Comparison of Hierarchical Navigable Small World Querying Variants

Mohamed Affan Dhankwala MDHANKW1@JH.EDU
Masters of Artificial Intelligence
Johns Hopkins University
Baltimore MA, USA

Abstract

High dimensional data is defined as data points with a high number of features–implying that each data point is defined by a high number of attributes. While expensive, high dimensional data has found its way into crucial fields such as computer vision, natural language processing (NLP) and others. Traditional techniques are either too slow or simply unable to account for the high dimensionality of these datasets. Approximate nearest neighbor (ANN) attempts to counter this concern by estimating the nearest neighbors to any query rather than computing them exactly. Navigable small world (NSW) is a technique based on ANN consisting of nodes connected by edges but frequently terminates at local minimums. Hierarchical navigable small world (HNSW) attempts to mitigates this issue by layering the graph at different levels of abstraction but is still susceptible to premature termination. This paper identifies two HNSW variants: Fast inference for graph-based approximate nearest neighbor search (FINGER) and distance adaptive beam search (DABS) that claim to reduce querying computations and improve recall rates, respectively. These variants are tested on four GloVe datasets ranging from 25 to 200 dimensions where we measure the comparison counts and recall rates of the models when retrieving k neighbors with k values between 5 and 50. FINGER managed to match and exceed HNSW recall rates at high levels of K while calculating fewer distances. DABS demonstrated high performance at the cost of calculating more distances on the 25 dimension dataset but failed to beat the other two variants recall rates on the other datasets due to poor tuning caused by computational resource limitation. It's performance also inverted at higher values of k on the 200 dimension dataset, hinting at some potential high dimensional proficiency. However, limited by dataset availability and computational resources, this claim could not be further analyzed. These results also do not support HNSW as being well performing in high-scale high-dimensional spaces at high levels of k. Further research could attempt to analyze the topological structure of HNSW layers and research HNSW-specific graph traversal methods that incorporate layer height and sparsity in their heuristic calculation.

1 Introduction

High dimensional data has become ever more prominent with today's grand computational power. Domains with high dimensional data include computer vision, natural language processing (NLP), fraud-detection services, gene classifiers, and recommender systems (such as movie or song recommenders). High dimensional data is categorized as data with a large number of features. Features are attributes of a data point and simple examples include color, weight, height, etc. High dimensional data can include hundreds or thousands

of features per data point. As one can imagine, analysis on this data can be extremely expensive with simple brute force methods. Bellman highlights the "curse of dimensionality" upon high dimensional datasets. This concept encompasses the inability for traditional data analysis to handle the extremely high dimensional data due to increased sparsity, lower distances between data points, and the exponentially growing computational requirements (Bayer et al.). To address the divergence in analysis methodologies and data size, specialized algorithms have been developed to efficiently handle and utilize these high dimensional data points.

Approximate nearest neighbor (ANN) search is one of the first of these specialized methods that attempts to handle high dimensional data. This technique introduces the concept of nodes. A node is a point in space representing a data point and has edges connecting it to other nodes–known as neighbors. ANN creates nodes for all data points and then connects simlar nodes with one another during the training process. This training process involves backpropagation to adjust the weights of the edges in a flexible manner (LeCun et al.). ANNs have become foundational as analysis tools in a widespread of high dimensional fields such as image recognition, speech processing, and autonomous systems. This model has also served as the backbone of the two other techniques we introduce.

The navigable small world (NSW) model was devised by pulling the node and neighbor concept from ANN and creating proximity graphs where nodes are connected to a subset of their nearest neighbors. Querying by traversing these proximity graphs is quick and allows NSW to suit the needs of systems with high dimensional data that query quickly and often. However, NSWs suffer from some shortcomings including employing querying tactics that involve some sort of greedy search and therefore sometimes returning suboptimal results due to being stuck at local minimums. Although offset slightly, utilizing heuristic search methods does not remove this issue entirely. A solution proposed by Malkov and Yashunin was the hierarchical navigable small world (HNSW). This model borrows the proximity graphs concept from NSW and creates multiple NSWs layered on top of one another. Higher layers contain fewer nodes than lower layers and the highest layer contains at most one or two nodes–these being the starting nodes. Querying through this structure is done by beginning at the starting nodes on the highest layer and working downwards. This model is well suited for high-scale, high-dimensional data due to its exceptional recall and logarithmic computational speed (Hajebi et al.). The downsides of this model primarily revolve around the extremely high storage cost of the structure and the imminent small chance of premature termination by being stuck in local minimum. This paper will present more information on HNSWs in a later section.

1.1 Introduction to Variants

While HNSW graphs have demonstrated impressive achievements when performing approximate nearest neighbor search on datasets, they are not without limitations. Querying speed, seen as one of the advantages of the HNSW structure, can degrade considerably when dealing with high dimensional data due to complex data distributions (Malkov and Yashunin). Although this degradation is not enough for it to be subpar to the other previously mentioned searching models, it limits the usefulness of this structure in high dimensional domains. These deficiencies call for the exploration of alternative querying variants.

One variant is known as the fast inference for graph-based approximate nearest neighbor search (FINGER). This method refines graph traversal by addressing and pruning "unnecessary intermediate" distance calculations by incorporating a distance function derived from angles between neighboring residual vectors (Chen et al.). By pre-computing these residuals, our FINGER-based HNSW stores more information during the building stage than a standard HNSW structure with the promise of quicker querying time that does not decay as quickly with increased dimensions.

Another variant to improve the performance of HNSW structures is known as distance adaptive beam search (DABS). HNSW structures do not simply consider one node at a time during the querying process, but rather a set of nodes. This set is known as the beam and the width of this beam correlates to the number of candidate nodes to compare the querying node with. However, HNSW maintains a fixed-sized beam thus failing to grasp higher candidate nodes in dense areas of the graph and conversely considering too many candidate nodes in sparse portions of the graph. DABS attempts to combat this deficiency by introducing a dynamically sized beam during traversal. This is to traverse a graph more efficiently and avoid premature convergence on local optima–indeed HNSW can still sometimes prematurely conclude on a local minimum as addressed by Malkov and Yashunin. This dynamically allocated beam size improves recall rates without much increase in computational overhead (Bezzina and Nicosia) making it well suited for addressing high-scale and high dimensional datasets.

1.2 Contributions

Both of the previously mentioned querying methodologies, FINGER and DABS, focus on improving the querying speed and recall, respectively, of an HNSW structure and both of them also claim to not hinder other metrics. In other words, FINGER claims to simplify the querying process while maintaining or boosting recall metrics (K. and Nicosia) while DABS makes a similar claim where recall is boosted with little to no difference in computational overhead (K. and Nicosia). However, abiding by the no free lunch theorem proposed by Wolpert and Macready, no algorithm is universally best for all possible problems. We can expand this theorem to include the previously mentioned HNSW structure and its respective querying variants. By testing the standard HNSW structure against its variants, we can determine the true extent of each variant's impact on the approximate nearest neighbor search. In this paper, we build an HNSW structure utilizing the hnswlib library and modify the querying methodology to default to FINGER or DABS and measure the impact these querying variants have on the recall and querying speed of our queries. The queries shall be extracted from mid to high dimensional GloVE twitter datasets consisting of words and their embeddings. We hypothesize that the FINGER structure will result in quicker querying time than both the standard HNSW structure and DABS but will fall short of both of them in terms of recall. Inversely, we hypothesize DABS to have the highest recall at the expense of the slowest querying speed. This is based on the reasoning that FINGER is skipping calculations that may reduce its approximation performance while DABS could venture and consider more data points during the searching process due to the high scale of our dataset. This pushes our hypothesis further to believe that DABS recall will be more resistant to higher dimensional datasets than HNSW and the FINGER variant.

2 Related Work

As previously stated in the introduction Malkov and Yashunin proposed the HNSW as an improvement from NSW structures by incorporating the concept of layers where each layer represents an NSW at a special level of abstraction. Higher layers are more sparse while lower layers are more dense–with the lowest consisting of all data points as nodes and edges connecting them. Even with this complex data structure, HNSW still relies on greedy heuristics for graph traversal and sometimes gets trapped in local optima similar to NSWs. This is ever more prominent in high-scale high dimensional datasets and therefore resulting in HNSW's performance degrading in high dimensional data distributions.

FINGER was researched and formally proposed by Chen et al. as an alternative to the greedy heuristic approach. This querying variant relies upon the geometric similarity of the dataset and calculates the distances between residuals in data. This increased speed is claimed to be accompanied with similar recall performance but we shall investigate this claim in this paper.

In parallel, Al-Jazzazi et al. developed and introduced the DABS variant that enhances searching quality by dynamically setting the beam width, which is the candidate pool of neighbors per nodes, over the course of the querying process. This allows for the DABS technique to adjust and consider an appropriate number of candidate nodes based on their distances with the query. This means both trimming far and unlikely neighbors in sparse areas of the graph and including many close neighbors in dense locations of the graph. The latter gives reason for us to believe that DABS may improve recall at the cost of computational speed.

3 HNSW Overview

In this portion of the paper we shall dive deeper into hierarchical navigable small world structures and emphasize on its construction, querying, and hyperparameters that influence performance.

3.1 Structure

Being layered navigable small worlds, HNSW structures consist of a base layer (level 0) identical to an NSW of all the nodes. This includes all data points as nodes and all edges connecting them to their neighbors. Upper layers (level > 0) are sparse reconstructions of the base layer with fewer connected nodes.

3.2 Construction

The construction of the HNSW involves analyzing all points of the data set and assigning a "max layer" $\ell = \left\lfloor -\log(U) \cdot \frac{1}{\log(M)} \right\rfloor$. Where U is a unform distribution $(0, 1)$ and M is the max number of neighbors per node per layer. This represents the highest layer a data point can be at. Most of these values end up being 0 thus limiting many data points to only exist in the base layer. Once all data points are placed at their respective layers, HNSW starts at the highest level and works downwards connecting all nodes at each layer with their neighbors. Each node's neighborhood can only have a maximum of M edges and approximating these

neighbors depends on the $ef_{construction}$ hyperparameter. This parameter controls how wide the beam is during searching for any data point's M neighbors during the construction phase.

3.3 Querying

Search queries begin at the topmost layer, the sparsest. These layers typically have one or two entry nodes so there is little distance heuristic that can be employed thus most queries simply drop to the second highest layer. At this layer, we search for ef_{search} number of candidate nodes and determine which set of those candidate nodes are closest to the query. The candidate nodes are stored in a priority queue indexed by distance to the query node. This priority queue is known as the beam and the width of the beam is static and controlled by the ef_{search} hyperparameter. At each step, the highest priority node is popped and its neighbors are traversed. If its neighbors are promising, they are added to the queue. The search concludes at the bottom layer when there are no closer candidates to explore and the top k nodes in the beam priority queue are returned as the k nearest neighbors. This querying method allows for logarithmic approximation of dense areas with relatively high computational speed (Aumüller et al.).

3.4 Hyperparmeters

We discussed three crucial hyperparameters that control the efficiency of this data structure. M controls the maximum number of neighbors any node can have per layer. Increasing this value implies a more connected graph which increases performance and accuracy at the cost of computational time. $Ef_{construction}$ is a hyperparameter used to control the beam width during construction of the HNSW structure. Increasing this parameter means neighbors are more likely to be the closest neighbors of any given node but this performance boost comes at the cost of computational speed. Similarly ef_{search} maintains the beam width during querying and increasing this parameter implies considering more neighbors during the searching process and thus boosting performance at the expense of querying speed.

4 FINGER Overview

To combat slow performance speeds at higher level datasets Chen et al. conducted an analysis and determined that many of the intermediate distance calculations provide little to no value on the final classification of query points. By removing these calculations and replacing them with additional saved neighbor information, he proposed the FINGER method.

4.1 Description

FINGER is designed to accelerate inference in ANN search by reducing the distance computations during graph traversal. Instead of relying on traditional exact distance calculations, FINGER opts to use approximate distances via residual vectors and angular estimates.

4.2 Algorithm

Given q is a query vector and x_i is a node in the graph, FINGER stores the residual vector $r_i = x_i - p$ where p is a reference point. From here, instead of calculating the exact euclidean distance, FINGER estimates with the law of cosines and inner products:

$$\|q - x_i\|^2 = \|q - p\|^2 + \|r_i\|^2 - 2\langle q - p, r_i \rangle$$

Where $\|q - x_i\|$ is the distance from the query to the reference point. $\|r_i\|$ is the magnitude of the residual vector and $\langle q - p, r_i \rangle$ is the inner product between the query residual and the stored residual vector. This effectively reduces the distance computations during the querying process but immensely slows down construction with all these additional pre-computings.

Each node stores the residual vectors from it to its neighbors allowing for efficient querying time by simply referencing those vectors. This acceleration in querying speed is greatly seen in high dimensional datasets.

5 DABS Overview

Since HNSW suffers at higher dimensional datasets, it is imperative to incorporate some flexibility in our model to customize itself with that dataset. In other words, having static hyperparameters may be efficient for some datasets and yet perform poorly on others. Al-Jazzazi et al. proposes a distance-ABS variant to improve the accuracy and robustness of the model over a multitude of high dimensional datasets.

5.1 Algorithm

DABS attempts to remove the static beam size that HNSW boasts with a dynamically altering beam size. Let q be the query vector and consider that the algorithm sets up a beam B containing nodes $x_1, x_2, ..., x_n$ with corresponding distances $d_1, d_2, ...d_n$ representing the distance between those nodes and the query vector. Now, the width of the beam w at each step is calculated as follows:

$$w = \min\left(W_{\max},\ |\{n_i \in B \mid d_i \leq d_{threshold}\}|\right)$$

This equation follows the bias that fewer candidates are chosen when the distances between candidates is large–indicating sparse distributions. On the other hand, more candidates are selected when distances between candidates is small–indicating dense distributions. $W_{\max}$ is the maximum beam size and $d_{threshold}$ is the dynamically computed distance cutoff and relates to the best observed distance of the distribution.

This algorithm also explores all neighbors of the nodes in the beam and evaluates and merges those candidates. This is the same behavior as in the HNSW structure. Once no further improvements in closest nodes can be made, the top k nodes are returned signaling the completion of the querying process.

6 Experiments

In this section of the paper, we shall migrate to the experimental focus on how we were able to address our hypothesis on how each variant compares with the default performance of

the HNSW structure. To do this, we will first introduce the datasets that were considered. After that, we will describe the process of determining the true labels and the benchmarking metrics that were used to compare the structures. Following this introductory material, we will present the results of our experiment and dive into the crux of analysis based on our study.

6.1 Dataset Overview

Since we wished to demonstrate the behavior of HNSW structures on high scale and high dimensional datasets, we focused on the global vectors for word representation (GloVe) twitter embeddings. GloVe is an unsupervised learning algorithm developed by Pennington et al. that consists of 1.2 million words and their embeddings pulled from over 2 billion tweets. These embeddings are trained to capture all sorts of figurative and acronym based language found on twitter. There are four twitter datasets dimensions of 25, 50, 100, and 200. The higher dimensional datasets represent more semantic information but, as can be expected, are more expensive to analyze. This paper utilizes all four of these large high dimensional datasets.

6.2 True Labels

A crucial segment of evaluating the performance with respect to recall of our structures depends on the true values of the dataset. Being a nearest-neighbor poised analysis, this meant determining the true nearest neighbors for all queries within the dataset.

We first initialized a list of 1000 query words. These words were randomly selected from the 25D dataset. Keep in mind that the words in all four of the 25D, 50D, 100D, and 200D are the same. Once these query words were extracted, we utilized a K dimensional (KD) tree to find the k nearest neighbors for all queries. Since we wish to measure the performance of our HNSW structures when tasked to find a variable number of neighbors, we considered k values in the range of [5, 50] with the increment of 5–meaning 5, 10, 15...50. KD trees were built and tasked to find the respective k neighbors for each of the 1000 queries and store that information in separate text files. This was repeated for all four dimensions of datasets. This process took well over 100 hours of continuous processing. However, once the neighbors were computed, they were stored in a text document and retrieved in relatively low time.

6.3 Benchmarking Metrics

With true nearest neighbors determined for all the queries, we had the luxury of utilizing recall as the main performance metric. After running any of the HNSW variants, we were presented a list of the k nearest neighbors. We compared this list with the precomputed true values list and determined how many of the true neighbors were included in our predicted neighbors list. This proportion was saved as the recall at k for the variant and graphed later.

Computational time was also critical in this research but simple wall clock time or CPU processing cycles is unreliable as a metric. This issue was further heightened by our ongoing true neighbor calculations that were being computed in parallel thus holding on to valuable

computing power. We used a simple comparisons metric that would increment whenever the data structure performed any distanced based calculation. Since all HNSW structures converged when there were no better neighbors, determining how many comparisons each variant made allowed for some discussion on which one converged quicker. While this metric can be associated to quicker completion time, it is flawed to claim a smaller comparison score guarantees quicker convergence as there is no consideration on each variant's computational complexity. Furthermore, if one variant's comparison score is double that of another, we cannot conclude the former took twice as long as the latter.

6.4 Experiment Results

In this section of the paper, we will present the results from our experimental analysis and analyze the results. We shall also tie the analysis back to the hypothesis in an attempt to deduce its validity. We will conclude the experiment portion by highlighting potential areas of improvement with respect to algorithmic and computational limitations.

6.4.1 Graphs

Below are graphs representing the performances of our models with respect to each of the GloVe datasets. Keep in mind that the DABS technique is written as "ABS":

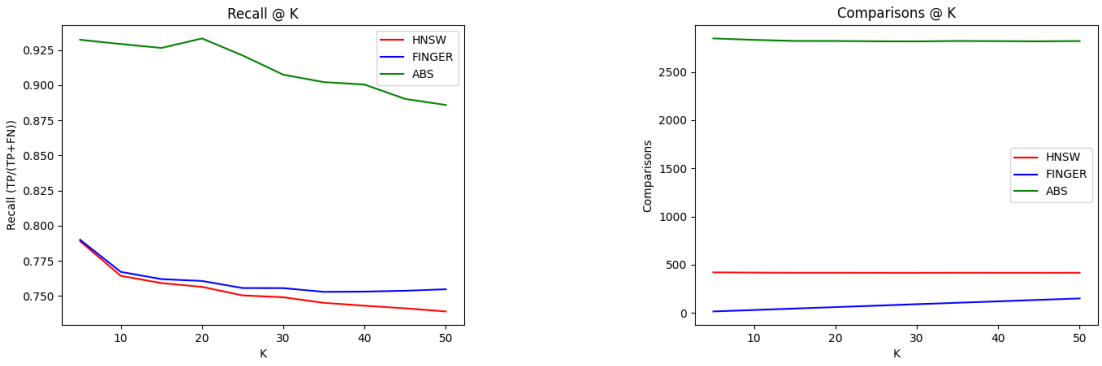

Figure 1: glove.twitter.27B.25d Dataset

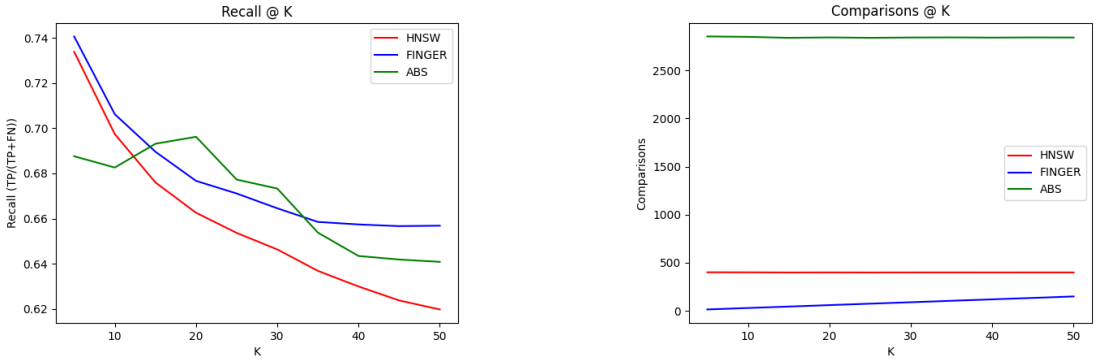

Figure 2: glove.twitter.27B.50d Dataset

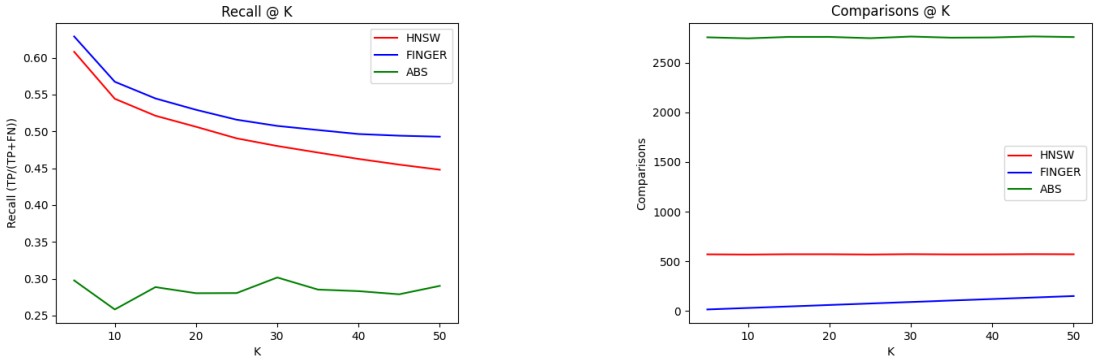

Figure 3: glove.twitter.27B.100d Dataset

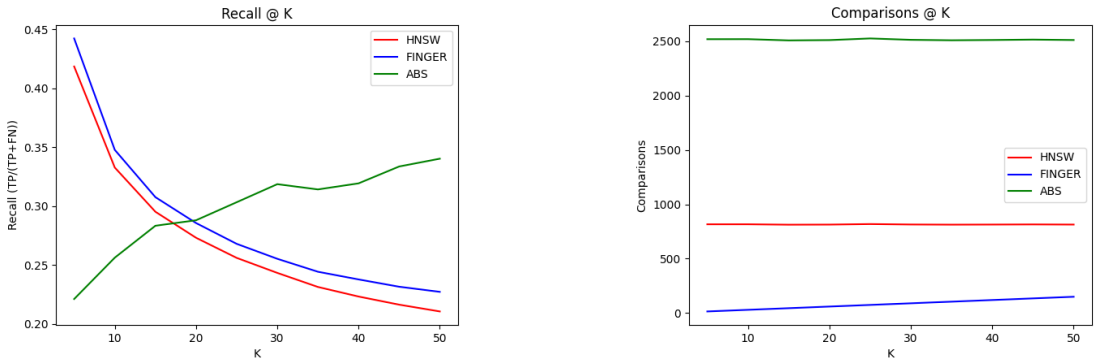

Figure 4: glove.twitter.27B.200d Dataset

6.4.2 Graph Analysis

Based on the comparisons @ K graphs for all of the datasets, it is apparent that DABS processes the largest number of distance calculations–exploring many more neighbors in dense

areas of the datasets. HNSW comparisons are much less and they follow an equally consistent line demonstrating that each layer is a well connected small world allowing for efficient traversal without many deviations in distance calculations. FINGER managed to conduct fewer distance calculations and still managed to match the recall rate of HNSW. FINGER, however observes a steeper slope than the other two metrics as K increases demonstrating more calculations and could hint at some internal deficiency in either the algorithm or our implementation. It is worth noting that all of these distance calculations seem to be almost exact across all datasets, further emphasizing the powerful small-world nature of each model. However, this performance came at the cost of expensive model build time–with the 200D FINGER HNSW taking well over 7 hours to build. The results from these comparisons graphs are in line with our original hypothesis that DABS would make the most distance calculations–due to the density of our dataset–followed by HNSW and finishing with FINGER–which traded model build speed and storage with querying speed. We also tracked the number of comparisons calculated by the KD-Tree algorithm when calculating the true neighbors and those values range as high as 1,193,467–effectively the size of the entire dataset–on the 200D dataset at 50 nearest neighbors. Although a tangent, this additional information places importance on exploring approximating nearest neighbor solutions over exact nearest neighbor solutions.

Unlike the comparison analysis, the recall analysis across all datasets paints a different picture of all models. At 25 dimensions, we see DABS performing extremely well at nearly 95% recall rate and HNSW and FINGER lagging between 72-80%. FINGER has a higher recall than HNSW at higher levels of k which is contradictory to what we had assumed in our original hypothesis. It seems that the extensive build time was the tradeoff for both less comparisons and more accurate results. However, as seen in the 50, 100, and 200 dimension datasets, all three model's performances decreased. The most notable being the DABS which loses its "highest recall" title midway through the 50 dimension dataset. FINGER and HNSW decrease as well but FINGER remains at a higher recall than HNSW and both observe visually similar trends across all values of k. One reason for the impressive downfall of DABS is that we tuned the technique on the 25D dataset–this accounts for its exceptionally high recall rate. When put into a higher dimensional space, data is now more compact and more similar. This stresses the maximum beam size and requires its own tuned set of hyperparameters. We chose not to go down this route as we were limited by computational resources. However, this trend seems to inverse at the 100D dataset. Recall values for DABS on this dataset follow a visually consistent trend–opposite to what is observed by the other two structures. We also see DABS recall rate increase as K increases on the 200D dataset. This is at complete odds with how our other models perform. Although the recall is still well under 40%, the trend indicates that higher dimensions at higher K values may allow DABS to regain its lost performance. The reason why DABS has higher recall at $k = 50$ rather than $k = 5$ could be due to the fact that DABS is able to approximate the nearest neighbors but not rank them in the correct order. This could hint at some superior performance of a dynamic beam width at higher dimensions at high levels of k. However, since we were limited by dataset size and computational resources to embed higher dimensional datasets, we were not able to test this theory. We can explain the major dip in DABS performance as a result of not properly tuning the model to each of the datasets

and, if given more time and greater computational resources, would investigate that route to determine how well the model's peak performance is on high dimensional data.

6.5 Potential Areas of improvement

As hinted in the previous discussion of recall, having a set of DABS hyperparameters that are customized for each dataset could boost the performance of this technique and allow us to better see how dynamically adjusting the beam search allows for better performance on higher dimensional data. However, since we are operating on relatively powerful hardware and still limited by computational resources, setting these parameters presents the question as to how viable a properly tuned DABS implementation would be given its expensive requirements. If given more time, we could also attempt to determine a more customized candidate pool sourcing method for FINGER. Currently, this algorithm sources a multiple of k nodes from its stored neighbors and determines the closest based on the residual calculations. However, this multiple is a static value and therefore explains the steady positive slope in FINGER comparisons. With more time, we could introduce a scaling function that multiplies k by a diminishing number. This value would be higher at smaller levels of k to capture the most candidates and be lower at larger values of k as we expect many of the candidates to already be the nearest neighbors.

7 Conclusion

After completion of the experiment and analysis on its results, we can revisit our original hypothesis and make statements regarding its validity. Our original hypothesis claimed that DABS would be the most computational expensive due to the density of our graph and thus would explore more nodes. FINGER was predicted to be the quickest due to its reduction in distance calculations and intense precomputations. Both of these predictions have been demonstrated and achieved. Our hypothesis predicted DABS, due to its heavily involved visitation of neighbors, would have the highest recall. This was partially correct as we see the DABS performing with the highest recall in the 25 dimensional dataset and regaining recall traction at higher levels of K on the 200D dataset but failing miserably in the 100D dataset and under-performing in the 50D dataset. As previously mentioned, we blame the lack of computational resources to properly tune this model for those higher dimensional datasets. We then predicted FINGER to perform at a lower recall rate than both DABS and HNSW due to it skipping the intermediate calculations. However, our data supports that not only was FINGER quicker but it also had higher recall than HNSW for all datasets. The intensive FINGER precomputations took more than 7 hours for the 200D dataset but seemed to allow both efficient speed and high recall rates for all levels of K.

The recall rates for all of the models seems to falter at the 100D dataset with HSNW and FINGER dropping below 50% at higher levels of k. Since our DABS dataset was poorly tuned, we cannot conclude whether DABS allows for the use of HNSW in higher dimensional spaces. There is some increase in performance at higher levels of K on the 200 dimensional dataset but there is no conclusive evidence proving that this increase continues in higher dimensions. Therefore, we do not have sufficient evidence to claim that HNSW is suitable for high scale high dimensional distributions with large values of k.

7.1 Further Research

Further research can be applied to determine if there are some DABS hyperparameter tunings that allow it to have higher levels of recall in high dimensional spaces but that is merely a follow up to the experiment presented in this paper. One of the main engineering issues faced in this paper were the limitations of the hnswlib python implementation. Being a C++ implemented library, there were a limited number of python bindings that were exposed. This prevented us from attempting a deeper dive into the graph structure and node edges of each layer. Access to these private structures opens a gateway into analyzing the topological properties of the HNSW graph layers. Another area of deficiency is that traversal within graph layers depends on previously defined graph traversal techniques such as greedy or heuristic search. These traversal techniques are tunnel visioned to find the closest node to the query point but fail to account for the HNSW structure as a whole. Research into developing some specialized graph traversal technique just for these multi-layered structured would not only benefit HNSW but could also aid in other layered structured such as hierarchical clustering (HC).

References

M. Al-Jazzazi, R. J. Campello, and M. E. Houle. Distance-adaptive beam search for provably accurate graph-based nearest neighbor search. *Machine Learning*, pages 1–28, 2022.

M. Aumüller, E. Bernhardsson, and A. Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 2020.

K. Bayer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? *International Conference on Database Theory*, 1999.

R. Bellman. Adaptive control processes: A guided tour. *Princeton University Press*, 1961.

K. Bezzina and G. Nicosia. Adaptive beam search for approximate nearest neighbor search. *Proceedings of the 2020 International Joint Converest on Neural Networks (IJCNN)*, 2020.

P. H. Chen, W.-C. Chang, J.-Y. Jiang, and H.-F. Yu. Finger: Fast inference for graph-based approximate nearest neighbor search. *Proceedings of the Association for Computing Machinery (ACM) Web Conference*, 2023.

K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang. Fast approximate nearest-neighbor search with k-nearest neighbor graph. *IJCNN*, 2011.

K. Bezzina K. and G. Nicosia. Finger: A fast and incremental graph-based nearest neighbor search algorithm. *Information Sciences*, pages 1231–1249, 2021.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, (7553):436–444, 2015.

Y. A. Malkov and D. A. Yashunin. Efficient and robust approximate nearest neghbor search using hierarchical navigable small world graphs. *Institute of Electrical and Electronics Engineers (IEEE) Transactions on Pattern Analysis and Machine Intelligence*, (4):824–836, 2018.

J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, (1):67–82, 1997.