

Cost Estimation for In-Memory Graph Search in Vector Databases

CS848 Project Proposal

Baoming Chang Student ID: 20959534
University of Waterloo

1 Problem and Motivation

Vector search supports semantic retrieval by finding items whose numerical representations are close to a query vector. Graph indexes such as Hierarchical Navigable Small World (HNSW) [1] examine fewer vectors by following links between nearby nodes. Keeping the index in memory avoids disk access, but each query still makes irregular accesses to vectors and graph edges while tracking visited nodes and maintaining a candidate queue. Some reads are served from cache, while others access main memory, and indirection may introduce additional loads. These memory accesses can lead to different query latencies even when two implementations perform the same distance calculations.

I propose to develop a cost model for these main-memory access paths, using HNSW search as the experimental setting. The model will account for the memory accesses required by each search operation, including accesses to vectors, graph references, and search state. For example, changing data placement may improve locality but add work to translate node identifiers into storage locations. I will explore which physical properties explain cost differences between implementations of the same logical search and whether the estimates help choose an implementation for a workload. Evaluation will consider both prediction accuracy and the runtime of the selected implementation, since a more accurate estimate does not necessarily lead to a faster choice.

2 Proposed Approach

Using the existing C++ implementation, I will study costs that search operation counts alone do not capture, focusing on how data is organized and accessed in memory. Comparisons will preserve the logical search and its results, allowing runtime differences to be examined without changing retrieval quality or reducing search work. The findings will guide the choice of memory access properties to include in the cost model.

One potential direction is to examine data layout and the timing of memory accesses. Graph reordering can improve cache locality by placing frequently co-accessed nodes close together [2], while prefetching may reduce waiting for memory accesses. However, changes to data layout may introduce extra address lookups, while prefetching can waste work on data that is never used. Efficient conventional implementations and simple layout choices will provide baselines for assessing these tradeoffs.

Building on cost modeling for hierarchical memory [3], I plan to combine operation counts with information about how much data is accessed and how often it is reused. A layout that performs well when the active data fits in cache may become less effective as the working set grows. Cache sizes and a small set of calibration measurements will inform the model's estimates of memory access cost. I will initially use a compact regression model to estimate query time and cost differences between alternative implementations.

The model will use measurements from representative queries to select a suitable implementation for subsequent queries in the workload. Choosing a configuration at the workload level allows profiling and prediction costs to be amortized across those queries. The overall benefit will depend on whether the resulting query-time savings offset these costs and the cost of layout preparation, particularly for short workloads.

3 Evaluation and Feasibility

The main experiments will use an existing four-million-vector subset of the Turing benchmark with 100-dimensional vectors. If time permits, smaller SIFT and GloVe workloads will provide additional checks using image descriptors and word embeddings. These additional workloads will help assess whether the model remains useful beyond the main benchmark. The C++ implementation will run under Ubuntu in WSL on a fixed CPU workstation, using a single pinned search thread with both the graph and vectors held in memory. Comparisons will use the same graph topology, query set, and search settings, with separate queries for training, validation, and testing. I will verify that equivalent implementations preserve the returned results and logical search work, and report recall@10 for the common search configuration.

I will compare the proposed model with a constant baseline and models based on distance counts, operation counts, or simple locality measures. Evaluation will examine prediction error and the accuracy of estimated cost differences between implementations. Configuration choices will be made using training and validation queries, then evaluated on test queries. I will compare model-based selection with direct tuning and report the additional query latency relative to the best fixed implementation in the measured set. Repeated runs will measure mean, p95 and p99 query latency alongside memory use. All implementations will use a consistent warmup procedure, with testing order varied across runs to limit measurement bias. Overall gains will account for profiling, prediction, and data preparation costs.

The existing implementation and benchmark data make the project feasible within the course timeline. The study will focus on static indexes, with the scope and experimental priorities refined as the work progresses. The prototype, experiments, and final report will be completed by the December 8 deadline.

References

- [1] Y. A. Malkov and D. A. Yashunin. 2020. Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE TPAMI* 42, 4, 824–836.
- [2] B. Coleman, S. Segarra, A. J. Smola, and A. Shrivastava. 2022. Graph Reordering for Cache-Efficient Near Neighbor Search. In *Advances in Neural Information Processing Systems*, Vol. 35.
- [3] S. Manegold, P. Boncz, and M. L. Kersten. 2002. Generic Database Cost Models for Hierarchical Memory Systems. In *VLDB*, 191–202.