Vector Search on Billion-Scale Data Collections

Ilias Azizi
Supervised by: Prof. Karima Echihabi and Prof. Themis Palpanas
UM6P, Université Paris Cité
ilias.azizi@um6p.com

ABSTRACT

Vector data is prevalent in various fields and is expected to grow further with the increasing popularity of embeddings. This data often grows to billions of vectors with thousands of dimensions, making analysis a complex task. A key component of this analysis is vector search, which aims to find similar vectors to a given input. Over the last decade, graph-based techniques have become the preferred method for vector search in many analytical tasks that do not require theoretical guarantees, thanks to their excellent query efficiency. However, these methods face scalability issues due to high memory consumption and long indexing times. In contrast, tree-based techniques provide theoretical guarantees on search and better indexing scalability, but their search efficiency falls behind graph-based approaches. We conduct an extensive evaluation of in-memory graph-based vector search approaches, highlighting the key design choices. We propose ELPIS, a new in-memory *ng*-approximate vector search technique, which combines the advantages of graph and tree-based index structures while mitigating the limitations inherent in each class. ELPIS outperforms the state-of-the-art in latency-optimized settings, reaching a recall of 0.99 by up to 2x faster for dataset sizes containing up to 1 billion vectors. We also pinpoint our future research directions, including extending the experimental study to out-of-core datasets and enhancing ELPIS to outperform the state-of-the-art in throughput-optimized settings.

VLDB Workshop Reference Format:
Ilias Azizi. Vector Search on Billion-Scale Data Collections. VLDB 2024 Workshop: VLDB Ph.D. Workshop.

VLDB Workshop Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/scalablesimilaritysearch/ELPIS.

1 INTRODUCTION

Vector data is widely used in scientific and business fields, and its use is expected to increase with the rise of embeddings [10]. The large size and high-dimensionality of this data, often reaching terabytes and thousands of dimensions, make its analysis challenging.

A vector search algorithm identifies, in a dataset $\mathbb{S}$ consisting of n d-dimensional vectors, elements similar to a given input vector V_Q. In this work, we abstract vector search as a k-nearest neighbor problem. The brute-force approach compares V_Q to each element in $\mathbb{S}$, with a time complexity of $O(nd)$, which is impractical for large datasets with high dimensionality. State-of-the-art methods typically reduce this time complexity by (i) limiting the number of processed vectors n using efficient indexing data structures and search algorithms that eliminate unnecessary comparisons to V_Q, and/or by (ii) decreasing the dimensionality d through concise and precise summarization techniques. Additionally, vector search algorithms can return exact or approximate answers. The approximate answers can either be with (δ-ϵ-approximate) or without (ng-approximate) guarantees on query accuracy [11, 12].

The efficiency of exact vector search has markedly improved over the last decade [11]. However, exact methods still fail to meet the query latency requirements of many analytical tasks. Consequently, extensive research has focused on approximate vector search, which sacrifices accuracy for efficiency [12]. These approaches utilize scans, trees, graphs, inverted indexes, hashing, or combinations of these data structures. In the past decade, graph-based techniques have become the preferred method for many real-world applications, as they can relax theoretical guarantees to achieve query latencies of a few milliseconds on terabyte-scale collections [28]. However, these methods face a major indexing scalability challenge on large datasets due to a very high memory footprint and a prohibitive indexing time [35].

This work aims to study graph-based ng-approximate vector search approaches. We provide an overview of state-of-the-art methods, describing their design principles and highlighting their strengths and limitations. We conduct an exhaustive experimental evaluation of existing techniques and share some insights that were never published before [2]. We also propose ELPIS [5], a new in-memory ng-approximate vector search approach, which addresses the indexing scalability challenge of existing graph-based methods while maintaining efficient query answering. Finally, we summarize our ongoing work and pinpoint future research directions.

2 RELATED WORK

The vector search problem has attracted significant research interest over the past fifty years. Recently, approximate vector search has garnered more attention due to its appealing trade-offs, providing very efficient retrieval of answers with high accuracy. The methods proposed are based on either scans, trees, graphs, hashing, inverted indexes, or hybrid designs that combine multiple data structures.

In this section, we provide an overview of the main data structures used by state-of-the-art approaches, then discuss their strengths and limitations.

Summarization Techniques. *Piecewise Aggregate Approximation* (PAA) [20] and *Adaptive Piecewise Constant Approximation* (APCA) [6] segment a vector into equal or arbitrary-length segments, approximating each by its mean. The *Extended Adaptive Piecewise Approximation* (EAPCA) [36] enhances APCA using both

Proceedings of the VLDB Endowment. ISSN 2150-8097.

the mean and standard deviation. Symbolic Aggregate Approximation (SAX) [22] discretizes PAA values into a binary representation. *Quantization* maps continuous values to a finite set of codewords, with scalar and vector quantization operating on individual dimensions or entire vectors respectively. Product quantization divides the vector into sub-vectors for quantization, while Optimized Product Quantization (OPQ) improves it by decorrelating dimensions beforehand.

Tree-based Indexes have been the method of choice for exact vector search for data series and generic high-dimensional vectors [3, 36]. Some works have proposed using tree indexes to support approximate search with [9, 12] or without guarantees [3, 12, 24]. The tree-based methods that support ng-approximate search either build multiple randomized kd-trees [24], segment the space into smaller dimensions indexed by an RDB tree [3], or use heuristics to select candidates from some leaf nodes.

Hash-based Approaches primarily use locality-sensitive hashing (LSH) [19] for approximate vector search with guarantees [12]. These methods use hash functions to group similar data points into the same bucket with high probability [18, 31]. Trade-offs in search accuracy and efficiency are controlled by user-defined parameters specifying error and probability thresholds, and index building parameters like the number of hash tables and random projections.

Graph-based Approaches are currently favored for ng-approximate vector search. They commonly utilize a proximity graph structure [17, 33], where each vertex represents a data point and edges connect similar vertices. The connectivity between vertices is often determined by a distance measure like the Euclidean distance [13]. Search typically starts from a set of initial points or seeds, chosen randomly or based on specific criteria. The search process begins from one seed as an entry node and others as initial candidate answers. It proceeds by visiting neighboring vertices in a best-first, greedy manner until no better matches are found [27]. State-of-the-art methods like ELPIS [5], HNSW [23], EFANNA [14], and NSG [16] employ similar search algorithms [27] but vary in graph construction and seed point selection.

Summary. These data structures operate on either raw data or data summaries. Tree-based approaches offer efficient indexing due to their hierarchical structure, but struggle with hard queries due to the discrete nature of their partitions. Hash-based methods provide theoretical guarantees due to their probabilistic nature, but incur overhead due to the need to maintain multiple hash tables. Graph-based approaches provide the best query performance due to their efficient pruning of the search space, but lack guarantees and are expensive to construct. Some methods like ELPIS [5] and SPTAG [34] combine multiple structures to balance these trade-offs.

3 OUR WORK

We conduct an exhaustive experimental evaluation of in-memory graph-based vector search approaches [2] where we describe their key design choices and highlight their strengths and weaknesses. We propose ELPIS [5], an in-memory *ng*-approximate vector search algorithm, which combines the advantages of graph and tree-based index structures while mitigating the limitations inherent in each class. We also pinpoint our ongoing and future research directions, including the extension of the experimental study to out-of-core datasets and the enhancement of ELPIS to outperform the state-of-the-art in throughput-optimized settings.

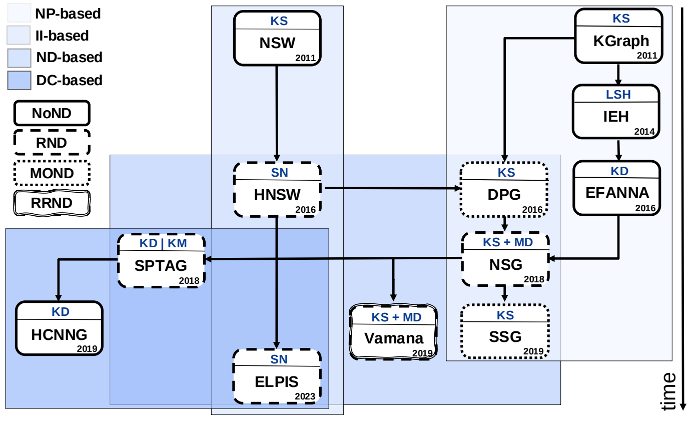

Figure 1: Graph-based vector indexes

3.1 Completed Work

3.1.1 Experimental Study. We evaluated ten state-of-the-art methods within a unified experimental framework using many evaluation criteria [2]. To facilitate reproducibility and support future research, we provide a public archive containing all source codes, datasets, and queries [1]. We present an elaborate discussion based on the deep insights gained about the graph-based approximate vector search problem. The main points are the following:

1. The index construction algorithm has an important impact on query performance. We identify and distinguish four main paradigms for constructing graph-based indexes:

Neighborhood Propagation (NP) refines an existing graph through a specified number of iterations of the neighborhood propagation process, also known as NNDescent [8]. During each iteration, nodes choose candidate neighbors from their own neighborhood list as well as the lists of their current neighbors.

Incremental Insertion (II) constructs a graph incrementally by adding vertices one at a time. Each new node connects with bidirectional edges to candidate neighbors found via beam search among the already inserted nodes [26]. Methods such as HNSW [23] and ELPIS reduce the neighborhood list size using the RND process, showing strong performance in indexing large datasets [4, 5, 12].

Neighborhood Diversification (ND) constructs a sparse graph, where nodes are not only connected to their closest neighbors but also to some distant neighbors. This reduces unnecessary comparisons during graph traversal, allowing quicker access to promising regions. ND-based methods have demonstrated significant efficiency improvements during searches [5, 15, 16, 21, 23, 30, 34].

Divide-and-Conquer (DC) methods partition the dataset into several, potentially overlapping, subsets and construct a separate graph for each. Methods like SPTAG [7] and HCNNG [25] merge these individual graphs into a single graph for beam search, while ELPIS [5] performs search in parallel on the separate graphs.

2. ND-based approaches have demonstrated superior performance compared to other methods, due primarily to the pruning of unnecessary edges during graph construction. We identify three

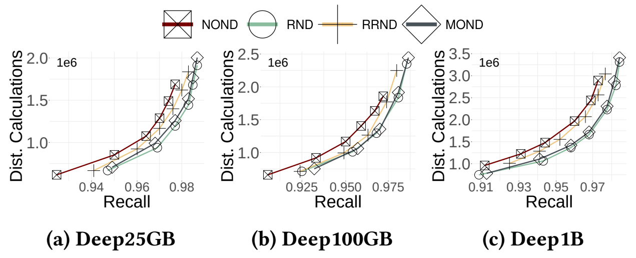

Figure 2: Evaluation of ND techniques (Deep)

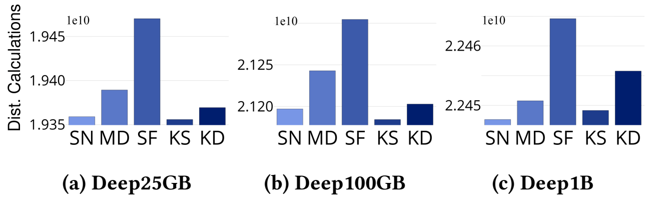

Figure 3: Evaluation of seed selection strategies (Deep)

strategies for diversification: Relative Neighborhood Diversification (RND), Relaxed Relative Neighborhood Diversification (RRND), and Maximum Orientation Diversification (MOND). RND, initially used by HNSW and later adopted by NSG, SPTAG, and ELPIS, selects a candidate neighbor as a neighbor for the current node, if it is closer to the current node than to any of the current node's existing neighbors, thus keeping only necessary edges for efficient traversal. RRND, introduced by VAMANA, is a more flexible version of RND that uses a relaxation factor to allow for a broader selection of neighbors, reducing the number of edges that get pruned. When this factor is set to one, RRND is equivalent to RND. Lastly, MOND, proposed by DPG and used in NSSG, focuses on maximizing the angles between connections. It selects neighbors based on their orientation to ensure that connections point in different directions. We compare these three ND strategies under a unified framework (same graph structure, dataset, and dataset size). The results, summarized in Figure 2, demonstrate that ND generally enhances search performance compared to non-diversified approaches (NOND), and RND and MOND have an equivalent performance overall.

3. The seed selection strategy, which determines the nodes used to start the search, has a significant impact on the number of distance calculations during both search and indexing. We identify four main seed selection strategies in the literature: Stacked-NSW (SN), Medoid (MD), Single Fixed Random Entry Point (SF), K-Sampled Random Seeds (KS), and KD-trees (KD). SN, used by HNSW, builds multi-level diversified NSW graphs with random nodes in each layer, organized hierarchically. KD, utilized by EFANNA, SPTAG-KDT, and HCNNG, builds one or more KD-trees from a dataset sample. The Single Fixed Random Entry Point (SF) method selects a random node as a permanent entry point for all searches. The K-Sampled Random Seeds (KS) selects k random nodes for each query to initialize candidate answers, a strategy supported by DPG, NSG, and VAMANA, which supplement the medoid entry point with random nodes. We evaluate these strategies using the same graph structure on the two popular real datasets Deep [29] and Sift [32], with sizes of 25GB, 100GB, and 1B. We run 100 queries for each strategy and extrapolate the results to 1 million queries, We report the number of distance calculations needed to achieve 0.99 accuracy in Figure 3. SN and KS emerged as the most efficient strategies across all scenarios, while SF and MD were the least efficient overall. The KD strategy performed well on Deep25GB and Deep100GB, but its performance declined on the billion-scale dataset. These results indicate the importance of adapting the seed selection strategy to the dataset size, and developing more sophisticated sampling methods and efficient lightweight index structures.

Figure 1 shows a roadmap of the state-of-the-art graph-based vector indexes, indicating the strategies used by each method in index construction, seed selection and diversification.

3.1.2 ELPIS. We propose ELPIS [5], a novel technique for in-memory ng-approximate vector search, that leverages the optimal features of both tree-based and graph-based methods to address the indexing scalability challenge of graph-based vector search techniques. ELPIS is the first in-memory graph-based vector search solution to combine EAPCA-based clustering with intra-query parallelism. It partitions the dataset into clusters using an EAPCA tree, then constructs parallel HNSW graph indexes for each cluster. This DC-based approach, coupled with ND pruning, improves indexing scalability by reducing the required outdegree. However, the resulting multiple graph structures present a search efficiency challenge. To address this, ELPIS employs multi-level pruning during search, utilizing EAPCA lower-bounding distances and k-th best-so-far (bsf) results. The search process begins with a heuristic depth-first search (DFS) traversal of the EAPCA tree to select an initial cluster, followed by beam search on its corresponding graph. The bsf results initialize search priority queues for other clusters, from which a subset is selected based on the bsf results and lower-bound distances between the query and the EAPCA summarization of each cluster. ELPIS then initiates concurrent beam searches on the selected clusters' graphs. This approach efficiently reduces unnecessary distance calculations compared to starting from scratch. Finally, ELPIS consolidates results from all candidate clusters to return the top-K answers.

We empirically demonstrate the efficacy, scalability, and robustness of ELPIS through extensive experiments on large real-world datasets from various domains. In Figure 4, we present our latest benchmark results for in-memory graph-based vector search on various Deep1B dataset sizes, 25GB, 100GB, 250GB and 380GB corresponding respectively to 66M, 266M, 666M and 1B vectors of 96 single-precision floating-points. ELPIS exhibits efficient indexing, achieving speeds 2 to 2.8 times faster than the best-performing methods, HNSW and VAMANA [30], on large scale Deep1 (Figure 4a). Furthermore, ELPIS requires less memory, utilizing up to 50% less, with less than 500GB (including the raw dataset vectors) of main memory needed to efficiently index a billion of vectors Deep1B dataset of 380GB size (Figure 4b). During the search phase, ELPIS consistently ranks among the top performers for small datasets and achieves the best performance for both Deep25GB (Figure 4c) and Deep1B (Figure 4d).

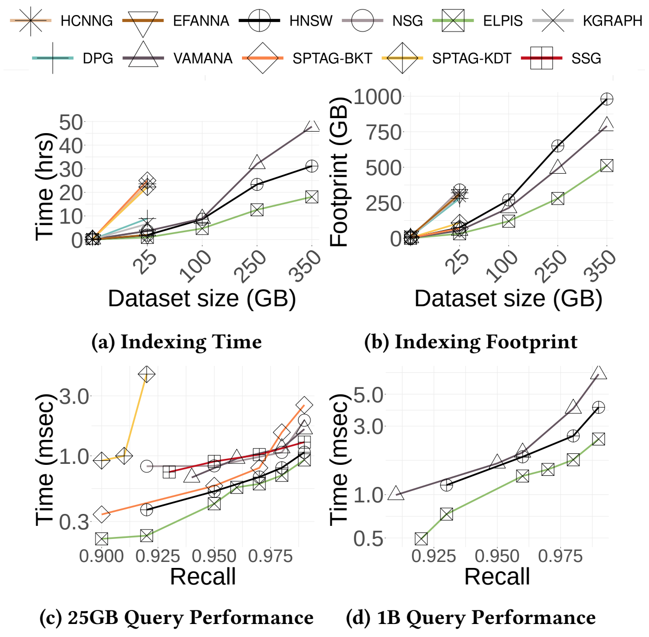

Figure 4: Index and search performance on the Deep dataset.

3.2 Ongoing and Future Work

Work In Progress. Our ongoing work focuses on enhancing the current latency-optimized ELPIS to support efficient query answering in throughput-optimized settings, leveraging graph merging techniques. Furthermore, we are expanding our evaluation to out-of-core graph-based vector search methods (e.g., DiskANN [30]) to assess the applicability of our findings to scenarios where datasets exceed main memory capacity.

Future Work. Currently, ELPIS leverages HNSW to construct graphs within its leaf nodes. We intend to develop a novel graph technique based on our experimental insights to further enhance the performance of ELPIS. Besides, we plan to adapt ELPIS for efficient out-of-core search using insights from our disk-based evaluation.

4 CONCLUSIONS

The goal of this work is to develop a graph-based index to support efficient approximate vector search on large datasets. First, we evaluate experimentally the state-of-the-art approaches, understand their key strengths and weaknesses, and pinpoint possible directions for improvement. Second, we propose a new technique called ELPIS, which is optimized for latency. Finally, we outline ongoing work and future research directions. In particular, we plan to extend our study to out-of-core data, enhance ELPIS to efficiently support throughput-optimized workloads, and propose a novel index to use within the ELPIS leaves, exploiting the insights learned about seed selection strategies and index diversification techniques.

REFERENCES

[1] ELPIS Archive. https://github.com/scalablesimilaritysearch/ELPIS, 2022.
[2] Graph-based vector similarity search: An experimental evaluation of the state-of-the-art (archive). *under submission*, 2024.
[3] 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.
[4] M. Aumüller, E. Bernhardsson, and A. J. Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Inf. Syst.*, 87, 2020.
[5] I. Azizi, K. Echihabi, and T. Palpanas. Elpis: Graph-based similarity search for scalable data science. *PVLDB*, 16(6), 2023.
[6] 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.
[7] 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.
[8] 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.
[9] 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.
[10] K. Echihabi, K. Zoumpatianos, and T. Palpanas. High-dimensional similarity search for scalable data science. ICDE, 2021.
[11] 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.
[12] 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.
[13] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer Publishing Company, Incorporated, 1st edition, 2012.
[14] C. Fu and D. Cai. Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph. *arXiv preprint arXiv:1609.07228*, 2016.
[15] 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.
[16] 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.
[17] K. R. Gabriel and R. R. Sokal. A new statistical approach to geographic variation analysis. *Systematic zoology*, 18(3):259–278, 1969.
[18] Q. Huang, J. Feng, Y. Zhang, Q. Fang, and W. Ng. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment*, 9(1):1–12, 2015.
[19] O. Jafari, P. Maurya, P. Nagarkar, K. M. Islam, and C. Crushev. A survey on locality sensitive hashing algorithms and their applications. *arXiv preprint arXiv:2102.08942*, 2021.
[20] 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.
[21] 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.
[22] 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.
[23] 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.
[24] 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.
[25] 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.
[26] 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.
[27] 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.
[28] 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.
[29] Skoltech Computer Vision. Deep billion-scale indexing. http://sites.skoltech.ru/compvision/noimi, 2018.
[30] 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.
[31] 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.
[32] TEXMEX Research Team. Datasets for approximate nearest neighbor search. http://corpus-texmex.irisa.fr/, 2018.
[33] G. T. Toussaint. Proximity graphs for nearest neighbor decision rules: recent progress. *Interface*, 34, 2002.
[34] 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.
[35] 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.
[36] 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.