DEG: Efficient Hybrid Vector Search Using the Dynamic Edge Navigation Graph

ZIQI YIN, Nanyang Technological University, Singapore
JIANYANG GAO, Nanyang Technological University, Singapore
PASQUALE BALSEBRE, Nanyang Technological University, Singapore
GAO CONG, Nanyang Technological University, Singapore
CHENG LONG, Nanyang Technological University, Singapore

Bimodal data, such as image-text pairs, has become increasingly prevalent in the digital era. The Hybrid Vector Query (HVQ) is an effective approach for querying such data and has recently garnered considerable attention from researchers. It calculates similarity scores for objects represented by two vectors using a weighted sum of each individual vector's similarity, with a query-specific parameter α to determine the weight. Existing methods for HVQ typically construct Approximate Nearest Neighbors Search (ANNS) indexes with a fixed α value. This leads to significant performance degradation when the query's α dynamically changes based on the different scenarios and needs.

In this study, we introduce the Dynamic Edge Navigation Graph (DEG), a graph-based ANNS index that maintains efficiency and accuracy with changing α values. It includes three novel components: (1) a greedy Pareto frontier search algorithm to compute a candidate neighbor set for each node, which comprises the node's approximate nearest neighbors for all possible α values; (2) a dynamic edge pruning strategy to determine the final edges from the candidate set and assign each edge an active range. This active range enables the dynamic use of the Relative Neighborhood Graph's pruning strategy based on the query's α values, skipping redundant edges at query time and achieving a better accuracy-efficiency trade-off; and (3) an edge seed method that accelerates the querying process. Extensive experiments on real-world datasets show that DEG demonstrates superior performance compared to existing methods under varying α values.

CCS Concepts: • **Information systems** → **Data management systems**; **Proximity search**.

Additional Key Words and Phrases: approximate nearest neighbor search, graph-based index, hybrid vector search

ACM Reference Format:
Ziqi Yin, Jianyang Gao, Pasquale Balsebre, Gao Cong, and Cheng Long. 2025. DEG: Efficient Hybrid Vector Search Using the Dynamic Edge Navigation Graph. *Proc. ACM Manag. Data* 3, 1 (SIGMOD), Article 29 (February 2025), 28 pages. https://doi.org/10.1145/3709679

1 Introduction

Nearest Neighbor Search (NNS) in high-dimensional Euclidean space has been a fundamental component of many applications, including databases [51, 66], information retrieval [72, 75], data mining [10], recommendation systems [58], and generative artificial intelligence (GAI) [71]. However, due to the curse of dimensionality [30, 69], existing NNS methods [32] often fail to meet

Authors' Contact Information: Ziqi Yin, ziqi003@e.ntu.edu.sg, Nanyang Technological University, Singapore; Jianyang Gao, jianyang.gao@ntu.edu.sg, Nanyang Technological University, Singapore; Pasquale Balsebre, pasquale001@e.ntu.edu.sg, Nanyang Technological University, Singapore; Gao Cong, gaocong@ntu.edu.sg, Nanyang Technological University, Singapore; Cheng Long, c.long@ntu.edu.sg, Nanyang Technological University, Singapore.

ACM 2836-6573/2025/2-ART29
https://doi.org/10.1145/3709679

practical efficiency requirements. To address this, Approximate Nearest Neighbor Search (ANNS) has been proposed, trading off some accuracy for significantly improved efficiency [39].

Over the past decades, various ANNS techniques have been proposed [1, 4, 21, 22, 24, 34, 56, 65, 67, 68], such as tree-based methods [1, 4, 56], hashing-based methods [65, 67], quantization-based methods [21, 22, 24, 34], and graph-based methods [18, 19, 26, 46, 47, 68]. Among them, graph-based methods provide superior performance compared to other methods [41, 68]. These methods construct a graph to index the dataset, where each node corresponds to an object in dataset and two nodes are associated with an edge if their distance satisfies some proximity property. At query time, a greedy search algorithm [19, 47, 68] is used to find approximate nearest neighbors. Most state-of-the-art graph-based ANNS indexes are constructed by approximating the Relative Neighborhood Graph (RNG) [63]. Its pruning strategy effectively removes some redundant edges based on edge length (distance between nodes), thereby achieving a better accuracy-efficiency tradeoff, as verified in [68] (details will be presented in Section 3.2).

In recent years, researchers started to explore various variants of ANNS queries [51], in order to handle more complex real-world scenarios. In particular, [66] studies the multi-vector query, where each object is described by multiple vectors, and the similarity score is computed by aggregating each individual vector's similarity score. For instance, intelligent video surveillance systems employ different vectors to represent the front face, side face, and posture of each individual recorded by the camera [2], which will subsequently be used for person identification.

In this paper, we investigate the Hybrid Vector Query (HVQ), which is one kind of multi-vector query, where each object involves two vectors. Specifically, an HVQ aims to retrieve the approximate nearest neighbors according to the weighted sum of the distances based on the two vectors, where the distance between vectors captures the dissimilarity between them and a query-specific parameter α determines the weight of each vector's distance (as to be formulated in Section 3.1). HVQ has many applications given the ubiquity of bimodal data in real-life scenarios, such as image-text data [54, 57, 59], spatio-textual data [9], and video-text data [2, 50]. In the case of geo-textual objects, one feature vector is derived from the object's textual description using language processing techniques, such as BERT [13], while the other feature vector represents its two-dimensional geographical coordinates. A HVQ query can be issued by a user looking for a 'Japanese sushi restaurant' near his/her position, with the parameter α capturing the query's preference over the trade-off between geographical proximity and semantic similarity. For example, machine learning techniques are used to learn a query-dependent weight α for a query [42], thereby better capturing the query's preference. Another example is in the intelligent video surveillance scenario [66], where each person can be represented by a face vector and a posture vector. This approach can enhance the accuracy of person recognition, where the weight of each vector could be different depending on the quality of each vector, such as the resolution of the image.

Existing methods and limitations. Two methods [66] have been proposed for the HVQ problem. Specifically, the first one constructs an ANNS index based on the weighted distance with $\alpha = 0.5$. In the query phase, it simply uses this index (which is built with $\alpha = 0.5$) for handling queries (which can have arbitrary values of α). The second one constructs separate ANNS indexes, one for each modality. During the query phase, it performs search on each index, retrieves similar objects in each modality, and then re-ranks all retrieved objects to obtain the approximate nearest neighbors. Graph-based ANNS indexes can be integrated with these two methods given their superior performance over other types of ANNS index. Additionally, when one of the feature vectors represents geographical coordinates, HVQ can function as the semantic-aware spatial keyword query [55]. The previous work on semantic-aware spatial keyword query [8, 55, 61] has encountered significant efficiency challenges, largely due to the curse of dimensionality of high-dimensional semantic vectors [30, 69] (as to be detailed in Section 2).

The two methods [66] both adopt the strategy of using a fixed α value (e.g., 0 or 0.5) during the index construction phase to build the index. **This is because existing ANNS indexes are constructed under the assumption that the distances between objects are certain and pre-determined.** For instance, graph-based ANNS indexes select neighbors for each node based on the edge lengths (the distances between objects), which assumes that the edge lengths are certain and pre-determined. Using a fixed α value during the indexing phase would help ensure that the distances between objects remain constant, thereby maintaining the proximity property in the ANNS indexes for the given α value. However, different queries can have different α values based on their needs and scenarios, the distances between objects would change as well, making the proximity property of the ANNS index ineffective and leading to severe performance degradation (as to be shown in Section 3.3).

Challenges. In this paper, we aim to develop a graph-based ANNS index that is capable of maintaining high performance for HVQ with varying α values. The key lies in how to construct a graph-based ANNS index that is capable of handling varying α values, which presents three challenges: **(1) How to compute the candidate neighbor set for each node.** Existing graph-based ANNS indexes typically acquire hundreds of approximate nearest neighbors for each node as the candidate neighbor set, avoiding considering all nodes in the dataset and thereby improving construction efficiency. However, as α changes, the distances between objects would change, and the approximate nearest neighbors of each node may vary significantly. Therefore, it becomes challenging to compute a candidate set for each node that comprises the approximate nearest neighbors for varying α values. **(2) How to determine edges from the candidate set.** A key idea of existing state-of-the-art graph-based ANNS indexes is to use the pruning strategy of Relative Neighborhood Graph (RNG) to prune some redundant candidate edges based on the edge lengths, thereby determining the final edges from the candidate set. According to the experimental evaluation [68], this pruning strategy can significantly improve search performance. However, for the HVQ problem, the α value varies at query time, making the edge lengths dynamic, and the RNG pruning strategy does not work. A straightforward solution is to fix an α value to prune edges during the index construction phase. However, as the query α varies, the RNG's property becomes ineffective, thereby leading to significant performance degradation (as to be shown in Section 3.3). How to design a new edge pruning strategy that can maintain the RNG's property under varying α values is an open problem. **(3) How to select the seed for the graph index in the HVQ problem.** According to [19], the start node (seed) of the graph impacts the search path length, which reflects the search efficiency. To reduce the search path length, [19] proposes to use the graph's approximate center as the seed. However, the center can vary significantly with different α values. An intuitive solution is to use multiple approximate centers for different α values, but this would degrade the search efficiency since some of the start nodes may not be good one for a particular α value.

Our method. In the first challenge, since α can be any arbitrary value in $[0, 1]$, finding the nearest neighbor of a given node for each possible α value becomes unfeasible. To address this, we propose to treat the problem as a multi-objective optimization problem, where the distance of each individual vector is considered as an objective function. Then, we use the Pareto frontier [45] as the candidate set, ensuring that the nearest neighbor is always within the Pareto frontier for varying α. However, in our context, finding the exact Pareto frontier for each node is expensive. To address this, we propose the Greedy Pareto Frontier Search (GPS) algorithm. By iteratively exploring the neighbor of neighbor and searching for Pareto frontiers within the small set, GPS efficiently finds high-quality approximate Pareto frontiers.

To handle the second challenge, we propose a novel dynamic edge pruning strategy. This strategy aims to maintain the property of the RNG for pruning at varying α values, as this property is

essential for enhancing the search performance [68]. To achieve this, we find that some edges would be pruned at certain α values based on the pruning strategy of RNG, while at other α values, they would be preserved. This motivates us to come up with the idea of assigning each edge a range called *active range*, covering the α values for that edge, within which it would not be pruned by the RNG's property. During the query phase, the edge is activated only if the query's specific α falls within its active range; otherwise, the edge would be ignored. This strategy ensures that for each query α value, the graph can dynamically prune edges based on the RNG's property, so the remaining graph formed by the activated edges satisfies the RNG's property, thereby ensuring high performance.

To tackle the issue of choosing appropriate seeds for the graph index, we propose a new edge seed method. This method uses nodes that are farthest from the center under varying α values as seeds, i.e., edge seeds. Since edge seeds are far from each other, at query time, the greedy search will automatically start from the seed node closest to the query and ignore distant seeds, thus avoiding the efficiency problem caused by multiple start nodes.

Based on these proposed techniques, we develop a novel Dynamic Edge Navigation Graph (DEG). During the index construction phase, DEG constructs the index by inserting nodes one by one, similar to previous methods [46, 47]. For each inserted node, DEG performs the GPS algorithm over the partially built graph to obtain the candidate neighbor set. Then it applies the dynamic edge pruning strategy to determine the edges from the candidate set. Finally, it checks whether each newly inserted node can update the edge seed set. At query time, DEG employs a variant of the greedy search algorithm, which dynamically skips some edges based on the query's α value and their active ranges.

The main contributions of this work are summarized as follows:

(1) We analyze the limitation of existing methods for the HVQ problem (Section 3.3). Specifically, these methods perform well for certain query α values but face significant performance degradation when the α value varies. This finding has not been reported in previous literature.
(2) We propose a new graph-based ANNS index called DEG to maintain high performance across varying α values. It comprises three technical contributions: (i) a greedy Pareto frontier search algorithm to compute a candidate neighbor set for each node, comprising the node's approximate nearest neighbors for varying α values; (ii) a dynamic edge pruning strategy that dynamically prunes edges at query time to maintain the property of the Relative Neighborhood Graph; and (iii) an edge seed method. To the best of our knowledge, this is the first ANNS index designed specifically for HVQ.
(3) We conduct extensive experiments on real-world datasets, which demonstrate that (i) DEG demonstrates the best performance compared to all baselines across different α settings; and (ii) DEG maintains high performance across different α settings without significant degradation.

2 Related Work

Approximate Nearest Neighbor Search. To address the ANNS problem, various methods [11, 19, 22, 24, 34, 43, 47, 48, 68] have been proposed, which can be classified into four categories: tree-based methods [1, 4, 56], hashing-based methods [11, 12, 20, 23, 28, 29, 38, 40, 43, 44, 60, 62, 64, 65, 67, 73], quantization-based methods [21, 22, 24, 34, 48], and graph-based methods [7, 18, 18, 19, 26, 27, 31, 41, 46, 47, 68]. According to experimental evaluations [41, 68], graph-based methods demonstrate superior performance compared with other methods. This is because other methods typically attempt to partition vectors into buckets and route queries to a small number of close buckets for fast retrieval, which is challenging in high-dimensional space [30, 69]. To the best of our knowledge, none of these ANNS techniques have been designed for the HVQ problem.

Graph-based ANNS Indexes. Most existing graph-based indexes [7, 18, 18, 19, 26, 27, 31, 41, 46, 47, 68] are derived from four fundamental types of graphs: Delaunay Graph (DG) [16], Relative Neighborhood Graph (RNG) [63], K-Nearest Neighbor Graph (KNNG) [53], and Minimum Spanning Tree (MST) [37]. According to the experimental evaluation [68], RNG-based ANNS indexes deliver state-of-the-art performance due to its pruning strategy.

Variants of ANNS Query. To address more complex real-world scenarios, several variants of the ANNS query have been proposed [51]. These variants of ANNS queries have been identified as a promising future research direction in recent vector database studies [25, 66]. For instance, [70] has introduced a hybrid ANNS query with attribute filtering to retrieve similar vectors that satisfy boolean predicates over their attributes. [66] introduces the multi-vector query, from which we define HVQ, and develops two solutions (as detailed in Section 3). However, both of these solutions have limitations in handling queries with different α (as shown in Section 3). To the best of our knowledge, none of the existing studies have identified or attempted to address these limitations, and answering multi-vector query is an open problem.

Hybrid Search. HVQ is a variant of hybrid search, which has been researched extensively. Existing studies aim to efficiently identify top-k objects ranked by monotone aggregation functions. The Threshold Algorithm (TA) and the No Random Access (NRA) algorithm [15] use pre-sorted lists for each attribute, but they face challenges with multi-dimensional data, as pre-computing distances and generating sorted lists is computationally expensive. To overcome this limitation, RR*-tree [17] uses reference objects to generate low-dimensional embeddings for each data object, which are then indexed by R-tree for efficient query processing. However, these approaches are limited to low-dimensional spaces and degrade to no better than a linear scan in high-dimensional settings.

There exist some attempts that investigate the integration of existing hybrid search techniques with hybrid vector search. Specifically, [6] develops a distributed system based on the Fusion baseline, employing a modified TA algorithm to filter vectors fetched from distributed machines. [66] uses the NRA algorithmto determine when the Fusion baseline should stop increasing k' in order to obtain exact top-k results. However, the experimental results [66] show that integrating NRA with the Fusion baseline has extremely high computational cost, and iteratively increasing the k' in the Fusion baseline is the most effective strategy.

There also exists recent work on exploring hybrid vector search for special cases, such as semantic-aware spatial keyword queries [8, 55, 61], where one attribute is a geographical coordinate and the other is a high-dimensional embedding. It takes geo-location and embedding as inputs to find top-k objects based on a weighted sum of geographical and embedding distances. However, these methods often focus on exact top-k results, and it is still an open problem to design efficient querying algorithms. Details on these methods are provided in the appendix[1] due to the page limitation.

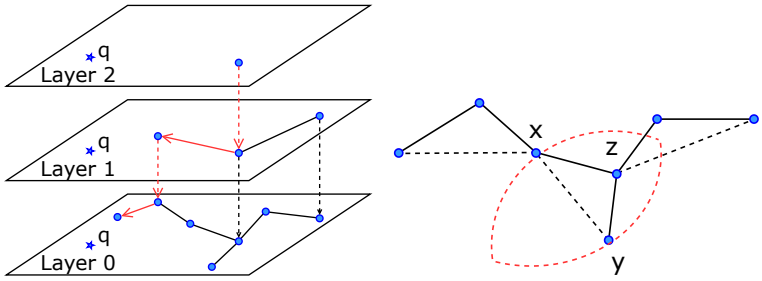
3 Preliminary and MOTIVATIONS

3.1 Problem Statement

We proceed to define the Hybrid Vector Query (HVQ).

Dataset. We consider a dataset D consisting of N objects. Each object $o \in D$ is characterized by two features: (1) one feature vector, denoted by $o.e$, defined in a d-dimensional Euclidean space E^d, where d is typically hundreds, and (2) the other feature vector, denoted by $o.s$, defined in an m-dimensional Euclidean space E^m, where m may vary from two to hundreds, depending on applications.

[1]https://github.com/Heisenberg-Yin/DEG/blob/main/SIGMOD2024_DEG_ready.pdf



(a) HNSW. (b) Pruning Strategy of RNG.

Fig. 1. Figure 1a illustrates the HNSW index. Figure 1b illustrates the pruning strategy of the Relative Neighborhood Graph (RNG).

Hybrid Vector Query (HVQ): Given a dataset D, a hybrid vector query $q = \langle e, s, \alpha, k \rangle$ consists of two feature vectors $q.e \in E^d$ and $q.s \in E^m$, a hyperparameter $q.\alpha \in [0, 1]$, and the number of objects to be returned $q.k$. HVQ aims to retrieve k objects with the minimum distance $Dist(q, o)$:

$$Dist(q, o) = q.\alpha \times \delta_e(q, o) + (1 - q.\alpha) \times \delta_s(q, o), \tag{1}$$

where $\delta_e(q, o) = \frac{\delta(q.e,\ o.e)}{e_{max}}$ and $\delta_s(q, o) = \frac{\delta(q.s,\ o.s)}{s_{max}}$. Here, $\delta(x, y)$ represents the Euclidean distance between vectors x and y, and e_{max} (resp. s_{max}) denotes the maximum Euclidean distance between $o.e$ (resp. $o.s$) of any two objects in the dataset and is used as a normalization factor. Following previous work [66], we use the weighted aggregated score as the similarity metric, and $q.\alpha \in [0, 1]$ is a parameter that allows to set preferences between the two vectors at query time.

Problem Statement. In this study, we aim to develop a graph-based ANNS index for the HVQ problem that maintains both accuracy and efficiency under different query α values.

Comparison of HVQ and Hybrid Queries. Several types of hybrid queries are related to HVQ, and we next discuss their differences. Early studies on hybrid queries [15, 17] typically assume that each object has multiple attributes, such as price, and aim to identify the top-k objects ranked by monotone aggregation functions, e.g., the sum of these attributes. These studies focus on numeric or low-dimensional attributes (e.g., geo-location) and aim to find the exact top-k results (as to be detailed in Section 2). Due to the curse of dimensionality [30, 69], they are not suitable for the HVQ problem. When one of the vectors is low-dimensional coordinates, HVQ can function as semantic-aware spatial keyword queries [55], which differ from traditional spatial keyword queries. However, existing methods for semantic-aware spatial keyword queries [8, 55, 61] often suffer from severe efficiency challenges (as to be detailed in Section 2).

3.2 Graph-based ANNS methods

According to experimental evaluations [41, 68], graph-based approaches have demonstrated superior performance compared to other ANNS techniques. They build a graph $G(V, E)$ where each node $x \in V$ corresponds to an object $o \in D$. The Hierarchical Navigable Small World graph (HNSW) [47], as illustrated in Figure 1a, is a state-of-the-art method. It comprises several layers, where layer 0 contains all objects, and each upper layer $i \geq 1$, randomly retains a subset of the objects from layer $i - 1$. Within each layer, a vertex is connected to several approximate nearest neighbors, while across adjacent layers, two vertices are connected only if they represent the same vector data.

At query time, the HNSW algorithm starts the search from the vertex in the top layer, as shown in Figure 1a. At each layer, the HNSW algorithm accesses all neighbors of the current vertex and

checks whether there exists a neighbor closer to the query than the current vertex. If so, it moves to the neighbor nearest to the query. This process continues within the layer until no closer neighbor to the query can be found. It then proceeds to the next layer and repeats the same procedure, using the current vertex as the starting vertex. This process continues until reaching layer 0. At layer 0, it conducts a greedy search algorithm, a method commonly used by graph-based ANNS indexes [18, 19, 26, 41, 46, 68]. Specifically, it maintains two sets, a candidate set $\mathcal{S}$ (a min-heap) and a result set $\mathcal{R}$ (a max-heap), where R stores the approximate nearest neighbors that have currently been found for the query, and S stores candidates that can potentially improve R. At each iteration, the object with the smallest distance in $\mathcal{S}$ is popped, and its neighbors are evaluated. If a neighbor has not been visited before and its distance from the query is smaller than the maximum distance in $\mathcal{R}$, the neighbor is added to $\mathcal{S}$ and $\mathcal{R}$. Here, new elements are inserted into $\mathcal{R}$ continuously. To avoid the high maintenance cost when $\mathcal{R}$ becomes large, which would reduce search efficiency, a hyperparameter ef_{search} is used to control its size. If the size of $\mathcal{R}$ exceeds ef_{search}, the object with the maximum distance in $\mathcal{R}$ is removed. The candidate set $\mathcal{S}$ is unbounded in size, as empirically it does not significantly reduce efficiency. The search stops and returns the k nearest objects in $\mathcal{R}$ when the minimum distance in $\mathcal{S}$ is larger than the maximum distance in $\mathcal{R}$. The hyperparameter ef_{search} controls the accuracy-efficiency trade-off at query time.

Clearly, the graph structure plays a key role in graph-based ANNS indexes, as verified in the work [68]. According to [68], state-of-the-art graph-based ANNS indexes (e.g., HNSW) are mostly constructed by approximating the Relative Neighborhood Graph (RNG) [63], and we next explain the RNG.

Relative Neighborhood Graph (RNG). The RNG $G(V, E)$ constructed on a dataset D has the following property: For $x, y \in V$, if x and y are connected by an edge $e \in E$, then for $\forall z \in V$, either $\delta(x, y) < \delta(x, z)$ or $\delta(x, y) < \delta(y, z)$. In other words, the longest edge of a triangle in RNG will be pruned. Note that the RNG property applies only to metric distances, as non-metric or non-linear distances do not satisfy the triangle inequality.

Figure 1b illustrates this property of RNG. The dashed edge (x, y) represents a potential edge and is pruned from RNG because it is the longest edge in the triangle (x, y, z). This pruning strategy removes some redundant neighbors and makes the neighbors distribute omnidirectionally, thereby reducing redundant search on the ANNS index [68]. For example, in the triangle (x, y, z), if all three edges are preserved, there exist two paths from x to z: $\{(x, z)\}$ and $\{(x, y), (y, z)\}$, which result in redundant calculations. According to the experimental evaluation [68], the pruning strategy of RNG can improve search performance significantly.

However, the time complexity of constructing the exact RNG on dataset D is $O(N^3)$ [33]. Therefore, many techniques [7, 18, 19, 27, 31, 41, 47] have been developed to approximate RNG and construct an ANNS index. Take HNSW as an example, which constructs RNG by continuously inserting nodes. For each node, the key is how to determine edges for each node. The neighbor selection strategy in HNSW consists of two steps, which is also widely used by other graph-based ANNS indexes [19, 27, 31, 46, 47]: (1) Obtaining $ef_{construction}$ approximate nearest neighbors as the candidate neighbor set, which avoids using all nodes in the graph as candidates, thus improving index construction efficiency. To obtain the candidate set, it performs a greedy search over the partially built graph; (2) Using the RNG's pruning strategy on the candidate set to prune some redundant candidate set and obtain the final M neighbors, thereby achieving a better accuracy-efficiency trade-off. Here, $ef_{construction}$ is a hyperparameter that controls the candidate set size, and M is a hyperparameter that determines the maximum number of neighbors per node.

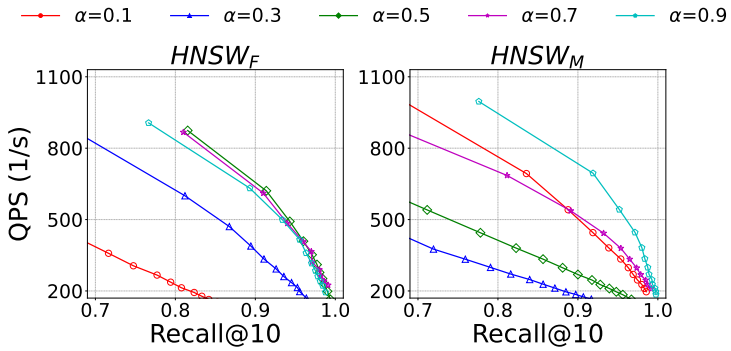


Fig. 2. The experiment results of HNSW_M and HNSW_F on the OpenImage dataset (up and right is better).

3.3 Motivations

Baselines. To the best of our knowledge, very little work has been done for the HVQ problem and previous work only presents two simple solutions by adapting existing ANNS indexes [66], which are detailed below:

- Fusion (abbr. F): This method builds an ANNS index based on the hybrid distance by fixing $\alpha = 0.5$ in Equation 1. During the query phase, it searches the built ANNS index to obtain the approximate result. The accuracy-efficiency trade-off is controlled by the search algorithm's hyperparameter (e.g., ef_{search}).
- Merging (abbr. M): This method builds an index for each feature vector separately. During the query phase, it issues a top-k' query for each query feature on the corresponding index, where $k' \geq k$. All the returned objects from every query feature are then reranked based on Equation 1 to find the approximate top-k results. The accuracy-efficiency trade-off is largely affected by the hyperparameter k'.

We integrate the Fusion method with the state-of-the-art ANNS index, HNSW [47]. This method is denoted as HNSW_F. For the Merging method, we use HNSW to build an ANNS index for each modality separately. When one of the feature vector's dimensions is low, we use the R-Tree [3] to index this modality and issue an exact top-k' query instead as the R-tree will perform better. This method is denoted as HNSW_M.

Limitations of Baselines. Both baselines share a common idea: fixing the α value (e.g., 0, 0.5, 1) to build the index during the construction phase. **This is because existing ANNS indexes are designed for the setting in which distances between objects are certain and predetermined.** Take graph-based ANNS indexes as an example, they typically selects approximate nearest neighbors for each node as the edge candidate set and pruning candidate edges based on edge length (distance between nodes). However, for the HVQ problem, as the query's α changes, the distances between objects may also change. The dynamic nature of distances in the HVQ problem exceeds the capabilities of existing ANNS techniques to handle, and a straightforward approach is to fix the value of α during the indexing phase.

The limitation of fixing α values to build ANNS indexes in the HVQ problem is that while this approach performs well for some query α values, it faces significant performance degradation for others. For example, HNSW_F constructs the index by fixing α at 0.5, ensuring that the edges satisfy the RNG's properties when α is 0.5. However, during the query phase, when $q.\alpha$ deviates significantly from 0.5, the properties of RNG quickly become ineffective, leading to a significant degradation in performance (as to be shown later). HNSW_M constructs separate indexes, each of which considers only one feature vector while neglecting the other. Therefore, each index can only retrieve similar objects within a modality. When $q.\alpha$ is around 0.5 during the query time,

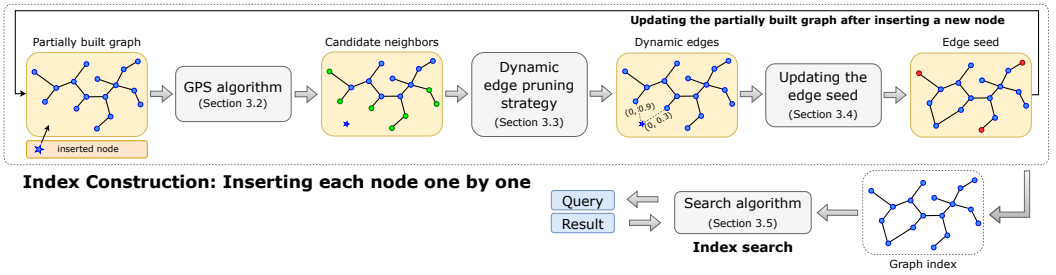


Fig. 3. The framework of the DEG, including the index construction phase and index search phase.

the separate indexes struggle to fetch high-quality candidates that are similar in both modalities for reranking, resulting in severe performance degradation (as to be shown later).

To illustrate the baselines' limitations, We evaluate HNSW_F and HNSW_M on the OpenImage dataset, which contains 500K images and textual descriptions. The textual content and images are transformed into embeddings using BERT [13] and ViT [14], respectively. The efficiency-accuracy trade-off results are shown in Figure 2 (hyperparameters details can be found in Section 5.1). We make the following observations: (1) HNSW_F performs the best when $q.\alpha$ is (e.g., 0.5) and degrades significantly when $q.\alpha$ deviates from 0.5 (e.g., 0.1 and 0.9); (2) HNSW_M performs better when $q.\alpha$ is close to 0 and 1 (e.g., 0.1 and 0.9) and degrades significantly when $q.\alpha$ is close to 0.5. This validates our analysis that adopting existing graph-based ANNS methods as baselines can perform well only for certain values of $q.\alpha$, but they exhibit significant performance degradation under different $q.\alpha$ values. These limitations motivate us to develop a new graph-based ANNS index that maintains high performance across varying α values.

4 The DEG Method

4.1 Overview

To overcome the limitations of baselines, and address the three challenges discussed in Section 1, we develop a new graph-based ANNS index, namely Dynamic Edge Navigation Graph (DEG), which includes three components, each addressing a challenge. We proceed to give an overview of the three components.

To tackle the first challenge of computing a candidate neighbor set for each node, we propose the GPS algorithm. The high-level idea of the GPS algorithm can be summarized in two points: 1) Finding the nearest neighbor of a node at any α value is equivalent to solving the problem of minimizing the distance function (Equation 1) at that α value. Finding solutions for each possible $\alpha \in [0, 1]$ is unfeasible. Then, we treat it as a multi-objective optimization problem, with each vector distance as an objective. To solve this, we compute the Pareto frontier as the solution, i.e., the neighbor candidate set, ensuring the nearest neighbor for any α value is included; 2) Finding the exact Pareto frontier is expensive. Therefore, the GPS algorithm aims to find approximate Pareto frontiers to reduce index construction costs. The core idea of the GPS algorithm is that a neighbor's neighbor is likely to be a neighbor. By iteratively exploring neighbors of neighbors, it continuously optimizes the approximate Pareto frontiers.

To address the second challenge of determining edges from the candidate set, we propose a dynamic edge pruning strategy that utilizes the RNG's pruning strategy to dynamically prune edges in DEG. Our key idea is as follows: 1) We assign an active range to each edge, covering the suitable α values for that edge, within which that edge will not be pruned by the RNG's property;

and 2) at query time, we use an edge for routing only if the query's α value intersects with its active range; otherwise, we ignore it. To achieve this, we take α into account in the pruning strategy of RNG and compute an active range for each edge such that for any α value, the remaining graph formed by the activated edges satisfies the RNG's property, thereby ensuring high performance.

To address the third challenge of finding the appropriate seed for the graph index, we propose a new edge seed method. This method uses nodes that are farthest from the center for varying α values as seed, i.e., edge seed. Compared to using multiple approximate centroids, edge seed ensures that they are far from each other. In greedy search, only edge nodes close to the query are activated, while others are ignored due to their large distance, thus avoiding the efficiency issue caused by multiple start nodes.

Figure 3 illustrates the framework of DEG, comprising the index construction phase and the index search phase. In the index construction phase, DEG builds the graph index by continuously inserting new nodes, similar to HNSW. It performs the GPS algorithm over the partially built graph to obtain candidate neighbors of the inserted node (shown as green nodes). Then, it utilizes the dynamic pruning strategy to prune some candidate edges, deriving the final dynamic edges (shown as dashed lines), where each edge is assigned an active range. Finally, it updates the edge seed (shown as red nodes). In the index search phase, the edge seed is used as the starting node, and a variant of the greedy search algorithm is employed, which dynamically skips some edges based on their active ranges and the query's α value.

In the rest of this section, we first present the three main components of DEG, namely (1) a greedy Pareto frontier search algorithm, called the GPS algorithm (Section 4.2); (2) a dynamic edge pruning strategy (Section 4.3); and (3) an edge seed method (Section 4.4). Then we present the search algorithm in Section 4.5.

4.2 Candidate Neighbor Acquisition

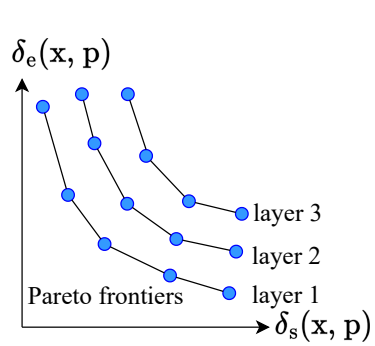
As previously discussed, finding the nearest neighbor of a node at any α value is equivalent to solving a multi-objective optimization problem. We formalize this problem as follows: Given an object $p \in D$, for other objects $x \in D \setminus \{p\}$, the multi-objective function is defined as $f(p, x) : x \rightarrow \mathbb{R}^2$, where $f_1(p, x) = \delta_e(p, x)$ and $f_2(p, x) = \delta_s(p, x)$. We propose using the Pareto frontier as the solution to this problem. Next, we introduce the concept of the Pareto frontier and explain why it is used as the candidate set.

Pareto Frontier [45]. Consider a dataset D and a multi-objective optimization problem described by $f(x) \colon x \rightarrow \mathbb{R}^c$, where $x \in D$. Each function $f_i(x) \rightarrow \mathbb{R}$ represents the objective function of the i-th task to be minimized, with $i \in C$ and $C = \{1, 2, \cdots, c\}$. For any $x, y \in D$, x dominates y if and only if (1) $\forall i \in C$, $f_i(x) \leq f_i(y)$ and (2) $\exists j \in C$, $f_j(x) < f_j(y)$. An object $x \in D$ is considered Pareto optimal if no other object in D dominates x. The Pareto frontier, also called the Pareto set, comprises all Pareto optimal objects in D.

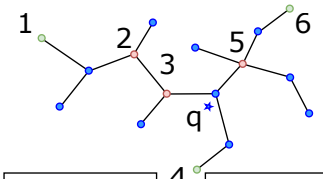
Figure 4a illustrates the Pareto frontier. For a node p, $\delta_s(x, p)$ and $\delta_e(x, p)$ denote the distance of each individual vector. The nodes within layer 1 lie at the bottom of the graph and constitute the Pareto frontier described above. Nodes in other layers will also be collected into the candidate neighbor set, as will be explained later. Next, we present the theorem that explains why the Pareto frontier can be used as the candidate set.

Theorem 4.1. *We denote the Pareto Frontier of the multi-objective function $f(p, x)$ as $PF(D, p) \subset D \setminus \{p\}$. For any $\alpha \in [0, 1]$, the nearest neighbor of p is contained in $PF(D, p)$.*

Due to page limitations, the proof is provided in the appendix. Therefore, we can compute $PF(D, p)$ for each object $p \in D$ as the candidate set, ensuring the nearest neighbors are always within $PF(D, p)$ when α varies. However, in a dataset with millions of objects, $PF(D, p)$ may contain only a dozen objects (e.g., 10), while we usually need hundreds of objects (e.g., 200) as the edge



(a) Pareto Frontier.



$\mathcal{S}_1$	
4	0.26
6	0.53
1	0.64

$\mathcal{S}_2$	
3	0.23
5	0.25
2	0.32

(b) Edge seed method.

Fig. 4. Figure 4a illustrates the Pareto frontiers and Figure 4b illustrates the difference between the edge seed acquisition method and the multiple centroids method.

candidates to select the final edges. To address this, we can choose multiple layers of $PF(D, p)$. Specifically, after finding the $PF(D, p)$, we remove the objects within $PF(D, p)$ from the dataset and search for a new $PF(D, p)$ until we obtain enough edge candidates. Figure 4a illustrates this, where layers 2 and 3 will also be collected as part of the candidate set.

The positive news is that for a given node p, with the distances $\delta_e(p, x)$ and $\delta_s(p, x)$ from other nodes to p already calculated, finding $PF(D, p)$ is equivalent to finding the two-dimensional skyline [5], which has efficient solutions [35, 36, 52, 74]. However, these methods cannot be efficiently applied to our problem directly.

Greedy Pareto Frontier Search. To address this, we introduce a novel Greedy Pareto Frontier Search algorithm (called GPS algorithm) for searching approximate Pareto frontiers on a partially built graph. The core idea of the GPS algorithm is that the neighbor of a neighbor is more likely to be a neighbor. It continuously explores the neighbors of neighbors and searches for the multi-layer Pareto frontier (skylines) within this small set. This approach allows us to obtain a high-quality approximate Pareto frontier and improve efficiency.

The detailed procedures of the GPS algorithm are summarized at Algorithm 1. It maintains a candidate set *Res* to store the discovered approximate Pareto frontiers, an indicator Vis to mark whether a node was added to *Res*, and another indicator *Flag* to mark whether a node's neighbors have been evaluated and added to *Res* (lines 2-4). If a node's neighbor is not evaluated, we call it a new node in *Res*. The GPS algorithm first adds all the start nodes (the seed) into *Res* as the initial candidates and marks them as visited (lines 5-7). Then it organizes these initial candidates into multi-layer Pareto frontiers using a modification of existing algorithm [5], referred to as *FindPF* (line 8). Specifically, *FindPF* repeatedly finds the Pareto frontier (skyline) within the candidate set *Res* using the existing algorithm [5], adds it to the result set, and removes it from the candidate set. When the result set reaches the size bound, it returns the first l layer Pareto frontiers that satisfy the size limit. Due to page limitations, the details of the algorithm are provided in the appendix. Next, the GPS algorithm iteratively performs a greedy search over graph G to optimize the discovered approximate Pareto frontiers (lines 9-23). Specifically, it scans each layer of the discovered approximate Pareto frontiers (line 11), collects the new nodes within that layer, denoted as $\mathcal{S}_k$ (line 12), and breaks if the set is not empty (lines 13-14). We only collect new nodes from the nearest Pareto frontier rather than from the entire set to avoid the neighbor explosion issue and improve efficiency. If no new nodes exist in *Res*, then we break the loop (lines 15-16). This is

Algorithm 1: GPS(G, q, ep, $ef_{construtionn}$)

Input: A partially built graph G, a query q, start node set ep, candidate pool size $ef_{construction}$
Output: Multi-layer Pareto frontiers $\{PF^1(q), \cdots, PF^l(q)\}$ for q, where $\sum_{i=1}^{l} |PF^i(q)| \leq ef_{construction}$

1 **Procedure** GPS($G, q, ep, ef_{construtionn}$):
2 $Res \leftarrow \emptyset$; // set of the pareto frontiers
3 $Vis \leftarrow \emptyset$; // set of the visited nodes
4 $Flag \leftarrow \emptyset$; // set of the explored nodes
5 **foreach** $v \in ep$ **do**
6 Res.add(v);
7 Vis.add(v);
8 $Res \leftarrow FindPF(Res, ef_{construtionn})$;
9 **while** $|Res| < ef_{construction}$ **do**
10 $NNS \leftarrow \emptyset$; // set of unexplored new nodes
11 **foreach** $S_k \in Res$ **do**
12 $NNS \leftarrow \{v \in \mathcal{S}_k \mid v \notin Flag\}$;
13 **if** $NNS \neq \emptyset$ **then**
14 **break**;
15 **if** $NNS = \emptyset$ **then**
16 **break**;
17 **foreach** $u \in NNS$ **do**
18 $Flag$.add(u) ; // mark u as old node
19 **foreach** $v \in neighbour(u)$ **do**
20 **if** $v \notin vis$ **then**
21 Res.add(v);
22 Vis.add(v);
// update the pareto frontiers using existing algorithm
23 $Res \leftarrow FindPF(Res, ef_{construtionn})$;
24 **return** Res;

because no new nodes within Res can be used to improve the results further. Otherwise, we add the neighbors of the new nodes to Res, mark these new nodes as old nodes, and mark their neighbors as visited (lines 17-22). After that, we optimize the candidate set Res by finding the multi-layer Pareto frontiers within Res using $FindPF$, and then updating Res accordingly (line 23).

Complexity Analysis. The dominant factor in the GPS algorithm's time complexity is the search path length, which determines the number of evaluated objects. In an extreme scenario where each layer of the Pareto frontier contains only one node, the GPS algorithm becomes equivalent to the greedy search algorithm with the longest search path length. In other cases, the GPS algorithm performs a greedy search for different α values. Given the bounded size of the candidate set, the search path length is shorter. Therefore, the time complexity of the GPS algorithm is $O(\log(N))$, the same as the greedy search.

Table 1. Examples.

Examples	$\delta_s(x,y)$	$\delta_e(x,y)$	$\delta_s(x,z)$	$\delta_e(x,z)$	$\delta_s(y,z)$	$\delta_e(y,z)$
(x_1,y_1,z_1)	0.3	0.4	0.8	0.9	0.1	0.7
(x_2,y_2,z_2)	0.5	0.7	0.2	0.4	0.3	0.5
(x_3,y_3,z_3)	0.2	0.6	0.4	0.5	0.3	0.4

4.3 Dynamic Edge Pruning Strategy

As discussed before, we aim to dynamically utilize the RNG's pruning strategy to prune edges. To achieve this, we take α into account the pruning strategy of the RNG, transforming the pruning condition of edge (x,y) into the following two formulas:

$$\alpha \times \delta_e(x,z) + (1-\alpha) \times \delta_s(x,z) < \alpha \times \delta_e(x,y) + (1-\alpha) \times \delta_s(x,y) \tag{2}$$

$$\alpha \times \delta_e(y,z) + (1-\alpha) \times \delta_s(y,z) < \alpha \times \delta_e(x,y) + (1-\alpha) \times \delta_s(x,y) \tag{3}$$

If both Equation 2 and Equation 3 hold for any $\alpha \in [0,1]$, then the edge (x,y) will be pruned due to the presence of node z according to the pruning strategy of RNG, as it becomes the longest edge in the triangle (x,y,z). However, the two equations may be satisfied for some values of α and not for others, which makes the pruning process challenging.

Example 1: Table 1 presents several examples to illustrate this challenge. In the first example, (x_1,y_1,z_1), Equation 2 does not hold for any α value. This means that the edge (x,y) will not be pruned due to the presence of node z according to RNG's pruning strategy. In the second example, (x_2,y_2,z_2), both equations are satisfied for any $\alpha \in [0,1]$, indicating that the edge (x_2,y_2) will be consistently pruned by node z_2, regardless of the α value. In the third example, (x_3,y_3,z_3), the first equation holds for $\alpha \in [\frac{2}{3},1]$ and the second equation holds for $\alpha \in [\frac{1}{3},1]$. This means that the edge (x_3,y_3) will be pruned due to the presence of node z when $\alpha \in [\frac{2}{3},1]$, as both conditions are satisfied in this range.

Example 1 shows that the RNG's pruning strategy is effective in some α cases, but not in others. We formalize this property into the following lemma.

Lemma 4.2. *If Equation 2 holds for $\alpha \in r_1^z$ and Equation 3 holds for $\alpha \in r_2^z$, where $r_1^z, r_2^z \subseteq [0,1]$, then the edge (x,y) will be pruned due to the presence of node z according to RNG's pruning strategy when $\alpha \in r_1^z \cap r_2^z$.*

Due to page limitations, the proof is provided in the appendix. Next, we demonstrate how to compute r_1^z and r_2^z. First, Equations 2 and 3 can be transformed into the following two formulas, respectively:

$$\alpha \times (\delta_e(x,z) - \delta_e(x,y) + \delta_s(x,y) - \delta_s(x,z)) < \delta_s(x,y) - \delta_s(x,z) \tag{4}$$

$$\alpha \times (\delta_e(y,z) - \delta_e(x,y) + \delta_s(x,y) - \delta_s(y,z)) < \delta_s(x,y) - \delta_s(y,z) \tag{5}$$

In Equation 4, the range of α that satisfies the inequality, denoted as r_1^z, is determined by the value and sign of the distance differences, which can be categorized into four cases based on their signs:

Case 1: If $\delta_s(x,y) - \delta_s(x,z) > 0$ and $\delta_e(x,z) - \delta_e(x,y) + \delta_s(x,y) - \delta_s(x,z) > 0$, then the inequality is satisfied for the range $r_1^z = \left[0, \min\left(1, \frac{\delta_s(x,y)-\delta_s(x,z)}{\delta_e(x,z)-\delta_e(x,y)+\delta_s(x,y)-\delta_s(x,z)}\right)\right]$.

Algorithm 2: Dynamic Edge Pruning Strategy

Input: A candidate set CS consisting of approximate Pareto frontiers, maximum edges M, a threshold value th to prune edges with a small use range
Output: Neighbor Set NS

```
1  Procedure DRNGPrune(CS, M, th):
2      Initialize the Neighbor Set NS = ∅;
3      foreach PF_i ∈ CS do
4          foreach x ∈ PF_i do
5              r^x ← ∅;
6              foreach y ∈ NS do
7                  Compute r^y;
8                  r[x] ← r[x] ∪ r^y;
9              u ← [0, 1] \ r[x] ;
10             if |u| ≥ th then
11                 NS.add(x, u);
12         if |NS| ≥ M then
13             break;
14     return NS;
```

Case 2: If $\delta_s(x,y) - \delta_s(x,z) < 0$ and $\delta_e(x,z) - \delta_e(x,y) + \delta_s(x,y) - \delta_s(x,z) \geq 0$, then the inequality cannot be satisfied for any $\alpha \in [0,1]$, resulting in $r_1^z = \emptyset$.
Case 3: If $\delta_s(x,y) - \delta_s(x,z) > 0$ and $\delta_e(x,z) - \delta_e(x,y) + \delta_s(x,y) - \delta_s(x,z) \leq 0$, then the inequality can be satisfied for any $\alpha \in [0,1]$, resulting in $r_1^z = [0,1]$.
Case 4: If $\delta_s(x,y) - \delta_s(x,z) < 0$ and $\delta_e(x,z) - \delta_e(x,y) + \delta_s(x,y) - \delta_s(x,z) < 0$, then the inequality is satisfied for the range $r_1^z = \left[\min\left(1, \frac{\delta_s(x,y)-\delta_s(x,z)}{\delta_e(x,z)-\delta_e(x,y)+\delta_s(x,y)-\delta_s(x,z)}\right), 1\right]$.
Example 2: The example (x_1, y_1, z_1) falls under case 2, resulting in $r_1^z = \emptyset$. The example (x_2, y_2, z_2) falls under case 3, leading to $r_1^z = [0,1]$. The example (x_3, y_3, z_3) falls under case 4, $r_1^z = [\frac{2}{3}, 1]$.

For Equation 5, such range can computed similarly, denoted as r_2^z. Thus, the pruning range of edge (x,y) due to the presence of node z is the intersection of r_1^z and r_2^z, denoted as $r^z = r_1^z \cap r_2^z$. The active range u of edge (x,y) for node z is then the complement of r^z, given by $u = [0,1] \setminus r^z$.
DEG's Pruning Strategy. Using the approximate Pareto frontiers derived by the GPS algorithm as the candidate set CS, we can apply this dynamic edge pruning strategy to it. As shown in Algorithm 2, we first initialize the neighbor set NS as an empty set (line 2). Then we sequentially traverse each layer of the Pareto frontier from nearest to farthest, gradually adding nodes to the NS (lines 3–13). Specifically, for each new node x, we compute its pruning range r^y with respect to each previously added node y and take their union $r[x]$ as the final pruning range (lines 6–8), ensuring that the new edge will not be pruned by any previously added edges. The active range u for this edge is the complement of $r[x]$ (line 9). To avoid maintaining edges that are not useful in most cases (e.g., $u = [0, 0.05]$), we set a threshold value th to further prune such edges (lines 10–11). Once we obtain M edges, we break the loop and return NS as the final neighbor set (lines 12–13).

However, our pruning strategy does not apply to multi-vector queries with more than two vectors, as the active range becomes a hyperplane in 2D space in these cases, which is challenging to compute, store, and utilize. We leave this for future work.

Lemma 4.3. *By considering all objects in the dataset as candidate neighbors for each node and applying the dynamic edge pruning strategy to obtain dynamic edges, the constructed graph becomes a*

Relative Neighborhood Graph [19] for any α value. This ensures that the nearest neighbor can always be found for the query using the greedy search algorithm.

Due to page limitations, the proof is provided in the appendix.

4.4 Index Construction

Edge Seed Acquisition. Another challenge discussed in Section 1 is how to choose the start nodes (also known as the seed) for the graph index. Existing graph-based ANNS methods, such as HNSW [47], randomly select a subset of nodes as the seed set, which forms the upper layers. To reduce search path length and improve search efficiency, one approach [19] sets the node closest to the graph's center as the seed. However, for the HVQ problem, as α changes, using a single centroid or a small number of random nodes as the seed results in reduced performance (as to be shown in Section 5.2.6). A straightforward method to maintain performance is to sample more random nodes or choose multiple centroids as seeds for different α values, but this reduces the efficiency during the query phase due to the multiple start nodes.

Figure 4b illustrates this issue, where the red nodes represent the potential multiple centers. At query time, the greedy search algorithm used by graph-based ANNS indexes maintains two sets: a min-heap candidate set $\mathcal{S}$ and a max-heap result set $\mathcal{R}$, as detailed in Section 3.2. The algorithm iteratively fetches the object with the minimum distance in $\mathcal{S}$ and adds its neighbors into $\mathcal{S}$ and $\mathcal{R}$. When we use multiple centroids as seeds, they all have similar distances to the query, as shown in Figure 4b by $\mathcal{S}_2$. This can lead to multiple parallel searches, causing many intermediate nodes to be visited repeatedly, thus reducing efficiency.

To address this, we propose a novel edge seed acquisition method. The core idea of this method is to choose the nodes farthest from the center under varying α value as the seeds, which are located at the edge of the graph. Figure 4b illustrates this method, where the green nodes indicate the edge seed. These edge seeds are much farther from the query. Therefore, during the greedy search process, these distant seeds will remain at the bottom of the candidate set $\mathcal{S}_1$, and their neighbors will not be considered and evaluated. From a high-level perspective, the edge seed method allows us to adaptively start the search from the edge node closest to the query while ignoring other distant edge nodes.

To obtain the edge seed, we employ an efficient yet highly effective method. Specifically, we first calculate the centroid c of D, where $c.e = \text{avg}(x.e)$ and $c.s = \text{avg}(x.s)$ for all $x \in D$. Then we maintain the inverse Pareto frontier of the centroid c, which consists of the nodes that do not dominate any other nodes based on the distance from the centroid, i.e., the most distant nodes from the centroid under varying α. The algorithm can be easily adapted from existing algorithm for finding two-dimensional skyline [5], so the details are not provided here. The time complexity of updating the seed set ep is $O(|ep| \log(|ep|) + |ep|)$ since we only need to maintain one layer of inverse Pareto frontier. The inverse Pareto frontier often contains only a dozen nodes, so the time complexity for updating is negligible.

Summary. Based on the modules proposed above, we summarize the index construction process. The details of the index construction phase are provided in Algorithm 3. Specifically, we first calculate the centroid c of D (line 2). Then we initialize the start node as the first node x_0 (line 3) and construct the graph by iteratively inserting new nodes (lines 4-11). For each newly inserted node x, we employ the GPS algorithm to search for approximate Pareto frontiers (line 6) and obtain the final neighbor set $NS(x)$ of node x among the approximate Pareto frontiers using Algorithm 2 (line 7). The derived neighbor set $NS(x)$ is treated as x's edges in G (line 8). Similar to previous studies [47], we also attempt to add reverse edges by trying to include x in $E(y)$, where $y \in NS(x)$ (lines 9-10). Finally, we update the edge seed set by determining whether x can be added to the inverse Pareto frontier of the centroid (line 11).

Algorithm 3: Index Construction

Input: A dataset D, candidate pool size $ef_{construction}$, maximum edges per node M, a threshold value th to prune edges with a small use range
Output: $G(V, E, ep)$, where ep denotes the seed set

Procedure DEGBuild(D, $ef_{construction}$, M, th):
  calculate the centroid c of D;
  Initialize graph $G(V, E, ep)$, $V = \{x_0\}$, $E = \emptyset$, $ep = \{x_0\}$;
  foreach $x \in D \setminus \{x_0\}$ **do**
    $V \leftarrow V \cup \{x\}$;
    $CS \leftarrow \texttt{GPS}(G, x, ep, ef_{construction})$;
    $NS(x) \leftarrow \texttt{DRNGPrune}(CS, M, th)$;
    $E(x) \leftarrow NS(x)$;
    foreach $y \in NS(x)$ **do**
      $E(y) \leftarrow \texttt{DRNGPrune}(E(y) \cup \{x\}, M, th)$;
    update ep as the inverse Pareto frontier of c;
  return G;

4.5 Search Algorithm

We next present the search algorithm of DEG. It is based on the greedy search algorithm [68], with two modifications. Firstly, it dynamically skips edges whose active ranges do not intersect with the query's α value. Secondly, we introduce an early stopping mechanism. When calculating the hybrid distance, we first compute one of the individual vector distances and compare it to the threshold distance in the greedy search algorithm to determine whether to skip this node early.

Specifically, the algorithm first initializes an empty min-heap $\mathcal{S}$ as the candidate set, an empty max-heap $\mathcal{R}$ as the result set. Next, the seed are added to $\mathcal{S}$, $\mathcal{R}$. In each iteration, the object with the smallest distance in $\mathcal{S}$ is fetched. If its distance to the query is larger than the maximum distance in $\mathcal{R}$, the loop breaks, and the top k objects in $\mathcal{R}$ are returned. Otherwise, its neighbors are checked, with some being skipped based on the active ranges and the query α value. If the distance of a neighbor to the query is smaller than the maximum distance in $\mathcal{R}$, the neighbor is pushed into $\mathcal{S}$ and $\mathcal{R}$. Here, the early termination mechanism is used for acceleration. If the size of $\mathcal{R}$ exceeds ef_{search}, the object with the maximum distance is popped from $\mathcal{R}$.

Complexity Analysis. Here, we analyze the time and space complexity of DEG. For space complexity, the main difference between DEG and previous graph-based ANNS indexes is the active range u stored for each edge. Compared to the high-dimensional vectors stored in memory, this additional storage is negligible. As for time complexity, during the query phase, DEG dynamically skips some edges, while the remaining edges satisfy the RNG property. Therefore, the search time complexity remains the same as that of previous RNG-based ANNS indexes [68], which is $O(\log(N))$.

5 Experiments

5.1 Evaluation Setup

Datasets. Our experiments are conducted on five real-world datasets: OpenImage, Ins-SG, Howto100M, CC3M, and Twitter-US. The statistics of datasets are listed in Table 2. Details of each dataset are stated as follows.

Table 2. Datasets Statistics.

Dataset	$\|D\|$	d	m	$\|Q\|$	Type
OpenImage	507,444	768	768	1,000	Text, Image
Ins-SG	1,000,000	768	2	1,000	Text, Coordinate
Howto100M	1,238,875	1,024	768	1,000	Text, Video
CC3M	3,131,153	768	768	1,000	Text, Image
Twitter-US	10,000,000	768	2	1,000	Text, Coordinate

- OpenImage [54]: The OpenImage dataset[2] is an open benchmark for object detection, image classification, and visual relationship detection. It comprises 500K training images and 41K validation images. Each image is paired with localized narratives provided by annotators. Images are converted into 768-dimensional vectors ($o.e$) using ViT [14], and localized narratives are transformed into 768-dimensional vectors ($o.s$) using BERT [13]. The training set is used as the database. The query set Q ($|Q|$ = 1,000) is randomly selected from the validation set.
- Ins-SG: The Ins-SG dataset is a real dataset that contains 1 million Instagram posts from Singapore. Each post has a geo-location and image. The images are transformed into vectors using ViT. The query set Q consists of another 1,000 collected posts.
- Howto100M [50]: The Howto100M dataset[3] is an open benchmark for learning text-video embeddings. It includes 136 million video clips, covering 23,000 activities in areas such as cooking, handcrafting, and fitness. Each video is paired with subtitles automatically downloaded from YouTube. We downloaded S3D [49] video embeddings from its official website and transformed textual descriptions into embeddings using BERT. We randomly sampled 1,000 video-text pairs to form the query set Q.
- CC3M [59]: The Conceptual Captions dataset[4] is an open benchmark for training and evaluating visual-language models. It comprises 3.3M image-text pairs. randomly selected 1,000 image-text pairs as the query set, and treated the remaining image-text pairs as the database. Each Image and its attached text are transformed into vectors using ViT and BERT, respectively.
- Twitter-US: The Twitter-US dataset is generated from 10 million real tweets in the USA. All of them contain geo-locations and text descriptions. The text descriptions are transformed into vectors using BERT. The query set Q consists of 1,000 real tweets collected together with the data. This dataset is an order of magnitude larger than those used by other memory-based graph ANNS index studies [68]. Therefore, we conduct the scalability study on this dataset.

Evaluation metrics. Exiting ANNS studies [41, 68] typically use recall rate $Recall@k = \frac{\mathcal{R} \cap \tilde{\mathcal{R}}}{k}$ to evaluate the accuracy of search results and queries per second QPS = $\frac{\#q}{t}$ to evaluate the search's efficiency. Here, $\mathcal{R}$ represents the result set retrieved by the index, $\tilde{\mathcal{R}}$ denotes the result set returned by a brute-force search, and $|\mathcal{R}| = |\tilde{\mathcal{R}}| = k$. QPS is the ratio of number of queries ($|Q|$) to search time (t); i.e., QPS= $\frac{|Q|}{t}$ [19]. In this work, we use recall@10 and QPS as the evaluation metrics.

Baselines. In addition to the two existing baselines discussed in Section 3.3, we consider a baseline called Overlay, and an ideal method called Oracle [5].

- Overlay (abbr. O): This method constructs five different graph-based ANNS indexes by setting α in the hybrid distance to 0.1, 0.3, 0.5, 0.7, 0.9, respectively. These separate graph-based indexes are then overlaid into a single graph-based index by merging their edges accordingly, with each

[2]https://google.github.io/localized-narratives/

[3]https://www.di.ens.fr/willow/research/howto100m/

[4]https://github.com/google-research-datasets/conceptual-captions

[5]We would like to thank the anonymous reviewers for suggesting the two methods

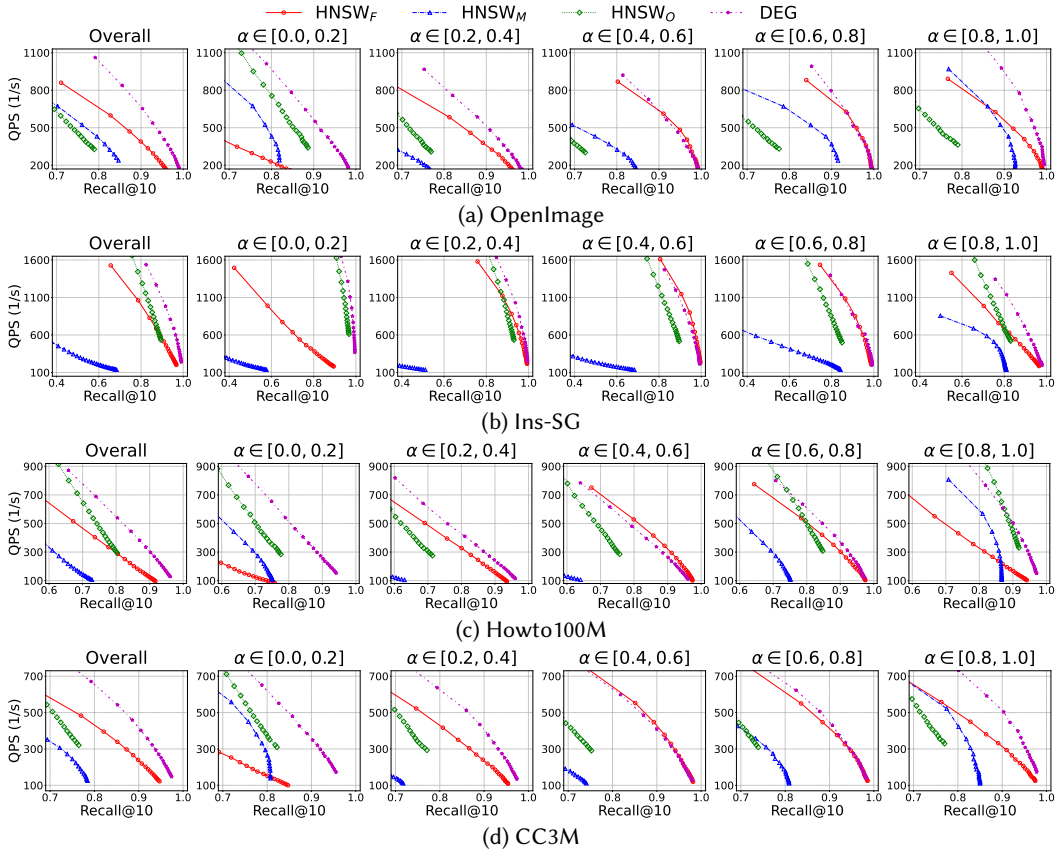
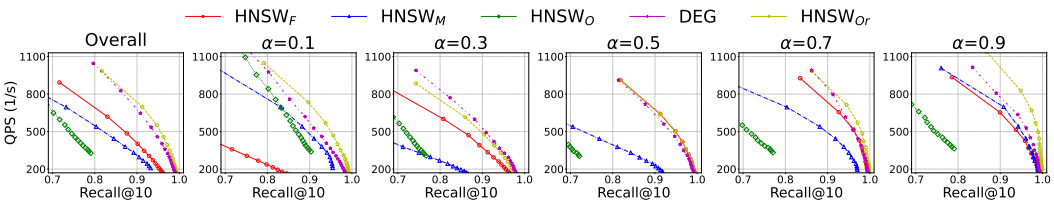


Fig. 5. The accuracy-efficiency trade-off results (upper and right is better).

edge assigned a value representing the α under which it was constructed. During the query phase, rather than traversing all edges, it only routes the edges with values closest to the query's α and ignores others. This actually restricts the search to the corresponding sub-index, i.e., we use the sub-index built with $q.\alpha = 0.1$ to handle queries with $\alpha \in [0, 0.2]$, the sub-index with $\alpha = 0.3$ for queries with $q.\alpha \in [0.2, 0.4]$, etc. The overlay operation eliminates the need to store five separate indexes, thereby reducing memory usage. The accuracy-efficiency trade-off is controlled by the search algorithm's hyperparameter (e.g., ef_{search}). We integrate this method with the HNSW index, denoted as HNSW$_O$.

- Oracle (abbr. Or): This method represents the ideal scenario where a separate graph-based ANNS is built for every possible α value, with searches conducted on the corresponding index. Although this method is not feasible in practical scenarios since $q.\alpha$ is unknown beforehand and can be an arbitrary value within $[0, 1]$, we use it to illustrate the performance gap between our proposed method and the ideal case. Specifically, we implement this approach using the HNSW index, denoted as HNSW$_{Or}$. To make the comparison feasible, we select five $q.\alpha$ values (0.1, 0.3, 0.5, 0.7, and 0.9), constructing a separate HNSW index for each, with $M = 40$ and $ef_{construction} = 200$. These five α values are used to generate five test query sets. We compare DEG with HNSW$_{Or}$ and other baselines on these query sets.

Parameter Settings. The three key parameters of HNSW, namely the candidate set size $ef_{construction}$, the maximum number of edges per node M, and the search set size ef_{search}, are set to 200, 40, and 10 by default for baselines HNSW$_F$ and HNSW$_M$, with other parameters set as recommended in the previous study [47]. For baseline HNSW$_O$, if the maximum number of edges per sub-index



(a) OpenImage

Fig. 6. The accuracy-efficiency trade-off results compared to HNSW$_{Or}$ (upper and right is better).

node is set to be M, the construction time and memory costs can be up to five times higher than other methods, leading to unfair comparison. Therefore, we set the maximum number of edges per node in each sub-index to 10 by default, so that each node in the overlaid HNSW index has a maximum of 50 edges, and thus the total number of edges in the overlaid graph index is comparable to others. Accordingly, we set the default value of $ef_{construction}$ for each sub-index to 50, which is five times the value of M, as applied previously. All other parameters for HNSW$_O$ are set the same as other baselines. Experiments on varying M and $ef_{construction}$ will be presented later in Section 5.2.5. For HNSW$_F$ and HNSW$_O$, we vary the hyperparameter ef_{search} from 10 to 200 in steps of 10 to control the accuracy-efficiency trade-off. For HNSW$_M$, we vary the hyperparameter k', the number of objects to be retrieved from each index, and then merged and reranked, from 10 to 200 in steps of 10 to control the accuracy-efficiency trade-off, with ef_{search} always set equal to k'. By default, we set M to 40 and $ef_{construction}$ to 200 for DEG, aligning with HNSW$_F$. When HNSW$_F$'s hyperparameters are adjusted, DEG is modified to the same value accordingly to ensure consistency and fair comparison unless otherwise specified. The threshold value th is consistently set to 0.1. The parameter α in Equation 1 allows to set preferences between different modalities. To evaluate the capability of methods to adapt to different query α values, we divide the α range into five intervals: [0, 0.2], [0.2, 0.4], [0.4, 0.6], [0.6, 0.8], and [0.8, 1]. We generate five random α values for each test query, one from each interval, producing five different query sets. Each set includes all test queries with α values from one specified interval. We report the overall average results of all sets for a comprehensive comparison. We also report the average results for each set for a detailed comparison.

Implementations. The baselines and DEG[6] are all implemented in C++. The implementations of HNSW[7] and R-tree[8] are sourced from publicly available code repositories. DEG and the baselines are implemented by following previous experimental evaluations[5], excluding SIMD, pre-fetching, and other hardware-specific optimizations. Although these optimizations can speed up index construction significantly, they are hardware-specific and could introduce unfairness in comparison. Our default experimental environment consists of an AMD Ryzen Threadripper PRO 5965WX CPU @ 7.00 GHz and 128GB of memory.

5.2 Experimental Results

5.2.1 ***Search Performance.*** The accuracy-efficiency trade-off results over the four datasets are shown in Figure 5. We have the following observations.

(1) DEG demonstrates the best performance compared to the baselines on all the datasets. Specifically, compared to the baselines, DEG consistently achieves the best overall performance across the four datasets. Additionally, across different α settings, DEG consistently delivers the best performance. For example, in the OpenImage dataset, among the baselines, HNSW$_F$ performs

[6]The code is available at https://github.com/Heisenberg-Yin/DEG.
[7]https://github.com/Lsyhprum/WEAVESS/
[8]https://github.com/nushoin/RTree

better when $\alpha \in [0.2, 1]$ and HNSW_O excels when $\alpha \in [0, 0.2]$. Our proposed DEG matches the performance of HNSW_F for $\alpha \in [0.4, 0.6]$ and outperforms the best baseline in all other query α settings. It is worth noting that DEG aims to maintain high performance across different α values rather than outperforming existing state-of-the-art graph-based ANNS indexes. It is as expected that DEG has similar performance as HNSW_F when $q.\alpha$ is close to 0.5 because HNSW_F is constructed with $\alpha = 0.5$.

(2) The baselines perform well in certain $q.\alpha$ settings but poorly in others, while DEG consistently achieves high performance across all $q.\alpha$ settings without significant degradation. The results show that when $q.\alpha$ is close to 0.1, HNSW_F faces significant performance degradation across the four datasets. Similarly, HNSW_M shows comparable degradation across the four datasets when α is close to 0.5. These results validate our analysis in Section 3.3. A similar pattern is observed with HNSW_O, which performs better at $\alpha \in [0, 0.2]$ on the OpenImage, Ins-SG, and CC3M datasets, and excels at $\alpha \in [0.8, 1.0]$ on the Howto100M dataset. This pattern may occur because when $q.\alpha$ is close to 0 or 1, the search becomes dominated by a single feature vector. If this modality's feature vector is easier for the graph-based ANNS to capture, the performance will improve. This reasoning is also supported by the performance of HNSW_M. In settings where HNSW_O performs better, HNSW_M also shows relatively better performance. For example, on the OpenImage and CCM datasets, for α in $[0, 0.2]$, HNSW_M outperforms HNSW_F, but performs worse in other α settings. The Ins-SG dataset is an exception and the reason for this will be explained later. Moreover, we also evaluate the performance of DEG when $\alpha = 0$ or 1, which reduces to normal vector queries. Experimental results show that DEG still delivers the best performance compared to the baselines for HVQ. Detailed results are provided in the appendix due to page limitations.

(3) DEG maintains similar advantages on larger datasets. Specifically, the CC3M dataset employs the same embedding techniques as the OpenImage dataset, but it is much larger. DEG shows similar advantages over the baselines on both datasets, validating that DEG also performs well on larger datasets.

(4) HNSW_M performs worse on the Ins-SG datasets. As shown in Figure 5b, HNSW_M shows significantly worse performance than HNSW_F when $\alpha = 0.1, 0.3, 0.5, 0.7$ on the Ins-SG dataset. This is because the HNSW_M struggles to retrieve high-quality candidates for reranking. Due to the inherently dense distribution of geographic coordinates compared to high-dimensional vectors, hundreds of objects can coexist within a small spatial scale, making geographically close objects difficult to distinguish from each other, and requiring embedding similarity to further determine the ranking results. However, both the R-Tree and HNSW used in HNSW_M fails to retrieve objects that exhibit similarity across both modalities, thereby resulting in notable performance degradation. It is worth mentioning that the throughput of R-Tree and HNSW is comparable, with 1,777 and 1,906 queries per second, respectively, when $k' = 10$. Therefore, the relatively low performance of HNSW_F is not due to HNSW having a slower querying speed.

5.2.2 ***Performance Gap Analysis with HNSW_{Or}.*** Here, we compare DEG with the HNSW_{Or} and other baselines on the OpenImage dataset for $\alpha = 0.1$, 0.3, 0.5, 0.7, and 0.9. The experimental results are shown in Figure 6. **The results demonstrate that the overall performance of DEG is comparable to that of HNSW_{Or} and significantly outperforms other baselines.** This suggests that the performance of DEG is on par with HNSW_{Or}.

5.2.3 ***Scalability Study.*** Here, we investigate the index construction cost and search performance of our proposed DEG and the baselines on the Twitter-us dataset. The index construction time for DEG is 44,492 seconds, approximately 12 hours. However, neither of the baselines completed the index construction within two days. This is consistent with the experimental results of [19], where HNSW could not be constructed on larger datasets and raised an Out-Of-Memory error. We infer

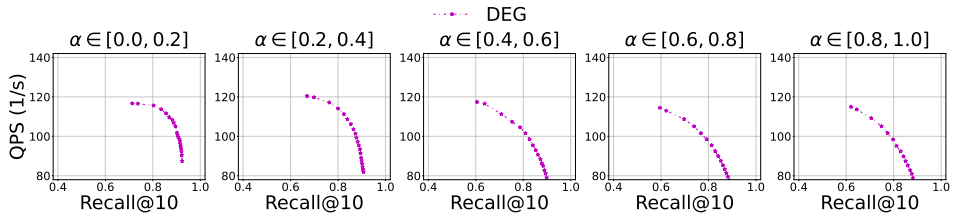
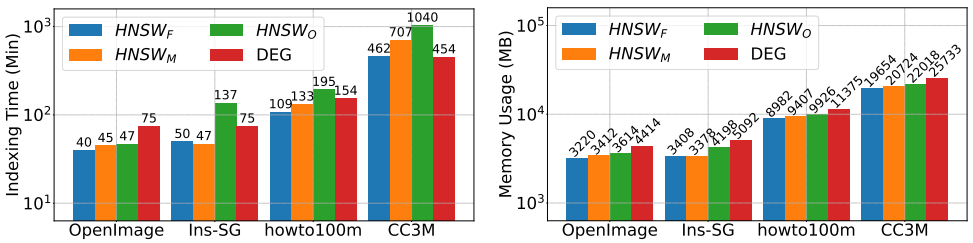


Fig. 7. Scalability study.

that this is due to HNSW's multi-layer mechanism, which causes its construction cost to increase exponentially with the size of the dataset. Therefore, we only report the results for DEG, which are shown in Figure 7. The results show that DEG exhibits stable performance for varying α, consistent with our observations in previous experiments.

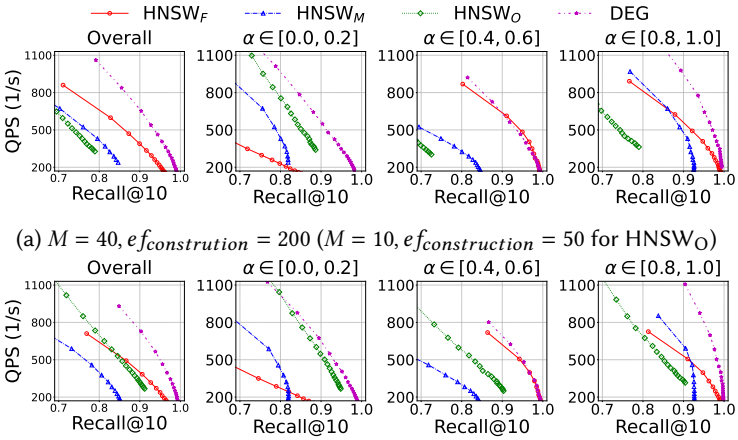
5.2.4 ***Index Cost.*** Figure 8a illustrates the construction time of DEG and the baselines on the four datasets with default parameter settings. **The results show that DEG's indexing time is comparable to those of HNSW_M and HNSW_F, and is significantly faster than that of HNSW_O.** For instance, on the largest dataset CC3M, DEG has a comparable time cost to HNSW_F, while being 1.6 times faster than HNSW_M and 2.3 times faster than HNSW_O. HNSW_M and HNSW_O are slower because they build multiple indexes. This validates our analysis in Section 4.5 that the DEG's construction time is comparable to previous graph-based ANNS indexes. On the Ins-SG dataset, HNSW_M has a slightly shorter construction time than HNSW_F. The reason is two-fold: (1) HNSW_M builds indexes separately, computing distances for individual vectors rather than hybrid vectors, resulting in similar construction times across datasets; (2) The construction of R-Tree is faster, leading to slightly quicker times on the Ins-SG dataset but slightly slower on others. Note that the high construction time is a result of our setting, as discussed in Section 5.1, where hardware-specific optimizations such as SIMD and pre-fetching instructions have been removed. With these optimizations, the construction time for million-scale datasets can be reduced to several minutes. Figure 8b shows the memory usage of DEG and the baselines. **The results show that DEG's memory usage is slightly higher than baseline methods due to its more complex index structure.** However, the high-dimensional vectors still dominate the memory consumption, making the difference negligible in its real-world application. Notably, maintaining identical construction time or memory consumption between baselines and DEG for a fair comparison is impractical. Therefore, we provide further comparisons in the appendix.



(a) Construction Time. (b) Index Size.

Fig. 8. The index size and construction time.

5.2.5 ***Parameter Sensitivity Study.*** Here, we examine how the performance of DEG and the baselines change is affected by the two key hyperparameters, M and $ef_{construction}$. Specifically, we



(a) $M = 40, ef_{construtionn} = 200$ ($M = 10, ef_{construction} = 50$ for HNSW$_O$)

(b) $M = 60, ef_{construtionn} = 300$ ($M = 14, ef_{construction} = 70$ for HNSW$_O$)

Fig. 9. The accuracy-efficiency trade-off results on OpenImage dataset with varying M and $ef_{construtionn}$.

increase M and $ef_{construction}$ from 40, 200 to 50, 250, then to 60, 300 for HNSW$_F$, HNSW$_M$, and DEG. For HNSW$_O$, we adjust the M and $ef_{construction}$ of each sub-index from 10, 50 to 12, 60, then to 14, 70, ensuring that they have comparative edges and allow for a fair comparison. Due to the page limitation, we only report the results for $\alpha \in [0, 0.2], [0.4, 0.6], [0.8, 1.0]$ when $M = 40, 60$, with the other results in the appendix. The results demonstrate that: (1) For the different number of edges M and candidate set size $ef_{construction}$, DEG maintains the best performance among the baselines. (2) With varying M and $ef_{construction}$, both baselines HNSW$_F$, HNSW$_M$, and DEG exhibit similar performance. This indicates that for graph-based ANNS indexes, once M and $ef_{construction}$ are fine-tuned to relatively large values, the key factors determining performance are no longer the hyperparameters, but rather the edge selection strategy, as to be shown later. (3) The performance of the HNSW$_O$ improves as M and $ef_{construction}$ increase, but but this leads to significantly higher index construction costs. For example, when $M = 14$ and $ef_{construction} = 70$, the overall performance of HNSW$_O$ becomes comparable to that of HNSW$_F$, but the build time for HNSW$_O$ is twice as long as that of HNSW$_F$. This indicates that although HNSW$_F$ offers better performance as M and $ef_{construction}$ increase, it comes at the cost of significantly higher indexing overhead.

5.2.6 ***Ablation study.*** Here, we investigate how the proposed components contribute to DEG's performance. Due to page limitations, we present results when $\alpha \in [0, 0.2], [0.4, 0.6], [0.8, 1]$, with full results in the appendix.

The DEG's pruning strategy. To verify the effectiveness of the DEG's pruning strategy, we compared DEG with DEG$_{None}$, which does not apply any pruning strategy but uses the approximate Pareto frontiers obtained by the GPS algorithm as edges directly and assigns each edge an active range $u = [0, 1]$. The experimental results are shown in Figure 10, which show that DEG consistently outperforms DEG$_{None}$ by a large margin across varying α. This validates the effectiveness of the DEG's pruning strategy and confirms that the edge selection strategy is the key factor in enhancing the search performance of graph-based ANNS indexes.

The active range. We further explore how the active range enhances DEG's search performance by proposing an alternative method, DEG$_{static}$. DEG$_{static}$ uses the same index as DEG but routes through all edges during the search phase, ignoring the active range. The experimental results on the OpenImage dataset are shown in Figure 11. The results show that the DEG consistently

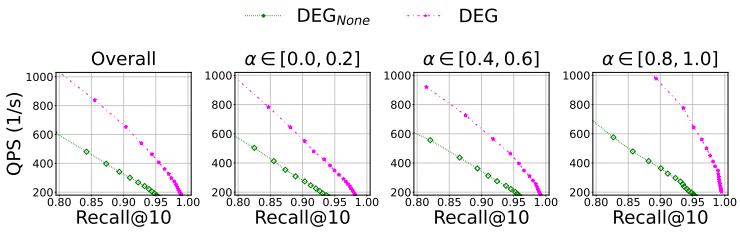


Fig. 10. Ablation Study for the DEG's pruning strategy.

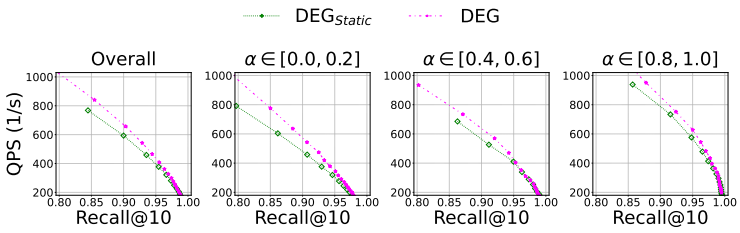


Fig. 11. Ablation Study for the Active Range.

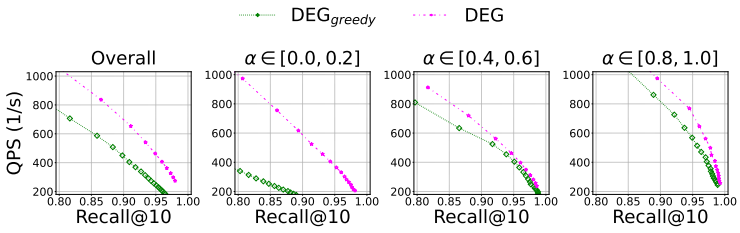


Fig. 12. Ablation Study for candidate acquisition method.

outperforms the DEG$_{static}$ for varying α, which indicates the active range can enhance search performance.

The candidate acquisition method. To verify the effectiveness of the GPS algorithm, we replaced the GPS algorithm in DEG with the greedy search algorithm, fixing $\alpha = 0.5$ during the index construction phase, which we call DEG$_{\text{greedy}}$. The experiment results on the OpenImage dataset are shown in Figure 12. The results validate that DEG consistently outperforms the DEG $_{\text{greedy}}$ for varying $q.\alpha$, proving that the GPS algorithm can acquire better candidates than the greedy search algorithm.

The edge seed method. To verify the effectiveness of the edge seed acquisition method, we consider two alternative methods, called DEG$_{\text{centroid}}$, which selects the group of points closest to the centroid as the seed. Specifically, it maintains the Pareto frontier of the graph center during the index construction phase and uses it as the starting point during the search phase. Another method is called DEG$_{\text{random}}$, which randomly selects some nodes, approximately the same size as the edge seed, as the starting point. The experimental results are shown in Figure 13. The results show that DEG consistently outperforms the two alternatives for varying α, which validates the edge seed method's superiority.

6 CONCLUSIONS AND FUTURE WORK

In this paper, we propose a novel ANNS index, DEG, for the hybrid vector query problem, which comprises three novel components: the greedy Pareto frontier search algorithm, the dynamic edge pruning strategy, and the edge seed method. One possible future direction is to extend our proposed index to multi-vector queries, where each object involves more than two vectors. Another potential

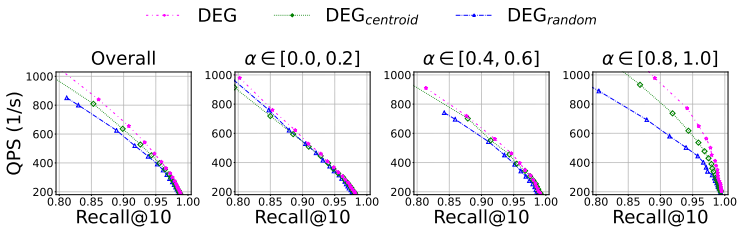


Fig. 13. Ablation Study for Edge Seed Acquisition method.

direction is to develop a disk-based version of our index, making it suitable for implementation in vector databases.

Acknowledgments

We thank the anonymous reviewers for their valuable feedback. This research is supported in part by Singapore MOE AcRF Tier-2 grants MOE-T2EP20221-0015 and MOE-T2EP20223-0004, and MOE AcRF Tier-1 grant RT6/23.

References

[1] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. 2018. HD-Index: Pushing the Scalability-Accuracy Boundary for Approximate kNN Search in High-Dimensional Spaces. *Proceedings of the VLDB Endowment* 11, 8 (2018), 906–919.

[2] Tadas Baltrušaitis, Chaitanya Ahuja, and Louis-Philippe Morency. 2018. Multimodal machine learning: A survey and taxonomy. *IEEE transactions on pattern analysis and machine intelligence* 41, 2 (2018), 423–443.

[3] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. 1990. The R*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*. 322–331.

[4] Alina Beygelzimer, Sham M. Kakade, and John Langford. 2006. Cover Trees for Nearest Neighbor. In *Machine Learning, Proceedings of the Twenty-Third International Conference (ICML 2006), Pittsburgh, Pennsylvania, USA, June 25-29, 2006 (ACM International Conference Proceeding Series, Vol. 148)*. 97–104.

[5] S. Borzsony, D. Kossmann, and K. Stocker. 2001. The Skyline Operator. In *Proceedings 17th International Conference on Data Engineering*. Heidelberg, Germany, 421–430.

[6] Petra Budıková. 2013. Towards Large-Scale Multi-Modal Image Search. (2013).

[7] Qi Chen, Haidong Wang, Mingqin Li, Gang Ren, Scarlett Li, Jeffery Zhu, Jason Li, Chuanjie Liu, Lintao Zhang, and Jingdong Wang. 2018. SPTAG: A library for fast approximate nearest neighbor search.

[8] Xinyu Chen, Jiajie Xu, Rui Zhou, Pengpeng Zhao, Chengfei Liu, Junhua Fang, and Lei Zhao. 2020. S2R-Tree: A Pivot-Based Indexing Structure for Semantic-Aware Spatial Keyword Search. *GeoInformatica* 24, 1 (2020), 3–25.

[9] Zhida Chen, Lisi Chen, Gao Cong, and Christian S. Jensen. 2021. Location- and Keyword-Based Querying of Geo-Textual Data: A Survey. *The VLDB Journal* 30, 4 (2021), 603–640.

[10] Thomas M. Cover and Peter E. Hart. 1967. Nearest neighbor pattern classification. *IEEE Trans. Inf. Theory* 13, 1 (1967), 21–27.

[11] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*, Jack Snoeyink and Jean-Daniel Boissonnat (Eds.). 253–262.

[12] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *Proceedings of the 20th ACM Symposium on Computational Geometry, Brooklyn, New York, USA, June 8-11, 2004*. 253–262.

[13] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* (2018).

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. 2020. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929* (2020).

[15] Ronald Fagin, Amnon Lotem, and Moni Naor. 2001. Optimal aggregation algorithms for middleware. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 102–113.

[16] Steven Fortune. 2004. Voronoi Diagrams and Delaunay Triangulations. In *Handbook of Discrete and Computational Geometry, Second Edition*. 513–528.

[17] Maximilian Franzke, Tobias Emrich, Andreas Zufle, and Matthias Renz. 2016. Indexing Multi-Metric Data. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. 1122–1133.

[18] Cong Fu, Changxu Wang, and Deng Cai. 2022. High Dimensional Similarity Search with Satellite System Graph: Efficiency, Scalability, and Unindexed Query Compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2022), 4139–4150.

[19] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2019. Fast Approximate Nearest Neighbor Search with the Navigating Spreading-out Graph. *Proceedings of the VLDB Endowment* 12, 5 (2019), 461–474.

[20] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-Sensitive Hashing Scheme Based on Dynamic Collision Counting. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2012, Scottsdale, AZ, USA, May 20-24, 2012*. 541–552.

[21] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.

[22] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Trans. Pattern Anal. Mach. Intell.* 36, 4 (2014), 744–755.

[23] Long Gong, Huayi Wang, Mitsunori Ogihara, and Jun Xu. 2020. iDEC: Indexable Distance Estimating Codes for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 13, 9 (2020), 1483–1497.

[24] Yunchao Gong, Svetlana Lazebnik, Albert Gordo, and Florent Perronnin. 2013. Iterative Quantization: A Procrustean Approach to Learning Binary Codes for Large-Scale Image Retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35, 12 (2013), 2916–2929.

[25] Rentong Guo, Xiaofan Luan, Long Xiang, Xiao Yan, Xiaomeng Yi, Jigao Luo, Qianya Cheng, Weizhi Xu, Jiarui Luo, Frank Liu, Zhenshan Cao, Yanliang Qiao, Ting Wang, Bo Tang, and Charles Xie. 2022. Manu: A Cloud Native Vector Database Management System. *Proceedings of the VLDB Endowment* 15, 12 (2022), 3548–3561.

[26] Ben Harwood and Tom Drummond. 2016. FANNG: Fast Approximate Nearest Neighbour Graphs. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. 5713–5722.

[27] Ben Harwood and Tom Drummond. 2016. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5713–5722.

[28] Jae-Pil Heo, Youngwoon Lee, Junfeng He, Shih-Fu Chang, and Sung-Eui Yoon. 2015. Spherical Hashing: Binary Code Embedding with Hyperspheres. *IEEE Trans. Pattern Anal. Mach. Intell.* 37, 11 (2015), 2304–2316.

[29] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-Aware Locality-Sensitive Hashing for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 9, 1 (2015), 1–12.

[30] Piotr Indyk and Rajeev Motwani. 1998. Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the Thirtieth Annual ACM Symposium on the Theory of Computing, Dallas, Texas, USA, May 23-26, 1998*. 604–613.

[31] Masajiro Iwasaki. 2015. Neighborhood Graph and Tree for Indexing Highdimensional Data. *Yahoo Japan Corporation. Retrieved August* 22 (2015), 2020.

[32] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, and Rui Zhang. 2005. iDistance: An adaptive B^+-tree based indexing method for nearest neighbor search. *ACM Trans. Database Syst.* 30, 2 (2005), 364–397.

[33] Jerzy W Jaromczyk and Mirosław Kowaluk. 1991. Constructing the relative neighborhood graph in 3-dimensional Euclidean space. *Discrete Applied Mathematics* 31, 2 (1991), 181–191.

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

[35] Christos Kalyvas and Theodoros Tzouramanis. 2017. A survey of skyline query processing. *arXiv preprint arXiv:1704.01788* (2017).

[36] Mohamed E. Khalefa, Mohamed F. Mokbel, and Justin J. Levandoski. 2008. Skyline Query Processing for Incomplete Data. In *2008 IEEE 24th International Conference on Data Engineering*. Cancun, Mexico, 556–565.

[37] Joseph B Kruskal. 1956. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical society* 7, 1 (1956), 48–50.

[38] Yifan Lei, Qiang Huang, Mohan S. Kankanhalli, and Anthony K. H. Tung. 2020. Locality-Sensitive Hashing Scheme Based on Longest Circular Co-Substring. In *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, Online Conference [Portland, or, USA], June 14-19, 2020*. 2589–2599.

[39] Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. 2020. Improving Approximate Nearest Neighbor Search through Learned Adaptive Early Termination. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2539–2554.

[40] Jinfeng Li, Xiao Yan, Jie Zhang, An Xu, James Cheng, Jie Liu, Kelvin Kai Wing Ng, and Ti-Chung Cheng. 2018. A General and Efficient Querying Method for Learning to Hash. In *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*. 1333–1347.

[41] Wen Li, Ying Zhang, Yifang Sun, Wei Wang, Mingjie Li, Wenjie Zhang, and Xuemin Lin. 2020. Approximate Nearest Neighbor Search on High Dimensional Data — Experiments, Analyses, and Improvement. *IEEE Transactions on Knowledge and Data Engineering* 32, 8 (2020), 1475–1488.

[42] Shang Liu, Gao Cong, Kaiyu Feng, Wanli Gu, and Fuzheng Zhang. 2023. Effectiveness Perspectives and a Deep Relevance Model for Spatial Keyword Queries. *Proc. ACM Manag. Data* 1, 1 (2023), 11:1–11:25.

[43] Yingfan Liu, Jiangtao Cui, Zi Huang, Hui Li, and Heng Tao Shen. 2014. SK-LSH: An Efficient Index Structure for Approximate Nearest Neighbor Search. *Proc. VLDB Endow.* 7, 9 (2014), 745–756.

[44] Kejing Lu, Hongya Wang, Wei Wang, and Mineichi Kudo. 2020. VHP: Approximate Nearest Neighbor Search via Virtual Hypersphere Partitioning. *Proc. VLDB Endow.* 13, 9 (2020), 1443–1455.

[45] Pingchuan Ma, Tao Du, and Wojciech Matusik. 2020. Efficient continuous pareto exploration in multi-task learning. In *International Conference on Machine Learning*. PMLR, 6522–6531.

[46] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate Nearest Neighbor Algorithm Based on Navigable Small World Graphs. 45 (2014), 61–68.

[47] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020), 824–836.

[48] Yusuke Matsui, Yusuke Uchida, Hervé Jégou, and Shin'ichi Satoh. 2018. A Survey of Product Quantization. *ITE Transactions on Media Technology and Applications* 6, 1 (2018), 2–10.

[49] Antoine Miech, Jean-Baptiste Alayrac, Lucas Smaira, Ivan Laptev, Josef Sivic, and Andrew Zisserman. 2020. End-to-end learning of visual representations from uncurated instructional videos. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 9879–9889.

[50] Antoine Miech, Dimitri Zhukov, Jean-Baptiste Alayrac, Makarand Tapaswi, Ivan Laptev, and Josef Sivic. 2019. HowTo100M: Learning a Text-Video Embedding by Watching Hundred Million Narrated Video Clips. In *ICCV*.

[51] James Jie Pan, Jianguo Wang, and Guoliang Li. 2024. Vector Database Management Techniques and Systems. In *Companion of the 2024 International Conference on Management of Data*. Santiago AA Chile, 597–604.

[52] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive Skyline Computation in Database Systems. *ACM Transactions on Database Systems* 30, 1 (2005), 41–82.

[53] Rodrigo Paredes and Edgar Chávez. 2005. Using the *k*-Nearest Neighbor Graph for Proximity Searching in Metric Spaces. In *String Processing and Information Retrieval, 12th International Conference, SPIRE 2005, Buenos Aires, Argentina, November 2-4, 2005, Proceedings (Lecture Notes in Computer Science, Vol. 3772)*. 127–138.

[54] Jordi Pont-Tuset, Jasper Uijlings, Soravit Changpinyo, Radu Soricut, and Vittorio Ferrari. 2020. Connecting Vision and Language with Localized Narratives. In *ECCV*.

[55] Zhihu Qian, Jiajie Xu, Kai Zheng, Pengpeng Zhao, and Xiaofang Zhou. 2018. Semantic-Aware Top-k Spatial Keyword Queries. *World Wide Web* 21, 3 (2018), 573–594.

[56] Parikshit Ram and Kaushik Sinha. 2019. Revisiting Kd-Tree for Nearest Neighbor Search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2019, Anchorage, AK, USA, August 4-8, 2019*. 1378–1388.

[57] Amaia Salvador, Nicholas Hynes, Yusuf Aytar, Javier Marin, Ferda Ofli, Ingmar Weber, and Antonio Torralba. 2017. Learning cross-modal embeddings for cooking recipes and food images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 3020–3028.

[58] J. Ben Schafer, Dan Frankowski, Jonathan L. Herlocker, and Shilad Sen. 2007. Collaborative Filtering Recommender Systems. In *The Adaptive Web, Methods and Strategies of Web Personalization (Lecture Notes in Computer Science, Vol. 4321)*. 291–324.

[59] Piyush Sharma, Nan Ding, Sebastian Goodman, and Radu Soricut. 2018. Conceptual Captions: A Cleaned, Hypernymed, Image Alt-text Dataset For Automatic Image Captioning. In *Proceedings of ACL*.

[60] Yufei Tao, Ke Yi, Cheng Sheng, and Panos Kalnis. 2010. Efficient and Accurate Nearest Neighbor and Closest Pair Search in High-Dimensional Space. *ACM Transactions on Database Systems* 35, 3 (2010), 20:1–20:46.

[61] George S. Theodoropoulos, Kjetil Nørvåg, and Christos Doulkeridis. 2024. Efficient Semantic Similarity Search over Spatio-Textual Data. In *Proceedings 27th International Conference on Extending Database Technology, EDBT 2024, Paestum, Italy, March 25 - March 28*. 268–280.

[62] Yao Tian, Xi Zhao, and Xiaofang Zhou. 2024. DB-LSH 2.0: Locality-Sensitive Hashing with Query-Based Dynamic Bucketing. *IEEE Transactions on Knowledge and Data Engineering* 36, 3 (2024), 1000–1015.

[63] Godfried T. Toussaint. 1980. The relative neighbourhood graph of a finite planar set. *Pattern Recognit.* 12, 4 (1980), 261–268.

[64] Ertem Tuncel, Hakan Ferhatosmanoglu, and Kenneth Rose. 2002. VQ-Index: An Index Structure for Similarity Searching in Multimedia Databases. In *Proceedings of the 10th ACM International Conference on Multimedia 2002, Juan Les Pins, France, December 1-6, 2002*. 543–552.

[65] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. 2014. Hashing for Similarity Search: A Survey. *CoRR* abs/1408.2927 (2014).

[66] 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 *SIGMOD '21: International Conference on Management of Data, Virtual Event, China, June 20-25, 2021*. 2614–2627.

[67] Jingdong Wang, Ting Zhang, Jingkuan Song, Nicu Sebe, and Heng Tao Shen. 2018. A Survey on Learning to Hash. *IEEE Trans. Pattern Anal. Mach. Intell.* 40, 4 (2018), 769–790.

[68] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A Comprehensive Survey and Experimental Comparison of Graph-Based Approximate Nearest Neighbor Search. *Proceedings of the VLDB Endowment* 14, 11 (2021), 1964–1978.

[69] Roger Weber, Hans-Jörg Schek, and Stephen Blott. 1998. A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, Ashish Gupta, Oded Shmueli, and Jennifer Widom (Eds.). 194–205.

[70] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. AnalyticDB-V: A Hybrid Analytical Engine towards Query Fusion for Structured and Unstructured Data. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3152–3165.

[71] 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.

[72] Wayne Xin Zhao, Jing Liu, Ruiyang Ren, and Ji-Rong Wen. 2022. Dense Text Retrieval based on Pretrained Language Models: A Survey. *CoRR* abs/2211.14876 (2022).
[73] Bolong Zheng, Xi Zhao, Lianggui Weng, Quoc Viet Hung Nguyen, Hang Liu, and Christian S. Jensen. 2022. PM-LSH: A Fast and Accurate in-Memory Framework for High-Dimensional Approximate NN and Closest Pair Search. *Vldb Journal* 31, 6 (2022), 1339–1363.
[74] Zhiyong Huang, Hua Lu, Beng Chin Ooi, and A.K.H. Tung. 2006. Continuous Skyline Queries for Moving Objects. *IEEE Transactions on Knowledge and Data Engineering* 18, 12 (2006), 1645–1658.
[75] Chun Jiang Zhu, Tan Zhu, Haining Li, Jinbo Bi, and Minghu Song. 2019. Accelerating Large-Scale Molecular Similarity Search through Exploiting High Performance Computing. In *2019 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2019, San Diego, CA, USA, November 18-21, 2019*. 330–333.

Received July 2024; revised September 2024; accepted November 2024