Navigating Labels and Vectors: A Unified Approach to Filtered Approximate Nearest Neighbor Search

YUZHENG CAI, Fudan University, China
JIAYANG SHI, Fudan University, China
YIZHUO CHEN, Fudan University, China
WEIGUO ZHENG, Fudan University, China

Given a query vector, approximate nearest neighbor search (ANNS) aims to retrieve similar vectors from a set of high-dimensional base vectors. However, many real-world applications jointly query both vector data and structured data, imposing label constraints such as attributes and keywords on the search, known as filtered ANNS. Effectively incorporating filtering conditions with vector similarity presents significant challenges, including index for dynamically filtered search space, agnostic query labels, computational overhead for label-irrelevant vectors, and potential inadequacy in returning results. To tackle these challenges, we introduce a novel approach called the *Label Navigating Graph*, which encodes the containment relationships of label sets for all vectors. Built upon graph-based ANNS methods, we develop a general framework termed *Unified Navigating Graph (UNG)* to bridge the gap between label set containment and vector proximity relations. *UNG* offers several advantages, including *versatility* in supporting any query label size and specificity, *fidelity* in exclusively searching filtered vectors, *completeness* in providing sufficient answers, and *adaptability* in integration with most graph-based ANNS algorithms. Extensive experiments on real datasets demonstrate that the proposed framework outperforms all baselines, achieving 10× speedups at the same accuracy.

CCS Concepts: • **Information systems → Specialized information retrieval**.

Additional Key Words and Phrases: Label navigating graph, filtered approximate nearest neighbor search

ACM Reference Format:
Yuzheng Cai, Jiayang Shi, Yizhuo Chen, and Weiguo Zheng. 2024. Navigating Labels and Vectors: A Unified Approach to Filtered Approximate Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 6 (SIGMOD), Article 246 (December 2024), 27 pages. https://doi.org/10.1145/3698822

1 Introduction

To handle the explosive growth and complexity of unstructured data, approximate nearest neighbor search (ANNS) has become a pivotal technique, which retrieves similar vectors (i.e., items) from a set of high-dimensional base vectors semantically representing an unstructured dataset [8, 23, 45]. It has been widely used in various real-world applications, such as recommendation systems [20, 22, 29], search engines [17, 24], retrieval-augmented generation for large language models [12, 53], and so on. To balance between efficiency and quality for ANNS queries, there are a huge number of studies [6, 7, 16, 19, 25, 26, 28, 35, 41, 55], and many vector databases [37, 43, 47] have been developed for production.

Authors' Contact Information: Yuzheng Cai, yuzhengcai21@m.fudan.edu.cn, Fudan University, Shanghai, China; Jiayang Shi, jiayangshi22@m.fudan.edu.cn, Fudan University, Shanghai, China; Yizhuo Chen, yizhuochen22@m.fudan.edu.cn, Fudan University, Shanghai, China; Weiguo Zheng, zhengweiguo@fudan.edu.cn, Fudan University, Shanghai, China.

ACM 2836-6573/2024/12-ART246
https://doi.org/10.1145/3698822

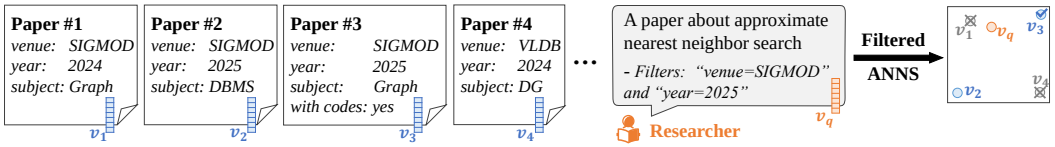


Fig. 1. A motivating example.

1.1 Motivation

Traditional ANNS systems focus solely on vector similarity, falling short in scenarios where attribute filtering is critical. To better serve complex real-world information retrieval needs, Filtered Approximate Nearest Neighbor Search (Filtered ANNS) has become a critical functionality, which extends traditional ANNS by allowing base vectors to satisfy attribute-based constraints [14, 15, 34, 44, 49]. Specifically, each vector v_i is associated with a label set f_i and satisfies the constraint if f_i covers the query label set f_q (i.e., $f_q \subseteq f_i$). In e-commerce scenarios, users can specify attributes such as brand and color, while still leveraging the power of ANNS to find products similar to their queries [14, 15]. For search engines, results can be filtered by domains or user privileges [14]. In academic scenarios, researchers may seek papers that are not only similar to a given topic, but also meet specific criteria such as publication year or conference venue [44]. Let us consider the following example.

Example 1.1. Figure 1 presents a set of papers with attributes, including the attributes venue, year, subject, and with codes. For example, the subject can take values such as "Graph", "DBMS" (Database Management System), and "DG" (Data Governance). Each paper #i also has a semantic embedding (i.e., high-dimension vector) v_i. Each value of an attribute is regarded as a label, and finding a SIGMOD'25 paper closely related to the researcher's requirement results in the query labels "venue=SIGMOD" and "year=2025". For the query embedding v_q representing the user requirement, filtered ANNS provides paper #3 as the top-1 answer, since v_3 is the nearest vector to v_q among those that cover all query labels.

1.2 Limitations of the Existing Methods

The traditional ANNS can be fledged with a variety of techniques: space partition [3, 4, 7, 16], hash transformation [1, 11, 13, 19], and graph search [26, 28, 32, 41]. However, as the labels of the query vector are agnostic in advance for filtered ANNS queries, it remains an open problem to build an effective index for these filtered vectors that are dynamically determined at runtime. Beyond that, it is challenging to balance efficiency and answer quality, and the prior algorithms struggle to overcome the following obstacles.

Versatility. In applications, the sizes and combinations of versatile query labels f_q can vary significantly. However, certain algorithms such as *NHQ* [44] and *CAPS* [15] assume that each base vector and query vector have the same number of attributes. For example, if papers in Figure 1 have attributes venue, subject, and year, then only query label sets with all these attributes are allowed, such as f_q = {"venue=SIGMOD", "subject=Graph", "year=2025"}. Thus, the query label sets f_q are required to have a fixed size, limiting their applicability to such particular queries. Furthermore, though algorithms such as *FilteredVamana* [14] and *StitchedVamana* [14] can support any size of f_q, they are tailored for the special case of $|f_q| = 1$, resulting in inferior performance when $|f_q| > 1$.

Fidelity. As the query labels strictly define a search range, to optimize efficiency and avoid unnecessary computations, an ideal solution is to conduct a fidelitous search only on those filtered vectors. One intuitive approach is to list all possible combinations of query labels and then build an

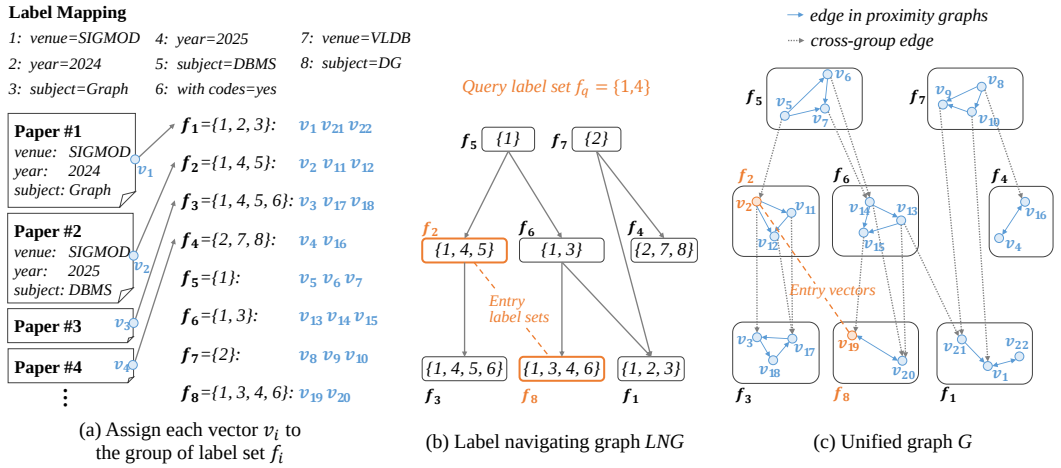


Fig. 2. Overview of the *Unified Navigating Graph* (*UNG*) framework.

ANNS index for each one [50]. But when the number of labels $|\mathcal{L}|$ is large, creating $2^{|\mathcal{L}|}$ separated indices becomes impractical [14, 44], resulting in duplication of base vectors that satisfy multiple query label constraints. Thus, to balance query efficiency and indexing costs, many state-of-the-art algorithms adopt a *filter-during-search* paradigm [14, 34, 44, 49], trying to approach those filtered vectors in a heuristic way. However, some label-irrelevant vectors may be still involved, which can potentially degrade query efficiency. In practice, how to ensure search fidelity still remains an open problem for filtered ANNS.

Completeness. In real-world scenarios, a filtered ANNS algorithm is expected to return a complete set of k filtered vectors, unless the dataset has fewer than k vectors satisfying the filter condition. However, such property is not guaranteed in many existing graph-based filtered ANNS approaches, including *NHQ* [44], *HQANN* [49], *FilteredVamana* [14], *StitchedVamana* [14] and *ACORN* [34]. This is because each node does not ensure connecting all the vectors that cover the query labels in their proximity graphs. The search is terminated once no neighbor satisfies the filter condition, especially for those query label sets f_q with low specificity (the fraction of vectors whose label set covers f_q). The users would be frustrated if they got only a small set of vectors (fewer than k) or an empty result set even if there were more feasible answers. To the best of our knowledge, the completeness of returned answers has not been addressed in the previous studies despite its importance.

1.3 Our Approach and Contributions

To optimize the efficiency of filtered ANNS without sacrificing accuracy, it is necessary to avoid the exploration of the label sets that fail to cover the query labels. Following the observation that *if a label set f covers the query label set f_q, any superset of f also covers f_q*, we introduce a general framework named *Unified Navigating Graph* (*UNG*). By utilizing the containment relationships among label sets, *UNG* enables fast retrieval of the dynamically filtered search space (i.e., all the vectors whose label sets cover f_q) where the advantages of the ANNS methods can be exerted.

As shown in Figure 2(a), *UNG* first partitions the dataset according to the label set f_i associated with each vector v_i, such that the vectors sharing the identical label set fall into the same group. Then, *UNG* builds a directed acyclic graph (DAG) named *Label Navigating Graph* (*LNG*, see Definition 3.3) in Figure 2(b) to encode the label set containment relationships, where an edge from label set f

Table 1. Properties of recent filtered ANNS methods.

	Versatility	Fidelity	Completeness	Adaptability
NHQ [44]	×	×	×	✓
CAPS [15]	×	×	✓	×
FilteredVamana [14]	✓	×	×	×
StitchedVamana [14]	✓	×	×	×
ACORN-γ (ACORN-1) [34]	✓	×	×	✓
***UNG* (ours)**	✓	✓	✓	✓

to f' indicates that f' is the *Minimum Superset* (Definition 3.2) of f. To enable efficient ANNS, as shown in Figure 2(c), for all vectors within the group of each label set, *UNG* can integrate existing graph-based ANNS approaches to build a proximity graph. Then, *Cross-group Edges* (Definition 3.7) are added following the direction of edges in *LNG*, which bridges the gap between vectors from different groups. It constitutes a *Unified Navigating Graph G* combining both the *LNG* and proximity graphs. Given a query vector v_q with label set f_q, by retrieving the *Entry Label Sets* (Definition 3.5) which directly cover all labels in f_q, a greedy search starting from the entry vectors with such entry label sets delivers the final nearest vectors for v_q.

Based on the entry label sets in *LNG*, the proposed *UNG* framework not only supports any combinations of query labels, but also works efficiently for any specificity, as we always perform graph-based ANNS within the exact filtered search space, ensuring the *versatility* for filtered ANNS. Moreover, once we start a greedy search from entry vectors, the cross-group edges built from label containment relations in *LNG* help to avoid label-irrelevant vectors, resulting in the merit of *fidelity*. Furthermore, since cross-group edges are required to exist between any two groups having edges in *LNG*, usually we can access most vectors satisfying the filter condition, and the *completeness* of returned answers is guaranteed by Theorem 5.4. Finally, as the proposed *UNG* framework can utilize many graph-based ANNS algorithms to build underlying indices, it also shows good *adaptability* to existing or custom-optimized proximity graphs. Table 1 compares the characteristics of our framework with recent filtered ANNS approaches.

Contributions. Our contributions are summarized as follows.

- We introduce the novel concept of the *Label Navigating Graph*, which serves as a dynamic guiding mechanism to strictly constrain the ANNS exploration within the filtered search space by leveraging the *Entry Label Sets* retrieved for the query labels.
- We propose a general *UNG* framework tailored for filtered ANNS, enabling seamless integration of various graph-based ANNS algorithms. It guarantees four fundamental properties: *versatility*, *fidelity*, *completeness*, and *adaptability*.
- By using Vamana as the underlying graph index, we conduct extensive experiments on 10 real-world datasets to compare the *UNG* framework with state-of-the-art filtered ANNS algorithms. In various scenarios, *UNG* exhibits the best performance in the trade-off between answer quality and query efficiency.

2 Problem Definition

Table 2 lists the frequently-used terms throughout the paper. In real-world scenarios, an attribute is a tag for an item (i.e., vector) that has multiple categorical values [14, 15, 44]. In Figure 1, a research paper may have attributes of "venue", "subject", and "year". For each vector, its attributes can be transformed into a set of labels by considering each categorical value of an attribute as a label. Thus, a paper may have a label set of {"venue=SIGMOD", "subject=Graph", "year=2025"}.

Table 2. Summary of notations.

Notation	Description
v_i (v_q)	a d-dimensional base (query) vector
f_i (f_q)	label set $\{l_1, l_2, \cdots\}$ of a base (query) vector
$\mathcal{D}$	a base dataset with $\lvert\mathcal{D}\rvert$ elements (v_i, f_i)
V	all vectors $\{v_1, v_2, \cdots, v_{\lvert V\rvert}\}$ in $\mathcal{D}$
V_f	all vectors associated with label set f
V_q	filtered vectors $\{v_i \in V \mid f_q \subseteq f_i\}$
$\mathcal{L}$	a finite universe of all possible labels, i.e., $\cup_{i=1}^{\lvert\mathcal{D}\rvert} f_i$
F	all possible label sets, i.e., $\cup_{i=1}^{\lvert\mathcal{D}\rvert}\{f_i\}$
F_q	filtered label sets $\{f_i \in F \mid f_q \subseteq f_i\}$
$\hat{F}_q$	entry label sets for query label set f_q
$\Delta(v_i, v_j)$	distance between two vectors $v_i, v_j \in \mathbb{R}^d$
S ($\tilde{S}$)	the exact (approximate) answer set for filtered ANNS
$G_f = (V_f, E_f)$	the proximity graph built on vectors V_f
$G = (V, E)$	the unified navigating graph built on all vectors V

Next, we formally define the task of *Filtered Approximate Nearest Neighbor Search* studied in this paper. A base dataset with $\mathcal{D} = \{(v_1, f_1), (v_2, f_2), \cdots, (v_{\lvert\mathcal{D}\rvert}, f_{\lvert\mathcal{D}\rvert})\}$ consists of $\lvert\mathcal{D}\rvert$ high-dimensional base vectors $V = \{v_1, v_2, \cdots, v_{\lvert V\rvert}\}$, in which $\lvert V\rvert = \lvert\mathcal{D}\rvert$. Each vector v_i is associated with a set of categorical labels $f_i = \{l_1, l_2, \cdots, l_{\lvert f_i\rvert}\}$, where $\lvert f_i\rvert$ denotes the number of labels assigned to vector v_i. Let $\mathcal{L} = \cup_{i=1}^{\lvert\mathcal{D}\rvert} f_i$ represent a finite universe of all possible labels, i.e., there are $\lvert\mathcal{L}\rvert$ distinct labels in the dataset. For a label $l \in \mathcal{L}$, its specificity $spec(l)$ is defined as the portion of vectors associated with label l, i.e., $spec(l) = \frac{\lvert\{(v_i, f_i) \in \mathcal{D} \mid l \in f_i\}\rvert}{\lvert\mathcal{D}\rvert}$. For a query vector v_q with label set $f_q \subseteq \mathcal{L}$, the task of *Filtered Nearest Neighbor Search* is to find the nearest base vectors associated with label sets that cover all labels in f_q. Denote the filtered vectors $V_q \subseteq V$ s.t. $v_i \in V_q$ iff $f_q \subseteq f_i$ ($1 \le i \le \lvert\mathcal{D}\rvert$). The distance between any two vectors $v_i, v_j \in \mathbb{R}^d$ is denoted as $\Delta(v_i, v_j) \in \mathbb{R}$.

Definition 2.1 (Filtered Nearest Neighbor Search, filtered NNS for short). Given a base dataset $\mathcal{D}$, the NNS task is to find a filtered subset of vectors $S \subseteq V_q$ of size k, s.t. for any $v_i \in V_q \backslash S$ and $v_j \in S$, $\Delta(v_q, v_j) \le \Delta(v_q, v_i)$, where k is a user-specified parameter.

Example 2.2. For the base dataset in Figure 1, the researcher asks for the top-k nearest base vectors with query label set $f_q = \{$"venue=SIGMOD", "year=2025"$\}$, where $k = 1$. Since $f_q \subset f_3$ and v_3 is the nearest one in V_q, paper #3 is returned as the result.

Generally, producing the exact answer S for the filtered NNS problem requires significant time. Aligned with prior works [14, 15, 34, 44], this paper targets to provide an approximate answer $\tilde{S}$.

Problem Statement. (Filtered Approximate Nearest Neighbor Search, filtered ANNS for short). With the same input as filtered NNS (Definition 2.1), the task is to find a subset $\tilde{S} \subseteq V_q$ that approximates the exact answer S for filtered NNS, maximizing both the answer quality (i.e., the size of $\tilde{S} \cap S$) and query efficiency.

3 Unified Navigating Graph

In this section, to efficiently obtain the filtered search space for ANNS, we introduce the *label navigating graph* built by integrating the relationships of label set containment. Then, we present a general framework named *Unified Navigating Graph* (*UNG*), combining the label navigating graph and proximity graphs to balance efficiency and quality for filtered ANNS.

3.1 Filtered Search Space

For the query label set $f_q = \{l_1, l_2, \cdots, l_{|f_q|}\}$, the filtered vectors V_q satisfy that $v_i \in V_q$ iff $f_q \subseteq f_i$. To obtain V_q, a brute-force solution is to check whether label set $f_q \subseteq f_i$ for each base vector $v_i \in V$, which results in a significantly large filtering time. To avoid iterating all the vectors, we can pre-compute a vector set for each possible label set in the dataset. Specifically, we denote all possible label sets as $F = \cup_{i=1}^{|\mathcal{D}|}\{f_i\}$. For each $f \in F$, during indexing phase, we compute the corresponding group of vectors $V_f = \{v_i \in V \mid f_i = f\}$. Thus, for the query label set f_q, we can iterate each $f \in F$ and check whether $f_q \subseteq f$ to obtain the filtered search space $V_q = \cup_{f_q \subseteq f} V_f$.

Example 3.1. Figure 2(a) presents 8 possible label sets F on the example dataset. Given the query label set $f_q = \{1, 4\}$, the filtered search space is $V_q = V_{f_2} \cup V_{f_3} \cup V_{f_8} = \{v_2, v_3, v_{11}, v_{12}, v_{17}, v_{18}, v_{19}, v_{20}\}$.

3.2 Label Navigating Graph

However, examining every possible label set $f \in F$ remains expensive. To obtain the filtered label sets defined as $F_q = \{f \in F \mid f_q \subseteq f\}$, we follow the intuition that *if a label set f covers all query labels f_q, any superset of f also covers f_q*. It motivates us to consider containment relationships between label sets in F, which allows us to derive the filtered search space V_q only using some entry label sets.

Definition 3.2 (Minimum Superset). For a label set f, label set $f' \in F$ is a minimum superset of f iff (1) $f \subset f'$ and (2) $\nexists\, f'' \in F$ s.t. $f \subset f'' \subset f'$.

Naturally, all the minimum superset relations among all label sets in F constitute a directed acyclic graph (DAG), referred to as the *Label Navigating Graph*.

Definition 3.3 (Label Navigating Graph, LNG for short). *LNG* is a DAG, where each vertex is a label set $f \in F$. A directed edge $e(f, f')$ from vertex f to another vertex f' exists in *LNG* iff f' is a minimum superset of f.

Example 3.4. Figure 2(b) shows the *LNG* built on all label sets F in Figure 2(a). For label sets $f_2 = \{1\}$, $f_3 = \{1, 4, 5\}$, and $f_8 = \{1, 4, 5, 6\}$ s.t. $f_2 \subset f_3 \subset f_8$, the edge from f_2 to f_8 cannot exist in *LNG*.

The label navigating graph *LNG* preserves the transitivity property of the superset relations. Specifically, a directed path in *LNG* from f to f'' consisting of edges $e(f, f')$ and $e(f', f'')$ implies that $f \subset f''$. With such property, we have the following observation.

Observation 3.1. *For any label set $f \in F$, all its supersets in F are its descendants in LNG.*

Definition 3.5 (Entry Label Sets). For query label set f_q, its entry label sets, denoted as $\hat{F}_q$, are defined as follows.

$$\hat{F}_q = \begin{cases} \{f_q\}, & \text{if } f_q \in F, \\ \text{all the minimum supersets of } f_q \text{ in } F, & \text{if } f_q \notin F. \end{cases}$$

Given the query label set f_q, we can first find its entry label sets $\hat{F}_q$. Then, without checking all label sets in F, the filtered label sets F_q can be derived from the *LNG* based on the retrieved $\hat{F}_q$.

Observation 3.2. *For the query label set f_q, the filtered label sets F_q are equal to the entry label set $\hat{F}_q$ and all their descendants in LNG.*

Example 3.6. As shown in Figure 2(b), given the query label set $f_q = \{1, 4\}$, the entry label sets $\hat{F}_q = \{f_2, f_8\}$. The filtered label sets $F_q = \{f_2, f_3, f_8\}$ are derived from $\hat{F}_q$.

3.3 Unifying Label Navigating Graph and Proximity Graphs

For each filtered label set $f \in F_q$, we can find the top-k nearest vectors in the corresponding group V_f. Merging the results for all filtered label sets delivers the final top-k vectors in the filtered search space $V_q = \cup_{f \in F_q} V_f$. Specifically, to find the top-k nearest vectors in each group V_f, we consider proximity-graph-based algorithms, as they have been proven to be practically efficient [2, 14, 23], where each vector usually has a degree bound d_{max}. It is feasible to build $|F|$ separate proximity graphs, since the groups V_f of different label sets $f \in F$ are disjoint. $|F|$ graphs result in a total size of $O(\sum_{f \in F} |V_f| d_{max}) = O(|\mathcal{D}| d_{max})$, which is similar to the single graph built upon all vectors for unfiltered ANNS.

However, when there are a large number of filtered label sets F_q, conducting ANNS searches on $|F_q|$ groups separately may still be inefficient. This is because for some groups where most vectors are far away from the query vector v_q, computing ANNS does not contribute to the final top-k results. Therefore, to dynamically avoid iterating over such groups during the search process, we propose the *Unified Navigating Graph* (*UNG*) framework described as follows.

Inspired by ANNS proximity graphs that approach the query vector greedily through graph traversal, we can also greedily explore a portion of groups that may lead to the final top-k nearest vectors. Specifically, to dynamically navigate the search towards such groups, we first build proximity graphs on vectors of each group V_f separately, and our general idea is to *add new edges to bridge the gaps between disparate groups*. Moreover, to avoid visiting any vectors that do not satisfy the filter condition, such new edges must always lead to vectors associated with the filtered label set in F_q. Recall that in *LNG*, if a label set $f \in F$ satisfies that $f_q \subseteq f$, then any descendant of f in *LNG* also belongs to F_q. Thus, we define the concept of *cross-group edges*.

Definition 3.7 (Cross-group Edge). An edge from vector v_i to vector v_j is a cross-group edge if there exists a directed edge from f_i to f_j in *LNG* (i.e., f_j is the minimum superset of f_i).

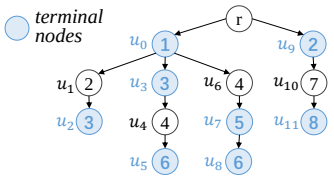
Additionally, we ensure that cross-group edges exist between any groups V_f and $V_{f'}$ if f' is an out-neighbor of f in *LNG*. Please refer to Section 5.2 for details on how to build such edges.

Example 3.8. In Figure 2(c), each group V_f has a proximity graph built on its vectors. We can add a cross-group edge from vector $v_2 \in V_{f_2}$ to vector $v_3 \in V_{f_3}$, following the edge from label set f_2 to f_3 in *LNG* (Figure 2(b)). For each edge from f to f' in *LNG*, there exists a corresponding cross-group edge from vector $v_i \in V_f$ to vector $v_j \in V_{f'}$.

Let $G_f = (V_f, E_f)$ denote the proximity graph built for group V_f, where E_f is the set of edges in G_f. Since we build cross-group edges following the edges in the label navigating graph, the final graph unifies both the label navigating graph and the proximity graph. This graph is called the *Unified Navigating Graph* $G = (V, E)$, where its edge set E consists of both the cross-group edges and the edges E_f within each proximity graph. The *Unified Navigating Graph* (*UNG*) framework offers the following properties.

Versatility. The *UNG* framework works for any size and combination of versatile query label sets f_q, since Observation 3.2 reveals that the filtered search space V_q can be derived from its entry label sets $\hat{F}_q$ (Definition 3.5). By traveling across different groups via cross-group edges, it naturally supports scenarios where the base label sets are required to be the supersets of query labels. When we only need those vectors with label sets that exactly match the query label set, the unified navigating graph G can also be used by disabling all cross-group edges during searching.

Adaptability. *UNG* is a general framework for filtered ANNS, in which most graph-based ANNS algorithms can be integrated seamlessly to build the proximity graph for each group V_f, such as *NSW* [25], *KGraph* [6], *NSG* [10], *Vamana* [41] and others.



(a) Trie tree T (b) Inverted lists $\mathbb{I}$

Fig. 3. An example trie tree T with the inverted lists $\mathbb{I}$.

4 Answering Filtered ANNS Queries

To answer a filtered ANNS query with query label set f_q, as shown in Figures 2(b)-(c), we first retrieve its entry label sets $\hat{F}_q$ through the label navigating graph *LNG* (in Section 4.1), and then search in the unified navigating graph G by starting from entry vectors associated with label set $f \in \hat{F}_q$ (in Section 4.2).

4.1 Retrieving the Entry Label Sets

As revealed in Observation 3.2, entry label sets $\hat{F}_q$ are essential for obtaining the filtered label sets F_q. To avoid the high cost of examining a large number of label sets, we use the well-known *Trie Tree* structure to efficiently retrieve $\hat{F}_q$. Without loss of generality, assume that all possible labels $\mathcal{L} = \{l_0, l_1, \cdots, l_{|\mathcal{L}|}\}$ are represented by integers $1, 2, \cdots, |\mathcal{L}|$ according to the decreasing order of label frequency $freq(l_j) = |\{f \in F \mid l_j \in f\}|$. We employ a trie tree T to check whether the query label set $f_q \in F$.

Definition 4.1 (Trie Tree). Given the label sets F for the base dataset with $\mathcal{D}$, its trie tree, denoted by T, is a tree structure, where each node u represents a label $l \in \mathcal{L}$, except for the root r that serves as the entrance. Any child of node u corresponds to a label larger than the one represented by u. And a node u is marked as a terminal node iff the path from r to u constitutes a label set $f \in F$.

Example 4.2. Figure 3(a) presents the trie tree T for all label sets F in Figure 2(a). The path from r to u_7 constitutes the label set $f_2 = \{1, 4, 5\}$, thus u_7 is a terminal node.

According to Definition 3.5, there are two cases for obtaining $\hat{F}_q$.
Case 1: $f_q \in F$. Using the trie tree T, lines 1-5 of Algorithm 1 determine whether $f_q \in F$. By Definition 3.5, it returns $\hat{F}_q = \{f_q\}$ if f_q can be represented by a path from root r to a terminal node u.
Case 2: $f_q \notin F$. In this case, $\hat{F}_q$ should consist of all the minimum supersets of f_q in F. In the trie tree T, any label set $f \in \hat{F}_q$ can be represented by a path p, which must include nodes covering all the labels in f_q. Thus, for each label $l \in \mathcal{L}$, we construct an inverted list $\mathbb{I}(l) = \{$all nodes in T corresponding to $l\}$ in the trie tree T. Since the largest label (i.e., integer) $l_{max} \in f_q$ has the smallest frequency, lines 8-10 of Algorithm 1 iterate over each node $u \in \mathbb{I}(l_{max})$ to check whether the path from root r to u includes all labels of f_q, using the function Examine(u). Traveling from all such nodes $u \in \mathbb{I}(l_{max})$ will lead to all supersets of f_q, i.e., the filtered label sets F_q.

Example 4.3. Figure 3 presents both the trie tree T and its inverted lists $\mathbb{I}$. Given the query label set $f_q = \{1, 4\}$, we extract the $\mathbb{I}(l_{max}) = \mathbb{I}(4) = \{u_4, u_6\}$. By checking the path from r to u_4 and u_6 respectively, we find that label sets $f_q \subset \{1, 3, 4\}$ and $f_q \subseteq \{1, 4\}$. By traveling from u_4 and u_6, we can obtain all other supersets of f_q, i.e., $\{1, 3, 4, 6\}$, $\{1, 4, 5\}$, and $\{1, 4, 5, 6\}$.

Since we only need the minimum supersets of f_q, it is unnecessary to search for all its supersets. For a path p' representing label set f', if it contains a sub-path p that ends with a terminal node

Algorithm 1 Retrieving entry label set

Require: Trie tree T with root r, and query label set $f_q \subseteq \mathcal{L}$
Ensure: Entry label sets $\hat{F}_q$
1: current node $c \leftarrow$ root r of T
2: **for** each label $l \in f_q$ in the increasing order **do**
3: **if** no child of c corresponds to l **then** break
4: $c \leftarrow$ child of c corresponds to l
5: **if** c is marked as a terminal node **then return** $\{f_q\}$
6: **return** GetMinimumSuperSets(f_q)

7: **function** GetMinimumSuperSets(f_q)
8: $\hat{F}_q^{(c)} \leftarrow \emptyset$, $l_{max} \leftarrow$ the largest label in f_q
9: **for** each node $u \in \mathbb{I}(l_{max})$ **do**
10: **if** Examine(u) **then** DFS(u, labels in path from r to u)
11: $\hat{F}_q \leftarrow$ all label sets with the smallest size in $\hat{F}_q^{(c)}$
12: **for** each $f \in \hat{F}_q^{(c)} \backslash \hat{F}_q$ with increasing label set size **do**
13: **if** $\nexists\, f' \in \hat{F}_q$ s.t. $f' \subset f$ **then** add f into $\hat{F}_q$
14: **return** $\hat{F}_q$

15: **function** Examine(u)
16: $c \leftarrow$ parent of u
17: **for** each label l in $f_q \backslash \{l_{max}\}$ under decreasing order **do**
18: **while** $c > l$ and c has parent **do**
19: $c \leftarrow$ parent of c
20: **if** $c \neq l$ or $c = r$ **then return** *False*
21: **return** *True*

22: **function** DFS(u, f)
23: **if** u is a terminal node **then** add f into $\hat{F}_q^{(c)}$
24: **else for** each child u' of u **do** DFS(u', $f \cup \{c\}$)

corresponding to label $l \geq l_{max}$, then f' is not a minimum superset of f_q, since the label set f represented by p satisfies $f_q \subset f \subset f'$. Thus, the depth-first search (DFS) in Algorithm 1 terminates when we encounter the first terminal node. All the label sets found by DFS constitute the candidates $\hat{F}_q^{(c)}$ for the entry label sets $\hat{F}_q$.

Example 4.4. For the query labels $f_q = \{1, 4\}$, Example 4.3 has located nodes u_4 and u_6 in the trie tree T. Searching from u_4 yields the label set $\{1, 3, 4, 6\}$. Starting from u_6, we visit terminal node u_7 and directly return $\{1, 4, 5\}$, without involving its superset $\{1, 4, 5, 6\}$.

To obtain entry label sets $\hat{F}_q$, line 11 first initializes $\hat{F}_q$ with all label sets f with the smallest size in $\hat{F}_q^{(c)}$, since $\nexists\, f' \in \hat{F}_q^{(c)}$ s.t. $f' \subset f$. Then, lines 12-13 attempt to add each candidate label set $f \in \hat{F}_q^{(c)}$ into $\hat{F}_q$. We process each label set $f \in \hat{F}_q^{(c)}$ and check whether it covers any existing entry label set $f' \in \hat{F}_q$. The increasing order of label set size ensures that $\hat{F}_q$ cannot contain two label sets f and f' s.t. $f' \subset f$.

Example 4.5. For the query labels $f_q = \{1, 4\}$, Example 4.4 delivers the candidate entry label sets $\{1, 4, 5\}$ and $\{1, 3, 4, 6\}$, which are actually equal to the entry label sets $\hat{F}_q$. However, when the query labels $f_q = \{3\}$, the candidates $\hat{F}_q^{(c)} = \{\{1, 2, 3\}, \{1, 3\}\} \neq \hat{F}_q$. By iterating through each of them in lines 12-13, only $\{1, 3\}$ is in $\hat{F}_q$.

Algorithm 2 Search on the unified navigating graph.

Require: Unified navigating graph G, query vector v_q with label set f_q
Ensure: Top-k filtered approximate nearest vectors $\tilde{S}$
1: $pq \leftarrow \emptyset$
2: **for** each entry label set $f \in \hat{F}_q$ obtained by Algorithm 1 **do**
3: $pq \leftarrow pq \cup$ random σ vectors in V_f
4: **while** pq has unexplored vectors **do**
5: pop the nearest unexplored v_i from pq, mark v_i explored
6: $pq \leftarrow pq \cup$ all out-neighbors of v_i in G
7: $pq \leftarrow$ top-w nearest vectors in pq w.r.t. v_q
8: **return** top-k nearest vectors w.r.t. v_q in pq

Retrieval time and space when $f_q \in F$. Lines 1-5 of Algorithm 1 require $O(|f_q|)$ time and space to return the entry label sets $\hat{F}_q$.
Retrieval time cost when $f_q \notin F$. Lines 8-10 require $O(|\mathbb{I}(l_{max})||\mathcal{L}|)$ time for function Examine(u). Additionally, performing DFS from different nodes $u \in \mathbb{I}(l_{max})$ explores different parts of the trie tree T, costing $O(|\hat{F}_q^{(c)}||\mathcal{L}|)$ time. Lines 12-13 require $O(|\hat{F}_q^{(c)}||\hat{F}_q||\mathcal{L}|)$ time to retrieve the final entry label set $\hat{F}_q$. Thus, the overall retrieval time complexity is $O\left(\left(|\mathbb{I}(l_{max})| + |\hat{F}_q^{(c)}||\hat{F}_q|\right)|\mathcal{L}|\right)$.
Retrieval space cost when $f_q \notin F$. It requires $O(|\mathcal{L}|)$ space for both functions Examine and DFS. Maintaining the candidate label sets $\hat{F}_q^{(c)}$ and final entry label sets $\hat{F}_q$ require $O(|\hat{F}_q^{(c)}|)$ space. Thus, the overall retrieval space complexity is $O(|\hat{F}_q^{(c)}| + |\mathcal{L}|)$.

Note that in real-world scenarios, the label frequencies often follow a power-law distribution [36]. If query label set f_q is composed of frequent labels, it is likely that $f_q \in F$ and it costs only $O(|f_q|)$ time and space. Conversely, when f_q contains infrequent labels and $f_q \notin F$, likely there are only a few label sets $f \in F$ satisfying $f_q \subset f$. Additionally, since l_{max} has the smallest frequency among all labels in f_q, practically both $|\mathbb{I}(l_{max})|$and $|\hat{F}_q| \leq |\hat{F}_q^{(c)}|$ are much smaller than $|F|$. Thus, the retrieving time and space are affordable in practice.

4.2 Search on Unified Navigating Graph

For the query label set f_q, the unified navigating graph G ensures that *if we search from vector v_i s.t. $f_q \subseteq f_i$, then for any vector v_j that v_i can reach in G, it also satisfies $f_q \subseteq f_j$*. Thus, we only need to start the search from the entry label sets $\hat{F}_q$ that cover query labels f_q, then any visited vectors will satisfy the filter condition. Specifically, for each entry label set $f \in \hat{F}_q$ found by Algorithm 1, Algorithm 2 randomly selects σ entry vectors and pushes them into the priority queue pq. To conduct the best-first search on the unified navigating graph G, at each iteration, we pop the nearest unexplored vector v_i from the pq, and then push all its out-neighbors into pq. In line with existing graph-based ANNS approaches, we set a maximum size limit $w > k$ for pq, and line 7 only preserves the top-w nearest vectors in pq. If all vectors in pq have been explored, we terminate the search and the top-k nearest vectors in pq are returned.

Example 4.6. As shown in Figure 2(c), for the query label set $f_q = \{1, 4\}$, its entry label sets are f_2 and f_8. When $\sigma = 1$, the priority queue can be initialized with vectors $v_2 \in V_{f_2}$ and $v_{19} \in V_{f_8}$.

As discussed in Section 3.3, the *UNG* framework ensures both *versatility* and *adaptability*. Additionally, Algorithm 2 further offers the following advantageous properties.

Fidelity. The entry label sets $\hat{F}_q$ always lead to all filtered label sets that cover f_q in the label navigating graph *LNG*, and the cross-group edges follow the connection in *LNG*. Thus, we always

Algorithm 3 Label navigating graph construction.

Require: All possible label sets F
Ensure: Trie tree T, inverted lists $\mathbb{I}$, label navigating graph *LNG*
1: initialize T with a root r, $\mathbb{I}(l) \leftarrow \emptyset$ for $l \in \mathcal{L}$
2: initialize *LNG* with $|F|$ vertices for all label sets in F
3: **for** each $f \in F$ in the decreasing order of label set size **do**
4: **for** each $f' \in$ GetMinimumSuperSets(f) **do**
5: add a directed edge from f to f' in *LNG*
6: InsertLabelSet(f, T); update $\mathbb{I}(l)$ for each $l \in f$
7: **return** T, $\mathbb{I}$, *LNG*
8: **function** InsertLabelSet(f, T)
9: current node $c \leftarrow r$
10: **for** each label $l \in f$ in the increasing order **do**
11: **if** no child of c corresponds to l **then**
12: create a new node for l, set it as a child of c
13: $c \leftarrow$ the child node of c corresponding to l
14: mark c as a terminal node

search within the filtered search space V_q. Once we start the search from those entry vectors, we will not need to check the label sets anymore.

Completeness. Generally, the proximity graph G_f built for each group V_f usually has good connectivity and a vector can reach many other vectors in V_f. Section 3.3 mentions that for any groups V_{f_i} and V_{f_j}, there must exist cross-group edges between them if an edge from f_i to f_j exists in *LNG*. Intuitively, by searching from vectors in the entry label sets, we can reach many vectors in the filtered search space V_q. By Theorem 5.4 discussed later, for any top-k query, Algorithm 2 always guarantees to return at least $\min\{k, |V_q|\}$ answers.

Overall time complexity. Note that the *UNG* is a general framework that can integrate many existing graph-based ANNS indices. When using the proximity graphs G_f built with a certain graph index, denote the number of vectors visited in Algorithm 2 as ρ. Recall that each vector $v_i \in \mathbb{R}^n$, and the priority queue pq can contain at most w vectors. Considering the worst time complexity of retrieving entry label sets in Section 4.1, the overall time complexity of Algorithm 2 is $O\left(\left(|\mathbb{I}(l_{max})| + |\hat{F}_q^{(c)}||\hat{F}_q|\right)|\mathcal{L}| + \rho(n + \log w)\right)$.

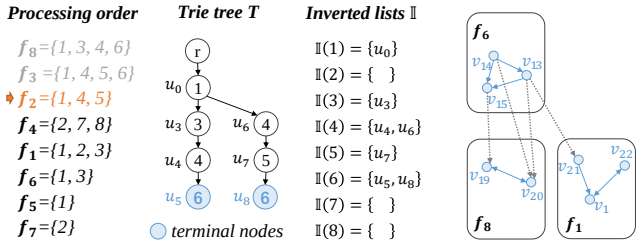
Overall space complexity. Considering the worst space complexity of retrieving entry label sets in Section 4.1, the overall space complexity of Algorithm 2 is $O\left(|\hat{F}_q^{(c)}| + |\mathcal{L}| + w\right)$.

5 Index construction

In this section, we begin by introducing the construction of the label navigating graph *LNG* in Section 5.1, then discuss how to build the unified navigating graph G in Section 5.2. Finally, Section 5.3 provides a brief introduction of how to manage data updates.

5.1 Constructing the Label Navigating Graph

Recall that the label navigating graph *LNG* consists of all minimum superset relations between label sets in F. In practice, we simultaneously construct the *LNG*, the trie tree T (Section 4.1), and the inverted lists $\mathbb{I}$ (Section 4.1). For each label set $f \in F$, Algorithm 3 extracts its minimum supersets to add edges to *LNG* in lines 4-5. To ensure that all minimum supersets can be extracted, we follow the decreasing order of label set size in line 3. Then, line 6 inserts f into T and updates the corresponding inverted lists $\mathbb{I}$.



(a) Building label navigating graph (b) Cross-group edges

Fig. 4. Illustrations for index construction.

Example 5.1. For all possible label sets F in Figure 2(a), assume that label sets f_3 and f_8 have been processed. Now we have the trie tree T and inverted lists $\mathbb{I}$ as illustrated in Figure 4(a). When processing f_2, its minimum supersets are f_3 and f_8, thus we add an edge from f_2 to f_3 and an edge from f_2 to f_8 in *LNG*. Then, we insert f_2 into T by marking node u_7 as a terminal node. Since we do not add any new node to T, the inverted lists $\mathbb{I}$ will not be updated.

Time and space complexity. Since there are $|F|$ distinct label sets and each has at most $|\mathcal{L}|$ labels, $O(|F||\mathcal{L}|)$ time and space are needed to construct the trie tree T. For each label $l \in \mathcal{L}$, as there are at most $|F|$ nodes corresponding to l in T, the inverted lists $\mathbb{I}$ for all labels consume $O(|F||\mathcal{L}|)$ time and space. As discussed in Section 4.1, obtaining the minimum supersets for each label set f_q consumes $O\left(\left(|\mathbb{I}(l_{max})| + |\hat{F}_q^{(c)}||\hat{F}_q|\right)|\mathcal{L}|\right)$ time and $O(|\hat{F}_q^{(c)}| + |\mathcal{L}|)$ space. Let d_{LNG} denote the average out-degree of label sets F in *LNG*, which is much smaller than $|F|$ in practice. Thus, building *LNG* requires $O(d_{LNG}|F|^2|\mathcal{L}|)$ time and $O(|F|(|\mathcal{L}| + d_{LNG}))$ space.

5.2 Constructing the Unified Navigating Graph

As discussed in Section 3.2, for vectors v_i and v_j in different groups (i.e., $f_i \neq f_j$), we consider building a directed edge from v_i to v_j if there exists a directed edge from f_i to f_j in *LNG*. However, it is infeasible to add edges between all pairs of v_i and v_j s.t. f_j is the minimum superset of f_i. Thus, to build the desired graph $G = (V, E)$, Algorithm 4 first utilizes graph-based ANNS approaches to build the proximity graph G_f for each group V_f. Then, by taking each vector v_i as the query vector in line 10, it conducts traditional ANNS to find the top-δ nearest vectors $\tilde{S}_{v_i}$ among vectors in $\cup_{f_j \text{ is the out-neighbor of } f_i \text{ in } LNG} V_{f_j}$. For each vector $v_j \in \tilde{S}_{v_i}$, we add a directed edge $e(v_i, v_j)$ from v_i to v_j in line 14.

Example 5.2. Figure 4(b) zooms in the unified navigating graph G in Figure 2(c), and label set f_6 has two out-neighbors f_8 and f_1 in the *LNG*. By taking $v_{13} \in V_{f_6}$ as the query vector, we can obtain its top-δ approximate nearest vectors $\tilde{S}_{v_{13},f_8}$ in group V_{f_8}, and $\tilde{S}_{v_{13},f_1}$ in V_{f_1}. When $\delta = 1$, $\tilde{S}_{v_{13}} = \{v_{20}\}$ by assuming that $\tilde{S}_{v_{13},f_8} = \{v_{20}\}$, $\tilde{S}_{v_{13},f_1} = \{v_{21}\}$, and $\Delta(v_{13}, v_{20}) < \Delta(v_{13}, v_{21})$. Thus, we add the edge $e(v_{13}, v_{20})$ to G. Similarly, we add the edges $e(v_{14}, v_{20})$ and $e(v_{15}, v_{19})$ by conducting unfiltered ANNS for v_{14} and v_{15}, respectively.

In Example 5.2, vectors in the group V_{f_6} always guide the search to V_{f_8} without visiting any vectors in V_{f_1}. To inherit all the connections of *LNG* for ensuring the connectivity of the unified navigating graph G, we add additional edges such as the edge $e(v_{13}, v_{21})$, where $v_{21} \in V_{f_1}$. We achieve this goal by ensuring that there are at least δ cross-group edges between any two groups V_f and $V_{f'}$, provided that f' is an out-neighbor of f in *LNG*. Such edges $E_{f,f'}$ can be collected in line 12, utilizing the top-δ nearest vectors $\tilde{S}_{v_i}$ for each $v_i \in V_f$ in line 10 of Algorithm 4.

Algorithm 4 Unified navigating graph construction.

Require: vectors V, and the label set f_i associated with each $v_i \in V$
Ensure: A unified navigating graph $G = (V, E)$
1: $F \leftarrow$ all possible label sets for vectors in V
2: initialize group V_f for each label set $f \in F$
3: build the label navigating graph *LNG* by Algorithm 3
4: build the proximity graph $G_f = (V_f, E_f)$ for each group V_f
5: $cnt_{f,f'} \leftarrow 0$ for each edge from f to f' in *LNG*, $E \leftarrow \cup_{f \in F} E_f$
6: **for** each $(v_i, f_i) \in \mathcal{D}$ **do**
7: $\quad f \leftarrow f_i$, $\tilde{S}_{v_i} \leftarrow \emptyset$
8: $\quad$ **for** each out-neighbor f' of f in *LNG* **do**
9: $\quad\quad \tilde{S}_{v_i,f'} \leftarrow$ top-δ nearest vectors in $V_{f'}$ w.r.t. v_i
10: $\quad\quad \tilde{S}_{v_i} \leftarrow$ top-δ nearest vectors in $\tilde{S}_{v_i} \cup \tilde{S}_{v_i,f'}$ w.r.t. v_i
11: $\quad\quad$ **if** $cnt_{f,f'} < \delta$ **then**
12: $\quad\quad\quad$ add $e(v_i, v_j)$ into E by picking a random $v_j \in \tilde{S}_{v_i,f'}$
13: $\quad\quad\quad cnt_{f,f'} \leftarrow cnt_{f,f'} + 1$
14: $\quad E \leftarrow E \cup \{e(v_i, v_j) \mid v_j \in \tilde{S}_{v_i}\}$
15: **return** $G = (V, E)$

Example 5.3. After finding the top-δ approximate nearest vectors $\tilde{S}_{v_{13},f_8}$ and $\tilde{S}_{v_{13},f_1}$ for $v_{13} \in V_{f_6}$, line 12 directly adds the cross-group edges $e(v_{13}, v_{20})$ and $e(v_{13}, v_{21})$ to the unified navigating graph G. When $\delta = 1$, for the other vectors in V_{f_6}, only edges $e(v_{14}, v_{20})$ and $e(v_{15}, v_{19})$ are added following the same process as in Example 5.2.

In practice, we set the parameters $\delta \leq d_{max}$, where d_{max} is the maximum degree used in building each proximity graph G_f. The size of G is given by $O(|V|(d_{max} + 2\delta)) = O(|V|d_{max})$, which is similar to the size of the single proximity graph built on all vectors V.

Parallel Implementations. In Algorithm 4, the $|F|$ proximity graphs are built for all the groups in line 4. Since there are no writing conflicts in the construction processes of different graphs, we use a single thread to construct a graph for each group, and the graphs across different groups are constructed in parallel. In line 6, we also iterate each $(v_i, f_i) \in \mathcal{D}$ in parallel, during which there are no writing conflicts for lines 12 and 14, as they only write the out-neighbor list for different vectors simultaneously.

Next, we show that the *UNG* framework guarantees the *completeness* for any top-k filtered ANNS query. Specifically, when $\delta + 1 \geq k$, it can return at least $\min\{k, |V_q|\}$ vectors in the filtered search space V_q. Note that Theorem 5.4 assumes that there are fewer than k vectors associated with the entry label sets $\hat{F}_q$, otherwise, we already have k vectors in the initialized priority queue.

Theorem 5.4. *Given the entry label sets $\hat{F}_q$, starting from all the vectors with $f \in \hat{F}_q$, traversal in the unified navigating graph G can reach at least $\min\{\delta + 1, |V_q|\}$ vectors in filtered search space V_q.*

Proof. We only need to consider the case that some vectors in V_q cannot be reached.

Cross-group edges always exist for any pair of groups s.t. their corresponding label sets have an edge in *LNG*. Since the query process starts from all vectors in all entry label sets, if it cannot reach any vector in the group $V_{f'} \subseteq V_q$, it also fails to visit vectors v_i in another group V_f s.t. (1) $f_q \subset f \subset f'$, (2) $f \notin \hat{F}_q$, and (3) the cross-group edge from v_i to $v_j \in V_{f'}$ exists in G. Thus, there must exist a group $V_{f^*} \subseteq V_q$ s.t. (1) we can only reach part of its vectors but not all of them, and (2) $f^* \notin \hat{F}_q$. To reach vector v^* in such a group V_{f^*}, we must pass through a cross-group edge $e(v^\dagger, v^*)$ after visiting $v^\dagger$ in another group. Note that $v^\dagger$ must connect to exactly δ cross-group edges in G, otherwise, those unreachable vectors in V_{f^*} can be connected to $v^\dagger$ when building cross-group

edges. Therefore, we at least can reach $v^\dagger$ and all its cross-group edges, i.e., we can reach $\delta + 1$ vectors in V_q. □

In practice, achieving *completeness* is typically possible even when $\delta + 1 < k$. This is because for queries with high specificity, it is easy to reach k vectors within a vast filtered search space V_q. On the other hand, when handling queries with low specificity, any group V_f that satisfies $f_q \subseteq f$ probably consists of a small number of vectors. In such cases, it is highly likely that each vector within V_f can establish connections with other vectors in the proximity graph G_f, thus we can access most vectors in V_q.

Time complexity. Recall that d_{LNG} is the average out-degree of label sets F in *LNG*. Algorithm 4 builds the *LNG* with $O(d_{LNG}|F|^2|\mathcal{L}|)$ time, as discussed in Section 5.1. As different graph-based ANNS methods vary in complexity, we assume that a certain algorithm needs $O(C)$ time to build all the proximity graphs G_f in line 4. Similar to the analysis in Section 4.2, to obtain the top-δ nearest vector within a proximity graph V_f, let w' denote the maximum size of the priority queue and ρ' denote the number of visited vectors. Hence, we have $w' \geq \delta$ and $\rho' \geq \delta$. To find the $\tilde{S}_{v_i,f'}$ for each vector $v_i \in V \subset \mathbb{R}^n$, line 9 requires $O(d_{LNG}|V|\rho'(n + \log w'))$ time in total. Lines 10-13 consume $O(d_{LNG}|V|\delta \log \delta)$ time in total. Therefore, the overall time complexity of Algorithm 4 is $O\left(C + d_{LNG}|F|^2|\mathcal{L}| + d_{LNG}|V|\rho'(n + \log w')\right)$.

Space complexity. As discussed in Section 5.1, building the *LNG* in Algorithm 4 requires $O(|F|(|\mathcal{L}|+ d_{LNG}))$ space. Recall that d_{max} denotes the degree bound for each vector in any proximity graph G_f. For most graph-based ANNS algorithms, building each G_f in line 4 takes $O(|V_f|d_{max})$ space. Similarly, lines 9-10 consume $O(d_{LNG}\delta + w' + |V|\delta)$ space, while lines 12-13 need $O(d_{LNG}|F|\delta)$ space in total. Thus, the overall space complexity of Algorithm 4 is $O\left(|F|(|\mathcal{L}| + d_{LNG}\delta) + |V|d_{max} + w'\right)$.

The Impact of Labels and Group Sizes. Since UNG partitions all vectors according to their label sets, the number of labels $|\mathcal{L}|$ and the number of groups $|F|$ will significantly affect the index structure. For small $|\mathcal{L}|$ and $|F|$, intuitively the index structure tends to be the underlying proximity graph. When $|\mathcal{L}|$ and $|F|$ become larger, each group becomes smaller and the cross-group edges will contribute to most index edges. Since the cross-group edges are linked by selecting top-δ closest vectors that satisfy the minimum superset relationships, such a graph can be partially regarded as a k-Nearest Neighbor Graph (kNNG) [32] with minimum superset constraints. Moreover, since UNG only visits vectors satisfying filter conditions, it can still achieve good performance for large $|\mathcal{L}|$ and $|F|$, as demonstrated in Section 6.4.3.

5.3 Dealing with Data Update

For simplicity, we focus on data insertion and deletion, as updating a vector or its labels can be handled through insertion following deletion. Note that *UNG* is a general framework built on a graph-based ANNS algorithm. Thus, the underlying graph index update mechanism applies when inserting or deleting vectors in a group V_f. Next, we discuss how to update the trie tree, label navigating graph (*LNG*), and cross-group edges in the *UNG* index.

Insertion. When adding a vector v_i, if its label set f_i is new, we insert f_i into the trie tree and update the *LNG* according to minimum superset relationships between all subsets and the minimum supersets of f_i. If f_i already exists, there is no need to update the trie tree and the *LNG*. Then, for each in-neighboring label set f' of f_i in the *LNG*, the cross-group edges will be rebuilt for all vectors in group $V_{f'}$. Finally, we create the cross-group edges that starts from v_i.

Deletion. When removing a vector v_i, if no vectors remain with label set f_i, we remove f_i from the trie tree and then update the *LNG* similarly. If f_i is still used, the trie tree and the *LNG* remain unchanged. Then, for each in-neighboring label set f' of f_i in the previous *LNG*, we rebuild cross-group edges for vectors in the group $V_{f'}$.

Table 3. Statistics of datasets, where NA means the dataset does not fit the scenario.

Dataset	Dim	# Base	# Query	Source	Label-equality			Label-containment					
					$\lvert\mathcal{L}\rvert$	Avg. spec.	Cardinality	$\lvert\mathcal{L}\rvert$	$\lvert F\rvert$	Cardinality	# LNG edges	d_{LNG}	Avg. spec.
SIFT1M	128	1,000,000	10,000	Image	7	0.08	12	12	2,279	4,095	12,045	5.3	0.18
Audio	192	53,387	200	Audio	7	0.08	12	12	1,106	4,095	5,279	4.8	0.18
GIST1M	960	1,000,000	1,000	Image	7	0.08	12	12	2,279	4,095	12,045	5.3	0.18
Msong	420	992,272	200	Audio	7	0.08	12	12	2,276	4,095	12,025	5.3	0.18
Paper	200	2,029,997	10,000	Text	7	0.08	12	12	2,592	4,095	14,002	5.4	0.18
Crawl	300	1,989,995	10,000	Text	7	0.08	12	12	2,584	4,095	13,952	5.4	0.18
Enron	1,369	94,987	200	Text	7	0.08	12	12	1,321	4,095	6,423	4.9	0.18
GloVe	100	1,183,514	10,000	Text	7	0.08	12	12	2,344	4,095	12,451	5.3	0.18
arXiv	768	132,687	1,000	Text	26	0.01	168	4,231	58,617	$> 10^{1000}$	96,739	1.7	2.5×10^{-4}
LAION1M	512	1,000,448	1,000	Image	30	0.01	24,360	NA	NA	NA	NA	NA	NA
Words	3,072	8,000	200	Text	NA	NA	NA	26	5,431	$> 10^{7}$	39,191	7.2	1.5×10^{-4}
TripClick	768	1,055,976	1,000	Text	NA	NA	NA	28	7,734	$> 10^{8}$	30,456	3.9	0.02
MTG	1,152	40,274	1,000	Image	NA	NA	NA	847	2,845	$> 10^{12}$	5,455	1.9	0.02

6 Experiments

In this section, we evaluate the performance of the *UNG* framework and compare it with state-of-the-art filtered ANNS algorithms.

6.1 Experimental Setup

In practice, the task of filtered ANNS has two different scenarios summarised from existing works.

- *Label-equality scenario.* Recent algorithms such as *NHQ* [44] and *CAPS* [15] assume that the number of labels for each base vector and query vector is fixed. Thus, for any vector $v_i \in V$, the filter requirement $f_q \subseteq f_i$ falls into the special case of $f_q = f_i$.
- *Label-containment scenario.* In this case, both base and query vectors may have different numbers of labels. It aligns with our general filter requirement of $f_q \subseteq f_i$.

The *versatility* of the *UNG* framework enables it to handle both cases. However, since *NHQ* [44] and *CAPS* [15] fail to deal with the label-containment scenario, we also conduct experiments on the label-equality scenario to compare *UNG* with them, in which algorithms such as *StitchedVamana* [14], *FilteredVamana* [14] and *ACORN* [34] also work.

Datasets. Table 3 summarises 13 real-world datasets used in the experiments. Next, we briefly introduce the following datasets.

- Words [9]. It consists of embeddings for 8,000 words, generated by the *text-embedding-3-large* model from OpenAI [30]. Each word is associated with a set of characters comprising the word, which is regarded as the label set.
- ArXiv [27]. It contains 132,687 research papers with valid arXiv IDs collected by [31], each of which has an aspect-specific embedding representing the methodology. For each paper, we extract the year and month from its arXiv ID, and obtain 26 distinct labels for 12 months and 14 years for the label-equality scenario. As for the label-containment scenario, we further extract three other attributes including "task", "method", and "dataset", each of which has 1678, 890, and 1637 different values, respectively. Each vector may have zero or several values for such attributes.
- TripClick [38] and LAION1M [39]. Recent work [34] employs them for filtered ANNS. Since only text and image data are publicly available, we generate all vectors and categorical labels based on the method provided in [34].
- MTG [42]. It consists of visual embeddings derived from cropped artwork of "Magic. The Gathering" (MTG) cards [40], which are generated with the OpenCLIP model [5] trained on the LAION-400M dataset [39]. The extracted attributes include "color", "loyalty", "power", and "mana

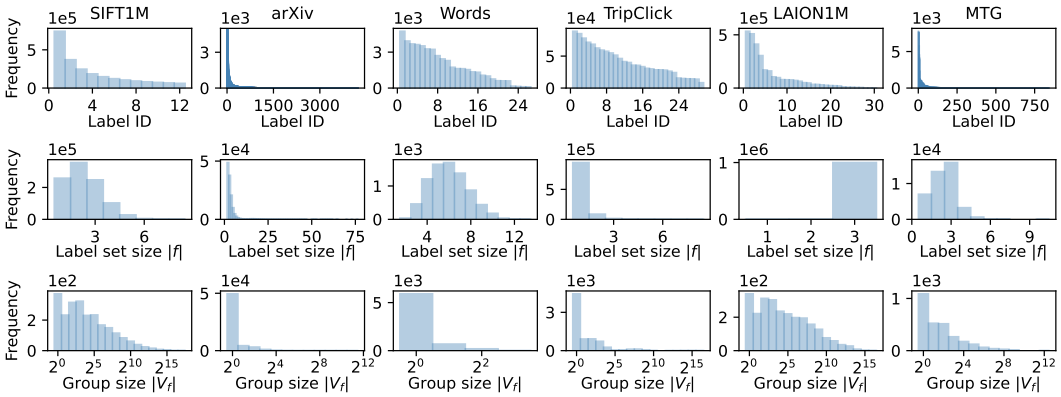


Fig. 5. Distributions of base label, size of base label sets, and the number of groups.

cost", each of which has 6, 12, 36, and 793 different values, respectively. A card may have multiple values for attributes "color" or "mana cost".

- The other datasets are used in prior works [14, 15, 44], which are not associated with labels except for the Paper dataset. Following prior works [15, 44], we use synthesized label sets for these datasets by considering different scenarios.
  Label-equality scenario. The synthetic labels are provided by [44]. Each base vector has three attributes, and each attribute can take 3, 2, and 2 distinct values, respectively. Thus, there are $|\mathcal{L}| = 7$ possible labels with a cardinality of 12.
  Label-containment scenario. In the field of information retrieval, tags of items often approximate the power law distribution [36]. Thus, [14] adopts the power law distribution, Zipf distribution, to generate labels. We directly use their source codes to obtain synthetic label sets for base vectors, in which $|\mathcal{L}| = 12$ by default, resulting in the cardinality of $2^{12} - 1$.

Figure 5 shows the synthetic label distribution of SIFT1M, while the other five datasets have real labels. There are a few label IDs that frequently appear in numerous vectors, while a large portion of label IDs rarely appear in the dataset.

Query vectors for the datasets Words, arXiv, TripClick, and LAION1M are randomly selected from base vectors, while the others have provided query vectors. For all datasets, the query labels are randomly generated following the distribution of base labels.

Metrics. For answering top-k filtered ANNS queries, the efficiency of an algorithm is measured by queries per second (QPS), which is the number of queries divided by the search time in seconds. The quality of search results is evaluated by *recall@k*= $\frac{|\tilde{S} \cap S|}{k}$, where $\tilde{S}$ represents the search results and S is the ground-truth top-k answers. In our experiments, we set $k = 10$ by default.

Algorithms and Parameters. Recent algorithms *NHQ* [44], *CAPS* [15], *StitchedVamana* [14], *FilteredVamana* [14], *ACORN-γ* [34], and *ACORN-1* [34] have been proven to outperform other approaches significantly, thus they are considered as baselines for comparing with the *UNG* framework. For each algorithm, we use recommended or default parameters if provided, and try to tune the other parameters on each dataset for a Pareto-optimal recall-QPS curve.

- *Filtered-Scan.* We use the trie tree index to efficiently locate all vectors satisfying the query filters, then conduct a linear scan over the filtered vectors to return the answers.
- *NHQ-NPG-NSW* [44]. For each dataset, we use the best value for the parameter M_{max} ranging from 16 to 128, and the best value for $efConstruction$ ranging from 10 to 160.

- *NHQ-NPG-KGraph* [44]. For each dataset, we choose the best values for the number of neighbors $K \in \{32, 48\}$. The maximum length of the search queue ranges from 7 to 300.
- *CAPS* [15]. It employs K-means clustering with 1024 clusters to construct the index. During the query process, the maximum number of candidates ranges from 100 to 30000.
- *StitchedVamana* [14]. In the label-equality scenario, we use the same indexing parameters as [14], i.e., $R_{small} = 32$, $L_{small} = 100$, and $R_{stitched} = 64$. The maximum length of search queue ranges from 10 to 510 for answering queries. In the label-containment scenario, since the above parameters result in the inferior performance, we set $R_{small} = 48$, $L_{small} = 200$, and $R_{stitched} = 96$ after tuning them. The maximum length of search queue ranges from 100 to 1400 during querying.
- *FilteredVamana* [14]. In the label-equality scenario, we also use the indexing parameters mentioned in [14], i.e., $R = 96$ and $L = 90$. The maximum length of search queue ranges from 10 to 510 for answering queries. In the label-containment scenario, for better performance, we also tune the parameters and set $R = 128$ and $L = 180$. The maximum length of search queue ranges from 100 to 1400 for query processing.
- *ACORN-γ* [34]. Following the parameters presented in [34], we set $M = 32$ and $M_\beta = 64$ for both scenarios across all datasets except TripClick and LAION1M. We set $M = M_\beta = 128$ and $\gamma = 80$ on the Tripclick dataset, while using $M = M_\beta = 32$ and $\gamma = 30$ on the LAION1M dataset. We use $\gamma = 80$ for datasets Words, arXiv, and MTG, while setting $\gamma = 12$ for the other datasets. The maximum length of the search queue for processing queries ranges from 5 to 400 in the label-equality scenario and from 5 to 1200 in the label-containment scenario. Furthermore, *ACORN-γ* uses the *filtered-scan* approach as a fallback for queries with specificity lower than $1/\gamma$. However, their official code [33] does not implement the specificity estimator or this fallback strategy. Thus, we utilize the actual specificity to implement such strategy.
- *ACORN-1* [34]. Aligning with [34], we use the same parameters as *ACORN-γ*, except that $\gamma = 1$ and $M_\beta = 2M$.
- *UNG*. For both scenarios, the *UNG* framework utilizes *Vamana* algorithm [41] to build the underlying proximity graphs by default, with $\alpha = 1.2$, degree bound $R = 32$ and the maximum queue length $L = 100$. To build cross-group edges, we set the parameter $\delta = 6$. To answer queries, the number of entry vectors $\sigma = 16$. The maximum length of search queue ranges from 10 to 400 in label-equality scenario, and it ranges from 10 to 1200 in label-containment scenario.

Implementation. Programs are implemented in C++ and compiled with -Ofast Optimization. All the experiments are executed on a Linux server with Intel(R) E5-2678v3 CPU @2.5GHz and 220GB of RAM. For all algorithms, we use 32 threads to construct the index. To process ANNS queries, we use a single thread in the label-equality scenario by default to align with previous works [15, 44]. For the label-containment scenario, we use 16 threads by default, and the impact of varying query threads will be discussed in Section 6.4.1. Moreover, the official code of ACORN [33] builds a bitmap for each query to indicate whether each base vector passes the filter condition. This cost is included for computing QPS to ensure a fair comparison. Our source code for the proposed *UNG* framework is available at https://github.com/YZ-Cai/Unified-Navigating-Graph.

6.2 Overall Query Performance

For the label-equality scenario, Figure 6 compares the query performance of the proposed *UNG* framework with existing methods. Unfortunately, the provided codes of *NHQ-NPG-NSW* and *NHQ-NPG-KGraph* [44] fail to return valid results on datasets Audio, Enron, arXiv, and LAION1M. Also, *ACORN-γ* and *ACORN-1* fail on the Crawl dataset. Among all datasets, our method is one of the best for the trade-off of query efficiency (QPS) and answer quality (recall). In datasets SIFT1M, Crawl, arXiv and LAION1M, *UNG* achieves 2× higher QPS than any other algorithm at 95% recall.

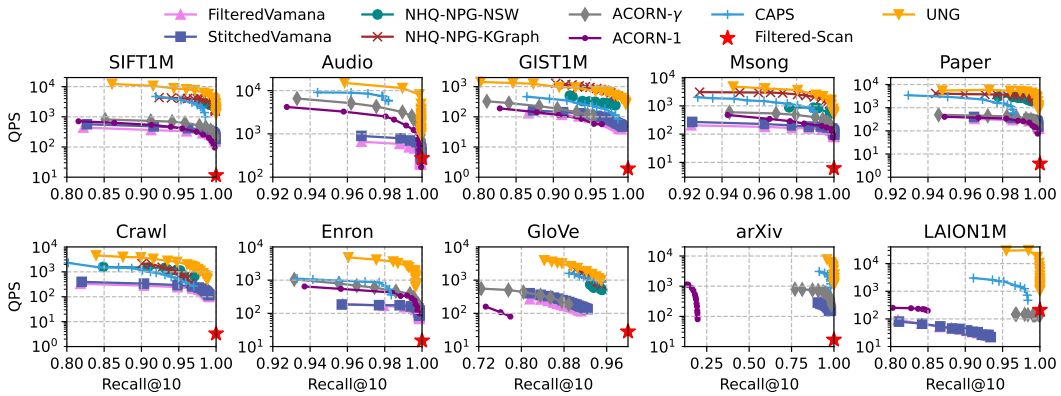


Fig. 6. Query performance in label-equality scenario

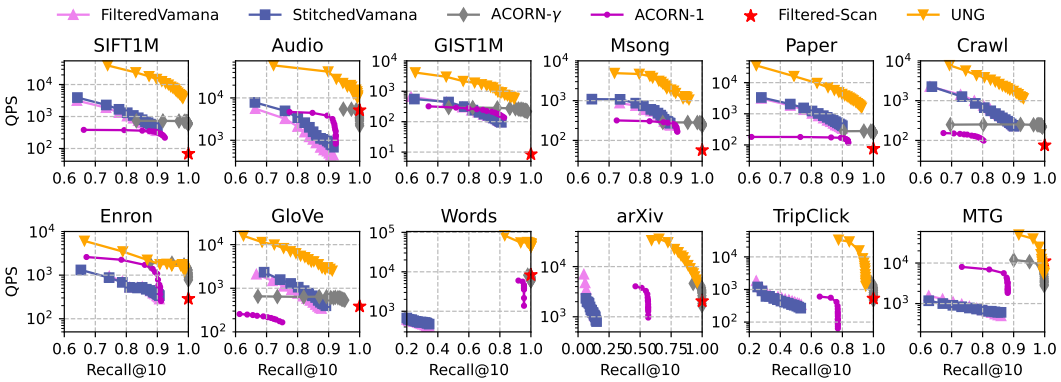


Fig. 7. Query performance in label-containment scenario

In the label-containment scenario, since *NHQ* [44] and *CAPS* [15] cannot address this scenario, we compare the *UNG* framework with *StitchedVamana* [14], *FilteredVamana* [14], *ACORN-γ* [34], and *ACORN-1* [34]. Figure 7 presents the results. Generally, the *UNG* framework is 10× faster than all graph-based methods at 95% recall, except for the Enron dataset. The proposed *UNG* framework achieves a good balance between accuracy and efficiency due to its *fidelity*, which ensures that only vectors satisfying the filter condition are considered, while the other algorithms may waste unnecessary time in exploring numerous irrelevant vectors.

6.3 Evaluating Index Construction

6.3.1 Index Construction Time. As the label sets of base vectors are different for label-equality and label-containment scenarios, Table 4 shows the index construction time for each scenario, where '-' indicates failure to return results. In the label-equality scenario, cross-group edges are unnecessary since all vectors have the same label set size. For the *UNG* framework, the time cost of processing label sets and constructing the label navigating graph *LNG* is marginal compared to the overall indexing process, since the resulting *LNG* has a small average degree d_{LNG}, as shown in Table 3.

6.3.2 Index Size. Table 5 reports the index size of different algorithms in each scenario. The *UNG* framework can achieve better performance even with a smaller degree bound for index construction,

Table 4. Index construction time (s), where NA means the dataset does not fit the scenario.

Scenario	Algorithm	SIFT1M	Audio	GIST1M	Msong	Paper	Crawl	Enron	GloVe	arXiv	LAION1M	Words	TripClick	MTG
Label-equality	*NHQ-NPG-NSW*	19	-	202	35	43	228	-	151	-	-	NA	NA	NA
	NHQ-NPG-KGraph	182	-	317	156	236	431	-	608	-	-	NA	NA	NA
	CAPS	151	55	419	344	249	259	204	176	168	238	NA	NA	NA
	StitchedVamana	99	4	1,345	474	385	692	55	79	35	1,274	NA	NA	NA
	FilteredVamana	202	11	2,397	721	604	1,126	75	228	64	1,429	NA	NA	NA
	ACORN-γ	282	7	1,984	656	628	-	96	167	589	1,563	NA	NA	NA
	ACORN-1	16	1	114	52	58	-	9	10	7	59	NA	NA	NA
	UNG	50	8	273	115	176	234	3	93	9	82	NA	NA	NA
	- Process labels & *LNG*	13	1	13	13	26	25	1	16	2	3	NA	NA	NA
	- Build proximity graphs	37	7	260	102	150	209	2	77	8	78	NA	NA	NA
Label-containment	*StitchedVamana*	152	8	2,666	566	536	1,401	93	272	133	NA	227	1,627	29
	FilteredVamana	358	16	3,724	942	1,014	2,382	116	363	88	NA	40	504	61
	ACORN-γ	282	7	1,984	656	628	1,555	96	167	589	NA	14	21,965	60
	ACORN-1	16	1	114	52	58	97	9	10	7	NA	2	962	3
	UNG	433	10	4,369	1,174	1,486	4,047	87	1,342	42	NA	4	479	6
	- Process labels & *LNG*	10	1	10	11	21	25	1	13	1	NA	3	2	1
	- Build proximity graphs	238	6	2,727	601	833	2,207	40	190	1	NA	1	179	5
	- Add cross-group edges	185	4	1,632	562	632	1,816	45	1,139	40	NA	1	297	1

Table 5. Index size (MB) for different scenarios, where NA means the dataset does not fit the scenario.

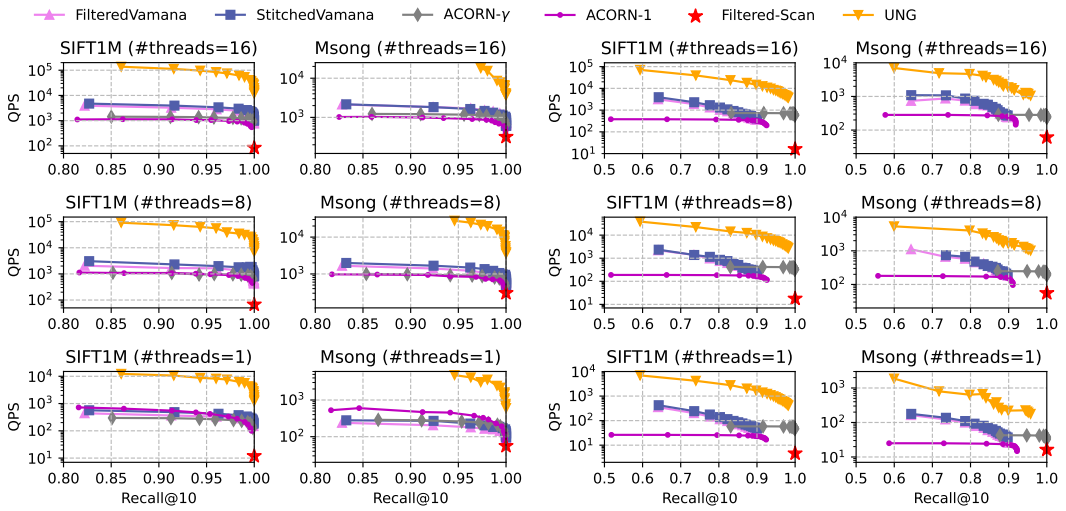
Scenario	Algorithm	SIFT1M	Audio	GIST1M	Msong	Paper	Crawl	Enron	GloVe	arXiv	LAION1M	Words	TripClick	MTG
Label-equality	*NHQ-NPG-NSW*	125	-	241	119	143	240	-	436	-	-	NA	NA	NA
	NHQ-NPG-KGraph	42	-	45	30	62	79	-	68	-	-	NA	NA	NA
	CAPS	8	1	12	9	17	17	6	10	4	10	NA	NA	NA
	StitchedVamana	526	22	430	437	918	629	39	300	49	740	NA	NA	NA
	FilteredVamana	362	16	281	309	658	458	27	223	50	456	NA	NA	NA
	ACORN-γ	486	26	486	482	987	-	46	572	111	567	NA	NA	NA
	ACORN-1	442	23	442	440	899	-	42	542	59	439	NA	NA	NA
	UNG	115	4	110	88	192	132	8	58	12	174	NA	NA	NA
Label-containment	*StitchedVamana*	645	25	548	516	1,125	800	45	381	91	NA	9	510	26
	FilteredVamana	490	24	423	441	931	741	40	393	69	NA	4	391	20
	ACORN-γ	486	26	486	482	987	967	46	572	111	NA	8	1,512	30
	ACORN-1	442	23	442	440	899	881	42	542	59	NA	4	1,702	17
	UNG	115	4	110	88	207	181	8	86	11	NA	4	241	6
Vector data size		489	40	3,663	1,590	1,549	2,278	497	452	389	1,958	94	3,098	178

resulting in a smaller index size compared to both *FilteredVamana* and *StitchedVamana*. Note that our index size is also significantly smaller than the vector data size presented in Table 5, which makes it practically feasible for real-world filtered ANNS applications.

6.4 Impacts of Factors

6.4.1 Varying Query Threads. Since parallelism may influence query performance, we adjust the number of threads for query processing in both scenarios. Note that the implementations of *NHQ* and *CAPS* can only support a single thread, therefore we are unable to compare them with our *UNG* framework when varying query threads. As shown in Figure 8, in both scenarios, *UNG* is consistently the best, demonstrating robustness across different thread counts.

6.4.2 Varying Specificity of Query Label Sets. Similar to previous work [14], we conduct experiments using query label sets constructed at various specificity levels. For the 75th, 50th, 25th, and 1st percentiles, the queries of SIFT1M have the specificity of 0.260, 0.062, 0.013, and 0.001, while the specificity for arXiv is 0.057, 0.001, 2.1×10^{-4}, and 7.5×10^{-5}. In the label-containment scenario, the *UNG* framework is still much better than all the baselines, and its performance improves with lower specificity, since the smaller filtered search space results in better accuracy and efficiency of *UNG*. In contrast, *StitchedVamana*, *FilteredVamana*, and *ACORN-1* are getting worse. It is because they struggle to find enough neighbors that meet query filters with low specificity, resulting in a



(a) Label-equality scenario (b) Label-containment scenario

Fig. 8. Varying the number of threads for query processing.

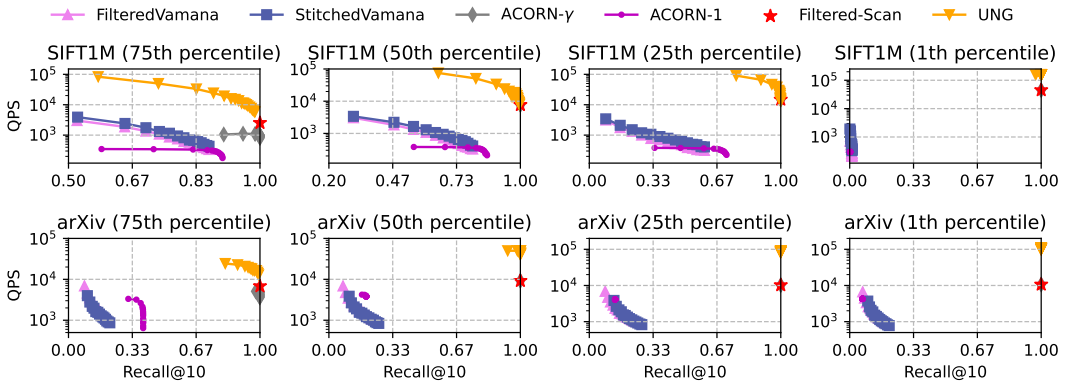


Fig. 9. Varying specificity of query label sets.

higher risk of getting disconnected during the search. Moreover, they have poorer performance on the arXiv dataset with thousands of labels, as their neighbor set is harder to cover so many labels. Note that the fallback strategy of *ACORN-γ* works when the specificity is lower than $1/\gamma$, which makes it always use *filtered-scan* approach at 50th, 25th, and 1st percentiles.

6.4.3 Varying Number of Labels for Base Vectors. For generating synthetic labels in the label-containment scenario, the number of labels $|\mathcal{L}|$ results in a cardinality of $2^{|\mathcal{L}|} - 1$. Take the SIFT1M dataset with 1 million vectors as an example. The number of groups for $|\mathcal{L}| = 36, 128, 512$, and 2048 is 91277, 422271, 754057, and 914610, respectively. When $|\mathcal{L}| = 2048$, each group is relatively small and most groups contain only one vector. The UNG framework connects a vector to its top-δ closest vectors that satisfy the minimum superset relationships, which can be considered a k-Nearest Neighbor Graph (kNNG) [32] with minimum superset constraints.

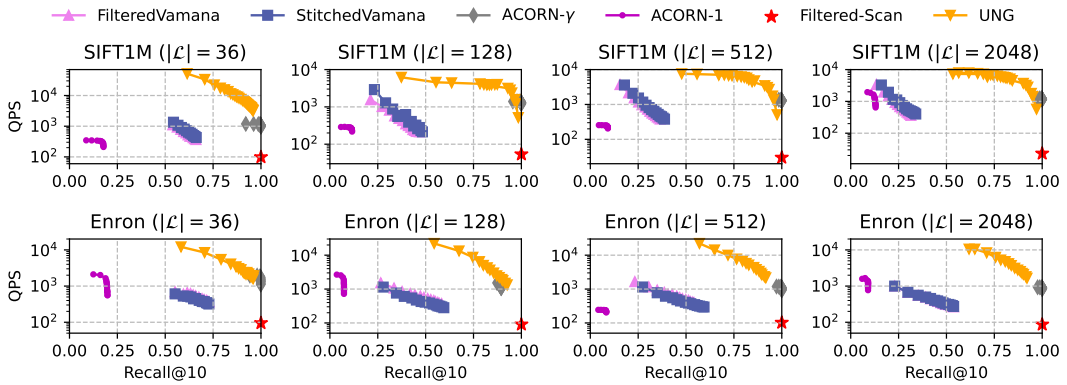


Fig. 10. Varying the number of labels $|\mathcal{L}|$ for base vectors.

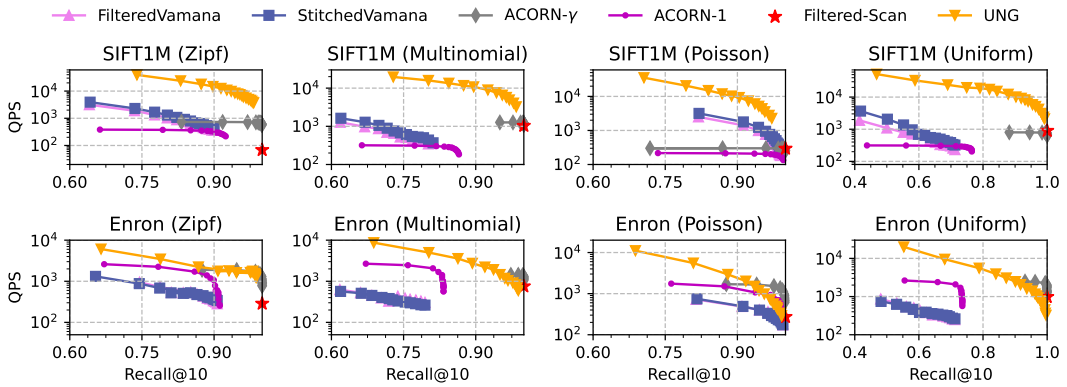


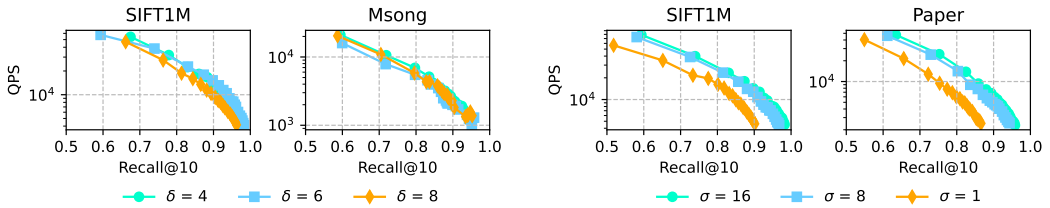
Fig. 11. Varying label distributions.

Figure 10 shows that *UNG* significantly outperforms the baselines like *StitchedVamana*, *FilteredVamana*, and *ACORN-1* when $|\mathcal{L}|$ varies, and the gap becomes larger with as $|\mathcal{L}|$ increases. For the proximity graphs of these baselines, the limited number of neighbors of each vector is insufficient to cover such a large number of labels. Thus, their graph traversal may terminate early, resulting in a significantly reduced recall, particularly when the cardinality exceeds 10^{600} for $|\mathcal{L}| = 2048$.

6.4.4 Varying Label Distribution. To examine the robustness of the UNG framework under different label distributions, we use four distributions to generate the base labels in the label-containment scenario. Similar to real-world datasets, the expected number of labels per vector is set to 3. The query labels are also randomly generated following the corresponding distribution. As shown in Figure 11, for the *FilteredVamana*, *StitchedVamana*, and *ACORN-1* algorithms, when certain label IDs frequently appear in numerous vectors, each vector is more likely to have neighbors with shared labels and small distances simultaneously. However, under the multinomial and uniform distributions, such advantages are not present and their neighbors may lack the required labels, resulting in poor connectivity and significantly reduced recall.

6.5 Impact of Parameters

When varying a certain parameter, we keep the others as the default settings outlined in Section 6.1.



(a) Varying δ for building cross-group edges

(b) Varying σ for selecting entry vectors

Fig. 12. Impact of parameters.

6.5.1 Varying δ for Building Cross-group Edges. In the unified navigating graph, each vector has at most δ edges connecting with other groups. While setting a large δ improves its connectivity, this results in a larger index size, thus we consider using small values of δ in practice. By varying the parameter $\delta \in \{4, 6, 8\}$ on datasets SIFT1M and Msong, such small values of δ have minimal influence on query performance, as shown in Figure 12(a).

6.5.2 Varying σ for Selecting Entry Vectors. For greedy search in the unified navigating graph, we randomly select σ entry vectors in group V_f for each entry label set f. When trying different values of $\sigma \in \{1, 8, 16\}$ on the datasets SIFT1M and Paper, Figure 12(b) shows that increasing the number of entry vectors leads to improved performance, as it allows for the exploration of a wider search space, and the greedy search is less prone to becoming trapped in local optima.

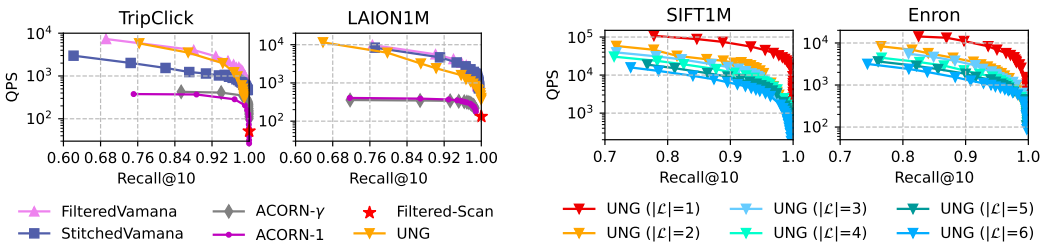
6.6 Capacity of Handling Other Scenarios

6.6.1 Label-overlap Scenario. Besides the label-equality and label-containment scenarios discussed previously, *UNG* is also capable of handling the label-overlap scenario. Specifically, given a query label set f_q, a base vector v_i with labels f_i satisfies the filter iff $f_i \cap f_q \neq \emptyset$. To answer such a query, UNG conducts a label-containment search for each query label $l \in f_q$ separately and then combines all results to obtain the final top-k vectors. Note that the search space for each $l \in f_q$ may share some vectors. We ensure that each vector is visited only once to avoid unnecessary computation.

The index construction uses the same parameters as the label-containment scenario. The search queue length for *UNG* ranges from 10 to 3000, while it ranges from 10 to 510 for *FilteredVamana* and *StitchedVamana* [14]. Figure 13(a) shows that *FilteredVamana* outperforms UNG since UNG conducts a greedy search $|f_q|$ times, whereas *FilteredVamana* performs the search only once. Moreover, *FilteredVamana* links vectors according to their overlapping labels, which is inherently compatible with the label-overlap scenario.

6.6.2 UNG for ANNS without Filters. The proposed *UNG* index also accommodates traditional ANNS queries without filters, even though each vector in the datasets is associated with a set of labels. Specifically, it is consistent with the label-overlap scenario when the query label set $f_q = \mathcal{L}$, where $\mathcal{L}$ represents the universe of all possible labels. In this case, we apply the same procedure as discussed in the label-overlap scenario above.

In Figure 13(b), $|\mathcal{L}| = 1$ indicates that there is only one group (equivalent to the Vamana index [41]). We can find that QPS decreases as $|\mathcal{L}|$ increases. This is because *UNG* considers filters when constructing cross-group edges, while some necessary edges in the traditional proximity graphs (e.g., Vamana for ANNS without filters) are broken since they do not follow the minimum superset relationship for the associated label sets. Consequently, the connectivity of the unified proximity graph is partially compromised.



(a) Label-overlap scenario

(b) Answering ANNS without filters

Fig. 13. Query performance of handling other scenarios.

Notably, the proposed UNG index can handle four different scenarios simultaneously: label-equality, label-containment, label-overlap, and ANNS without filters, while maintaining reasonable recall and QPS. This capability allows for the concurrent processing of diverse requests using a single index, thereby enhancing the applicability and scalability of the system.

6.6.3 Limitation Discussion. Experiments (Section 6.4) demonstrate that the *UNG* framework is robust when varying query threads, query specificity, number of labels $|\mathcal{L}|$, number of groups $|F|$, as well as the label distribution. However, *UNG* shows inferior performance in the label-overlap scenario and the traditional ANNS scenarios without filters. As discussed in Sections 6.6.1, *UNG* conducts greedy search $|f_q|$ times for the label-overlap scenario, while $|f_q| = |\mathcal{L}|$ for ANNS without filters. In such scenarios, for the same query label set, the search space of *UNG* involves more groups compared to label-containment scenario, indicating that cross-group edges are used more frequently. However, since such edges only appear between label sets that have minimum superset relationships, their connectivity is inferior to that of edges within a single group.

7 Related work

7.1 ANNS Approaches

Partition-based Approaches. Such approaches partition all vectors to prune unnecessary subspaces during query processing, including algorithms such as Inverted multi-index (IMI) [3], FAISS-IVF [7], Ball Tree [4], VP-tree [52], and R-tree [16].

Hashing-based Approaches. The general idea is to transform vectors into low-dimensional representations. Well-known algorithms include Locality-Sensitive Hashing (LSH) [1], C2LSH [11], QALSH [18], Product Quantization (PQ) [19], and Optimized PQ (OPQ) [13].

Graph-based Approaches. Methods based on graph search have become a highly effective option for ANNS due to the proven superior trade-off for answer quality and query efficiency [23, 45, 46]. There are dozens of existing algorithms, such as NSW [25], *kNNG* [32], KGraph [6], HNSW [26], NSG [10], Vamana [41], and *APG* [55].

7.2 Filtered ANNS Approaches

Filter-then-search. In this paradigm, an algorithm first examines the label sets for base vectors and then performs ANNS queries within the filtered search space. AnalyticDB-V [48], Milvus [43], and Weaviate [47] obtain all the relevant vectors by checking their label sets, such as using a B-tree index. Then, they scan the filtered vectors by brute force or build a bitmap to ensure that only filtered vectors are considered when conducting traditional ANNS searches. Such an idea works well for queries with low specificity [14, 15], but will be prohibitively inefficient when a large

number of vectors satisfy the constraints. To avoid the high cost of dynamically specifying the search space, *MA-NSW* [50] adopts another idea of building a separate index for each possible query label set. Though we can directly identify the search space for any query, it results in significant memory consumption when there are many labels, since each vector can be duplicated in different indices and the number of possible query label sets grows exponentially.

Search-then-filter. Its general idea is to use traditional ANNS to find some nearest candidates, then check their label sets to return those satisfying the filter condition. Vearch [21] directly adopts the idea, while PASE [51] iteratively pops a batch of nearest vectors to check their labels until k filtered vectors are found. Also, both AnalyticDB-V [48] and Milvus [43] implement this paradigm for certain scenarios. In contrast to the filter-then-search paradigm, search-then-filter works well for queries with high specificity, since it is more likely to encounter a filtered vector among the candidates. However, queries with low specificity require a large number of candidates, resulting in significantly reduced performance.

Filter-during-search. Algorithms following this paradigm will check the label sets during the search process. *AIRSHIP* [54] starts from some examined filtered vectors and performs the graph search, during which the label sets of each neighbor are checked. Algorithms *FilteredVamana* and *Stitched-Vamana* presented in [14] build the proximity graphs by linking two vectors that share common labels. Then, they also start from some selected vectors and only explores base vectors that share common labels with the query label set. *NHQ* [44] and *HQANN* [49] use the label sets by computing a fusion distance metric, which is the weighted sum of vector distance and a specialized distance for label sets. They perform a greedy search over the composite proximity graph built using such fusion distance. Recent algorithm *ACORN* [34] constructs a denser HNSW with compression techniques, where each point is connected to a significantly larger number of neighbors. To simulate HNSW operating within a filtered search space, the algorithm ignores neighbors that do not meet the specified filter criteria during the search process. Different from the graph-based filtered ANNS algorithms mentioned above, [15] proposes an IVF-based method named *CAPS*. It follows the traditional IVF algorithm to select some partitions, then it navigates through an attribute frequency tree (AFT) built for each partition to locate the sub-partition that contains the filtered vectors.

A recent study SeRF [56] addresses the problem of range-filtered ANNS. In their scenario, each vector is associated with an attribute that has a total order, and the range-filtered search only considers vectors with attribute values within the specified query range. Since the query filter is a range and each filtered vector has a unique attribute that falls within this range, SeRF fails to handle both the label-equality and label-containment scenarios.

8 Conclusion

In this paper, we study the problem of filtered approximate nearest neighbor search and develop a novel filter-then-search framework *UNG*. By capturing the containment relationships among label sets, *UNG* avoids iterating all these label sets exhaustively during query processing and enables the construction of efficient proximity graphs for the filtered vectors. Thus, *UNG* offers four fundamental properties: *versatility*, *fidelity*, *completeness*, and *adaptability*. We conduct extensive experiments on 10 datasets and the results consistently demonstrate that *UNG* runs much faster than the baselines at the same accuracy.

Acknowledgments

This work was substantially supported Key Projects of the National Natural Science Foundation of China (Grant No. U23A20496) and Shanghai Science and Technology Innovation Action Plan (Grant No. 21511100401). Weiguo Zheng is the corresponding author.

References

[1] Alexandr Andoni and Piotr Indyk. 2006. Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions. In *2006 47th Annual IEEE Symposium on Foundations of Computer Science (FOCS'06)*. https://doi.org/10.1109/focs.2006.49

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

[3] Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence* 37, 6 (2014), 1247–1260.

[4] Lawrence Cayton. 2008. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the 25th international conference on Machine learning - ICML '08*. https://doi.org/10.1145/1390156.1390171

[5] Mehdi Cherti, Romain Beaumont, Ross Wightman, Mitchell Wortsman, Gabriel Ilharco, Cade Gordon, Christoph Schuhmann, Ludwig Schmidt, and Jenia Jitsev. 2023. Reproducible scaling laws for contrastive language-image learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2818–2829.

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

[7] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[8] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2021. New trends in high-d vector similarity search: al-driven, progressive, and distributed. *Proceedings of the VLDB Endowment* 14, 12 (2021), 3198–3201.

[9] Efarrall. 2024. Word Embeddings. https://huggingface.co/datasets/efarrall/word_embeddings.

[10] 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).

[11] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*. 541–552.

[12] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* (2023).

[13] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (Apr 2014), 744–755. https://doi.org/10.1109/tpami.2013.240

[14] 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.

[15] Gaurav Gupta, Jonah Yi, Benjamin Coleman, Chen Luo, Vihan Lakshman, and Anshumali Shrivastava. 2023. CAPS: A Practical Partition Index for Filtered Similarity Search. *arXiv preprint arXiv:2308.15014* (2023).

[16] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 47–57.

[17] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management - CIKM '13*. https://doi.org/10.1145/2505515.2505665

[18] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 9, 1 (2015), 1–12.

[19] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[20] Vihan Lakshman, ChoonHui Teo, Xiaowen Chu, Priyanka Nigam, Abhinandan Patni, Pooja Maknikar, and SVN Vishwanathan. [n. d.]. Embracing Structure in Data for Billion-Scale Semantic Product Search. ([n. d.]).

[21] Jie Li, Haifeng Liu, Chuanghua Gui, Jianyu Chen, Zhenyuan Ni, Ning Wang, and Yuan Chen. 2018. The Design and Implementation of a Real Time Visual Search System on JD E-commerce Platform. In *Proceedings of the 19th International Middleware Conference Industry*. https://doi.org/10.1145/3284028.3284030

[22] Sen Li, Fuyu Lv, Taiwei Jin, Guli Lin, Keping Yang, Xiaoyi Zeng, Xiao-Ming Wu, and Qianli Ma. 2021. Embedding-based Product Retrieval in Taobao Search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. https://doi.org/10.1145/3447548.3467101

[23] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2019. Approximate nearest neighbor search on high dimensional data—experiments, analyses, and improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2019), 1475–1488.

[24] Yiding Liu, Weixue Lu, Suqi Cheng, Daiting Shi, Shuaiqiang Wang, Zhicong Cheng, and Dawei Yin. 2021. Pre-trained Language Model for Web-scale Retrieval in Baidu Search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. https://doi.org/10.1145/3447548.3467149

[25] 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.
[26] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.
[27] Malteos. 2022. Aspect Paper Embeddings. https://huggingface.co/datasets/malteos/aspect-paper-embeddings.
[28] Javier Vargas Munoz, Marcos A Gonçalves, Zanoni Dias, and Ricardo da S Torres. 2019. Hierarchical clustering-based graphs for large scale approximate nearest neighbor search. *Pattern Recognition* 96 (2019), 106970.
[29] Priyanka Nigam, Yiwei Song, Vijai Mohan, Vihan Lakshman, Weitian (Allen) Ding, Ankit Shingavi, Choon Hui Teo, Hao Gu, and Bing Yin. 2019. Semantic Product Search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. https://doi.org/10.1145/3292500.3330759
[30] OpenAI. 2024. New embedding models and API updates. https://openai.com/blog/new-embedding-models-and-api-updates.
[31] Malte Ostendorff, Till Blume, Terry Ruas, Bela Gipp, and Georg Rehm. 2022. Specialized document embeddings for aspect-based similarity of research papers. In *Proceedings of the 22nd ACM/IEEE Joint Conference on Digital Libraries*. 1–12.
[32] Rodrigo Paredes and Edgar Chávez. 2005. *Using the k-Nearest Neighbor Graph for Proximity Searching in Metric Spaces*. 127–138. https://doi.org/10.1007/11575832_14
[33] Liana Patel. 2024. ACORN. https://github.com/stanford-futuredata/ACORN.
[34] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. 2024. ACORN: Performant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.
[35] Yun Peng, Byron Choi, Tsz Nam Chan, Jianye Yang, and Jianliang Xu. 2023. Efficient approximate nearest neighbor search in multi-dimensional databases. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–27.
[36] Casper Petersen, Jakob Grue Simonsen, and Christina Lioma. 2016. Power law distributions in information retrieval. *ACM Transactions on Information Systems (TOIS)* 34, 2 (2016), 1–37.
[37] Pinecone. 2019. The vector database to build knowledgeable AI. https://www.pinecone.io/.
[38] Navid Rekabsaz, Oleg Lesota, Markus Schedl, Jon Brassey, and Carsten Eickhoff. 2021. Tripclick: The Log Files of a Large Health Web Search Engine. In *Proceedings of the 44th International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2507–2513.
[39] Christoph Schuhmann, Richard Vencu, Romain Beaumont, Robert Kaczmarczyk, Clayton Mullis, Aarush Katta, Theo Coombes, Jenia Jitsev, and Aran Komatsuzaki. 2021. LAION-400M: Open Dataset of Clip-filtered 400 Million Image-text Pairs. *arXiv preprint arXiv:2111.02114* (2021).
[40] Scryfall. 2024. Scryfall Magic: The Gathering Card Search. https://scryfall.com/.
[41] SuhasJayaram Subramanya, Fnu Devvrit, HarshaVardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. *Neural Information Processing Systems,Neural Information Processing Systems* (Nov 2019).
[42] TrevorJS. 2024. Mtg Scryfall Cropped Art Embeddings. https://huggingface.co/datasets/TrevorJS/mtg-scryfall-cropped-art-embeddings-open-clip-ViT-SO400M-14-SigLIP-384.
[43] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, Kun Yu, Yuxing Yuan, Yinghao Zou, Jiquan Long, Yudong Cai, Zhenxiang Li, Zhifeng Zhang, Yihua Mo, Jun Gu, Ruiyi Jiang, Yi Wei, and Charles Xie. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*. Association for Computing Machinery, 2614–2627.
[44] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2024. An efficient and robust framework for approximate nearest neighbor search with attribute constraint. *Advances in Neural Information Processing Systems* 36 (2024).
[45] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2101.12631* (2021).
[46] 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. *arXiv preprint arXiv:2408.13899* (2024).
[47] Weaviate. 2022. Weaviate – Vector Database. https://weaviate.io/.
[48] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. AnalyticDB-V. *Proceedings of the VLDB Endowment* (Aug 2020), 3152–3165. https://doi.org/10.14778/3415478.3415541
[49] Wei Wu, Junlin He, Yu Qiao, Guoheng Fu, Li Liu, and Jin Yu. 2022. HQANN: Efficient and robust similarity search for hybrid queries with structured and unstructured constraints. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*. 4580–4584.
[50] Xiaoliang Xu, Chang Li, Yuxiang Wang, and Yixing Xia. 2020. Multiattribute approximate nearest neighbor search based on navigable small world graph. *Concurrency and Computation: Practice and Experience* (Dec 2020). https:

//doi.org/10.1002/cpe.5970

[51] Wen Yang, Tao Li, Gai Fang, and Hong Wei. 2020. PASE: PostgreSQL Ultra-High-Dimensional Approximate Nearest Neighbor Search Extension. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. https://doi.org/10.1145/3318464.3386131

[52] P.N. Yianilos. 1993. Data structures and algorithms for nearest neighbor search in general metric spaces. *Symposium on Discrete Algorithms,Symposium on Discrete Algorithms* (Jan 1993). https://doi.org/10.5555/313559.313789

[53] Penghao Zhao, Hailin Zhang, Qinhan Yu, Zhengren Wang, Yunteng Geng, Fangcheng Fu, Ling Yang, Wentao Zhang, and Bin Cui. 2024. Retrieval-Augmented Generation for AI-Generated Content: A Survey. *arXiv preprint arXiv:2402.19473* (2024).

[54] Weijie Zhao, Shulong Tan, and Ping Li. 2022. Constrained Approximate Similarity Search on Proximity Graph. (Oct 2022).

[55] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. 2023. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. *Proceedings of the VLDB Endowment* 16, 8 (2023), 1979–1991.

[56] Chaoji Zuo, Miao Qiao, Wenchao Zhou, Feifei Li, and Dong Deng. 2024. SeRF: Segment Graph for Range-Filtering Approximate Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–26.

Received April 2024; revised July 2024; accepted August 2024