Supplementary Material for Paper "Graph-Based Vector Search: An Experimental Evaluation of the State-of-the-Art"

1 Neighborhood Diversification techniques

During the early development of graph-based approaches for similarity search, methods like Kgraph [34] suggested approximating the K-nearest neighbor (KNN) graph using NNDescent [35], which employs the propagation and sharing of neighborhood lists between connected nodes. Conversely, NSW [108] proposed a different approach to constructing the graph by incrementally inserting nodes into the graph. In NSW, each node's neighborhood list is retrieved through beam search on the graph, and bidirectional edges are constructed to ensure connectivity between the newly inserted node and the previously inserted ones. This method places a stronger emphasis on long-range connections. Despite minor or major differences among these methods, they all encounter a common issue: a larger number of distance calculations during the search process. This is because these methods require a large outdegree to achieve good accuracy during graph search.

Since then, various approaches have been employed to refine and enhance the graph data structure for similarity search. The primary objective has been to render these graph structures as sparse as possible by pruning unnecessary edges of the structure while maintaining good connectivity and proximity within the graph. This is achieved by diversifying the connections of each node to avoid including edges that lead to the same direction, thus preventing redundant calculations. Instead, the aim is to have more diversified connections that assist the graph in avoiding local minima while reducing the necessary outdegree to achieve high accuracy. These approaches, collectively known as Neighborhood Diversification (ND) methods, have been identified and classified into three fundamental strategies in the literature: Relative Neighborhood Diversification (RND), Relaxed Relative Neighborhood Diversification (RRND), and Maximum-Oriented Neighborhood Diversification (MOND).

We Consider the following:

- X_q is the query node, i.e. the node to be inserted in the graph.

- R_q, the list of current closest neighbors to X_q.
- C_q, the list of candidate neighbors to X_q not yet in R_q.
- X_j is a candidate neighbor being considered for inclusion in R_q and is part of C_q.
- X_i are nodes already in the set R_q.
- $\text{dist}(X_i, X_j)$ represents the Euclidean distance between nodes X_i and X_j in the d-dimensional space.

Definition 1 (RND) *The node X_j is added to R_q if and only if the following condition holds:*

$$\forall X_i \in R_q, dist(X_q, X_j) < dist(X_i, X_j) \quad (Eq.\ 1)$$

Definition 2 (RRND) *The node X_j is added to R_q if and only if the following condition holds:*

$$\forall X_i \in R_q, dist(X_q, X_j) < \alpha \cdot dist(X_i, X_j) \quad (Eq.\ 2)$$

Where α is the relaxation factor (e.g., $\alpha = 1.5$).

Definition 3 (MOND) *The node X_j is added to R_q if and only if the following condition holds:*

$$\forall X_i \in R_q, \cos(\angle X_j X_q X_i) < \cos(\theta) \quad (Eq.\ 3)$$

Where θ is the angle threshold (e.g., $60°$).

HNSW [91] was the first to employ such approach to improve the performance of NSW by using RND, which was inspired from the relative neighborhood graph [125], this same ND approach has been adapted in various following methods such as NSG [53], SPTAG [29], ELPIS [11]. Meanwhile, DPG [83] and NSSG [52] use MOND and VAMANA [121] employs RRND.

While it's intuitive how RRND is an approximation of RND, meaning that if RRND prunes an edge, we can be sure that this edge will also be pruned under RND. In contrast, if an edge is pruned by RND, it may not necessarily be pruned by RRND, as RRND relaxes the pruning condition to control the density of the graph with the parameter $\alpha \geq 1$.

However, MOND uses a different approach from RND. It employs the angles between edges to prune the neighbors of a current node and diversify the neighborhood list by maximizing the angles between the neighbors. In the next section, we will prove that MOND is an approximation of RND.

1.1 Proof that MOND and RND are Approximations of RND

To demonstrate that MOND (Maximum Oriented Neighborhood Diversification) is an approximation of RND (Relative Neighborhood Diversification), we consider a scenario involving three vectors: $\mathbf{X}_q$ (representing the query vector), $\mathbf{X}_i$ (a neighbor already added to the list of neighbors $\mathbf{R}$), and $\mathbf{X}_j$ (a candidate for inclusion in $\mathbf{R}$). Our objective is to show that if RND selects $\mathbf{X}_j$ into $\mathbf{R}$, then MOND will also include $\mathbf{X}_j$ in its list of neighbors, though not necessarily the other way around.

We initiate the proof by examining the conditions for $\mathbf{X}_j$ to be added to $\mathbf{R}$ by RND, with the following terminologies:

- $\mathbf{X}_q$ is the query vector.
- $\mathbf{X}_i$ is a neighbor already in $\mathbf{R}$.
- $\mathbf{X}_j$ is a candidate neighbor being considered for inclusion in $\mathbf{R}$.

We denote the Euclidean distance between vectors $\mathbf{X}_a$ and $\mathbf{X}_b$ as $\|\mathbf{X}_{ab}\|$, simplifying the notation. Thus, we have the RND condition as follows:

$$\begin{aligned}
\|\mathbf{X}_{qj}\| > \|\mathbf{X}_{qi}\| &\implies \|\mathbf{X}_{qj}\|^2 > \|\mathbf{X}_{qi}\|^2 \\
&\implies -\|\mathbf{X}_{qj}\|^2 < -\|\mathbf{X}_{qi}\|^2 \\
&\implies \|\mathbf{X}_{qi}\|^2 - \|\mathbf{X}_{qj}\|^2 < 0 \\
&\implies \|\mathbf{X}_{qj}\|^2 + \|\mathbf{X}_{iq}\|^2 - \|\mathbf{X}_{jq}\|^2 < \|\mathbf{X}_{iq}\|^2 \qquad \text{(I)}
\end{aligned}$$

$$\begin{aligned}
\|\mathbf{X}_{qj}\| > \|\mathbf{X}_{qi}\| &\implies 2\|\mathbf{X}_{qj}\| \cdot \|\mathbf{X}_{qi}\| > 2\|\mathbf{X}_{qi}\|^2 \\
&\implies \frac{1}{2\|\mathbf{X}_{qj}\| \cdot \|\mathbf{X}_{qi}\|} < \frac{1}{2\|\mathbf{X}_{qi}\|^2} \qquad \text{(II)}
\end{aligned}$$

By combining the results from (I) and (II), we deduce:

$$\frac{\|\mathbf{X}_{qj}\|^2 + \|\mathbf{X}_{qi}\|^2 - \|\mathbf{X}_{qj}\|^2}{2\|\mathbf{X}_{qj}\| \cdot \|\mathbf{X}_{qi}\|} < \frac{1}{2}$$

This implies:

$$\cos(\angle \mathbf{X}_i \mathbf{X}_q \mathbf{X}_j) < \frac{1}{2}$$

Hence, if RND adds $\mathbf{X}_j$ to its list of $\mathbf{X}_q$'s neighbors, MOND will also do the same. Thus, MOND is an approximation of RND. To prove that the inverse is not true, RND is not always satisfied when MOND is, we can take a small example in 2D with:
- $X_q = (0, 0)$ - $X_i = (2, 0)$ - $X_j = (1.8, 2.5)$

Now, let's calculate the distances $\text{dist}(X_q, X_i)$ and $\text{dist}(X_q, X_j)$:

$$\text{dist}(X_q, X_i) = \sqrt{(2-0)^2 + (0-0)^2} = \sqrt{4+0} = \sqrt{4} = 2$$

$$\text{dist}(X_q, X_j) = \sqrt{(1.8-0)^2 + (2.5-0)^2} = \sqrt{3.24+6.25} = \sqrt{9.49} \approx 3.08$$

thus, X_j will be pruned following RND. Next, we will see if the same holds using MOND:

$$\overrightarrow{X_q X_i} = (2-0, 0-0) = (2, 0)$$

$$\overrightarrow{X_q X_j} = (1.8-0, 2.5-0) = (1.8, 2.5)$$

$$\overrightarrow{X_q X_i} \cdot \overrightarrow{X_q X_j} = (2 \cdot 1.8) + (0 \cdot 2.5) = 3.6$$

$$|\overrightarrow{X_q X_i}| = \sqrt{2^2 + 0^2} = \sqrt{4} = 2$$

$$|\overrightarrow{X_q X_j}| = \sqrt{1.8^2 + 2.5^2} = \sqrt{3.24+6.25} = \sqrt{9.49} \approx 3.08$$

$$\cos(\theta) = \frac{\overrightarrow{X_q X_i} \cdot \overrightarrow{X_q X_j}}{|\overrightarrow{X_q X_i}| \cdot |\overrightarrow{X_q X_j}|} = \frac{3.6}{2 \times 3.08} \approx 0.586$$

thus, MOND will not prune X_j from X_q neighborhood list.

1.2 Pruning Ratio of Neighborhood Diversification

In the previous section, we demonstrated how RND prunes more aggressively than RRND, which relaxes the pruning condition using the factor alpha, and MOND through proofs. Our theoretical results align with the pruning ratio of the three methods empirically, as shown in the Figure 1 below. We display the average pruning ratio across the nodes during graph building for the Deep and Sift datasets:

The pruning of each method is calculated during the construction of the incremental insertion-based graph. For each node, we use the KNN results retrieved by running a beam search on the partial graph of the already inserted nodes as the candidate neighbors list. The pruning ratio is calculated using the ratio of the candidate neighbors list size before and after the neighborhood diversification applied. A low value signifies a low pruning, and vice versa.

From our results in both the Deep and Sift datasets, we observed that RND ranked first with a large pruning ratio of 20% and 25%, respectively. MOND ranked second with 2% and 4%, while RRND, using a default value of alpha=1.5, ranked last with 0.6% and 0.7% of average pruning ratio in the respective datasets.

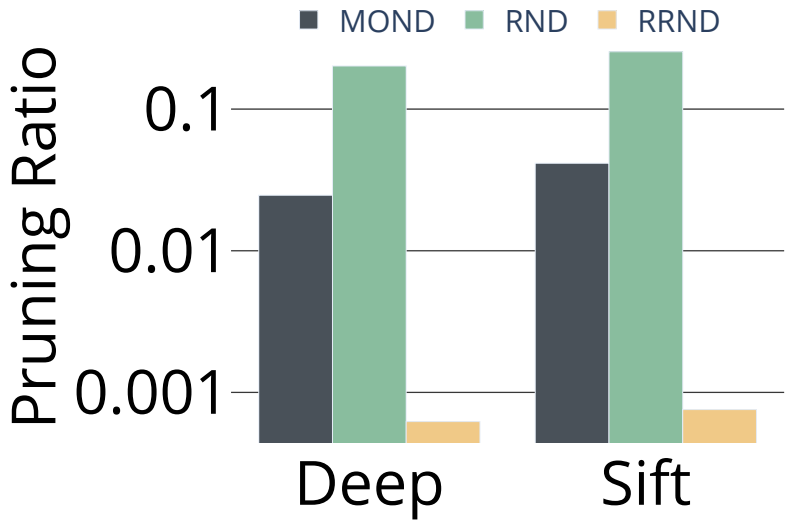
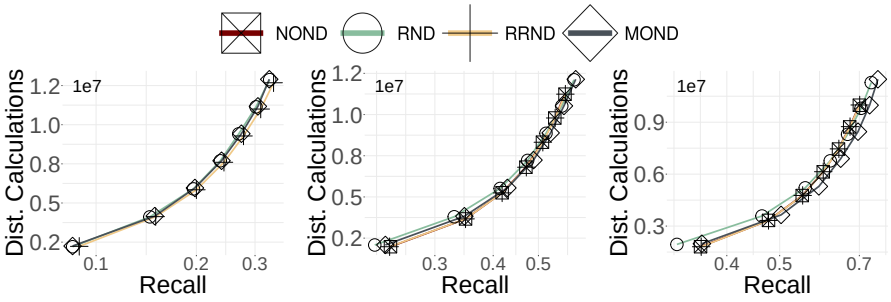


Figure 1: Average pruning ratio across nodes during graph building for 25GB datasets.

1.3 ND Performance accross random datasets

We compare different ND approaches with II based graph across random dataset with different skewness, Pow1, Pow5 and Pow50 25GB datasets.



(a) Pow1 (b) Pow5 (c) Pow50

Figure 2: ND methods performance on random datasets

Figure 2 shows that the three ND strategies have comparable performance when the data distribution is uniform (Pow1), but when the distribution is skewed, MOND takes the lead and RRND becomes competitive with RND. We believe that MOND exhibits a superior performance because on the highly-

skewed datasets, the distribution is concentrated within a specific region, thus the orientation pruning of MOND becomes more effective.

2 Beam search

Beam search, originally proposed by Raj Reddy in 1977 [112], is a fundamental algorithm employed in various fields such as speech recognition, machine translation, and natural language processing. It involves exploring a limited number of hypotheses, termed the "beam," to enhance efficiency compared to exhaustive search methods. The algorithm sacrifices completeness and optimality for efficiency, as it may prune potential goal states and does not guarantee the best solution. The time complexity of beam search varies based on parameters like the beam width and the maximum allowed path length within the graph. In proximity graphs, such as those used in spatial data analysis, beam search can be adapted to efficiently explore nearby nodes or edges, contributing to its versatility in different applications.

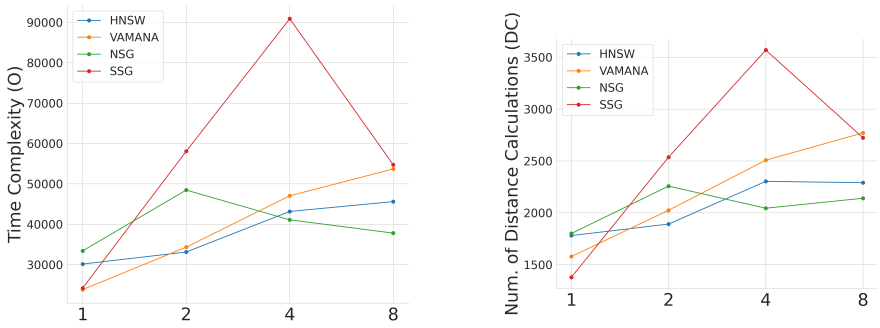
The time complexity of beam search can be expressed using the following formula:

$$\text{Time Complexity} = O(B \times m \times T) \quad \text{(Eq. 4)}$$

Where B represents the beam width, 'm' denotes the average branching factor, and in the context of graph search, the average outdegree, and 'T' signifies the maximum allowed path length within the graph. In the worst-case scenario, 'T' equals the diameter of the graph, representing the longest permissible path. This formula quantifies the computational overhead of beam search, taking into account the beam width parameter, the average out-degree, and the graph diameter. However, since there exist no guarantees over the search accuracy for a given beam width, the time complexity cannot be estimated without running a search on a set of queries to priorly know the beamwidth required for the target accuracy (sometimes the gap between the beamwidth of two methods may be larger than their differences in outdegree and diameter). It's also important to mention that using different seed selection approaches may affect time complexity, even more when the required number of NN is small.

Below, we show some analysis on the theoretical time complexity and how this complexity aligns with the empirical performance of four state-of-the-art graph-based algorithms. For each method, we build the graph using max outdegrees of 40, 40, 50, and 60 respectively for datasets Deep1M, Deep2M, Deep4M, and Deep8M. The goal is to analyze the time complexity of different methods and if it aligns with the empirical efficiency of the methods, both in time and the number of distance calculations. All the numbers are retrieved for a targeted accuracy of 0.99.

In Figure 3, we illustrate how the theoretical time complexity of the beam search aligns with the algorithm efficiency, specifically the number of distance calculations across different dataset sizes. As the dataset size increases, we can observe an improvement in the approximation of the empirical complexity



(a) Theoretical Time Complexity

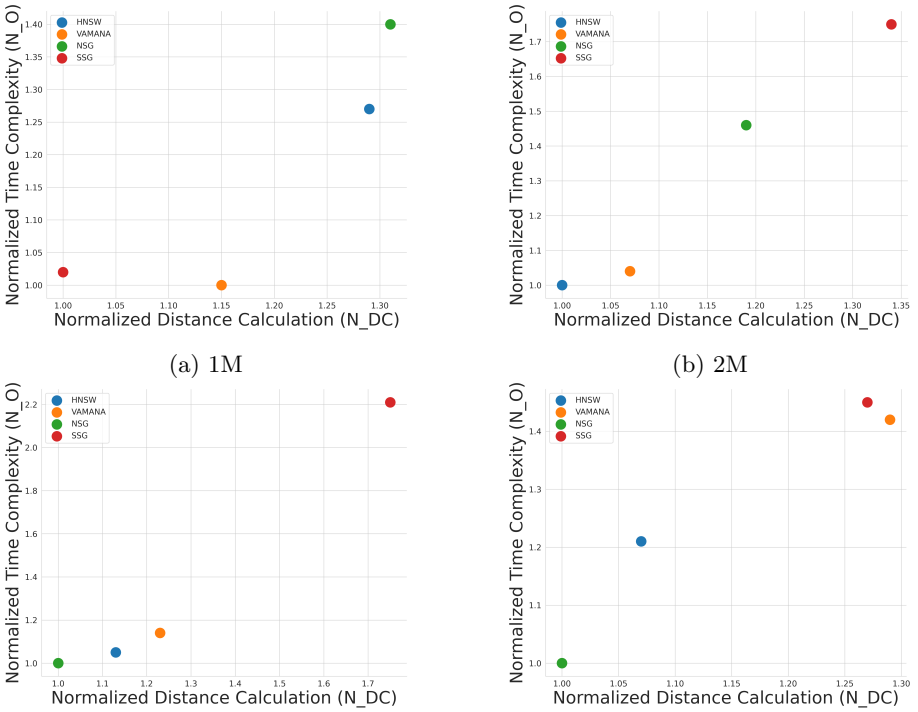
(b) Empirical efficiency

Figure 3: Theoretical and empirical beams search efficiency across different deep1b dataset sizes

(comparing 1M to 2M, 4M, and 8M), as the ranking between methods in terms of beam search efficiency is approximated using the beam search time complexity formula in Eq. 4. In Figure 4, we present the comparison between the time complexity and search efficiency of four methods for different deep1b dataset scales. We also add tables 1, 2, 3, 4 for more detailed numbers.

Table 1: Deep1M

	HNSW	**VAMANA**	**NSG**	**SSG**
m	27.4	39.7	34.8	33.6
T	11	10	12	12
B	100	60	80	60
O	30140	23820	33408	24192
N_O	1.27×10^0	1.00×10^0	1.40×10^0	1.02×10^0
DC	1780	1578	1800	1378
N_DC	1.29×10^0	1.15×10^0	1.31×10^0	1.00×10^0
TIME	0.3	0.29	0.22	0.2
N_TIME	1.50×10^0	1.45×10^0	1.10×10^0	1.00×10^0
O/DC	17	15	19	18
O/TIME	100467	82138	151855	120960



(a) 1M

(b) 2M

(c) 4M

(d) 8M

Figure 4: Normalized Distance Calculation (N_DC) vs Normalized Time Complexity (N_O) across different Deep1B dataset sizes

Table 2: Deep2M

	HNSW	**VAMANA**	**NSG**	**SSG**
m	27.6	39	34.61	31.9
T	12	11	14	14
B	100	80	100	130
O	33120	34320	48454	58058
N_O	1.00×10^0	1.04×10^0	1.46×10^0	1.75×10^0
DC	1891	2024	2257	2536
N_DC	1.00×10^0	1.07×10^0	1.19×10^0	1.34×10^0
TIME	0.32	0.4	0.29	0.37
N_TIME	1.10×10^0	1.38×10^0	1.00×10^0	1.28×10^0
O/DC	18	17	21	23
O/TIME	103500	85800	167083	156914

Table 3: Deep4M

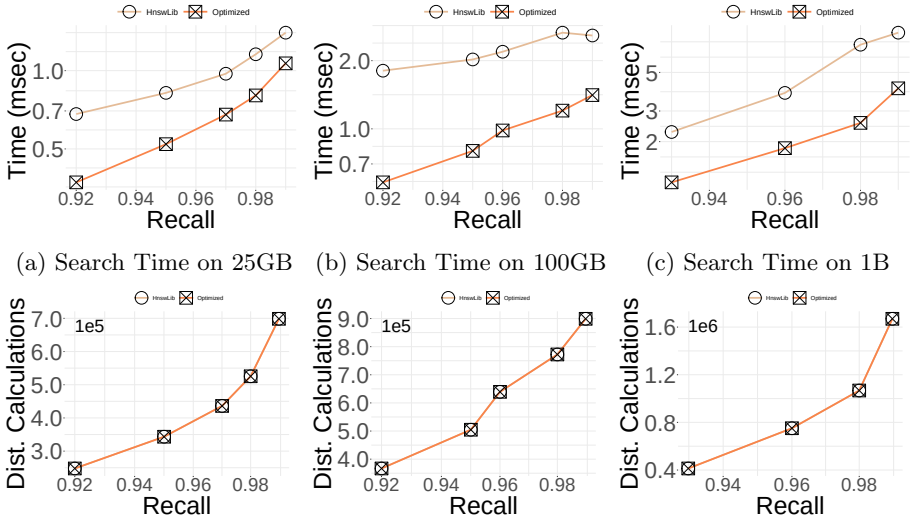
	HNSW	**VAMANA**	**NSG**	**SSG**
m	33.2	49	39.13	35.5
T	13	12	15	16
B	100	80	70	160
O	43160	47040	41086.5	90880
N_O	1.05×10^0	1.14×10^0	1.00×10^0	2.21×10^0
DC	2303	2507	2044	3569
N_DC	1.13×10^0	1.23×10^0	1.00×10^0	1.75×10^0
TIME	0.4	0.5	0.27	0.54
N_TIME	1.48×10^0	1.85×10^0	1.00×10^0	2.00×10^0
O/DC	19	19	20	25
O/TIME	107900	94080	152172	168296

Table 4: Deep8M

	HNSW	**VAMANA**	**NSG**	**SSG**
m	38	59	42	38
T	15	13	15	16
B	80	70	60	90
O	45600	53690	37800	54720
N_O	1.21×10^0	1.42×10^0	1.00×10^0	1.45×10^0
DC	2290	2770	2140	2723
N_DC	1.07×10^0	1.29×10^0	1.00×10^0	1.27×10^0
TIME	0.33	0.48	0.31	0.42
N_TIME	1.06×10^0	1.55×10^0	1.00×10^0	1.35×10^0
O/DC	20	19	18	20
O/TIME	138182	111854	121935	130286

3 Optimized ELPIS and Hnsw

We optimized the search of both HNSW implementations from Hnswlib [1] and Elpis [2] using a single priority queue during search instead of double priority queues. This implementation is consistent with practices common in other graph methods for beam search and has shown to accelerate the search process for HNSW, attributed to its hardware-friendly nature, resulting in a performance up 2.8x faster on 1B scale dataset. In Figure 7, we illustrate how optimizing the beam search of HNSW implementation in Hnswlib has improved the HNSW code to be much faster while incurring a similar number of distance calculations.



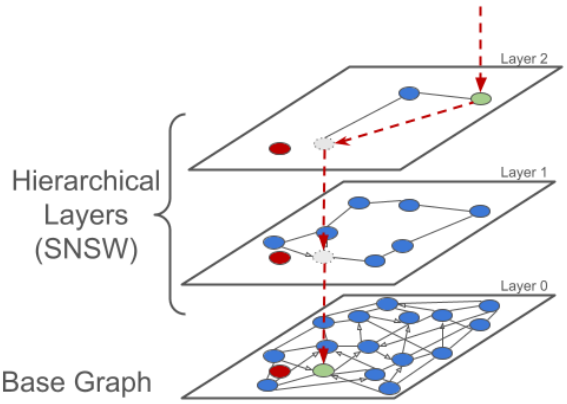
(a) Search Time on 25GB (b) Search Time on 100GB (c) Search Time on 1B

(d) Num. Dcs on 25GB (e) Num. Dcs on 100GB (f) Num. Dcs on 1B

Figure 5: Time and Number of distance calculations vs. accuracy on Deep1B dataset

HNSW Parameters and Hierarchical layers distribution

One of our findings suggests that employing hierarchical layers of multi-resolution stacked NSW graphs, as proposed by Yury Malkov and incorporated into the HNSW approach [98], can enhance search performance at a large scale. These structures draw inspiration from the skipped list data structure [a]. In the example below, our HNSW implementation consists of two hierarchical layers: layer 2 and layer 1, each containing two NSW graphs, namely the SNSW, and the base graph at level 0.



The SNSW is built incrementally by a multilevel sampling, For each node of the base graph, we randomly generate a respective maximum level L following the formula :

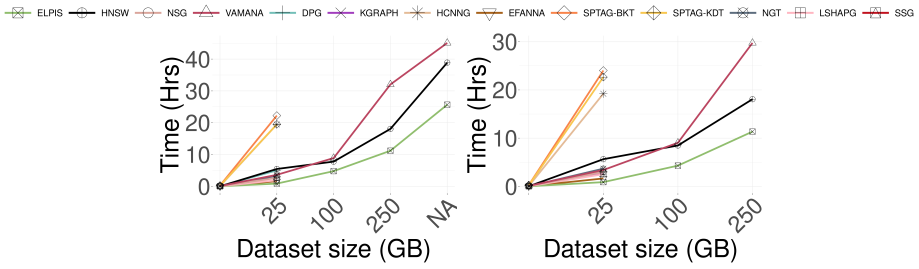
$$L = \frac{-ln(unif(0,1))}{ln(M/2)},$$

where unif(0,1) represents a uniformly random number between 0 and 1, ln() the natural logarithm, and M is base layer graph maximum outdegree to control the probability distribution of a node's maximum layer. A higher M increases the graph connectivity, thus HNSW concentrates nodes in lower layers and the number of levels decreases to save the distance calculations for the beam search at the base layer. Conversely, as M decreases, HNSW pushes more nodes to the top layer to improve the seed quality to improve the performance on the sparse graph. In the table below, we report the number and size of hierarchical layers for different values of M, on the Deep1M dataset.

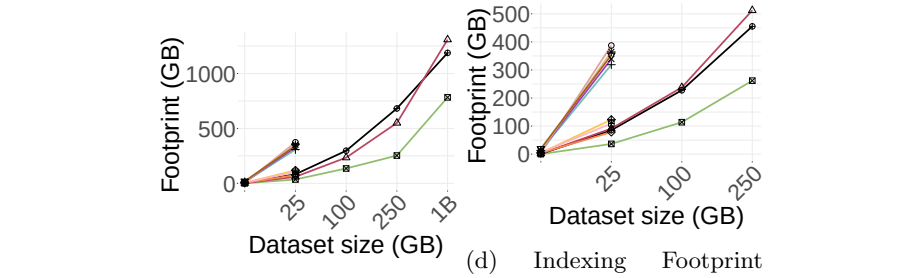
M	Number of hierarchical layers	HL1	HL2	HL3	HL4	HL5	HL6	HL7	HL8	HL9
5	9	200058	40583	7988	1581	304	76	11	3	1
10	6	100636	10060	988	116	9	1	0	0	0
15	5	67031	4438	292	18	1	0	0	0	0
20	4	50487	2501	140	7	0	0	0	0	0

The aim of the hierarchical layers is to establish coarse neighborhoods at the top layers, facilitating long jumps between nodes in the search space. As we descend through the layers, we encounter finer neighborhoods until reaching the base layer. During the search process, we initiate a greedy search from a predetermined node at the highest layer. Upon reaching a local minimum, we restart the search from that node in the layer below, iterating until locating the local minimum node at layer 1. Subsequently, the SNSW outputs this node for use as an entry point, initiating the beam search within a region close to the query at the base layer.

4 Indexing Time and Footprint

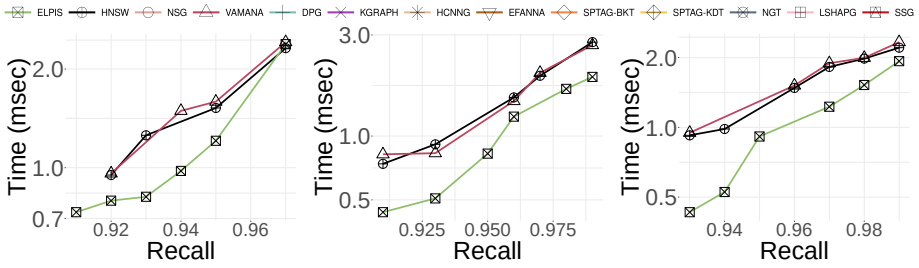


(a) Indexing Time (Sift) (b) Indexing Time (Sald)



(c) Indexing Footprint (Sift) (d) Indexing Footprint (Sald)

5 Search



(a) sald100GB (b) Deep250GB (c) sift250GB

Figure 7: Search