COMP5311: Graph-Based ANNS Indexes of High-Dimensional Vectors

Elton Chun-Chai Li Jimmy Kwun-Hang Lau Sirui Han

Zhaoteng YE Jiahao Zhan

ABSTRACT

Efficient similarity search in high-dimensional data is crucial for numerous applications, driving the development of graph-based Approximate Nearest Neighbor (ANN) search techniques. This survey offers a comprehensive exploration of this prominent field. It begins with the fundamental principles of organizing data points into proximity graphs for efficient neighbor retrieval and then traces the evolution of core graph-based algorithms. The evolution spans from early Navigable Small World (NSW) graphs to advanced hierarchical and refined graph structures like HNSW and NSG. The construction and search mechanisms of these algorithms are examined, analyzing their underlying principles and inherent trade-offs in search accuracy, efficiency, and scalability. Furthermore, the survey investigates the adaptation of graph-based ANN search to multi-modal data, addressing the challenges and solutions for cross-modal similarity search. The problem of query hardness is also addressed, analyzing how query difficulty varies and exploring methods to measure and mitigate its impact on search performance. The survey concludes by outlining open challenges and future research directions in graph-based ANN search.

1 INTRODUCTION

The amount of high-dimensional data has increased dramatically. Finding similar items within these massive datasets is a critical operation in many applications, including recommendation systems, information retrieval, and multimedia search. However, the sheer size and dimensionality of this data pose significant challenges for traditional search methods. Exact nearest neighbor search becomes computationally infeasible, motivating the development of Approximate Nearest Neighbor (ANN) search techniques, which prioritize efficiency over absolute accuracy [11, 12, 17].

Among the various ANN search methods, graph-based approaches have emerged as a powerful paradigm, offering an excellent trade-off between search speed and accuracy [45, 46]. These methods construct a proximity graph where data points are nodes and edges connect similar points, enabling efficient search by navigating this graph structure [13, 46]. This survey aims to provide a comprehensive overview of graph-based ANN search, exploring its fundamental principles, key algorithms, important extensions, and open challenges.

Specifically, we will delve into the core graph-based ANN algorithms, tracing their evolution from early foundations like Navigable Small World (NSW) graphs [34] to modern hierarchical and refined graph structures such as HNSW and NSG [16, 36]. This evolution has been driven by the continuous need for improved search efficiency, better scalability to larger datasets, and more robust performance across diverse data distributions. We will examine the construction and search techniques employed by these algorithms, highlighting their strengths and weaknesses. Furthermore, we will explore how graph-based ANN search is adapted to handle multi-modal data, a crucial requirement for many modern applications [12]. Finally, we will analyze the problem of query hardness, addressing how the difficulty of ANN search can vary significantly between different queries [6, 7, 30, 47], and how to measure and mitigate this challenge [5, 6, 21, 47].

The survey is structured as follows: We begin with essential background on the ANN search problem and evaluation metrics (Section 2). We then delve into the heart of core graph-based ANN algorithms (Section 3), tracing their evolution from early foundations to modern construction and search techniques. Building on this foundation, we explore key extensions enabling multi-modal search (Section 4) and analyze the practical challenge of query hardness (Section 5). Finally, we synthesize the major open challenges and future directions (Section 6) before concluding (Section 7).

2 BACKGROUND

2.1 Problem Definition

The task of finding the most similar items to a given query within a large collection is fundamental across numerous computational domains. Formally, this is often framed as the k-Nearest Neighbor Search (k-NNS) problem.

Definition 2.1 (Nearest Neighbor Search - NNS). Given a dataset X comprising N points in a d-dimensional space $\mathbb{R}^d$, and a query point $q \in \mathbb{R}^d$, the objective of NNS is to identify the point $x \in X$ with the minimum distance $D(x, q)$ to the query. Let $NN(q)$ denote this point: $NN(q) = \arg\min_{x \in X} D(x, q)$ The k-Nearest Neighbors (k-NN) search identifies the set $R_k \subset X$ such that $|R_k| = k$ and contains the k points in X closest to q.

Finding the exact nearest neighbor(s) is computationally expensive, especially for large datasets (N) and high dimensions (d), often requiring costs proportional to $N \times d$ [17]. This linear dependency on dataset size renders brute-force methods computationally infeasible for the massive datasets commonly encountered in modern applications [17]. The challenge is significantly exacerbated in high-dimensional spaces, a scenario increasingly prevalent due to the success of machine learning techniques that generate rich vector embeddings to represent complex multi-modal data [12]. As dimensionality d increases, phenomena collectively termed the "Curse of Dimensionality" by Richard Bellman [11] arise, leading to data sparsity and making the concept of a "nearest neighbor" less meaningful as distances between points tend to concentrate [11]. This motivates the use of approximation techniques.

Definition 2.2 (Approximate Nearest Neighbor Search - ANNS). Given X and q, ANNS aims to efficiently find a point $\hat{x} \in X$ that is close to $NN(q)$. Often, the goal is to satisfy $D(\hat{x}, q) \leq (1 + \epsilon)D(NN(q), q)$ for some $\epsilon > 0$. More generally, for k-NN,

ANNS seeks an approximate result set $\hat{R}_k \subset X$ ($|\hat{R}_k| = k$). ANNS algorithms allow for a small, controllable sacrifice in accuracy in exchange for substantial improvements in search speed, memory usage, and overall computational efficiency [17]. This trade-off between accuracy (recall) and efficiency (latency, throughput) is central to ANNS, enabling practical solutions for large-scale, high-dimensional search problems where exactness is not strictly required [17]. The utility of ANNS is underscored by its widespread application in diverse fields, including recommendation systems, information retrieval, data mining, pattern recognition, computer vision, natural language processing (NLP), and vector databases supporting Retrieval-Augmented Generation (RAG) systems [17].

The quality of ANNS is typically evaluated by its search efficiency (e.g., throughput or latency) and accuracy, commonly measured by Recall@k: Recall@k = $\frac{|\hat{R}_k \cap R_k|}{k}$ where R_k is the true k-NN set. Graph-based ANNS methods, which construct and traverse proximity graphs over the dataset X, are known for achieving favorable trade-offs between accuracy and efficiency [45] and form the core subject of this survey.

2.2 Principles of Graph-Based ANNS

Graph-based ANNS (GB-ANNS) methods operate by constructing a data structure, specifically a proximity graph, where the relationships between data points guide the search process towards approximate nearest neighbors.

2.2.1 Proximity Graph Construction. The core concept of GB-ANNS involves representing each data point in the dataset as a vertex (or node) in a graph. Edges are then established between vertices that are considered "nearby" or "proximal" according to a chosen distance metric (e.g., Euclidean distance (L_2), cosine similarity) [13]. This constructed graph itself serves as the index structure, facilitating efficient navigation through the high-dimensional space [13]. The extraordinary ability of graphs to express neighbor relationships allows these algorithms to evaluate fewer data points compared to other indexing structures while achieving high accuracy [46].

The design of practical GB-ANNS algorithms is often inspired by theoretical graph structures known to have desirable properties for nearest neighbor search. Foundational concepts include:

- **K-Nearest Neighbor Graphs (KNNG):** Graphs where each vertex is connected to its exact k nearest neighbors in the dataset. Building an exact KNNG can be computationally expensive [46].
- **Relative Neighborhood Graphs (RNG):** Graphs where an edge exists between two points u and v if no other point w is closer to both u and v than they are to each other [46]. RNG-based pruning strategies are noted for yielding state-of-the-art performance in some evaluations [50].
- **Delaunay Graphs (DG):** Geometric structures with strong theoretical properties for proximity, but generally inefficient to construct in high dimensions without prior information [35].
- **Minimum Spanning Trees (MST):** An MST links all data points using the shortest possible total "length" of connecting lines (edges based on distance), forming a simple tree with no loops. This sparse graph reveals core data connections and serves as a foundational building block for more complex ANNS graph structures [46].

Practical algorithms typically construct approximations or variations of these ideal graphs to balance construction cost with search performance [46].

2.3 Core Search Mechanisms

Once the proximity graph index is constructed, queries are processed by navigating this graph structure.

2.3.1 Greedy Search / Best-First Search. This is the fundamental search strategy employed by most GB-ANNS algorithms [13]. The process typically begins at one or more designated *entry points* in the graph [52]. The algorithm maintains a dynamic list, pool, or priority queue of candidate nodes to visit, ordered by their distance to the query vector q. In each step, the algorithm selects the most promising candidate (i.e., the unexplored node closest to q) from the list, computes distances to its neighbors in the graph, and adds these neighbors to the candidate list [52]. A set of the overall closest nodes encountered so far is maintained as the potential result set [52]. The search continues iteratively, exploring progressively closer regions of the graph. Termination typically occurs when a stopping criterion is met, such as when all candidates in the list are farther from the query than the k-th nearest neighbor already found, or when the distances to unexplored nodes begin to consistently increase, indicating the search is moving away from the true nearest neighbors [52]. A significant portion of the search time is often consumed by the repeated distance computations [45].

2.3.2 Beam Search. To manage the computational cost, especially the number of distance calculations, a common heuristic applied within the greedy search framework is *beam search* [18, 48, 49]. Beam search limits the size of the dynamic candidate list (the "beam") to a fixed width, often denoted by parameters like ef or L [28]. At each step, after exploring neighbors, the candidate list is pruned to retain only the closest candidates within the beam width [38]. This bounds the number of nodes considered at each stage, controlling the trade-off between search thoroughness (and thus recall) and computational effort (latency/QPS) [18, 48, 49].

2.3.3 Entry Point Selection. The choice of the starting node(s) significantly influences search efficiency. Simple strategies include selecting a random node [41] or using a fixed, centrally located node (e.g., the medoid) [28]. More sophisticated *adaptive* methods aim to select an entry point closer to the query vector itself. One approach involves pre-clustering the dataset using k-means; candidate entry points are chosen as the actual data points nearest to each cluster centroid. Given a new query, a brute-force search among these candidate entry points identifies the closest one to start the main graph traversal [39]. Theoretical analysis suggests adaptive entry points can improve search performance bounds under certain conditions [39].

2.4 Theoretical Considerations and Graph Properties

While GB-ANNS algorithms are often evaluated empirically, theoretical concepts help understand their behavior and limitations.

2.4.1 Monotonicity Concepts. An ideal search graph would allow monotonic traversal, where every step along a path from a start node to a target node strictly decreases the distance to the target. A graph guaranteeing such a path between any two nodes is called a *Monotonic Search Network (MSNET)* [49]. However, constructing perfect MSNETs is often impractical for large, high-dimensional datasets. Recognizing that practical approximate graphs may require occasional "backward" steps, the concepts of *b-monotonic paths* and *B-MSNETs* were introduced, where a b-monotonic path allows up to b steps that increase the distance to the target [39]. The *Approximate Monotonic Search Network (AMSNET)* concept similarly considers distance backtracking [49]. These relaxed monotonicity concepts provide a theoretical framework for analyzing the search performance on the types of imperfect, approximate graphs commonly built in practice, bridging the gap between theory and the algorithms evaluated experimentally [49].

2.4.2 Small World Properties. The goal of leveraging NSW principles [34, 41] is to achieve efficient navigation. Ideally, the graph structure allows greedy search to converge to the target neighborhood in a number of steps that scales logarithmically or polylogarithmically with the dataset size n [35]. This is facilitated by the presence of both short-range and long-range links.

2.4.3 Graph Quality Metrics. Beyond search performance, properties of the constructed graph itself are sometimes evaluated. These include *connectivity* (e.g., number of connected components, ensuring the entire dataset is reachable), *average out-degree* (average number of neighbors per node, related to index size and search branching factor), and the *recall of true neighbors within the graph* (percentage of nodes whose true nearest neighbor is directly connected via an edge) [46]. While intuitively important, experimental evaluations have shown that higher graph quality metrics do not always guarantee better search performance [46]. For instance, HNSW might have lower graph quality than DPG but achieve better search results in some cases [46]. Poor connectivity, specifically nodes with low in-degree, has been identified as a cause of recall loss in HNSW, as these nodes are hard to reach during search [32].

3 GRAPH-BASED INDEX ANNS

3.1 Principles of Graph-Based Index ANNS

Graph-based Approximate Nearest Neighbor Search (ANNS) methods represent a dominant paradigm for efficient similarity search in high-dimensional spaces. These techniques fundamentally operate by constructing a proximity graph where each data point from the dataset is a node, and edges connect nodes that are considered "nearby" according to a chosen distance metric, such as Euclidean distance or cosine similarity [13]. This graph structure itself serves as the index, guiding the search process [13]. The inherent ability of graphs to articulate neighbor relationships allows these algorithms to achieve high accuracy while evaluating significantly fewer data points compared to other indexing strategies [46]. The construction of these proximity graphs is often inspired by theoretical graph structures like K-Nearest Neighbor Graphs (KNNG), Relative Neighborhood Graphs (RNG) [46], and Delaunay Graphs (DG) [36], aiming to balance construction cost with search performance.

3.1.1 Navigable Small World (NSW) Graphs. The Navigable Small World (NSW) graph approach, introduced by Malkov et al. [34], was an early and influential method. NSW graphs represent the dataset as a graph where data points are vertices and edges signify proximity. Construction is incremental: new elements are added and connected to their approximate nearest neighbors within the existing graph structure. A key characteristic of NSW is the preservation of links established early in the construction; these links, initially short-range, naturally evolve into the long-range connections essential for efficient navigation as the graph expands, thereby creating the "navigable small world" property without explicit engineering of link length distributions [34]. Search is typically a greedy graph traversal starting from an entry point, iteratively moving to the neighbor closest to the query until a local minimum is found. NSW graphs aim to emulate the "small-world phenomenon" by incorporating both short-range links for accuracy and long-range links for rapid traversal, targeting logarithmic or poly-logarithmic search complexity [34]. While foundational, NSW can suffer from a poly-logarithmic search complexity ($O(\log^2(N))$) for a fixed recall and the risk of greedy search terminating in local minima [34]. Early NSW variants also faced challenges with power-law scaling in routing steps on certain data distributions, such as clustered or low-dimensional data , which motivated hierarchical extensions [36].

3.2 Hierarchical Navigable Small World (HNSW) Graphs

To address the limitations of flat NSW graphs, Malkov and Yashunin proposed the Hierarchical Navigable Small World (HNSW) algorithm [36], which has become one of the most popular and successful GB-ANNS methods [45]. HNSW introduces a multi-layer hierarchy of proximity graphs, drawing inspiration from probability skip lists. The uppermost layers are sparse, featuring fewer nodes and predominantly long-range connections to facilitate rapid traversal across the data space. In contrast, lower layers are progressively denser, incorporating more nodes and shorter-range links, which enables precise, fine-grained search within local neighborhoods [36].

The construction of an HNSW graph is an incremental process. When a new element is inserted, it is assigned a maximum layer level randomly, following an exponentially decaying probability distribution, ensuring that upper layers remain sparse. The element is then inserted into each layer from the top down to the base layer (layer 0). At every layer, a search is performed to identify the element's nearest neighbors within that layer, and connections are established. HNSW employs heuristics to choose which neighbors to connect to, thereby enhancing graph quality and search efficacy, especially for clustered data distributions. Key parameters influencing construction include M, the maximum number of connections per node in layers 1 and above; $efConstruction$, the size of the

dynamic candidate list used during neighbor discovery in the construction phase; and $Mmax0$, the maximum number of connections in the base layer, typically set to $2 \times M$ [36].

Searching in HNSW mirrors its layered construction [36]. The process initiates at a designated entry point in the topmost layer. A greedy search, usually implemented as a beam search, is conducted within this layer to find the node(s) closest to the query vector. These nodes then serve as entry points for the search in the subsequent layer below. This iterative refinement continues, descending through the hierarchy until the base layer, which contains all data points, is reached, where a final search is performed. The primary search parameter, ef or $efSearch$, dictates the size of the candidate beam at each layer. This hierarchical navigation strategy allows HNSW to achieve an expected search complexity of $O(\log(N))$.

HNSW is recognized for its state-of-the-art performance in balancing recall and query throughput across diverse benchmarks [45]. It also offers support for efficient dynamic updates (insertions and, in some implementations, deletions) and demonstrates robustness across various distance metrics and data types, including non-metric spaces where basic NSW struggled. However, HNSW is not without its drawbacks. It can inherit the power-law scaling issues of NSW on certain datasets. Graph connectivity problems, such as nodes with low in-degree being difficult to reach, can negatively impact recall [32]. Achieving very high recall often necessitates a large $efSearch$ value, which can increase latency due to more extensive exploration [32]. The hierarchical structure can also lead to a significant memory footprint, and index construction times can be considerable, especially for large, high-dimensional datasets. Furthermore, HNSW's performance is sensitive to parameter tuning [53], and different implementations (e.g., in nmslib, hnswlib, Faiss) exhibit varying trade-offs regarding build time, memory efficiency, update support, and query speed [45].

3.3 Navigating Spreading-out Graph (NSG)

The NSG algorithm, introduced by Fu et al. [16], focuses on constructing a high-quality, navigable, flat graph. It often starts with a pre-computed approximate k-NN graph (e.g., built using methods like KGraph or NN-Descent [51]) and then refines this initial graph to optimize search paths and ensure good "spreading-out" properties [46]. This refinement typically involves selecting a navigation starting point (e.g., the graph's medoid) and then applying specific edge pruning and addition strategies, sometimes based on concepts like Monotonic Relative Neighborhood Graphs (MRNG), to ensure that greedy search can effectively converge [16]. The aim is to create a sparse, directed graph that offers near-logarithmic search complexity and low memory overhead while ensuring graph connectivity, often through mechanisms like a DFS spanning tree with reconnection for isolated nodes [16].

NSG was developed to address challenges in balancing search efficiency, indexing complexity, and scalability, particularly for billion-scale datasets [16]. The underlying MRNG concept aims for a monotonic search network, which, in theory, guarantees logarithmic search complexity by ensuring paths where each step moves closer to the query, thus eliminating backtracking [16]. While constructing a perfect MRNG can be complex, NSG provides a practical approximation. Key parameters for NSG include R (final out-degree), L (candidate list size during build/search), and C (candidate pool size for refinement) [28]. Search is performed via a greedy graph traversal starting from the designated entry point.

NSG has demonstrated strong performance, reportedly achieving significantly faster search speeds than HNSW and quantization-based methods like IVFPQ [29] at high precision, while using considerably less memory than hierarchical graph indices in some experiments [16]. Its deterministic structure, compared to HNSW's probabilistic layering, can reduce redundant comparisons and improve cache locality [16]. The algorithm's scalability has also been validated in distributed settings [16]. However, NSG's construction often relies on an initial k-NN graph, which adds a preliminary step compared to incremental methods like HNSW [28]. Its relative performance compared to HNSW and Vamana can vary depending on the dataset and parameter settings, and it might require more hops in disk-based scenarios compared to graphs specifically optimized for disk, like Vamana, due to potentially larger graph diameters [28]. The success of NSG in industrial applications, such as Alibaba's search engine [43], highlights its practical relevance for large-scale, high-dimensional retrieval [16].

3.4 Vamana (DiskANN)

The Vamana algorithm, integral to the DiskANN system, is engineered for efficient ANNS on datasets that exceed main memory capacity, primarily utilizing Solid State Drives (SSDs) [28]. Unlike hierarchical structures, Vamana constructs a flat, non-hierarchical, directed proximity graph [28]. Although optimized for disk-based operations, the Vamana graph structure can also be effectively deployed in purely in-memory scenarios.

Vamana's construction process is iterative and distinct. It typically commences by initializing a graph where each vertex is connected to a set of random out-neighbors, up to a predefined limit R. This initial graph undergoes refinement over several passes, usually two. In each pass, for every data point p (processed in a random order), a greedy search is initiated from a global entry point, such as the dataset's medoid. This search aims to identify approximate neighbors of p, and the set of visited nodes, V_p, is accumulated. The cornerstone of the refinement is the `RobustPrune` procedure. This procedure meticulously selects at most R out-neighbors for p from the visited set V_p. The selection is guided not only by proximity to p but also by a distance threshold parameter, α, which encourages the retention of some longer-range edges crucial for efficient navigation. Backward edges are subsequently added from the newly selected neighbors back to p, and `RobustPrune` may be reapplied if necessary to ensure the maximum out-degree constraint R is maintained. The overarching objective of this construction methodology is to produce graphs with a smaller diameter, thereby minimizing the number of sequential disk reads required during search, a critical factor for SSD-based performance. Vamana also supports building indices on overlapping partitions of a dataset which can then be merged, enabling the indexing of datasets that are too large for the available RAM. Key construction parameters include R (the maximum out-degree), L (the search list size during the construction phase), and α (controlling pruning aggressiveness).

Search within a Vamana graph is performed using a standard greedy graph traversal, commonly implemented as a beam search

[28]. The search process begins from a designated entry point, which is often the medoid of the dataset identified during the graph construction phase.

Vamana's primary strength is its demonstrated effectiveness within the DiskANN system, enabling billion-scale ANNS on commodity hardware (SSD with moderate RAM) while achieving high recall and low query latencies. Its in-memory performance has been reported as competitive with, and in some cases superior to, HNSW and NSG, especially on high-dimensional datasets like GIST1M. Some studies also indicate that Vamana can achieve faster indexing times compared to HNSW and NSG. The smaller graph diameter, a result of the `RobustPrune` strategy, is particularly beneficial for reducing disk I/O operations. Furthermore, Vamana is versatile, serving as the foundation for various in-memory, disk-based, and filtered ANNS solutions such as FilteredVamana and StitchedVamana [19, 28].

However, Vamana is not without its limitations. Some comparative evaluations suggest that Vamana's index construction can be slower than HNSW's. While designed for disk, its performance might be less favorable than HNSW in scenarios with highly constrained RAM. The flat graph structure, when used in purely in-memory searches, might not offer the very rapid initial traversal phase that HNSW's upper layers provide in certain situations. Additionally, its performance on particularly challenging datasets, such as Text2Image, was noted to be less robust compared to HNSW or HCNNG in at least one billion-scale comparative study.

3.5 Evaluation Frameworks for Graph-Based ANNS

The experimental evaluation of GB-ANNS algorithms is crucial for understanding their performance characteristics and enabling meaningful comparisons. A robust evaluation methodology considers benchmark datasets, performance metrics, and standardized evaluation protocols and tools [13, 46].

3.5.1 Benchmark Datasets. The choice of datasets profoundly influences evaluation outcomes. Performance can vary considerably across datasets with different properties, so evaluations ideally employ a diverse collection. Key characteristics to vary include dataset size (cardinality, ranging from millions to billions of data points), dimensionality (low to very high), and data type and distribution. Real-world datasets often include image features (e.g., SIFT1M, GIST1M, DEEP1M), text embeddings (e.g., GloVe, NYTimes), and audio features [13, 46, 53]. Synthetic datasets may also be used for controlled experiments to isolate specific algorithmic behaviors. The nature of queries, such as standard k-NN queries, range queries [37], or queries on out-of-distribution data, also forms an important aspect of dataset selection [13]. Commonly cited datasets include SIFT1M, GIST1M, DEEP1M, GloVe, various billion-scale datasets like SIFT1B and DEEP1B, and specialized datasets like Text2Image for cross-modal scenarios [13, 46].

3.5.2 Standard Performance Metrics. A comprehensive evaluation of GB-ANNS algorithms necessitates measuring multiple performance facets. For **search quality**, Recall@k is the most fundamental metric, representing the fraction of the true k nearest neighbors found within the top k results returned by the algorithm; variants like Recall@1, Recall@10, or Recall@100 are commonly reported [9, 46]. For range searches, range recall measures the fraction of true neighbors within a specified query range that are retrieved [13].

Regarding **search efficiency**, Queries Per Second (QPS) or throughput indicates how many queries can be processed in a unit of time and is often plotted against recall to illustrate the trade-off. Latency, or query time, measures the average time taken to process a single query, typically in milliseconds [49]. Both QPS and latency are highly dependent on the underlying hardware [13]. A more hardware-agnostic measure is the number of distance computations performed per query, reflecting the algorithmic work [13]. The number of hops or query path length, meaning the number of nodes visited or edges traversed during the search, is also relevant, particularly for disk-based systems where hops can correlate with I/O operations [46].

Finally, **index characteristics and build performance** are critical. Build time, the wall-clock time required to construct the index, is increasingly important for dynamic datasets but is sometimes overlooked in query-focused benchmarks [13]. Memory usage, encompassing both the RAM consumed during search and the storage space for the index (on disk or in memory), is a key factor for scalability and cost, especially with billion-scale datasets [13]. Intrinsic graph quality metrics, such as average out-degree, graph connectivity, and the percentage of true nearest neighbors directly linked in the graph, can offer insights but do not always directly correlate with search performance [46]. For dynamic scenarios, insertion and deletion speeds are also pertinent metrics.

3.5.3 Evaluation Protocols and Tools. Standardized protocols and publicly available tools facilitate reproducible and comparable evaluations. Benchmarking frameworks like 'ann-benchmarks' [4] are widely used, providing pre-processed datasets, containerized algorithm implementations, and scripts for running experiments. Other frameworks, such as WEAVESS [46], have been developed with reimplementations of multiple algorithms in a common language to ensure fairer comparisons of core algorithmic ideas. For vector database systems, tools like VectorDBBench focus on system-level aspects beyond standalone algorithm performance.

Parameter tuning is a critical aspect, as GB-ANNS algorithms are often highly sensitive to their hyperparameters [53]. Common tuning approaches include grid search over a predefined parameter space on a validation set, or more advanced techniques like Bayesian optimization, which can be more efficient in finding good parameter configurations [46]. Research is also active in developing automated tuning methods that can select optimal parameters without requiring repeated, costly index rebuilding [53].

Given the hardware dependency of metrics like QPS and latency, clear reporting of the experimental hardware setup (CPU, RAM, GPU, SSD specifications) is essential [13]. Some studies may intentionally disable hardware-specific optimizations (like SIMD instructions) to focus on algorithmic comparisons [46]. The degree of parallelism, such as the number of CPU threads used for index construction and search, should also be specified [13]. Ensuring reproducibility through publicly available code, datasets, parameters, and evaluation scripts is vital for scientific progress [46]. However, a persistent challenge is balancing standardized

Table 1: Comparison of HNSW, NSG, and Vamana (DiskANN) ANNS Algorithms

Feature	HNSW	NSG	Vamana (DiskANN)
Core Structure	Multi-layer hierarchy of proximity graphs.	High-quality, navigable, flat graph (often directed).	Flat (single-layer) directed proximity graph.
Construction Principles	Incremental insertion; random max layer level (exp. decay); top-down layer insertion; heuristics like M, efConstruction, $Mmax0$.	Often starts with pre-computed approx. k-NN graph; refines using strategies like MRNG-based pruning/addition; designated navigation start point (e.g., medoid); parameters R, L, C.	Random initialization + iterative refinement; `RobustPrune` procedure with α parameter; medoid entry point; parameters R, L, α.
Recall vs. QPS/Latency	State-of-the-art performance, $O(\log(N))$ expected search complexity.	Strong performance, potentially faster than HNSW at high precision in some cases; near-logarithmic complexity.	In-memory: competitive/superior to HNSW/NSG on some datasets (e.g., GIST1M). Disk-based: high recall with low latency for billion-scale data.
Build Time	Can be considerable, especially for large, high-dimensional datasets.	Relies on an initial k-NN graph construction, adding a preliminary step.	Reported faster than HNSW/NSG in some studies, can be slower than HNSW in others.
Memory Usage	Hierarchical structure can lead to a significant memory footprint.	Generally low memory overhead; reported to use considerably less memory than hierarchical graphs in some experiments.	Memory usage is dependent on the configured graph degree R and is optimized for disk-based systems.
Scalability (In-Memory)	Widely used and performs well on large datasets.	Scalability validated, including in distributed settings.	Good in-memory performance; forms the basis for the scalable DiskANN system.
Suitability for Disk	Not primarily designed for disk; hierarchical nature might pose challenges [36].	Might require more hops (disk I/Os) than Vamana due to potentially larger graph diameters [28].	Specifically designed for disk (as DiskANN); smaller graph diameter beneficial for minimizing disk I/O.
Dynamic Updates	Supports efficient dynamic updates (insertions and, in some implementations, deletions) [36].	Less emphasis on dynamic updates; construction typically not incremental.	Supports incremental insertions; forms basis for Filtered/StitchedVamana.

benchmarking using authors' original, highly optimized implementations against the fairness sought by reimplementing algorithms in a uniform framework, as the former may include varying levels of non-algorithmic optimizations while the latter might miss specific crucial optimizations [13, 46].

4 MULTI-MODAL GRAPH-BASED INDEX

4.1 Importance of Cross-Modal Search

Imagine typing "sunset beach" into a search engine and expecting it to find images of similar scenes. This task, known as cross-modal approximate nearest neighbor search (ANNS), involves matching a query from one data type (e.g., text) to items in another (e.g., images). It's a cornerstone of modern applications like image search engines, recommendation systems, and AI assistants, where users expect seamless interaction across diverse data modalities. For example, in e-commerce, a customer might describe a "red leather handbag with gold accents" and anticipate relevant product images, even without knowing specific details. Similarly, in digital libraries, researchers may seek visuals corresponding to textual descriptions, while retrieval-augmented generation (RAG) systems rely on cross-modal search to enhance AI responses with multimodal context.

The importance of cross-modal search lies in its ability to bridge heterogeneous data types, enabling intuitive and efficient access to information. As multimedia content proliferates, the demand for such capabilities grows, making cross-modal search indispensable for improving user experience and operational efficiency. However, the challenge arises from the distinct distributions of text and image data in their respective embedding spaces. Unlike traditional ANNS, where queries and database items share similar distributions, cross-modal search must contend with this disparity, rendering conventional methods less effective and necessitating specialized solutions.

4.2 Background and Challenges in Cross-Modal ANNS

4.2.1 The Rise of Cross-Modal Retrieval. Approximate Nearest Neighbor Search (ANNS) is a pivotal operation in numerous domains, from recommendation systems to large-scale information retrieval. Traditionally, ANNS assumes that queries and database items (base data) originate from the same modality, sharing a common distribution in a high-dimensional embedding space. However, with the advent of multimodal neural models—such as CLIP, which embeds text and images into a shared space—cross-modal ANNS has gained prominence. This paradigm allows a query from one modality (e.g., text) to retrieve the most similar items from another (e.g., images or videos), supporting applications like text-to-image search or video retrieval.

The allure of cross-modal ANNS lies in its versatility. It powers real-world systems where users interact naturally across modalities, such as searching for products with textual descriptions or

enhancing AI responses with visual context. Yet, this flexibility introduces a significant hurdle: the inherent distribution gap between modalities. Text embeddings and image embeddings, even when projected into a shared space, exhibit distinct statistical properties, making cross-modal queries out-of-distribution (OOD) relative to the base data.

4.2.2 Out-of-Distribution Queries: A Fundamental Challenge. In traditional ANNS, queries are in-distribution (ID), meaning they align with the base data's distribution. Graph-based indexes like HNSW or NSG thrive under this assumption, connecting spatially proximate vectors to enable efficient beam search. However, cross-modal queries violate this premise. Quantitative analyses, such as those using the Mahalanobis distance, reveal that OOD queries deviate 10 to 100 times farther from the base data than ID queries across datasets like Text-to-Image, LAION, and WebVid. Moreover, the k-nearest neighbors of an OOD query are spatially dispersed, unlike the clustered neighbors of an ID query.

This dispersion and distance exacerbate search inefficiency. For instance, experiments on HNSW show that OOD queries require over 500 hops to achieve a recall@10 of 0.93 on LAION, compared to just 48 hops for ID queries—a tenfold increase in search path length. Similarly, partition-based methods like IVF must probe significantly more clusters for OOD queries, reducing performance by up to 2.5 times on real-world datasets. These findings underscore a critical mismatch: existing ANNS methods, optimized for ID workloads, falter when faced with the OOD nature of cross-modal search.

4.2.3 Limitations of Traditional Approaches. State-of-the-art ANNS indexes assume that queries are proximate to the base data and that their nearest neighbors are spatially close. Graph-based methods construct approximate k-nearest neighbor graphs where edges link nearby vectors, expecting beam search to converge rapidly. However, OOD queries break these assumptions. Their nearest neighbors, scattered across the embedding space, require extensive graph traversal, increasing the number of hops and computational cost. Previous attempts, like RobustVamana, mitigate this by incorporating query vectors into the graph, yet they still lag significantly behind ID performance, with search speeds 3.9 to 10 times slower on cross-modal datasets.

This performance degradation highlights the need for a tailored solution that leverages the query distribution to enhance cross-modal search efficiency, rather than relying solely on base data proximity.

4.3 RoarGraph: Design and Methodology

4.3.1 Conceptual Foundation. RoarGraph emerges as a novel graph index designed specifically for cross-modal ANNS, addressing the OOD challenge by integrating query distribution knowledge into its structure. Unlike traditional indexes that prioritize spatial proximity among base data, RoarGraph redefines connectivity based on query perspectives. Its key innovation is to ensure that base data vectors, which are nearest neighbors to queries, are closely linked in the graph, even if they are spatially distant in the embedding space. This query-guided approach minimizes search detours and accelerates convergence to the true nearest neighbors.

4.3.2 Construction Process. The construction of RoarGraph involves three key steps:

(1) **Query-Base Bipartite Graph Construction:** A bipartite graph is built with query vectors and base data vectors as two disjoint sets. Edges connect each query to its top-N_q nearest neighbors in the base data, capturing the query's perspective on relevance.
(2) **Neighborhood-Aware Projection:** The bipartite graph is projected onto the base data, creating edges between base vectors that share common query neighbors. This step transforms query-defined proximity into direct connections, with a degree limit of M to maintain efficiency.
(3) **Connectivity Enhancement:** Supplementary edges are added using beam search to ensure all base vectors are reachable and to optimize search paths. This step incorporates reverse edges and leverages the projected graph to enhance navigability.

4.3.3 Search Algorithm. RoarGraph utilizes a standard beam search algorithm for querying. Starting from an initial node (e.g., the base data medoid), the search iteratively expands to the closest unvisited neighbors based on a priority queue of size L, continuing until convergence. The query-guided structure of RoarGraph ensures that OOD neighbors are accessible via short paths, enhancing search efficiency.

4.4 Experimental Validation and Performance Analysis

To assess the efficacy of RoarGraph, extensive experiments were conducted on three large-scale cross-modal datasets: Text-to-Image (10M vectors, 200 dimensions), LAION (10M vectors, 512 dimensions), and WebVid (2.5M vectors, 512 dimensions). These datasets feature text queries against image or video base data, with similarity measured via inner product (MIPS) or cosine distance. The performance of RoarGraph was benchmarked against state-of-the-art graph indexes, including HNSW, NSG, τ-MNG, and RobustVamana, using recall@k (for $k = 1, 10, 100$) and queries per second (QPS) as the primary metrics. All algorithms were implemented using their official codes and run in single-thread mode on a machine equipped with dual Intel Xeon Gold 5318Y CPUs and 512 GB of memory. RoarGraph consistently outperformed all baselines across all datasets and recall levels. For instance, at a recall@10 of at least 0.9, RoarGraph achieved significant speed-ups: 1.84× on Text-to-Image, 2.58× on LAION, and 3.56× on WebVid compared to the fastest competing method. Moreover, RoarGraph demonstrated the capability to reach exceptionally high recall levels, such as recall@100 ≥ 0.99, which were unattainable by other methods within practical time constraints. This superior performance is attributed to RoarGraph's query-guided structure, which ensures that the nearest neighbors of OOD queries are connected via short paths, thereby reducing the number of hops required during search. Indeed, analysis of routing hops revealed that RoarGraph reduced the search path length to 10.9%–44.1% of that required by HNSW, confirming its efficiency in navigating OOD queries.

Ablation studies further validated the contributions of each phase in RoarGraph's construction. The query-base bipartite graph alone

provided only modest performance, as its high out-degrees led to increased computational overhead during search. However, the neighborhood-aware projection significantly boosted QPS by controlling node degrees and transforming query-defined proximity into direct edges between base vectors. The connectivity enhancement step, while introducing slight overhead at lower recall levels due to increased node degrees, proved instrumental in achieving high recall by improving graph navigability and reducing path lengths. For example, on the Text-to-Image dataset, RoarGraph with connectivity enhancement was 1.49× faster than the projected graph alone at high recall levels. RoarGraph also exhibited robustness and scalability. Even when constructed with a reduced query set size (e.g., 10% of the base data), it retained 70.8%–88.7% of its peak QPS at recall@100 ≥ 0.95, while still outperforming traditional methods like HNSW by 1.44–4.38× at recall@10 = 0.9. Additionally, RoarGraph handled in-distribution (ID) queries competitively, ensuring its versatility across different query types. In terms of resource efficiency, RoarGraph's index sizes were comparable to those of NSG, ranging from 5.07 GB to 20.64 GB, and its construction time, though higher than that of HNSW, was mitigated by using smaller query sets, offering a practical trade-off between build cost and search performance. These results underscore RoarGraph's ability to deliver superior cross-modal search performance by aligning its graph structure with the query distribution, thereby addressing the fundamental challenges posed by OOD queries. Its efficiency, scalability, and robustness make it a compelling solution for large-scale multimedia retrieval systems.

4.5 Implications and Future Directions

RoarGraph represents a significant advancement in cross-modal ANNS, addressing the OOD challenge by aligning graph structure with query needs. Its ability to connect dispersed neighbors enhances search efficiency, making it invaluable for multimodal retrieval systems. Future research could explore adaptive updates for evolving query distributions, integrate vector quantization for scalability, or optimize multi-threaded performance. Extending RoarGraph to other OOD scenarios or refining deletion mechanisms are also promising directions.

5 QUERY HARDNESS ANALYSIS

Approximate Nearest Neighbor (ANN) search is a fundamental operation, with graph-based indexes emerging as state-of-the-art for many high-dimensional vector search tasks. A significant practical challenge is that their performance can vary dramatically with the specific query point [6, 7, 30, 47]. While many "simple queries" are processed efficiently, "hard queries" demand substantially more computational resources (e.g., distance calculations, graph traversals). This unpredictability impacts Quality of Service, resource provisioning, and downstream applications like recommendation systems [20, 23, 33] or Retrieval-Augmented Generation (RAG), where low recall on hard queries can impair model accuracy [47, 54]. Understanding and quantifying this query "hardness", the inherent difficulty of finding nearest neighbors for a specific query [6, 21, 47], is therefore crucial. Challenges in nearest neighbor search are often linked to the "curse of dimensionality" [11], where contrast between nearest and farthest neighbors diminishes. However, observed performance variability in graph-based ANN suggests local data characteristics and the specific graph topology around a query play a crucial role. Effectively measuring query hardness is essential for evaluating index robustness, optimizing performance, and ensuring reliable benchmarking, particularly as standard random workloads may not adequately stress test indexes. This section surveys query hardness analysis for graph-based ANN. We first review established metrics derived primarily from local data distribution (Local Intrinsic Dimensionality, Relative Contrast, Query Expansion) [6, 21], discussing their conceptual basis. We then examine recently proposed graph-native measures, like Steiner-Hardness [47], designed to capture search complexities within the graph structure itself, as distribution-centric measures often prove insufficient for graph-based indexes.

5.1 Common Hardness Measures

5.1.1 Relative Contrast (RC). Building upon earlier investigations into search "meaningfulness" in high dimensions [10, 15], He et al. [21] introduced Relative Contrast (RC) to assess ANN search difficulty. The motivation was to move beyond purely asymptotic analyses and evaluate the simultaneous influence of dimensionality, sparsity, and database size. Originally a global property of a dataset X, Relative Contrast (C_r) is the ratio of the expected distance between a random query q and a random database point x ($D_{mean} = E_q[E_x[D(x,q)]]$) to the expected distance between q and its actual nearest neighbor ($D_{min} = E_q[\min_{x_i \in X} D(x_i, q)]$) [21]:

$$C_r = \frac{D_{mean}}{D_{min}}$$

Intuitively, C_r captures the separability of the nearest neighbor from the bulk of the dataset. A high C_r value indicates the nearest neighbor is distinctly closer, suggesting an easier search. Conversely, as C_r approaches 1, the nearest neighbor becomes almost indistinguishable, signifying high difficulty [21]. He et al. linked C_r to Locality Sensitive Hashing (LSH) complexity [21]. The concept was adapted to Local Relative Contrast (LRC) for individual queries, typically defined for query q and k as the ratio between the average distance from q to all dataset points (d_{mean}) and the distance to its actual k-th nearest neighbor ($dist(q, x_k^*)$) [6], used in benchmarking ANN algorithms [6, 47].

5.1.2 Local Intrinsic Dimensionality (LID). Local Intrinsic Dimensionality (LID) [24] quantifies local complexity based on the distribution of distances from a query point, mirroring how an m-dimensional Euclidean ball's volume grows with r^m. Formally, for a continuous random distance variable X with CDF $F_X(r)$, LID at distance r (where $F_X(r) > 0$) is [24, 25]:

$$ID_X(r) \triangleq \lim_{\epsilon \to 0^+} \frac{\ln(F_X((1+\epsilon)r)/F_X(r))}{\ln(1+\epsilon)}$$

Under smoothness conditions, this is equivalent to [24, 25]:

$$ID_X(r) = \frac{r \cdot f_X(r)}{F_X(r)}$$

where $f_X(r)$ is the PDF. This established LID as quantifying the (lack of) local discriminative power of the distance measure [24, 25]. Higher LID implies a harder k-NN/ANN search task [26]. LID

estimation was advanced by connecting it to Extreme Value Theory (EVT) [3]. The Pickands-Balkema-de Haan theorem [8, 40] implies that distances exceeding a high threshold converge to a Generalized Pareto Distribution (GPD), and $\lim_{r\to 0} ID_X(r)$ relates to a GPD parameter. Amsaleg et al. [2, 3] derived several estimators, with the Maximum Likelihood Estimator (MLE), equivalent to the Hill estimator [22], being prominent:

$$\hat{ID}_X^* = -\left(\frac{1}{k}\sum_{i=1}^{k}\ln\frac{r_i}{r_k}\right)^{-1}$$

where r_i are distances to the k-nearest neighbors. This MLE is widely used due to its properties and empirical performance [3]. Aumüller et al. [6] empirically showed LID's utility for characterizing query hardness, but noted it might not fully capture variations across datasets. Wang et al. [47] later reported a correlation of only 0.599 between LID and Number of Distance Calculations (NDC) for HNSW on the Deep dataset, underscoring its limitations for graph-based indexes.

5.1.3 Query Expansion (QE). Query Expansion (QE) examines relative separation between nearby neighbors. Its roots are in Locality Sensitive Hashing (LSH) parameter tuning [14, 27, 42]. Ahle et al. [1] introduced expansion for adaptive LSH, measuring how much a search radius r can expand before doubling the enclosed points. For k-NN benchmarking [6], QE for query q and k is commonly defined as the ratio of the distance to its $(2k)$-th nearest neighbor (d_{2k}) to the distance to its k-th nearest neighbor (d_k) [6, 47]:

$$QE_q = \frac{d_{2k}}{d_k}$$

A QE_q value close to 1 suggests a dense region where neighbors are hard to distinguish (harder query). A large QE_q implies better separation (easier search) [47]. QE is related to (Generalized) Expansion Dimension [6] and has been used in benchmarking, though sometimes found less predictive than LID or RC [6].

5.1.4 ϵ-Hardness. ϵ-hardness, introduced by Zoumpatianos et al. [56] for data series indexing (building on concepts like [10]), reflects how the density near a query's true answer affects index pruning. It quantifies data points "competitively close" to the query's actual nearest neighbor ($x^{(1)}$). The set of ϵ-near neighbors ($\mathcal{N}^\epsilon(q)$) for query q is [56]:

$$\mathcal{N}^\epsilon(q) = \{x \in \mathcal{D} \mid DIST(x,q) \le (1+\epsilon)MINDIST(q)\}$$

where $MINDIST(q) = DIST(x^{(1)}, q)$. Then, ϵ-hardness, $\alpha^\epsilon(q)$, is [56]:

$$\alpha^\epsilon(q) = \frac{|\mathcal{N}^\epsilon(q)|}{|\mathcal{D}|}$$

High ϵ-hardness implies a harder query due to many points challenging index pruning [56].

5.1.5 Limitations for Graph-Based Search. While RC, LID, QE, and ϵ-hardness offer insights, their primary focus on vector data and distances [47] means they largely ignore the graph topology that dictates search in indexes like HNSW [36] or NSG [16]. Actual computational effort (e.g., NDC) is determined by the path navigated within this graph [47]. Consequently, points spatially close (influencing LID or ϵ-hardness) might be inaccessible during graph traversal if they lack appropriate edges or are far from the search path [47]. As Wang et al. [47] highlighted, and supported by empirical observations [6, 47], distance-based measures can lose predictive power for graph indexes. This gap underscores the need for "graph-native" measures.

5.2 Graph-Native Hardness Measures: Steiner-Hardness

The limitations of distribution-centric measures necessitate "graph-native" metrics directly analyzing graph connectivity. A prominent approach is **Steiner-Hardness** [47].

5.2.1 Minimum Effort (ME) Framework. Steiner-Hardness is grounded in the Minimum Effort (ME) framework. Its core idea is a theoretical lower bound on query effort based purely on graph connectivity, termed **Minimum Effort (ME)**. **ME@Acc** is the minimum vertices accessed within graph G to retrieve $Acc \times k$ of the true k-NNs (N_k), starting from an optimal entry point. Basic ME is idealized; practical greedy search often involves a fast initial phase finding one k-NN, then a costlier phase for remaining neighbors, with limited exploration [47]. ME is refined with constraints:

(1) The **Entry Point Constraint** models search effectively starting from one of the k-NNs discovered initially. To account for uncertainty in which k-NN is found first, it requires a minimum fraction (p) of true k-NNs to serve as viable starting points within the solution subgraph (Y) to reach target recall.
(2) The **Accessible Candidate Constraint** models greedy search's limited exploration by constraining accessed vertices (V_Y) to lie within a distance radius $(1+\delta)\times d_k$ of the query, using the critical point δ_0 (smallest δ for a valid solution).
(3) **Decision Cost Inclusion ($ME^P_{\delta_0}$@*Acc*-exhaustive)** accounts for costs (distance calculations) when evaluating neighbors. It modifies the objective to minimize total *cost* of subgraph Y, where node cost reflects its neighbor set size in G ($Cost(Y) = |\cup_{v\in V_Y} DS(v)|$), penalizing high out-degree nodes.

Ablation studies [47] validated these refinements, showing each is crucial for ME to accurately model query effort.

5.2.2 Steiner-Hardness Definition and Validation. For index independence [31], Steiner-Hardness is computed on a representative approximate Monotonic Relative Neighborhood Graph (MRNG) [16]. This approximation is built using a limited candidate pool (efC), mirroring practical index construction (e.g., HNSW's 'efConstruction') [47]. MRNG is chosen for structural similarities to common graph indexes, its deterministic pruning, and primary dependence on data distribution [47]. Formally, **Steiner-Hardness** for query q is $ME^P_{\delta_0}$@*Acc*-exhaustive on this approximate MRNG, given target recall *Acc* and probabilistic bound p. Computation of ME variants, linked to NP-hard Directed Steiner Tree (DST) [55] and related problems, uses heuristic solvers [44, 47].

Empirical evaluations [47] show Steiner-Hardness's significantly stronger correlation with actual query effort (NDC). On the Deep dataset with HNSW, Steiner-Hardness achieved 0.978 correlation with NDC, versus LID's 0.599. Across various datasets and indexes,

Steiner-Hardness averaged 0.75 correlation, substantially higher than ϵ-hardness (0.50), LID (0.42), QE (0.26), and RC (0.31) [47]. The underlying $ME^{P}_{\delta_0}$@*Acc*-exhaustive framework itself accurately described query effort on given graph indexes (average correlations e.g., 0.95 GIST, 0.98 Rand), significantly outperforming metrics like ϵ-effort [47].

5.3 Implications and Applications of Accurate Query Hardness Measurement

Accurate query hardness measures are critical. Previous benchmarking, often using random query sampling, tended to be dominated by "simple" queries, potentially leading to overly optimistic evaluations that don't reflect true robustness under stress [47]. Steiner-Hardness enables **unbiased query workloads** [47]. By sampling queries for a uniform hardness distribution across the spectrum, while ensuring dataset representativeness, researchers can create challenging yet fair testbeds. Empirical results [47] using such workloads showed benefits: they corrected hardness skew (e.g., reducing simple queries from 78-82% to 20%), led to new insights and revised index rankings (HNSW, NSG showed greater robustness on hard queries; KGraph was surprisingly competitive, suggesting pruning strategy improvements), and quantified cost disparities (hard queries 10-50x costlier than simple ones). Such workloads facilitate more comprehensive robustness testing, fairer index comparisons, and provide nuanced views to guide improvements in index construction and search algorithms [47].

5.4 Section Summary

The shift from purely distribution-centric analysis to graph-native measures like Steiner-Hardness [47] is vital for advancing graph-based ANN indexes. Traditional measures (LID, RC, QE, ϵ-hardness), while offering some insights, are fundamentally limited for graph indexes as they largely ignore the graph topology dictating search paths and effort. Steiner-Hardness, by directly modeling minimum traversal effort on a representative graph structure, offers a much more accurate reflection of graph search difficulties. Its strong empirical correlation with actual query effort and its ability to enable unbiased benchmarks are crucial. These benchmarks, in turn, facilitate more robust index evaluations, fairer comparisons, and ultimately drive progress in developing more resilient and efficient ANN solutions.

6 OPEN CHALLENGES AND FUTURE DIRECTIONS

Graph-based Approximate Nearest Neighbor (ANN) search has seen remarkable success, yet several open challenges and promising future directions remain.

6.1 General Graph-Based ANN Search

While graph-based ANN methods offer strong performance, optimizing the trade-off between search efficiency, indexing complexity, and scalability for billion-scale datasets is an ongoing need. Parameter tuning is challenging due to high hyperparameter sensitivity and the time-intensive nature of finding optimal configurations. The memory footprint of graph indexes, especially hierarchical structures like HNSW, can also limit applicability to very large datasets. Furthermore, developing efficient and scalable techniques for dynamic datasets with frequent updates is crucial. Future work should focus on: automated tuning methods to select optimal parameters without costly index rebuilding; techniques to reduce index memory footprint for greater scalability; more efficient and scalable methods for dynamic updates to adapt to changing data distributions; and investigating novel graph structures and search algorithms to further improve the fundamental trade-offs.

6.2 Query Hardness and Multi-Modal Search

Query hardness—the significant variability in search difficulty across queries—poses a substantial challenge, particularly in multi-modal search. This is exacerbated in cross-modal search by the inherent distribution gap between modalities, making some cross-modal queries considerably harder. Effectively addressing query hardness is paramount for consistent performance and reliability in both uni-modal and multi-modal ANN search. Future research should prioritize: developing new query hardness measures tailored for cross-modal search, considering its out-of-distribution nature and unique embedding space characteristics; investigating how query hardness influences various cross-modal ANN search methods; designing cross-modal ANN indexes and search algorithms robust to query hardness for consistent performance across diverse query difficulties; and exploring query hardness prediction to optimize cross-modal search, perhaps through adaptive resource allocation or the selection of different search strategies based on predicted difficulty.

6.3 Extending to Broader Multi-Modal Scenarios

Despite progress in multi-modal ANN search, opportunities exist to extend graph-based methods to more complex and realistic multi-modal scenarios. Future directions include: addressing scenarios with more than two modalities, such as combinations of text, image, audio, and video, which introduces new challenges in data representation, similarity measurement, and index design; exploring fine-grained multi-modal search focusing on intricate inter-modal relationships, for instance, finding images where a text-specified object has a particular pose; advancing beyond simple similarity search to tasks demanding multi-modal reasoning and understanding, including question answering, visual dialogue, and cross-modal information retrieval; and developing graph-based ANN methods to efficiently handle dynamic multi-modal data where data and inter-modal relationships constantly evolve.

7 CONCLUSION

This survey has provided a comprehensive overview of graph-based Approximate Nearest Neighbor (ANN) search, a crucial technique for enabling efficient similarity search in high-dimensional data. We have explored the fundamental principles of constructing and navigating proximity graphs, tracing the evolution of core graph-based algorithms from early Navigable Small World (NSW) graphs to advanced hierarchical structures like HNSW and refined graphs like NSG. The survey has highlighted the trade-offs between search efficiency, accuracy, and index complexity inherent in these methods.

Furthermore, we have examined the adaptation of graph-based ANN search to multi-modal data, addressing the challenges posed by out-of-distribution queries and showcasing solutions like RoarGraph that leverage query distribution information to enhance search performance. We have also delved into the problem of query hardness, analyzing how search difficulty varies across queries and discussing the limitations of traditional hardness measures for graph-based indexes, emphasizing the need for graph-native measures like Steiner-Hardness.

Despite the significant progress in graph-based ANN search, open challenges remain. These include the need for more efficient handling of dynamic datasets, automated parameter tuning, reduced memory footprint, and further exploration of graph structures to optimize search. Future research should also focus on developing robust methods for cross-modal search that are resilient to query hardness and can effectively handle more complex multi-modal scenarios.

In conclusion, graph-based ANN search continues to be a vibrant and evolving field, with ongoing research pushing the boundaries of efficiency, scalability, and applicability to increasingly complex data and search tasks.

REFERENCES

[1] Thomas D Ahle, Martin Aumüller, and Rasmus Pagh. 2017. Parameter-free locality sensitive hashing for spherical range reporting. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 239–256.

[2] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E Houle, Ken-ichi Kawarabayashi, and Michael Nett. 2015. Estimating local intrinsic dimensionality. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 29–38.

[3] Laurent Amsaleg, Oussama Chelly, Teddy Furon, Stéphane Girard, Michael E Houle, Ken-ichi Kawarabayashi, and Michael Nett. 2018. Extreme-value-theoretic estimation of local intrinsic dimensionality. *Data Mining and Knowledge Discovery* 32, 6 (2018), 1768–1805.

[4] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374.

[5] Martin Aumüller and Matteo Ceccarello. 2019. The role of local intrinsic dimensionality in benchmarking nearest neighbor search. In *Similarity Search and Applications: 12th International Conference, SISAP 2019, Newark, NJ, USA, October 2–4, 2019, Proceedings 12*. Springer, 113–127.

[6] Martin Aumüller and Matteo Ceccarello. 2021. The role of local dimensionality measures in benchmarking nearest neighbor search. *Information Systems* 101 (2021), 101807.

[7] Martin Aumüller and Matteo Ceccarello. 2023. Recent Approaches and Trends in Approximate Nearest Neighbor Search, with Remarks on Benchmarking. *IEEE Data Eng. Bull.* 46, 3 (2023), 89–105.

[8] August A Balkema and Laurens De Haan. 1974. Residual life time at great age. *The Annals of probability* 2, 5 (1974), 792–804.

[9] Philip A Bernstein, Siddharth Gollapudi, Suryansh Gupta, Ravishankar Krishnaswamy, Sepideh Mahabadi, Sandeep Silwal, Gopal R Srinivasa, Varun Suriyanarayana, Jakub Tarnawski, Haiyang Xu, et al. [n.d.]. Graph-based algorithms for nearest neighbor search with multiple filters. ([n. d.]).

[10] Kevin Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. 1999. When is "nearest neighbor" meaningful?. In *Database Theory—ICDT'99: 7th International Conference Jerusalem, Israel, January 10–12, 1999 Proceedings 7*. Springer, 217–235.

[11] Edgar Chávez, Gonzalo Navarro, Ricardo Baeza-Yates, and José Luis Marroquín. 2001. Searching in metric spaces. *ACM Comput. Surv.* 33, 3 (Sept. 2001), 273–321. https://doi.org/10.1145/502807.502808

[12] Meng Chen, Kai Zhang, Zhenying He, Yinan Jing, and X. Sean Wang. 2024. RoarGraph: A Projected Bipartite Graph for Efficient Cross-Modal Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 17, 11 (July 2024), 2735–2749. https://doi.org/10.14778/3681954.3681959

[13] Magdalen Dobson, Zheqi Shen, Guy E Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. 2023. Scaling graph-based anns algorithms to billion-size datasets: A comparative analysis. *arXiv preprint arXiv:2305.04359* (2023).

[14] Wei Dong, Zhe Wang, William Josephson, Moses Charikar, and Kai Li. 2008. Modeling LSH for performance tuning. In *Proceedings of the 17th ACM conference on Information and knowledge management*. 669–678.

[15] Damien François, Vincent Wertz, and Michel Verleysen. 2007. The concentration of fractional distances. *IEEE Transactions on Knowledge and Data Engineering* 19, 7 (2007), 873–886.

[16] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2017. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143* (2017).

[17] Ali Ganbarov, Jicheng Yuan, Anh Le-Tuan, Manfred Hauswirth, and Danh Le-Phuoc. 2024. Experimental comparison of graph-based approximate nearest neighbor search algorithms on edge devices. *arXiv preprint arXiv:2411.14006* (2024).

[18] Jianyang Gao and Cheng Long. 2023. High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–27.

[19] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, et al. 2023. Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. In *Proceedings of the ACM Web Conference 2023*. 3406–3416.

[20] Zhen Gong, Xin Wu, Lei Chen, Zhenzhe Zheng, Shengjie Wang, Anran Xu, Chong Wang, and Fan Wu. 2023. Full index deep retrieval: End-to-end user and item structures for cold-start and long-tail item recommendation. In *Proceedings of the 17th ACM Conference on Recommender Systems*. 47–57.

[21] Junfeng He, Sanjiv Kumar, and Shih-Fu Chang. 2012. On the difficulty of nearest neighbor search. *arXiv preprint arXiv:1206.6411* (2012).

[22] Bruce M Hill. 1975. A simple general approach to inference about the tail of a distribution. *The annals of statistics* (1975), 1163–1174.

[23] Kohei Hirata, Daichi Amagata, Sumio Fujita, and Takahiro Hara. 2022. Solving diversity-aware maximum inner product search efficiently and effectively. In *Proceedings of the 16th ACM Conference on Recommender Systems*. 198–207.

[24] Michael E. Houle. 2013. Dimensionality, Discriminability, Density and Distance Distributions. In *2013 IEEE 13th International Conference on Data Mining Workshops*. 468–473. https://doi.org/10.1109/ICDMW.2013.139

[25] Michael E Houle. 2017. Local intrinsic dimensionality I: an extreme-value-theoretic foundation for similarity applications. In *Similarity Search and Applications: 10th International Conference, SISAP 2017, Munich, Germany, October 4-6, 2017, Proceedings 10*. Springer, 64–79.

[26] Michael E Houle, Erich Schubert, and Arthur Zimek. 2018. On the correlation between local intrinsic dimensionality and outlierness. In *Similarity Search and Applications: 11th International Conference, SISAP 2018, Lima, Peru, October 7–9, 2018, Proceedings 11*. Springer, 177–191.

[27] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. *Proceedings of the Annual ACM Symposium on Theory of Computing* 30 (1998), 604–613. https://doi.org/10.1145/276698.276876

[28] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in neural information processing Systems* 32 (2019).

[29] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128.

[30] Conglong Li, Minjia Zhang, David G Andersen, and Yuxiong He. 2020. Improving approximate nearest neighbor search through learned adaptive early termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2539–2554.

[31] Peng-Cheng Lin and Wan-Lei Zhao. 2019. Graph based nearest neighbor search: Promises and failures. *arXiv preprint arXiv:1904.02077* (2019).

[32] Jun Liu, Zhenhua Zhu, Jingbo Hu, Hanbo Sun, Li Liu, Lingzhi Liu, Guohao Dai, Huazhong Yang, and Yu Wang. 2022. Optimizing graph-based approximate nearest neighbor search: Stronger and smarter. In *2022 23rd IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 179–184.

[33] Malte Ludewig, Iman Kamehkhosh, Nick Landia, and Dietmar Jannach. 2018. Effective nearest-neighbor music recommendations. In *Proceedings of the ACM Recommender Systems Challenge 2018*. 1–6.

[34] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68.

[35] Yury A. Malkov and Dmitry A. Yashunin. 2016. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *arXiv preprint arXiv:1603.09320* (2016). https://doi.org/10.48550/arXiv.1603.09320

[36] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* 42, 4 (April 2020), 824–836. https://doi.org/10.1109/TPAMI.2018.2889473

[37] Magdalen Dobson Manohar, Taekseung Kim, and Guy E Blelloch. 2025. Range Retrieval with Graph-Based Indices. *arXiv preprint arXiv:2502.13245* (2025).

[38] Magdalen Dobson Manohar, Zheqi Shen, Guy Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. 2024. Parlayann: Scalable and deterministic parallel graph-based approximate nearest neighbor search algorithms. In *Proceedings of the 29th ACM SIGPLAN Annual Symposium on Principles and Practice of Parallel Programming*. 270–285.

[39] Yutaro Oguri and Yusuke Matsui. 2024. Theoretical and Empirical Analysis of Adaptive Entry Point Selection for Graph-based Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2402.04713* (2024).

[40] James Pickands III. 1975. Statistical inference using extreme order statistics. *the Annals of Statistics* (1975), 119–131.

[41] Alexander Ponomarenko, Yury Malkov, Andrey Logvinov, and Vladimir Krylov. 2011. Approximate nearest neighbor search small world approach. In *International Conference on Information and Communication Technologies & Applications*, Vol. 17.

[42] Malcolm Slaney, Yury Lifshits, and Junfeng He. 2012. Optimal parameters for locality-sensitive hashing. *Proc. IEEE* 100, 9 (2012), 2604–2623.

[43] Alibaba Search Team. 2023. *Billion-Scale Vector Retrieval System in Taobao Search*. Technical Report. Alibaba Group. Internal Technical Report.

[44] Saskia van der Hoeven. 2023. Efficient solution methods for the directed Steiner tree problem. (2023).

[45] Mengzhao Wang, Haotian Wu, Xiangyu Ke, Yunjun Gao, Yifan Zhu, and Wenchao Zhou. 2025. Accelerating Graph Indexing for ANNS on Modern CPUs. *arXiv preprint arXiv:2502.18113* (2025).

[46] Mengzhao Wang, Xiaoyang Xu, Qiang Yue, and Yuxiang Chen. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. In *Proceedings of the VLDB Endowment*, Vol. 14. 1964–1977.

[47] Zeyu Wang, Qitong Wang, Xiaoxing Cheng, Peng Wang, Themis Palpanas, and Wei Wang. 2024. Steiner-hardness: A query hardness measure for graph-based ann indexes. *Proceedings of the VLDB Endowment* 17, 13 (2024), 4668–4682.

[48] Zeyu Wang, Haoran Xiong, Zhenying He, Peng Wang, et al. 2024. Distance comparison operators for approximate nearest neighbor search: Exploration and benchmark. *arXiv preprint arXiv:2403.13491* (2024).

[49] Ming Yang, Yuzheng Cai, and Weiguo Zheng. 2024. CSPG: Crossing Sparse Proximity Graphs for Approximate Nearest Neighbor Search. *Advances in Neural Information Processing Systems* 37 (2024), 103076–103100.

[50] Ziqi Yin, Jianyang Gao, Pasquale Balsebre, Gao Cong, and Cheng Long. 2025. DEG: Efficient Hybrid Vector Search Using the Dynamic Edge Navigation Graph. *Proceedings of the ACM on Management of Data* 3, 1 (2025), 1–28.

[51] Liu Yingfan, Cheng Hong, and Cui Jiangtao. 2021. Revisiting k-Nearest Neighbor Graph Construction on High-Dimensional Data : Experiments and Analyses. arXiv:2112.02234 [cs.DS] https://arxiv.org/abs/2112.02234

[52] Hsiang-Fu Yu. 2023. More-efficient approximate nearest-neighbor search. https://www.amazon.science/blog/more-efficient-approximate-nearest-neighbor-search

[53] Xiaoyao Zhong, Haotian Li, Jiabao Jin, Mingyu Yang, Deming Chu, Xiangyu Wang, Zhitao Shen, Wei Jia, George Gu, Yi Xie, et al. 2025. VSAG: An Optimized Search Framework for Graph-based Approximate Nearest Neighbor Search. *arXiv preprint arXiv:2503.17911* (2025).

[54] Shuyan Zhou, Uri Alon, Frank F Xu, Zhiruo Wang, Zhengbao Jiang, and Graham Neubig. 2022. Docprompting: Generating code by retrieving the docs. *arXiv preprint arXiv: 2207.05987* (2022).

[55] Leonid Zosin and Samir Khuller. 2002. On directed Steiner trees. In *SODA*, Vol. 2. 59–63.

[56] Kostas Zoumpatianos, Yin Lou, Ioana Ileana, Themis Palpanas, and Johannes Gehrke. 2018. Generating data series query workloads. *The VLDB Journal* 27 (2018), 823–846.