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

Big Sequence Management: A Glimpse on the Past, the Present, and the Future

Themis Palpanas

the date of receipt and acceptance should be inserted later

Abstract There is an increasingly pressing need, by several applications in diverse domains, for developing techniques able to index and mine very large collections of sequences, or data series. Examples of such applications come from biology, astronomy, entomology, the web, and other domains. It is not unusual for these applications to involve numbers of data series in the order of hundreds of millions to billions, which are often times not analyzed in their full detail due to their sheer size. In this work, we describe recent efforts in designing techniques for indexing and mining truly massive collections of data series that will enable scientists to easily analyze their data. We show that the main bottleneck in mining such massive datasets is the time taken to build the index, and we thus introduce solutions to this problem. Furthermore, we discuss novel techniques that adaptively create data series indexes, allowing users to correctly answer queries before the indexing task is finished. We also show how our methods allow mining on datasets that would otherwise be completely untenable, including the first published experiments using one billion data series. Finally, we present our vision for the future in big sequence management research.

Keywords data management · data indexing · data analytics · data series

1 Introduction

[**Motivation.**] Data series have gathered the attention of the data management community for almost two decades [49,12,35]. Data series are one of the most common types of data, and are present in virtually every scientific and social domain: they appear as audio sequences [26], shape and image data [54], financial [47], environmental monitoring [42] and scientific data [22], and they

T. Palpanas
Paris Descartes University, Paris, France
E-mail: themis@mi.parisdescartes.fr

have many diverse applications, such as in health care, astronomy, biology, economics, and others.

Recent advances in sensing, networking, data processing and storage technologies have significantly eased the process of generating and collecting tremendous amounts of data series at extremely high rates and volumes. It is not unusual for applications to involve numbers of sequences in the order of hundreds of millions to billions [1,2].

[Data Series.] A *data series*, or *data sequence*, is an ordered sequence of data points[1]. Formally, a data series $T = (p_1, ...p_n)$ is defined as a sequence of points $p_i = (v_i, t_i)$, where each point is associated with a value v_i and a time t_i in which this recording was made, and n is the size (or length) of the series. If the dimension that imposes the ordering of the sequence is time then we talk about *time series*, though, a series can also be defined over other measures (e.g., angle in radial profiles in astronomy, mass in mass spectroscopy, position in genome sequences, etc.).

A key observation is that analysts need to process and analyze a sequence (or subsequence) of values as a single object, rather than the individual points independently, which is what makes the management and analysis of data sequences a hard problem. Note that even though a sequence can be regarded as a point in n-dimensional space, traditional multi-dimensional approaches fail in this case, mainly due to the combination of the following two reasons: (a) the dimensionality is typically very high, i.e., in the order of several hundreds to several thousands, and (b) dimensions are strictly ordered (imposed by the sequence itself) and neighboring values are correlated.

[Need for Data Series Indexing.] In this context, nearest neighbor queries are of paramount importance, since they form the basis of virtually every data mining, or other complex analysis task involving data series. However, nearest neighbor queries across a large collection of data series are challenging, because data series collections grow very large in practice, with datasets including billions, or even trillions of data series [13,40]. Thus, methods for answering nearest neighbor queries rely on two main techniques: data summarization and indexing. Data series summarization is used to reduce the dimensionality of the data series [28,39,32,3,27,15,34], and then indexes are built on top of these summarizations [39,49,5,46,52].

Nevertheless, as the data series collections grow in size, the operation of indexing these collections can itself become the bottleneck in the entire process. As an answer to this problem, we have developed the iSAX2.0 [12] and iSAX2+ [13], the first data series indexes that inherently support bulk loading, and thus aim to minimize the index building time. Bulk loading refers to mechanisms that allow us to insert at once a large quantity of data in an index, and as a result lead to fast index-building times. Furthermore, we describe the ADS+ index [57,58], which is the first data series index than can start answering queries correctly before the entire index has been built. This

[1] For the rest of this paper, we are going to use the terms *data series* and *sequence* interchangeably.

goal is achieved by building very fast the main-memory part of the index (i.e., only the inner nodes), and deferring the materialization of the (expensive) leaf nodes to query time. This novel approach considerably shrinks the data-to-query gap, allowing users to start answering queries much faster than any previous approach, and enabling truly exploratory analysis on very large data series collections.

[**Need for Data Series Management Systems.**] There are important reasons why data Series (or Sequence) Management Systems (SMSs) are on the cusp of becoming a focal point for research activity in data management. The solutions that are currently available require custom code and the development of ad hoc systems for various tasks, requiring huge investments in time and effort, and duplication of effort across different teams. Even existing approaches based on DBMSs [7], Column Stores [50], or Array Databases [51]) do not provide a viable solution, since they have not been designed for managing and processing sequence data. Therefore, they do not offer a suitable declarative query language, storage model, auxiliary data structures (such as indexes), and optimization mechanism that can support a variety of sequence query workloads in an efficient manner.

We argue that a SMS is necessary in order to enable big sequence analytics, since it will offer the abstractions, tools, and automations needed for achieving this goal. Just like databases abstracted the relational data management problem and offered a black box solution that is now omnipresent, the proposed system will make it feasible for analysts that are not experts in data series management, as well as common users, to tap in the goldmine of the massive and ever-growing data series collections they (already) have.

[**Contributions.**] The contributions of this work can be summarized as follows.

- We briefly review the work relevant to data series summarization, and data series indexing. We present in more detail the iSAX summarization method, and discuss how it can be used to construct a data series index. Furthermore, we give an overview of the first data series indexes that support bulk loading, namely, iSAX2.0 and iSAX2+, which lead to index-building times considerably faster than previous approaches, allowing us to index datasets with 1 billion data series.
- We describe the first adaptive data series index, ADS+, which reduces by an additional order of magnitude the time needed by the index before it is ready to start answering queries. The ADS+ index starts by a minimal tree structure based on summarizations of the data series. Then, the index structure is continuously enriched as more queries arrive: each query that is not covered by the current contents of the index, triggers additional data to be brought inside the index, thus adaptively and automatically expanding subtrees in the hot branches of the index. This enables ADS+ to answer several hundreds of thousands of queries by the time that state-of-the-art techniques are still in the index creation phase.

– We argue for the need to develop a general-purpose sequence management system, and discuss the features of such a system: (a) it should be able to cope with big data sequences, that is, massive collections of sequences, which can be heterogeneous (i.e., originate from disparate domains and thus exhibit very different characteristics), and which can have uncertainty in their values (e.g., due to inherent errors in the measurements); (b) it should efficiently support a wide range of sequence queries and mining operations at a scalable fashion, while exploiting the benefits of physical and logical independence; and (c) it should support cost-based optimization, which will enable the system to automatically pick the right storage and execution strategies for answering different queries.

Paper Organization. The rest of this paper[2] is organized as follows. We structure our discussion in three main sections: we briefly review the main research directions and results in the literature in Section 2; we describe the current state of the art in data series indexing in Section 3; and we present our vision for the future in Section 4. Finally, we conclude in Section 5.

Note that the focus of this paper is on the data management problems relevant to massive sequence collections, and not on data mining and analysis, which we do not discuss here. Nevertheless, we argue that in most cases, the correct data management techniques can lead to significant time efficiency benefits for the mining and analysis algorithms.

2 The Past: Summarizations and Indexes

2.1 On Data Series Queries

There are various types of data sequence queries that analysts need to perform: (a) simple Selection-Projection-Transformation (SPT) queries, and (b) more complex Data-Mining (DM) queries. Simple SPT queries are those that select sequences and project points based on thresholds, point positions, or specific sequence properties (e.g., above, first 10 points, peaks), or queries that transform sequences using mathematical formulas (e.g., average). An example SPT query could be one that returns the first x points of all the sequences that have at least y points above a threshold. The majority of these queries could be handled (albeit not optimally) by current database management systems, which nevertheless, lack a domain specific query language that would support and facilitate such processing.

DM queries on the other hand are more complex by nature: the processing has to take into consideration the entire sequence, and treat as a single object, therefore being much more complex to process. Examples under this category are: queries by content (range and similarity queries, nearest neighbors), clustering, classification, outlier patterns, frequent sub-sequences, and

[2] A more detailed analysis of the topics discussed in this paper can be found in our previous studies [38,29,37,12,17,13,57,18,59,58,36].

others. These queries cannot be supported by current data management systems, since they require specialized data structures, algorithms and storage methods in order to be performed efficiently.

Note that the data series datasets and queries may refer to either static, or streaming data. In the case of streaming data series, we are interested in the sub-sequences defined by a sliding window. The same is also true for static data series of very large size (e.g., an electroencephalogram, or a genome sequence), which we divide into sub-sequences using a sliding (or shifting window). The length of these sub-sequences is chosen so that it can contain the patterns of interest.

One of the most basic data mining tasks is that of finding similar data series in a database [3]. The query comes in the form of a data series X and it says "find me the data series in the database which is most similar to X". Similarity search is an integral part of most data mining procedures, such as clustering [53], classification and deviation detection [11,16].

2.2 On Data Series Summarizations

A common approach for answering such queries is to perform a dimensionality reduction, or summarization technique. Several such summarizations have been proposed, such as the Discrete Fourier Transform (DFT) [3], the Discrete Wavelet Transform (DWT) [15], the Piecewise Aggregate Approximation (PAA) [28,56], the Adaptive Piecewise Constant Approximation (APCA) [14], or the Symbolic Aggregate approXimation (SAX) [34].

Note that recent studies suggest that on average, there is little to differentiate between these summarizations in terms of fidelity of approximation [19, 37] (even though it *is* the case that certain representations favor particular data types, e.g., DFT for star-light-curves, APCA for bursty data, etc.).

These summarizations are usually accompanied by distance bounding functions that relate distances in the summarized space to distances in the original space through either lower or upper-bounding. With such bounding functions, we can index data series directly in the summarized space [39,49,5,46,52], and use these indexes to efficiently answer nearest neighbor queries on large data series collections.

2.3 On Data Series Indexing

Even though recent studies have shown that in certain cases sequential scans can be performed very efficiently [40], such techniques are only applicable when the database consists of a single, long data series, and queries are looking for potential matches in small subsequences of this long data series. Such approaches, however, do not bring benefit to the general case of querying a mixed database of several data series. Therefore, indexing is required in order to efficiently support data exploration tasks, which involve ad-hoc queries, i.e., the query workload is not known in advance.

A large set of indexing methods have been proposed for the different data series summarization methods, including traditional multidimensional [21,39, 9,29] and specialized [49,5,46,52] indexes. Moreover, various distance measures have been presented that work on top of such indexes, e.g., Discrete Time Warping (DTW) and Euclidean Distance (ED).

Indexing can significantly reduce the time to answer DM queries. Nevertheless, recent studies have observed that the mere process of building the index can be prohibitively expensive in terms of time cost [12,13,57]: e.g., the process of creating the index for 1 billion data series takes several days to complete. This problem can be mitigated by the bulk loading technique. Bulk-loading has been studied in the context of traditional database indexes, such as B-trees and R-trees, and other multi-dimensional index structures [43, 4,30,23,24,20].

In the following section, we give an overview of iSAX 2.0 [12] and iSAX2+ [13], two data series indexes that implement a bulk loading strategy.

2.4 On the iSAX Summarization and Family of Indexes

The Piecewise Aggregate Approximation (PAA) [28,56] is a summarization technique that segments the data series in equal parts and calculates the average value for each segment. An example of a PAA representation can be seen in Figure 1; in this case the original data series is divided into 4 equal parts. Based on PAA, Lin et al. [34] introduced the Symbolic Aggregate approXimation (SAX) representation that partitions the value space in segments of sizes that follow the normal distribution. Each PAA value can then be represented by a character (i.e., a small number of bits) that corresponds to the segment that it falls into. This leads to a representation with a very small memory footprint, an important requirement for managing very large data series collections. A segmentation of size 3 can be seen in Figure 1, where the data series is represented with the SAX word "10 10 11".

The SAX representation was later extended to indexable SAX (iSAX) [49], which allows variable cardinality for each character of a SAX representation. An iSAX representation is composed of a set of characters that form a word, and each word represents a data series. In the case of a binary alphabet, with a word size of 3 characters and a maximum cardinality of 2 bits, we could have a set of data series (two in the following example) represented with the following words: $00_2 10_2 01_2$, $00_2 11_2 01_2$, where each character has a full cardinality of 2 bits and each word corresponds to one data series. Reducing the cardinality of the second character in each word, we get for both words the same iSAX representation: $00_2 1_1 01_2$ (1_1 corresponds to both 10 and 11, since the last bit is trailed when the cardinality is reduced). By starting with a cardinality of 1 for each character in the root node and by gradually performing splits by increasing the cardinality by one character at a time, one can build a tree index [49,48]. Such cardinality reductions can be efficiently calculated with bit mask operations.

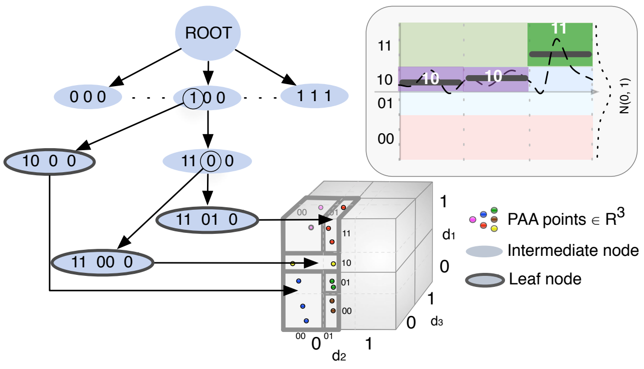


Fig. 1 An example of iSAX and SAX representations [57].

2.4.1 The iSAX 2.0 and iSAX2+ Indexes

Inserting a large collection of time series into the index iteratively is a very expensive operation, involving a high number of disk I/O operations [12,13]. This is because for each time series, we have to store the raw data series on disk, and insert into the index the corresponding iSAX representation. In order to speedup the process of building the index, we developed iSAX 2.0 [12] and iSAX2+ [13], the first data series indexes with a bulk loading strategy.

The key idea is to effectively group the data series that will end up in a particular subtree of the index, and process them all together. In order to achieve this goal, we use two main memory buffer layers, namely, the First Buffer Layer (FBL), and the Leaf Buffer Layer (LBL) [13]. The FBL corresponds to the children of the root of the index, while the LBL corresponds to the leaf nodes. The role of the buffers in FBL is to cluster together data series that will end up in the same subtree of the index, rooted in one of the direct children of the root. In contrast, the buffers in LBL are used to gather all the data series of leaf nodes, and flush them to disk.

The algorithm operates in two phases, which alternate until the entire dataset is processed, as follows (for more details, refer to [13]). During Phase 1, the algorithm reads data series and inserts them in the corresponding buffer in the FBL. This phase continues until the main memory is full. Then Phase 2 starts, where the algorithm proceeds by moving the data series contained in each FBL buffer to the appropriate LBL buffers. During this phase, the algorithm processes the buffers in FBL sequentially. For each FBL buffer, the algorithm creates all the necessary internal and leaf nodes, in order to index these data series. When all data series of a specific FBL buffer have been moved down to the corresponding LBL buffers, the algorithm flushes these LBL buffers to disk.

The difference between iSAX 2.0 [12] and iSAX2+ [13] is that the former treats the data series raw values (i.e., the detailed sequence of all the values of the data series) and their summarizations (i.e., the iSAX representations) together, while the latter uses just the summarizations in order to build the index, and only processes the raw values in order to insert them to the correct leaf node. In both cases, the goal is to minimize the random disk accesses, by making sure that the data series that end up in the same leaf node of the index are (temporarily) stored in the same (or contiguous) disk pages. Indeed, the experiments demonstrate that iSAX 2.0 and iSAX2+ significantly outperform previous approaches, reducing the time required to index 1 billion data series by 72% and 82%, respectively.

3 The Present: Adaptive Indexing

The target of indexing techniques is to make query processing efficient, so that analysts can repeatedly fire several exploratory queries with quick response times. However, even with a data series index that implements bulk loading, the amount of time required to build the index can be a significant bottleneck: for example, it takes more than a full day to build a state-of-the-art index over a data set of 1 billion data series in a modern server machine [57]. The main cost components of indexing are: (a) reading the data to be indexed, (b) spilling the indexed data and structures to disk, and (c) incurring the computation costs of figuring out where each new data entry belongs to (in the index structure). As the data size grows, the total indexing cost increases dramatically, to a degree where it creates a big and disruptive gap between the time when the data is available and the time when one can actually have access to the data. In fact, as the data grows, the query processing cost increasingly becomes a smaller fraction of the total cost (indexing + querying) [57].

As data sizes grow even bigger, waiting for several days before posing the first queries can be a major show-stopper for many applications both in businesses and in sciences. In addition, firing exploratory queries, i.e., queries which are not known a priori, is becoming quickly a common scenario. That is, in many cases, analysts and scientists need to explore the data before they can figure out what the next query is, or even which experiment to perform next; the output of one query inspires the formulation of the next query, and drives the experimental process.

In this section, we describe the ADS+ index, which enable fast indexing and a low data to query gap, when dealing with very large collections of data series.

3.1 The ADS+ Index

Even though iSAX 2.0 and iSAX2+ can effectively cope with very large data series collections, users still have to wait for extended periods of time before being able to start answering queries. We would instead like to allow users to answer queries much sooner.

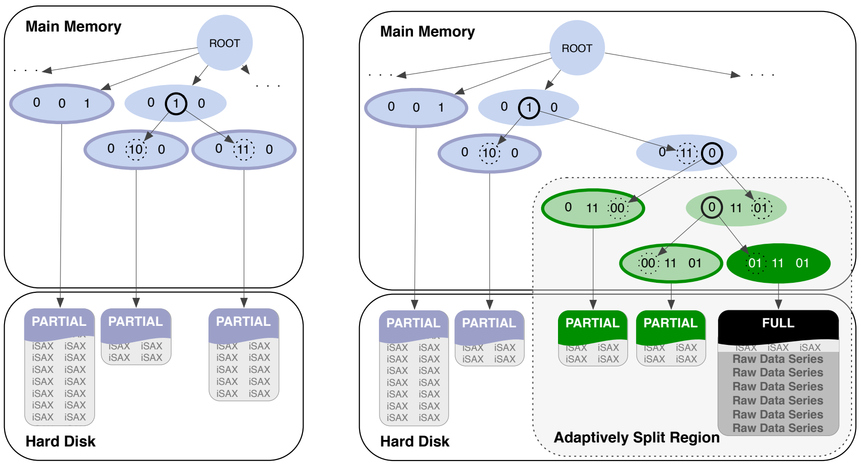
The ADS+ index [57] answers this problem by performing only a few basic steps, mainly creating the basic skeleton of the index tree, which contains condensed information on the input data series. As queries arrive, ADS+ fetches data series from the raw data and moves only those data series needed to correctly answer the queries inside the index. Future queries may be completely covered by the contents of the index, or alternatively ADS+ adaptively and incrementally fetches any missing data series directly from the raw data set. When the workload stabilizes, ADS+ can quickly serve fully contained queries while as the workload shifts, ADS+ may temporarily need to perform some extra work to adapt before stabilizing again. In addition, ADS+ does not require a fixed leaf size; it dynamically and adaptively adjusts the leaf size in hot areas of the index; all leaves start with a reasonably big size to guarantee fast indexing times, but the more a given area is queried, the more the respective leaves are split into smaller ones to enhance query times.

3.1.1 Proposed Algorithm

The main intuition (for more details, refer to [57]) is that one can quickly build the index tree using a large leaf size, saving time from very expensive split operations, and rely on queries that are then going to force splits in order to reduce the leaf sizes in the hot areas of the index. ADS+ uses two different leaf sizes: a big build-time leaf size for optimal index construction, and a small query-time leaf size for optimal access costs. This allows us to make future queries benefit from every split operation performed, finding the relevant data by traversing the tree, and not by scanning larger leaves. Initially, the index tree is built as in plain ADS, with a constant leaf size, equal to build-time leaf size. In traditional indexes, this leaf size remains the same across the lifetime of the index. In our case, when a query that needs to search a partial leaf arrives, ADS+ refines its index structure on-the-fly by recursively splitting the target leaf, until the target sub-leaf becomes smaller or equal to the query-time leaf size.

Adaptive and on demand leaf splitting allow ADS+ to have both fast index building and fast query processing. It does not waste time on creating fine-grained versions of each sub-tree of the index, but rather concentrates on the parts that are related to the current workload. When queries focus to a subset of the dataset, ADS+ does not need to exhaustively index and optimize all data; it rather concentrates on the most related sub-trees of the index.

Another optimization that gives ADS+ a lightweight behavior is that it delays leaf materialization even further. In particular, when traversing the tree for query processing, which leads to adaptive leaf splitting, ADS+ does not materialize the initial big leaf, nor all the leaves it creates on its way to the target small leaf. For example, when ADS+ needs to split a big leaf X and this results in X being split recursively into n new nodes until we reach the target leaf Z with a small leaf size, ADS+ fully materializes only the leaf Z. For the rest of the leaves, ADS+ uses the partial information contained in the leaves to perform the splits, i.e., the iSAX representations. This results in (a)



(a) ADS+ after index building. (b) ADS+ index after a query.

Fig. 2 The ADS+ index [57].

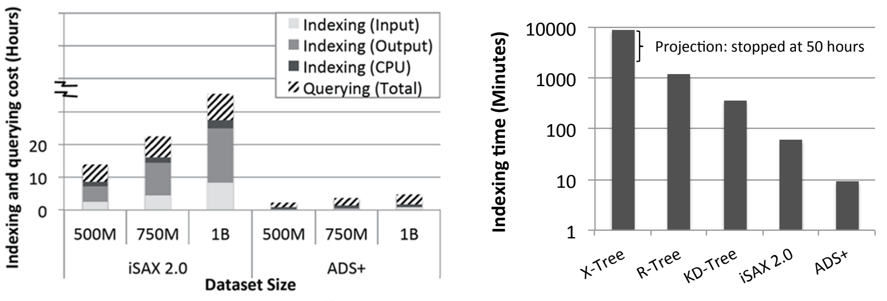
less computation as opposed to having to split based on raw data, (b) less I/O as SAX representations are much smaller, and (c) it enhances the adaptive behavior of ADS+ as it materializes only the truly interesting data that the queries are targeting.

An example of this process is shown in Figure 2. Figure 2(a) depicts the state of ADS+ after initialization and before any query has arrived, while Figure 2(b) shows how a single query results in adaptive splits of the right sub-tree until the target leaf node is fully materialized; intermediate nodes remain in partial mode and with a variable leaf size.

3.1.2 Experimental Results

For the purposes of the experimental evaluation, we implemented from scratch an optimized version of iSAX 2.0 in C and compiled with GCC 4.6.3 under Ubuntu Linux 12.04.2. We used an Intel Xeon machine with 64GB of RAM and 4x 2TB, SATA, 7.2K RPM Hard Drives in RAID0. All algorithms are set such as they make maximum use of all available memory.

We study the behavior up to 1 billion data series and with 10^5 random queries. Regarding leaf sizes, we use the optimal leaf size observed for each index strategy, i.e., 20K for iSAX 2.0, and for ADS+ 2K build-time and 10 query-time leaf size. Figure 3.1.2(a) shows the total time needed to build the index and answer all queries. Across all data sizes, ADS+ consistently outperforms iSAX 2.0 by a big margin. For 1 billion data series, ADS+ answers all 10^5 queries in less than 5 hours, while iSAX 2.0 needs more than 35 hours. By adaptively expanding the tree and adjusting leaf sizes only for the hot workload parts, ADS+ enjoys a $7x$ gain over full indexing in iSAX 2.0. Also, the rate at which the cost of ADS+ grows is significantly smaller than that of



(a) time to index and answer 10^5 queries for big data series collections (b) time to index 100 million data series

Fig. 3 Performance comparison between ADS+ and other indexes [57].

iSAX 2.0; For example, going from 500M to 1B data series, iSAX 2.0 needs more than twice the time, while ADS+ enjoys a sub-linear cost increase.

One interesting question is how indexes which are tailored for data series search compare against state-of-the-art spatial indexes. In this experiment, we compare ADS+ and iSAX 2.0 against KD-Tree [8], R-Tree [21], and X-Tree [9], which is a state-of-the-art adaptive version of R-Tree. Here, we use a set of 100 million data series. Figure 3.1.2(b) depicts the time needed to complete the index building phase for each index. Overall, both data series tailored indexes, iSAX 2.0 and ADS+, significantly outperform the more generic spatial indexes. For example, iSAX 2.0 is one order of magnitude faster than R-Tree while ADS+ is two orders of magnitude faster, and more than an order of magnitude faster than KD-Tree. The raw benefit comes from the fact iSAX 2.0 and ADS+ are tailored to perform efficient comparisons of SAX representations (with bitwise operations). ADS+ being adaptive enjoys further benefits as we discuss in previous experiments as well. X-Tree is significantly slower as a result of its more expensive index building phase which focuses on minimizing overlap between nodes. Naturally, this helps query processing times as less overlap allows queries to focus faster on data of interest. However, as we scale to big data, index building is the main bottleneck and thus X-Tree is prohibitively expensive.

4 The Future: Sequence Management System

Even though analysts in a variety of domains need to manage and process increasingly large data series collections, there is currently no general-purpose solution for the efficient management of sequence datasets. The techniques and tools that are available are rather fragmented, each one addressing only specific and narrow needs.

As a result, the few expert analysts need to invest heavily in the development of customized tools for processing their datasets in order to identify patterns, gain insights, detect abnormalities, and extract useful knowledge, while the many analysts that are not experts are simply not able to process their data. Consider for instance, that for several of their analysis tasks, neuroscientists are currently reducing each of their 3,000 point long sequences to a single number (the global average) in order to be able to analyze their huge datasets [1].

We note that current relational DBMSs [7], Column Stores [50], and Array Databases [51] could eventually be used to store and process sequences. Nevertheless, they cannot efficiently support complex data mining queries, (that is, queries that treat the entire sequence as a single object, such as sequence similarity queries, clustering, classification, etc.), which require fast distance computations among the sequences in the collection, since they do not natively support any mechanisms for pruning the search space.

Consequently, these systems cannot offer optimization functionality for the execution of DM queries, which is a key requirement for efficient processing and analysis of very large sequence collections. Therefore, in this section we argue for the need to design and develop a general-purpose Sequence Management System (SMS).

A key element of a SMS is the design of a cost-based optimizer for the execution of sequence queries, with a special focus on complex data mining queries. There is currently no optimizer available for sequence queries, even though it is a necessary component for efficient and scalable processing and analytics. As we discuss next, traditional approaches fail in our setting, and therefore, major breakthroughs are needed in this direction.

The optimizer should depend on and be closely related to the storage and indexing solutions for sequences, two research areas that should also be addressed. The design of the data model should accommodate various sequence summarization techniques, including novel techniques for uncertain sequences, and innovative access methods (i.e., storage and indexing) that will be able to adapt to the user needs (i.e., the query workload). Moreover, particular attention should be paid to optimizations specific to data sequence techniques relevant to modern hardware and distributed environments.

In Figure 4, we illustrate the general architecture of a SMS. We elaborate on the individual components of the system in the following sections. We discuss optimization last, since it touches on the rest of the components, and also include a discussion on the need for a data sequence benchmark.

4.1 Data Model

As we mentioned earlier, neither the relational model nor the array model can adequately capture the characteristics of sequences. In the case of relational data, there are various options available for translating sequences into relations and each one of them has significant limitations. On the other hand, in Array Databases we lack the expressive power to define collections of sequences, and are restricted to defining large multi-dimensional matrices that encode both sequence and meta-data on an equal basis, which hinders efficiency.

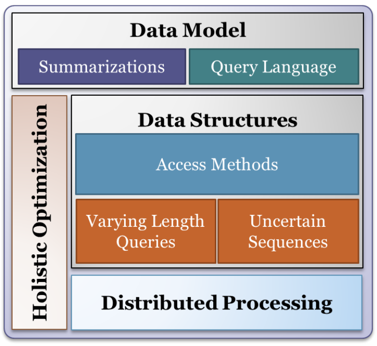


Fig. 4 The architecture of a data series management system.

An ideal sequence model should instead be able to effectively describe collections of sequences and allow us to do operations on them. It should allow us for example to select sequences based on meta-data or based on their values, project them as complete sequences, or sub-sequences, and join them in a variety of ways for computing calculations. At the same time such a model should intuitively allow for both intra-sequence and inter-sequence aggregations, and be compatible with different sequence summarization methods. Finally, the corresponding query language could be based on previous works [31, 44], suitably extended to deal with data series as single objects, as well as with DM queries.

4.2 Data Structures

A large collection of access methods has been proposed in the literature, able to evaluate different queries under various settings, including both indexes and scan-based methods. Recent work in this area is encouraging [13, 57], with iSAX2+ demonstrating scalability to dataset sizes 2-3 orders of magnitude more than the current state of the art, and ADS+ exhibiting a further 7-fold improvement in the time to prepare an index on 1 billion data series and answer 100,000 *approximate* queries.

Other promising directions should also be explored, such as methods that rely on fast scans of the data [27, 40]. These directions can provide viable alternatives to the indexes discussed above, and in several situations can be the access method of choice. This is especially true given the data management trend on large-scale parallelization, the usage of compression, multi-cores, SIMD architectures and the exploitation of available GPUs [41].

We also propose to extend these techniques along two orthogonal dimensions: supporting queries of varying length, and uncertain sequences. We note that existing techniques only consider collections of data series with the same length, leading to indexes that can answer queries of a fixed (predefined) length. As a result, new access methods that also consider varying length queries have to be developed. Contrary to previous approaches [25], we argue that the information already captured by certain data sequence indexes can be exploited, and is possible to develop new varying-length query answering techniques on top of this.

In several cases, data sequences can be uncertain, that is, the raw data have an inherent uncertainty in their values (e.g., because of errors introduced by the measurement devices), and integrate the solutions to the proposed system. There exist promising studies on modeling and analyzing uncertain sequences [6, 55, 45], but more work is needed in order to improve the quality and time performance [17]. A promising direction in this respect is the modeling of uncertain sequences with possible world semantics based on full-joint distributions, which can retain the correlation information among neighboring points [18]. Nevertheless, there are still important scalability issues to be overcome in order for such techniques to be used with large sequence collections.

4.3 Distributed Processing

During the last years there has been a lot of research on MapReduce systems, where various methods have been proposed to support the indexing of large multidimensional data [33], where an index is distributed among several compute nodes. Nevertheless, up to this point work on sequential data query processing using MapReduce has mainly concentrated on efficiently performing parallel scans of the complete dataset, while all indexing-related studies only consider read-only operations. Even though various approaches have been proposed for speeding up iterative algorithms, none of the proposed models is a suitable match for the algorithms and techniques we need, where timely communications among workers play a crucial role in reducing the amount of total work done. Therefore, there is need for more work in this area, taking into consideration new paradigms as well [10].

4.4 Cost based optimization

As we discussed above, there can be multiple different execution strategies for answering the same query, including the various choices of serial scans, indexes, and processing methods (e.g., parallelization, GPU, etc.). The challenge in choosing the right execution strategy is to estimate the amount of data that such a query will need to access before executing it. For example, a fast parallel SIMD-enabled scan on compressed data might be a better option than the use of a non-optimized index when SIMD instructions are available, but not a better choice when such instructions are not available. All these characteristics have to be exploited by the cost-based optimization models, and considered in a way that is transparent to the user. This problem becomes even more challenging when complex queries involving several operators need to be executed (e.g., consider an analysis task that combines a series of SPT operators as a pre-processing step, and then applies a DM operator).

While in traditional relational databases there are simple and efficient ways in order to estimate query selectivity [7], this is not the case for sequence similarity queries that lie in the heart of most sequence mining algorithms. The challenges in this context arise from the combination of the very high dimensional and sequential nature (i.e., the inherent correlations among neighboring values) of these data.

Up to this point, no efficient methods have been proposed to solve this problem, and ground-breaking work needs to be done. We believe that a promising direction is to carefully study the hardness of a query: being able to control the effort needed to answer a query can be the right step stone for solving the inverse problem, that of estimating the effort it will take to answer a query, before executing it.

4.5 Data Series Benchmarking

Despite the rich literature on methods for indexing and answering similarity queries on data sequences, we note the absence of any related benchmarks. We argue for the need of fair benchmarks that can stress-test sequence processing techniques in a controlled way and to pre-defined levels of query hardness. Such benchmarks will be designed to capture differences in the quality of summarization methods, indexes and storage methods, when working in *combination*, which is what makes the design of such a benchmark a challenging task. Our ongoing work constitutes the first solution towards this directions: it hows that the amount of effort employed by data series indexes can be consistently captured across different indexing approaches, using implementation-invariant measures [59].

5 Conclusions

In this work, we discussed the state-of-the-art data series indexing approaches that can cope with the data deluge. We reviewed the iSAX 2.0 and iSAX2+ indexes, which are the first specifically designed for very large collections of data series, and use novel algorithms for efficient bulk loading. We also described the first adaptive indexing approach, ADS+, where the index is built incrementally and adaptively, resulting in a very fast initialization process. We experimentally validated the proposed algorithms, including the first published experiments to consider datasets of size up to one billion data series, showing that we can deliver orders of magnitude improvements in the time required to build the index, and to start answering queries.

Furthermore, we observed that even though data series are a very common data type, there is currently no system that can inherently accommodate, manage, and support complex analytics for this type of data. Therefore, in this paper we argue for the special nature of the sequences data type, and articulate the necessity for rigorous work on data series management systems. We propose a sequence management system that will employ a data model specialized to sequences. The system will be distributed by design, and consider the large volume of sequences, their heterogeneity (in terms of properties and characteristics), and possible uncertainty in their values. Finally, the system will support cost-based optimization, thus, leading to the desired scalability for big sequence analytics.

Acknowledgements
I would like to thank my collaborators (in alphabetical order): Alessandro Camerra, Johannes Gehrke, Stratos Idreos, Eamonn Keogh, Michele Linardi, and Yin Lou. Special thanks go to Kostas Zoumpatianos, who has been the driving force behind several of the ideas discussed in this paper.

References

1. Adhd-200. `http://fcon_1000.projects.nitrc.org/indi/adhd200/`, 2011.
2. Sloan digital sky survey. `https://www.sdss3.org/dr10/data_access/volume.php`, 2015.
3. R. Agrawal, C. Faloutsos, and A. N. Swami. Efficient similarity search in sequence databases. In *FODO*, 1993.
4. N. An, R. Kanth, V. Kothuri, and S. Ravada. Improving performance with bulk-inserts in oracle r-trees. In *VLDB*, pages 948–951. VLDB Endowment, 2003.
5. I. Assent, R. Krieger, F. Afschari, and T. Seidl. The ts-tree: Efficient time series search and retrieval. In *EDBT*, 2008.
6. J. Aßfalg, H. Kriegel, P. Kröger, and M. Renz. Probabilistic similarity search for uncertain time series. In *SSDBM*, 2009.
7. M. M. Astrahan, M. W. Blasgen, D. D. Chamberlin, K. P. Eswaran, J. Gray, P. P. Griffiths, W. F. K. III, R. A. Lorie, P. R. McJones, J. W. Mehl, G. R. Putzolu, I. L. Traiger, B. W. Wade, and V. Watson. System R: relational approach to database management. *TODS*, 1(2):97–137, 1976.
8. J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9), Sept. 1975.
9. S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *VLDB*, pages 28–39, 1996.
10. P. Bernstein, S. Bykov, A. Geller, G. Kliot, and J. Thelin. Orleans: Distributed virtual actors for programmability and scalability. MSR-TR-2014-41, 2014.
11. Y. Bu, T. wing Leung, A. W. chee Fu, E. Keogh, J. Pei, and S. Meshkin. Wat: Finding top-k discords in time series database. In *SDM*, pages 449–454, 2007.
12. A. Camerra, T. Palpanas, J. Shieh, and E. Keogh. iSAX 2.0: Indexing and mining one billion time series. In *ICDM*, 2010.
13. 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+. *KAIS*, 39(1):123–151, 2014.
14. K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally adaptive dimensionality reduction for indexing large time series databases. In *SIGMOD*, 2002.
15. K.-P. Chan and A.-C. Fu. Efficient time series matching by wavelets. In *ICDE*, 1999.
16. V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: a survey. *ACM Computing Surveys*, 41(3):1–58, 2009.
17. M. Dallachiesa, B. Nushi, K. Mirylenka, and T. Palpanas. Uncertain time-series similarity: Return to the basics. *PVLDB*, 5(11):1662–1673, 2012.
18. M. Dallachiesa, T. Palpanas, and I. F. Ilyas. Top-k nearest neighbor search in uncertain data series. *PVLDB*, 8(1):13–24, 2014.
19. H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. In *PVLDB*, 2008.

20. P. W. Eljas Soisalon-Soininen. Single and Bulk Updates in Stratified Trees: An Amortized and Worst-Case Analysis. In *Computer Science in Perspective*, pages 278–292, 2003.
21. A. Guttman. R-trees: A dynamic index structure for spatial searching. In *SIGMOD*, 1984.
22. P. Huijse, P. A. Estévez, P. Protopapas, J. C. Principe, and P. Zegers. Computational intelligence challenges and applications on large-scale astronomical time series databases. *IEEE Comp. Int. Mag.*, 9(3):27–39, 2014.
23. B. S. Jochen Van den Bercken. An Evaluation of Generic Bulk Loading Techniques. In *VLDB*, pages 461–470, 2001.
24. P. W. Jochen Van den Bercken, Bernhard Seeger. A Generic Approach to Bulk Loading Multidimensional Index Structures. In *VLDB*, 1997.
25. S. Kadiyala and N. Shiri. A compact multi-resolution index for variable length queries in time series databases. *KAIS*, 15(2):131–147, 2008.
26. K. Kashino, G. Smith, and H. Murase. Time-series active search for quick retrieval of audio and video. In *ICASSP*, 1999.
27. S. Kashyap and P. Karras. Scalable knn search on vertically stored time series. In *KDD*, 2011.
28. E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality reduction for fast similarity search in large time series databases. *KAIS*, 3(3):263–286, 2000.
29. E. J. Keogh, T. Palpanas, V. B. Zordan, D. Gunopulos, and M. Cardle. Indexing large human-motion databases. In *VLDB*, pages 780–791, 2004.
30. J. V. J. S. V. Lars Arge, Klaus Hinrichs. Efficient Bulk Operations on Dynamic R-Trees. *Algorithmica*, 33(1):104–128, 2002.
31. A. Lerner and D. Shasha. Aquery: Query language for ordered data, optimization techniques, and experiments. In *VLDB*, 2003.
32. C.-S. Li, P. Yu, and V. Castelli. Hierarchyscan: a hierarchical similarity search algorithm for databases of long sequences. In *ICDE*, 1996.
33. H. Liao, J. Han, and J. Fang. Multi-dimensional index on hadoop distributed file system. In *NAS*, 2010.
34. J. Lin, E. Keogh, S. Lonardi, and B. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *DMKD*, 2003.
35. J. Lin, R. Khade, and Y. Li. Rotation-invariant similarity in time series using bag-of-patterns representation. *J. Intell. Inf. Syst.*, 39(2), 2012.
36. T. Palpanas. Data series management: The road to big sequence analytics. *SIGMOD Rec.*, 44(2):47–52, 2015.
37. T. Palpanas, M. Vlachos, E. J. Keogh, and D. Gunopulos. Streaming time series summarization using user-defined amnesic functions. *IEEE Trans. Knowl. Data Eng.*, 20(7):992–1006, 2008.
38. T. Palpanas, M. Vlachos, E. J. Keogh, D. Gunopulos, and W. Truppel. Online amnesic approximation of streaming time series. In *ICDE*, pages 339–349, 2004.
39. D. Rafiei and A. Mendelzon. Similarity-based queries for time series data. In *SIGMOD*, 1997.
40. T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. Batista, M. B. Westover, Q. Zhu, J. Zakaria, and E. J. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *KDD*, 2012.
41. V. Raman, G. K. Attaluri, R. Barber, N. Chainani, D. Kalmuk, V. KulandaiSamy, J. Leenstra, S. Lightstone, S. Liu, G. M. Lohman, T. Malkemus, R. Müller, I. Pandis, B. Schiefer, D. Sharpe, R. Sidle, A. J. Storm, and L. Zhang. DB2 with BLU acceleration: So much more than just a column store. *PVLDB*, 6(11):1080–1091, 2013.
42. U. Raza, A. Camerra, A. L. Murphy, T. Palpanas, and G. P. Picco. Practical data prediction for real-world wireless sensor networks. *IEEE Trans. Knowl. Data Eng.*, accepted for publication, 2015.
43. E. A. R. Rupesh Choubey, Li Chen. GBI: A Generalized R-Tree Bulk-Insertion Strategy. In *SSD*, pages 91–108, 1999.
44. R. Sadri, C. Zaniolo, A. M. Zarkesh, and J. Adibi. A sequential pattern query language for supporting instant data mining for e-services. In *VLDB*, 2001.
45. S. R. Sarangi and K. Murthy. DUST: a generalized notion of similarity between uncertain time series. In *KDD*, 2010.

46. P. Schäfer and M. Högqvist. Sfa: A symbolic fourier approximation and index for similarity search in high dimensional datasets. In *EDBT*, 2012.
47. D. Shasha. Tuning time series queries in finance: Case studies and recommendations. *IEEE Data Eng. Bull.*, 22(2):40–46, 1999.
48. J. Shieh and E. Keogh. iSAX: disk-aware mining and indexing of massive time series datasets. *DMKD*, 19(1):24–57, 2009.
49. J. Shieh and E. J. Keogh. *i*sax: indexing and mining terabyte sized time series. In *KDD*, pages 623–631, 2008.
50. M. Stonebraker, D. J. Abadi, A. Batkin, X. Chen, M. Cherniack, M. Ferreira, E. Lau, A. Lin, S. Madden, E. J. O'Neil, P. E. O'Neil, A. Rasin, N. Tran, and S. B. Zdonik. C-store: A column-oriented DBMS. In *VLDB*, 2005.
51. M. Stonebraker, P. Brown, A. Poliakov, and S. Raman. The architecture of scidb. In *SSDBM*, 2011.
52. Y. Wang, P. Wang, J. Pei, W. Wang, and S. Huang. A data-adaptive and dynamic segmentation index for whole matching on time series. *PVLDB*, 6(10):793–804, 2013.
53. T. Warren Liao. Clustering of time series data - a survey. *Pattern Recognition*, 38(11):1857–1874, 2005.
54. L. Ye and E. J. Keogh. Time series shapelets: a new primitive for data mining. In *KDD*, 2009.
55. M. Yeh, K. Wu, P. S. Yu, and M. Chen. PROUD: a probabilistic approach to processing similarity queries over uncertain data streams. In *EDBT*, 2009.
56. B. Yi and C. Faloutsos. Fast Time Sequence Indexing for Arbitrary Lp Norms. In *VLDB*, 2000.
57. K. Zoumpatianos, S. Idreos, and T. Palpanas. Indexing for interactive exploration of big data series. In *SIGMOD*, 2014.
58. K. Zoumpatianos, S. Idreos, and T. Palpanas. RINSE: interactive data series exploration with ADS+. *PVLDB*, 8(12):1912–1923, 2015.
59. K. Zoumpatianos, Y. Lou, T. Palpanas, and J. Gehrke. Query workloads for data series indexes. In *KDD*, 2015.