ELPIS: Graph-Based Similarity Search for Scalable Data Science

Ilias Azizi
UM6P, Université Paris Cité
ilias.azizi@um6p.ma

Karima Echihabi
UM6P
karima.echihabi@um6p.ma

Themis Palpanas
Université Paris Cité & IUF
themis@mi.parisdescartes.fr

ABSTRACT

The recent popularity of learned embeddings has fueled the growth of massive collections of high-dimensional (high-d) vectors that model complex data. Finding similar vectors in these collections is at the core of many important and practical data science applications. The data series community has developed tree-based similarity search techniques that outperform state-of-the-art methods on large collections of both data series and generic high-d vectors, on all scenarios except for no-guarantees *ng*-approximate search, where graph-based approaches designed by the high-d vector community achieve the best performance. However, building graph-based indexes is extremely expensive both in time and space. In this paper, we bring these two worlds together, study the corresponding solutions and their performance behavior, and propose ELPIS, a new strong baseline that takes advantage of the best features of both to achieve a superior performance in terms of indexing and ng-approximate search in-memory. ELPIS builds the index 3x-8x faster than competitors, using 40% less memory. It also achieves a high recall of 0.99, up to 2x faster than the state-of-the-art methods, and answers 1-NN queries up to one order of magnitude faster.

PVLDB Reference Format:
Ilias Azizi, Karima Echihabi, and Themis Palpanas. "ELPIS: Graph-Based Similarity Search for Scalable Data Science". PVLDB, 16(6): 1548 - 1559, 2023.
doi:10.14778/3583140.3583166

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at http://www.mi.parisdescartes.fr/~themisp/elpis.

1 INTRODUCTION

High-dimensional (high-d) data abounds in many real applications including agriculture, biology, finance, and smart cities, with terabyte-scale data collections reaching thousands of dimensions. Extracting value from these collections requires complex data analytical tasks. This is challenging to achieve efficiently because of the large volume and dimensionality of the data. A key component of these tasks is *similarity search*, a fundamental algorithm that is the focus of our study, and at the core of many important and practical data science applications. In data integration, it has been used to automate entity resolution [32], complete missing values [55], and support data discovery [18, 81, 127]. It has powered recommender systems of online billion-dollar enterprises [76, 117], and enabled information retrieval [123], classification [37, 96] and outlier detection [11–14, 75, 88, 89]. Similarity search has also been exploited in software engineering [3, 85] to automate API mappings and predict program dependencies and I/O usage and in cybersecurity to profile network usage and detect intrusions and malware [31].

Similarity search finds the most similar objects in a dataset to a given query object. It is often reduced to k-nearest neighbor (k-NN) search, which represents the objects as points in R^d space, and returns the k closest vectors in the dataset $\mathbb{S}$ to a given query vector V_Q according to some distance measure, such as the Euclidean distance. The brute force approach for k-NN search consists of comparing V_Q to every single candidate in $\mathbb{S}$. Supporting efficient similarity search on high-d data relies on devising accurate dimensionality reduction techniques and developing specialized data structures and algorithms [41, 42]. Exact solutions return the true k-NN of V_Q while approximate methods sacrifice accuracy for efficiency. We call approximate solutions that do not provide any quality guarantees on the results *ng*-approximate, and δ-ϵ-approximate for those who provide such guarantees, where ϵ is the approximation error and δ, the probability that this error will not be exceeded. When $\delta = 1$, a δ-ϵ-approximate method becomes ϵ-approximate, and when $\epsilon = 0$, an ϵ-approximate method becomes exact [42, 43].

Although a variety of techniques have been proposed for efficient exact k-NN search in-memory and on-disk [9, 16, 21–23, 42, 45, 68, 73, 74, 91, 92, 94, 95, 120–122, 124], their performance is unsatisfactory for some data science applications [6, 36, 38, 40, 87]. Therefore, we have witnessed an increased interest by the research community in solving the approximate k-NN problem [5, 42, 43, 119]. Such solutions can be classified into tree-based [9, 43, 45, 121], hash-based [43, 59, 112], and graph-based approaches [29, 48, 50, 70, 79, 111]. Tree-based indexing methods partition the dataset space into embedded hierarchical sub-spaces using a tree data structure, where similar data points belong in the same leaves. Hash-based indexing methods map the dataset vectors into different buckets of hash codes using multiple hash tables, and guarantee with high probability that similar data are hashed into the same buckets. Graph-based indexing methods structure the dataset into a proximity graph, where data points are represented as vertices and each vertex is connected to a set of similar vertices.

Each of the three families of similarity search methods has its advantages and disadvantages [43]. Tree-based techniques are efficient at index building with a new class of extensions [43] supporting all three flavors of search: exact, δ-ϵ-approximate and *ng*-approximate search. These extensions achieve the best performance on all scenarios except on *ng*-approximate search, where their efficiency is still unsatisfactory for many real applications. Hash-based

Proceedings of the VLDB Endowment, Vol. 16, No. 6 ISSN 2150-8097.
doi:10.14778/3583140.3583166

techniques support δ-ϵ-approximate search with additional theoretical guarantees on query efficiency, but are not scalable due to the high index construction time, high memory footprint and low empirical search performance. Theoretical guarantees on accuracy are provided on the distance approximation error, but this does not always translate into good recall empirically. Graph-based methods offer the best performance in practice for *ng*-approximate search, but building the graph structure on large datasets is extremely expensive both in time and space. Moreover, such techniques do not offer any guarantees on search quality and efficiency. Despite these disadvantages, graph-based approaches remain the methods of choice for many real applications such as recommendation systems [62, 104, 110, 126] that require a very low query latency (a few milliseconds per query on billion-scale collections), and can tolerate a lack of theoretical guarantees on the quality of the answers as long as a high recall (≥ 0.90) can be achieved empirically.

The data series community has developed scalable tree-based similarity search techniques that outperform state-of-the-art methods on large collections of both data series and generic high-d vectors, on all scenarios except for *ng*-approximate search, where graph-based approaches designed by the high-d vector community achieve the best performance. However building graph-based indexes on massive datasets is extremely expensive both in time and space [43, 108, 119]. The problem is compounded in scenarios where main memory is limited, or when the index needs to be rebuilt multiple times, either due to frequent updates [50], or to new analytical needs (since query accuracy is not determined only at query-time, but also by the quality of the graph index [43]), despite the use of parallelism during index building.

We propose ELPIS to address the scalability bottleneck of graph-based indexes, and therefore to support the large variety of data science applications that depend on similarity search [34, 36, 39]. Instead of constructing and querying a large graph, ELPIS adopts a divide-and-conquer approach. It splits the dataset into multiple clusters[1] based on the EAPCA summarization and builds a graph for each cluster in parallel. During search, ELPIS exploits the EAPCA lower-bounding distance to select an initial set of approximate answers to guide the search and determine the clusters to search and the order to search them. Both the index building and query answering algorithms exploit multithreading and SIMD. During query answering, multiple threads search different clusters in parallel. While we focus on single-node systems, the proposed ideas and learned lessons can be exploited in a distributed setting, where each dataset cluster is built/queried on a different node.

In this paper, we make the following contributions:

• We conduct a brief survey of the state-of-the-art tree- and graph-based similarity search methods. We describe their chronological development and highlight their key design principles.

• We propose ELPIS, a new index for in-memory *ng*-approximate similarity search that takes advantage of the best features of both tree-based and graph-based families to achieve a superior performance in terms of indexing and query answering. This is the first in-memory graph-based similarity search solution using EAPCA-based clustering, and employing intra-query parallelism.

[1]In the context of ELPIS, we use the term *cluster* to refer to a leaf node of the ELPIS index tree, following the observation that the leaves of an index can be interpreted as a clustering of the given dataset [58].

• We empirically demonstrate the efficacy, scalability and robustness of ELPIS by conducting thorough experiments on real large datasets from different domains: neuroscience, seismology, deep network embeddings and computer vision; this is the first time that graph-based similarity search techniques are evaluated on real datasets from neuroscience and seismology. ELPIS builds the index 3x-8x faster than competitors, using 40% less memory. It also achieves a high recall of 0.99, up to 2x faster than the state-of-the-art methods on most scenarios, and answers 1-NN queries up to one order of magnitude faster.

• We share key insights and lessons learned that can help the community better understand the impact of the following design choices on graph-based similarity search performance: (i) building/querying multiple graphs, each representing a cluster of a large dataset, as opposed to building/querying one large graph representing the whole dataset; (ii) the clustering approach used to split a large dataset into multiple chunks; (iii) the graph structure used in each cluster; and (iv) the strategies adopted for pruning clusters and vertices within each cluster.

Note that ELPIS fits within the scope of ng-approximate methods, i.e., methods that do not provide any theoretical guarantees on efficiency and accuracy, but have excellent empirical performance [119]. Nevertheless, these approaches are very popular [108], and have several important applications, including enhancing web search results (e.g., at Microsoft Bing [69]), and image similarity search (e.g., at Facebook/Meta [62]).

2 PRELIMINARIES & RELATED WORK

2.1 Definitions

The *ng*-approximate similarity search problem can be modeled as an approximate k-NN search problem in high-dimensional vector space. Data points are represented as d-dimensional vectors in $\mathbb{R}^d$, and the *dissimilarity* between the points is measured using the Euclidean distance $dist$. We consider a dataset collection $\mathbb{S}$ of n d-dimensional points and a query vector V_Q.

Definition 1. *The Euclidean distance between any two points* V_C, V_Q *in* $\mathbb{R}^d$ *is* $dist(V_C, V_Q) = \sqrt{\sum_{i=1}^{d} \left(V_C^i - V_Q^i\right)^2}$, *such that* i *is an integer and* $1 \leq i \leq d$.

Definition 2. *Given an integer* k, *a* ***k-NN query*** *retrieves the set of vectors* $\mathbb{A} = \{\{V_{C_1}, ..., V_{C_k}\} \subseteq \mathbb{S} | \forall\ V_C \in \mathbb{A}$ *and* $\forall\ V_{C'} \notin \mathbb{A},\ dist(V_Q, V_C) \leq dist(V_Q, V_{C'})\}$ *[43].*

Definition 3. *Given a query* V_Q, *an ng-****approximate*** *algorithm produces results,* V_C, *that are at a distance* $dist(V_Q, V_C) \leq (1 + \theta) d(V_Q, [k\text{-th NN of } V_Q])$, *for an arbitrary value* $\theta \in \mathbb{R}_{>0}$. *[43]*

We use the terms *ng*-approximate search, approximate k-NN search, approximate search and ANN search interchangeably.

2.2 Tree-based Similarity Search

Tree-based similarity search approaches organize the data in a tree structure to help prune candidates during search. Some indexing methods first reduce the dimensionality of the original data and then index the summarizations [8, 56, 103], while others perform dimensionality reduction and indexing in one step [17, 35, 73, 74,

121, 125, 128], or index the high-d data directly [26]. Some tree-based methods support only exact search, while others support also *ng*-approximate and δ-ϵ-approximate search [25, 33, 43].

Summarization Techniques. The *Discrete Haar Wavelet Transform* (DHWT) [20] transforms each high-d vector into a multi-level hierarchical structure using the Haar wavelet decomposition. The *Discrete Fourier Transform* (DFT) [2, 44, 99, 100] decomposes the original vector into frequency coefficients and uses a subset as a summary. The *Piecewise Aggregate Approximation* (PAA) [63] and *Adaptive Piecewise Constant Approximation* (APCA) [19] techniques divide the high-d vector respectively into equi-length and variable-length segments, with each segment being represented by the mean value of the corresponding points. The *Extended Adaptive Piecewise Approximation* (EAPCA) [121] extends APCA by representing each segment using both the standard deviation and the mean. The *Symbolic Aggregate Approximation* (SAX) [71] first transforms a vector using PAA then represents the mean values of the segments with a discrete set of symbols.

Similarity Search Methods. The state-of-the-art tree-based similarity search methods designed for data series [42] include the *DSTree* [121] approach, which uses the EAPCA summarization technique. The *iSAX* family [86] includes a large number of methods based on the iSAX summarization. The *iSAX 2.0* index [15] adds bulk-loading support to the original *iSAX* index [105] and improves its splitting policy; *iSAX2+* [17] further optimizes bulk-loading, while *ADS+* [128] proposes data-adaptive index building and *Dumpy* [122] data-adaptive node-splitting. ParIS+ [94] and Messi [92, 93] are parallel iSAX-based approaches and Hercules [35] is a parallel index using both the iSAX and EAPCA summarizations.

The high-d vector community contributed many tree-based techniques [102], including the *M-tree* [26], which is a multidimensional metric-space access method that leverages the relative distances between data points to divide them using hyper-spheres. *Flann* [82] is an in-memory ensemble technique for *ng*-approximate search including both randomized kd-trees [107] and hierarchical k-means trees [82]. *HD-index* [4] is an *ng*-approximate approach that partitions the original space into disjoint partitions of lower dimensionality, and indexed by an RBD tree (a modified B+tree).

2.3 Graph-based Similarity Search

The most popular *ng*-approximate similarity search techniques are based on graphs [30, 43, 48–50, 61, 70, 78, 79, 97, 111, 119]. These approaches typically build a proximity graph structure $G(\mathbb{V}, \mathbb{E})$, such that $\mathbb{V}$ is the set of vertices where each vertex represents a point $V \in \mathbb{S}$ and $\mathbb{E} \subset \mathbb{S} \times \mathbb{S}$ is the set of edges that connect similar vertices. Two vertices are connected with an edge if the points they represent are close in $\mathbb{R}^d$ space according to some distance measure, often the Euclidean distance. For a given query $V_Q \in \mathbb{R}^d$, search generally starts from a set of initial entry points or seeds, which can be random or satisfy some conditions, then visits neighboring vertices using a best-first search greedy approach, returning the set $\mathbb{A}$ of k approximate neighbors to V_Q when no better candidates can be found. State-of-the-art approximate graph-based similarity search approaches typically exploit a set of known base proximity graph structures and the same greedy search algorithm but differ in the way they construct the graph and select the entry points during search. Note that the larger the out-degree of vertices in the graph, the harder the routing task becomes, since many candidate vertices have to be pruned before finding the right path to reach the k nearest neighbors. On the other hand, lacking good edges may lead the search into local minima results. State-of-the-art approaches aim at finding the right trade-off between constructing short and long range edges, keeping the graph as sparse as possible, to minimize the number of distance calculations, and guard against convergence to local minima results.

2.3.1 Base Proximity Graphs. One of the most general proximity graphs in the literature is the *Delaunay Graph* (DG) or *Delaunay Triangulation* (DT) [67]. The DG is a planar dual graph for the *Voronoi Diagram* [47], where each vertex is the center of its own voronoi cell, and two vertices are connected if and only if their corresponding voronoi cells share at least one edge. The DG $G(\mathbb{V}, \mathbb{E})$ constructed over $\mathbb{S}$ must satisfy the DT: $\forall q, p, r \in \mathbb{V}, (q, p), (q, r), (r, p) \in \mathbb{E}$ if no other vertex from $\mathbb{V}$ is inside the circumcircle passing through q, p, s. Greedy search on DG is guaranteed to find the exact nearest neighbors [28]. Nevertheless, using DG for high-dimensional data is impractical as the graph becomes almost fully connected when the dimension d grows [28]. Therefore, it is more common in practice to use a proximity graph that is a subgraph of the DG [51, 80, 115]. Below, we describe the four most popular subgraphs of the DG used in the approximate nearest neighbor search literature. Assume a graph $G(\mathbb{V}, \mathbb{E})$ is constructed on a set of points $\mathbb{S}$.

Gabriel Graph. G is a *Gabriel Graph* (GG) [51] in Euclidean space if it guarantees that $\forall (u, v) \in \mathbb{V}^2, (u, v) \in \mathbb{E} \Leftrightarrow O(u, v) \cap \mathbb{V} \setminus \{u, v\} = \emptyset$; $O(u, v)$ is the circle passing through u and v with diameter uv.

Relative Neighbor Graph. G is a *Relative Neighbor Graph* (RNG) [114] if it is an undirected graph that is a subset of the GG graph [51], and is built using an edge selection policy based on the notion of 'relatively close' neighbors [66]. According to such a policy, two vertices $(u, v) \in \mathbb{V}^2$ are considered relatively close if and only if $dist(u, v) \leq max[dist(u, w), dist(w, v)]$ for all $w = 1, ..., n, w \neq u, v$.

Minimum Spanning Tree. G is a *Minimum Spanning Tree* (or Minimum Weight Spanning Tree) if and only if it is a connected acyclic graph with the minimal sum of weights of edges $\sum_{i=1}^{|\mathbb{E}|} w(e_i)$ [54]. In the context of a proximity graph, the weight of an edge corresponds to the distance between its two vertices.

Nearest Neighbor Graph. G is a *k-Nearest Neighbor Graph* (k-NNG) if it is a directed graph where each vertex is connected to its exact or approximate k-nearest neighbors [29, 72].

2.3.2 Base Search Algorithms. Graph-based similarity search methods typically perform nearest neighbor search using the beam search algorithm [101], a variation of best-first search [90]. Best-first search is a class of search algorithms that examine a graph by managing a set of promising nodes. The algorithm selects the most promising node with respect to a given problem, and adds its neighbors to the same set until no promising candidates can be found. In the case of k-NN search, the most promising node is the closest node to the query. Beam search [101] is a variant of best-first search where only L promising candidates are kept during the search process. The L parameter is called the beam width and is used to tune the efficiency/accuracy tradeoffs. When $L = 1$, a beam search is equivalent to a greedy search, and when $L \approx N$ where

N is the graph size, beam search is equivalent to best-first search. Note that although beam search does not guarantee to find the best candidates, it is used by most state-of-the-art graph-based similarity search methods. Some start the beam search with random (e.g., HNSW [79]) or predefined starting points during index building (e.g., VAMANA [111], which uses the centroid of the dataset), while others select seeds during query answering using pre-built indexes such as the KD-tree and LSH [48, 61, 83].

2.3.3 State-of-the-art Approaches. The research community has been actively working on improving the scalability of graph-based approximate k-NN search methods. Compared to earlier works [29, 48, 57, 61, 97], the current methods are one order of magnitude faster for both indexing and query answering [70, 72, 106, 119].

KGRAPH [29] and *NNdescent* [30] reduce the construction cost of an exact k-NNG and approximate k-NNG by refining a random initial graph. *Iterative Expanding Hashing* [61] (IEH) exploits *NNdescent* with hashing methods to generate initial candidates for graph construction instead of random selection, while *EFANNA* [48] initializes neighbors of each vertex using randomized truncated kd-trees [27] and refines the graph into an approximate k-NNG through *NNdescent*. The *Navigable Small World* (NSW) technique [78, 97] exploits the notion of a small world network [64, 65] to build the graph with long range edges, and adapts VoroNet's [7] peer-to-peer small world network to the graph-based similarity search problem. Nevertheless, the resulting graph suffers from a scalability bottleneck due to the high out-degree of vertices. NSW's authors addressed these issues in the *Hierarchical Navigable Small World* (HNSW) graph [79], which builds multi-resolution layers of NSW graphs, refined following an RNG process to reduce the out-degree of vertices. The number of nodes in each layer increases while descending the hierarchical structure of layers. The bottom most layer contains all the points in the dataset, and hierarchical layers are exploited during search to converge fast to the region that contains the query's nearest neighbors.

Inspired by HNSW's performance, subsequent methods have further exploited the RNG refinement process. The *Diversified Proximity Graph* (DPG) [70] builds an approximate k-NNG based on an RNG-refined KGRAPH [29]. The *Navigating Spreading-out Graph* (NSG) [50] builds an approximate k-NNG based on an initial EFANNA graph, refined through a monotonic RNG process [50], which also guarantees the existence of a monotonic path between any two nodes in the graph (A path is said to be monotonic if and only if on each hop, the distance of the query to the current node is strictly inferior to the distance of the query to the previous node in the path.) *VAMANA* [111] refines a random initial graph through two RNG passes. *SPTAG* [24] splits the dataset into multiple subsets using TPTrees [118] and merges the multiple subsets after building exact k-NNGs on each one of them. The *Hierarchical Clustering-Based Graph* [83] follows the same strategy with a hierarchical clustering splitting and builds an MST graph on each subset.

3 THE ELPIS APPROACH

We now describe ELPIS, a new baseline for graph-based similarity search that achieves better performance in both indexing and in-memory *ng*-approximate similarity search.

3.1 Index construction

During index construction, ELPIS first splits the dataset into multiple clusters using Hercules [35], where each leaf is considered a cluster, then builds graph structures in parallel on the tree's leaves. We opted for the state-of-the-art index tree Hercules [35] to cluster the dataset because it efficiently intertwines index building and data-adaptive dimensionality reduction, leading to well-clustered leaves, i.e., similar high-d vectors are stored in the same leaf. Moreover, the Hercules tree is based on the EAPCA summarization [121], used during search for pruning clusters. Any state-of-the-art graph-based similarity approach can be used within a leaf. We chose HNSW [79], because it led to the best overall performance (cf. Section 4).

Hercules builds its index tree using a double buffer to read the dataset from disk into memory in batches, while the vectors previously loaded into memory are being inserted in the tree. Multiple threads insert vectors in parallel into the tree. Each thread traverses the binary index tree to find the appropriate leaf, routing left or right depending on the split policy of the visited node. The resulting tree will have leaves that contain the original high-d vectors and internal nodes that record statistics about the vectors belonging to their subtrees. Each node has its own EAPCA segmentation and the vectors in each node are segmented using the same policy.

ELPIS then proceeds with building an HNSW graph for each leaf in parallel. The main coordinator thread initializes n_1 leafCoordinator threads. Each leafCoordinator selects a leaf for which the graph was not built yet, and creates n_2 leafWorker threads. Both the leafCoordinator and the leafWorkers contribute in building the graph within their corresponding leaf. Each one of them reads a vector from the dataset and inserts it into the graph. Once the graph has been built, the leafCoordinator materializes it into a separate file on disk, and selects a new leaf to process. Index construction terminates once the graphs for all index leaves have been built.

3.2 Query Answering

Answering a query V_Q with ELPIS proceeds in two major steps: (1) the Hercules tree is traversed to find k initial best-so-far (bsf) neighbors to V_Q, which are then used to select a list of l candidate leaves, i.e., clusters; and (2) a beam search is performed in parallel on the graph structures corresponding to the l candidate leaves.

During the first step, ELPIS traverses the Hercules tree, which is pre-loaded into memory, to find the leaf where V_Q would have been inserted if it belonged to the dataset. Then, it performs a beam search on the HNSW graph corresponding to this leaf, returning the k closest vectors to V_Q as first bsf answers. ELPIS resumes the traversal of the index tree using both the k^{th} bsf answer and the EAPCA segmentation to prune nodes. The algorithm then returns, as candidate clusters, a list of l leaves sorted in increasing order of their LB_{EAPCA} distance to V_Q.

In the second step, multiple threads process the candidate clusters in parallel, starting with the leaves that have the lowest LB_{EAPCA} distances to V_Q. (Exploiting parallelism within the same graph is still an open problem and is beyond the scope of this paper, so each leaf graph is processed by only one thread but the same thread can process multiple leaves, one at a time.) A thread processes a leaf by running a beam search on the corresponding HNSW graph and returning k bsf answers. Each thread maintains a local

priority queue which stores the k bsf answers corresponding to the processed leaf and a local k^{th}_{dist}, the Euclidean distance between V_Q and the k^{th} bsf answer in the current leaf. The threads use a readers-writer lock to synchronize access to a global k^{th}_{dist}, the Euclidean distance between V_Q and the k^{th} bsf answer across all leaves. The global k^{th}_{dist} bsf is updated every time a thread finds a better local k^{th}_{dist} bsf answer. Once a thread finishes search within a leaf, it uses its existing priority queue to warm up a new search on the next leaf with the lowest LB_{EAPCA} to V_Q. Search terminates either after the l candidate clusters are processed or once the LB_{EAPCA} distances between V_Q and the next leaf to be processed is larger than the k^{th}_{dist}. This is because (i) the LB_{EAPCA} distance guarantees the lower-bounding property, i.e., that the Euclidean distance between any two points in the original high-d space is guaranteed to be larger than or equal to their LB_{EAPCA} distance; and (ii) the list of candidate leaves is sorted in increasing order of LB_{EAPCA}. Once all threads terminate, ELPIS computes the final results: it aggregates the answers from all local priority queues, and selects the k answers with the smallest Euclidean distance to V_Q.

4 EXPERIMENTAL EVALUATION

Setup. All methods were compiled with GCC 8.2.0 under Ubuntu Linux 20.04 (Rocky Linux 8.5 on HPC) with the default compilation flags and the optimization level 3. Experiments were conducted on an Intel(R) Xeon(R) Platinum 8276 server (4 sockets, 28 cores per socket, and 1 thread per core, 35MB cache) with 1.5TB RAM.

Algorithms. We compare ELPIS against the best performing state-of-the-art ng-approximate methods [43, 119], HNSW [79], NSG [50], VAMANA [111], EFANNA [48], HCNNG [83], DPG [70] and KGRAPH [29], and for each method, we use the most efficient C/C++ implementation publicly available. We also compare ELPIS to Hercules [35] and QALSH [59], the state-of-the-art methods in exact and probabilistic similarity search respectively. All methods exploit multithreading and SIMD vectorization. To ensure a fair comparison across methods, we clear the caches between query workloads, and we disable certain optimizations used by some techniques, such as pre-warming the caches with query searches (VAMANA) and using an L2-normalized Euclidean distance (NSG, EFANNA and VAMANA). Unless stated otherwise, all algorithms are tuned to reach the right accuracy/efficiency tradeoffs.

Datasets. We use the following five real datasets covering a variety of domains from deep network embeddings, computer vision, neuroscience and seismology: (i) *Deep* [109] contains 1 billion vectors of 96 dimensions extracted from the last layers of a convolutional neural network; (ii) *Sift* [60, 113] consists of 1 billion SIFT vectors of size 128 representing image feature descriptions; (iii) *SALD* [116] contains neuroscience MRI data and includes 200 million data series of size 128; (iv) *Seismic* [46] contains 100 million data series of size 256 representing earthquake recordings at seismic stations worldwide; and (v) *Gist* [113] contains 1 million images of 960 dimensions. For Deep, SALD, Seismic and Sift, we select subsets of different sizes from each dataset and we refer to each subset with the name of the dataset followed by the subset size in GBs (e.g., Deep25GB). We refer to the 1-million and 1-billion datasets with the 1M and 1B prefixes, respectively. To evaluate ELPIS on datasets with different distributions, we also generated three random 25GB datasets RandPow0, RandPow5 and RandPow50, each with 256 dimensions, following the power law distribution [84] using three power law exponents: 0 (uniform [98]), 5 and 50 (very skewed).

Queries. Query workloads consist of 100 high-d query vectors ran one after the other, i.e., not in batch mode, which is a common scenario in a real setting where the queries are not known in advance [52, 53, 87]. Experiments that report numbers for 1 million queries extrapolate the results of the 100-query workloads. For the Deep, Gist and Sift datasets, queries were randomly sampled from the publicly available query workloads. For the SALD and Seismic datasets, real query workloads were not available, so we randomly sampled 100 queries from the corresponding dataset and excluded them during index building. In one experiment, we use queries of different difficulty for the Deep dataset, labeled with a percentage (1%-10%). These queries were generated by randomly selecting vectors from the dataset and perturbing them using different levels of Gaussian noise (μ = 0, σ^2 = 0.01-0.1) [129]. The percentage indicates the value of σ^2. We also generated a 100-query workload for each of the power law distribution datasets. Our experiments cover k-NN queries, where k $\in [1, 100]$ but for space considerations, we only report the 10-NN results per previous studies [108]. The overall trends remain the same (detailed results can be found in [1]).

Measures. We measure the accuracy of a k-NN query S_Q using Recall [77]: $Recall(S_Q) = \frac{\#\ true\ neighbors\ returned\ by\ Q}{k}$. The reported recall is the average across the query workload.

4.1 Results

We compare ELPIS to the state-of-the-art in-memory graph-based *ng*-approximate similarity search approaches, and demonstrate the following points: (i) careful design choices render building/querying multiple graphs (each representing one of the clusters of a large dataset) more efficient than building/querying a single large graph representing the entire dataset; (ii) using the EAPCA adaptive segmentation technique to cluster the dataset leads to better indexing and query answering performance compared to K-means; (iii) LB_{EAPCA} leads to better pruning of the clusters compared to the centroid-based distance, and warming the local priority queues with some bsf answers (at the beginning of the search within a cluster), leads to better performance than maintaining a global priority queue during search across all clusters; (iv) the number of clusters and type of graph structure used within each cluster have a significant impact on performance.

4.1.1 Index Construction Performance. We now study the indexing scalability of ELPIS in terms of both wall-clock time and memory footprint. We compare ELPIS to HNSW [79], NSG [50], VAMANA [111], EFANNA [48], HCNNG [83], DPG [70] and KGRAPH [29] on the Deep dataset ranging in size from 1 million to 1 billion vectors (up to 350GB). (Other datasets exhibit the same trends and are omitted for brevity). Since the indexing cost depends on the accuracy required during search, the numbers reported for each method are those required for the index to be able to reach an accuracy of 0.99 during search; for DPG, KGRAPH and HCNNG, we report the numbers for the best indexes we could build, which reached an accuracy below 0.8. All methods were included in the

1-million experiments. Due to scalability issues, we could not report results on the 25GB experiments for HCNNG (indexing time exceeded 24 hours) and KGRAPH/DPG (could not reach an acceptable accuracy, i.e., recall > 0.8). Due to the low performance on the 25GB experiments of VAMANA and EFANNA (indexing a 25GB dataset required over 300GB RAM and indexing a 100GB dataset needed more than the 1.4TB of available memory) and NSG (since it uses EFANNA as a base graph), we excluded them from experiments with larger datasets.

Indexing Time. Figure 1 shows that on the 25GB dataset, ELPIS can build its index 2x and 5x faster than HNSW and NSG, respectively, and over an order of magnitude faster than the other competitors. On the other dataset sizes, ELPIS is twice faster than its second best competitor, HNSW. Since NSG [50] is built on top of EFANNA [48], we include the time to build both indexing structures. Although VAMANA [111] builds the graph based on a random initial graph, it spends more than 7 hours to create the Deep25GB index. This is due to the fact that it has a two-pass refinement process, and needs a higher out-degree parameter compared to NSG.

Indexing Footprint. Figure 2 compares the main memory footprint of index building for the state-of-the-art methods (including the raw data). We measure the maximum amount of memory used by each technique during its index construction process[2]. ELPIS occupies at least 40 per cent less memory resources than competitors when constructing the index. As the dataset size grows, HNSW suffers from a steeper increase in footprint compared to ELPIS. We also measure the final size of the index once the index has been built (including the raw data). Figure 3 shows that the final index size is almost the same for all methods on most dataset sizes, but as the dataset size grows, ELPIS has a relatively lower disk footprint. This experiment shows that some methods require a memory footprint that is far larger than the final size of the index. For example, HCNNG keeps in-memory multiple random samples of the data while building the index.

4.1.2 Query Answering Performance. In the following experiments, we compare the query answering scalability of ELPIS against the competitors on 5 real datasets ranging in size from 1 million to 1 billion vectors.

Footprint & Beam Width. We evaluate the memory footprint and the beam width required to reach an accuracy of 0.99 (except for DPG, KGRAPH and HCNNG, which could not reach accuracies above 0.8) on multiple subsets of the Deep dataset ranging from 1 million to 1 billion. Figure 4 shows the main memory footprint is dominated by the size of the index, as we reported in Figure 3. In Figure 5, we evaluate the average beam width required by the different methods to reach 0.99 accuracy. Recall that the beam width is the size of the priority queue used during the beam search. We can observe that ELPIS requires up to a 40% smaller beam width across all dataset sizes, which translates into better query efficiency.

Efficiency/Accuracy with different real datasets. Overall, ELPIS, HNSW and NSG display the best performance, with NSG winning on the small datasets and ELPIS outperforming all methods on the large datasets. Figure 6 shows that ELPIS and NSG perform the best on Sift1M and Seismic1M. NSG keeps its superiority on the other datasets, whereas the performance of ELPIS degrades on Deep1M, SALD1M and Gist1M. ELPIS is more competitive on the 25GB datasets (Fig. 7), winning on Deep25GB and Sift25GB and performing equally to NSG on Seismic25GB and SALD25GB. Note that no method is able to reach an accuracy beyond 0.8 on the Seismic dataset, so we report the results for the low recall values. Due to the large indexing footprint of NSG, we could not continue the evaluation with NSG on the larger datasets because building the EFANNA graph (on which NSG is based) requires more than the available memory (1.4TB). Figure 8 demonstrates the superiority of ELPIS on the large datasets of 1 billion vectors, being up to an order of magnitude faster for 0.95 accuracy. Similar trends are observed over subsets ranging from 100GB to 250GB (detailed results in [1]).

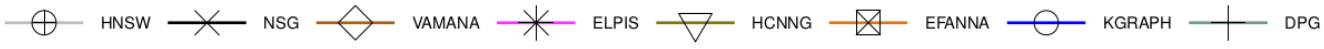
Efficiency/Accuracy with different query workloads. We also run experiments where we vary the difficulty of queries and observe the performance of ELPIS and its strongest contenders (NSG and HNSW) on the Deep25GB dataset. Figure 9 clearly shows that ELPIS maintains its superiority on all query workloads.

Efficiency/Accuracy with different data distributions. ELPIS achieves the best performance on RandPow0 (Fig. 7e) and RandPow5 [1]. On the hardest dataset, RandPow0, we observe that the competitors cannot reach a recall beyond 0.7. We believe that their search algorithm gets stuck in a local minimum, whereas the multi-cluster search of ELPIS is able to reach for the answers in different regions of the data space. As the power law exponent increases, the data becomes more concentrated in a dense region and the performance of all methods improves. In the extreme case of RandPow50 (Fig. 7f), all methods have comparable efficiency.

Comparison to methods with answer guarantees. To better illustrate the gap in performance between similarity search approaches that support guarantees on query accuracy and those that do not, we compare ELPIS (ng-approximate search) to the following state-of-the-art methods: Hercules [35] (exact search), and QALSH [59] (δ-ϵ-approximate search, which includes the LSH family). Figure 10 shows that QALSH doesn't reach a recall higher than 0.75, whereas ELPIS can reach a 0.99 recall for all values of k. For the same recall, ELPIS is over five orders of magnitude faster than QALSH. As an exact technique, Hercules always reaches a recall of 1, but it is two orders of magnitude slower than ELPIS.

4.1.3 Choosing the clustering technique. We evaluate three different clustering techniques: EAPCA-based clustering, exact K-means and approximate K-means. Exact K-means is the K-means algorithm in [10] that continues iterating until all centroids stabilize. Approximate K-means refers to a modified version of the exact K-means which stops execution after a certain number of iterations (user-defined). We evaluate different numbers of iterations and choose the one that leads to the same accuracy as the exact K-means. For a fair comparison, we use the number of clusters produced by the EAPCA clustering (which is not known in advanced, but determined adaptively) as the number of clusters in both K-means algorithms. Note that the points allocated to each cluster may be different across clustering techniques. The set of clusters produced by each technique is then fed to the same algorithm, which builds in parallel an HNSW graph in each cluster. In the case of ELPIS, the clusters correspond to the EAPCA-based Hercules tree leaves, and the entire tree represents the index. For the other techniques, the index consists only of the set of clusters.

[2]We read the Virtual Memory Peak from the proc pseudo filesystem.



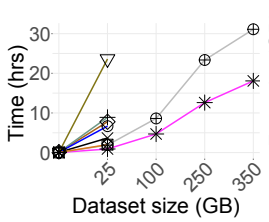


Figure 1: Indexing Time

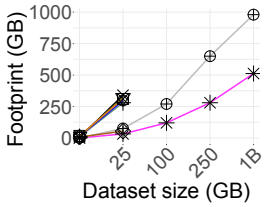


Figure 2: Indexing Memory Footprint

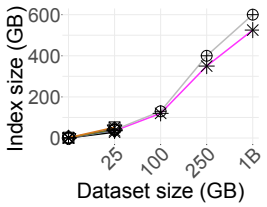


Figure 3: Indexing Disk Footprint

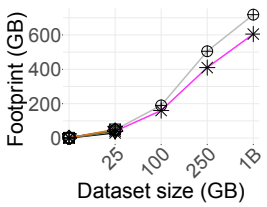


Figure 4: Query Memory Footprint

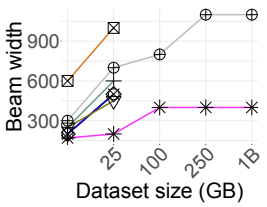
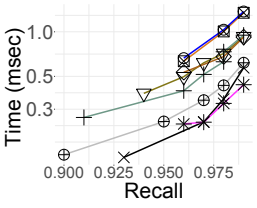
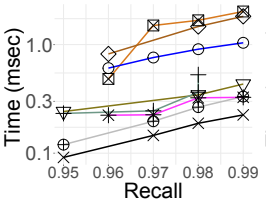


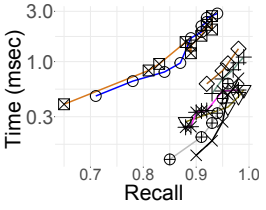
Figure 5: Query Beam Width



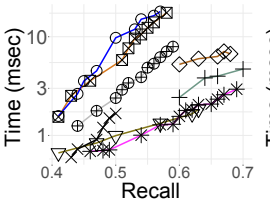
(a) Sift1M



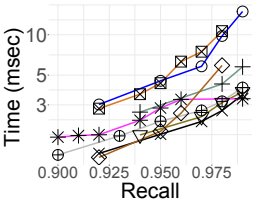
(b) Deep1M



(c) SALD1M

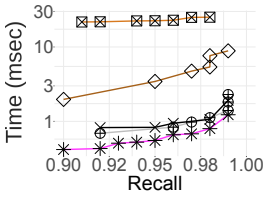


(d) Seismic1M

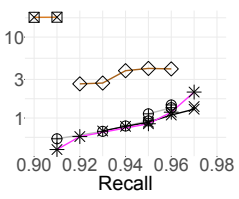


(e) Gist1M

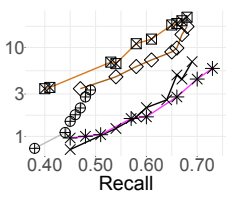
Figure 6: Query performance on 1M vectors



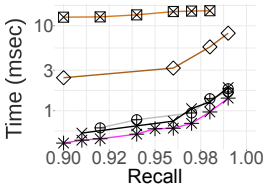
(a) Deep



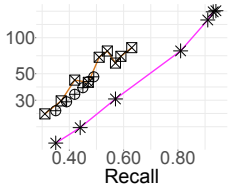
(b) SALD



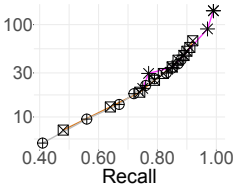
(c) Seismic



(d) Sift

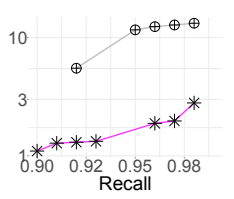


(e) RandPow0

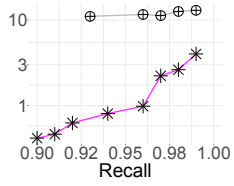


(f) RandPow50

Figure 7: 25GB datasets

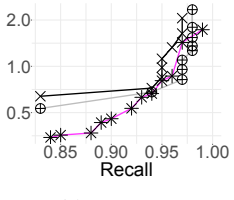


(a) Deep

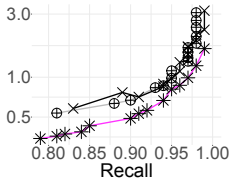


(b) Sift

Figure 8: 1B datasets



(a) 1% noise

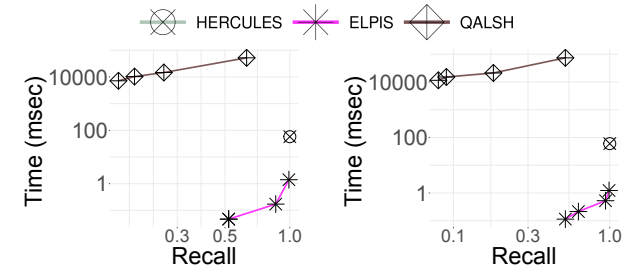


(b) 10% noise

Figure 9: Varying workloads

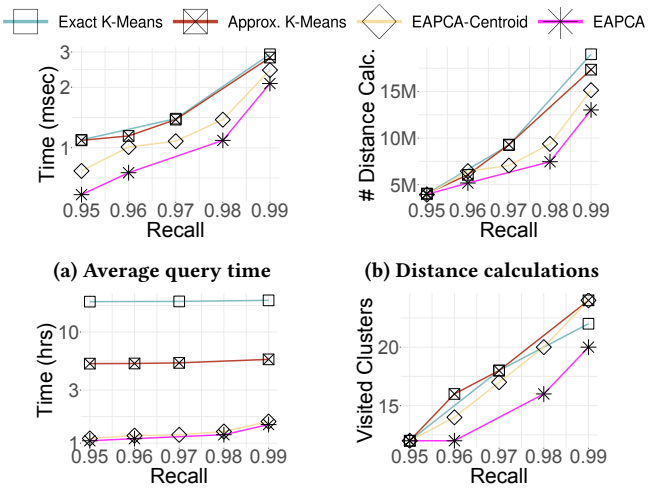
The query answering algorithm within each cluster is the same across techniques, but the pruning of clusters depends on the clustering approach: in the case of EAPCA-based clustering, we search clusters in ascending order of LB_{EAPCA} (lower bounding distance between the query and the EAPCA representations of the clusters), and we prune clusters using both LB_{EAPCA} and k^{th}_{dist}, the distance between the query and the current k^{th} bsf answer, obtained by traversing the Hercules tree. Both exact and approximate K-means prune clusters based on the distance between the query and the cluster centroids. We also evaluate the pruning of the clusters using EAPCA-based clustering with centroid-based pruning (EAPCA-Centroid). EAPCA-Centroid uses the same clusters obtained by the EAPCA clustering, but prunes clusters using the distances to the cluster centroids (instead of LB_{EAPCA} and k^{th}_{dist}).

Figure 11 summarizes the results of this experiment on the Deep25GB dataset. To ensure a fair comparison between EAPCA and K-means, we use the same number of clusters. Since in EAPCA, this number is found adaptively and cannot be enforced, we use the number of leaves that result from building the EAPCA tree as the number of clusters for the K-means algorithms (in the case of Deep25GB, this number is 26). The exact K-means algorithm requires 551 iterations to converge, while approximate K-means requires 40 iterations to reach the accuracy levels of exact K-means. We observe that on average for a given query, ELPIS delivers the



(a) Sift (b) Deep

Figure 10: Query performance vs. methods with guarantees



(a) Average query time (b) Distance calculations

(c) Idx+1M queries (d) Visited clusters

Figure 11: K-means vs EAPCA (Deep25GB)

same accuracy as previous methods up to 1.5x faster (Figs. 11a-11b). At the same time, it builds the index and answers 1 million queries 5x-15x faster than competitors (Fig. 11c). We report that ELPIS builds its index 4x and 60x faster than the solutions using approximate and exact K-means, respectively, even though they are building the same number of clusters. This is because both exact and approximate K-means spend a significant amount of time to find the centroids and populate them with vectors, whereas ELPIS uses the efficient index building algorithm of Hercules. We can see in Figure 11d that the EAPCA clustering, used by ELPIS, achieves the same recall as EAPCA-Centroid by visiting a smaller number of clusters. Since they both use the exact same clusters and query algorithm within each cluster, this means that EAPCA prunes the clusters better than EAPCA-Centroid. For instance, EAPCA-Centroid needs to visit 20 clusters whereas EAPCA only needs to visit 16. Besides, by visiting the same number of clusters, EAPCA can reach a higher recall than EAPCA-Centroid, because it more intelligently selects the most promising clusters to search.

4.1.4 Choosing the graph structure within each cluster. In this experiment, we evaluate different state-of-the-art graph structures to choose the best performing one. We divide a dataset into several clusters using the EAPCA dynamic segmentation, then evaluate the indexing and query performance of different types of graphs

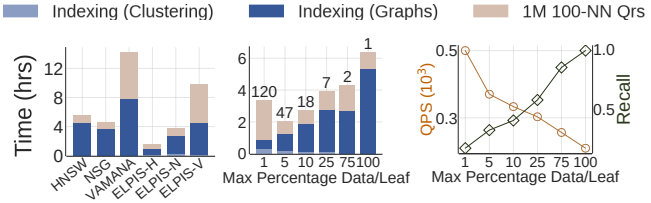


Figure 12: Varying graph structures (Deep25GB)

Figure 13: Varying cluster sizes (Deep25GB)

Figure 14: Querying one variable-size cluster (Deep25GB)

within the clusters (ELPIS-H using HNSW; ELPIS-N and ELPIS-V using NSG and VAMANA respectively), and compare against the performance of the original graphs. Figure 12 demonstrates that using HNSW within the clusters leads to best performance than using NSG or VAMANA, in both indexing and query answering.

4.1.5 Choosing the number of clusters. In this experiment, we show that it is not sufficient to divide the search space to improve the performance of graph-based methods; careful tuning should be applied, in order to find the optimal number of clusters. Hercules determines the number of clusters adaptively. It takes as input the maximum amount of data that can fit in one leaf (*max_leaf_size*), then creates enough leaves to fit similar vectors together in the same leaf. Figure 13 reports on the x-axis the value of *max_leaf_size* as a percentage of the size of the dataset, and at the top of each bar the corresponding number of clusters. When the percentage of data is 100, this corresponds to the original HNSW built on the full dataset (i.e., one cluster). We observe that splitting the dataset into many small clusters leads to better indexing performance, but slow query answering, whereas using a small number of large clusters leads to slower indexing and querying, where the clustering cost becomes marginal compared to the cost of building a graph within each cluster. For our experiments, we choose *max_leaf_size* to be between 5% and 10% of the dataset size since it leads to the best tradeoff. Additionally, we ran an experiment where we vary the size of the clusters and perform the ELPIS search algorithm only on one cluster, keeping all the other parameters fixed. Figure 14 indicates that as the size of a cluster increases, the accuracy improves, but throughput (expressed in Queries Per Second (QPS)) decreases.

5 CONCLUSIONS AND FUTURE WORK

We proposed ELPIS, a new solution for in-memory *ng*-approximate similarity search over massive collections of high-dimensional vectors. We demonstrated the efficacy of ELPIS on large datasets from various domains, where ELPIS considerably outperforms its competitors. Finally, we shared key insights and lessons learned that can help the community better understand the impact of different design choices on graph-based similarity search performance, and to make further progress in this area. A theoretical analysis of the performance and reliability properties of graph-based similarity search methods is an interesting and challenging open problem.

ACKNOWLEDGMENTS

This work used the UM6P African Supercomputing Center (ASCC).

REFERENCES

[1] Elpis Archive. http://www.mi.parisdescartes.fr/~themisp/elpis/, 2022.

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

[3] U. Alon, M. Zilberstein, O. Levy, and E. Yahav. Code2vec: Learning distributed representations of code. 3(POPL), 2019.

[4] 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.

[5] M. Aumüller, E. Bernhardsson, and A. Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. In *International Conference on Similarity Search and Applications*, pages 34–49. Springer, 2017.

[6] M. Aumüller, E. Bernhardsson, and A. J. Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Inf. Syst.*, 87, 2020.

[7] O. Beaumont, A.-M. Kermarrec, L. Marchal, and É. Rivière. Voronet: A scalable object network based on voronoi tessellations. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–10. IEEE, 2007.

[8] 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.

[9] J. S. Beis and D. G. Lowe. Shape indexing using approximate nearest-neighbour search in high-dimensional spaces. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition*, pages 1000–1006. IEEE, 1997.

[10] H.-H. Bock. Clustering methods: a history of k-means algorithms. *Selected contributions in data analysis and classification*, pages 161–172, 2007.

[11] P. Boniol, M. Linardi, F. Roncallo, and T. Palpanas. Automated anomaly detection in large sequences. In *ICDE*, 2020.

[12] P. Boniol and T. Palpanas. Series2graph: Graph-based subsequence anomaly detection for time series. *PVLDB*, 13(11), 2020.

[13] P. Boniol, J. Paparrizos, T. Palpanas, and M. J. Franklin. SAND: streaming subsequence anomaly detection. *Proc. VLDB Endow.*, 14(10):1717–1729, 2021.

[14] S. Byers and A. E. Raftery. Nearest-neighbor clutter removal for estimating features in spatial point processes. *JASA*, 93(442), 1998.

[15] 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.

[16] A. Camerra, J. Shieh, T. Palpanas, T. Rakthanmanon, and E. Keogh. Beyond One Billion Time Series: Indexing and Mining Very Large Time Series Collections With *i*SAX2+. *Knowledge and information systems*, 39(1):123–151, 2014.

[17] 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.

[18] R. Castro Fernandez, E. Mansour, A. A. Qahtan, A. Elmagarmid, I. Ilyas, S. Madden, M. Ouzzani, M. Stonebraker, and N. Tang. Seeping semantics: Linking datasets using word embeddings for data discovery. In *ICDE*, 2018.

[19] 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.

[20] K.-P. Chan and A. W.-C. Fu. Efficient time series matching by wavelets. In *Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337)*, pages 126–133, Mar 1999.

[21] M. Chatzakis, P. Fatourou, E. Kosmas, T. Palpanas, and B. Peng. Odyssey: A Journey in the Land of Distributed Data Series Similarity Search. *Proc. VLDB Endow.*, 2023.

[22] G. Chatzigeorgakidis, D. Skoutas, K. Patroumpas, T. Palpanas, S. Athanasiou, and S. Skiadopoulos. Local similarity search on geolocated time series using hybrid indexing. In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 179–188, 2019.

[23] G. Chatzigeorgakidis, D. Skoutas, K. Patroumpas, T. Palpanas, S. Athanasiou, and S. Skiadopoulos. Efficient range and knn twin subsequence search in time series. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 2022.

[24] Q. Chen, H. Wang, M. Li, G. Ren, S. Li, J. Zhu, J. Li, C. Liu, L. Zhang, and J. Wang. *SPTAG: A library for fast approximate nearest neighbor search*, 2018.

[25] P. Ciaccia and M. Patella. PAC nearest neighbor queries: Approximate and controlled search in high-dimensional and metric spaces. In D. B. Lomet and G. Weikum, editors, *Proceedings of the 16th International Conference on Data Engineering, San Diego, California, USA, February 28 - March 3, 2000*, pages 244–255. IEEE Computer Society, 2000.

[26] 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. Lochovsky, 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.

[27] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*, pages 537–546, 2008.

[28] D. P. Dobkin, S. J. Friedman, and K. J. Supowit. Delaunay graphs are almost as good as complete graphs. *Discrete & Computational Geometry*, 5(4):399–407, 1990.

[29] W. Dong. Kgraph, an open source library for k-nn graph construction and nearest neighbor search. www.kgraph.org, 2022.

[30] W. Dong, C. Moses, and K. Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*, pages 577–586, 2011.

[31] S. Dua and X. Du. *Data Mining and Machine Learning in Cybersecurity*. Auerbach Publications, USA, 1st edition, 2011.

[32] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang. Distributed representations of tuples for entity resolution. *VLDBJ*, 11(11), 2018.

[33] K. Echihabi. Truly Scalable Data Series Similarity Search. In *Proceedings of the VLDB 2019 PhD Workshop*, 2019.

[34] K. Echihabi. High-Dimensional Similarity Search: From Time Series to Deep Network Embeddings. In *SIGMOD*, 2020.

[35] K. Echihabi, P. Fatourou, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. Hercules Against Data Series Similarity Search. *PVLDB*, 15(10), 2022.

[36] K. Echihabi, T. Palpanas, and K. Zoumpatianos. New Trends in High-D Vector Similarity Search: AI-driven, Progressive, and Distributed. *Proc. VLDB Endow.*, 14(12):3198–3201, 2021.

[37] K. Echihabi, T. Tsandilas, A. Gogolou, A. Bezerianos, and T. Palpanas. ProS: Data Series Progressive k-NN Similarity Search and Classification with Probabilistic Quality Guarantees. *VLDBJ*, 2023.

[38] K. Echihabi, K. Zoumpatianos, and T. Palpanas. Big Sequence Management: on Scalability (tutorial). In *IEEE BigData*, 2020.

[39] K. Echihabi, K. Zoumpatianos, and T. Palpanas. Scalable machine learning on high-dimensional vectors: From data series to deep network embeddings. In *WIMS 2020: The 10th International Conference on Web Intelligence, Mining and Semantics*, pages 1–6. ACM, 2020.

[40] K. Echihabi, K. Zoumpatianos, and T. Palpanas. Big Sequence Management: Scaling Up and Out (tutorial). In *EDBT*, 2021.

[41] K. Echihabi, K. Zoumpatianos, and T. Palpanas. High-dimensional similarity search for scalable data science. ICDE, 2021.

[42] 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), 2018.

[43] K. Echihabi, K. Zoumpatianos, T. Palpanas, and H. Benbrahim. Return of the Lernaean Hydra: Experimental Evaluation of Data Series Approximate Similarity Search. *PVLDB*, 13(3), 2019.

[44] 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.

[45] 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*, pages 202–209, 2000.

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

[47] S. Fortune. Voronoi diagrams and delaunay triangulations. *Computing in Euclidean geometry*, pages 225–265, 1995.

[48] C. Fu and D. Cai. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228*, 2016.

[49] C. Fu, C. Wang, and D. Cai. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[50] C. Fu, C. Xiang, C. Wang, and D. Cai. Fast approximate nearest neighbor search with the navigating spreading-out graph. *Proc. VLDB Endow.*, 12(5):461–474, 2019.

[51] K. R. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic zoology*, 18(3):259–278, 1969.

[52] A. Gogolou, T. Tsandilas, K. Echihabi, A. Bezerianos, and T. Palpanas. Data Series Progressive Similarity Search with Probabilistic Quality Guarantees. In *SIGMOD*, 2020.

[53] 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*, 2019.

[54] R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[55] M. Günther, M. Thiele, and W. Lehner. Retro: Relation retrofitting for in-database machine learning on textual data. *arXiv preprint arXiv:1911.12674*, 2019.

[56] 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.

[57] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[58] J. M. Hellerstein, Y. Ioannidis, H. Jagadish, T. Johnson, R. Ng, V. Poosala, K. A. Ross, and K. C. Sevcik. The new jersey data reduction report. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 20(4), 1997.

[59] 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.

[60] 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.

[61] Z. Jin, D. Zhang, Y. Hu, S. Lin, D. Cai, and X. He. Fast and accurate hashing via iterative nearest neighbors expansion. *IEEE transactions on cybernetics*, 44(11):2167–2177, 2014.

[62] J. Johnson, M. Douze, and H. Jégou. Billion-scale similarity search with gpus. *IEEE Trans. Big Data*, 7(3):535–547, 2021.

[63] 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.

[64] J. Kleinberg et al. Small-world phenomena and the dynamics of information. *Advances in neural information processing systems*, 1:431–438, 2002.

[65] J. M. Kleinberg. Navigation in a small world. *Nature*, 406(6798):845–845, 2000.

[66] P. M. Lankford. Regionalization: theory and alternative algorithms. *Geographical Analysis*, 1(2):196–212, 1969.

[67] D.-T. Lee and B. J. Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980.

[68] O. Levchenko, B. Kolev, D. E. Yagoubi, R. Akbarinia, F. Masseglia, T. Palpanas, D. E. Shasha, and P. Valduriez. Bestneighbor: efficient evaluation of knn queries on large time series databases. *Knowl. Inf. Syst.*, 63(2):349–378, 2021.

[69] C. Li, M. Zhang, D. G. Andersen, and Y. He. Improving approximate nearest neighbor search through learned adaptive early termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, 2020.

[70] 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*, 32(8):1475–1488, 2019.

[71] 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, San Diego, California, USA*, 2003.

[72] P.-C. Lin and W.-L. Zhao. Graph based nearest neighbor search: Promises and failures. *arXiv preprint arXiv:1904.02077*, 2019.

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

[74] M. Linardi and T. Palpanas. Scalable data series subsequence matching with ULISSE. *VLDB J.*, 29(6):1449–1474, 2020.

[75] M. Linardi, Y. Zhu, T. Palpanas, and E. J. Keogh. Matrix profile goes MAD: variable-length motif and discord discovery in data series. *Data Min. Knowl. Discov.*, 34(4):1022–1071, 2020.

[76] G. Linden, B. Smith, and J. York. Amazon.com recommendations: item-to-item collaborative filtering. *IEEE Internet Computing*, 7(1), 2003.

[77] J. Makhoul, F. Kubala, R. E. Schwartz, and R. M. Weischedel. Performance measures for information extraction. 2007.

[78] 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.

[79] Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 42(4):824–836, 2020.

[80] D. W. Matula and R. R. Sokal. Properties of gabriel graphs relevant to geographic variation research and the clustering of points in the plane. *Geographical analysis*, 12(3):205–222, 1980.

[81] R. J. Miller. Open data integration. *PVLDB*, 11(12), 2018.

[82] 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.

[83] J. V. Munoz, M. A. Gonçalves, Z. Dias, and R. d. S. Torres. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recognition*, 96:106970, 2019.

[84] M. E. Newman. Power laws, pareto distributions and zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[85] T. D. Nguyen, A. T. Nguyen, and T. N. Nguyen. Mapping api elements for code migration with vector representations. In *ICSE*, 2016.

[86] T. Palpanas. Evolution of a Data Series Index: the iSAX Family of Data Series Indexes. *Communications in Computer and Information Science (CCIS)*, "accepted for publication, 2020".

[87] T. Palpanas and V. Beckmann. Report on the First and Second Interdisciplinary Time Series Analysis Workshop (ITISA). *ACM SIGMOD Record*, 48(3), 2019.

[88] J. Paparrizos, P. Boniol, T. Palpanas, R. Tsay, A. J. Elmore, and M. J. Franklin. Volume under the surface: A new accuracy evaluation measure for time-series anomaly detection. *Proc. VLDB Endow.*, 15(11):2774–2787, 2022.

[89] J. Paparrizos, Y. Kang, P. Boniol, R. S. Tsay, T. Palpanas, and M. J. Franklin. TSB-UAD: an end-to-end benchmark suite for univariate time-series anomaly detection. *Proc. VLDB Endow.*, 15(8):1697–1711, 2022.

[90] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[91] B. Peng, P. Fatourou, and T. Palpanas. Paris: The next destination for fast data series indexing and query answering. In *IEEE International Conference on Big Data (IEEE BigData)*, 2018.

[92] B. Peng, P. Fatourou, and T. Palpanas. Messi: In-memory data series indexing. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 337–348. IEEE, 2020.

[93] B. Peng, P. Fatourou, and T. Palpanas. Fast data series indexing for in-memory data. *VLDBJ*, 30(6), 2021.

[94] B. Peng, P. Fatourou, and T. Palpanas. Paris+: Data series indexing on multi-core architectures. *TKDE 33(5)*, 2021.

[95] B. Peng, P. Fatourou, and T. Palpanas. SING: sequence indexing using gpus. In *37th IEEE International Conference on Data Engineering, ICDE*, pages 1883–1888, 2021.

[96] F. Petitjean, G. Forestier, G. I. Webb, A. E. Nicholson, Y. Chen, and E. J. Keogh. Dynamic time warping averaging of time series allows faster and more accurate classification. In *ICDM*, 2014.

[97] A. Ponomarenko, Y. Malkov, A. Logvinov, and V. Krylov. Approximate nearest neighbor search small world approach. In *International Conference on Information and Communication Technologies & Applications*, volume 17, 2011.

[98] Python API. openmc.stats.PowerLaw. https://docs.openmc.org/en/stable/pythonapi/generated/openmc.stats.PowerLaw.html, 2022.

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

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

[101] D. R. Reddy. Speech understanding systems: A summary of results of the five-year research effort. *Department of Computer Science Technical Report. Carnegie Mellon University*, 1977.

[102] 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.

[103] 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, 2012.

[104] L. Shi. Trading-off among accuracy, similarity, diversity, and long-tail: a graph-based recommendation approach. In *Proceedings of the 7th ACM Conference on Recommender Systems*, pages 57–64, 2013.

[105] 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, 2008.

[106] L. C. Shimomura, R. S. Oyamada, M. R. Vieira, and D. S. Kaster. A survey on graph-based methods for similarity searches in metric spaces. *Information Systems*, 95:101507, 2021.

[107] 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.

[108] H. V. Simhadri, G. Williams, M. Aumüller, M. Douze, A. Babenko, D. Baranchuk, Q. Chen, L. Hosseini, R. Krishnaswamy, G. Srinivasa, S. J. Subramanya, and J. Wang. Results of the neurips'21 challenge on billion-scale approximate nearest neighbor search. *CoRR*, abs/2205.03763, 2022.

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

[110] L. Song, P. Pan, K. Zhao, H. Yang, Y. Chen, Y. Zhang, Y. Xu, and R. Jin. Large-scale training system for 100-million classification at alibaba. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2909–2930, 2020.

[111] S. J. Subramanya, R. Kadekodi, R. Krishaswamy, and H. V. Simhadri. Diskann: Fast accurate billion-point nearest neighbor search on a single node. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 13766–13776, 2019.

[112] 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. *Proceedings of the VLDB Endowment*, 2014.

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

[114] G. T. Toussaint. The relative neighbourhood graph of a finite planar set. *Pattern recognition*, 12(4):261–268, 1980.

[115] G. T. Toussaint. Proximity graphs for nearest neighbor decision rules: recent progress. *Interface*, 34, 2002.

[116] 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.

[117] J. Wang, P. Huang, H. Zhao, Z. Zhang, B. Zhao, and D. L. Lee. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *KDD*, 2018.

[118] J. Wang, N. Wang, Y. Jia, J. Li, G. Zeng, H. Zha, and X.-S. Hua. Trinary-projection trees for approximate nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 36(2):388–403, 2013.

[119] M. Wang, X. Xu, Q. Yue, and Y. Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *Proc. VLDB Endow.*, 14(11):1964–1978, jul 2021.

[120] Q. Wang and T. Palpanas. Deep learning embeddings for data series similarity search. In *KDD '21: The 27th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 1708–1716, 2021.

[121] Y. Wang, P. Wang, J. Pei, W. Wang, and S. Huang. A data-adaptive and dynamic segmentation index for whole matching on time series. *Proceedings of the VLDB Endowment*, 6(10):793–804, 2013.

[122] Z. Wang, Q. Wang, P. Wang, T. Palpanas, and W. Wang. Dumpy: A compact and adaptive index for large data series collections. In *ACM SIGMOD*, 2023.

[123] K. Williams, L. Li, M. Khabsa, J. Wu, P. C. Shih, and C. L. Giles. A web service for scholarly big data information extraction. In *ICWS*, 2014.

[124] D. E. Yagoubi, R. Akbarinia, F. Masseglia, and T. Palpanas. Dpisax: Massively distributed partitioned isax. In *ICDM*, 2017.

[125] D.-E. Yagoubi, R. Akbarinia, F. Masseglia, and T. Palpanas. Massively distributed time series indexing and querying. *TKDE 31(1)*, 2020.

[126] H. Zhao, Q. Yao, J. Li, Y. Song, and D. L. Lee. Meta-graph based recommendation fusion over heterogeneous information networks. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 635–644, 2017.

[127] E. Zhu, F. Nargesian, K. Q. Pu, and R. J. Miller. Lsh ensemble: internet-scale domain search. *Proceedings of the VLDB Endowment*, 9(12):1185–1196, 2016.

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

[129] 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.