Open access • Journal Article • DOI:10.1007/S10115-020-01518-4

BestNeighbor: efficient evaluation of kNN queries on large time series databases

Oleksandra Levchenko, Boyan Kolev, Djamel Edine Yagoubi, Reza Akbarinia ...+4 more authors

Institutions: University of Montpellier, University of Paris, New York University

Published on: 01 Feb 2021 - Knowledge and Information Systems (Springer London)

Topics: Pruning (decision trees)

BestNeighbor: Efficient Evaluation of kNN Queries on Large Time Series Databases

Oleksandra Levchenko, Boyan Kolev, Djamel-Edine Yagoubi, Reza Akbarinia, Florent Masseglia, Themis Palpanas, Patrick Valduriez, Dennis Shasha

Noname manuscript No.
(will be inserted by the editor)

BestNeighbor: Efficient Evaluation of kNN Queries on Large Time Series Databases

Oleksandra Levchenko · Boyan Kolev · Djamel-Edine Yagoubi · Reza Akbarinia · Florent Masseglia · Themis Palpanas · Dennis Shasha · Patrick Valduriez

Received: date / Accepted: date

Abstract This paper presents parallel solutions (developed based on two state-of-the-art algorithms iSAX and sketch) for evaluating kNN (k nearest neighbor) queries on large databases of time series, compares them based on various measures of quality and time performance, offers a tool that uses the characteristics of application data to determine which algorithm to choose for that application and how to set the parameters for that algorithm. Specifically, our experiments show that: (i) iSAX and its derivatives perform best in both time and quality when the time series can be characterized by a few low frequency Fourier Coefficients, a regime where the iSAX pruning approach works well. (ii) iSAX performs significantly less well when high frequency Fourier Coefficients have much of the energy of the time series. (iii) A random projection approach based on sketches by contrast is more or less independent of the frequency power spectrum. The experiments show the close relationship between pruning ratio and time for exact iSAX as well as between pruning ratio and the quality of approximate iSAX. Our toolkit analyzes typical time series of an application (i) to determine optimal segment sizes for iSAX and (ii) when to use Parallel Sketches instead of iSAX. Our algorithms have been implemented using Spark, evaluated over a cluster of nodes, and have been applied to both real and synthetic data. The results apply to any databases of numerical sequences, whether or not they relate to time.

Keywords Time Series · Parallel Indexing · Distributed Querying · Fourier Coefficients · Frequency Analysis · Filtering

Oleksandra Levchenko, Boyan Kolev, Djamel Edine Yagoubi,
Reza Akbarinia, Florent Masseglia,Patrick Valduriez
Inria, University of Montpellier, CNRS, LIRMM, France
E-mail: firstname.lastname@inria.fr

Themis Palpanas
Université de Paris, France
E-mail: themis@mi.parisdescartes.fr

Dennis Shasha
Dep. of Computer Sc., New York University
E-mail: shasha@cs.nyu.edu

1 Introduction

Nowadays people are able to monitor various indicators for their personal activities (*e.g.*, through smart-meters or smart-plugs for electricity or water consumption), or professional activities (*e.g.*, through the sensors installed on plants by farmers). Sensor technology is also improving over time and the number of sensors is increasing, *e.g.*, in finance and seismic studies. This results in the production of large and complex data, usually in the form of time series (or *TS* in short) [21, 41, 49, 34, 35, 16, 37, 31, 27, 30] that challenge knowledge discovery.

With such complex and massive sets of time series, fast and accurate similarity search is critical to performing many data mining tasks like Shapelets, Motifs Discovery, Classification or Clustering [34, 29, 52].

In order to improve the performance of such similarity queries, indexing [12, 32, 10, 11] has been successfully used in a variety of settings and applications [13, 43, 2, 46, 6, 50, 23, 28].

In this work, we focus on two distributed time series indexing methods of quite different natures – a hash-based and a tree-based one.

The hash-based method ParSketch follows a locality-sensitive hashing (LSH) strategy that uses a number of grid structures to hash similar items to the same buckets with high probability. ParSketch processes a query time series by performing a number of hash lookups to identify candidate neighbors, which are then verified by explicit distance computation.

By contrast, the tree-based method DPiSAX builds an index tree using multiresolution symbolic representations of time series items, so that leaf nodes are represented by high resolution symbols. Thus, candidate neighbors for a query are found by traversing the tree down to the leaf node whose symbolic representation is closest to the representation of the query.

The two methods are distributed, which allows us to examine their abilities to scale to a large number of time series and operate in a distributed data processing framework, such as Apache Spark. However, the different nature of the two methods leads to different partitioning approaches and method-specific parallelization of the query processing.

The partitioning of DPiSAX uses the same multiresolution property as centralized iSAX and is based on representations of time series items at some basic resolution. This means that a single query is first quickly routed to one of the partitions using that basic resolution, where the rest of the search down the tree is done locally at the partition. So, each query is performed serially, at one partition of the index.

In ParSketch, the input dataset is simply split into disjoint subsets of equal sizes, each assigned to a partition, where local grid structures are built. This means that a single query is broadcast to all partitions, where candidate subsets are searched locally. So, the processing of one query makes hash lookups in all partitions of the index, but is done in parallel.

The major objective of both methods is to do fast approximate search with high quality results. So, one of our goals is to evaluate the quality of the results in relationship to the response time. Moreover, DPiSAX enables exact search as a second step, where it uses the lower bounding properties of the symbolic representation to prune candidates based on the current best-so-far neighbors.

Our experiments show that these pruning capabilities are sensitive to the frequency spectrum of the input dataset. DPiSAX pruning seems to be quite efficient when energy is concentrated in the first few Fourier coefficients (which is the case of random walks). Exact DPiSAX is slow (and approximate DPiSAX is inaccurate) when energy is spread across the frequency spectrum (the extreme case of which is white noise), yielding nearly all available time series as candidates.

By contrast, ParSketch is less sensitive to the energy distribution across Fourier coefficients. This can be explained by the different dimensionality reduction techniques used by the two methods: ParSketch is based on random projection, while DPiSAX uses piecewise aggregate approximation (PAA) as a first step to reduce dimensionality. If there are enough high frequency components, the time series will oscillate substantially within a piece, but the approximation (which is based on the mean within the piece) will not be sensitive to such oscillations.

This paper compares four parallel methods for solving nearest neighbor queries: (i) parallel linear search in which we compare the query time series directly with every time series in the database, (ii) Exact distributed iSAX (to be described below), (iii) Approximate distributed iSAX, (iv) and the random sketch approach ParSketch. Methods (ii, iii, and iv) all require indexes so perform best when there are many queries against an unchanging database.

The rest of this paper is organized as follows. In Section 2, we define the problem we address in the paper and present the related background. In Section 3 and Section 4, we describe the details of our parallel index construction and query processing algorithms. Section 5 introduces our approach to suggest an indexing method by analyzing the frequency characteristics of the time series. In Section 6, we present a detailed experimental evaluation for comparing our approaches. In Section 7, we discuss the related work. Finally, we conclude in 8.

2 Problem Definition and Background

2.1 Time series and kNN query

A time series X is a sequence of values $X = \{x_1, ..., x_n\}$. We assume that every time series has a value at every timestamp $t = 1, 2, ..., n$. The length of X is denoted by $|X|$. Figure 1a shows a time series of length 16, which will be used as running example throughout this paper. (Note that while time series are our main use case, any database and queries on sequences of numerical values can use the tools described in this paper.)

Given two time series (or vectors) of real numbers, $X = \{x_1, ..., x_n\}$ and $Y = \{y_1, ..., y_m\}$ such that $n = m$, the Euclidean distance between X and Y is defined as [13]: $ED(X, Y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$. The Euclidean distance is one of the main similarity measurement methods used in time series analysis. In this work, we assume that the distance between the time series is measured by using the Euclidean function.

The problem of similarity queries is one of the most important problems in time series analysis and mining. We construe "similarity" to mean finding the k nearest neighbors (k-NN) of a query.

Definition 1 (Exact k nearest neighbors) Given a query time series Q and a set of time series D, let $R = ExactkNN(Q, D)$ be the set of k nearest neighbors

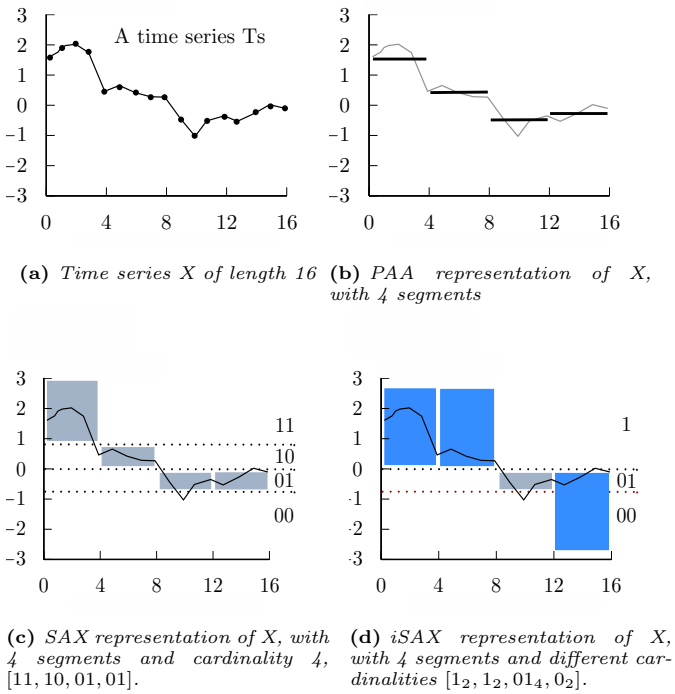
of Q from D. Let $ED(X, Y)$ be the Euclidean distance between the points X and Y, then the set R is defined as follows:

$$(R \subseteq D) \wedge (|R| = k) \wedge (\forall a \in R, \forall b \in (D - R), ED(a, Q) \leq ED(b, Q))$$

Definition 2 (Approximate k nearest neighbors) Given a set of time series D, a query time series Q, and $\epsilon > 0$. We say that $R = AppkNN(Q, D)$ is the approximate k nearest neighbors of Q from D, if $ED(a, Q) \leq (1 + \epsilon)ED(b, Q)$, where a is the k^{th} nearest neighbor from R and b is the true k^{th} nearest neighbor.

3 Parallel iSAX-based kNN Search

In this section, we first describe the iSAX representation [42], and then present DPiSAX, our parallel iSAX-based solution for kNN search over large time series datasets.



(a) *Time series X of length 16* **(b)** *PAA representation of X, with 4 segments*

(c) *SAX representation of X, with 4 segments and cardinality 4,* $[11, 10, 01, 01]$. **(d)** *iSAX representation of X, with 4 segments and different cardinalities* $[1_2, 1_2, 01_4, 0_2]$.

Fig. 1: *A time series X is discretized by obtaining a PAA representation and then using predetermined break-points to map the PAA coefficients into SAX symbols. Here, the symbols are given in binary notation, where 00 is the first symbol, 01 is the second symbol, etc. The time series of Figure 1a in the representation of Figure 1d is [fourth, third, second, second] (which becomes* $[11, 10, 01, 01]$ *in binary). The representation of that time series in Figure 1c becomes* $[1_2, 1_2, 01_4, 0_2]$*, where* $1_{(2)}$ *means that* 1 *is the selected symbol among 2 possible choices, 01 is the selected symbol among 4 possible choices, etc.*

3.1 iSax Representation

For very large time series databases, it is important to estimate the distance between two time series very quickly. There are several techniques, providing lower bounds by segmenting time series. One popular method, is called indexable Symbolic Aggregate approXimation (iSAX) representation [42,43]. The iSAX representation is used to represent time series in our index.

The iSAX representation extends the SAX representation [26]. This latter representation is based on the PAA representation [25] which allows for dimensionality reduction while providing an important lower bounding property. The idea of PAA is to have a fixed segment size, and minimize dimensionality by using the mean values of each segment. Example 1 gives an illustration of PAA.

Example 1 Figure 1b shows the PAA representation of X, the time series of Figure 1a. The representation is composed of $w = |X|/l$ values, where l is the segment size. For each segment, the set of values is replaced with their mean. The length of the final representation w is the number of segments (and, usually, $w << |X|$).

The SAX representation takes as input the reduced time series obtained using PAA. It discretizes this representation into a predefined set of symbols, with a given cardinality, where a symbol is a binary number. The cardinality of a symbol is the number of possible distinct values it can take. Example 2 gives an illustration of the SAX representation.

Example 2 In Figure 1c, we have converted the time series X to SAX representation with size 4, and cardinality 4 using the PAA representation shown in Figure 1b. We denote SAX(X) = [11, 10, 01, 01].

The iSAX representation uses a variable cardinality for each symbol of SAX representation, each symbol is accompanied by a number that denotes its cardinality. We defined the iSAX representation of time series X as $iSAX(X)$ and we call it the iSAX word of the time series X. For example, the iSAX word shown in Figure 1d can be written as $iSAX(X) = [1_2, 1_2, 01_4, 0_2]$.

The lower bounding approximation of the Euclidean distance for iSAX representation $iSAX(X) = \{x'_1, ..., x'_w\}$ and $iSAX(Y) = \{y'_1, ..., y'_w\}$ of two time series X and Y is defined as [42]:

$$MINDIST(iSAX(X), iSAX(Y)) = \sqrt{\frac{n}{w}} \sqrt{\sum_{i=1}^{w} (dist(x'_i, y'_i))^2}$$

, where the function $dist(x'_i, y'_i)$ is the distance between two iSAX symbols x'_i and y'_i. The lower bounding condition is formulated as:

$$MINDIST(iSAX(X), iSAX(Y)) \leq ED(X, Y)$$

Using a variable cardinality allows the iSAX representation to be indexable. We can build a tree index as follows. Given a cardinality b, an iSAX word length w and leaf capacity th, we produce a set of b^w children for the root node, insert the time series to their corresponding leaf, and gradually split the leaves by increasing the cardinality by one character if the number of time series in a leaf node rises above the given threshold th.

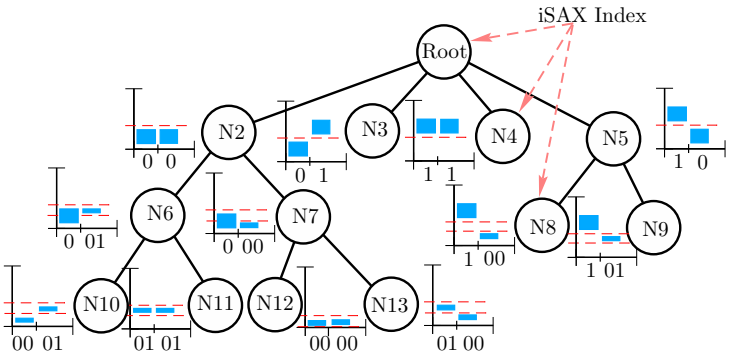


Fig. 2: *Example of iSAX Index*

Example 3 Figure 2 illustrates an example of iSAX index, where each iSAX word has 2 symbols and a maximum cardinality of 4. The root node has 2^2 children while each child node forms a binary sub-tree. There are three types of nodes: *root node*, *internal node* (N2, N5, N6, N7) and *terminal node or leaf node* (N3, N4, N8, N9, N10, N11, N12, N13). Each leaf node links to a disk file that contains the corresponding time series (up to *th* time series).

Note that previous studies have shown that the iSAX index is robust with respect to the choice of parameters (word length, cardinality, leaf threshold) [43, 6,51]. Moreover, it can also be used to answer queries with the Dynamic Time Warping (DTW) distance, through the use of the corresponding lower bounding envelope [22].

3.2 DPiSAX

Here, we describe DPiSAX (Distributed Partitioned iSAX) [48], our parallel solution for constructing a parallel iSAX-based index over large sets of time series by making the most of the parallel environment and carefully distributing the workload.

DPiSAX is based on a sampling phase that allows anticipating the distribution of time series among the computing nodes. Such anticipation is mandatory for efficient query processing, since it will allow, later on, to decide which partition contains the time series that actually correspond to the query. DPiSAX splits the full dataset for distribution into partitions using the partition table constructed at the sampling stage. Then each worker builds an independent iSAX index on its partition, with the iSAX representations having the highest possible cardinalities. Alongside an efficient node splitting policy, it allows to preserve index tree's load balance and to improve query performance, hence taking full advantage of time series indexing in distributed environments.

3.2.1 Index construction

The index construction by DPiSAX proceeds in two steps.

- Partitioning. The first stage is done as follows. Given a desired number of partitions P and a time series dataset D, the algorithm takes a sample S of size L time series from D using stratified sampling, and emits its iSAX words $SWs = \{iSAX(ts_i), i = 1, ..., L\}$, initially assigned to a single partition. At each iteration, DPiSAX divides the sample by splitting the biggest partition into two sub-partitions, until the number of partitions reaches P, using the same splitting policy, as defined for iSAX tree index construction. As a result, the binary tree is converted to a partition table, where each leaf-node represents a partition, described with its iSAX word, and will become a root node of each sub-tree of the distributed index.
- Parallel sub-index creation. On the next stage, the input dataset D, is mapped to iSAX representation $DWs = \{iSAX(ts_i), i = 1, ..., N\}$ with the highest possible cardinalities of each symbol. Then, the database partitions are distributed among the available workers. Each worker builds locally its iSAX index sub-tree on the given partition of time series and stores it on HDFS in JSON format. Alongside, each leaf node (terminal node) is stored to HDFS as a file with corresponding time series ids and their iSAX representations.

3.2.2 Query processing

Given a collection of queries Q, in the form of time series, and the index constructed in the previous section for a database D, we consider the problem of finding time series in D that are similar to Q, according to the definitions of approximate k-NN and exact k-NN searches:

- **Approximate search**: Given a batch of queries Q, DPiSAX approximate search starts by obtaining the iSAX representations of all queries time series using the highest possible cardinalities. The master identifies the target partition for each query by checking the query's iSAX representation with the iSAX word of each entry in the partition table and sends the query to the worker in charge of the target partition. On each local index (sub-tree), the approximate search is done by traversing the local index to the terminal node that has the same iSAX representation as the query. The target terminal node contains at least one and at most th iSAX words, where th is the leaf threshold.
  On the next stage of search a parallel computation of the Euclidean distance is performed in order to obtain the k nearest neighbors among all the candidates for a queries in a batch, retrieved from terminal nodes.
- **Exact search**:
  The exact search uses the approximate search result $AKNN$ for a batch of queries Q as best-so-far k nearest neighbors. This result is broadcast among the workers in order to examine the index sub-trees that may contain the time series that are probably more similar to Q than those of $AKNN$. The lower bound distance $MINDIST$ is computed for each node on the sub-tree to determine nodes with probable closer candidates. If the lower bound of a node is already higher than the current best-so-far, the node and its sub-tree are excluded from the search.
  The k nearest neighbors are found by computing direct correlations on the candidates subset for each query in parallel.

4 Parallel Sketch-based kNN Search

This section presents *ParSketch*, our sketch-based parallel solution for indexing and similarity search over big time series datasets.

4.1 The Sketch Approach

The sketch approach, as developed by Kushilevitz et al. [24], Indyk et al. [17], and Achlioptas [1], provides a very nice guarantee: with high probability a random mapping taking b points in R^m to points in $(R^d)^{2b+1}$ (the (2b+1)-fold cross-product of R^d with itself) approximately preserves distances (with higher fidelity the larger b is).

In our version of this idea, given a point (a time series or a window of a time series) $\mathbf{t} \in R^m$, we compute its dot product with N random vectors $\mathbf{r}_i \in \{1, -1\}^m$. This results in N inner products called the *sketch* (or random projection) of t_i. Specifically, $sketch(t_i) = (\mathbf{t_i} \bullet \mathbf{r_1}, \mathbf{t_i} \bullet \mathbf{r_2}, ..., \mathbf{t_i} \bullet \mathbf{r_N})$. We compute sketches for $t_1, ..., t_b$ using the same random vectors $r_1, ..., r_N$.

The theoretical underpinning for the utilization of sketches is given by the Johnson-Lindenstrauss lemma [20].

Lemma 1 *Given a collection C of m time series, for any two time series $\overrightarrow{x}, \overrightarrow{y} \in C$, if $\epsilon < 1/2$ and $n = \frac{9logm}{\epsilon^2}$, then*

$$(1 - \epsilon) \leq \frac{\| \overrightarrow{s}(\overrightarrow{x}) - \overrightarrow{s}(\overrightarrow{y}) \|^2}{\| \overrightarrow{x} - \overrightarrow{y} \|} \leq (1 + \epsilon)$$

holds with probability $1/2$, where $\overrightarrow{s}(\overrightarrow{x})$ is the sketch of $\overrightarrow{x}$ of at least n dimensions.

The Johnson-Lindenstrauss lemma implies that the distance $\|\mathbf{sketch(t_i)} - \mathbf{sketch(t_j)}\|$ is a good appproximation of $\|\mathbf{t_i} - \mathbf{t_j}\|$ provided the dimensionality of the sketches (r) is large enough. Specifically, if $\|\mathbf{sketch(t_i)} - \mathbf{sketch(t_j)}\| < \|\mathbf{sketch(t_k)} - \mathbf{sketch(t_m)}\|$, then it's likely that $\|\mathbf{t_i} - \mathbf{t_j}\| < \|\mathbf{t_k} - \mathbf{t_m}\|$, because the ratio between the sketch distance and the real distance is close to one.

A sketch of a time series t is a vector of dot products: element i of the sketch is the dot product between t and the ith random vector. Thus the full sketch contains as many dot products as there are random vectors.

The data structure consists of a set of grids. Each grid maintains the sketch values corresponding to the dot products between a specific set of random vectors and all time series. Let $|g|$ be the number of random vectors assigned to each grid, and N be the total number of random vectors, then the total number of grids is $b = N/|g|$. (We make sure that $|g|$ divides N.) The distance between two time series in different grids may differ. We consider two time series similar if they are similar in a given (large) fraction of grids.

Example 4 Let's consider two time series t_1=(2, 2, 5, 2, 6, 5) and t_2=(2, 1, 6, 5, 5, 6). Suppose that we have generated four random vectors as follows : r_1=(1, -1, 1, -1, 1, 1), r_2=(1, 1, 1, -1, -1, 1), r_3=(-1, 1, 1, 1, -1, 1) and r_4=(1, 1, 1, -1, 1, 1). Then the sketches of t_1 and t_2, *i.e.* the inner products computed as described above, are

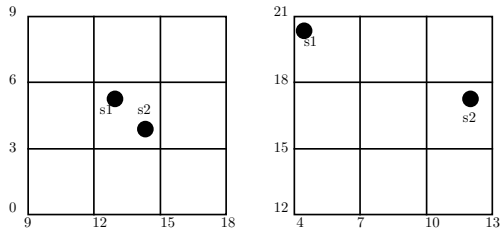


Fig. 3: *Two series (s_1 and s_2) may be similar in some dimensions (here, illustrated by $Grid_1$) and dissimilar in other dimensions ($Grid_2$). The higher the similarity between t_1 and t_2, the larger the fraction of grids in which the series are close.*

respectively s_1=(14, 6, 6, 18) and s_2=(13, 5, 11, 15). In this example, we create two grids, $Grid_1$ and $Grid_2$, as depicted in figure 3. $Grid_1$ is built according to the sketches calculated with respect to vectors r_1 and r_2 (where t_1 has sketch values 14 and 6 and t_2 has sketch values 13 and 5). In other words, $Grid_1$ captures the values of the sketches of t_1 and t_2 on the first two dimensions (vectors). $Grid_2$ is built according to vectors r_3 and r_4 (where t_1 has sketch values 6 and 18 and t_2 has sketch values 11 and 15). Thus, $Grid_2$ captures the values of the sketches on the last two dimensions. We observe that t_1 and t_2 are close to one another in $Grid_1$. On the other hand, t_1 and t_2 are far apart in $Grid_2$.

4.1.1 Partitioning Sketch Vectors

Multi-dimensional search structures don't work well for more than four dimensions in practice [40]. For this reason, as indicated in Example 4, we adopt a first algorithmic framework that partitions each sketch vector into subvectors and builds grid structures for the subvectors as follows:

- Partition each sketch vector s of size N into groups of some size $|g|$.
- The ith group of each sketch vector s is placed in the ith grid structure (of dimension $|g|$).
- If two sketch vectors s and s' are within distance $c \times d$ in more than a given fraction f of the groups, then the corresponding time series are candidate highly correlated time series and should be checked exactly.

For example, if each sketch vector is of length $N = 40$, we might partition each one into ten groups of size $|g| = 4$. This would yield 10 grid structures, where time series items are assigned to grid cells, so that close items are grouped in the same grid cells. Suppose that the fraction f is 90%, then a time series t is considered as similar to a database time series t', if they are similar (assigned to the same cell) in at least nine grids.

Grid granularity can be adjusted to control the tradeoff between efficiency and accuracy. Coarser grids have larger grid cells (i.e. more time series assigned to the same cell), which leads to a larger number of candidates to process (slower execution), but lower probability to miss a true positive (higher accuracy). The grid granularity is defined through the parameter *grid_size* that specifies the number of cells per grid dimension. At the cell assignment step, grids are divided into cells

in a way that results in a uniform distribution of items across grid cells. This is supported by a sampling phase that infers the distribution and defines the cell borders along each dimension of each grid.

4.2 ParSketch

Our ParSketch solution takes full advantage of parallel data processing, both while constructing indexes and querying them for time series similarity search.

4.2.1 Index construction

Time-series index construction on the input dataset D within distributed data processing frameworks proceeds as follows:

1) A random transformation matrix R (composed of N random vectors) is generated, where each element $r_{i,j} \in R$ is a random value in $\{1, -1\}$. This matrix will be used to compute the sketch (of size N) of each time series $t \in D$, by computing the dot product of t with R. Once generated, R is broadcast to all worker nodes.

2) ParSketch strives to place an approximately equal number of time series within each cell of each grid. The borders of these variable sized cells are defined by a *breakpoint table*. This is done by sampling: ParSketch takes a sample S of size L time series from the input dataset D. Based on random projection of S with the random transformation matrix R, ParSketch defines the cell breakpoints B_G for each dimension at each grid G, considering the parameter $grid\_size$ and the distribution of values at each of the N sketch dimensions. The resulting breakpoints table is also broadcast to worker nodes.

3) At the sketch computation stage the dot product of time series t, where $t \in D$, with the random transformation matrix R results in a vector of much lower dimension: $s_j = t_j \times R$. The input dataset D is partitioned horizontally, so that each time series $t \in D$ is entirely handled at the same worker node. Then, at each node, sketch vectors over D are built locally and then split into equal subvectors of given size. Each subvector corresponds to a grid. Thus, each sketch is assigned to a grid cell in each of the grids G using the breakpoints B_G, defined at the sampling stage.

4) We use a cluster of relational database instances, previously created and distributed at the nodes, to persist the indexed time series. One database instance stores the indexing of one partition of D into a relation with the structure ($grid\_id$, $cell\_id$, t_id), *i.e.,* the cell assignment at each grid of a given time series t with identifier t_id. Thus, the entire contents of a particular cell in a particular grid is spread across index nodes.

4.2.2 Query processing

Given a collection of queries Q, in the form of distributed time series dataset, and a previously constructed index on a dataset D, we consider the problem of finding time series that are similar to Q in D. We perform such a search in the following steps:

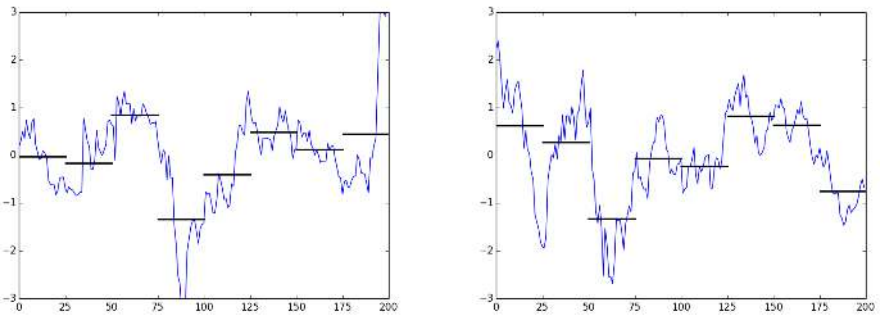


Fig. 4: *Two normalized time series of finance stock prices (and their PAA representations) with substantial power at lower frequency components. The iSAX symbols corresponding to these PAA values can distinguish different time series from one another.*

1) Similarly to the grid construction, at each node, sketches over Q are computed in parallel, using the same random matrix R. Sketches are then split into subvectors in parallel and assigned to grid cells at each grid G, using the same breakpoints sequences B_G. The resulting cell assignments of all queries are then broadcast to worker nodes.

2) Each node checks the cell assignments of all queries against its partition of the index. Then, the full list of candidates for each query is retrieved in parallel by all worker nodes from their database instances. If a subvector of a query time series q lands in the same grid cell as a database time series t, then t is a possible match for q. Two time series (a query and a database one whose sketch is stored in the grids) are considered to be similar if they are assigned to the same grid cell in a large user-tunable fraction of grids. The candidates that meet these conditions are filtered in a SELECT statement to each database, thus efficiently reducing the data to be processed at early stages of the search. For instance, the entire filtering of candidates for a given query q is pushed down to the databases and done through the parallel execution of the aforementioned SELECT statement, which selects all time series t_i collocated with q in grid cells, then groups by t_i and counts, in order to finally emit those t_i that are collocated with q in at least the desired fraction f of grids.

3) Because sketching is approximate, each candidate match between a query q and data vector t is checked by performing direct similarity computation. The kNN results of a query q are the k candidates that have the highest similarity to it.

5 Selecting the Best Indexing Approach based on Frequency Coefficients

To choose the best indexing and querying approach between DPiSAX and ParSketch for a given dataset, we have developed a tool called *BestNeighbor*. It makes its decision based on the Fourier coefficients derived from the discrete Fourier Transform of the data.

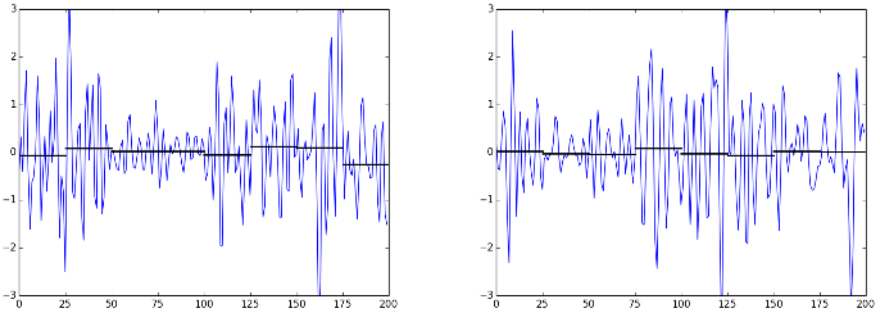


Fig. 5: *Two quite different time series (normalized) from the seismic dataset (and their PAA representations) with substantial power at higher frequency components. The PAA values are all close to 0, masking the differences.*

We start from the observation that the absolute value of the i^{th} coefficient of the discrete Fourier transform of a time series represents the amplitude of a wave signal component whose full cycle repeats i times through the entire series (we also refer to this signal as the i^{th} *frequency component*). Intuitively (and in fact), this relates to how the series should be divided into PAA segments for iSAX. In particular, having fewer than i segments would fail to capture useful piecewise summarization of the i^{th} frequency component, because one PAA segment would correspond to more than one full cycle of the component. However, there is a cost to having too many segments because the iSAX data structure might then grow significantly.

Experiments back this up. DPiSAX performs extremely well in terms of both quality and response time when the data resembles a random walk, even when segment sizes are large, with most of the energy in the low Fourier coefficients. One might call that the *iSAX-friendly* regime. As the energy spreads to higher Fourier coefficients, iSAX pruning becomes less effective for large segment sizes, leading to a higher time burden for exact iSAX and lower precision for approximate iSAX. In those regimes, the sketch approach offers quite high precision (over 90% across both simulated and real data sets) and a time performance far below that of exact iSAX. That would be the *iSAX-unfriendly* regime.

For a given number of segments, the frequency composition of a dataset has a big impact on performance. For example, Figure 4 shows two time series of stock prices whose spectrum concentrates energy in the first 4 frequency components. The PAA of the 8 segments take values from a quite wide range, which leads to a good variety of iSAX representations of time series with similar spectral characteristics. By contrast, the two seismic series on Figure 5 are characterized by higher frequencies, up to 53. One can easily notice that the mean of each segment tends to be close to 0 (assuming z-normalization), which significantly restricts the possible iSAX grammar for that class of time series. This inevitably leads to poor pruning of candidates, as the iSAX representations of any two (even very distant) time series tend to show significant similarity.

To analyze the frequency spectrum of a time series dataset D for iSAX-friendliness, we use the following strategy:

1. Take a sample S of the input dataset D.
2. For each series t in S, apply the Fast Fourier Transform (FFT). The energy distribution of t is a vector $E(t)$ of energy components, where each element $E_i(t)$ is the square of the absolute value of the i^{th} Fourier coefficient, i.e. proportional to the energy of the i^{th} frequency component.
3. The averages of each energy component over all series constitute the mean energy distribution $E(S)$ for the entire sample S, which we consider a good enough approximation of the energy distribution $E(D)$ of the input dataset D.
4. Sort $E(D)$ in descending order and take the highest energy components $HEC(D)$ that hold $p\%$ of the total energy. The parameter $p\%$ is configurable by the user. By default, it is set to 80%. The least and greatest frequency indexes in $HEC(D)$, l_D and g_D respectively, indicate the range of Fourier coefficients that concentrates the majority of the energy.

Low values of g_D, meaning that high frequencies do not have significant energy impact, are generally favorable for iSAX. By contrast, when g_D is greater than the number of segments, the problem depicted at Figure 5 occurs. In such cases, increasing the number of segments can help increase the diversity of iSAX symbols, hence improve the pruning to some reasonable values.

Our experiments (detailed in the next section) on different synthetic datasets with diverse bandwidths (g_D), varying the number of iSAX segments, show that the pruning ratio (the fraction of the dataset filtered out by approximate iSAX to ease the exact search) stays close to zero until the number of segments reaches some $minseg(g_D)$, where it seems to take off and start making iSAX exact search useful. To approximate $minseg(g_D)$, we made a simple regression analysis of the results from our measurements, arriving at the following formula (detailed explanations presented in Section 6.3):

$$minseg(g_D) = \frac{(7g_D - 20)}{4}$$

For a reasonable upper bound on the number of segments, we assessed the pruning ratio, setting the number of segments to $2g_D$, inspired by the Nyquist sampling rate, which by definition is twice the highest frequency of a signal (and, as Nyquist showed, is high enough to reconstruct the signal). This generally leads to a very good pruning, hence fast exact search. However, for higher values of g_D, this entails slow indexing and requires more space for storing the index, since the size of the index structure grows with the number of segments. In such cases, the user may choose a number of segments between $minseg(g_D)$ and $2g_D$, to optimize the tradeoff between indexing time and exact search time.

BestNeighbor does a Fourier analysis of any dataset to determine where most of the power is. If there is substantial power at least up to the 30^{th} coefficient, then the dataset is in an "iSAX-unfriendly" regime, so using ParSketch instead would be recommended. Otherwise, the tool outputs the recommended range of values for the number of segments that may lead to good iSAX indexing and exact search times.

The *BestNeighbor* tool finds the highest energy components that concentrate the bulk of the energy. In the current version of *BestNeighbor* and based on our experiments, we set the parameter $p\%$ to 80%, *i.e.*, to choose the components that hold 80% of the total energy. This works well in our experiments. However, if

this default value is not appropriate for the user's dataset, he/she can change the parameter accordingly. Testing on samples of the data may help set this parameter, though we suspect the default will work well.

The time complexity for the analysis of frequency spectrum is $O(a * b \log b)$, where a - size of input sample dataset, b - length of 1 time series from dataset. To determine the type of dataset in terms of frequency composition and to get recommendations on which indexing algorithm to use, the *BestNeighbor* tool requires relatively small input. In our experiments we used dataset samples of size up to 1000 time series ($a \leq 1000$), 200-256 points each ($b \leq 256$). Hence dataset analysis time complexity will be insignificant in the full cycle of indexing large time series data.

6 Performance Evaluation

In this section, we report experimental results for comparing the quality and the performance of DPiSAX and ParSketch for indexing different time series datasets.

6.1 Setup

Our experiments were conducted on a cluster[1] of 16 compute nodes each having two 8 core Intel Xeon E5-2630 v3 CPUs, 128 GB RAM, 2x558GB capacity storage per node. The cluster is running under Hadoop version 2.7, Spark v. 2.4 and PostgreSQL v. 9.4 as a relational database system.

Compared approaches. We evaluate the performance of four kNN search approaches: 1) ParSketch; 2) Exact version of DpiSAx; 3) Approximate version of DPiSAX; 4) Parallel Linear Search (PLS), which is a parallel version of the UCR Suite fast sequential search (with all applicable optimizations in our context: no computation of square root, and early abandoning) [34].

We performed our performance evaluation using different synthetic and real datasets.

Synthetic datasets. For the purpose of experimentation, we generated several synthetic data sets:

1. Random Walk input dataset: whose sizes/volumes vary between 50 million and 500 million time series each of size 256 time points. At each time point, a random walk generator cumulatively adds to the value of the previous time point a random number drawn from a Gaussian distribution N(0,1).
2. Random dataset: contains 200 million time series each again of length 256, representing "white noise". Each point is randomly drawn from a Gaussian distribution N(0,1).
3. Fourier slice data of the form $F_{x,y}$. This is done by first generating the frequency spectra of the time series and then applying an inverse Fourier transform plus some additional white noise to generate the series themselves. The spectrum generation assigns high amplitudes to x subsequent frequency components, starting from the y^{th} one, and low ones for the rest. Randomness is applied to the phase shifts of all frequencies, so that the generated set of series

[1] http://www.grid5000.fr

can enjoy a good variety. In our notation, when y is not explicitly specified, $y = 0$ is assumed.

Real datasets. In our experiments, we used several real datasets representing various domains such as seismology, finance, astronomy, neuroscience and image processing.

1. The seismic dataset, Seismic, was obtained from the IRIS Seismic Data Access archive [19]. It contains seismic instrument recording from thousands of stations worldwide and consists of 40 million data series of 200 values each.
2. The two finance datasets, StockP and StockR, are based on historical finance data downloaded using the Yahoo Finance API[2] that contains end-of-day quotes for over 40000 stock symbols for the period from Jan 2010 to Mar 2018. After preprocessing, the StockP dataset contains 72 million series of 200 end-of-day quotes. Each series represents a sub-period of 200 consecutive days for a particular stock symbol. The StockR dataset contains the price returns (fractional change of price from one day to another) of the quotes from StockP.
3. The astronomy dataset, Astro, represents celestial objects and was obtained from [45]. The dataset consists of 104 million data series of size 256.
4. The neuroscience dataset, SALD, obtained from [38] represents MRI data, including 209 million data series of size 128.
5. The image processing dataset, Deep1B, retrieved from [36], contains 279 million Deep1B vectors of size 96 extracted from the last layers of a convolutional neural network.

Measures. For each dataset and for each method (approximate iSAX, exact iSAX, and ParSketch), we measure the pruning ratio (when available), the precision, and the time in seconds. Pruning is the fraction of the database that are discarded by the iSAX index. Precision is defined to be correlation of the k^{th} time series found by a particular method divided by the k^{th} closest time series found by direct computation of correlation.

Queries. We evaluated the performance of the tested methods with three types of query workloads, batches of 10, 100 and 1000 queries. In our experiments, the default number of queries is 100.

6.2 Results

Using different synthetic datasets, Figures 6 and 7 show the query performance results[3] for different approaches while setting k to 1 and 10 respectively. In both figures, we see the query response time and precision of ParSketch, and exact/approximate versions of DPiSAX. We can see, for instance, that on the Random Walk dataset, DPiSAX approximate search is fast and takes 40 seconds (yellow bar) for a precision of 98% (green bar), while exact gives 100% precision (as expected for exact search) and is nearly as fast (52 seconds) as the pruning (green bar) is high - 98%. The query takes 78 seconds with parSketch, where the precision is 99.99%. These numbers may be compared, for instance, with those reported for the Random $F_{5,4}$

[2] http://finance.yahoo.com

[3] Indexing time responses are not reported here since they mainly depend on the cluster size.

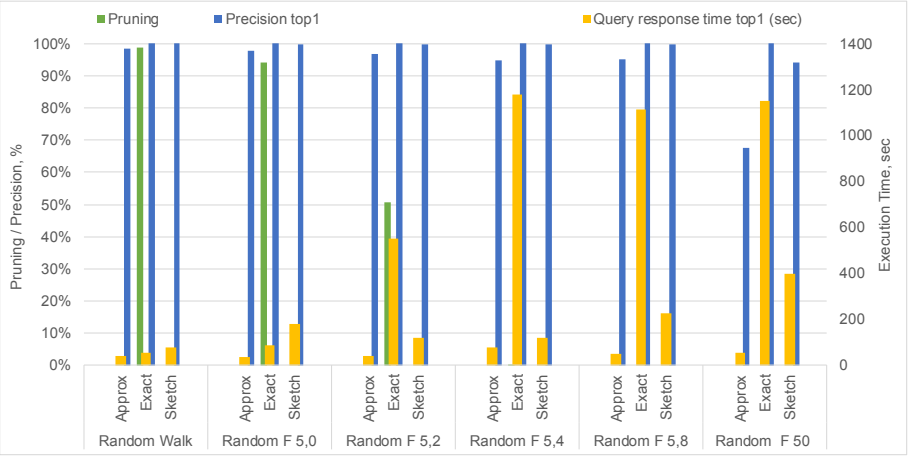


Fig. 6: *Synthetic datasets kNN=1 comparison. For the top-1 nearest neighbor query, the relative quality and time performance depends strongly on the dataset. When the lowest 5 Fourier coefficients contain most of the energy of the time series iSAX pruning works very well so both approximate iSAX gives high precision and exact iSAX is very fast. As the energy spreads to higher Fourier coefficients, iSAX pruning deteriorates with the result that approximate iSAX loses in precision and exact iSAX increases in time. Sketch time and precision stay about the same for all frequency distributions, though also degrades slightly as higher Fourier coefficients acquire more energy.*

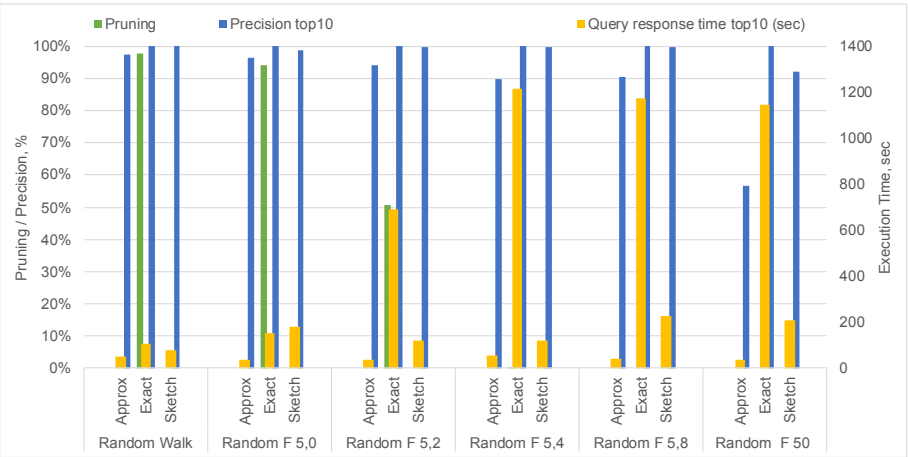


Fig. 7: *Synthetic datasets kNN=10 comparison. For the top-10 nearest neighbor query, the results are qualitatively similar to those for top-1 queries: iSAX works well when most of the power is in the lower Fourier coefficients, but less well when energy spreads to higher Fourier coefficients. Sketch time and precision vary much less.*

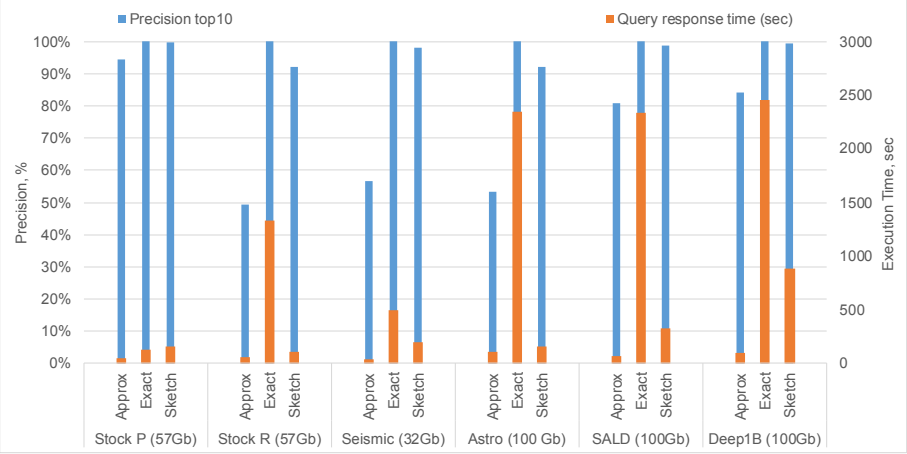


Fig. 8: *Comparison of Real Datasets: Stock Prices datasets (a random walk distribution model) have most of their power in the first few coefficients. The other datasets have power in multiple coefficients. For those cases, iSAX pruning works less well and so the quality of approximate iSAX suffers and the time of exact iSAX increases. ParSketch is slower than exact iSAX in the Random Walk dataset but is otherwise faster.*

dataset, where approximate takes 75 seconds for a precision of 94%, but exact search with zero pruning increases in time significantly, to 1179 seconds. For this dataset, parSketch returns result in 117 second for a precision of 99.85%.

Figure 8 shows the performance results using real datasets. As shown, DPiSAX performance varies across the datasets. The stock price dataset is like a random walk with most of the energy of its time series in the low Fourier coefficients. For that dataset, approximate DPiSAX is fast (52 seconds, orange bar) and quite accurate (94%, blue bar) and exact DPiSAX is almost as fast (126 seconds) and perfectly accurate. This is not true for the other data sets.

We studied the effect of Fourier coefficients on the quality of the kNN results by DPiSAX and ParSketch, in particular for $k = 1$ and $k = 10$. Table 1 shows the results sorted in ascending order by g_D that indicates the greatest of the Fourier coefficients that concentrate the majority of the time series energy. These results were measured through a setup with 8 PAA segments for the iSAX words. As seen, low values of g_D indicate that the data set will work well on iSAX (iSAX-friendliness), because it returns high quality results with lower response time than that of ParSketch.

As the number of high energy components increases, the iSAX pruning ratio significantly decreases. The reason is that in the low frequency case, different time series will acquire different PAA symbols. High frequencies within a PAA segment are not captured within the mean calculation of the PAA calculation, so very different time series will acquire the same PAA symbol. This reduces the efficiency of the index.

Furthermore, we studied the effect of varying the iSAX word length w (number of segments) on the quality of iSAX results, on various synthetic datasets with different frequency characteristics. The results are presented in Table 2, where, for each particular dataset D and its highest important Fourier coefficient g_D,

Table 1: *Frequency analysis of all datasets (real and synthetic). The range $[l_D..g_D]$ specifies the slice of Fourier coefficients that collectively account for at least 80% of the energy. iSAX pruning and quality of iSAX and sketches are displayed for both top-1 and top-10 nearest neighbors queries. The table is sorted by g_D, low values of which are favorable for efficient iSAX pruning and therefore the quality of approximate iSAX, which is observed in both top-1 and top-10 cases. Sketch quality is only slightly sensitive to the frequency characterization.*

dataset* D	series length	high energy freq. range $[l_D..g_D]$	iSAX pruning top 1	iSAX quality top 1	sketch quality top 1	iSAX pruning top 10	iSAX quality top 10	sketch quality top 10
R F_1	256	[1..1]	0.793	0.975	0.999	0.785	0.969	0.999
Stock P	200	[1..4]	0.989	0.964	1.000	0.963	0.944	0.999
RW	256	[1..4]	0.988	0.984	1.000	0.979	0.974	1.000
R $F_{5,0}$	256	[1..5]	0.942	0.979	0.999	0.926	0.964	0.989
R $F_{5,2}$	256	[3..7]	0.508	0.968	0.999	0.413	0.940	0.998
R $F_{5,4}$	256	[5..9]	0.000	0.946	0.998	0.000	0.900	0.998
R F_{10}	256	[1..10]	0.361	0.907	0.995	0.124	0.806	0.989
R $F_{5,8}$	256	[9..13]	0.000	0.950	0.999	0.000	0.905	0.998
R F_{20}	256	[1..20]	0.000	0.777	0.974	0.000	0.669	0.959
Deep1B	96	[1..47]	0.077	0.895	0.998	0.041	0.842	0.995
R F_{50}	256	[1..50]	0.000	0.676	0.941	0.000	0.565	0.921
SALD	128	[1..52]	0.000	0.898	0.994	0.004	0.807	0.989
Seismic	200	[5..53]	0.020	0.809	0.996	0.020	0.565	0.995
Stock R	200	[10..99]	0.022	0.637	0.933	0.000	0.493	0.921
Astro	256	[1..127]	0.004	0.654	0.946	0.002	0.535	0.921
RN	256	[1..127]	0.000	0.701	0.928	0.000	0.525	0.907

* R $F_{x,y}$ - randomly generated dataset with x subsequent high amplitude frequency components, starting from the y^{th} one; RW - Random Walk; RN - Random Noise.

the rows corresponding to $w = minseg(g_D)$ and $w = 2g_D$ are highlighted. The findings show that pruning is ineffective when $w < minseg(g_D)$. Increasing w helps improve the quality, which becomes particularly high when w reaches $2g_D$. However, raising w to 60 or more makes the indexing very slow, even slower than linear search for 100 queries. For this reason, we consider a dataset D to be in an "iSAX-unfriendly" regime, if its highest frequency g_D is greater than 30.

The first rows of the table show that iSAX pruning can be good even with very few segments (as few as 2). However, too few segments may lead to partitioning limitations that result in very few partitions, hence underexploiting the parallelization capabilities of the index. This explains the large indexing times for these cases. So, for efficient indexing, a minimum of 8 segments is recommended.

These results confirm our claims in Section 5, that:

1. low values of g_D (meaning that high frequencies do not have significant impact) are favorable for iSAX;
2. pruning is reasonable when the number of iSAX segments w is at least $minseg(g_D)$;
3. setting w to $2g_D$ leads to effective pruning;
4. iSAX exact search gets very expensive when $g_D > 30$; in such cases, ParSketch is recommended as a better approximate solution.

Readers interested in seeing more comparison results over different datasets are invited to visit the following web page: `http://imitates.gforge.inria.fr/`. In addition to performance results, the visitors can visually see randomly chosen

Table 2: *iSAX PAA segments parametrization. We vary the number of segments w for certain datasets and study the tradeoffs of DPiSAX performance in terms of quality and time of index construction and query answering. The rows corresponding to $w = minseg(g_D)$ and $w = 2g_D$ are highlighted. Based on these results, the tool BestNeighbor (described in section 5) does an analysis of any given dataset to determine if it is in an "iSAX-friendly" regime. If so, the tool outputs a recommended range of values for the number of segments for a given dataset, which corresponds to the highlighted rows in Table 2 (and could be used directly to tune DPiSAX parameter* `--wordLength`*). That may lead to efficient iSAX indexing and search time. If the dataset is determined to be "iSAX-unfriendly" - the tool recommends using ParSketch.*

dataset D	greatest high energy freq. g_D	number of segments w	index construction time (s)	pruning top 10	quality top 10	approximate search time (s)	exact search time (s)
Random Walk	4	**2**	**6900**	**0.611**$^{(*)}$	**0.982**	**51**	**808**
		4	747	0.851$^{(*)}$	0.941	45	254
		8	*751*	*0.961*$^{(*)}$	*0.945*	*31*	*102*
		16	850	0.988	0.943	31	69
		32	1117	0.994	0.927	28	30
		48	1387	0.996	0.935	32	26
		60	1548	0.996	0.930	28	26
Random $F_{5.0}$	5	**4**	**649**	**0.351**$^{(*)}$	**0.905**	**104**	**986**
		8	541	0.910$^{(*)}$	0.952	23	151
		10	*538*	*0.969*	*0.962*	*21*	*84*
		16	640	0.987$^{(*)}$	0.961	32	66
		32	890	0.993	0.965	37	67
		48	1157	0.994	0.962	37	65
		60	1320	0.995	0.965	32	56
Random $F_{5.2}$	7	4	824	0.000$^{(*)}$	0.856	179	1698
		8	**566**	**0.389**$^{(*)}$	**0.928**	**38**	**755**
		14	*590*	*0.958*	*0.960*	*21*	*72*
		16	644	0.975$^{(*)}$	0.962	39	87
		32	881	0.992	0.963	21	52
		48	1137	0.994	0.966	21	49
		60	1295	0.994	0.961	21	48
Random $F_{5.4}$	9	4	936	0.000	0.843	204	1827
		8	560	0.000$^{(*)}$	0.890	54	1179
		11	**594**	**0.315**	**0.933**	**34**	**808**
		16	838	0.890$^{(*)}$	0.959	24	174
		18	*654*	*0.954*	*0.966*	*37*	*112*
		32	825	0.990$^{(*)}$	0.967	27	62
		48	1058	0.993	0.964	35	65
		60	1308	0.995	0.965	21	47
Random F_{10}	10	4	680	0.001	0.685	99	1519
		8	548	0.094$^{(*)}$	0.778	28	1070
		13	**586**	**0.694**	**0.847**	**23**	**384**
		16	636	0.851$^{(*)}$	0.857	27	222
		20	*659*	*0.961*	*0.878*	*29*	*95*
		32	867	0.987$^{(*)}$	0.860	30	69
		48	1131	0.993	0.869	26	56
		60	1286	0.996	0.877	25	50
Random $F_{5.8}$	13	4	630	0.000	0.845	1245	1715
		8	762	0.000$^{(*)}$	0.904	43	1218
		16	667	0.090$^{(*)}$	0.947	41	1073
		18	**713**	**0.406**	**0.950**	**38**	**723**
		26	*789*	*0.944*	*0.964*	*40*	*137*
		32	890	0.979$^{(*)}$	0.965	30	78
Random F_{20}	20	8	551	0.000	0.661	33	1180
		16	624	0.002$^{(*)}$	0.743	43	1130
		32	**884**	**0.559**$^{(*)}$	**0.768**	**21**	**537**
		40	*1017*	*0.816*	*0.769*	*21*	*248*
		48	1076	0.897$^{(*)}$	0.751	32	175
		60	1213	0.950$^{(*)}$	0.751	37	114
Random F_{30}	30	8	564	0.000	0.585	35	1162
		16	676	0.000	0.68	23	1135
		32	866	0.003$^{(*)}$	0.706	22	1119
		48	**1137**	**0.256**	**0.699**	**22**	**843**
		60	*1289*	*0.590*$^{(*)}$	*0.697*	*24*	*497*

$^{(*)}$ Pruning ratio measures for particular combinations of g_D and w that were used as training examples for the regression model to define $minseg(g_D)$.

time series queries (up to 1000 for each dataset) and their k nearest neighbor time series.

6.3 Defining $minseg(g_D)$

As introduced in Section 5 and later validated through the experiments, for low values of w, the pruning ratio stays equal or close to zero (i.e., no useful pruning from the iSAX data structure). The result is that exact iSAX search resolves to a linear search (comparing the query time series to each item in the database) with the added overhead of pointlessly traversing the entire iSAX index tree to attempt pruning. For this reason, we consider pruning to be "reasonable" if it filters out at least a small fraction of the database. At some point as we increase w the pruning ratio takes off.

The $minseg(g_D)$ formula helps determine the minimal number of segments (word length) w for iSAX to achieve a reasonable pruning. To define the formula, we first utilized an empirical approach to build a training set for a regression model to approximate the pruning ratio as a function of g_D (the greatest high energy frequency of a dataset D) and w. Out of our collection of synthetic datasets, each characterized by its g_D, we took the measured pruning ratio for various w among the values $\{2, 4, 8, 16, 32, 48, 60\}$.

For each particular dataset (and g_D), we have selected a representative range of w values that lead to a range of pruning ratios, i.e., keeping at most one training example with zero pruning ratio and at most one with high pruning (≥ 0.95). The training set is in fact a subset of Table 2, where the 24 training examples are marked with $^{(*)}$ in the "pruning ratio" column.

Then, we explored different setups for linear regression using the pruning ratio as target and combinations of g_D, w, $\frac{1}{g_D}$, $\frac{1}{w}$, $\frac{g_D}{w}$, $\frac{w}{g_D}$ as features. The most significant regression coefficients were attributed to the features $\frac{1}{w}$ and $\frac{g_D}{w}$. The linear regression on those features showed the satisfactory R^2 score of 0.85, leading to the following approximation of pruning ratio P_{approx} (coefficients rounded to 1 decimal place for simplicity):

$$P_{approx} = 3.4\frac{1}{w} - 1.2\frac{g_D}{w} + 1.2$$

Further, we aim at finding a threshold C, such that $P_{approx} \geq C$ holds for all training examples, for which the measured pruning ratio is "reasonable", and does not hold for those where the pruning ratio is "close to zero". We determined that when $C = 0.5$, this condition is satisfied. Solving the inequality $P_{approx} \geq 0.5$ leads to $w \geq minseg(g_D)$, where:

$$minseg(g_D) = 1.75g_D - 5$$

After defining $minseg(g_D)$ as above (note that $2.0g_D$ would be the Nyquist rate), we measured the iSAX pruning for all datasets, setting the number of segments to $minseg(g_D)$ and $2.0g_D$, as shown in Table 2.

7 Related Work

In the context of time series data mining, several techniques have been developed and applied to time series data, *e.g.*, clustering, classification, outlier detection, pattern identification, motif discovery, and others. The idea of indexing time series is relevant to all these techniques. Note that, even though several databases have been developed for the management of time series (such as Informix Time Series, InfluxDB, OpenTSDB, and DalmatinerDB based on RIAK), they do not include similarity search indexes, focusing instead on (temporal) SQL-like query workloads. Thus, they cannot efficiently support similarity search queries, which is the focus of our study.

Indexes often make the response time of lookup operations sublinear in the database size. Relational systems have mostly been supported by hash structures, B-trees, and multidimensional structures such as R-trees, with bit vectors playing a supporting role. Such structures work well for lookups, but only adequately for similarity queries. The problem of indexing time series using centralized solutions has been widely studied in the literature, *e.g.*, [3,4,14,44,7]. For instance, in [3], Assent et al. propose the TS-tree (time series tree), an index structure for efficient retrieval and similarity search over time series. The TS-tree provides compact summaries of subtrees, thus reducing the search space very effectively. To ensure high fanout, which in turn results in small and efficient trees, index entries are quantized and dimensionally reduced.

In [4], Cai et al. use Chebyshev polynomials as a basis for dealing with the problem of approximating and indexing d-dimensional trajectories and time series. They show that the Euclidean distance between two d-dimensional trajectories is lower bounded by the weighted Euclidean distance between the two vectors of Chebyshev coefficients, and use this fact to create their index.

In [14], Faloutsos et al. use R*-trees to locate multi dimensional sequences in a collection of time series. The idea is to map a large time series sequence into a set of multi-dimensional rectangles, and then index the rectangles using an R*-tree. Our work is able to use the simplest possible multi-dimensional structure, the grid structure, because our problem is simpler as we will see.

7.1 iSAX-based Indexes

In [44], Shieh and Keogh propose a multiresolution symbolic representation called indexable Symbolic Aggregate approXimation (iSAX) which is based on the SAX representation. The advantage of iSAX over SAX is that it allows the comparison of words with different cardinalities, and even different cardinalities within a single word. iSAX can be used to create efficient indices over very large databases. Several works have then built upon the iSAX representation, including iSAX2+ [5,6], Adaptive Data Series Index (ADS+) [50,51], Compact and Contiguous Sequence Infrastructure (Coconut) [23], Parallel Index for Sequences (ParIS) [33], and Ultra Compact Index for Variable-Length Similarity Search (ULISSE) [28]. A recent study is comparing the performance of several different time series indexes [10].

The iSAX2+ index [6] was specifically designed for very large collections of time series, proposing new mechanisms and corresponding algorithms for efficient bulk loading and node splitting. The authors described algorithms for efficient

handling of the raw time series data during the bulk loading process, by using a technique that uses main memory buffers to group and route similar time series together down the tree, performing the insertion in a lazy manner. In [50], instead of building the complete iSAX2+ index over the complete dataset and querying only later, Zoumpatianos et al. propose to adaptively build parts of the index, only for the parts of the data on which the users issue queries. Coconut [23] proposed a compact and contiguous data layout that is based on sortable iSAX representations, while ULISSE [28] focused on the problem of supporting similarity search queries on sequences of variable-length. All these indexes have been developed for a centralized environment, and cannot scale up to very high volumes of time series. The ParIS index [33] was recently proposed for taking advantage of the modern hardware parallelization opportunities within a single compute node. ParIS describes techniques that use the Single Instruction Multiple Data (SIMD) instructions, as well as the multi-core and multi-socket architectures, for parallel index creation and query answering. As such, ParIS is complementary to our DPiSAX approach.

7.2 Sketch-based Solutions

Our sketch-based method takes advantage of random vectors. The basic idea is to multiply each time series (or in a sliding window context, each window of a time series) with a set of random vectors. The result of that operation is a *sketch* for each time series consisting of the distance (or similarity) of the time series to each random vector. Then two time series can be compared by comparing sketches.

Note that the sketch approach we advocate is a kind of Locality Sensitive Hashing [15], by which similar items are hashed to the same buckets with high probability. In particular, the sketch approach is similar in spirit to SimHash [8], in which the vectors of data items are hashed based on their angles with random vectors.

Random projection has been widely used in the literature for dimensionality reduction. For example, in [39], Schneider et al. propose scalable density-based clustering algorithms using random projections. Their clustering algorithms achieve significant speedup compared to equivalent density-based techniques, with good clustering quality in Euclidean space. Wilkinson et al. [47] introduce a classifier designed to address the curse of dimensionality and exponential complexity by using random projections. Their classifier organizes a set of random projections into a decision list used for scoring new data points.

In [18], random projection is used for discovering representative trends over time series, *e.g.*, finding average trend that is the subsequence whose total distance to all other subsequences is the smallest. The authors propose approximate algorithms that work on a pool of sketches made from the original time series. In [9], Dasgupta takes advantage of random projections for learning high-dimensional Gaussian mixture models. He proposes an efficient algorithm that returns with high probability the true centers of the Gaussians, given a precision threshold specified by the user.

In ParSketch, we use random projection for creating an efficient index over large time series datasets. We parallelize the sketch approach both at index creation time

and at query processing time. Experiments show excellent and nearly linear gains in performance.

On the whole, the main limitation of the existing solutions in the literature is that they do not scale to very big databases containing billions of time series. For example, with the state of the art iSax-based solution [5], indexing a database of one billion time series takes several days, while our parallel solutions allow us to index it in some hours.

8 Conclusions

We have studied the performance and quality characteristics of the two state-of-the-art distributed methods for indexing massive time series databases, DPiSAX and ParSketch. DPiSAX excels when most of the energy in a database of time series lies in the low order Fourier coefficients. Otherwise, the DPiSAX pruning ratio suffers and therefore both accuracy of approximate DPiSAX and time for exact DPiSAX. In that case, ParSketch exhibits much better accuracy, thanks to its insensitivity to energy distribution across Fourier coefficients. These results hold across simulated and real datasets.

We have introduced a utility that estimates DPiSAX-friendliness based on a Fourier analysis of a sample of a given time series database to suggest an indexing method to use.

For purposes of reproducibility, our datasets and code (including for the random generators) are available here: `http://imitates.gforge.inria.fr/code.html`.

Acknowledgements The research leading to these results has received funding from the European Union's Horizon 2020, under grant agreement No. 732051. Shasha was supported by an INRIA international chair.

References

1. Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003.
2. Ira Assent, Ralph Krieger, Farzad Afschari, and Thomas Seidl. The ts-tree: Efficient time series search and retrieval. In *EDBT Conf.*, pages 252–263, 2008.
3. Ira Assent, Ralph Krieger, Farzad Afschari, and Thomas Seidl. The ts-tree: efficient time series search and retrieval. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, pages 252–263, 2008.
4. Yuhan Cai and Raymond Ng. Indexing spatio-temporal trajectories with chebyshev polynomials. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 599–610. ACM, 2004.
5. A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. isax 2.0: Indexing and mining one billion time series. In *Proceedings of the 2010 IEEE International Conference on Data Mining*, ICDM '10, pages 58–67, 2010.
6. A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. J. Keogh. Beyond one billion time series: indexing and mining very large time series collections with iSAX2+. *Knowledge and Information Systems (KAIS)*, 39:123–151, 2014.
7. Alessandro Camerra, Jin Shieh, Themis Palpanas, Thanawin Rakthanmanon, and Eamonn J. Keogh. Beyond one billion time series: indexing and mining very large time series collections with i SAX2+. *Knowl. Inf. Syst.*, 39(1):123–151, 2014.

8. Moses S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing (STOC)*, pages 380–388, 2002.
9. Sanjoy Dasgupta. Learning mixtures of gaussians. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, pages 634–, 1999.
10. Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. The lernaean hydra of data series similarity search: An experimental evaluation of the state of the art. *PVLDB*, 12(2):112–127, 2018.
11. Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. Return of the Lernaean Hydra: Experimental Evaluation of Data Series Approximate Similarity Search. *PVLDB*, 2019.
12. Philippe Esling and Carlos Agon. Time-series data mining. *ACM Comput. Surv.*, 45(1):12:1–12:34, December 2012.
13. Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. *SIGMOD Rec.*, 23(2):419–429, 1994.
14. Christos Faloutsos, M. Ranganathan, and Yannis Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, pages 419–429, 1994.
15. Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pages 518–529, 1999.
16. Pablo Huijse, Pablo A. Estévez, Pavlos Protopapas, Jose C. Principe, and Pablo Zegers. Computational intelligence challenges and applications on large-scale astronomical time series databases. *IEEE Comp. Int. Mag.*, 9(3):27–39, 2014.
17. Piotr Indyk. Stable distributions, pseudorandom generators, embeddings and data stream computation. In *41st Annual Symposium on Foundations of Computer Science (FOCS)*, pages 189–197, 2000.
18. Piotr Indyk, Nick Koudas, and S. Muthukrishnan. Identifying representative trends in massive time series data sets using sketches. In *International Conference on Very Large Data Bases (VLDB)*, pages 363–372, 2000.
19. I.R.I. for Seismology with Artificial Intelligence. Seismic data access. `http://ds.iris.edu/data/access/`, 2019.
20. W. B. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in Modern Analysis and Probability*, volume 26 of *Contemporary Mathematics*, pages 189–206, 1984.
21. Kunio Kashino, Gavin Smith, and Hiroshi Murase. Time-series active search for quick retrieval of audio and video. In *ICASSP*, 1999.
22. Eamonn J. Keogh. Exact indexing of dynamic time warping. In *VLDB*, 2002.
23. Haridimos Kondylakis, Niv Dayan, Kostas Zoumpatianos, and Themis Palpanas. Coconut: A scalable bottom-up approach for building data series indexes. *PVLDB*, 11(6), 2018.
24. Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbor in high dimensional spaces. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 614–623, 1998.
25. J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *SIGMOD*, 2003.
26. J. Lin, E. Keogh, L. Wei, and S. Lonardi. Experiencing sax: A novel symbolic representation of time series. *Data Min. Knowl. Discov.*, 15(2):107–144, 2007.
27. Michele Linardi and Themis Palpanas. ULISSE: ultra compact index for variable-length similarity search in data series. In *ICDE*, 2018.
28. Michele Linardi and Themis Palpanas. Scalable, variable-length similarity search in data series: The ulisse approach. *PVLDB*, 2019.
29. Michele Linardi, Yan Zhu, Themis Palpanas, and Eamonn J. Keogh. Matrix profile X: VALMOD - scalable discovery of variable-length motifs in data series. In *SIGMOD*, 2018.
30. Michele Linardi, Yan Zhu, Themis Palpanas, and Eamonn J. Keogh. VALMOD: A suite for easy and exact detection of variable length motifs in data series. In *SIGMOD*, 2018.
31. Themis Palpanas. Data series management: The road to big sequence analytics. *SIGMOD Record*, 44(2):47–52, 2015.
32. Themis Palpanas. Evolution of a Data Series Index. *CCIS*, 1197, 2020.
33. Botao Peng, Themis Palpanas, and Panagiota Fatourou. ParIS: The Next Destination for Fast Data Series Indexing and Query Answering. *IEEE BigData*, 2018.

34. T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*, 2012.
35. Usman Raza, Alessandro Camerra, Amy L. Murphy, Themis Palpanas, and Gian Pietro Picco. Practical data prediction for real-world wireless sensor networks. *IEEE Trans. Knowl. Data Eng.*, accepted for publication, 2015.
36. S. C. Vision. Deep billion-scale indexing. `http://sites.skoltech.ru/compvision/noimi`, 2019.
37. W.H.Baumgartner et al. G.Ponti C.R.Shrader P. Lubinski H.A.Krimm F. Mattana J. Tueller S. Soldi, V. Beckmann. Long-term variability of agn at hard x-rays. *Astronomy & Astrophysics*, 2014.
38. S. University. Southwest university adult lifespan dataset (sald). `http://fcon_1000.projects.nitrc.org/indi/retro/sald.html?utm_source=newsletter&utm_medium=email&utm_content=SeeData&utm_campaign=indi-1`, 2019.
39. Johannes Schneider and Michail Vlachos. Scalable density-based clustering with quality guarantees using random projections. *Data Min. Knowl. Discov.*, 31(4):972–1005, July 2017.
40. D. Shasha and Y. Zhu. *High Performance Discovery in Time series, Techniques and Case Studies*. Springer, 2004.
41. Dennis Shasha. Tuning time series queries in finance: Case studies and recommendations. *IEEE Data Eng. Bull.*, 22(2):40–46, 1999.
42. J. Shieh and E. Keogh. isax: Indexing and mining terabyte sized time series. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '08, pages 623–631, 2008.
43. J. Shieh and E. Keogh. isax: Disk-aware mining and indexing of massive time series datasets. *DMKD*, 19(1):24–57, 2009.
44. Jin Shieh and Eamonn Keogh. iSAX: Indexing and mining terabyte sized time series. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*, pages 623–631, 2008.
45. Soldi, S., Beckmann, V., Baumgartner, W. H., Ponti, G., Shrader, C. R., Lubi´nski, P., Krimm, H. A., Mattana, F., and Tueller, J. Long-term variability of agn at hard x-rays. *A&A*, 563:A57, 2014.
46. Yang W., Peng W., Jian P., Wei W., and Sheng H. A data-adaptive and dynamic segmentation index for whole matching on time series. *PVLDB*, 6(10):793–804, 2013.
47. Leland Wilkinson, Anushka Anand, and Dang Nhon Tuan. Chirp: A new classifier based on composite hypercubes on iterated random projections. In *International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 6–14. ACM, 2011.
48. Djamel Edine Yagoubi, Reza Akbarinia, Florent Masseglia, and Themis Palpanas. Dpisax: Massively distributed partitioned isax. In *ICDM*, pages 1135–1140, 2017.
49. Lexiang Ye and Eamonn J. Keogh. Time series shapelets: a new primitive for data mining. In *KDD*, 2009.
50. K Zoumpatianos, S Idreos, and T Palpanas. Indexing for interactive exploration of big data series. In *Proceedings of the International Conference on Management of Data (SIGMOD)*, SIGMOD '14, pages 1555–1566, 2014.
51. Kostas Zoumpatianos, Stratos Idreos, and Themis Palpanas. ADS: the adaptive data series index. *VLDB J.*, 25(6):843–866, 2016.
52. Kostas Zoumpatianos and Themis Palpanas. Data series management: Fulfilling the need for big sequence analytics. In *ICDE*, 2018.

Authors' Biographies

Oleksandra Levchenko is a research engineer at Inria since 2015. She received the PhD degree in computer science from the Ministry of Education and Science of Ukraine in 2013. Worked as an associated professor of computer science at Odessa National Polytechnic University, Ukraine. Her research interests include distributed computing, big data processing, time series, machine learning, computer vision.

Boyan Kolev received the PhD degree in Computer Science from the Bulgarian Academy of Sciences in 2006. In 2013, he joined the Zenith team of Inria as a research engineer working on scientific data management. Prior to joining Inria, he had worked on data management in several IT enterprises. His research interests include big data, query languages, time series, and machine learning. Since 2020, he is a research engineer at LeanXcale.

Djamel-Edine Yagoubi is a data scientist at StarClay. He did his PhD in Computer Science at Inria, in the Zenith team and the University of Montpellier. He has worked in the field of Big Data analytics, particularly Massively Distributed Time Series Indexing and Querying.

Reza Akbarinia is a research scientist at Inria. He received his Ph.D. degree in Computer Science from the University of Nantes in 2007. His research focuses on data management and analysis in large-scale distributed systems (P2P, grid, Cloud) and data privacy. He has authored and co-authored two books and several technical papers in main DB conferences and journals. He has served as PC member in several conferences, such as SIGMOD, VLDB, ICDE, EDBT, CIKM, etc.

Florent Masseglia is a scientific researcher in computer science at Inria since 2002. He works in Montpellier, in the Zenith team of Inria, on the analysis of very large scientific data. These data, derived from observations, experiments and simulation are indeed complex, often very large, and are at the heart of important issues to better understand the studied domains (agronomy, biology, medicine).
`http://www.florent-masseglia.info`

Themis Palpanas is Senior Member of the French University Institute (IUF), director of the Data Intelligence Institute of Paris (diiP), and Professor of computer science at the University of Paris. He is the author of 9 US patents (3 implemented in world-leading commercial data management products) and 2 French patents. He is the recipient of 3 Best Paper awards, and the IBM Shared University Research (SUR) Award. He is serving as Editor in Chief for BDR Journal, Associate Editor for PVLDB 2019 and TKDE journal, and Editorial Advisory Board member for IS journal.

Dennis E. Shasha is a Professor of Computer Science at the Courant Institute of New York University. His research interests include data science, pattern recognition, database tuning, biological computing, wireless communication, and concurrent verification. Shasha is an ACM Fellow and the recipient of an Inria Internaional Chair. He is co-editor in chief of Information Systems, and is or has been the puzzle columnist for CACM, Dr. Dobb's Journal, and Scientific American.

Patrick Valduriez is a senior researcher at Inria, working on distributed data management. He has been associate editor of major journals such as VLDBJ and DAPD and has served as PC chair or general chair of major conferences such as EDBT, SIGMOD and VLDB. He obtained the best paper award at VLDB 2000. He was the recipient of the 1993 IBM scientific prize in Computer Science in France and the 2014 Innovation Award from Inria - French Academy of Science. He is an ACM Fellow.