Efficient Approximate Nearest Neighbor Search via Data-Adaptive Parameter Adjustment in Hierarchical Navigable Small Graphs

Huijun Jin
Computer Science
Yonsei University
Seoul, Korea
jinhuijun@yonsei.ac.kr

Jieun Lee
Computer Science
Yonsei University
Seoul, Korea
jieun199624@yonsei.ac.kr

Shengmin Piao
Artificial Intelligence
Yonsei University
Seoul, Korea
shengminp@yonsei.ac.kr

Sangmin Seo
Computer Science
Yonsei University
Seoul, Korea
ssm6410@yonsei.ac.kr

Sein Kwon
Computer Science
Yonsei University
Seoul, Korea
seinkwon97@yonsei.ac.kr

Sanghyun Park†
Computer Science
Yonsei University
Seoul, Korea
sanghyun@yonsei.ac.kr

***Abstract*—Hierarchical Navigable Small World (HNSW) graphs are a state-of-the-art solution for approximate nearest neighbor search, widely applied in areas like recommendation systems, computer vision, and natural language processing. However, the effectiveness of the HNSW algorithm is constrained by its reliance on static parameter settings, which do not account for variations in data density and dimensionality across different datasets. This paper introduces Dynamic HNSW, an adaptive method that dynamically adjusts key parameters — such as the M (number of connections per node) and ef (search depth) — based on both local data density and dimensionality of the dataset. The proposed approach improves flexibility and efficiency, allowing the graph to adapt to diverse data characteristics. Experimental results across multiple datasets demonstrate that Dynamic HNSW significantly reduces graph build time by up to 33.11% and memory usage by up to 32.44%, while maintaining comparable recall, thereby outperforming the conventional HNSW in both scalability and efficiency.**

***Keywords*—Approximate Nearest Neighbor Search, Hierarchical Navigable Small World, Dynamic Parameter Tuning, Data-adaptive**

I. Introduction

Nearest Neighbor (NN) search is an essential and critical problem for tasks such as clustering, information retrieval in recommendation systems, and natural language processing [1]-[4]. K-Nearest Neighbor (KNN) search is one of the most widely used approaches for quickly and accurately identifying the nearest neighbors of a given data point [5]. However, as datasets grow rapidly in both size and dimensionality, traditional exact K-Nearest Neighbor (KNN) methods face significant computational challenges, making them impractical for real-time use in large-scale environments [6].

To address these scalability challenges consistent with approximate computing principles, various Approximate Nearest Neighbor (ANN) search algorithms have been introduced to mitigate computational overhead while ensuring high recall performance [7]-[10]. Among these, the Hierarchical Navigable Small World (HNSW) algorithm has emerged as one of the most effective solutions [11]. HNSW employs a graph-based approach designed to efficiently navigate large datasets and maintain high recall, making it particularly suitable for tasks like large-scale image retrieval, text mining, and personalized content recommendations [12]-[14].

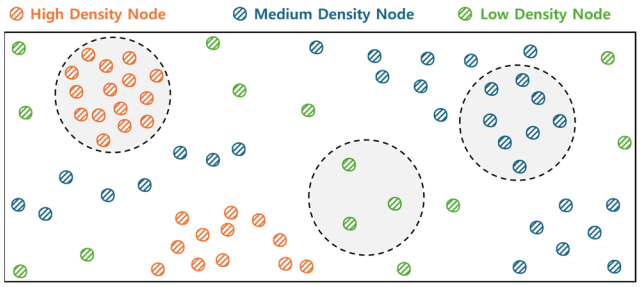

Fig. 1. Example of data density distribution in 2D space.

However, HNSW algorithm relies on static parameter settings for M (maximum number of neighbors) and ef (search depth) during graph construction. These parameters are fixed globally for the entire dataset, which can result in suboptimal performance when data density and dimensionality vary significantly [15]. As shown in Fig. 1, data points in a two-dimensional space exhibit varying densities: high-density nodes (orange) contain more neighbors within a fixed radius, while low-density nodes (green) have fewer neighbors, indicating sparsity. These variations influence parameter settings and impact ANN search performance [16].

High-density areas would benefit from higher M and ef values to capture more neighboring nodes, while low-density regions require lower values to avoid wasting resources. Static settings for these parameters fail to adapt to these local characteristics, leading to inefficiencies—either by missing key neighbors in dense regions or consuming excessive resources in sparse ones. Understanding these density variations enables a more adaptive and efficient approach. Unfortunately, the static nature of HNSW's parameter settings overlooks these variations, resulting in reduced efficiency.

† Corresponding Author

978-3-9826741-0-0/DATE25/© 2025 EDAA

This issue of static parameter settings is not unique to HNSW. Other graph-based methods, such as the Navigating Spreading-out Graph (NSG) [17], DiskANN [18], and EFANNA [19], also struggle with globally fixed parameters, which limit their adaptability to different data characteristics. Although some techniques, like learned indices and adaptive termination strategies [20]-[22], have explored adaptive mechanisms, they are often bound by either global adjustments, tailored heuristics, or tight assumptions on the underlying data distribution. Consequently, these approaches do not fully address the need for a more fine-grained and generalizable mechanism that can dynamically respond to heterogeneous data densities without extensive manual tuning.

This paper presents Dynamic HNSW (DHNSW), a novel adaptation of the HNSW algorithm that introduces, for the first time, data-adaptive parameter tuning at the node level. By enabling more localized adjustments that better reflect the heterogeneous density variations within the dataset, DHNSW overcomes the limitations of static or globally adaptive methods. Traditional HNSW relies on globally fixed values for key parameters, such as M and ef, which can result in inefficiencies when data densities and dimensionalities vary across the dataset. Additionally, even within the same dataset, there are nodes with varying data densities, further complicating the optimal tuning of these parameters. To address this, DHNSW introduces a data-adaptive framework that dynamically adjusts parameters based on local data characteristics. By leveraging a density estimation technique (Random Projection - K Nearest Neighbors, RP-KNN [23]), DHNSW increases its parameters in high-density data points to enhance connectivity and recall, while reducing them in low-density areas to optimize memory usage and computation costs. This approach improves scalability and significantly reduces graph build time and memory usage, while still maintaining robust recall, providing a flexible solution that adapts without manual tuning.

This study offers the following key contributions:

- **Dynamic Parameter Optimization**: Introduces a novel method to adaptively adjust HNSW parameters based on local data characteristics. To the best of our knowledge, this approach is the first to provide localized, adaptive control at the node level, enabling more efficient handling of varying data densities.
- **Data-Adaptive Algorithmic Framework**: Develops an algorithmic approach that dynamically fine-tunes key parameters as the graph is constructed, ensuring improved efficiency across diverse data distributions.
- **Comprehensive Evaluation and Analysis**: Demonstrates DHNSW's effectiveness through extensive experiments, showing reductions in build time and memory usage while maintaining competitive recall, confirming its adaptability and scalability for large-scale, high-dimensional ANN search.

II. Background and Motivation

First introduced by Malkov and Yashunin [11], HNSW has established itself as a cornerstone in ANN search due to its high efficiency and strong recall performance [12]-[14]. The algorithm constructs a hierarchical, multi-layer proximity graph, where each layer progressively refines the dataset representation. The top layer contains a small set of nodes offering a broad, global perspective, while lower layers present increasingly denser and more fine-grained neighborhoods. Each node represents a data point, connected to its nearest neighbors via edges formed by heuristic similarity measures. By traversing from the coarser, upper layers down to the denser, lower layers, HNSW efficiently approximates nearest neighbors at various scales. This tiered, navigable structure is key to HNSW's success in combining speed, accuracy, and scalability.

Two critical parameters dominate the graph-building process in HNSW: M, which defines the maximum number of connections a node can have, and ef, the size of the candidate list during graph construction. A higher M value creates a denser graph, improving search recall but increasing memory usage and computational costs. Similarly, a larger ef value enables deeper graph exploration, enhancing recall at the expense of greater computational overhead.

Static values for M and ef are often inefficient due to varying data densities. In high-density regions, a fixed M may fail to provide sufficient connections to capture the local neighborhood structure, and a fixed ef may limit search depth, reducing recall. Conversely, in low-density regions, the same static values can lead to unnecessary connections and excessive search depths, causing increased memory usage and redundant computations without significant recall gains. Adaptive tuning of M and ef based on local density is thus critical to balance memory, computational efficiency, and search performance.

Real-world datasets often exhibit regions with varying data densities [24]. Parameters optimized for one density level may lead to increased memory usage or reduced recall in others, creating trade-offs in performance. These limitations become more pronounced as dataset size and dimensionality grow, resulting in longer graph build times and higher memory consumption [11], [12], [17]. To address this, we propose a data-adaptive parameter setting approach that dynamically adjusts M and ef based on local data characteristics, optimizing performance and resource efficiency across all density levels.

III. Methodology

The DHNSW graph-building process involves several key steps: first, the ranges for the parameters M and ef are determined by computing the density and dimensionality of the dataset, ensuring that the algorithm adapts to the data structure. Next, for each node, these parameters are dynamically adjusted based on the node's local density estimation. Nodes with higher density estimations are assigned higher M and ef values to ensure sufficient connectivity and recall, while nodes in sparser regions are assigned lower values to optimize memory and computational efficiency. These dynamically adjusted parameters are then used to insert the nodes into the HNSW graph. The detailed process for constructing the DHNSW graph is outlined in Algorithm 1.

Input: The algorithm takes several key inputs, with M_{init} and ef_{init} being user-provided values for the initial value of M and ef, respectively. If the user does not specify these values, default parameters are used, where M is set to 16 and ef is set to 100.

Algorithm 1: DHNSW Graph Construction

Input: Initial parameters M_{init}, ef_{init}, Dataset D, # of data n, Data point x

Output: Updated HNSW with all elements inserted

1 Calculate local data density, $\rho \leftarrow$ RP-KNN (D)

2 Set a reference point, $ef_{ref} \leftarrow ef_{init} + \left(\frac{dim(D)}{\alpha}\right)^2$

3 Calculate boundaries $M_{low}, M_{high}, ef_{low}, ef_{high}$ based on M_{init} and ef_{ref}

4 **for** $q = 1$ to n **do**

5 $M_q \leftarrow M_{low} + \left(\frac{\rho_q - \rho_{min}}{\rho_{max} - \rho_{min}}\right) \times (M_{high} - M_{low})$

6 $ef_q \leftarrow ef_{low} + \left(\frac{\rho_q - \rho_{min}}{\rho_{max} - \rho_{min}}\right) \times (ef_{high} - ef_{low})$

7 Insert x_q into HNSW based on M_q and ef_q

8 **end for**

9 **return** Updated HNSW

The dataset D consists of the data provided by the user, with n representing the total number of data points, and each data point is denoted as x.

Line 1: We calculate local data density $\rho = \{\rho_1, \dots \rho_n\}$ of dataset $D = \{x_1, \dots, x_n\}$, where $x_i \in \mathbb{R}^d$ is a vector data point of dimension d, using the RP-KNN method. The RP-KNN method efficiently estimates density in high-dimensional spaces by projecting the data onto a lower-dimensional subspace using random projections [23]. This reduces the dimensionality of the data while preserving the relative distances between data points with high probability, according to the Johnson-Lindenstrauss lemma [25]-[27]. In the projected subspace, a KNN search is then performed to identify the K nearest neighbors of each data point. The density is estimated based on the distances to neighboring points, where shorter distances indicate higher density and longer distances suggest lower density. This method reduces the computational complexity of local density estimation while maintaining accuracy, enabling efficient handling of high-dimensional datasets.

Line 2: To address the curse of dimensionality [28], where distances between data points become increasingly indistinguishable as dimensionality grows, we calculate ef_{ref} as a reference point fo ef_{init} r. This calculation is based on the dataset's dimensionality, ensuring an appropriate adjustment of the search depth for each data point, as follows:

$$ef_{ref} = \left\lfloor ef_{init} + \left(\frac{dim(D)}{\alpha}\right)^2 \right\rfloor \quad (1)$$

Equation (1) is used to adjust the ef_{init} value based on the dimensionality of the dataset. The parameter α, referred to as the adjustment factor, controls the magnitude of this adjustment, allowing the algorithm to adapt dynamically to different dataset dimensions and optimize performance across varying datasets. In this formula, we use the floor function to ensure that ef_{ref} is an integer, rounding down any non-integer results to maintain valid values for graph construction and search processes. The floor function also applies to all other formulas in the paper where M and ef are involved, since these parameters must always be integers.

Line 3: In order to dynamically adjust the key parameters (M_q and ef_q) for each node x_q, we first compute the mean (ρ_μ) and standard deviation (ρ_σ) of the local data density values. These statistics allow the algorithm to set appropriate parameter ranges that are flexible and adaptive to the dataset's local characteristics.

To define the range for M_q, the algorithm dynamically adjusts its lower and upper bounds based on local density characteristics. The lower bound M_{low}, defined in (2), is set to a minimum value of 2 to ensure that even low-density nodes have at least two connections. This threshold acts as a practical safeguard to maintain the graph's stability and functionality. Meanwhile, the upper bound M_{high}, defined in (3), allows for more connections in denser nodes, enhancing recall performance.

These bounds are determined using the coefficient of variation (CV) [29], $\frac{\rho_\sigma}{\rho_\mu}$, which quantifies the relative variability in local density by normalizing the standard deviation (ρ_σ) relative to the mean (ρ_μ). Unlike absolute measures like the mean and variance, which provide only a global view, the CV captures local density fluctuations, making it particularly effective in high-dimensional datasets. By incorporating the CV, the algorithm adapts M_q to relative changes in density, ensuring that connections are dynamically scaled to reflect the data's local characteristics.

$$M_{low} = max\left(2, \left\lfloor M_{init} - M_{init} \times \left(\frac{\rho_\sigma}{\rho_\mu}\right) \times \lambda \right\rfloor\right) \quad (2)$$

$$M_{high} = \left\lfloor M_{init} + M_{init} \times \left(\frac{\rho_\sigma}{\rho_\mu}\right) \times \lambda \right\rfloor \quad (3)$$

The search depth ef_q is dynamically adjusted to balance recall and computational efficiency by defining its bounds based on local density. The minimum search depth, ef_{low}, calculated using (4), is capped at 10 to ensure sufficient candidate searches and maintain recall even in sparse nodes. Conversely, the upper bound ef_{high}, defined in (5), allows for deeper searches in denser regions, ensuring that more accurate nearest neighbors are found as density increases. These bounds are defined as follows:

$$ef_{low} = max\left(10, \left\lfloor ef_{ref} - ef_{ref} \times \left(\frac{\rho_\sigma}{\rho_\mu}\right) \times \lambda \right\rfloor\right) \quad (4)$$

$$ef_{high} = \left\lfloor ef_{ref} + ef_{ref} \times \left(\frac{\rho_\sigma}{\rho_\mu}\right) \times \lambda \right\rfloor \quad (5)$$

Line 4-7: Given the calculated bounds, M_q and ef_q are dynamically adjusted for each data point x_q based on its local density. The dynamic M_q value, calculated using (6), scales the number of connections proportionally between M_{low} and M_{high} based on the local density ρ_q. Similarly, the search depth ef_q, defined in (7), adjusts proportionally between ef_{low} and ef_{high}, ensuring that high-density nodes benefit from deeper searches while avoiding excessive computations in low-density nodes.

These dynamic adjustments optimize both connectivity and search efficiency across diverse density regions:

$$M_q = M_{low} + \left\lfloor \left(\frac{\rho_q - \rho_{min}}{\rho_{max} - \rho_{min}} \right) \times (M_{high} - M_{low}) \right\rfloor \quad (6)$$

$$ef_q = ef_{low} + \left\lfloor \left(\frac{\rho_q - \rho_{min}}{\rho_{max} - \rho_{min}} \right) \times (ef_{high} - ef_{low}) \right\rfloor \quad (7)$$

Finally, each data node, with its dynamically adjusted parameters, is inserted into the HNSW graph.

IV. Experiments

A. Experimental Setup

The experiments were performed on a server configured with a 12th Gen Intel(R) Core(TM) i7-12700F CPU and 94GB of RAM, providing sufficient computational capacity to process large-scale datasets. The benchmark datasets used in this study include MNIST [30], GloVe100K [31], SIFT1M [32], and GIST1M [33]. These datasets, detailed in Table I, were specifically chosen to represent a wide range of data characteristics, including variations in density distributions and feature dimensions, providing a comprehensive evaluation of the proposed DHNSW algorithm across different scenarios. For example, MNIST represents relatively uniform density in a lower-dimensional space, while SIFT1M and GIST1M exhibit more complex, high-dimensional distributions.

The aim of the experiments is to assess the performance improvements of the DHNSW algorithm compared to the vanilla HNSW. We designed the experiments with a focus on three key metrics: build time, memory usage, and recall. The purpose of these evaluations is to demonstrate how dynamic parameter adjustment in DHNSW addresses the limitations of static parameter settings in HNSW. This allows us to validate that DHNSW can efficiently scale across datasets of varying densities and dimensionalities, which is critical for optimizing both speed and resource use.

Build Time: This metric measures the total time required to build the HNSW graph for each dataset. It is essential for understanding the efficiency of the graph construction process, particularly for large datasets where the time cost can become significant. A reduction in build time indicates improved computational efficiency, which is critical for real-time applications.

Memory Usage: Defined as the peak memory consumption during the graph construction phase, this metric assesses the algorithm's ability to manage memory resources effectively. Lower memory usage is particularly beneficial in environments with limited memory capacity, making the algorithm more suitable for deployment in a range of computational settings.

TABLE I. Dataset Detail

Dataset	Dimension	Size	# of Samples
MNIST	784	52 MB	60,000
GloVe100K	300	990 MB	100,000
SIFT1M	128	550 MB	1,000,000
GIST1M	960	5.37 GB	1,000,000

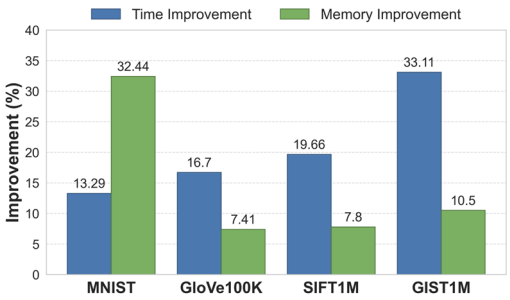

Fig. 2. Build time and memory usage improvements across datasets.

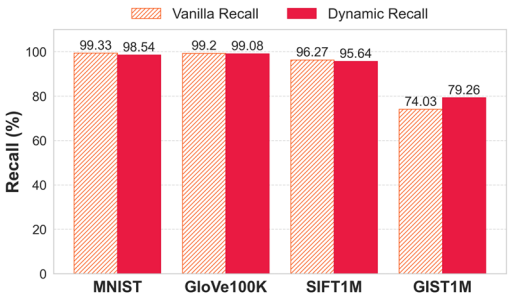

Fig. 3. Recall comparison between vanilla HNSW and DHNSW.

Recall: This metric is calculated as the ratio of true nearest neighbors correctly identified by the algorithm out of the total number of true nearest neighbors. Recall assesses how effectively the algorithm captures relevant neighbors within the top-k results, thereby reflecting the quality of constructed graph.

B. Baseline Performance Comparison Across Datasets

This experiment aims to establish a baseline performance comparison between the vanilla HNSW and the proposed DHNSW across four benchmark datasets. We set the parameters with initial M as 16, initial ef as 100, α as 100, and λ as 1.5, which serves as the default values for the experiments. The results are depicted in Fig. 2 and 3, which highlight the significant improvements achieved by the DHNSW algorithm over the vanilla HNSW across all evaluated metrics.

Build Time Improvement: DHNSW demonstrated a notable performance improvement in build time compared to the vanilla HNSW, as shown in Fig. 2. Specifically, for the larger datasets, such as SIFT1M and GIST1M, the improvements were significantly enhanced. This enhancement is attributed to the dynamic adjustment of the parameters M and ef, which allows the algorithm to adapt to local density variations and dimensions, optimizing the graph construction process and minimizing redundant computations. Specifically, the GIST1M dataset, with the highest dimensionality, saw the most significant reduction in build time(33.11%), demonstrating the scalability of the DHNSW approach.

Memory Usage Improvement: As depicted in Fig. 2, the memory usage results indicate the advantage of the DHNSW in managing memory consumption more efficiently than the vanilla HNSW. Across all datasets, the DHNSW required less peak memory, with reductions ranging from 7.41% to 32.44%, depending on the dataset size and complexity. These savings were particularly evident in datasets with varying density

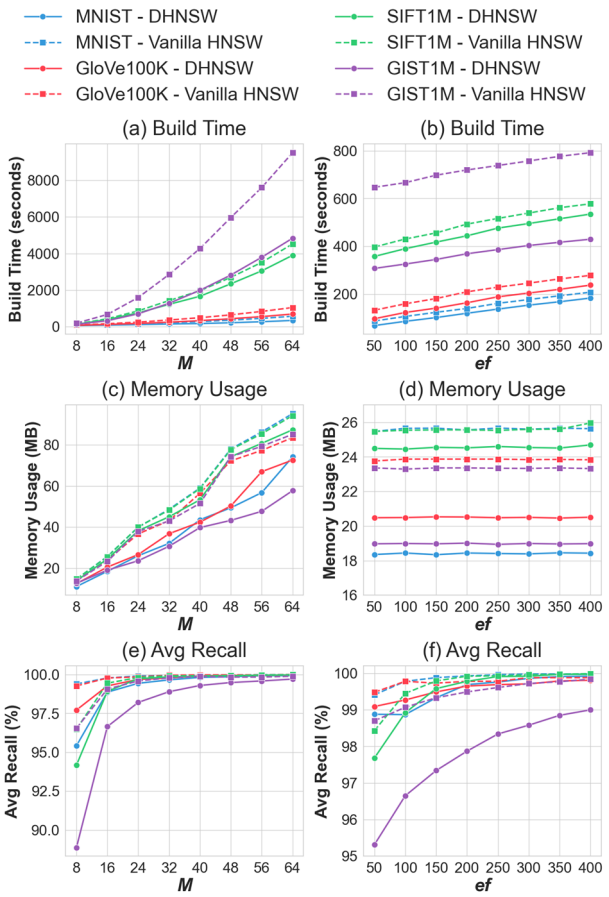

Fig. 4. Performance comparison across varying $\boldsymbol{M}$ and $\boldsymbol{ef}$ values for DHNSW and vanilla HNSW.

distributions, where dynamic parameter adjustments prevented over-allocation of memory resources. For example, the MNIST dataset experienced a 32.44% reduction in memory usage, while the GIST1M dataset showed a 10.5% decrease, further validating the effectiveness of the DHNSW in various scenarios.

Recall Trade-Off: Fig. 3 illustrates that DHNSW achieves high recall rates across all datasets, closely matching or even exceeding those of vanilla HNSW. The recall rates for MNIST, GloVe100K, and SIFT1M datasets remain above 95%, demonstrating DHNSW's effectiveness in preserving recall. Notably, for the GIST1M dataset, DHNSW achieves a recall rate of 79.26%, showing significant improvement and effective handling of high-dimensional data. This result confirms that DHNSW maintains a strong balance between recall and computational efficiency, making it well-suited for large-scale applications requiring both high performance and resource efficiency.

C. *Robustness Analysis Across Different Initial Parameter Settings*

The second experiments evaluate the robustness of the DHNSW algorithm under various initial parameter settings. To enable efficient testing while preserving dataset characteristics, a subset of 10,000 data points was selected from each dataset. The performance of DHNSW is compared to vanilla HNSW by systematically varying two critical parameters, M and ef.

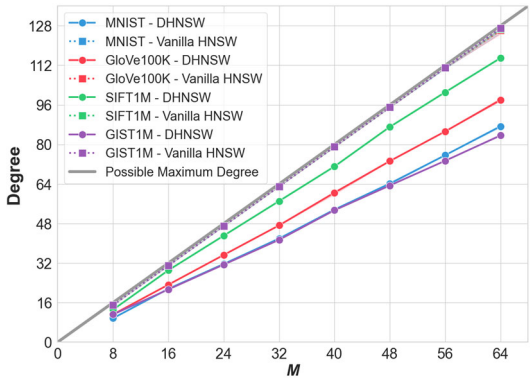

Fig. 5. Degree variation with $\boldsymbol{M}$ parameter adjustments.

Specifically, the M parameter was varied from 8 to 64 in increments of 8, while the ef parameter was adjusted from 50 to 400 in equal in increments of 50, ensuring comprehensive coverage of parameter space during the experiment. For these evaluations, α and λ were set to their default values of 100 and 1.5, respectively.

Build Time: As observed in Fig. 4 (a) and (b), which depict the build time across different values of M and ef, respectively, DHNSW consistently achieves a reduction in build time compared to vanilla HNSW across all datasets. For example, across all datasets, the build time for DHNSW is consistently lower than that of vanilla HNSW. On average, DHNSW demonstrates approximately 29.04% improvement in build time, with particularly notable reductions in higher-dimensional datasets like GIST1M. This result highlights DHNSW's ability to dynamically adapt to varying data densities without incurring significant computational costs.

Memory Usage: The memory usage results, as shown in Fig. 4. (c) and (d), indicate that DHNSW consistently requires less memory than vanilla HNSW across all tested configurations. On average, DHNSW reduces memory usage by about 20.70%, with savings particularly evident in datasets with varying density distributions. This dynamic adjustment of resource usage based on local data density confirms DHNSW's adaptability to different data characteristics while maintaining performance.

Average Recall: Fig. 4 (e) and (f) illustrate the average recall rates for various values of M and ef. DHNSW demonstrates a competitive recall performance, closely approaching that of vanilla HNSW across most configurations. While there are slight differences in recall rates at lower values of M and ef, DHNSW achieves nearly equivalent recall as the parameters increase, especially in datasets like GIST1M. This shows that DHNSW can reach high recall levels with moderate parameter adjustments, effectively balancing computational efficiency with performance. Overall, DHNSW maintains a strong recall rate, proving itself to be a scalable solution with minimal compromise on recall across diverse settings.

To further evaluate DHNSW's efficiency, we conducted an additional analysis on the average degree of nodes while varying the parameter M. In this experiment, we observed that the vanilla HNSW algorithm maintained an average degree close to the maximum possible degree, defined as twice the user-specified M

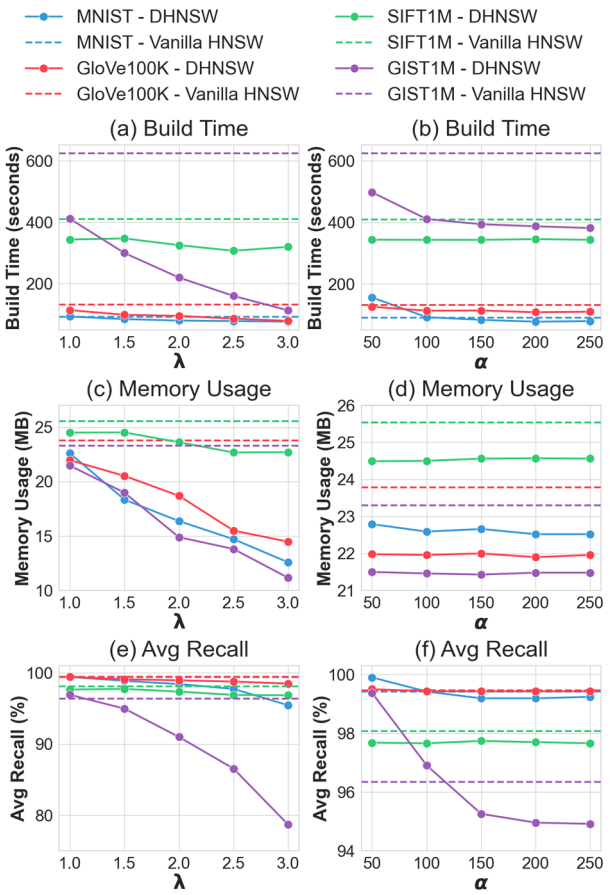

Fig. 6. Impact of expansion factor (λ) and adjustment factor (α) on the performance of DHNSW and vanilla HNSW.

value, across all datasets. This fixed connectivity results in a high memory footprint and computational overhead.

In contrast, DHNSW showed a noticeable reduction in average degree as M increased, dynamically adjusting node connections based on local data density. As demonstrated in the second experiment, this reduction directly leads to lower memory usage, allowing DHNSW to be more memory-efficient by avoiding unnecessary connections in high-density regions. This adaptive behavior not only improves memory efficiency but also enhances scalability, particularly for large datasets, while maintaining a high recall rate comparable to that of vanilla HNSW.

D. Effect of Different Hyperparameter Settings

Furthermore, the performance of DHNSW was analyzed with respect to two key hyperparameters: the expansion factor (λ) and the adjustment factor (α). Unlike the vanilla HNSW, which does not consider dataset-specific characteristics, the DHNSW algorithm allows for the flexible adjustment of key hyperparameters. These hyperparameters enable fine-tuning of the algorithm to better align with the varying density and dimensionality of different datasets, optimizing the balance between build time, memory usage, and recall performance. By setting these parameters appropriately, DHNSW can effectively adapt its behavior to diverse real-world applications. In this experiment, the M and ef values were kept at their default settings of 16 and 100, respectively, and the same subset of 10,000 data points per dataset was used as in this experiment.

Effect of Expansion Factor (λ): As λ increases, DHNSW demonstrates consistent reductions in build time and memory usage across all datasets (Fig. 6 (a) and (c)), with especially notable improvements in GIST1M. This effect is due to the higher resource demands required by high-dimensional data to manage distance calculations and connections. By increasing the expansion factor, DHNSW effectively adapts to these demands, reducing computational overhead. In contrast, other datasets exhibit either stable or slightly improved performance, highlighting the critical role of the expansion factor in optimizing graph construction, particularly for complex, high-dimensional datasets.

Regarding average recall (Fig. 6 (e)), while increasing λ leads to some decrease in recall for GIST1M, other datasets maintain stable recall levels. For GIST1M, the trade-off between recall and efficiency gains in build time and memory usage highlights DHNSW's ability to optimize computational performance while still preserving competitive recall in complex, high-dimensional cases.

Effect of Adjustment Factor (α): As shown in Fig. 6 (b) and (d), the adjustment factor α has a relatively stable impact on build time and memory usage across datasets, with minor fluctuations. However, Fig. 6 (f) shows that lower values of α yield higher recall, especially in GIST1M. This suggests that fine-tuning α can help DHNSW achieve optimal recall rates while maintaining efficient performance across different parameter settings.

V. Conclusion

This study proposed DHNSW, an enhanced version of the HNSW algorithm that dynamically adjusts the key parameters M and ef based on local data characteristics. This represents the first attempt in the field to introduce dynamic parameter tuning at the node level in an HNSW framework. The experimental results across multiple benchmark datasets demonstrate that DHNSW significantly improves graph build time and memory usage while maintaining competitive recall rates. However, one of the future directions for enhancing DHNSW is to develop strategies for automatically adjusting the expansion factor λ and adjustment factor α to suit different dataset characteristics. By incorporating adaptive mechanisms that can dynamically tune these hyperparameters based on the specific data distribution, DHNSW could achieve an even better balance between recall and efficiency. This approach may involve more advanced density estimation techniques or learning-based methods, ultimately making DHNSW more adaptable and robust across diverse real-world applications.

Acknowledgment

This work was supported by the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korean government (MSIT) (IITP-2017-0-00477, (SW starlab) Research and development of the high performance in-memory distributed DBMS based on flash memory storage in an IoT environment).

REFERENCES

[1] S. Dasgupta and Y. Freund, "Random Projection Trees and Low Dimensional Manifolds," in Proceedings of the ACM Symposium on Theory of Computing, 2008, pp. 537–546.

[2] G. Salton, "The SMART Retrieval System—Experiments in Automatic Document Processing," Prentice Hall, 1971.

[3] J. Wang, N. Zhang, J. Jiang, and Y. Gong, "Fast Approximate Nearest Neighbor Search with the Hierarchical Navigable Small World Graph," in Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS), 2017.

[4] J. Liu and W. B. Croft, "Cluster-based retrieval using language models," in Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, 2004, pp. 186-193.

[5] C. Cover and P. Hart, "Nearest neighbor pattern classification," IEEE Trans. Inf. Theory, vol. 13, no. 1, pp. 21-27, 1967.

[6] A. Andoni, P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor search," in Proc. IEEE Symp. Foundations of Computer Science (FOCS), pp. 459-468, 2006.

[7] C. Liu, M. Yu, and J. Hu, "Memory-Efficient High-Dimensional Approximate Nearest Neighbor Search," in Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD), 2020, pp. 1367-1376.

[8] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with GPUs," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 41, no. 9, pp. 2077-2090, Sep. 2019.

[9] B. Kulis, P. Jain, K. Grauman, "Fast Similarity Search for Learned Metrics," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 3 1, no. 12, pp. 2143-2157, 2009.

[10] W. Dong, C. Moses, and K. Li, "Efficient K-Nearest Neighbor Graph Construction for Generic Similarity Measures," Proceedings of the 20th International Conference on World Wide Web, 2011, pp. 577-586.

[11] Y. Malkov and D. Yashunin, "Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs," IEEE Trans. Pattern Anal. Mach. Intell., vol. 40, no. 4, pp. 824-836, April 2018.

[12] M. Aumueller, E. Bernhardsson, and A. Faithfull, "ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," Proceedings of the 31st International Conference on Neural Information Processing Systems (NeurIPS), Long Beach, CA, USA, Dec. 2017, pp. 851-863.

[13] D. Fu, J. Liu, and B. Cheng, "HNSW-based large-scale image retrieval for personalized recommendations," IEEE Access, vol. 8, pp. 137892-137904, 2020.

[14] J. Zhang, X. Xu, and S. Liu, "Acceler ating HNSW for large-scale approximate nearest neighbor search on GPUs," Proceedings of the 2021 IEEE International Conference on Big Data (Big Data), Orlando, FL, USA, Dec. 2021, pp. 1248-1255.

[15] S. Harwood and T. Drummond, "FANNS: Fast approximate nearest neighbor search with variable parameter settings," Proceedings of the 2021 IEEE International Conference on Computer Vision (ICCV), Montreal, QC, Canada, Oct. 2021, pp. 1094-1103.

[16] C. Foster and B. Kimia, "Computational Enhancements of HNSW Targeted to Very Large Datasets," in *Lecture Notes in Computer Science*, vol. 14289, Springer, 2023.

[17] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast Approximate Nearest Neighbor Search with the Navigating Spreading-out Graph," Proceedings of the VLDB Endowment, vol. 12, no. 5, pp. 461-474, 2019.

[18] S. Subramanya, M. K. Somasundaram, and D. P. Singh, "DiskANN: Fast and Accurate Billion-scale Nearest Neighbor Search on a Single Node," IEEE Transactions on Big Data, vol. 7, no. 3, pp. 561-573, Sept. 2021.

[19] L. Wang, J. Wang, J. Li, and H. Li, "EFANNA: An Extremely Fast Approximate Nearest Neighbor Search Algorithm Based on kNN Graph," Proceedings of the 22nd ACM International Conference on Multimedia, Orlando, FL, USA, Nov. 2014, pp. 757-760.

[20] A. Kraska, A. Beutel, E. H. Chi, J. Dean, and N. Polyzotis, "The Case for Learned Index Structures," Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data, Houston, TX, USA, 2018, pp. 489-504.

[21] H. Zhang, Q. Li, and S. Jiang, "G-Learned Index: Enabling Efficient Learned Index on GPU," IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 10, pp. 2934-2947, Oct. 2021.

[22] Z. Zhong, Y. Zhang, Y. Chen, C. Li, and C. Xing, "Learned index on GPU," in 2022 IEEE 38th International Conference on Data Engineering Workshops (ICDEW), Kuala Lumpur, Malaysia, May 2022, pp. 117-122.

[23] E. B. Tavakoli, A. Beygi, and X. Yao, "RPkNN: An OpenCL-Based FPGA implementation of the dimensionality-reduced kNN algorithm using random projection," IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 30, no. 4, pp. 549-552, Apr. 2022.

[24] A. Dasgupta, R. Kumar, and T. Sarlós, "Fast locality-sensitive hashing," in Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), San Diego, CA, USA, 2011, pp. 1073-1081.

[25] W. B. Johnson and J. Lindenstrauss, "Extensions of Lipschitz mappings into a Hilbert space," Contemporary Mathematics, vol. 26, pp. 189-206, 1984.

[26] S. Dasgupta and A. Gupta, "An elementary proof of the Johnson-Lindenstrauss lemma," Random Structures & Algorithms, vol. 22, no. 1, pp. 60-65, Jan. 2003.

[27] D. Achlioptas, "Database-friendly random projections: Johnson-Lindenstrauss with binary coins," Journal of Computer and System Sciences, vol. 66, no. 4, pp. 671-687, Jun. 2003.

[28] R. Bellman, Adaptive Control Processes: A Guided Tour, Princeton University Press, 1961.

[29] A. K. Gupta and S. Nadarajah, "Handbook of Beta Distribution and Its Applications," Statistics: A Series of Textbooks and Monographs, vol. 175, Chapman and Hall/CRC, 2004.

[30] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[31] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 2014, pp. 1532-1543.

[32] H. Jegou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 33, no. 1, pp. 117-128, Jan. 2011.

[33] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," International Journal of Computer Vision, vol. 42, no. 3, pp. 145-175, May 2001.