Return of the Lernaean Hydra: Experimental Evaluation of Data Series Approximate Similarity Search

Karima Echihabi
IRDA, Rabat IT Center,
ENSIAS, Mohammed V Univ.
karima.echihabi@gmail.com

Kostas Zoumpatianos
Harvard University
kostas@seas.harvard.edu

Themis Palpanas
Université de Paris
themis@mi.parisdescartes.fr

Houda Benbrahim
IRDA, Rabat IT Center,
ENSIAS, Mohammed V Univ.
houda.benbrahim@um5.ac.ma

ABSTRACT

Data series are a special type of multidimensional data present in numerous domains, where similarity search is a key operation that has been extensively studied in the data series literature. In parallel, the multidimensional community has studied approximate similarity search techniques. We propose a taxonomy of similarity search techniques that reconciles the terminology used in these two domains, we describe modifications to data series indexing techniques enabling them to answer approximate similarity queries with quality guarantees, and we conduct a thorough experimental evaluation to compare approximate similarity search techniques under a unified framework, on synthetic and real datasets in memory and on disk. Although data series differ from generic multidimensional vectors (series usually exhibit correlation between neighboring values), our results show that data series techniques answer approximate queries with strong guarantees and an excellent empirical performance, on data series and vectors alike. These techniques outperform the state-of-the-art approximate techniques for vectors when operating on disk, and remain competitive in memory.

1. INTRODUCTION

Motivation. A data series is a sequence of ordered values[1]. Data series are ubiquitous, appearing in nearly every

---

[1]The order attribute can be angle, mass, time, etc. [118]. When the order is time, the series is called a *time series*. We use *data series*, *time series* and *sequence* interchangeably.

domain including science and engineering, medicine, business, finance and economics [80, 135, 121, 141, 107, 72, 129, 19, 87, 97, 155, 68]. The increasing presence of IoT technologies is making collections of data series grow to multiple terabytes [117]. These data series collections need to be analyzed in order to extract knowledge and produce value [119]. The process of retrieving similar data series (i.e., similarity search), forms the backbone of most analytical tasks, including outlier detection [35, 28], frequent pattern mining [127], clustering [84, 130, 128, 153], and classification [39]. Thus, to render data analysis algorithms and pipelines scalable, we need to make similarity search more efficient.

Similarity Search. A large number of data series similarity search methods has been studied, supporting exact search [7, 137, 124, 81, 127, 110], approximate search [136, 85, 10, 46, 49], or both [32, 134, 152, 33, 163, 157, 88, 96, 95, 122, 158, 90, 123]. In parallel, the research community has also developed exact [23, 67, 22, 26, 44, 154, 57] and approximate [73] similarity search techniques geared towards generic multidimensional vector data[2]. In the past few years though, we are witnessing a renewed interest in the development of approximate methods [74, 156, 142, 18, 103].

This study is the first experimental comparison of the efficiency and accuracy of data series approximate similarity search methods ever conducted. Specifically, we evaluate the accuracy of both data series specific approximate similarity search methods, as well as that of approximate similarity search algorithms that operate on general multidimensional vectors. Moreover, we propose modifications to data series techniques in order to support approximate query answering with theoretical guarantees, following [43].

Our experimental evaluation covers in-memory and out-of-core experiments, the largest publicly available datasets, extensive comparison criteria, and a new set of methods that have never been compared before. We thus differ from other experimental studies, which focused on the efficiency of exact search [54], the accuracy of dimensionality reduction techniques and similarity measures for classification tasks [83, 52, 20], or in-memory high-dimensional methods [93, 17, 112]. In this study, we focus on the problem of *approximate whole matching similarity search in collec-*

---

[2]A comprehensive survey of techniques for multidimensional vectors can be found elsewhere [132].

tions with a very large number of data series, i.e., similarity search that produces approximate (not exact) results, by calculating distances on the whole (not a sub-) sequence. This problem represents a common use case across many domains [62, 142, 18, 103, 47, 105, 64, 119].

Contributions. Our key contributions are as follows:

1. We present a similarity search taxonomy that classifies methods based on the quality guarantees they provide for the search results, and that unifies the varied nomenclature used in the literature. Following this taxonomy, we include a brief survey of similarity search approaches supporting approximate search, bringing together works from the data series and multidimensional data research communities.

2. We propose a new set of approximate approaches with theoretical guarantees on accuracy and excellent empirical performance, based on modifications to the current data series exact methods.

3. We evaluate all methods under a unified framework to prevent implementation bias. We used the most efficient C/C++ implementations available for all approaches, and developed from scratch in C the ones that were only implemented in other programming languages. Our new implementations are considerably faster than the original ones.

4. We conduct the first comprehensive experimental evaluation of the efficiency and accuracy of data series approximate similarity search approaches, using synthetic and real series and vector datasets from different domains, including the two largest vector datasets publicly available. The results unveil the strengths and weaknesses of each method, and lead to recommendations as to which approach to use.

5. Our results show that the methods derived from the exact data series indexing approaches generally surpass the state-of-the-art techniques for approximate search in vector spaces. This observation had not been made in the past, and it paves the way for exciting new developments in the field of approximate similarity search for data series and multidimensional data at large.

6. We share all source codes, datasets, and queries [6].

2. DEFINITIONS AND TERMINOLOGY

Similarity search represents a common problem in various areas of computer science. In the case of data series, several different flavors have been studied in the literature, often times using overloaded and conflicting terms. We summarize here these variations, and provide definitions, thus setting a common language (for more details, see [54]).

On Sequences. A *data series* $S(p_1, p_2, ..., p_n)$ is an ordered sequence of points, p_i, $1 \leq i \leq n$. The number of points, $|S| = n$, is the length of the series. We denote the i-th point in S by $S[i]$; then $S[i : j]$ denotes the *subsequence* $S(p_i, p_{i+1}, ..., p_{j-1}, p_j)$, where $1 \leq i \leq j \leq n$. We use $\mathbb{S}$ to represent all the series in a collection (dataset). Each point in the series may represent the value of a single variable, i.e., *univariate series*, or of multiple variables, i.e., *multivariate series*. If these values encode errors, or imprecisions, we talk about uncertain data series [16, 160, 133, 48, 49].

Note that in the context of similarity search, a data series of length n can be represented as a single point in an n-dimensional space. Then the values and length of S are referred to as *dimensions* and *dimensionality*, respectively.

On Distance Measures. A data series *distance* is a function that measures the (dis)similarity of two data series [27, 50, 15, 41, 151, 108]. The distance between a query series,

S_Q, and a candidate series, S_C, is denoted by $d(S_Q, S_C)$. The Euclidean distance is the most widely used, and one of the most effective for large series collections [52]. Some similarity search methods also rely on the *lower-bounding distance* (distances in the reduced dimensionality space are guaranteed to be smaller than or equal to distances in the original space) [33, 163, 134, 152, 157, 96, 44, 81] and *upper-bounding distance* (distances in the reduced space are larger than the distances in the original space) [152, 81].

On Similarity Search Queries. We assume a data series collection, $\mathbb{S}$, a query series, S_Q, and a distance function $d(\cdot, \cdot)$. A *k-Nearest-Neighbor (k-NN) query* identifies the k series in the collection with the smallest distances to the query series, while an *r-range query* identifies all the series in the collection within range r from the query series.

Definition 1. *[54] Given an integer k, a **k-NN query** retrieves the set of series $\mathbb{A} = \{\{S_{C_1}, ..., S_{C_k}\} \subseteq \mathbb{S} | \forall \ S_C \in \mathbb{A}$ and $\forall \ S_{C'} \notin \mathbb{A}, \ d(S_Q, S_C) \leq d(S_Q, S_{C'})\}$.*

Definition 2. *[54] Given a distance r, an **r-range query** retrieves the set of series $\mathbb{A} = \{S_C \in \mathbb{S} | d(S_Q, S_C) \leq r\}$.*

We additionally identify *whole matching (WM)* queries (similarity between an entire query series and an entire candidate series), and *subsequence matching (SM)* queries (similarity between an entire query series and all subsequences of a candidate series).

Definition 3. *[54] A **WM query** finds the candidate data series $S \in \mathbb{S}$ that matches S_Q, where $|S| = |S_Q|$.*

Definition 4. *[54] A **SM query** finds the subsequence $S[i : j]$ of a candidate data series $S \in \mathbb{S}$ that matches S_Q, where $|S[i : j]| = |S_Q| < |S|$.*

In practice, we have WM queries on large collections of short series [55, 3], SM queries on large collections of short series [1], and SM queries on collections of long series [59]. Note that a SM query can be converted to WM [96, 95].

On Similarity Search Methods. The similarity search algorithms (k-NN or range) that always produce correct and complete answers are called *exact*. Algorithms that do not satisfy this property are called *approximate*. An *ε-approximate* algorithm guarantees that its distance results have a relative error no more than ϵ, i.e., the approximate distance is at most $(1 + \epsilon)$ times the exact one. A *δ-ε-approximate* algorithm, guarantees that its distance results will have a relative error no more than ϵ (i.e., the approximate distance is at most $(1 + \epsilon)$ times the exact distance), with a probability of at least δ. An *ng-approximate* (no-guarantees approximate) algorithm does not provide any guarantees (deterministic, or probabilistic) on the error bounds of its distance results.

Definition 5. *[54] Given a query S_Q, and $\epsilon \geq 0$, an **ε-approximate** algorithm guarantees that all results, S_C, are at a distance $d(S_Q, S_C) \leq (1 + \epsilon) \ d(S_Q, [k\text{-th NN of } S_Q])$ in the case of a k-NN query, and distance $d(S_Q, S_C) \leq (1 + \epsilon)r$ in the case of an r-range query.*

Definition 6. *[54] Given a query S_Q, $\epsilon \geq 0$, and $\delta \in [0, 1]$, a **δ-ε-approximate** algorithm produces results, S_C, for which $Pr[d(S_Q, S_C) \leq (1 + \epsilon) \ d(S_Q, [k\text{-th NN of } S_Q])] \geq \delta$ in the case of a k-NN query, and $Pr[d(S_Q, S_C) \leq (1 + \epsilon)r] \geq \delta)$ in the case of an r-range query.*

Definition 7. *[54] Given a query S_Q, an **ng-approximate** algorithm produces results, S_C, that are at a distance $d(S_Q, S_C) \leq (1 + \theta) \ d(S_Q, [k\text{-th NN of } S_Q])$ in the case of a k-NN query, and distance $d(S_Q, S_C) \leq (1 + \theta)r$ in the case of an r-range query, for an arbitrary value $\theta \in \mathbb{R}_{>0}$.*

In the data series literature, *ng-approximate* algorithms have been referred to as *approximate*, or *heuristic* search [33, 163, 134, 152, 157, 96]. Unless otherwise specified, we will refer to *ng-approximate* algorithms simply as approximate. Approximate matching in the data series literature consists of pruning the search space, by traversing one path of an index structure representing the data, visiting at most one leaf, to get a baseline best-so-far (bsf) match. In the multidimensional literature, ng-approximate similarity search is also called *Approximate Nearest Neighbor (ANN)* [74], ϵ-approximate 1-NN search is called *c-ANN* [142], and ϵ-approximate k-NN search is called *c-k-ANN* [71], where c stands for the approximation error and corresponds to $1+\epsilon$.

Observe that when $\delta = 1$, a δ-ϵ-approximate method becomes ϵ-approximate, and when $\epsilon = 0$, an ϵ-approximate method becomes exact [43]. It it also possible that the same approach implements both approximate and exact algorithms [137, 152, 33, 163, 134].

Scope. In this study, we focus on *univariate* series with *no uncertainty*, where each point is drawn from the domain of real values, $\mathbb{R}$, and we evaluate *approximate* methods for *whole matching* in datasets containing a *very large number of series*, using *k-NN queries* and the *Euclidean distance*. This scenario is key to numerous analysis pipelines in practice [153, 165, 118, 119], in fields as varied as neuroscience [65], seismology [78], retail data [92], and energy [91].

3. SIMILARITY SEARCH PRIMER

Similarity search methods aim at answering a query efficiently by limiting the number of data points accessed, while minimizing the I/O cost of accessing raw data on disk and the CPU cost when comparing raw data to the query (e.g., Euclidean distance calculations). These goals are achieved by exploiting summarization techniques, and using efficient data structures (e.g., an index) and search algorithms. Note that solutions based on sequential scans are geared to exact similarity search [127, 110], and cannot support efficient approximate search, since all candidates are always read.

Answering a similarity query using an index typically involves two steps: a filtering step where the pre-built index is used to prune candidates and a refinement step where the surviving candidates are compared to the query in the original high dimensional space [67, 154, 57, 33, 163, 134, 152, 22, 157, 96]. Some exact [22, 134, 154, 57] and approximate methods [142, 18] first summarize the original data and then index these summarizations, while others tie together data reduction and indexing [33, 163, 152]. Some approximate methods return the candidates obtained in the filtering step [18]. There also exist exact [44] and approximate [103] methods that index high dimensional data directly.

A variety of data structures exist for similarity search indexes, including trees [67, 22, 33, 163, 152, 157, 96, 142, 134], inverted indexes [74, 75, 156, 18], filter files [154, 57, 163], hash tables [73, 29, 51, 36, 98, 120, 109, 100, 61, 115, 71] and graphs [14, 37, 12, 150, 102, 131, 76, 103]. There also exist multi-step approaches, e.g., Stepwise [81], that transform and organize data according to a hierarchy of resolutions.

Next, we outline the *approximate* similarity search methods (refer also to Table 1) and their summarization techniques. (*Exact* methods are detailed in [54]).

3.1 Summarization Techniques

Random projections (used by SRS [142]) reduce the original high dimensional data into a lower dimensional space by multiplying it with a random matrix. The Johnson-Lindenstrauss (JL) Lemma [77] guarantees that if the projected space has a large enough number of dimensions, there is a high probability that the pairwise distances are preserved, with a distortion not exceeding $(1 + \epsilon)$.

Piecewise Aggregate Approximation (PAA) [82] and *Adaptive Piecewise Constant Approximation* (APCA) [34] are segmentation techniques that approximate a data series S using l segments (of equal/arbitrary length, respectively). The approximation represents each segment with the mean value of its points. The *Extended APCA* (EAPCA) [152] technique extends APCA by representing each segment with both the mean and the standard deviation.

Quantization is a lossy compression process that maps a set of infinite numbers to a finite set of codewords that together constitute the codebook. A *scalar* quantizer operates on the individual dimensions of a vector independently, whereas a *vector* quantizer considers the vector as a whole (leveraging the correlation between dimensions [66]). The size k of a codebook increases exponentially with the number of bits allocated for each code. A *product* quantizer [74] splits the original vector of dimension d into m smaller subvectors, on which a lower-complexity vector quantization is performed. The codebook then consists of the cartesian product of the codebooks of the m subquantizers. Scalar and vector quantization are special cases of product quantization, where m is equal to d and 1, respectively.

(i) *Optimized Product Quantization* (OPQ) (used by IMI [62]) improves the accuracy of the original product quantizer [74] by adding a preprocessing step consisting of a linear transformation of the original vectors, which decorrelates the dimensions and optimizes space decomposition. A similar quantization technique, CK-Means, was proposed in [114] but OPQ is considered the state-of-the-art [79, 106].

(ii) The *Symbolic Aggregate Approximation* (SAX) [94] technique starts by transforming the data series into l real values using PAA, and then applies a *scalar* quantization technique to represent the PAA values using discrete symbols forming an alphabet of size a, called the cardinality of SAX. The l symbols form the SAX representation. The *iSAX* [138] technique allows comparisons of SAX representations of different cardinalities, which makes SAX indexable.

(iii) *The Karhunen-Loève transform (KLT)*. The original VA+file method [57] first converts a data series S of length n using KLT into n real values to de-correlate the data, then applies a *scalar* quantizer to encode the real values as discrete symbols. As we will explain in the next subsection, for efficiency considerations, we altered the VA+file to use the *Discrete Fourier Transform* (DFT) instead of KLT. DFT [7, 56, 125, 126] approximates a data series using l frequency coefficients, and can be efficiently implemented with Fast Fourier Transform (FFT), which is optimal for whole matching (alternatively, the MFT algorithm [8] is adapted to subsequence matching since it uses sliding windows).

3.2 Approximate Similarity Search Methods

There exist several techniques for approximate similarity search [73, 63, 31, 70, 46, 38, 9, 145, 142, 62, 103, 159] [25, 116, 161]. We focus on the 7 most prominent techniques designed for multidimensional data, and we also describe the approximate search algorithms designed specifically for data series. We also propose a new set of techniques that can answer δ-ϵ-approximate queries based on modifications to existing exact similarity methods for data series.

Table 1: Similarity search methods used in this study ("•" indicates our modifications to original methods). All methods support in-memory data, but only methods ticked in last column support disk-resident data.

		Matching Accuracy				Representation		Implementation		
		exact	ng-appr.	ϵ-appr.	δ-ϵ-appr.	Raw	Reduced	Original	New	Disk-resident Data
Graphs	HNSW		[103]			✓		C++		
	NSG		[60]			✓		C++		
Inv. Indexes	IMI		[18, 62]				OPQ	C++		✓
LSH	QALSH				[71]		Signatures	C++		
	SRS				[142]		Signatures	C++		
Scans	VA+file	[57]	•	•	•		DFT	MATLAB	C	✓
Trees	Flann		[111]			✓		C++		
	DSTree	[152]	[152]	•	•		EAPCA	Java	C	✓
	HD-index		[13]				Hilbert keys	C++		✓
	iSAX2+	[33]	[33]	•	•		iSAX	C#	C	✓

3.2.1 State-of-the-Art for Multidimensional Vectors

Flann [111] is an in-memory ensemble technique for *ng*-approximate nearest neighbor search in high-dimensional spaces. Given a dataset and a desired search accuracy, Flann selects and auto-tunes the most appropriate algorithm among randomized kd-trees [139] and a new proposed approach based on hierarchical k-means trees [111].

HD-index [13] is an *ng*-approximate nearest neighbor technique that partitions the original space into disjoint partitions of lower dimensionality, then represents each partition by an RBD tree (modified B+tree with leaves containing distances of data objects to reference objects) built on the Hilbert keys of data objects. A query Q is partitioned according to the same scheme, searching the hilbert key of Q in the RDB tree of each partition, then refining the candidates first using approximate distances based on triangular and Ptolemaic inequalities then using the real distances.

HNSW. HNSW [103] is an in-memory *ng*-approximate method that belongs to the class of proximity graphs that exploit two fundamental geometric structures: the Voronoi Diagram (VD) and the Delaunay Triangulation (DT). A VD is obtained when a given space is decomposed using a finite number of points, called *sites*, into regions such that each site is associated with a region consisting of all points that are closer to it than to any other site. The DT is the dual of the VD. It is constructed by connecting sites with an edge if their regions share a side. Since constructing a DT for a generic metric space is not always possible (except if the DT is the complete graph) [113], proximity graphs, which approximate the DT by conserving only certain edges, have been proposed [14, 37, 12, 150, 102, 131, 76, 103]. A k-NN graph is a proximity graph, where only the links to the closest neighbors are preserved. Such graphs suffer from two limitations: (i) the curse of dimensionality; and (ii) the poor performance on clustered data (the graph has a high probability of being disconnected). To address these limitations, the Navigable Small World (NSW) method [102] proposed to heuristically augment the approximation of the DT with long range links to satisfy the small world navigation properties [86]. The HNSW graph [103] improves the search efficiency of NSW by organizing the links in hierarchical layers according to their lengths. Search starts at the top layer, which contains only the longest links, and proceeds down the hierarchy. HNSW is considered the state-of-the-art [17].

NSG [60] is a recent in-memory proximity graph approach that approximates a graph structure called MRNG [60] which belongs to the class of Monotonic Search Networks (MSNET). Building an MRNG graph for large datasets be-

comes impractical; that is why the state-of-the-art techniques approximate it. NSG approximates the MRNG graph by relaxing the monotonicity requirement and edge selection strategy, and dropping the longest edges in the graph.

IMI. Among the different quantization-based inverted indexes proposed in the literature [74, 75, 156, 18], IMI [62, 18] is considered the state-of-the-art [106]. This class of techniques builds an inverted index storing the list of data points that lie in the proximity of each codeword. The codebook is the set of representative points obtained by performing clustering on the original data. When a query arrives, the *ng*-approximate search algorithm returns the list of all points corresponding to the closest codeword (or list of codewords).

LSH. The LSH family [11] encompasses a class of randomized algorithms that solve the δ-ϵ-approximate nearest neighbor problem in sub-linear time, for $\delta < 1$. The main intuition is that two points that are nearby in a high dimensional space, will remain nearby when projected to a lower dimensional space [73]. LSH techniques partition points into buckets using hash functions, which guarantee that only nearby points are likely to be mapped to the same bucket. Given a dataset $\mathbb{S}$ and a query S_Q, L hash functions are applied to all points in $\mathbb{S}$ and to the query S_Q. Only points that fall at least once in the same bucket as S_Q, in each of the L hash tables, are further processed in a linear scan to find the δ-ϵ-approximate nearest-neighbor. There exist many variants of LSH, either proposing different hash functions to support particular similarity measures [29, 51, 36, 61], or improving the theoretical bounds on query accuracy (i.e., δ or ϵ), query efficiency or the index size [98, 120, 109, 100, 61, 115, 142, 71] [99]. In this work, we select SRS [142] and QALSH [71] to represent the class of LSH techniques because they are considered the state-of-the-art in terms of footprint and accuracy, respectively [13]. SRS answers δ-ϵ-approximate queries using size linear to the dataset size, while empirically outperforming other LSH methods (with size super-linear to the dataset size [29]). QALSH is a query-aware LSH technique that partitions points into buckets using the query as anchor. Other LSH methods typically partition data points before a query arrives, using a random projection followed by a random shift. QALSH, does not perform the second step until a query arrives, thus improving the likelihood that points similar to the query are mapped to the same bucket.

3.2.2 State-of-the-Art for Data Series

While a number of data series methods support approximate similarity search [136, 85, 10, 46, 32, 134, 152, 33,

163], we focus on those that fit the scope of this study, i.e., methods that support out-of-core k-NN queries with Euclidean distance. In particular, we examine DSTree [152], iSAX2+ [33], and VA+file [57], the three data series methods that perform the best in terms of exact search [54], and also inherently support ng-approximate search.

DSTree [152] is a tree index based on the EAPCA summarization technique and supports ng-approximate and exact query answering. Its dynamic segmentation algorithm allows tree nodes to split vertically and horizontally, unlike the other data series indexes which allow either one or the other. The DSTree supports a lower and upper bounding distance and uses them to calculate a QoS measure that determines the optimal way to split any given node. We significantly improved the efficiency of the original DSTree Java implementation by developing it from scratch in C and optimizing its buffering and memory management, making it 4 times faster across datasets ranging between 25-250GB.

SAX-based indexes include different flavors of tree indexes based on SAX summarization. The original iSAX index [137] was enhanced with a better splitting policy and bulk-loading support in iSAX 2.0 [32], while iSAX2+ [33] further optimized bulk-loading. ADS+ [163] then improved upon iSAX2+ by making it adaptive, Coconut [88, 89, 90] by constructing a compact and contiguous data layout, and DPiSAX [157, 158], ParIS [122] and MESSI [123] by exploiting parallelization. Here, we use iSAX2+, because of its excellent performance [54] and the fact that the SIMS query answering strategy [163] of ADS+, Coconut, and ParIS is not immediately amenable to approximate search with guarantees (we plan to extend these methods in our future work). We do not include DPiSAX and MESSI, because they are distributed, and in-memory only, algorithms, respectively.

TARDIS [162] is a distributed indexing method that supports exact and ng-approximate kNN queries. It improves the efficiency and accuracy of iSAX by building a more compact, k-ary tree index, exploiting word-level (instead of character-level) cardinality, and using a novel conversion scheme between SAX representations. We do not include TARDIS in the experimental evaluation since it is a distributed algorithm (built in Scala for Spark).

VA+file [57] is a skip-sequential method that improves the accuracy and efficiency of the VA-file [154]. Both techniques create a file that contains quantization-based summarizations of the original multidimensional data. Search proceeds by sequentially reading each summarization, calculating its lower bounding distance to the query, and accessing the original multidimensional vector only if the lower bounding distance is less than the current *best-so-far (bsf)* answer. We greatly improved the performance of the original VA+file by approximating KLT with DFT [57, 101] and implementing it in C instead of Matlab. In the rest of the text, whenever we mention the VA+file, we refer to the modified version.

3.2.3 Extensions of Data Series Methods

We now propose extensions to the data series methods described above, that will allow them to support ϵ-approximate and δ-ϵ-approximate search (in addition to ng-approximate that they already support). Due to space limitations, we only discuss the tree-based methods (such as iSAX2+ and DSTree); skip-sequential techniques (such as VA+file) can be modified following the same ideas.

The exact 1-NN search algorithms of DSTree and iSAX2+ are based on an optimal exact NN algorithm first proposed

---

Algorithm 1 exactNN(S_Q,idx)

1: $bsf.dist \leftarrow \infty$; $bsf.node \leftarrow NULL$;
2: **for** each $rootNode$ in idx **do**
3:     $result.node \leftarrow rootNode$;
4:     $result.dist \leftarrow$ calcMinDist(S_Q,$rootNode$);
5:     push $result$ to $pqueue$
6: $bsf \leftarrow$ ng-approxNN(S_Q,idx);
7: add bsf to $pqueue$;
8: **while** $result \leftarrow$ pop next node from $pqueue$ **do**
9:     $n \leftarrow result.node$;
10:     **if** $n.dist > bsf.dist$ **then** break;
11:     **if** n is a leaf **then**　　　　　　▷ a leaf node
12:         **for** each S_C in n **do**
13:             $realDist \leftarrow$ calcRealDist(S_Q,S_C);
14:             **if** $realDist < bsf.dist$ **then**
15:                 $bsf.dist \leftarrow realDist$;
16:                 $bsf.node \leftarrow n$;
17:     **else**　　　　　　　　　　▷ an internal node
18:         **for** each $childNode$ in n **do**
19:             $minDist \leftarrow$ calcMinDist(S_Q,$childNode$);
20:             **if** $minDist < bsf.dist$ **then** add $childNode$ to
21:                 $pqueue$ with priority $minDist$;
22: **return** bsf

---

for PMR-Quadtree [69], which was then generalized for any hierarchical index structure that is constructed using a conservative and recursive partitioning of the data [24].

Algorithm 1 describes an index-invariant algorithm for exact 1-NN search. It takes as arguments a query S_Q and an index idx. Lines 1-5 initialize the *best-so-far (bsf)* answer and a priority queue with the root node(s) of the index in increasing order of lower bounding (*lb*) distances (the *lb* distance is calculated by the function *calcMinDist*). In line 6, the *ng*-approxNN function traverses one path of the index tree visiting one leaf to return an *ng*-approximate bsf answer, which is added to the queue (line 7). In line 8, the algorithm pops nodes from the queue, terminating in line 10 if the *lb* distance of the current node is greater than the current *bsf* distance (the *lb* distances of all remaining nodes in the queue are also greater than the *bsf*). Otherwise, if the node is a leaf, the *bsf* is updated if a better answer is found (lines 11-16); if the node is an internal node, its children are added to the queue provided their *lb* distances are greater than the *bsf* distance (lines 18-21).

We can use Algorithm 1 for *ng*-approximate search, by visiting one leaf and returning the first *bsf*. This *ng*-approximate answer can be anywhere in the data space

We extend approximate search in Algorithm 1 by introducing two changes: (i) allow the index to visit up to *nprobe* leaves (user parameter); and (ii) apply the modifications suggested in [43] to support δ-ϵ-approximate NN search. The first change is straightforward, so we only describe the second change in Algorithm 2. To return the ϵ-approximate NN of S_Q, S_ϵ, $bsf.dist$ is replaced with $bsf.dist/(1 + \epsilon)$ in lines 10 and 20. To return the δ-ϵ-approximate NN of S_Q, $S_{\delta\epsilon}$, we also modify lines 1 and 16.

The distance $r_\delta(Q)$ is initialized in line 1 using $F_Q(\cdot)$, S_Q and δ. $F_Q(\cdot)$ represents the relative distance distribution of S_Q. Intuitively, $r_\delta(Q)$ is the maximum distance from S_Q, such that the sphere with center S_Q and radius $r_\delta(Q)$ is empty with probability δ. As proposed in [45], we use $F(\cdot)$, the overall distance distribution, instead of $F_Q(\cdot)$ to estimate $r_\delta(Q)$. The delta radius $r_\delta(Q)$ is then used in line 16 as a stopping condition. When $\delta = 1$, Algorithm 2 returns $S_{\delta\epsilon}$, the ϵ-approximate NN of S_Q, and when $\delta = 1$ and $\epsilon = 0$, Algorithm 2 becomes equivalent to Algorithm 1,

Algorithm 2 <u>delta</u>Epsilon NN(S_Q,idx,δ,ϵ, $F_Q(.)$)

1: $bsf.dist \leftarrow \infty$; $bsf.node \leftarrow NULL$;
   $r_\delta(Q) \leftarrow$ calcDeltaRadius(S_Q,δ, $F_Q(.)$);
2: $bsf \leftarrow$ ng-approxNN(S_Q,idx);
3: add bsf to $pqueue$;
4: **for** each $rootNode$ in idx **do**
5:     $result.node \leftarrow rootNode$;
6:     $result.dist \leftarrow$ calcMinDist(S_Q,$rootNode$);
7:     push $result$ to $pqueue$
8: **while** $result \leftarrow$ pop next node from $pqueue$ **do**
9:     $n \leftarrow result.node$;
10:    **if** $n.dist > bsf.dist/(1+\epsilon)$ **then** break;
11:    **if** n is a leaf **then**   ▷ a leaf node
12:        **for** each S_C in n **do**
13:            $realDist \leftarrow$ calcRealDist(S_Q,S_C);
14:            **if** $realDist < bsf.dist$ **then**
15:                $bsf.dist \leftarrow realDist$;
16:                $bsf.node \leftarrow n$;
                   if $bsf.dist \leq (1+\epsilon)\ r_\delta(Q)$ **then** exit;
17:    **else**   ▷ an internal node
18:        **for** each $childNode$ in n **do**
19:            $minDist \leftarrow$ calcMinDist(S_Q,$childNode$);
20:            **if** $minDist < bsf.dist/(1+\epsilon)$ **then** add
21:                $childNode$ to $pqueue$ with priority $minDist$;
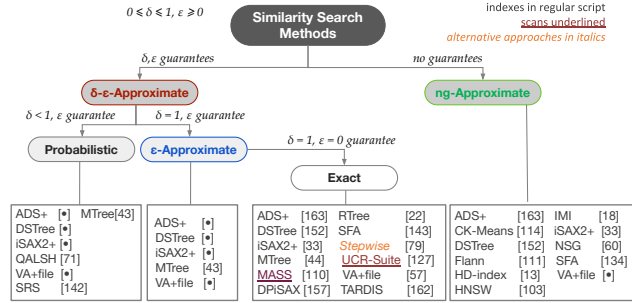22: **return** bsf

Figure 1: Taxonomy of similarity search methods.

i.e., it returns S_x, the exact NN of S_Q. Our implementations generalize Algorithm 2 to the case of $k \geq 1$. These modifications are straightforward and omitted for the sake of brevity. A proof of correctness for Algorithm 2 can be found in [43, 42] for $k = 1$ and $k \geq 1$, respectively.

3.3 Taxonomy of Similarity Search Methods

Figure 1 presents a taxonomy of similarity search methods based on the type of guarantees they provide (methods with multiple types of guarantees are included in more than one leaf of the taxonomy). We call probabilistic the general δ-ϵ-approximate methods. When $\delta = 1$ we have the ϵ-approximate methods. Setting $\delta = 1$ and $\epsilon = 0$, we get the exact methods. Finally, methods that provide no guarantees are categorized under ng-approximate. Here, we cover 7 state-of-the-art methods from the high-dimensional literature, Flann, HD-index, HNSW, IMI, NSG, QALSH and SRS, as well as the 3 best methods from the data series community [54], iSAX2+, DSTree and VA+file.

4. EXPERIMENTAL EVALUATION

We assessed all methods on the same framework. Source code, datasets, queries, and all results are available in [6].

4.1 Experimental Setup

Environment. All methods were compiled with GCC 6.2.0 under Ubuntu Linux 16.04.2 with their default compilation flags; optimization level was set to 2. Experiments were run on a server with two Intel Xeon E5-2650 v4 2.2GHz CPUs, 75GB[3] of RAM, and 10.8TB (6 x 1.8TB) 10K RPM SAS hard drives in RAID0 with a throughput of 1290 MB/sec.

Algorithms. We use the most efficient C/C++ implementation available for each method: iSAX2+ [2], DSTree [2] and VA+file [2] representing exact data series methods with support for approximate queries; and HNSW [5], Faiss IMI [4], SRS [149], FLANN [111], and QALSH [71] representing strictly approximate methods for vectors. We ran experiments with the HD-index [13] and NSG [60], but since they could not scale for our smallest 25GB dataset, we do not report results for them. We extended DSTree, iSAX2+ and VA+file with Algorithm 2, approximating r_δ with density histograms on a 100K data series sample, following the C++ implementation of [43]. All methods are single core implementations, except for HNSW and IMI that make use of multi-threading and SIMD vectorization. Data series points are represented using single precision values and methods based on fixed summarizations use 16 dimensions.

Datasets. We use synthetic and real datasets. Synthetic datasets, called *Rand*, were generated as random-walks using a summing process with steps following a Gaussian distribution (0,1). Such data model financial time series [56] and have been widely used in the literature [56, 33, 165]. Our four real datasets cover domains as varied as deep learning, computer vision, seismology, and neuroscience. *Deep1B* [140] comprises 1 billion vectors of size 96 extracted from the last layers of a convolutional neural network. *Sift1B* [75, 146] consists of 1 billion SIFT vectors of size 128 representing image feature descriptions. To the best of our knowledge, these two vector datasets are the largest publicly available real datasets. *Seismic100GB* [59], contains 100 million data series of size 256 representing earthquake recordings at seismic stations worldwide. *Sald100GB* [148] contains neuroscience MRI data and includes 200 million data series of size 128. In our experiments, we vary the size of the datasets from 25GB to 250GB. The name of each dataset is suffixed with its size. We do not use other real datasets that have appeared in the literature [40, 17], because they are very small, not exceeding 1GB in size.

Queries. All our query workloads consist of 100 query series run asynchronously, i.e., not in batch mode. Synthetic queries were generated using the same random-walk generator as the *Rand* dataset (with a different seed, reported in [6]). For the Deep1B and Sift1B datasets, we randomly select 100 queries from the real workloads that come with the datasets archives. For the other real datasets, query workloads were generated by adding progressively larger amounts of noise to data series extracted from the raw data, so as to produce queries having different levels of difficulty, following the ideas in [164]. Our experiments cover *ng*-approximate and δ-ϵ-approximate k-NN queries, where k $\in [1, 100]$. We also include results for exact queries to serve as a yardstick.

Scenarios. Our experimental evaluation proceeds in four main steps: (i) we tune methods to their optimal parame-

---

ters (§4.2.1); (ii) we evaluate the indexing scalability of the methods (§4.2.2); (iii) we compare in-memory and out-of-core scalability and accuracy of all methods (§4.2.3-§4.2.4); and (iv) we perform additional experiments on the best performing methods for disk-resident data (§4.2.4).

Measures. We assess methods using the following criteria: (1) Scalability and search efficiency using: *wall clock time* (input, output, CPU and total time), *throughput* (# of queries answered per minute), and two implementation-independent measures: the *number of random disk accesses* (# of disk seeks) and the *percentage of data accessed*.
(2) Search accuracy is assessed using: *Avg_Recall*, *Mean Average Precision (MAP)*, and *Mean Relative Error (MRE)*. Recall is the most commonly used accuracy metric in the approximate similarity search literature. However, since it does not consider rank accuracy, we also use MAP [147] that is popular in information retrieval [104, 30] and has been proposed recently in the high-dimensional community [13] as an alternative accuracy measure to recall. For a workload of queries $S_{Q_i} : i \in [1, N_Q]$, these are defined as follows.

- $Avg\_Recall(workload) = \sum_{i=1}^{N_Q} Recall(S_{Q_i})/N_Q$
- $MAP(workload) = \sum_{i=1}^{N_Q} AP(S_{Q_i})/N_Q$
- $MRE(workload) = \sum_{i=1}^{N_Q} RE(S_{Q_i})/N_Q$

where:
- $Recall(S_{Q_i}) = \frac{\text{\# true neighbors returned by } Q_i}{k}$
- $AP(S_{Q_i}) = \frac{\sum_{r=1}^{k}(P(S_{Q_i}, r) \times rel(r))}{k}, \forall i \in [1, N_Q]$
  - $P(S_{Q_i}, r) = \frac{\text{\# true neighbors among the first } r \text{ elements}}{r}$.
  - $rel(r)$ is equal 1 if the neighbor returned at position r is one of the k exact neighbors of S_{Q_i} and 0 otherwise.
- $RE(S_{Q_i}) = \frac{1}{k} \times \sum_{r=1}^{k} \frac{d(S_{Q_i}, S_{C_r}) - d(S_{Q_i}, S_{C_i})}{d(S_{Q_i}, S_{C_i})}$. S_{C_i} is the exact nearest neighbor of S_{Q_i} and S_{C_r} is the r-th NN retrieved[4]. Without loss of generality, we do not consider the case where $d(S_{Q_i}, S_{C_i}) = 0$. (i.e., range queries with radius zero, or kNN queries where the 1-NN is the query itself[5].)
(3) Size, using the *main memory* footprint of the algorithm.

Procedure. Experiments involve two steps: index building and query answering. Caches are fully cleared before each step, and stay warm between consecutive queries. For large datasets that do not fit in memory, the effect of caching is minimized for all methods. All experiments use workloads of 100 queries. Results reported for workloads of 10K queries are extrapolated: we discard the 5 best and 5 worst queries of the original 100 (in terms of total execution time), and multiply the average of the 90 remaining queries by 10K.

4.2 Results

4.2.1 Parametrization

We start by fine tuning each method (graphs omitted for brevity). In order to understand the speed/accuracy trade-offs, we fix the total memory size available to 75GB. The optimal parameters for DSTree, iSAX2+ and VA+file are set according to [54]. For indexing, the buffer and leaf sizes are set to 60GB and 100K, respectively, for both DSTree and iSAX2+. iSAX2+ is set to use 16 segments. VA+file uses a 20GB buffer and 16 DFT symbols. For SRS, we set M (the projected space dimensionality) to 16 so that the representations of all datasets fit in memory. The settings

were the same for all datasets. The fine tuning for HNSW and IMI proved more tricky and involved many testing iterations since the index building parameters strongly affect the speed/accuracy of query answering and differ greatly across datasets. For this reason, different parameters were chosen for different datasets. For the in-memory method HNSW, we set efConstruction (the number of neighbors considered during index construction) to 500, and M (the number of bi-directional edges created for every new node during indexing) to 4 for the Rand25GB dataset. For Deep25GB and Sift25GB, we set efConstruction to 500 and M to 16. To tune the Faiss implementation of IMI, we followed the guidelines in [4]. For the in-memory datasets, we set the index factory key to PQ32_128,IMI2x12,PQ32 and the training size to 1048576, while for disk based datasets, the index key is PQ32_128,IMI2x14,PQ32 and the training size 4194304. To tune δ-ϵ-approximate search performance and accuracy, we vary δ and ϵ for SRS and ϵ for DSTree, iSAX2+ and VA+file (except in one experiment where we also vary δ). For *ng*-approximate search, we vary the *nprobe* parameter for DSTree/iSAX2+/IMI/VA+file (*nprobe* represents the number of visited leaves for DSTree/iSAX2+, the number of visited raw series for VA+file, and the number of inverted lists for IMI), and the *efs* parameter for HNSW (which represents the number of non-pruned candidates).

4.2.2 Indexing Efficiency

In this section, we evaluate the indexing scalability of each method by varying the dataset size. We used four synthetic datasets of sizes 25GB, 50GB, 100GB and 250GB, two of which fit in memory (total RAM was 75GB).

Figure 2a shows that iSAX2+ is the fastest method at index building in and out of memory, followed by VA+file, SRS, DSTree, FLANN, QALSH, IMI and HNSW. Even though IMI and HNSW are the only parallel methods, they are the slowest at index building. Although FLANN is slow at indexing the 50GB dataset, we think this is more due to memory management issues in the code, which cause swapping. For HNSW, the major cost is building the graph structure, whereas IMI spends most of the time on determining the clusters and computing the product quantizers. We also measured the breakdown of the indexing time and found out that all methods can be significantly improved by parallelism except iSAX2+ and QALSH that are I/O bound. In terms of footprint, the DSTree is the most memory-efficient, followed by iSAX2+. IMI, SRS, VA+file and FLANN are two orders of magnitude larger, while QALSH and HNSW are a further order of magnitude bigger (Figure 2b).

(a) Indexing time   (b) Size in memory
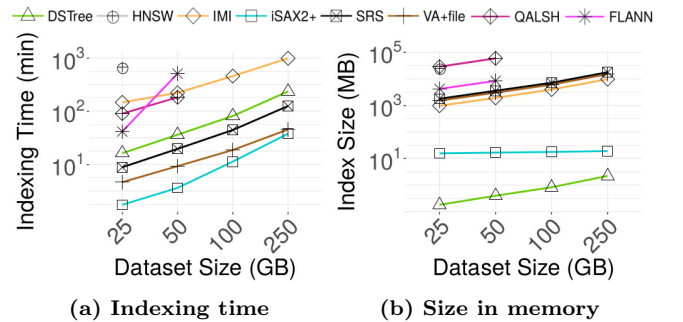
Figure 2: Comparison of indexing scalability

---

[4]Note that in Definition 5, ϵ is an upper bound on $RE(S_{Q_i})$.
[5]In these cases, the MRE definition can be extended to use the symmetric mean absolute percentage error [58].

4.2.3 Query Answering Efficiency and Accuracy: in-Memory Datasets

We now compare query answering efficiency and accuracy, in addition to the indexing time, thus, measuring how well each method amortizes index construction time over a large number of queries, and the level of accuracy achieved.

Summary. For our in-memory experiments, we used four datasets of 25GB each: two synthetic (with series of length 256 and 16384, respectively), and two real: Deep25GB and Sift25GB. We ran 1NN, 10NN and 100NN queries on the four datasets and we observed that, while the running times increase with k, the relative performance of the methods stays the same. Due to lack of space, Figure 3 shows the 100NN query results only (full results are in [6]), which we discuss below. Note that HNSW, QALSH and FLANN store all raw data in-memory, while all other approaches use the memory to store their data structures, but read the raw data from disk; IMI does not access the raw data at all (it only uses the in-memory summaries).

Short Series. For ng-approximate queries of length 256 on the Rand25GB dataset, HNSW has the largest throughput for any given accuracy, followed by FLANN, IMI, DSTree and iSAX2+ (Figure 3a). However, HNSW does not reach a MAP of 1, which is only obtained by the data series indexes (DSTree, iSAX2+, VA+file). The skip-sequential method VA+file performs poorly on approximate search since it prunes per series and not per cluster like the tree-based methods do. When indexing time is also considered, iSAX2+ wins for the workload consisting of 100 queries (Figure 3c), and DSTree for the 10K queries (Figure 3e).

Regarding δ-ϵ-approximate search, DSTree offers the best throughput/accuracy tradeoff, followed by iSAX2+, SRS, VA+file and finally QALSH. SRS does not achieve a MAP higher than 0.5, while DSTree and iSAX2+ are at least 3 times faster than SRS for a similar accuracy (Figure 3b). When we consider the combined indexing and querying times, iSAX2+ wins over all methods for 100 queries (Figure 3d), and DSTree wins for 10K queries (Figure 3f).

Long Series. In this experiment, we use dataset sizes of 25GB, and query length of 16384. For ng-approximate search, we report the results only for iSAX2+, DSTree and VA+file. We ran several experiments with IMI and HNSW building the indexes using different parameters, but obtained a MAP of 0 for IMI for all index configurations we tried, and ran into a segmentation fault during query answering with HNSW. DSTree outperforms both iSAX2+ and VA+file in terms of throughput and combined total cost for the larger workload (Figures 3g and 3k), whereas iSAX2+ wins for the smaller workload when the combined total cost is considered (Figure 3i). We note also that the performance of FLANN deteriorates with the increased dimensionality.

For δ-ϵ-approximate queries, Figure 3h shows that DSTree and VA+file outperform all other methods for large MAP values, while DSTree and iSAX2+ have higher throughput for small MAP values. Note that the SRS accuracy decreases when compared to series of length 256, with the best MAP value now being 0.25. This is due to the increased information loss, as for both series lengths the number of dimensions in the projected space is 16. When index building time is considered, VA+file wins for the small workload (Figure 3j), and iSAX2+ and DSTree win for the large one (Figure 3l). We do not report numbers for QALSH because the algorithm ran into a segmentation fault for series of length 16384.

Real Data. We ran the same set of experiments with real datasets. For ng-approximate queries, HNSW outperforms the query performance of other methods by a large margin (Figures 3m and 3s). When indexing time is considered, HNSW loses its edge due to its high indexing cost to iSAX2+ when the query workload consists of 100 queries (Figures 3o and 3u) and to DSTree for the 10K workload (Figures 3q and 3w). HNSW does not achieve a MAP of 1, while DSTree and ISAX2+ both do, yet at a high cost.

DSTree clearly wins on Sift25GB and Deep25GB among δ-ϵ-approximate methods (Figures 3n, 3t, 3r, and 3x), except for the scenario of indexing plus answering 100 queries, where iSAX2+ has the least combined cost (Figures 3p and 3v). This is because DSTree's query answering is very fast, but its indexing cost is high, so it is amortized only with a large query workload (Figures 3r and 3x). We observe a similar trend for both Sift25GB and Deep25GB, except the degradation of the performance of SRS, which achieves a very low accuracy of 0.01 on Deep25GB, despite using the most restrictive parameters ($\delta = 0.99$ and $\epsilon = 0$).

Comparison of Accuracy Measures. In the approximate similarity search literature, the most commonly used accuracy measures are approximation error and recall. The approximation error evaluates how far the approximate neighbors are from the true neighbors, whereas recall assesses how many true neighbors are returned. In our study, we refer to the recall and approximation error of a workload as Avg_Recall and MRE respectively. In addition, we use a third measure called MAP because it takes into account the order of the returned candidates and thus is more sensitive than recall. Figures 5a and 5b compare all three measures for the popular real dataset Sift25GB (we use the 25GB subset to include in-memory methods as well). We observe that for any given workload, the Avg_Recall is equal to MAP for all methods, except for IMI. This is because IMI returns the short-listed candidates based on distance calculations on the compressed vectors, while the other methods further refine the candidates by sorting them based on the Euclidean distance of the query to the raw data. Figure 5b illustrates the relationship between MAP and MRE. Note that the value of the approximation error is not always indicative of the actual accuracy. For instance, an MRE of about 0.5 for iSAX2 sounds acceptable (some popular LSH methods only work with $\epsilon >= 3$ [144, 61]), yet it corresponds to a very low accuracy of 0.03 as measured by MAP (Figures 5b). Note that MAP can be more useful in practice, since it takes into account the actual ranks of the true neighbors returned, whereas MRE is evaluated only on the distances between the query and its neighbors.

4.2.4 Query Answering Efficiency and Accuracy: on-Disk Datasets

We now report results (Figure 4) for on-disk experiments, excluding the in-memory only HNSW, QALSH and FLANN. **Synthetic Data**. DSTree and iSAX2+ outperform by far the rest of the techniques on both ng-approximate and δ-ϵ-approximate queries. iSAX2+ is particularly competitive when the total cost is considered with the smaller workload (Figures 4c and 4d). The querying performance of SRS degraded on-disk due to severe swapping issues (Figure 4b), therefore we do not include this method in further disk-based experiments. Although IMI is much faster than both iSAX2+ and DSTree on ng-approximate search, its accuracy
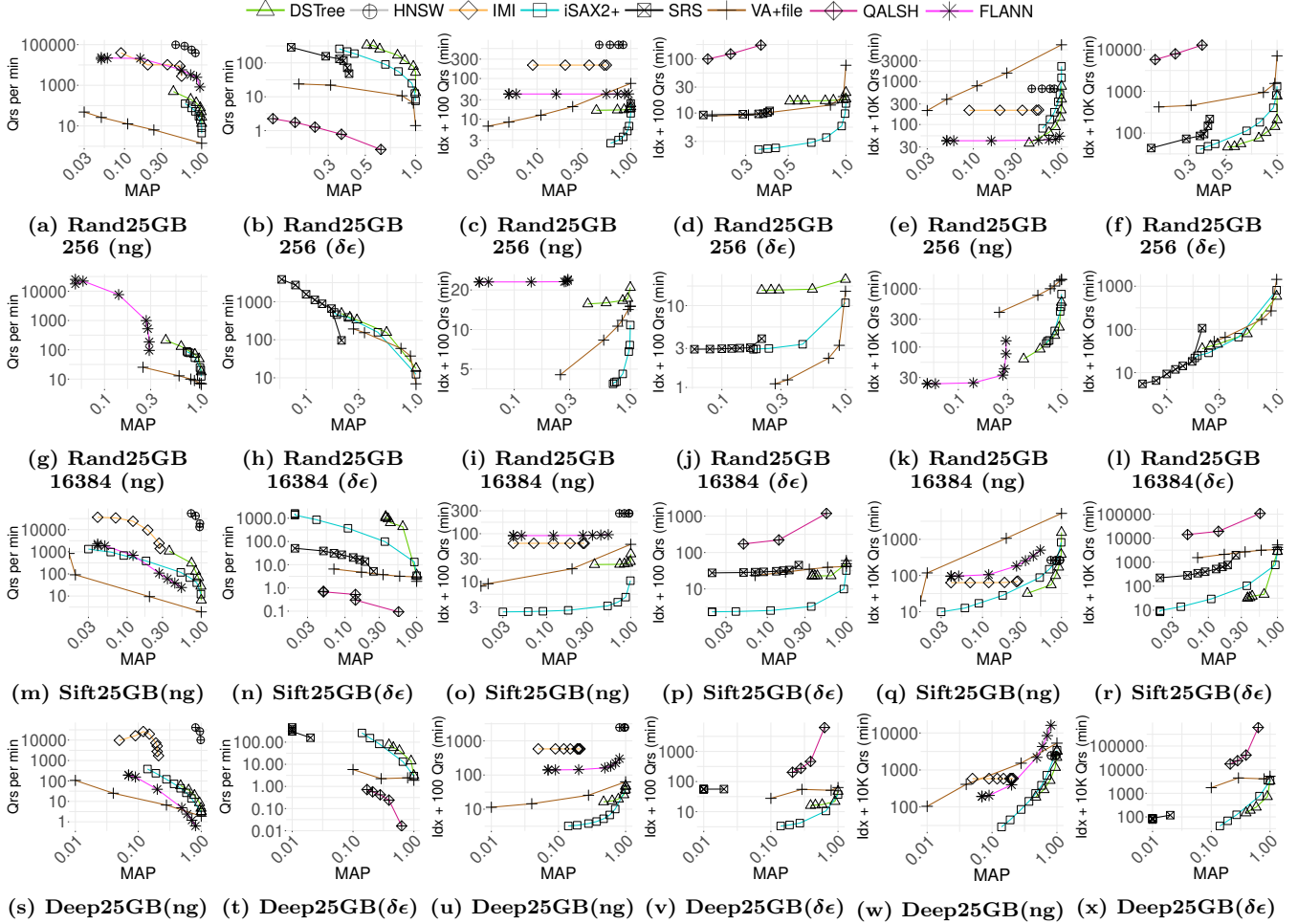
Figure 3: Efficiency vs. accuracy in memory (100NN queries)

(a) **Rand25GB 256 (ng)**   (b) **Rand25GB 256 ($\delta\epsilon$)**   (c) **Rand25GB 256 (ng)**   (d) **Rand25GB 256 ($\delta\epsilon$)**   (e) **Rand25GB 256 (ng)**   (f) **Rand25GB 256 ($\delta\epsilon$)**

(g) **Rand25GB 16384 (ng)**   (h) **Rand25GB 16384 ($\delta\epsilon$)**   (i) **Rand25GB 16384 (ng)**   (j) **Rand25GB 16384 ($\delta\epsilon$)**   (k) **Rand25GB 16384 (ng)**   (l) **Rand25GB 16384($\delta\epsilon$)**

(m) **Sift25GB(ng)**   (n) **Sift25GB($\delta\epsilon$)**   (o) **Sift25GB(ng)**   (p) **Sift25GB($\delta\epsilon$)**   (q) **Sift25GB(ng)**   (r) **Sift25GB($\delta\epsilon$)**

(s) **Deep25GB(ng)**   (t) **Deep25GB($\delta\epsilon$)**   (u) **Deep25GB(ng)**   (v) **Deep25GB($\delta\epsilon$)**   (w) **Deep25GB(ng)**   (x) **Deep25GB($\delta\epsilon$)**

is extremely low. In fact, the best MAP accuracy achieved by IMI plummets to 0.05, whereas DSTree and iSAX2+ have much higher MAP values (Figure 4a).

Real Data. DSTree outperforms all methods on both Sift250GB and Deep250GB. The only exception is iSAX2+ having an edge when the combined indexing and search costs are considered for the smaller workload (Figures 4i, 4j, 4o and 4p) and being equally competitive on *ng*-approximate query answering (Figures 4g, 4h).

Best Performing Methods. The earlier results show that VA+file is outperformed by DSTree and iSAX2+, and that SRS and IMI have very low accuracy on the large datasets. We thus conduct further experiments considering only iSAX2+ and DSTree (recall that HNSW is an in-memory approach only): see Figures 6, 7 and 8.In terms of query efficiency/accuracy tradeoff, DSTree outperforms iSAX2+ on all datasets, except for Sald100GB (Figure 6d), and for low MAP values on Seismic100GB (Figure 6e).

Amount of data accessed. As expected, both DSTree and iSAX2+ need to access more data as the accuracy increases. Nevertheless, we observe that to achieve accuracies of almost 1, both methods access close to 100% of the data for Sift250GB (Figure 6g), Deep250GB (Figure 6h) and Seismic100GB (Figure 6j), compared to 10% of data accessed on Sald100GB (Figure 6i) and Rand250GB. (Figure 6f). The percentage of accessed data also varies among real datasets,

Deep250GB and Sift250GB requiring the most. Note that for some datasets, a MAP of 1 is achievable with minimal data access. For instance DSTree needs to access about 1% of the data to get a MAP of 1 on Sald100GB (Figure 6i).

Number of Random I/Os. To understand the nature of the data accesses discussed above, we report the number of random I/Os in Figure 6 (bottom row). Overall, iSAX2+ incurs a higher number of random I/Os for all datasets. This is because iSAX2+ has a larger number of leaves, with a smaller fill factor than DSTree [54]. For instance, the large number of random I/Os incurred by iSAX2+ (Figure 6o) is what explains the faster runtime of DSTree on the Seismic100GB dataset (Figure 6e), even if DSTree accesses more data than iSAX2+ for higher MAP values (Figure 6j). The Sald100GB dataset is an exception to this trend as iSAX2+ outperforms DSTree on all accuracies except for MAP is 1 (Figure 6d), because it accesses less data incuring almost the same random I/O (Figures 6i and 6n).

Effect of k. Figure 7 summarizes experiments varying k on different datasets in-memory and on-disk. We measure the total time required to complete a workload of 100 queries for each value of k. We observe that finding the first neighbor is the most costly operation, while finding the additional neighbors is much cheaper.

Effect of δ and ϵ. In Figure 8, we describe in more detail how varying δ and ϵ affects the performance of DSTree
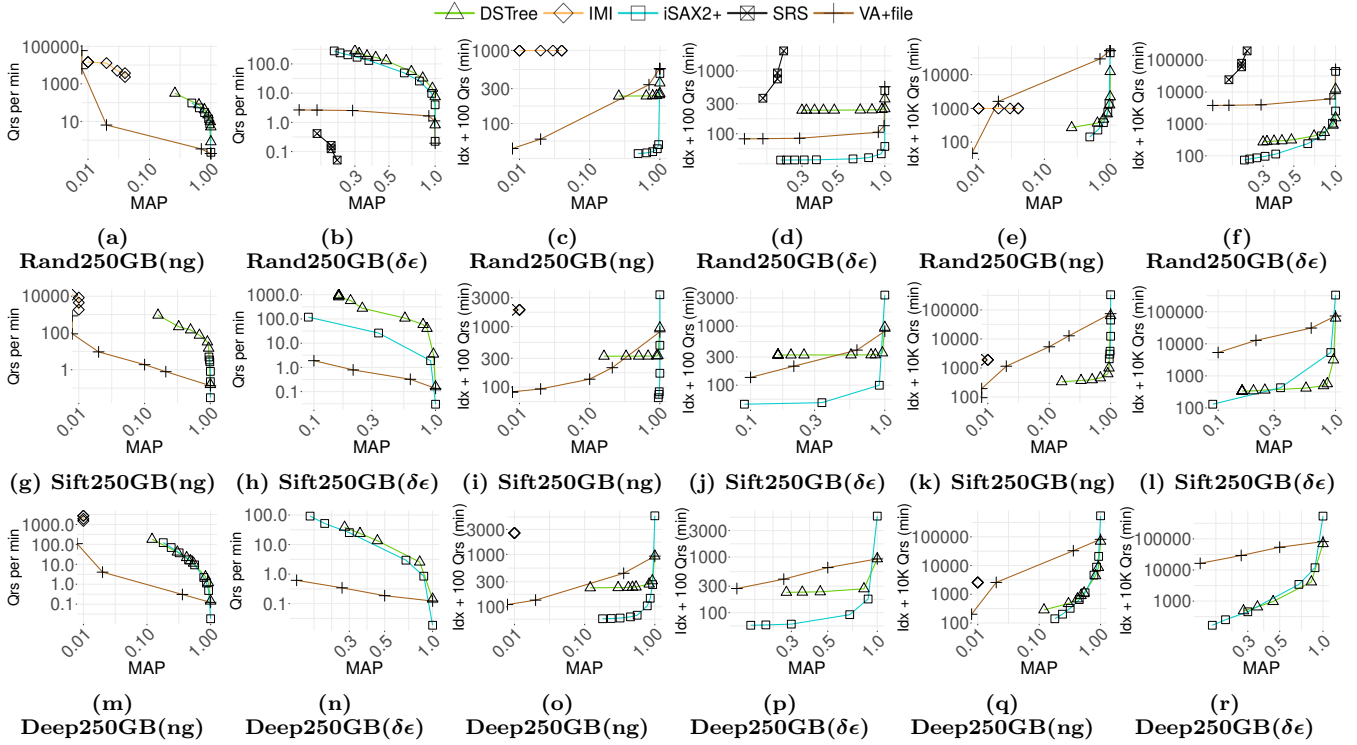
Figure 4: Efficiency vs. accuracy on disk (100NN queries)

and iSAX2+. Figure 8a shows that the throughput of both methods increases dramatically with increasing ϵ. For example, a small value of $\epsilon = 5$ increases the throughput of iSAX2+ by two orders of magnitude, when compared to exact search ($\epsilon = 0$). Moreover, note that both methods return the actual exact answers for small ϵ values, and accuracy drops only as ϵ goes beyond 2 (Figure 8b). In addition, Figure 8c shows that the actual approximation error MRE is well below the user-tolerated threshold (represented by ϵ), even for ϵ values well above 2. The above observations mean that these methods can be used in approximate mode, achieving very high throughput, while still returning answers that are exact (or very close to the exact).

As the probability δ increases, throughput stays constant and only plummets when search becomes exact ($\delta = 1$ in Figure 8d). Similarly, accuracy also stays constant, then slightly increases (for a very high δ of 0.99), reaching 1 for exact search (Figure 8e). Accuracy plateaus as δ increases, because the first *ng*-approximate answer found by both algorithms is very close to the exact answer (Figures 8b and 8c) and better than the approximation of r_δ, thus the stopping condition is never triggered. When a high value of δ is used, the stopping condition takes effect for some queries, but the runtime is very close to that of the exact algorithm.

5. DISCUSSION

In the approximate NN search literature, experimental evaluations ignore the answering capabilities of data series methods. This is the first study that aims to fill this gap.

Unexpected Results. Some of the results are surprising:
(1) *Effectiveness of δ.* LSH techniques (like SRS and QALSH) exploit both δ and ϵ to tune the efficiency/accuracy

tradeoff. We consider that they still fall short of expectations, because for a low ϵ, high values of δ still produce low MAP and low values of δ still result in slow execution (Figure 3). In the case of our extended methods, using ϵ yielded excellent empirical results, but introducing the probabilistic stop condition based on δ was ineffective (Figures 8-d,8-e). We believe that this is due to the inaccuracy of the (histogram-based) approximation of r_δ. Therefore, improving the approximation of r_δ, or devising novel approaches are interesting open research directions that will further improve the efficiency of these methods.

(2) *Approximate Query Answering with Data Series Indexes Performed Better than LSH.* Approximate query answering with DSTree and iSAX2+ outperfom SRS and QALSH (state-of-the-art LSH-based methods) both in space and time, while supporting better theoretical guarantees. This surprising result opens up exciting research opportunities, that is, devising efficient disk-based techniques that support both *ng*-approximate and δ-ϵ-approximate search with top performance [53]. Note that data series indexes developed for distributed platforms [157, 162] also have the potential of outperforming LSH techniques [21, 143] if extended following the ideas discussed in Section 2.

(3) *Our results vs. the literature.* Our results for the in-memory experiments are in-line with those reported in the literature, confirming that HNSW achieves the best accuracy/efficiency tradeoff when only query answering is considered [17] (Figures 3a, 3m, 3s). However, when indexing time is taken into account, HNSW loses its edge to iSAX2+/DSTree for both small (Figures 3c, 3o, 3u) and large (Figures 3e, 3q, 3w) query workloads.

Our results for IMI show a dramatic decrease in accuracy, in terms of MAP and Avg_Recall for the Sift250GB
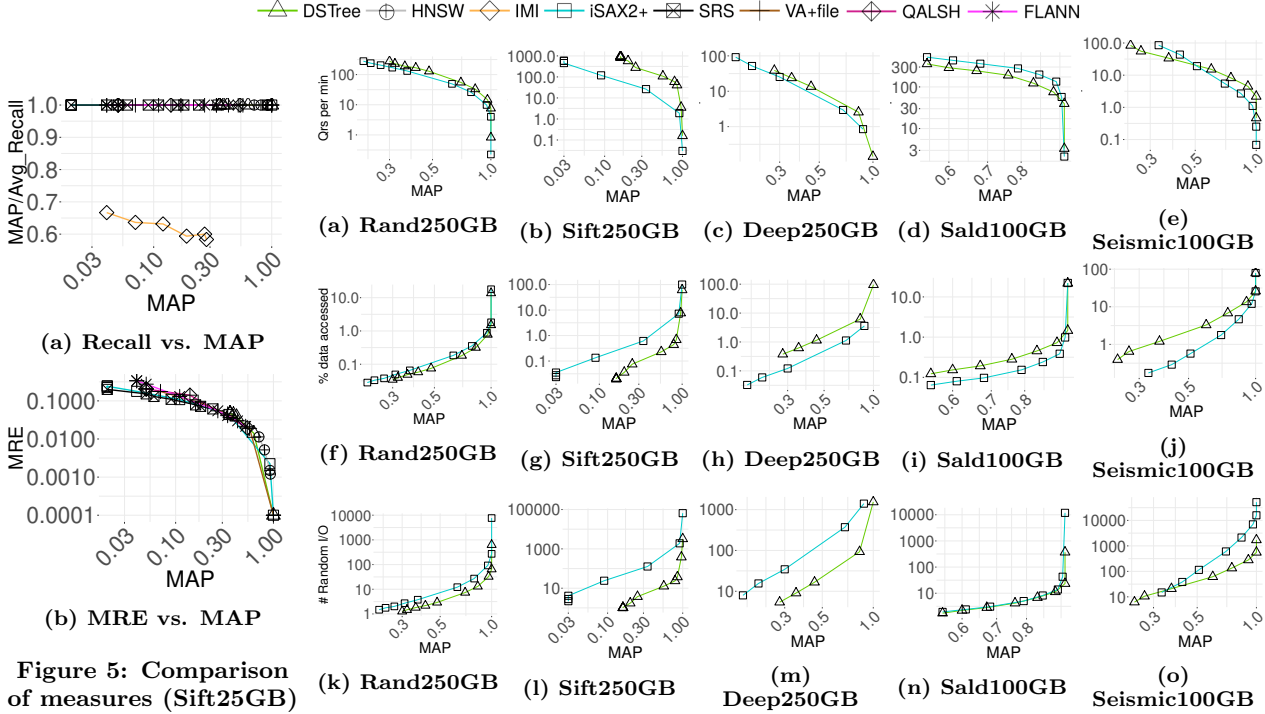
(a) Rand250GB (b) Sift250GB (c) Deep250GB (d) Sald100GB (e) Seismic100GB

(f) Rand250GB (g) Sift250GB (h) Deep250GB (i) Sald100GB (j) Seismic100GB

(k) Rand250GB (l) Sift250GB (m) Deep250GB (n) Sald100GB (o) Seismic100GB

Figure 6: Efficiency vs. accuracy for the best methods (ϵ-approximate)

(a) Recall vs. MAP

(b) MRE vs. MAP

Figure 5: Comparison of measures (Sift25GB)

(a) Rand25GB (b) Sift25GB (c) Deep25GB

(d) Rand250GB (e) Sift250GB (f) Deep250GB

Figure 7: Efficiency vs. k (ϵ-approximate)

(a) Time vs. ϵ ($\delta = 1$) (b) MAP vs. ϵ ($\delta = 1$) (c) MRE vs. ϵ ($\delta = 1$)

(d) Time vs. δ ($\epsilon = 0$) (e) MAP vs. δ ($\epsilon = 0$)

Figure 8: Accuracy and efficiency vs. δ and ϵ

and Deep250GB datasets, while high Avg_Recall values have been reported in the literature for the full Sift1B and Deep1B datasets [159, 4]. We thoroughly investigated the reason behind such a discrepancy and ruled out the following factors: the Z-normalization of the Sift1B/Deep1B datasets, the size of the queries, and the number of NN. We believe that our results are different for the following reasons: (a) our queries return only the number of NN requested, while the smallest candidate list considered in [159] is 10,000 for a 1-NN query; and (b) the results in [4] were obtained using training on a GPU with un-reported training sizes and times (we believe both were very large), while our focus was to evaluate methods on a CPU and account for training time. The difference in the accuracy results is most prob-
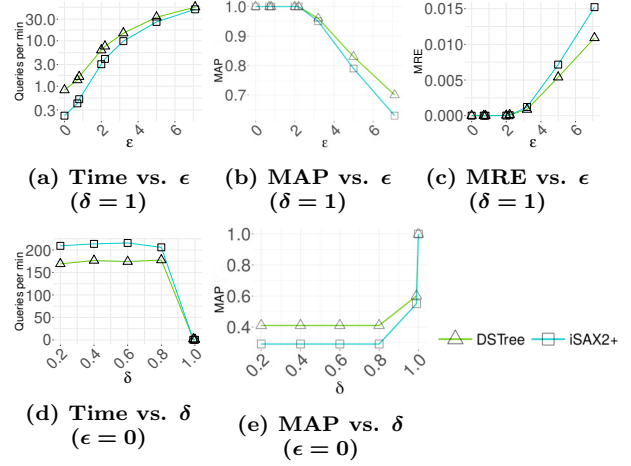
ably due to the fact that the training samples used in [4] were larger than the recommended numbers we used (1 million/4 million for the 25GB/250GB datasets, respectively). We tried to support this claim by running experiments with different training sizes: (i) we observed that increasing the training sizes for the smaller datasets improves the accuracy (the best results are reported in this study); (ii) we could not run experiments on the CPU with large training sizes for the 250GB datasets, because training was very slow: we stopped the execution after 48 hours; (iii) we tried a GPU-enabled server for training, but ran into a documented bug[6]. **Practicality of QALSH, IMI and HNSW.** Although QALSH provides better accuracy than SRS, it does so at a

---
[6]https://github.com/facebookresearch/faiss/issues/67

high cost: it needs to build a different index for each desired query accuracy. This is a serious drawback, while our extended methods offer a neat alternative since the index is built once and the desired accuracy is determined at query time. Although LSH methods (such as SRS) provide guarantees on the accuracy of search results, they are expensive both in time and space. The *ng*-approximate methods overcome these limitations. IMI and HNSW are considered the state-of-the-art in this category, and while they deliver better speed-accuracy tradeoffs than QALSH and SRS, they suffer from two major limitations: (a) having no guarantees can lead them to return incomplete result sets, for instance retrieving only a subset of the neighbors for a k-NN query and returning null values for the others; (b) they are very difficult to tune, which hinders their practicality. In fact, the speed-accuracy tradeoff is not determined only at query time, but also during index building, which means that an index may need to be built many times using different parameters before finding the right speed-accuracy tradeoff. This means that the optimal settings may differ across datasets, and even for different dataset sizes of the same dataset. Moreover, if the analyst builds an index with a particular accuracy target, and then their needs change, they will have to rebuild the index from scratch and go through the same process of determining the right parameter values.

For example, we built the IMI index for the Deep250GB dataset 8 times. During each run that required over 42 hours, we varied the PQ encoding sizes, the number of centroids, and training sizes but still could not achieve the desired accuracy. Regarding HNSW, we tried three different combinations of parameters (M/efConstruction = 4/500, 16/500, 48/200) for each dataset before choosing the optimal one; each run took over 40 hours on the small Deep25GB. Overall, we observe that using IMI and HNSW in practice is cumbersome and time consuming. Developing auto-tuning methods for these techniques is both an interesting problem and a necessity.

Importance of guarantees. In the approximate search literature, accuracy has been evaluated using recall, and approximation error. LSH techniques are considered the state-of-the-art in approximate search with theoretically proven sublinear time performance and probabilistic guarantees on accuracy (approximation error). Our results indicate that using the approximate search functionality of data series techniques provides tighter bounds than LSH (since δ can be equal to 1), and a much better performance in practice, with experimental accuracy levels well above the theoretical accuracy guarantees (Figure 8c). Note that LSH techniques can only provide probabilistic answers ($\delta < 1$), whereas our extended methods can also answer exact and ϵ-approximate queries ($\delta = 1$). A promising research direction is to improve the existing guarantees on these new methods, or establish additional ones: (1) by adding guarantees on query time performance; or (2) by developing probabilistic or deterministic guarantees on the recall or MAP value of a result set, instead of the commonly used distance approximation error. Remember that recall and MAP are better indicators of accuracy, because even small approximation errors may still result in low recall/MAP values (Figure 5b).

Improvement of ng-approximate methods. Our results indicate that *ng*-approximate query answering with exact methods offers a viable alternative to existing methods,
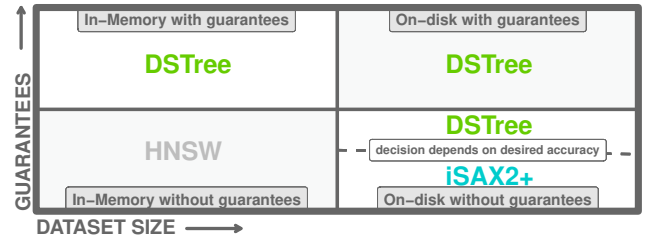

Figure 9: Recommendations (query answering).

particularly because index building is much faster and query efficiency/accuracy tradeoffs can be determined at query time. Besides, the performance of DSTree and iSAX2+ supporting ng-approximate and δ-ϵ-approximate search can be greatly improved by exploiting modern hardware (including SIMD vectorization, multi-cores, multi-sockets, and GPUs).

Incremental approximate k-NN. We established that, on some datasets, a kNN query incurs a much higher time cost as k increases. Therefore, a future research direction is to build δ-ϵ-approximate methods that support incremental search, i.e., returning the neighbors one by one as they are found. The current approaches return the k nearest neighbors all at once which impedes their interactivity.

Progressive Query Answering. The excellent empirical results with approximate search using exact methods paves the way for another very promising research direction: progressive query answering [64]. New approaches can be devised to return intermediate results with increasing accuracy until the exact answers are found.

Recommendations. Choosing the best approach to answer an approximate similarity search query depends on a variety of factors including the accuracy desired, the dataset characteristics, the size of the query workload, the presence of an existing index and the hardware. Figure 9 illustrates a decision matrix that recommends the best technique to use for answering a query workload using an existing index. Overall, DSTree is the best performer, with the exceptions of ng-approximate queries, where iSAX2+ also exhibits excellent performance, and of in-memory datasets, where HNSW is the overall winner. Accounting for index construction time as well, DSTree becomes the method of choice across the board, except for small workloads, where iSAX2+ wins.

6. CONCLUSIONS

We presented a taxonomy for data series similarity search techniques, proposed extensions of exact data series methods that can answer δ-ϵ-approximate queries, and conducted a thorough experimental evaluation of the state-of-the-art techniques from both the data series and vector indexing communities. The results reveal the pros and cons of the various methods, demonstrate the benefits and potential of approximate data series methods, and point to unexplored research directions in the approximate similarity search field.

References

[1] ADHD-200. `http://fcon_1000.projects.nitrc.org/indi/adhd200/`, 2018.

[2] Lernaean Hydra Archive. `http://www.mi.parisdescartes.fr/~themisp/dsseval/`, 2018.

[3] Sloan Digital Sky Survey. `https://www.sdss3.org/dr10/data_access/volume.php`, 2018.

[4] Faiss. `https://github.com/facebookresearch/faiss/`, 2019.

[5] Hnswlib - fast approximate nearest neighbor search. `https://github.com/nmslib/hnswlib`, 2019.

[6] Lernaean Hydra Archive II. `http://www.mi.parisdescartes.fr/~themisp/dsseval2/`, 2019.

[7] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence databases. pages 69–84, 1993.

[8] S. Albrecht, I. Cumming, and J. Dudas. The momentary Fourier transformation derived from recursive matrix transformations. In *Proceedings of 13th International Conference on Digital Signal Processing*, volume 1, pages 337–340 vol.1, Jul 1997.

[9] G. Amato and P. Savino. Approximate Similarity Search in Metric Spaces Using Inverted Files. In *Proceedings of the 3rd International Conference on Scalable Information Systems*, InfoScale '08, pages 28:1–28:10, 2008.

[10] and X. Sean Wang. Supporting content-based searches on time series via approximation. In *Proceedings. 12th International Conference on Scientific and Statistica Database Management*, pages 69–81, July 2000.

[11] A. Andoni, P. Indyk, and I. P. Razenshteyn. Approximate Nearest Neighbor Search in High Dimensions. *CoRR*, abs/1806.09823, 2018.

[12] K. Aoyama, K. Saito, H. Sawada, and N. Ueda. Fast Approximate Similarity Search Based on Degree-reduced Neighborhood Graphs. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '11, pages 1055–1063, New York, NY, USA, 2011. ACM.

[13] A. Arora, S. Sinha, P. Kumar, and A. Bhattacharya. HD-index: Pushing the Scalability-accuracy Boundary for Approximate kNN Search in High-dimensional Spaces. *PVLDB*, 11(8):906–919, 2018.

[14] S. Arya and D. M. Mount. Approximate Nearest Neighbor Queries in Fixed Dimensions. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 271–280, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[15] J. Aßfalg, H. Kriegel, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Similarity Search on Time Series Based on Threshold Queries. In *Advances in Database Technology - EDBT 2006, 10th International Conference on Extending Database Technology, Munich, Germany, March 26-31, 2006, Proceedings*, pages 276–294, 2006.

[16] J. Aßfalg, H. Kriegel, P. Kröger, and M. Renz. Probabilistic Similarity Search for Uncertain Time Series. In *Scientific and Statistical Database Management, 21st International Conference, SSDBM 2009, New Orleans, LA, USA, June 2-4, 2009, Proceedings*, pages 435–443, 2009.

[17] M. Aumüller, E. Bernhardsson, and A. Faithfull. ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. In *Similarity Search and Applications - 10th International Conference, SISAP 2017, Munich, Germany, October 4-6, 2017, Proceedings*, pages 34–49, 2017.

[18] A. Babenko and V. Lempitsky. The Inverted Multi-Index. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(6):1247–1260, June 2015.

[19] M. Bach-Andersen, B. Romer-Odgaard, and O. Winther. Flexible Non-Linear Predictive Models for Large-Scale Wind Turbine Diagnostics. *Wind Energy*, 20(5):753–764, 2017.

[20] A. J. Bagnall, J. Lines, A. Bostrom, J. Large, and E. J. Keogh. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. *Data Min. Knowl. Discov.*, 31(3):606–660, 2017.

[21] B. Bahmani, A. Goel, and R. Shinde. Efficient Distributed Locality Sensitive Hashing. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, CIKM '12, pages 2174–2178, New York, NY, USA, 2012. ACM.

[22] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA*, pages 322–331. ACM, 1990.

[23] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[24] S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel. A Cost Model for Nearest Neighbor Search in High-dimensional Data Space. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '97, pages 78–86, New York, NY, USA, 1997. ACM.

[25] S. Berchtold, C. Böhm, and H.-P. Kriegal. The Pyramid-technique: Towards Breaking the Curse of Dimensionality. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, SIGMOD '98, pages 142–153, New York, NY, USA, 1998. ACM.

[26] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree: An Index Structure for High-Dimensional Data. In *Proceedings of the 22th International Conference on Very Large Data Bases*, VLDB '96, pages 28–39, San Francisco, CA, USA, 1996. Morgan Kaufmann Publishers Inc.

[27] D. J. Berndt and J. Clifford. Using Dynamic Time Warping to Find Patterns in Time Series. In *AAAIWS*, pages 359–370, 1994.

[28] P. Boniol, M. Linardi, F. Roncallo, and T. Palpanas. Automated Anomaly Detection in Large Sequences. *ICDE*, 2020.

[29] A. Broder. On the Resemblance and Containment of Documents. In *Proceedings of the Compression and Complexity of Sequences 1997*, SEQUENCES '97, pages 21–, Washington, DC, USA, 1997. IEEE Computer Society.

[30] C. Buckley and E. M. Voorhees. Evaluating evaluation measure stability. In *SIGIR*, pages 33–40. ACM, 2000.

[31] B. Bustos and G. Navarro. Probabilistic Proximity Searching Algorithms Based on Compact Partitions. *J. of Discrete Algorithms*, 2(1):115–134, Mar. 2004.

[32] A. Camerra, T. Palpanas, J. Shieh, and E. J. Keogh. iSAX 2.0: Indexing and Mining One Billion Time Series. In G. I. Webb, B. Liu, C. Zhang, D. Gunopulos, and X. Wu, editors, *ICDM*, pages 58–67. IEEE Computer Society, 2010.

[33] 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+. *Knowl. Inf. Syst.*, 39(1):123–151, 2014.

[34] K. Chakrabarti, E. Keogh, S. Mehrotra, and M. Pazzani. Locally Adaptive Dimensionality Reduction for Indexing Large Time Series Databases. *ACM Trans. Database Syst.*, 27(2):188–228, June 2002.

[35] V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)*, 41(3):15, 2009.

[36] M. S. Charikar. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the Thiry-fourth Annual ACM Symposium on Theory of Computing*, STOC '02, pages 380–388, New York, NY, USA, 2002. ACM.

[37] E. Chávez and E. Sadit Tellez. Navigating K-Nearest Neighbor Graphs to Solve Nearest Neighbor Searches. In J. F. Martínez-Trinidad, J. A. Carrasco-Ochoa, and J. Kittler, editors, *Advances in Pattern Recognition*, pages 270–280, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[38] E. Chavez Gonzalez, K. Figueroa, and G. Navarro. Effective Proximity Retrieval by Ordering Permutations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30(9):1647–1658, Sept. 2008.

[39] Y. Chen, E. K. Garcia, M. R. Gupta, A. Rahimi, and L. Cazzanti. Similarity-based classification: Concepts and algorithms. *J. Mach. Learn. Res.*, 10:747–776, June 2009.

[40] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The UCR Time Series Classification Archive, July 2015. `www.cs.ucr.edu/~eamonn/time_series_data/`.

[41] Y. Chen, M. A. Nascimento, B. C. Ooi, and A. K. H. Tung. SpADe: On Shape-based Pattern Detection in Streaming Time Series. In *Proceedings of the 23rd International Conference on Data Engineering, ICDE 2007, The Marmara Hotel, Istanbul, Turkey, April 15-20, 2007*, pages 786–795, 2007.

[42] P. Ciaccia and M. Patella. the power of distance distributions: Cost models and scheduling policies for quality-controlled similarity queries.

[43] P. Ciaccia and M. Patella. PAC Nearest Neighbor Queries: Approximate and Controlled Search in High-Dimensional and Metric Spaces. In *ICDE*, pages 244–255, 2000.

[44] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An Efficient Access Method for Similarity Search in Metric Spaces. In M. Jarke, M. Carey, K. R. Dittrich, F. Lo-chovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *Proceedings of the 23rd International Conference on Very Large Data Bases (VLDB'97)*, pages 426–435, Athens, Greece, Aug. 1997. Morgan Kaufmann Publishers, Inc.

[45] P. Ciaccia, M. Patella, and P. Zezula. A Cost Model for Similarity Queries in Metric Spaces. In *Proceedings of the Seventeenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '98, pages 59–68, New York, NY, USA, 1998. ACM.

[46] R. Cole, D. E. Shasha, and X. Zhao. Fast window correlations over uncooperative time series. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 743–749, 2005.

[47] R. Cole, D. E. Shasha, and X. Zhao. Fast window correlations over uncooperative time series. In *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Chicago, Illinois, USA, August 21-24, 2005*, pages 743–749, 2005.

[48] M. Dallachiesa, B. Nushi, K. Mirylenka, and T. Palpanas. Uncertain Time-Series Similarity: Return to the Basics. *PVLDB*, 5(11):1662–1673, 2012.

[49] M. Dallachiesa, T. Palpanas, and I. F. Ilyas. Top-k Nearest Neighbor Search in Uncertain Data Series. *PVLDB*, 8(1):13–24, 2014.

[50] G. Das, D. Gunopulos, and H. Mannila. Finding similar time series. *Principles of Data Mining and Knowledge Discovery*, pages 88–100, 1997.

[51] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive Hashing Scheme Based on P-stable Distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, pages 253–262, New York, NY, USA, 2004. ACM.

[52] 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. *PVLDB*, 1(2):1542–1552, 2008.

[53] K. Echihabi. Truly scalable data series similarity search. In *Proceedings of the VLDB 2019 PhD Workshop, co-located with the 45th International Conference on Very Large Databases (VLDB 2019), Los Angeles, California, USA, August 26-30, 2019.*, 2019.

[54] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. The Lernaean Hydra of Data Series Similarity Search: An Experimental Evaluation of the State of the Art. *PVLDB*, 12(2):112–127, 2018.

[55] ESA. SENTINEL-2 Mission, 2018.

[56] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *SIGMOD*, pages 419–429, New York, NY, USA, 1994. ACM.

[57] H. Ferhatosmanoglu, E. Tuncel, D. Agrawal, and A. El Abbadi. Vector Approximation Based Indexing for Non-uniform High Dimensional Data Sets. In *Proceedings of the Ninth International Conference on Information and Knowledge Management*, CIKM '00, pages 202–209, New York, NY, USA, 2000. ACM.

[58] B. E. Flores. A pragmatic view of accuracy measurement in forecasting. *Omega*, 14(2):93 – 98, 1986.

[59] I. R. I. for Seismology with Artificial Intelligence. Seismic Data Access. `http://ds.iris.edu/data/access/`, 2018.

[60] C. Fu, C. Xiang, C. Wang, and D. Cai. Fast Approximate Nearest Neighbor Search with the Navigating Spreading-out Graph. *PVLDB*, 12(5):461–474, 2019.

[61] J. Gan, J. Feng, Q. Fang, and W. Ng. Locality-sensitive Hashing Scheme Based on Dynamic Collision Counting. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, pages 541–552, New York, NY, USA, 2012. ACM.

[62] T. Ge, K. He, Q. Ke, and J. Sun. Optimized Product Quantization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 36(4):744–755, Apr. 2014.

[63] A. Gionis, P. Indyk, and R. Motwani. Similarity Search in High Dimensions via Hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB '99, pages 518–529, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[64] A. Gogolou, T. Tsandilas, T. Palpanas, and A. Bezerianos. Progressive Similarity Search on Time Series Data. In *Proceedings of the Workshops of the EDBT/ICDT 2019 Joint Conference, EDBT/ICDT 2019, Lisbon, Portugal, March 26, 2019.*, 2019.

[65] X. Golay, S. Kollias, G. Stoll, D. Meier, A. Valavanis, and P. Boesiger. A new correlation-based fuzzy logic clustering algorithm for FMRI. *Magnetic Resonance in Medicine*, 40(2):249–260, 1998.

[66] R. M. Gray and D. L. Neuhoff. Quantization. *IEEE Trans. Inf. Theor.*, 44(6):2325–2383, Sept. 2006.

[67] A. Guttman. R-Trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts, June 18-21, 1984*, pages 47–57, 1984.

[68] G. Hébrail. *Practical data mining in a large utility company*, pages 87–95. Physica-Verlag HD, Heidelberg, 2000.

[69] G. R. Hjaltason and H. Samet. Ranking in Spatial Databases. In *Proceedings of the 4th International Symposium on Advances in Spatial Databases*, SSD '95, pages 83–95, Berlin, Heidelberg, 1995. Springer-Verlag.

[70] M. E. Houle and Jun Sakuma. Fast approximate similarity search in extremely high-dimensional data sets. In *21st International Conference on Data Engineering (ICDE'05)*, pages 619–630, April 2005.

[71] Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng. Query-aware Locality-sensitive Hashing for Approximate Nearest Neighbor Search. *PVLDB*, 9(1):1–12, 2015.

[72] 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.

[73] P. Indyk and R. Motwani. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, STOC '98, pages 604–613, New York, NY, USA, 1998. ACM.

[74] H. Jegou, M. Douze, and C. Schmid. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1):117–128, Jan 2011.

[75] H. Jegou, R. Tavenard, M. Douze, and L. Amsaleg. Searching in one billion vectors: Re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 861–864, May 2011.

[76] Z. Jiang, L. Xie, X. Deng, W. Xu, and J. Wang. Fast Nearest Neighbor Search in the Hamming Space. In *Proceedings, Part I, of the 22Nd International Conference on MultiMedia Modeling - Volume 9516*, MMM 2016, pages 325–336, Berlin, Heidelberg, 2016. Springer-Verlag.

[77] W. Johnson and J. Lindenstrauss. Extensions of Lipschitz mappings into a Hilbert space. In *Conference in modern analysis and probability (New Haven, Conn., 1982)*, volume 26 of *Contemporary Mathematics*, pages 189–206. American Mathematical Society, 1984.

[78] Y. Kakizawa, R. H. Shumway, and M. Taniguchi. Discrimination and clustering for multivariate time series. *Journal of the American Statistical Association*, 93(441):328–340, 1998.

[79] Y. Kalantidis and Y. Avrithis. Locally Optimized Product Quantization for Approximate Nearest Neighbor Search. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2329–2336, June 2014.

[80] K. Kashino, G. Smith, and H. Murase. Time-series active search for quick retrieval of audio and video. In *ICASSP*, 1999.

[81] S. Kashyap and P. Karras. Scalable kNN search on vertically stored time series. In C. Apt, J. Ghosh, and P. Smyth, editors, *KDD*, pages 1334–1342. ACM, 2011.

[82] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra. Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and Information Systems*, 3(3):263–286, 2001.

[83] E. Keogh and S. Kasetty. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. *Data Min. Knowl. Discov.*, 7(4):349–371, Oct. 2003.

[84] E. Keogh and M. Pazzani. An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. In R. Agrawal, P. Stolorz, and G. Piatetsky-Shapiro, editors, *Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 239–241, New York City, NY, 1998. ACM Press.

[85] E. Keogh and P. Smyth. A Probabilistic Approach to Fast Pattern Matching in Time Series Databases. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, KDD'97, pages 24–30. AAAI Press, 1997.

[86] J. Kleinberg. The Small-world Phenomenon: An Algorithmic Perspective. In *Proceedings of the Thirty-*

second Annual ACM Symposium on Theory of Computing, STOC '00, pages 163–170, New York, NY, USA, 2000. ACM.

[87] S. Knieling, J. Niediek, E. Kutter, J. Bostroem, C. Elger, and F. Mormann. An online adaptive screening procedure for selective neuronal responses. *Journal of Neuroscience Methods*, 291(Supplement C):36 – 42, 2017.

[88] H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas. Coconut: A Scalable Bottom-Up Approach for Building Data Series Indexes. *PVLDB*, 11(6):677–690, 2018.

[89] H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas. Coconut palm: Static and streaming data series exploration now in your palm. In *SIGMOD*, pages 1941–1944, 2019.

[90] H. Kondylakis, N. Dayan, K. Zoumpatianos, and T. Palpanas. Coconut: Sortable summarizations for scalable indexes over static and streaming data series. *VLDBJ*, accepted for publication, 2019.

[91] K. Košmelj and V. Batagelj. Cross-sectional approach for clustering time varying data. *Journal of Classification*, 7(1):99–109, 1990.

[92] M. Kumar, N. R. Patel, and J. Woo. Clustering seasonality patterns in the presence of errors. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, July 23-26, 2002, Edmonton, Alberta, Canada*, pages 557–563, 2002.

[93] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin. Approximate Nearest Neighbor Search on High Dimensional Data - Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2019.

[94] J. Lin, E. J. Keogh, S. Lonardi, and B. Y. Chiu. A symbolic representation of time series, with implications for streaming algorithms. In *Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery, DMKD 2003, San Diego, California, USA, June 13, 2003*, pages 2–11, 2003.

[95] M. Linardi and T. Palpanas. Scalable, variable-length similarity search in data series: The ulisse approach. *PVLDB*, 11(13):2236–2248, 2018.

[96] M. Linardi and T. Palpanas. ULISSE: ULtra compact Index for Variable-Length Similarity SEarch in Data Series. In *ICDE*, 2018.

[97] M. Linardi, Y. Zhu, T. Palpanas, and E. J. Keogh. Matrix Profile X: VALMOD - Scalable Discovery of Variable-Length Motifs in Data Series. 2018.

[98] T. Liu, A. W. Moore, A. Gray, and K. Yang. An Investigation of Practical Approximate Nearest Neighbor Algorithms. In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, NIPS'04, pages 825–832, Cambridge, MA, USA, 2004. MIT Press.

[99] Y. Liu, J. Cui, Z. Huang, H. Li, and H. T. Shen. SK-LSH: An Efficient Index Structure for Approximate Nearest Neighbor Search. *PVLDB*, 7:745–756, 2014.

[100] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. Multi-probe LSH: Efficient Indexing for High-dimensional Similarity Search. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 950–961. VLDB Endowment, 2007.

[101] C. Maccone. Advantages of KarhunenLove transform over fast Fourier transform for planetary radar and space debris detection. *Acta Astronautica*, 60(8):775 – 779, 2007.

[102] Y. Malkov, A. Ponomarenko, A. Logvinov, and V. Krylov. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45:61 – 68, 2014.

[103] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs. *CoRR*, abs/1603.09320, 2016.

[104] C. D. Manning, P. Raghavan, and H. Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[105] M. Mannino and A. Abouzied. Qetch: Time Series Querying with Expressive Sketches. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, pages 1741–1744, 2018.

[106] Y. Matsui, Y. Uchida, H. Jégou, and S. Satoh. A Survey of Product Quantization. *ITE Transactions on Media Technology and Applications*, 6(1):2–10, 2018.

[107] K. Mirylenka, V. Christophides, T. Palpanas, I. Pefkianakis, and M. May. Characterizing Home Device Usage From Wireless Traffic Time Series. In *EDBT*, pages 551–562, 2016.

[108] K. Mirylenka, M. Dallachiesa, and T. Palpanas. Data Series Similarity Using Correlation-Aware Measures. In *Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Chicago, IL, USA, June 27-29, 2017*, pages 11:1–11:12, 2017.

[109] R. Motwani, A. Naor, and R. Panigrahy. Lower Bounds on Locality Sensitive Hashing. *SIAM J. Discrete Math.*, 21(4):930–935, 2007.

[110] A. Mueen, Y. Zhu, M. Yeh, K. Kamgar, K. Viswanathan, C. Gupta, and E. Keogh. The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance, August 2017. http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html.

[111] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *VISAPP International Conference on Computer Vision Theory and Applications*, pages 331–340, 2009.

[112] B. Naidan, L. Boytsov, and E. Nyberg. Permutation Search Methods Are Efficient, Yet Faster Search is Possible. *PVLDB*, 8(12):1618–1629, 2015.

[113] G. Navarro. Searching in Metric Spaces by Spatial Approximation. *The VLDB Journal*, 11(1):28–46, Aug. 2002.

[114] M. Norouzi and D. J. Fleet. Cartesian K-Means. In *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '13, pages 3017–3024, 2013.

[115] R. O'Donnell, Y. Wu, and Y. Zhou. Optimal Lower

Bounds for Locality-Sensitive Hashing (Except When Q is Tiny). *ACM Trans. Comput. Theory*, 6(1):5:1–5:13, Mar. 2014.

[116] B. C. Ooi, K.-L. Tan, K.-L. Tan, C. Yu, and S. Bressan. Indexing the Edges&Mdash;a Simple and Yet Efficient Approach to High-dimensional Indexing. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '00, pages 166–174, New York, NY, USA, 2000. ACM.

[117] T. Palpanas. Data Series Management: The Road to Big Sequence Analytics. *SIGMOD Record*, 44(2):47–52, 2015.

[118] T. Palpanas. Big Sequence Management: A glimpse of the Past, the Present, and the Future. In R. M. Freivalds, G. Engels, and B. Catania, editors, *SOFSEM*, volume 9587 of *Lecture Notes in Computer Science*, pages 63–80. Springer, 2016.

[119] T. Palpanas and V. Beckmann. Report on the first and second interdisciplinary time series analysis workshop (itisa). *SIGMOD Rec.*, "Accepted for publication, 2019.

[120] R. Panigrahy. Entropy Based Nearest Neighbor Search in High Dimensions. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, SODA '06, pages 1186–1195, Philadelphia, PA, USA, 2006. Society for Industrial and Applied Mathematics.

[121] P. Paraskevopoulos, T.-C. Dinh, Z. Dashdorj, T. Palpanas, and L. Serafini. Identification and Characterization of Human Behavior Patterns from Mobile Phone Data. In *D4D Challenge session, NetMob*, 2013.

[122] B. Peng, P. Fatourou, and T. Palpanas. ParIS: The Next Destination for Fast Data Series Indexing and Query Answering. *IEEE BigData*, 2018.

[123] B. Peng, P. Fatourou, and T. Palpanas. MESSI: In-Memory Data Series Indexing. *ICDE*, 2020.

[124] D. Rafiei. On Similarity-Based Queries for Time Series Data. In *Proceedings of the 15th International Conference on Data Engineering, Sydney, Austrialia, March 23-26, 1999*, pages 410–417, 1999.

[125] D. Rafiei and A. Mendelzon. Similarity-based Queries for Time Series Data. *SIGMOD Rec.*, 26(2):13–25, June 1997.

[126] D. Rafiei and A. O. Mendelzon. Efficient Retrieval of Similar Time Sequences Using DFT. *CoRR*, cs.DB/9809033, 1998.

[127] T. Rakthanmanon, B. J. L. Campana, A. Mueen, G. E. A. P. A. 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 Q. Yang, D. Agarwal, and J. Pei, editors, *KDD*, pages 262–270. ACM, 2012.

[128] T. Rakthanmanon, E. J. Keogh, S. Lonardi, and S. Evans. Time series epenthesis: Clustering time series streams requires ignoring some data. In *Data Mining (ICDM), 2011 IEEE 11th International Conference on*, pages 547–556. IEEE, 2011.

[129] 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.*, 27(8), 2015.

[130] P. P. Rodrigues, J. Gama, and J. P. Pedroso. ODAC: Hierarchical Clustering of Time Series Data Streams. In J. Ghosh, D. Lambert, D. B. Skillicorn, and J. Srivastava, editors, *SDM*, pages 499–503. SIAM, 2006.

[131] G. Ruiz, E. Chávez, M. Graff, and E. S. Téllez. Finding Near Neighbors Through Local Search. In *Proceedings of the 8th International Conference on Similarity Search and Applications - Volume 9371*, SISAP 2015, pages 103–109, Berlin, Heidelberg, 2015. Springer-Verlag.

[132] H. Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[133] S. R. Sarangi and K. Murthy. DUST: a generalized notion of similarity between uncertain time series. In *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25-28, 2010*, pages 383–392, 2010.

[134] P. Schäfer and M. Högqvist. SFA: A Symbolic Fourier Approximation and Index for Similarity Search in High Dimensional Datasets. In *Proceedings of the 15th International Conference on Extending Database Technology*, EDBT '12, pages 516–527, New York, NY, USA, 2012. ACM.

[135] D. Shasha. Tuning Time Series Queries in Finance: Case Studies and Recommendations. *IEEE Data Eng. Bull.*, 22(2):40–46, 1999.

[136] H. Shatkay and S. B. Zdonik. Approximate queries and representations for large data sequences. In *Proceedings of the Twelfth International Conference on Data Engineering*, pages 536–545, Feb 1996.

[137] 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, New York, NY, USA, 2008. ACM.

[138] 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, New York, NY, USA, 2008. ACM.

[139] C. Silpa-Anan and R. Hartley. Optimised KD-trees for fast image descriptor matching. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, June 2008.

[140] Skoltech Computer Vision. Deep billion-scale indexing. http://sites.skoltech.ru/compvision/noimi, 2018.

[141] S. Soldi, V. Beckmann, W. Baumgartner, G. Ponti, C. R. Shrader, P. Lubiński, H. Krimm, F. Mattana, and J. Tueller. Long-term variability of AGN at hard X-rays. *Astronomy & Astrophysics*, 563:A57, 2014.

[142] Y. Sun, W. Wang, J. Qin, Y. Zhang, and X. Lin. SRS: Solving c-approximate Nearest Neighbor Queries in High Dimensional Euclidean Space with a Tiny Index. *PVLDB*, 8(1):1–12, 2014.

[143] N. Sundaram, A. Turmukhametova, N. Satish,

T. Mostak, P. Indyk, S. Madden, and P. Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *PVLDB*, 6(14):1930–1941, 2013.

[144] Y. Tao, K. Yi, C. Sheng, and P. Kalnis. Efficient and Accurate Nearest Neighbor and Closest Pair Search in High-dimensional Space. *ACM Trans. Database Syst.*, 35(3):20:1–20:46, July 2010.

[145] E. S. Tellez, E. Chávez, and G. Navarro. Succinct Nearest Neighbor Search. In *Proceedings of the Fourth International Conference on SImilarity Search and APplications*, SISAP '11, pages 33–40, New York, NY, USA, 2011. ACM.

[146] TEXMEX Research Team. Datasets for approximate nearest neighbor search. `http://corpus-texmex.irisa.fr/`, 2018.

[147] A. Turpin and F. Scholer. User Performance Versus Precision Measures for Simple Search Tasks. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 11–18, New York, NY, USA, 2006. ACM.

[148] 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=See%20Data&utm_campaign=indi-1`, 2018.

[149] D. W. G. UNSW. SRS - Fast Approximate Nearest Neighbor Search in High Dimensional Euclidean Space With a Tiny Index. `https://github.com/DBWangGroupUNSW/SRS`, 2019.

[150] J. Wang, J. Wang, G. Zeng, R. Gan, S. Li, and B. Guo. Fast Neighborhood Graph Search Using Cartesian Concatenation. In *2013 IEEE International Conference on Computer Vision*, pages 2128–2135, Dec 2013.

[151] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. Experimental Comparison of Representation Methods and Distance Measures for Time Series Data. *Data Min. Knowl. Discov.*, 26(2):275–309, Mar. 2013.

[152] 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.

[153] T. Warren Liao. Clustering of time series dataa survey. *Pattern Recognition*, 38(11):1857–1874, 2005.

[154] R. Weber, H.-J. Schek, and S. Blott. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pages 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[155] B. M. Williams and L. A. Hoel. Modeling and forecasting vehicular traffic flow as a seasonal arima process: Theoretical basis and empirical results. *Journal of Transportation Engineering*, 129(6):664–672, 2003.

[156] Y. Xia, K. He, F. Wen, and J. Sun. Joint Inverted Indexing. *2013 IEEE International Conference on Computer Vision*, pages 3416–3423, 2013.

[157] D. E. Yagoubi, R. Akbarinia, F. Masseglia, and T. Palpanas. DPiSAX: Massively Distributed Partitioned iSAX. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 1135–1140, 2017.

[158] D.-E. Yagoubi, R. Akbarinia, F. Masseglia, and T. Palpanas. Massively distributed time series indexing and querying. *TKDE (to appear)*, 2019.

[159] A. B. Yandex and V. Lempitsky. Efficient Indexing of Billion-Scale Datasets of Deep Descriptors. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2055–2063, June 2016.

[160] 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, 12th International Conference on Extending Database Technology, Saint Petersburg, Russia, March 24-26, 2009, Proceedings*, pages 684–695, 2009.

[161] C. Yu, B. C. Ooi, K.-L. Tan, and H. V. Jagadish. Indexing the Distance: An Efficient Method to KNN Processing. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 421–430, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[162] L. Zhang, N. Alghamdi, M. Y. Eltabakh, and E. A. Rundensteiner. TARDIS: Distributed Indexing Framework for Big Time Series Data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1202–1213, April 2019.

[163] K. Zoumpatianos, S. Idreos, and T. Palpanas. ADS: the adaptive data series index. *The VLDB Journal*, 25(6):843–866, 2016.

[164] K. Zoumpatianos, Y. Lou, I. Ileana, T. Palpanas, and J. Gehrke. Generating data series query workloads. *The VLDB Journal*, 27(6):823–846, Dec. 2018.

[165] K. Zoumpatianos, Y. Lou, T. Palpanas, and J. Gehrke. Query Workloads for Data Series Indexes. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, pages 1603–1612, 2015.