Optimizing Multi-Attribute Filtering in HNSW with Bitsets and Roaring Bitmaps

Amine Idrissi

December 28, 2024

Contents

1 Introduction

This report discusses a project undertaken as part of the Database Management course, under the supervision of Professor Karima Echihabi and Anas Ait Omar. The project focused on optimizing multi-attribute filtering in Approximate Nearest Neighbor (ANN) search using Hierarchical Navigable Small World (HNSW) indexing. The objective was to explore two approaches for filtering: bitsets and Roaring Bitmaps. Both methods were implemented to efficiently filter data based on multiple attributes and improve the performance of the filtering process in HNSW.

2 Background

The topic of HNSW and ANN search was new to me, so I spent some time understanding the fundamental concepts. I began by researching what Approximate Nearest Neighbor (ANN) search is, as well as how HNSW works. HNSW uses a graph structure with multiple layers, where each layer is connected in a skip list manner, creating a navigable small-world graph. This structure helps efficiently perform nearest neighbor searches by exploring multiple candidate nodes in each layer. I then investigated how filtering is implemented in HNSW. Specifically, I looked at how attributes of data points are checked to determine if they match the query, which is a critical step in reducing the number of candidates for nearest neighbor search. This process requires $O(p)$ time complexity, where p is the number of attributes.

3 Proposed Solution

The goal of this project is to optimize the filtering process in HNSW by using two methods: bitsets and Roaring Bitmaps. These methods allow us to quickly evaluate whether a data point meets the query's filtering criteria based on multiple attributes.

3.1 Bitset Approach

The bitset approach uses a 'std::bitset' to represent each attribute of a data point as a binary value. This allows for efficient bitwise operations to check if a data point matches the query criteria.

3.2 Roaring Bitmap Approach

Roaring Bitmaps, which are designed for compressed representation of sparse data, are used to store attributes. This allows for efficient set operations, particularly when the data is sparse. Roaring Bitmaps reduce memory usage compared to regular bitsets by storing only the bits that are set, making them more efficient when dealing with sparse data.

4 Implementation

The implementation is done in C++ using the HNSW library. The two filtering approaches (bitset and Roaring Bitmap) were implemented, allowing the filtering process to be evaluated based on different levels of sparsity. The results are saved in CSV files, which are then processed by a Python script to plot the results.

4.1 Bitset Implementation

In the bitset approach, each data point is represented as a bitset, with each bit corresponding to an attribute. The filtering process involves checking if a point's bitset matches the query's bitset using bitwise operations.

4.2 Roaring Bitmap Implementation

In the Roaring Bitmap approach, each data point is represented using a Roaring Bitmap. This allows for efficient filtering, particularly when the data is sparse, since Roaring Bitmaps only store set bits and compress the storage.

4.3 Benchmarking the Approaches

To evaluate the performance of the two filtering approaches, a benchmark was created using multiple probabilities in the Bernoulli distribution. These probabilities represent the sparsity of the data. The Bernoulli distribution was used to generate attributes for the data. For each probability, the following metrics were measured: search latency and memory usage.

The results were saved in CSV files containing the following fields:

- **Probability**: The probability used in the Bernoulli distribution, which represents the sparsity of the data.
- **Sparsity**: The sparsity of the data, indicating the proportion of non-zero attributes.
- **Search Latency**: The time taken to perform the nearest neighbor search.
- **Memory**: The memory usage of the filtering approach.

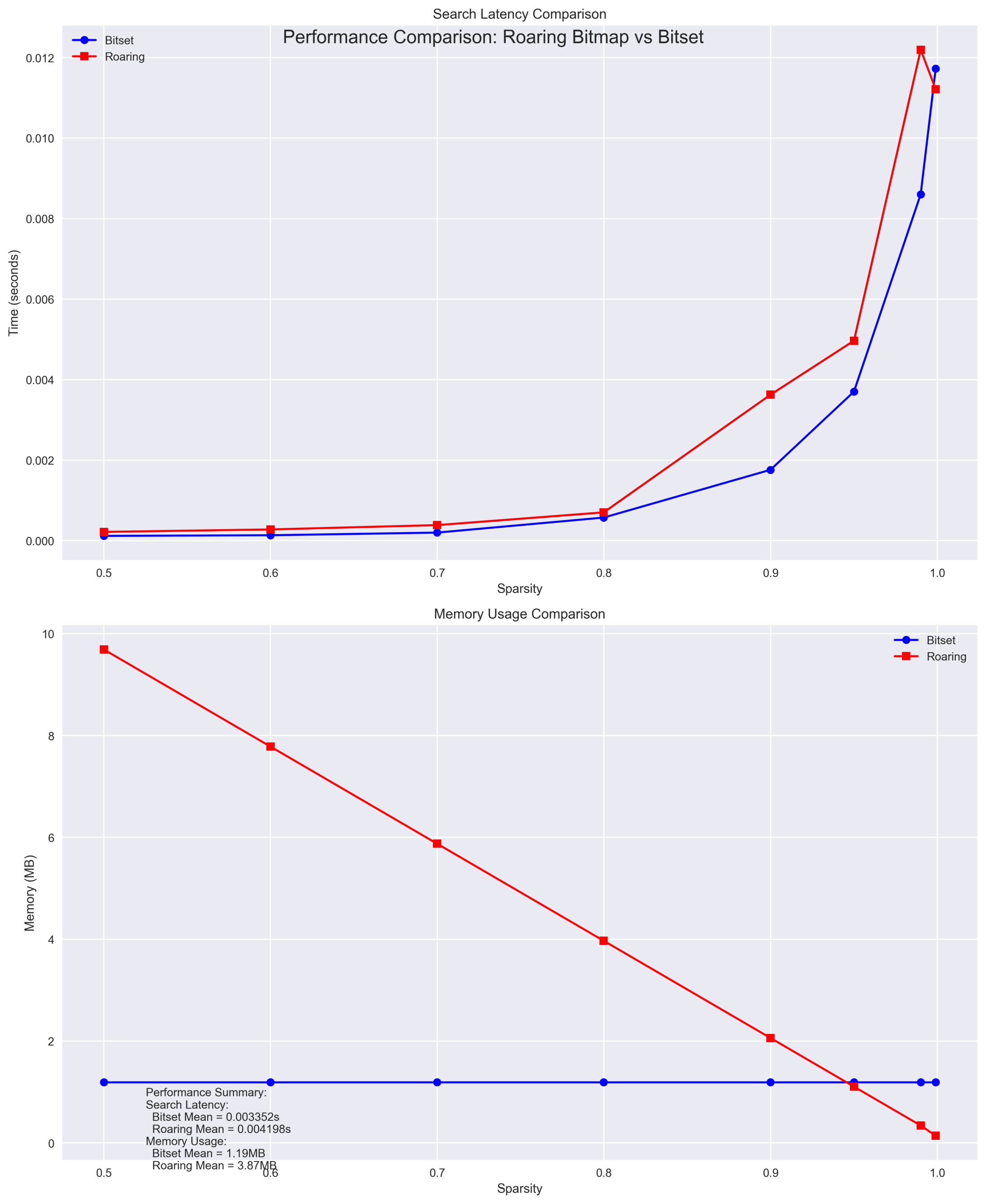
4.4 Results and Performance Comparison

The results presented in this section are based on 10,000 data points, each with 1,000 attributes. The performance of the bitset and Roaring Bitmap approaches is compared, and the following observations are made:

- **Latency**: Interestingly, as the sparsity increases (i.e., fewer attributes are set to 1), more time is needed for both approaches. Sparse queries tend to result in more frequent rejections during filtering because fewer points match the criteria. This forces the HNSW graph traversal to explore more nodes in order to find enough neighbors, which increases the total search time. In contrast, dense queries often result in many candidates passing the filter quickly, minimizing the need for additional traversal.

- **Memory Usage**: For bitsets, the memory usage remains constant because the size of the bitset is fixed and does not depend on sparsity. However, Roaring Bitmaps use less memory as the sparsity increases, due to their compressed storage. Therefore, Roaring Bitmaps are more efficient for sparse data.

Figure 1 illustrates the performance comparison between the two approaches.



```
Performance Summary:
Search Latency:
  Bitset Mean = 0.003352s
  Roaring Mean = 0.004198s
Memory Usage:
  Bitset Mean = 1.19MB
  Roaring Mean = 3.87MB
```

Figure 1: Performance comparison between the bitset and Roaring Bitmap approaches.

5 Challenges Faced

Several challenges were faced during the project:

- **Project Complexity**: The overall complexity of the project was a challenge at first, particularly with understanding the intricacies of HNSW and implementing the filtering techniques. However, with consistent effort and research, I was able to overcome these challenges and implement the solution successfully.
- **Technical Challenges**: Debugging and optimizing the implementation of HNSW in C++ required significant effort. I resolved this by using debugging tools and reviewing documentation to fix memory management and compiler issues.
- **Time Management**: Balancing the project work with other academic responsibilities was a challenge. I managed this by creating a detailed schedule and prioritizing tasks to ensure steady progress.

6 Conclusion

This project provided valuable insights into optimizing multi-attribute filtering in HNSW using bitsets and Roaring Bitmaps. The benchmark results showed that sparsity significantly affects the performance of both filtering methods, with latency increasing for sparse queries and memory usage benefiting from the compression of Roaring Bitmaps in sparse data. Moreover, AI tools were helpful in some stages of the project, particularly in understanding the key concepts of the HNSW algorithm. They also assisted in code refinement and documentation, improving both the structure of the code and the clarity of the documentation. Finally, I am grateful to Professor Karima and Anas Ait Omar for the opportunity to work on this project, which will contribute to my career development.