child N', we check whether its MBR is contained within the given input rectangle q (Lines 11–12). If so, its MBR, the time series bundles, and the number of objects per bundle are added to the intermediate results (Line 13) as a *tuple* with the following components:

$$\langle mbr, \{(mbts_1, cnt_1), ..., (mbts_k, cnt_k)\}\rangle.$$

Each such tuple indicates the MBR of a node (mbr), consisting of the coordinates of the lower left and upper right point, as well as k pairs denoting the bundles of the node along with the corresponding number of objects per bundle. If the MBR is not contained in the input rectangle q, we check whether it overlaps with q and if so, we add the child node to the queue (Line 15). If not, this MBR is located outside the input rectangle, and thus we can skip searching this subtree. Once no more nodes are left to search, the intermediate results are finally returned (Line 16).

Step 2: Bundles Clustering. The aforementioned traversal method returns tuples, each containing the bundles residing in the input rectangle, the corresponding nodes' MBRs and the number of objects per bundle. Next, k-means clustering is executed on the average time series of each bundle. Line 2 of Algorithm 1 calls the clustering procedure. Initially, for each tuple (Line 20), we iterate over its bundles (Line 21) and generate a new tuple per bundle of the following format:

$$\langle T_{avg}, mbts, cnt, mbr\rangle$$

This new tuple contains an average time series, the bundle itself ($mbts$), the number cnt of objects enclosed in this bundle, and the MBR (mbr) this bundle belongs to (Line 23). The average time series T_{avg} is calculated by averaging the upper and lower bound of each bundle (Line 22), i.e., average value at each time point. The resulting collection of tuples is fed to the k-means algorithm (Line 24) in order to return the required number k of bundles to be created. This clustering generates a clustered collection of tuples using the calculated average time series. These results are then forwarded to step 3 (Line 25).

Step 3: Bundles Calculation and MBR Assignment. During this step, the clustered tuples received from step 2 are used to calculate the final bundles, the corresponding l MBRs and total number of objects per MBR are assigned to each bundle. The final bundles are calculated in a similar manner to the MBTS bundles during BTSR-tree construction. More specifically, at each time point, we obtain the maximum and minimum value among the corresponding upper and lower bounds for the bundles of each cluster (see Section 4.1). Then, we apply k-means clustering on each bundle's MBRs, obtaining a total of l new MBRs per bundle along with an aggregate count of the time series contained therein. The final result is then used for visualization.

Line 3 of Algorithm 1 calls the corresponding procedure. For each cluster of tuples received from step 2 (Line 28), we loop over its members (Line 31) and we use each tuple's bundle to update the upper and lower bounds (Line 32). Then, we apply k-means clustering on the cluster's MBRs, obtaining l new MBRs along with their counts (Line 33). Once the bounds and the corresponding list of MBRs for the current bundle have been calculated, we issue an

aggregated tuple to the final result (Line 34). This tuple has the following components:

$\langle mbts', \{(mbr_1, cnt_1), ..., (mbr_l, cnt_l)\}\rangle$

where $mbts'$ is a resulting bundle, along with l MBRs associated with it. Each MBR is accompanied with the corresponding number of objects (i.e., raw time series) therein. The final result with all such tuples is then returned in order to generate the visualization (Line 36).

Algorithm 1: Bundles summarization of geolocated time series.

```
Input: Input rectangle q; number k of bundles to be generated; number l of
       MBRs per bundle
Output: A list R_b containing tuples of bundles, MBRs, and object counts
1  R ← IndexTraversal(q)                                   // Step 1
2  R_c ← BundlesClustering(R, k)                           // Step 2
3  R_b ← BundlesCalculation(R_c, l)                        // Step 3
4  return R_b
5  Procedure IndexTraversal(q)
6      R ← ∅
7      Q ← Root.getChildren()
8      while Q ≠ ∅ do
9          N ← Q.getNext()
10         if N is not leaf then
11             foreach N' ∈ N.getChildren() do
12                 if q.contains(N'.mbr) then
13                     R ← R ∪ {⟨N'.mbr, {(N'.mbts, N'.cnt)}⟩}
14                 else if q.overlaps(N'.mbr) then
15                     Q ← Q ∪ N'.getChildren()
16     return R
17 Procedure BundlesClustering(R, k)
18     R_c ← ∅
19     C ← ∅
20     foreach t ∈ R do
21         foreach b ∈ t.mbts do
22             T_avg ← avg(b.up, b.lo)
23             C ← C ∪ {⟨T_avg, b, t.cnt(b), t.mbr⟩}
24     R_c ← kmeans(C.avg, k)
25     return R_c
26 Procedure BundlesCalculation(R_c, l)
27     R_b ← ∅
28     foreach Cl ∈ R_c.clusters do
29         mbts ← ∅
30         mbr ← ∅
31         foreach t ∈ Cl do
32             mbts ← updateMBTS(mbts, t.mbts)
33             {(mbr, cnt)} ← kmeans(l, t.mbr, t.cnt)
34         R_b ← R_b ∪ {⟨mbts, {(mbr, cnt)}⟩}
35     return R_b
```

5. Computing tile map summaries

In the following, we present our second visualization method, which allows the user to draw one or more timeboxes on the time series domain. This triggers a traversal of our hybrid geo-*i*SAX index to obtain the geolocated time series in the currently visible map area and also fully contained within these timeboxes. The result comes in the form of tiles, each spanning between two *i*SAX breakpoints, along with a count per tile indicating the number of time series whose SAX symbol resides within that tile. This count is used to generate the visualization in the form of tile map. A predefined number of MBRs and corresponding counts is also returned, generated by clustering the locations of the resulting geolocated time series and used for the spatial part of the visualization. Whenever the user zooms in/out or pans over the map, or whenever she draws a new timebox, the procedure is repeated, the index is traversed and the visualization is regenerated. Next, we first outline the original structure of the *i*SAX index and then we introduce its geo-*i*SAX variant, which enables evaluation of timebox queries and also maintenance of spatial information in its nodes. As we discuss next, those two extensions provide the necessary support for computing tile map summaries as specified in Section 3.2.

5.1. The iSAX family of indices

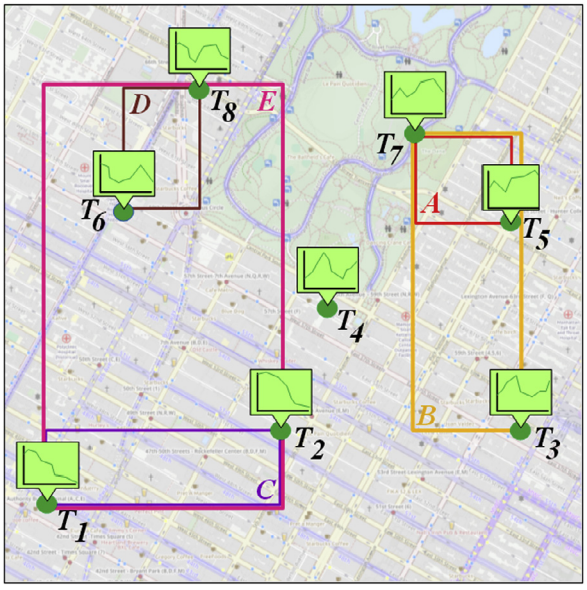
The *Symbolic Aggregate approXimation* (SAX) is a multi-resolution representation of a time series introduced in [3]. It can be derived from its *Piecewise Aggregate Approximation* (PAA) [15,16] by quantizing the PAA segments on the v-axis. As exemplified in Fig. 5(b), a time series T_2 is transformed to a PAA representation of w=3 words (along the t axis) with real-valued coefficients (the horizontal red bars). Fig. 5(c) lists the SAX representations computed similarly for all time series shown in Fig. 5(a); note that locations are completely ignored by *i*SAX and only the time series part of objects is indexed. To get a SAX representation for a time series, these coefficients are discretized along the v-axis using *breakpoints* (shown with dashed lines) assuming a $\mathcal{N}(0, 1)$ Gaussian distribution that enables generation of equi-probable symbols for a given cardinality ($b = 4$ symbols are used in this example). By using bitwise representations for these symbols, coarser SAX values can be obtained from more refined ones, by simply ignoring trailing bits.

By completely ignoring the spatial locations and using the SAX representations of all time series in this dataset, an *i*SAX index [3] can be built. Indeed, the root node captures the complete *i*SAX space and itself does not contain any SAX words, but only points to its children nodes (in the worst case, their number is $2w$). Each leaf has a pointer to a disk file containing the raw time series that represents. The leaf itself also stores the *i*SAX word of highest cardinality among these time series. An internal node designates a split in SAX space and is created when the number of time series contained by a leaf node exceeds a fixed capacity. This split is binary and is made at a given position $j = 1..w$ of the SAX word using a round-robin policy, so it always yields two children that differ on their j-th symbol while replicating the rest from their parent node. In essence, the SAX space represented by every node fully contains the union of the SAX spaces of its subtree.

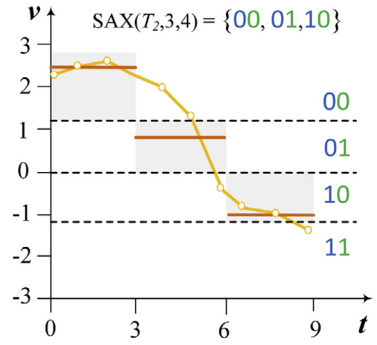
*i*SAX can answer *similarity queries*, and thus can be also used in k-nearest neighbor search [3]. Searching for time series similar to a given q simply traverses the *i*SAX tree, looking for a leaf node having the same *i*SAX word as q. The respective raw times series are fetched from disk and a sequential scan identifies those matching with q. Improvements over the original *i*SAX basically alleviate the bottleneck of expensive I/O when building the index for large datasets. Our methodology presented next is based on the latest *i*SAX2+ [1].

5.2. The geo-iSAX index

We introduce geo-*i*SAX, a time series-first *hybrid* variant of the *i*SAX index that allows significant traversing speed-ups by pruning both on the spatial and time series domain. This is achieved by storing in each node of the tree, apart from the SAX word of the geolocated time series it contains, the MBR that they form. Initially, the time series part of the index is built, following the procedure described in Section 5.1. As a next step, similarly to the BTSR-tree index, we traverse the index in a bottom-up fashion, first generating the MBRs of the geolocated time series contained in the leaf nodes. As we go upwards, we update the MBR information of each visited inner node, using the MBRs of its children nodes, until we



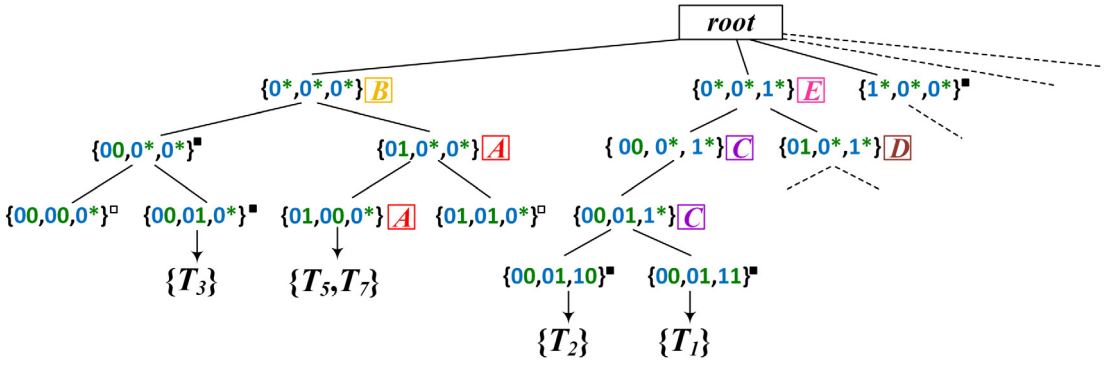
(a) Sample dataset with MBRs as maintained by geo-*i*SAX.



(b) SAX of time series T_2 using $w = 3$ words and cardinality $b = 4$.

$$SAX(T_1, 3, 4) = \{00, 01, 11\}$$
$$SAX(T_2, 3, 4) = \{00, 01, 10\}$$
$$SAX(T_3, 3, 4) = \{00, 01, 01\}$$
$$SAX(T_4, 3, 4) = \{10, 01, 01\}$$
$$SAX(T_5, 3, 4) = \{01, 00, 00\}$$
$$SAX(T_6, 3, 4) = \{00, 10, 01\}$$
$$SAX(T_7, 3, 4) = \{01, 00, 00\}$$
$$SAX(T_8, 3, 4) = \{01, 10, 10\}$$

(c) SAX of all time series.



(d) The geo-*i*SAX index with letters indicating MBRs (depicted in 5(a)) attached to each node. Nodes with filled boxes indicate a degenerate MBR consisting of a single point; nodes with hollow boxes indicate no data. Subtrees under dash lines are not shown for brevity.

Fig. 5. Sample dataset with MBRs and SAX representations of time series as maintained by the geo-*i*SAX index.

reach the root, whose MBR will contain the geolocated time series of the whole dataset. Fig. 5(d) illustrates the structure of the geo-*i*SAX tree created over a sample dataset of geolocated time series.

Due to the fact that the *i*SAX index is created to solely index time series data, the MBRs generated by our hybrid variant may be highly overlapping, mitigating the pruning potential while traversing the index. To alleviate the negative effects of the numerous overlaps, we introduce an alternative *splitting policy* for geo-*i*SAX, which attempts to minimize the overlapping area, while maintaining the total area covered by the MBRs that occur after a split at the lowest possible levels. Recall that the original *i*SAX index selects the split dimension (i.e., the segment of a node's word on which the split will occur) for a node using a round robin approach. Our method loops over all split dimensions and for each one, it calculates the SAX word that would occur upon splitting on it. Then, it generates the MBRs for that specific split using the location of the geolocated time series contained in the node to be split. For each split dimension, it computes the sum of the two new MBRs' intersection area and the total area that they cover. The selected split is the one that generates the smaller sum. Due to the rather small number of word segments in an *i*SAX index, this procedure does not incur high construction costs, only slightly affecting the overall index construction time.

5.3. Deriving tile map summaries from the geo-iSAX index

Next, we present our summarization approach for tile map visualization of geolocated time series, obtained using timeboxes. In order to maintain low latency even for large datasets, we traverse geo-*i*SAX to obtain the resulting geolocated time series in a timely fashion. However, to avoid false negatives, we need to ensure that the pruned nodes do not contain any qualifying geolocated time series, as we discuss next.

Pruning Rule for Timebox Queries on geo-iSAX. When traversing the geo-*i*SAX index, we first evaluate whether its MBR satisfies the spatial constraint q (i.e., intersects with or is within q). If so, we

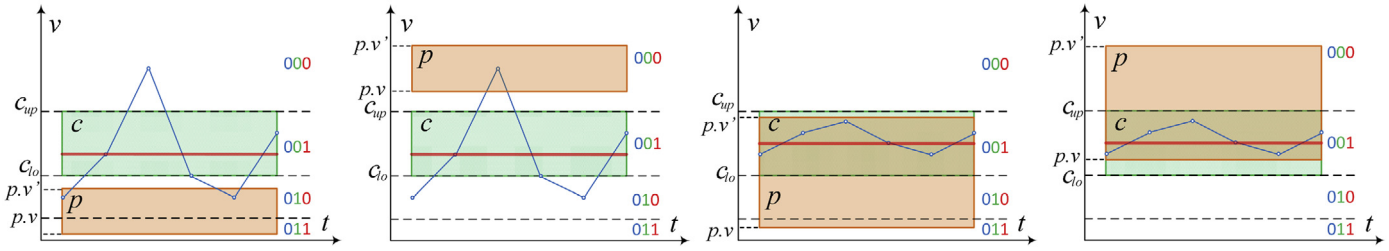


Fig. 6. Cases where a timebox p is either outside or intersecting a given tile c.

need to check the timebox constraint p, i.e., whether the time series it contains *certainly* have data points that reside outside the given timebox. Consider the time series in Fig. 6. Without loss of generality, we suppose a SAX word of length $w = 1$ and cardinality $b = 8$, i.e., $SAX(T, 1, 8) = \{010\}$. The solid red horizontal line is the PAA value that classifies this segment of the time series to the SAX value 010. Tile c defined by the two breakpoints that contain the PAA value is shown in green color, while the user-defined timebox p specifying a value range $[v, v')$ is depicted in orange.

As can be noticed in Fig. 6(a), the upper side of the timebox is below the lower side of tile c, i.e., $p.v' < c_{lo}$. In this case, we can safely prune a node that is described by this SAX word, because there is at least one data point outside the timebox, "pulling" the PAA upwards and within c. For the same reasons, we can prune a node whose breakpoint-defined area is completely below the given timebox, as depicted in Fig. 6(b). In this case, the lower side of the timebox is above the upper side of c ($p.v > c_{up}$), indicating that there is at least one data point outside the timebox "pulling" the PAA value downwards and within c.

On the other hand, in case the timebox intersects with tile c, we cannot safely prune the corresponding node. For example, in Fig. 6(c), $c_{lo} < p.v' < c_{up}$, so there may exist a time series that is fully contained in the timebox and thus be part of the result. The same stands for the example in Fig. 6(d), where $c_{lo} < p.v < c_{up}$. Finally, for the cases where timebox p fully contains tile c or vise-versa, it is trivially deduced that no pruning can apply.

We stress that the above observations only hold when in the time axis the given timebox is aligned to the segments of the SAX words in the index. For example, for a time series of length $n = 10$ and a word length of $w = 5$, the resulting segments will have length equal to $n/w = 2$. Thus, the timebox must be aligned to the data points t_0, t_2, t_4, t_6 and t_8.

Traversal Algorithm over geo-iSAX. Algorithm 2 outlines the process for producing the tile map visualizations of geolocated time series. The process takes as input a spatial rectangle q, a user-defined timebox p and the number k of MBRs that will be returned. It produces a list R_t containing k tuples of MBRs along with time series counts and the tile map, as defined in Eq. 4. For each inner node, the procedure checks whether its MBR intersects rectangle q and whether its SAX word might represent time series within timebox p. If a leaf node is reached and both constraints are met, we iterate over its raw geolocated time series and add the ones qualifying for the timebox to the final result R_t, after properly updating the breakpoint tile counts.

In more detail, the procedure takes place as follows. Starting from the root of the geo-*i*SAX index, we iterate over each node's children (Lines 5-6 in Algorithm 2). Next, we check whether the node to be evaluated is a leaf node (Line 8) and if it is not, we iterate over its children (Line 9) and check whether each one's MBR either is contained or intersects the spatial rectangle (Line 10). If so, we check whether its SAX word could represent time series within the timebox (Line 11) and if this is the case, we add it to

Algorithm 2: Tile map summarization of geolocated time series.

Input: The input rectangle q; the timebox p; the number of MBRs to be generated k
Output: A list R_t containing a tile map and tuples of MBRs and object counts

```
1  R_t ← IndexTraversal(Root, q, p, k)
2  return R_t
3  Procedure IndexTraversal(Root, q, p, k)
4      R_t ← ∅, R ← ∅, tmap ← ∅, L ← ∅
5      Q ← Root.getChildren()
6      while Q ≠ ∅ do
7          N ← Q.getNext()
8          if N is not leaf then
9              foreach N' ∈ N.getChildren() do
10                 if q.contains(N'.mbr) ∨ q.overlaps(N'.mbr) then
11                     if timebox(N'.sax, p) then
12                         Q ← Q ∪ N'.getChildren()
13         else
14             foreach T ∈ N'.getChildren() do
15                 if timebox(T, p) then
16                     tmap ← updateCounts(tmap, T.sax)
17                     R ← R ∪ T
18             L ← L ∪ {R.mbr, |R|}
19     {⟨mbr, cnt⟩} ← kmeans(L, k)
20     R_t ← {⟨tmap, {⟨mbr, cnt⟩}⟩}
21     return R_t
22 Procedure timebox(X, p)
23     if isSax(X) then
24         foreach s ∈ {p.t_min, p.t_max} do
25             c ← breakpoints(X_s)
26             if p.v_min > c_up ∨ p.v_max < c_lo then
27                 return False
28         return True
29     else
30         foreach t ∈ {p.t_min, p.t_max} do
31             if X_t > p.v_max ∨ X_t < p.v_min then
32                 return False
33         return True
```

the queue Q to be evaluated (Line 12). At this point, we should mention that, in order to avoid expensive calculations, we first perform the spatial check as it is less computationally expensive than the timebox check, avoiding the latter in case the node's MBR is outside the input rectangle. If the currently evaluated node is a leaf node (Line 13), we iterate over the geolocated time series that it contains (Line 14) and check whether it is fully contained within the user-defined timebox (Line 15). If so, we update the tile counts matrix *tmap* (Line 16) and add the corresponding raw time series to the set R (Line 17). Finally, we add to a list L the MBR of the set of qualifying time series along with its size (Line 18).

Table 1
Datasets used in the experiments.

Dataset	Area (km^2)	Number of time series	Length n of each time series
Water	114	822	168
Taxi	2,500	417,960	168
Synthetic	114	4,000,000	168

Upon exiting the loop, we apply k-means clustering on the resulting centroids and obtain k tuples containing MBRs along with time series counts.

The procedure that checks whether a SAX word could represent time series that are fully contained within the given timebox is described in detail starting from Line 22 in Algorithm 2. It takes as input a timebox and a raw time series or SAX word that we want to check against the given timebox. We initially check whether the given X argument is a SAX word (Line 23). If it is, we iterate over all the segments that are contained (Line 22) in the time range defined by the timebox and for each segment we obtain the *i*SAX breakpoints that enclose it (Line 25). Afterwards, we check whether the lower side of the timebox is above the upper side of the tile defined by the obtained breakpoints, or vice versa, as depicted in Fig. 6. If this is true, we return false. If this is not the case for any of the segments, the method returns true (Line 28). A similar procedure is followed for the case that argument X is a raw time series (Lines 29-33).

6. Experimental evaluation

In this section, we evaluate the proposed visualization methods. We first describe our experimental setup including the datasets that we use in the evaluation. Next, we present illustrative visualizations over real-world geolocated time series, as well as scalability results against a synthetic dataset containing 4 million geolocated time series.

6.1. Experimental setup

All experiments were conducted on a Dell PowerEdge M910 with 256 GB RAM and 4 Intel Xeon E7-4830 CPUs, each containing 8 cores clocked at 2.13 GHz. We assume that all indices fit in memory, hence parameter selection for their construction was based on this assumption. We use two real-world datasets selected from different application domains and with diverse characteristics. In addition, we generated a synthetic dataset to test the scalability of our method. Table 1 lists a summary of the main characteristics for each dataset.

DAIAD water consumption (water) Courtesy of the DAIAD project,[1] we acquired a geolocated time series dataset of hourly water consumption for 822 households in Alicante, Spain from 1/1/2015 to 20/1/2017. In order to get a more representative dataset for our tests, we calculated the average weekly time series per household, which is the average consumption value per hour of the week. The length of each resulting time series is $24 \times 7 = 168$ values across the week.

NYC taxi dropoffs (taxi) This dataset contains time series extracted from yellow taxi rides in New York City during 2015. The original data[2] provides pick-up and drop-off locations, as well as corresponding timestamps for each ride. For each month, we generated time series by applying a uniform spatial grid over the entire city (cell side was 200 meters) and counting all drop-offs therein for each day of the week at the time granularity of one hour. Thus, we obtained the number of drop-offs for 24×7 time intervals in every cell, which essentially captures the weekly fluctuation of taxi destinations there. The centroid of each cell is used as the geolocation of the corresponding time series.

Table 2
Parameters tested in the experiments.

Parameter	Values
Dataset size	1000K, 2000K, 3000K, **4000K**
Map scale	1:50000, 1:25000, 1:20000, 1:15000, 1:10000, **1:5000**, 1:500

Synthetic water consumption (synthetic) To examine the scalability of our algorithms, we generated a synthetic dataset comprising 4 million geolocated time series by inflating the water consumption dataset. This was achieved by using the original time series as seeds and introducing some random variations in their location and pattern. We chose the water dataset so as to generate a more densely populated dataset (Alicante is a medium-sized city) to stress-test our summarization methods. In scalability tests, we also make use of randomly chosen subsets from this synthetic dataset.

Table 2 lists the range of values for the parameters used in our tests concerning both methods; default values are shown in bold.

6.2. Evaluation of bundle summarization

We first present a detailed evaluation of our method concerning bundle summaries. Specifically, we present two visualization examples on two real-world datasets. Then, we evaluate the scalability of our method in terms of different map scales and dataset sizes.

6.2.1. Map visualizations

The visualization for the bundle summary depicts the MBTS derived for the most representative patterns of time series at the currently visible area of the map. Once our summarization method returns the results, the corresponding MBRs contained in the current view and zoom level are drawn on the map, along with the number of the geolocated time series that belong to the selected bundle. This number is depicted using circles, colored green for small numbers, yellow for larger and red for more densely populated MBRs, thus easily conveying the local intensity of this pattern. The bundles are listed on the left of the map, using confidence bands to indicate their upper and lower bounds. The average time series of each bundle is also depicted. A user can scroll this list and select the bundle of their preference. Once a bundle is selected, the contents of the map are updated accordingly with the respective MBRs and aggregates.

Fig. 7 shows an example of the bundle summary visualization using the water dataset, for $k = 5$ bundles and $l = 3$ MBRs. The depicted area is in the center of Alicante, in the most densely populated zone of the city. In this example, `Bundle 4` is selected (indicated with a green colored frame) and the relevant MBRs are shown on the map (using red colored boxes). This indicates that inside each depicted MBR there exists a specific number of geolocated time series that have been clustered to the chosen bundle. As mentioned, each geolocated time series in this dataset represents hourly water consumption of a household across one week. Different consumption behaviors have been grouped together and a daily pattern for each bundle can be noticed which is due to the Circadian rhythmic way that people consume water [41]. The rather large number of geolocated time series in the bundle, considering the zoom level and the extent of the MBRs, intuitively

[1] http://daiad.eu/.
[2] http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml.

Fig. 7. Visualizing water consumption patterns in the city center of Alicante (map scale 1:5000).

suggests that neighboring families tend to have similar water consumption behavior.

Fig. 8 illustrates another example, this time using the taxi dataset in New York City, for $k = 5$ bundles and $l = 11$ MBRs. This dataset is significantly larger, and the zoom level selected in this example is lower (a larger geographic area is visible), hence the MBRs contain a larger number of time series. In this figure, we choose `Bundle 1`, which represents the rather quieter taxi dropoff zones in Manhattan, as the total number of dropoffs there is rarely over 60 during any hour of the week. In this example, there is also a clear daily routine in all bundles, with the dropoffs reaching a local maximum twice per day, suggesting the rush hours in New York City, when people commute to and from their work. In almost all bundles, the daily pattern is significantly different on Saturdays and Sundays, which confirms the intuition that during weekends people do not tend to commute in a routinely fashion. Overall, such visual representations of information digested from massive time series data can easily catch users' attention to important phenomena and ongoing trends, confirming the usefulness of our approach.

6.2.2. Performance evaluation

Parameters In preliminary tests, we fine-tuned parameters used against the synthetic dataset. Conclusively, for the scalability evaluation, we built the BTSR-tree index setting the minimum and maximum number of entries per node to $m = 750$ and $M = 2000$, respectively. Note that the index fits in memory, so such large parameter values do not have a negative impact on performance. For an evaluation of the BTSR-tree index under different parameter settings, please refer to [8]. Regarding the number of bundles, we set $k_0 = 5$ for its leaf nodes. The number k of bundles and the number l of MBRs per bundle for the traversal algorithm is set to be equal to the number of bundles at the leafs, i.e., $k = l = k_0 = 5$.

Baseline method for detailed bundle summaries In order to determine the trade-off between responsiveness and accuracy of our method (Section 4.2), we compare it with a baseline approach, which involves more *detailed summarization*. The latter utilizes the raw geolocated time series retrieved from the spatial filtering and generates the summaries by first applying k-means clustering on the time series domain to obtain the bundles, followed by another clustering in the spatial domain within each bundle to obtain its respective MBRs.

Results We evaluate the performance of our approach on larger datasets in terms of response time for different zoom levels on the map using map scales, since zooming-in requires deeper traversal of the BTSR-tree index. The comparison is performed for subsets of the synthetic dataset of different size. We measure the *accuracy* both in spatial and time series domains, by calculating the mean Euclidean distance of each object located within the spatial area of interest from its closest bundle and MBR, respectively. Since this is intended as an interactive application, where the summarization method is triggered as soon as the user moves the map, *response times* must be adequately small. In our method, this is facilitated by the fact that the search along a path stops once it encounters a node whose MBR is contained in the actual map extent (rectangle). However, for the same reason, the responsiveness is expected to come at the expense of the summaries' level of detail, since the inner nodes contain coarser information regarding the time series they contain in their sub-trees.

Fig. 9 depicts traversal costs for different map scales over the areas covered by the three datasets. More specifically, the water and synthetic datasets cover the area of the city of Alicante, Spain, whereas the taxi dataset the wider metropolitan area of New York City. Response time in all cases is equal or lower than one second. The synthetic dataset, due to its very high density is significantly slower than the rest, however still the results are obtained in less than a second. The response for the water dataset is almost instant due to its small size and very low density. Initially, in all cases, at the largest scale, the visible area of the map contains all the time series in the dataset, thus it only has to retrieve information from the root of the index. Then, as we zoom in, more nodes have to be visited, as the MBRs of the accessed nodes begin to overlap

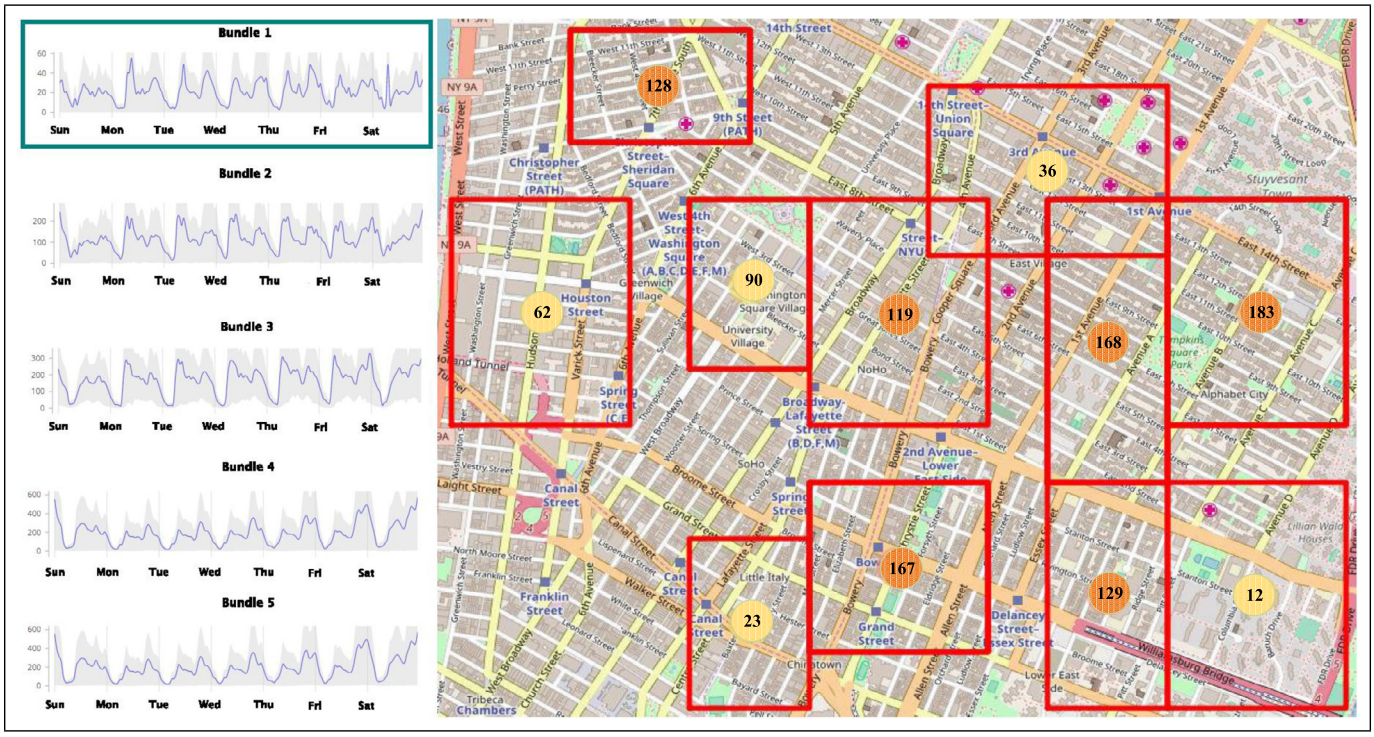


Fig. 8. Visualization of taxi dropoff patterns in Manhattan, NYC (map scale 1:10000).

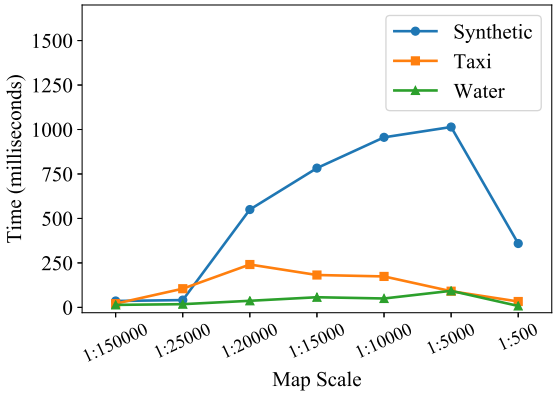


Fig. 9. Execution time for different map scales.

with the map rectangle and their children have to be retrieved. The worst case for the synthetic dataset is at scale 1:5000, which roughly corresponds to a large neighborhood of the city, where many time series are located. For the taxi dataset, the worst case is at 1:20000, which corresponds to the wider Manhattan area and then the response time gradually drops due to the lower dataset density. The number of nodes accessed in each case is proportional to the response times, ranging from one node (the root) in case of the smaller map scale (all city) up to 165 at scale of 1:5000 for the synthetic dataset, one up to 53 for the taxi dataset and one up to 15 for the water dataset. Interestingly, fewer node accesses are required in all cases at the very large scale of 1:500, since the respective small map area overlaps with fewer nodes and most of the search space is pruned.

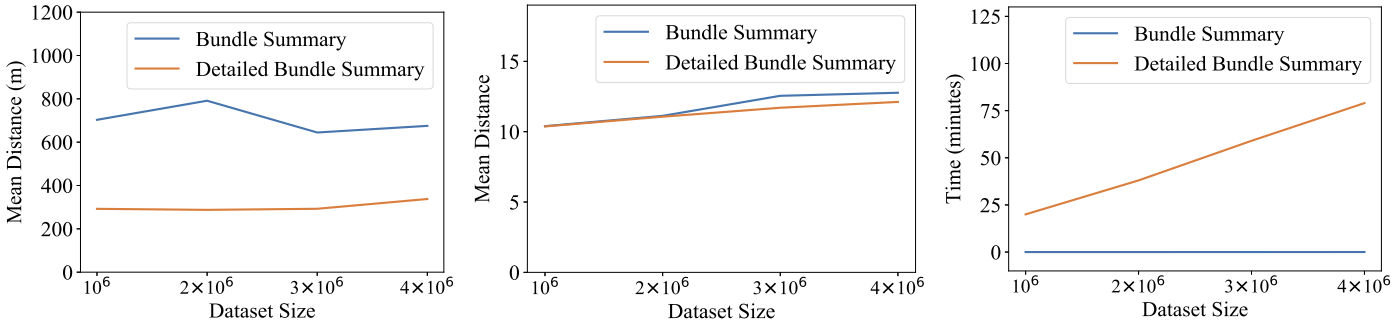
Fig. 10 depicts the accuracy and responsiveness comparison between the bundle summarization method and its detailed version. In the spatial domain (Fig. 10(a)), it is apparent that the mean Euclidean distance of the raw data from the closest MBR centers is close to two times larger for our bundle summary, indicating a loss in accuracy, as expected. The mean distance is not heavily affected by the dataset size at any case, indicating a rather stable summary quality, independent of the amount of the results. The case is quite different in the time series domain (Fig. 10(b)), where the mean distance of the raw time series from the closest bundle's average time series is only slightly larger for the bundle summary – especially for smaller datasets –, indicating a rather good summary quality. There is a slight worsening trend in both cases as the size of the dataset is increased, possibly due to the tendency of the summary to be more generic as the number of results increases, with a larger number of BTSR-tree nodes taking part in the summary calculation. However, this loss in accuracy in both domains is largely compensated in terms of execution time (Fig. 10(c)), with the bundle summary being close to one second in all cases, while the detailed approach is linearly slowed down as the dataset size increases, requiring more than an hour to generate the summary against 4 million objects. Overall, this test confirms that our proposed method for bundle summaries can offer a really large speedup in terms of response time with tolerable concessions in accuracy, even against a heavily dense synthetic dataset where a large number of time series are contained within a small area.

6.3. Evaluation of tile map summarization

Next, we present an evaluation of our method for obtaining tile map summaries. We first demonstrate two examples of visual exploration via summarization over two real-world geolocated time series datasets. Then, we compare the scalability of our method with a baseline detailed summary in terms of different map scales and dataset sizes.

6.3.1. Map visualizations

This visualization depicts a tile map summary of the geolocated time series that are contained within the currently visible part of the map. Whenever the user either moves around, zooms in and out the map, or draws a timebox on the time series domain, our summarization method is invoked to traverse the geo-*i*SAX index



(a) Accuracy in spatial domain. (b) Accuracy in time series domain. (c) Response time.

Fig. 10. Assessment of bundle summarization.

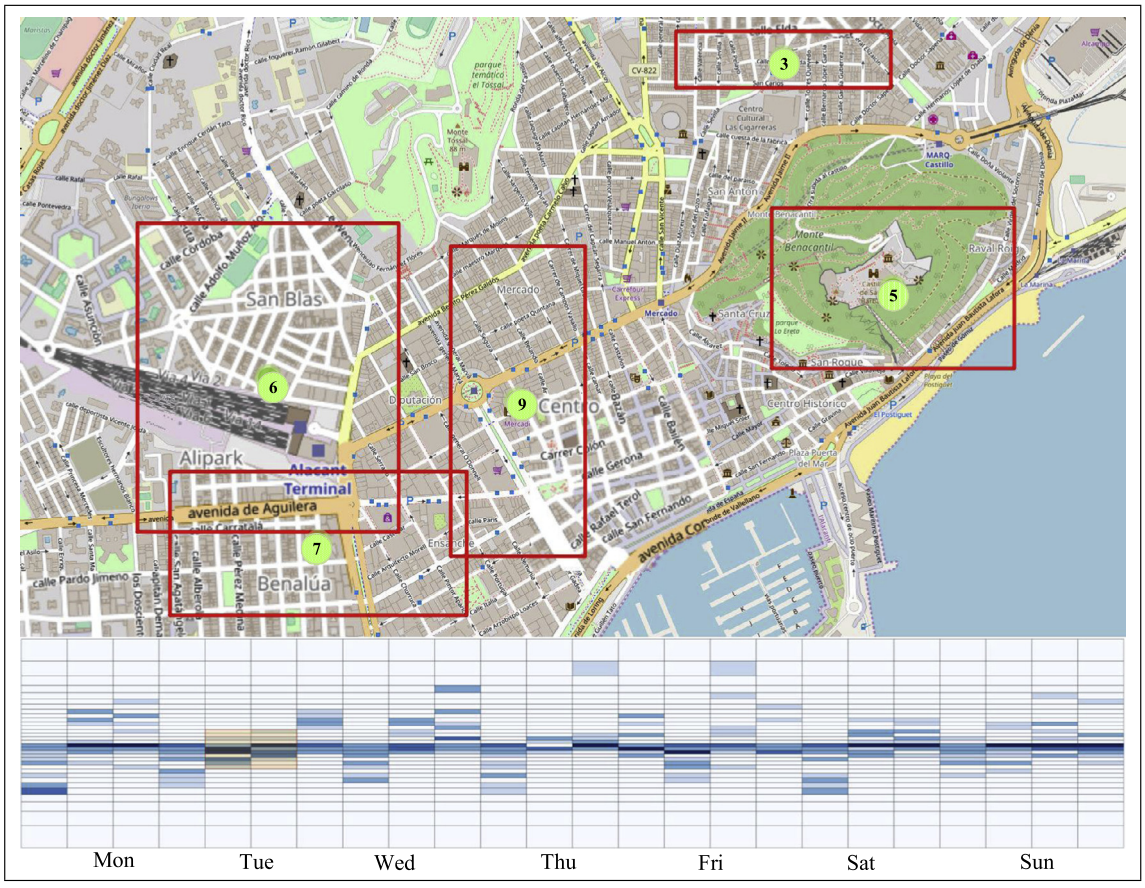


Fig. 11. Visualization of water consumption tile map summary in the city center of Alicante (map scale 1:5000).

and return the MBRs and tile map that correspond to the visible area. Once the results are returned, they are drawn on the map and time series frames respectively. Similarly to the bundle summary and in order to present the local density, the number of geolocated time series contained within each MBR is depicted using circles colored green for small numbers, yellow for larger and red for more densely populated MBRs. The time series frame is located on the bottom of the map and is essentially a depiction of the returned tile map, using lighter shades of blue for less populated tiles and darker shades for the more dense ones. The timeboxes are drawn on this frame, allowing a selection of arbitrary ranges on the value axis, while, on the time axis, the selection is forced to be aligned with the *i*SAX segments.

An example of the tile map visualization is illustrated in Fig. 11, for the central area of Alicante, with $k = 5$ MBRs, a SAX word length of $w = 24$ and a maximum cardinality of $b = 32$. An example timebox (depicted as an orange box) spans two *i*SAX segments (corresponding to Tuesdays on the time axis) and has a range of one standard deviation on the value axis. The associated MBRs are depicted using red colored boxes. In this example, and within the chosen timebox, there exists a rather small amount of time series, indicating that not many households tend to maintain a lower water consumption behavior during Tuesdays.

Another example of the tile map visualization, depicting an area in Manhattan, New York is illustrated in Fig. 12, with $k = 5$ MBRs, a SAX word length of $w = 24$ and a maximum cardinality of $b = 64$. This time, the timebox spans three *i*SAX segments (representing Wednesdays) and a value range of one standard deviation. Since the selected range of taxi dropoffs is rather small, especially considering the fact that Manhattan is a busy area during the whole

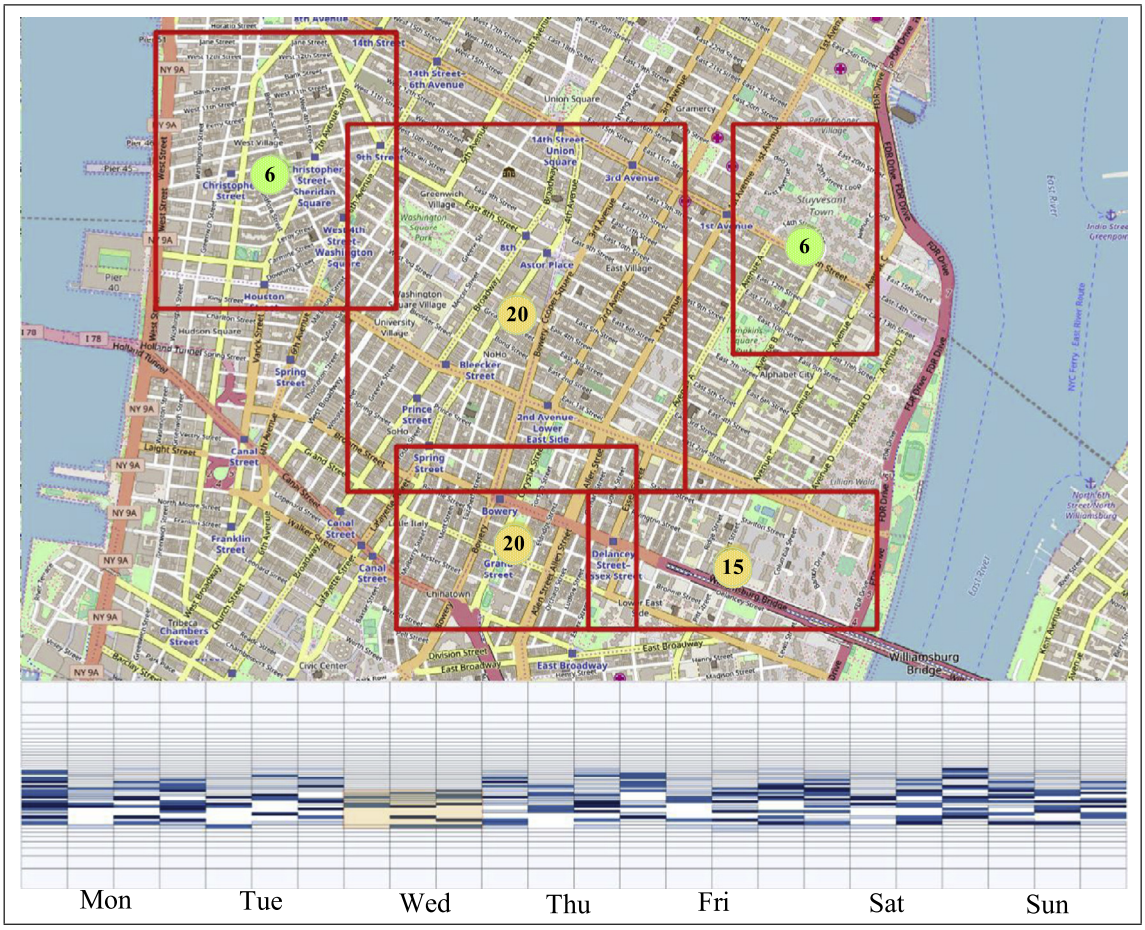


Fig. 12. Visualizing taxi dropoff tile map in Manhattan, NYC (map scale 1:10000).

Table 3
Parameters in tests for tile map summaries.

Parameter	Values
Timebox time range ($\|p_t\|$)	1, **2**, 3, 4, 5
Timebox value range ($\|p_v\|$)	1σ, 1.5σ, **2σ**, 2.5σ, 3σ

week, the depicted MBRs indicate that, during Wednesdays, there are quieter zones in these specific areas. Such information could be leveraged for searching a quieter neighborhood within the city.

6.3.2. Performance evaluation

Parameters Similarly to the bundle summary, we performed preliminary tests against the synthetic dataset to fine-tune the parameters. We built the geo-*i*SAX index setting the maximum number of entries per node to $M = 200$, a maximum cardinality of $b = 512$ and a default SAX word length of $w = 8$. We have set the number of resulting MBRs to $k = 5$ for its leaf nodes. Table 3 lists the range of values for the rest of the parameters.

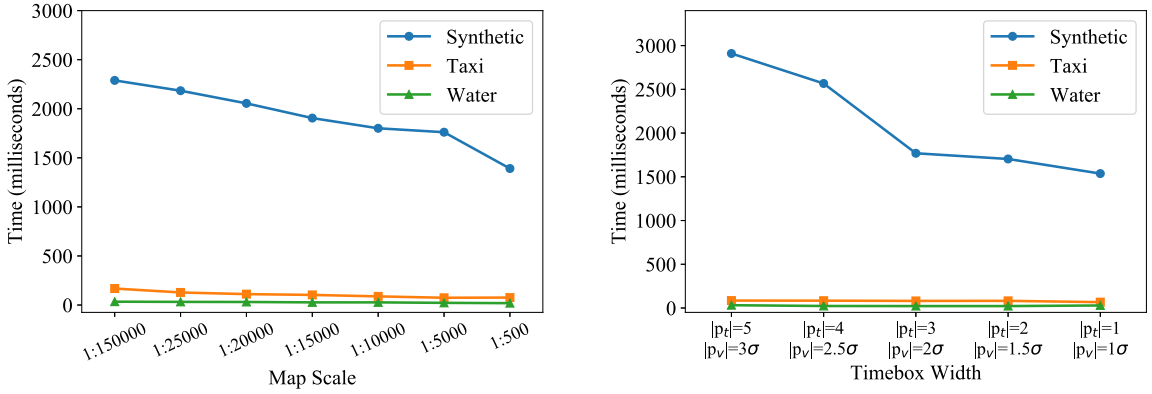
Baseline method for detailed summaries Similarly to the bundle summary, we also compare with a more detailed baseline implementation in terms of accuracy and time required to fetch the results. The spatial part of the detailed summary is generated by performing k-means clustering on the filtered raw time series themselves and generating the resulting MBRs, instead of performing the clustering on each node's resulting MBRs. The tile map of the detailed summary has the exact same structure with the one of tile map summary. The difference among the two tile maps, since in the detailed summary the raw time series are used, lies in the way the count of each tile is augmented. Instead of increasing the count of the tiles where the SAX word's segments of a time series reside, we increase the count of the tiles where each time series point falls within.

Results We evaluate the performance of the tile map summary for different zoom levels and timebox sizes. For the comparison between the two tile maps we use the *root mean squared error* (RMSE) on the difference among the counts between each pair of corresponding tiles. As mentioned, the detailed tile map is generated using the raw time series. This essentially generates tile maps with larger counts, since instead of only incrementing the count once for a SAX word's segment of a time series, it may be incremented multiple times, as each word's segment is derived from n/w time series points (n is the length of time series and w is the SAX word length). To compensate this, we divide the count of each tile of the detailed summary with n/w.

Conclusively, the root mean squared error among two tile maps is calculated as follows:

$$RMSE(tmap, tmap') = \sqrt{\frac{1}{w \times h} \sum_{i=1}^{w} \sum_{j=1}^{h} (c[i,j].cnt - \frac{c'[i,j].cnt}{n/w})^2} \quad (6)$$

where $tmap$ and $tmap'$ are respectively the tile maps from our method (Section 5.3) and the detailed summary, while $c[i,j].cnt$ and $c'[i,j].cnt$ are the corresponding counts of the (i,j) tile in both summaries and h is the number of y-axis breakpoints. For



(a) Response time for different map scales.

(b) Response time for different timebox sizes.

Fig. 13. Response time for different map scales and timebox sizes.

measuring the spatial accuracy, we follow the same rationale as in the evaluation of bundle summaries. Similarly, we evaluate the trade-off between responsiveness and accuracy of our approach. The detailed summary is expected to have a higher accuracy; however, since both spatial and time series summaries are calculated using the raw geolocated time series, we expect a trade-off in responsiveness, especially for larger datasets.

Fig. 13(a) shows the traversal costs for different map scales for each of the three datasets. For the taxi and water datasets, the response time is almost instant due to their smaller sizes, ranging up to at most 100 milliseconds. For the case of the synthetic dataset and due to its high density, the response time is higher, starting from approximately 2.25 seconds for larger spatial areas of interest and falling down to 1.5 seconds as the map scale becomes larger (i.e., smaller areas). The rather higher response time for the tile map summary is due to the fact that in order to be calculated, the geo-*i*SAX's leaf nodes have to be accessed to obtain the locations and maximum cardinality SAX words to generate the summary. However, it should be noted that this is a rather worst case scenario, since the synthetic dataset was generated to be very highly dense as a stress test. The response time is not dramatically reduced for larger map scales, since geo-*i*SAX is a time series-first hybrid index and high overlapping of its MBRs is expected, so larger spatial areas of interest tend to intersect with the MBRs of many nodes, negatively impacting performance. As it is apparent, at a map scale of 1:500, the performance is more abruptly improved as the nodes of the index begin to be more aggressively pruned.

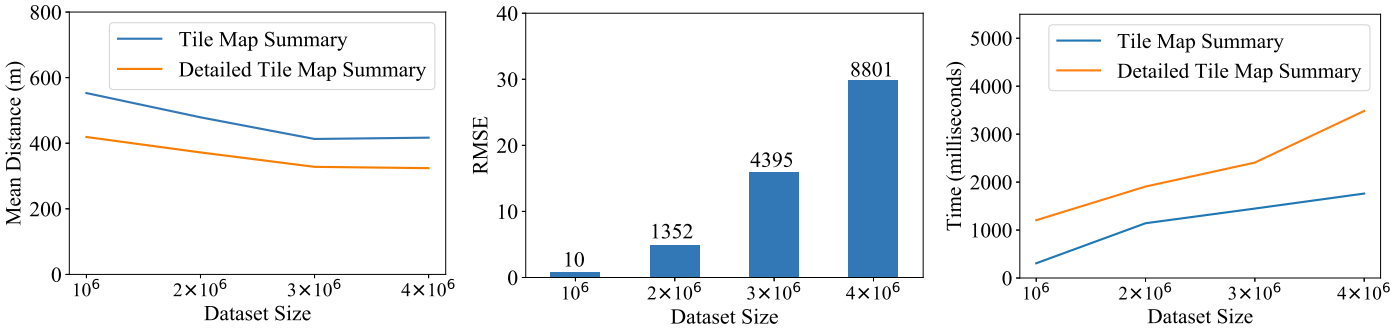
Similar results are also observed for different timebox sizes, as depicted in Fig. 13(b). The taxi and water datasets almost instantly generate the summaries along all timebox sizes. As a reminder, the timebox size is measured in terms of number of SAX segments selected in the time axis and standard deviation range selected in the value axis. In the figure, time range $|p_t|$ denotes the number of SAX segments, whereas value range $|p_v|$ expresses the standard deviation range selected for each timebox. For larger timebox sizes, the index traversal is slower than expected, as more nodes tend to satisfy the rather loose constraints. For the same reasons, the response time is improved up to around 1500 milliseconds as the timebox size gets smaller.

Fig. 14 demonstrates the trade-off between accuracy and responsiveness by comparing the proposed tile map summary and its detailed implementation. As it can be easily observed, the difference between the spatial mean Euclidean distance among the raw data and their closest MBRs (Fig. 14(a)) is between 150 and 200 meters for all dataset sizes. In both cases, it is rather larger for smaller datasets, slowly diminishing as their size gets larger, indicating an improvement of both summaries for larger data. It is worth noting that the difference from the baseline implementation also seems to be slightly reduced for larger datasets. Fig. 14(b) illustrates the RMSE between the two tile maps using Equation 2. The number at the top of each bar is the count of results returned using the same constraints (spatial rectangle, timebox) over each dataset. It is apparent that the quality of the tile map summary worsens for a larger number of participating time series, increasingly diverging from the detailed version. Still, the loss in both spatial and tile map accuracy is compensated in terms of response time as depicted in Fig. 14(c). The detailed implementation requires up to twice the time to generate the summary for all dataset sizes, while its overhead compared to the tile map summary is increasing with larger datasets. Consequently, there is a clear trade-off between accuracy and response time in both domains, allowing an increase in responsiveness without much sacrifice in accuracy.

7. Conclusions and future work

In this paper, we introduced methods for map-based visual exploration over large geolocated time series data. To that end, we proposed two summarization approaches over geolocated time series, which allow a visual analytics application to retrieve the required information. The results can be displayed on a map, depicting the spatial distribution of the data in the form of MBRs for both approaches. Each approach also provides a time series summary, via time series bundles or tile maps respectively. To speed up the retrieval of the results, we employ two hybrid indexing techniques that allow pruning in both the spatial and the time series domains. Our experiments on a large-scale synthetic dataset indicated that the visualizations can be rendered fast, enabling efficient exploration in map-based applications; in the worst case, response time is up to a couple of seconds. Additionally, we examined indicative demonstrations of the visualizations generated from two real-world datasets in different application domains, confirming their helpfulness in jointly exploring both the time series themselves as well as their geographic distribution.

Our ongoing and future work focuses on supporting more detailed visual analytics and identifying more fine-grained patterns through visual exploration. An interesting direction would be to support drilling-down in a particular summarized result and discover whether there are differentiations in the distributions of its constituent, more detailed patterns, both in spatial and time series domains. Moreover, we will focus on supporting more complex time series distance measures that may boost the quality of our summaries.



(a) Accuracy in spatial domain. (b) Accuracy in time series domain. (c) Response time.

Fig. 14. Assessment of tile map summarization.

Acknowledgements

This work was partially funded by the EU H2020 projects SLIPO (731581), SmartDataLake (825041) and the NSRF 2014-2020 project HELIX (5002781).

References

[1] A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, E.J. Keogh, Beyond one billion time series: indexing and mining very large time series collections with iSAX2+, Knowl. Inf. Syst. 39 (1) (2014) 123–151.
[2] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, E.J. Keogh, Querying and mining of time series data: experimental comparison of representations and distance measures, Proc. VLDB Endow. 1 (2) (2008) 1542–1552.
[3] J. Shieh, E.J. Keogh, iSAX: indexing and mining terabyte sized time series, in: SIGKDD, 2008, pp. 623–631.
[4] P. Chronis, G. Giannopoulos, S. Athanasiou, Open issues and challenges on time series forecasting for water consumption, in: EDBT/ICDT Workshops, 2016.
[5] K. Chan, A.W. Fu, Efficient time series matching by wavelets, in: ICDE, 1999, pp. 126–133.
[6] A. Camerra, T. Palpanas, J. Shieh, E.J. Keogh, iSAX 2.0: indexing and mining one billion time series, in: ICDM, 2010, pp. 58–67.
[7] K. Zoumpatianos, S. Idreos, T. Palpanas, Indexing for interactive exploration of big data series, in: SIGMOD, 2014, pp. 1555–1566.
[8] G. Chatzigeorgakidis, D. Skoutas, K. Patroumpas, S. Athanasiou, S. Skiadopoulos, Indexing geolocated time series data, in: SIGSPATIAL, 2017, pp. 19:1–19:10.
[9] A. Guttman, R-trees: a dynamic index structure for spatial searching, in: SIGMOD, 1984, pp. 47–57.
[10] G. Chatzigeorgakidis, D. Skoutas, K. Patroumpas, S. Athanasiou, S. Skiadopoulos, Map-based visual exploration of geolocated time series, in: Proceedings of the Workshops of the EDBT/ICDT 2018 Joint Conference, EDBT/ICDT 2018, Vienna, Austria, March 26, 2018, 2018, pp. 92–99.
[11] A. Graps, An introduction to wavelets, IEEE Comput. Sci. Eng. 2 (2) (1995) 50–61.
[12] I. Popivanov, R.J. Miller, Similarity search over time-series data using wavelets, in: ICDE, 2002, pp. 212–221.
[13] S. Kashyap, P. Karras, Scalable kNN search on vertically stored time series, in: SIGKDD, 2011, pp. 1334–1342.
[14] J. Lin, E.J. Keogh, L. Wei, S. Lonardi, Experiencing SAX: a novel symbolic representation of time series, Data Min. Knowl. Discov. 15 (2) (2007) 107–144.
[15] E.J. Keogh, K. Chakrabarti, M.J. Pazzani, S. Mehrotra, Dimensionality reduction for fast similarity search in large time series databases, Knowl. Inf. Syst. 3 (3) (2001) 263–286.
[16] B. Yi, C. Faloutsos, Fast time sequence indexing for arbitrary Lp norms, in: VLDB, 2000, pp. 385–394.
[17] T. Palpanas, Big sequence management: a glimpse of the past, the present, and the future, in: SOFSEM, 2016, pp. 63–80.
[18] L. Chen, G. Cong, C.S. Jensen, D. Wu, Spatial keyword query processing: an experimental evaluation, Proc. VLDB Endow. 6 (3) (2013) 217–228.
[19] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, T. Suel, Text vs. space: efficient geo-search query processing, in: CIKM, 2011, pp. 423–432.
[20] D. Mintz, T. Fitz-Simons, M. Wayland, Tracking air quality trends with sas/graph, in: Proceedings of the 22nd Annual SAS User Group International Conference, SUGI'97, 1997, pp. 807–812.
[21] D.A. Keim, M. Ankerst, H.-P. Kriegel, Recursive pattern: a technique for visualizing very large amounts of data, in: Proceedings of the 6th Conference on Visualization'95, IEEE Computer Society, 1995, p. 279.
[22] T. Lammarsch, W. Aigner, A. Bertone, J. Gärtner, E. Mayr, S. Miksch, M. Smuc, Hierarchical temporal patterns and interactive aggregated views for pixel-based visualizations, in: Information Visualisation, 2009 13th International Conference, IEEE, 2009, pp. 44–50.
[23] U. Jugel, Z. Jerzak, G. Hackenbroich, V. Markl, M4: a visualization-oriented time series data aggregation, Proc. VLDB Endow. 7 (10) (2014) 797–808.
[24] L. Battle, R. Chang, M. Stonebraker, Dynamic prefetching of data tiles for interactive visualization, in: SIGMOD, 2016, pp. 1363–1375.
[25] E. Wu, L. Battle, S.R. Madden, The case for data visualization management systems: vision paper, Proc. VLDB Endow. 7 (10) (2014) 903–906.
[26] D. Mottin, M. Lissandrini, Y. Velegrakis, T. Palpanas, New trends on exploratory methods for data analytics, Proc. VLDB Endow. 10 (12) (2017) 1977–1980.
[27] B. Eravci, H. Ferhatosmanoglu, Diversity based relevance feedback for time series search, Proc. VLDB Endow. 7 (2) (2013) 109–120.
[28] K. Zoumpatianos, S. Idreos, T. Palpanas, Rinse: interactive data series exploration with ADS+, Proc. VLDB Endow. 8 (12) (2015) 1912–1915.
[29] S. Chan, L. Xiao, J. Gerth, P. Hanrahan, Maintaining interactivity while exploring massive time series, in: IEEE VAST, 2008, pp. 59–66.
[30] R. Neamtu, R. Ahsan, E. Rundensteiner, G. Sarkozy, Interactive time series exploration powered by the marriage of similarity distances, Proc. VLDB Endow. 10 (3) (2016) 169–180.
[31] K. Rong, P. Bailis ASAP, Prioritizing attention via time series smoothing, Proc. VLDB Endow. 10 (11) (2017) 1358–1369.
[32] W. Javed, N. Elmqvist, Stack zooming for multi-focus interaction in time-series data visualization, in: Visualization Symposium, PacificVis, 2010 IEEE Pacific, IEEE, 2010, pp. 33–40.
[33] J. Zhao, F. Chevalier, R. Balakrishnan, Kronominer: using multi-foci navigation for the visual exploration of time-series data, in: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, ACM, 2011, pp. 1737–1746.
[34] J. Zhao, F. Chevalier, E. Pietriga, R. Balakrishnan, Exploratory analysis of time-series with chronolenses, IEEE Trans. Vis. Comput. Graph. 17 (12) (2011) 2422–2431.
[35] H. Hochheiser, B. Shneiderman, Dynamic query tools for time series data sets: timebox widgets for interactive exploration, Inf. Vis. 3 (1) (2004) 1–18.
[36] E. Keogh, H. Hochheiser, B. Shneiderman, An augmented visual query mechanism for finding patterns in time series data, in: International Conference on Flexible Query Answering Systems, Springer, 2002, pp. 240–250.
[37] A. Aris, B. Shneiderman, C. Plaisant, G. Shmueli, W. Jank, Representing unevenly-spaced time series data for visualization and interactive exploration, in: IFIP Conference on Human–Computer Interaction, Springer, 2005, pp. 835–846.
[38] J. Paparrizos, L. Gravano, k-Shape: efficient and accurate clustering of time series, in: SIGMOD, 2015, pp. 1855–1870.
[39] P. Rigaux, M. Scholl, A. Voisard, Spatial Databases with Application to GIS, Morgan Kaufmann Publishers Inc., 2002.
[40] H. Hochheiser, B. Shneiderman, Interactive exploration of time series data, in: The Craft of Information Visualization, Elsevier, 2003, pp. 313–315.
[41] J. Aschoff, Circadian rhythms in man, Science 148 (3676) (1965) 1427–1432.