Software Development for Information Systems

Project Report

Dimitris Dimochronis Nikos Pentasilis Giorgos Ragkos

January 12, 2025

Contents

1 Introduction

In this report, we provide our results from our implementation of the algorithms provided in the papers for DISK-ANN. The implementation is based on C++. Additionally, we make recommendations about optimizing and parallelize parts of the algorithms to fasten and improve their performance. Our suggestions aim to make algorithms suitable for less powerful machines and make them affordable for the masses.

2 Choice of Structures

Nodes are stored in structs with their index number in dataset, their data and their out-neighbors. Datasets are stored in vectors for quick random indexing and easier methods. Out-neighbors for each node are stored in set to avoid the check for duplicates in every insertion. Float is used as data type for storing data and euclidean distances, as it provides accuracy and less-time complexity in operations.

3 Optimizations

This chapter describes the optimizations implemented to improve the performance on indexing, time and query search. Their performance is analyzed in the next chapter.

3.1 Medoid calculation

3.1.1 Medoid calculation from subset of dataset

The medoid point is calculated on a subset of the initial dataset, so as to reduce the time needed.

3.1.2 Random medoid calculation

The medoid is defined randomly, so no calculation are needed for the medoid.

3.2 Connection between nodes with different filters

This section is about initialization of graph in Filtered & Stitched Vamana.

3.2.1 Random edges between nodes with same or different filters

The graph is initialized by adding to each node up to R random neighbors, no matter what their filter is. This approach, creates accessible paths from a filter to another so as to improve search for unfiltered queries. On filtered queries it is supposed to have no effect.

3.2.2 Search start nodes

Filtered Greedy Search for unfiltered queries gets as initial node for each filter the result of Filtered Greedy for k=1, searching for the query on the sub-graph with f_q.

This technique is very efficient as it ensures that the search starts from the closest point to the query in the certain sub-graph.

4 Parallelization

4.1 Parallelization in Filtered Vamana

As well as there are no edges between nodes with different filters, the Filtered Vamana can be reduced to the indexing on each sub-graph for every filter. So, the algorithm computes every sub-graph for each

filter and, finally, connect them all to the same graph. The computation of each graph can be parallelized so as to be computed much faster, as far as, there are not edges that can cause conflicts/critical sections.

4.2 Parallelization in Stitched Vamana

In Stitched Vamana, Vamana calls can be parallelized without conflicts and critical sections because there are no edges between nodes with different filters.

4.3 Parallelization in Query search

Parallelization in query search can reduce the total search time, as plenty of searches can be done simultaneously. This approach do not reduce average search time per query but minimized the total search time for massive searches.

5 Performance Analysis

All runs are made under these circumstances:
Dataset: DUMMY for filtered & stitched, ANN_SIFT10K for Vamana,
Queries: contest-queries-release-1m for filtered & stitched, ANN_SIFT10K for Vamana,
Parameters: k=100, L=100, R=15, a=1.2.

5.1 Optimizations

Results are the averages of 150 executions.

5.1.1 Medoid

Classic medoid calculation

Vamana:

- Total Recall@100 = 97.99%,
- Index creation time = 20.1s,

Stitched:

- Total Recall@100 = 94.8%,
- Unfiltered Recall@100 = 90.27%,
- Filtered Recall@100 = 99.38%,
- Index creation time = 6s,

Calculation from subset of dataset

Vamana:

- Total Recall@100 = 97.99%,
- Index creation time = 19s,

Stitched:

- Total Recall@100 = 94.81%,
- Unfiltered Recall@100 = 90.29%,
- Filtered Recall@100 = 99.37%,
- Index creation time = 6s

The recall rate remained the same (i.e., 97.99%), but the indexing time decreased slightly.

Random medoid

Vamana:

- Total Recall@100 = 98.05%,
- Index creation time = 19.1s

Stitched:

- Total Recall@100 = 94.81%,
- Unfiltered Recall@100 = 90.29%,
- Filtered Recall@100 = 99.37%,
- Index creation time = 6s

By using a random point as the medoid, we observed that the recall rate slightly increased, and the indexing time reduced again.

After taking into consideration the results above, we have decided to keep the random medoid choice.

5.1.2 Connection between nodes with different filters

Empty graph

Without any optimization or parallelization and with not-initialized graph, the results are:

For Filtered:

- Total Recall@100 = 60%
- Unfiltered Recall@100 = 21%,
- Filtered Recall@100 = 99%,
- Average search time = 1.03ms,
- Index creation time = 32s,

For Stitched:

- Total Recall@100 = 57.07%
- Unfiltered Recall@100 = 15.09%,
- Filtered Recall@100 = 99.38%,
- Average search time = 0.9912ms,
- Index creation time = 10.9s,

The filtered queries search bring out an excellent recall rate, as the search is targeted at the sub-graph corresponding to each query's filter.

In contrast, the unfiltered search has a very low recall rate, as there are no edges between nodes belonging to different filters, making it impossible to search across the entire graph.

The search time per query performs excellent.

To improve the recall rate, the sub-graphs corresponding to different filters need to be connected, enabling searches across the entire graph.

Random initialized

With random initialization in the graph between nodes with same or different filter, the results are:// For Filtered:

- Total Recall@100 = 83%,
- Unfiltered Recall@100 = 67%,
- Filtered Recall@100 = 99%,
- Average search time = 1.05ms,

- Index creation time = 34s,

For stitched:

- Total Recall@100 = 57.42%,
- Unfiltered Recall@100 = 15.8%,
- Filtered Recall@100 = 99.37%,
- Average search time = 0.9875ms,
- Index creation time = 10.6s,

The recall rate for the filtered queries remains the same, but there is a significant increase for the unfiltered search in Filtered Vamana. This is expected, as the random initialization of edges favors unfiltered queries by creating paths to the entire graph. The filtered recall is not affected by the new edges, as the search focuses only at points with same/common filter. However, the recall for Stitched Vamana remains very low.

The search time per query remains constant compared to the initial implementation. This is expected since the modification affects the graph creation process and not the search itself (R remains constant).

The graph creation time increases slightly, but this is negligible.

However, the accuracy of unfiltered queries can not be accepted.

Greedy search results initialization

For Filtered:

- Total Recall@100 = 95%
- Unfiltered Recall@100 = 91%,
- Filtered Recall@100 = 99%,
- Average search time = 9.5ms,
- Index creation time = 32s

For Stitched:

- Total Recall@100 = 94.81%
- Unfiltered Recall@100 = 90.27%,
- Filtered Recall@100 = 99.38%,
- Average search time = 8.6ms,
- Index creation time = 10.2s

This technique is very efficient as it ensures that the search starts from the closest point to the query, in the certain sub-graph.

An excellent recall rate is observed for the filtered queries, as expected, but also for the unfiltered queries.

The search time per query is almost ten times higher than the previous ones. This is the overhead of this specific optimization in which the trade-off between recall rate and search time is obvious.

This optimization offers a very significant increase in the recall rate but at a substantial cost to the search speed.

5.2 Parametrization

5.2.1 L

Performance for $L \in [100, 200]$ can be seen above. The recall increases in same way with L and so do average search time and indexing time. Both are expected, as the indexing and search enlarge the length of their searching lists, which provides bigger accuracy but adds overhead.

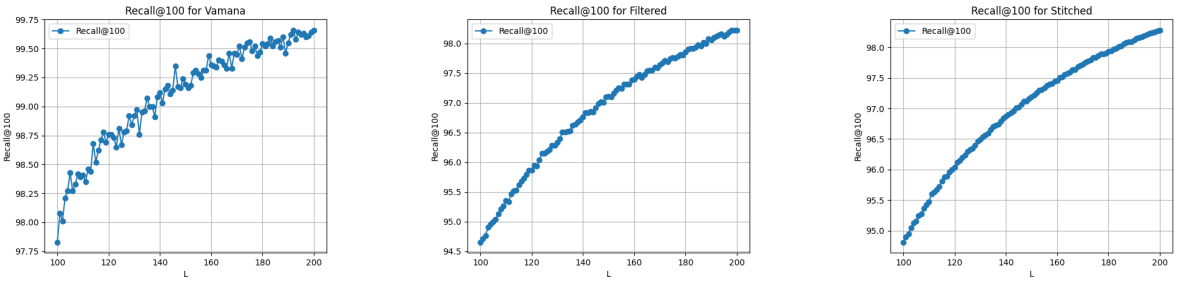

Figure 5.1: Recall for L $\in [100, 200]$

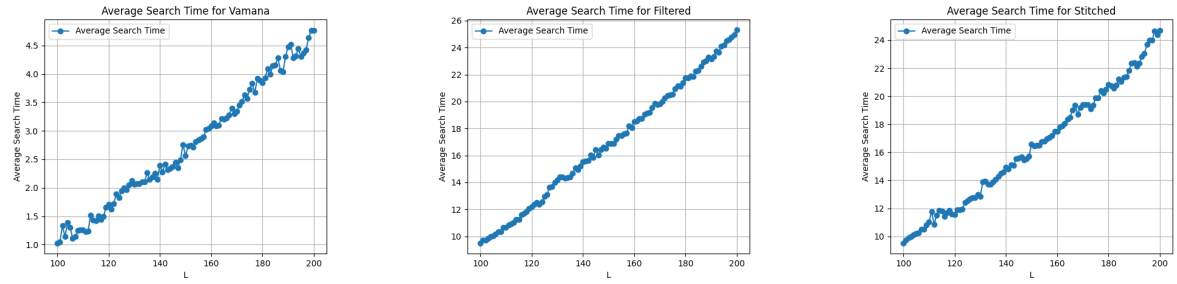

Figure 5.2: Average search time for L $\in [100, 200]$

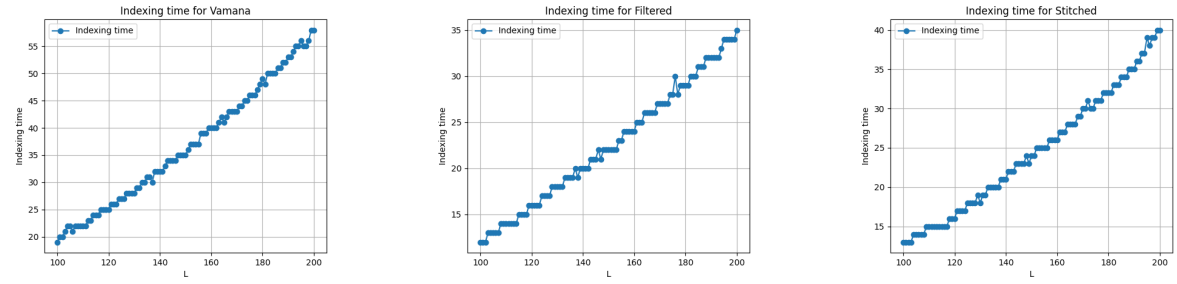

Figure 5.3: Indexing time for L $\in [100, 200]$

5.2.2 R

Performance for $R \in [15, 100]$ can be seen above. Recall improves by adding more edges and adds connection with more nodes. The graph size increases do as result of more edges to store.

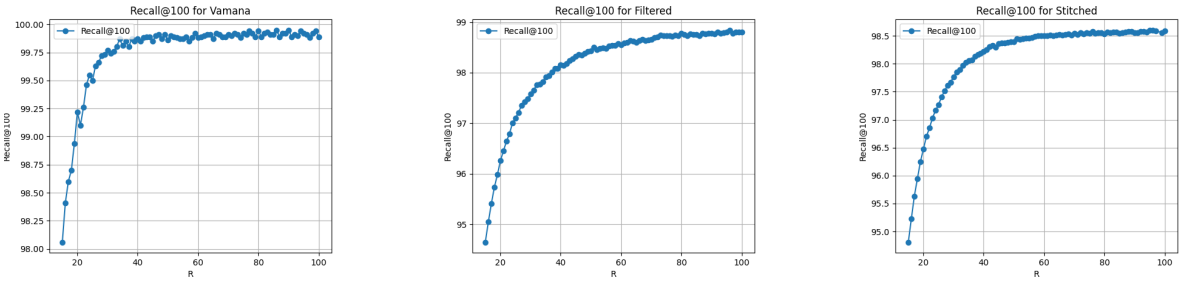

Figure 5.4: Recall for R $\in [15, 100]$

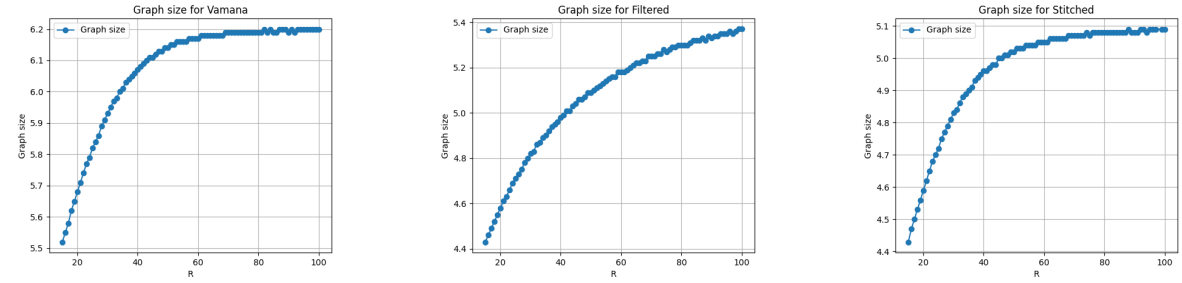

Figure 5.5: Graph size for R $\in [15, 100]$

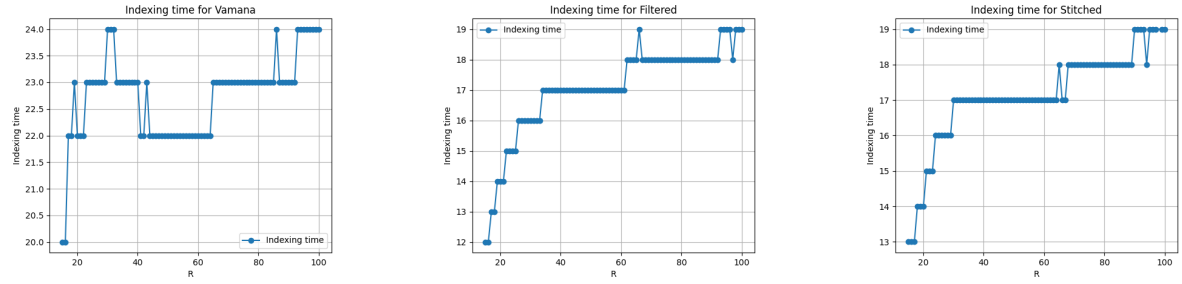

Figure 5.6: Indexing time for R $\in [15, 100]$

5.2.3 a

Performance for $a \in [1, 2]$ with step 0.01 can be seen above. We observe a curve in the recall, where it achieves the best performance for a = 1.2 with recall around 95%. Indexing time increases along with a, as it is more indulgent to cutting edges.

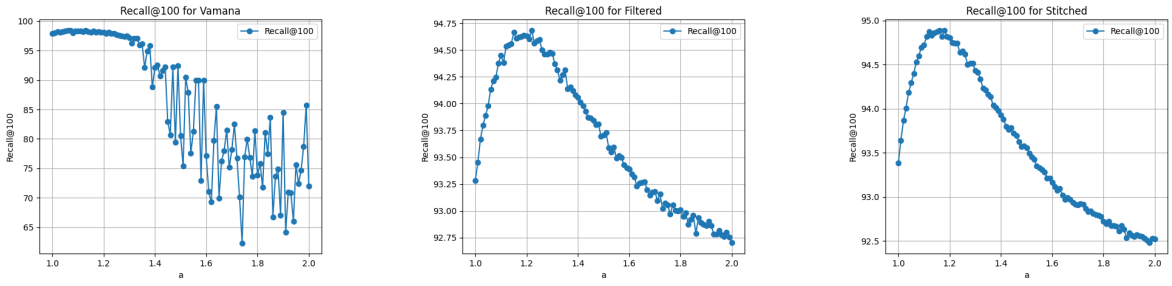

Figure 5.7: Recall for a $\in [1, 2]$

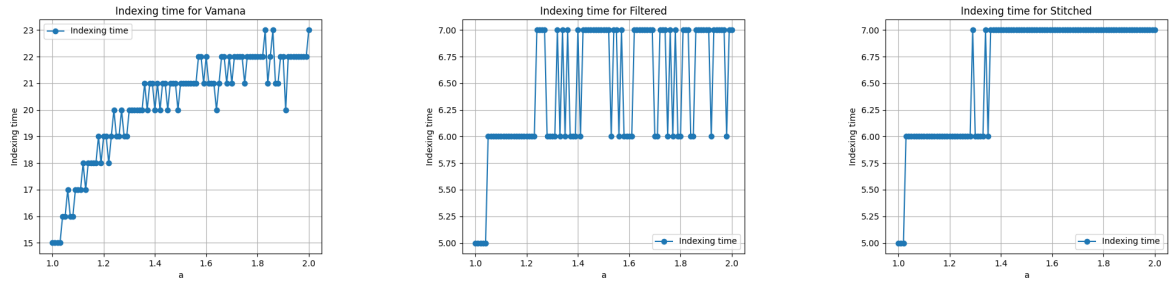

Figure 5.8: Indexing time for a $\in [1, 2]$

5.3 Parallelization

After running the algorithms for 100 execution for 1 and 16 threads, the mean results are:
Vamana 1 thread:

- Queries/s = 100,
- Average search time/query = 1ms,
- Index creation time = 17s

Vamana 16 threads:

- Queries/s = 100,
- Average search time/query = 1.01ms,
- Index creation time = 16s

Filtered 1 thread:

- Queries/s = 113,
- Average search time/query = 8.6ms,
- Index creation time = 11s

Filtered 16 threads:

- Queries/s = 556,

- Average search time/query = 28.1103ms,
- Index creation time = 6s

Stitched 1 thread:

- Queries/s = 113,
- Average search time/query = 8.56ms,
- Index creation time = 10s

Stitched 16 threads:

- Queries/s = 556,
- Average search time/query = 28.1145ms,
- Index creation time = 5s

5.4 Indexing on 1M datasets

The results of the algorithms for the 1M datasets with R=40, L=100, a=1.2 are:

- Vamana did not finish after 3 days.
- Filtered:
  - Total Recall@100 = 96.7049%,
  - Unfiltered Recall@100 = 94.9155%,
  - Filtered Recall@100 = 98.4801%,
  - Queries/s = 0.0001,
  - Average search time/query = 2s,
  - Index creation time = 12hr,
  - Index size = 517,5 MB
- Stitched:
  - Total Recall@100 = 96.7646%,
  - Unfiltered Recall@100 = 95.0056%,
  - Filtered Recall@100 = 98.5095%,
  - Queries/s = 0.0001,
  - Average search time/query = 2s,
  - Index creation time = 13hr,
  - Index size = 521,5 MB

5.5 Vamana on unfiltered queries

By running Vamana indexing on unfiltered queries of DUMMY dataset the results are:

- Total Recall@100 = Unfiltered Recall@100 = 93.7766%,
- Queries/s = 2516,
- Average search time/query = 3.69555s,
- Index creation time = 18s,
- Index size = 4.45MB

5.6 Best performance parameters

Big datasets are tested on WSL with 16GB RAM on the main system. The rest of the experiments are made on Mac with M2.
For 10k datasets searching for 100 ANN we recommend: R=20, L=110, a=1.2, index time = 7s and recall = 96% for Filtered and Stitched and R = 15 L = 100, index time = 16s and recall = 98% for Vamana.
For 1M dataset searching for 100 ANN for Filtered & Stitched, we recommend R=40, L=100, a=1.2.

6 Conclusions

This report presents the optimizations that implemented for Approximate Nearest Neighbor (ANN) search using the Vamana, Filtered Vamana, and Stitched Vamana algorithms. The optimizations aimed to improve the indexing time, the recall rates and the query performance through efficient graph initialization, medoid selection and parallelization techniques. The performance Analysis shows the changes to recall rate and search time from each optimization. Based on the experiments we establish that the combination of the random medoid selection, start nodes with results of Filtered Greedy and the parallelization presents the most efficient way to improve the performance of ANN search. Additionally, results from large datasets (1M) can provide that the Filtered and Stitched Vamana algorithms outperform the classic Vamana in both recall and indexing time for both filtered and unfiltered queries.

7 References

1. Suhas Jayaram Subramanya, Devvrit, Rohan Kadekodi, Ravishankar Krishaswamy, and Harsha Vardhan Simhadri. 2019. DiskANN: fast accurate billion-point nearest neighbor search on a single node. Proceedings of the 33rd International Conference on Neural Information Processing Systems. Curran Associates Inc., Red Hook, NY, USA, Article 1233, 13766–13776.
`https://dl.acm.org/doi/abs/10.5555/3454287.3455520`

2. Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, Amit Singh, and Harsha Vardhan Simhadri. 2023. Filtered-DiskANN: Graph Algorithms for Approximate Nearest Neighbor Search with Filters. In Proceedings of the ACM Web Conference 2023 (WWW '23). Association for Computing Machinery, New York, NY, USA, 3406–3416.
`https://doi.org/10.1145/3543507.3583552`