Hilbert Forest in the SISAP 2025 Indexing Challenge

Yasunobu Imamura[1][0009−0005−7472−5754], Takeshi Shinohara[2][0000−0002−7451−7374], Naoya Higuchi[3][0009−0004−2377−1681], Kouichi Hirata[2][0000−0003−0814−8395], and Tetsuji Kuboyama[4][0000−0003−1590−0231]

[1] THIRD INC., Shinjuku, Tokyo, Japan
[2] Kyushu Institute of Technology, Kawazu 680-4, Iizuka, Japan
`shino.kyutech@gmail.com, hirata@ai.kyutech.ac.jp`
[3] Sojo University, Ikeda 4-22-1, Nishi-ku, Kumamoto, Japan
`nac24nh@gmail.com`
[4] Gakushuin University, Mejiro 1-5-1, Toshima, Tokyo, Japan
`ori-sisap2025@tk.cc.gakushuin.ac.jp`

Abstract. We report our participation in the SISAP 2025 Indexing Challenge using a novel indexing technique called the Hilbert forest. The method is based on the fast Hilbert sort algorithm, which efficiently orders high-dimensional points along a Hilbert space-filling curve, and constructs multiple Hilbert trees to support approximate nearest neighbor search. We submitted implementations to both Task 1 (approximate search on the PUBMED23 dataset) and Task 2 (k-nearest neighbor graph construction on the GOOAQ dataset) under the official resource constraints of 16 GB RAM and 8 CPU cores. The Hilbert forest demonstrated competitive performance in Task 1 and achieved the fastest construction time in Task 2 while satisfying the required recall levels. These results highlight the practical effectiveness of Hilbert order–based indexing under strict memory limitations.

Keywords: Approximate Nearest Neighbor Search · Hilbert Sort · Hilbert Tree · Hilbert Forest

1 Introduction

The SISAP Indexing Challenge 2025 [4] provided a common platform for evaluating indexing techniques under strict computational resource constraints. This paper reports our participation in the challenge using the *Hilbert forest*, a recently developed indexing method for approximate k-nearest neighbor (ANN) search. The Hilbert forest is based on the fast Hilbert sort, which orders high-dimensional points along a Hilbert space-filling curve and builds multiple Hilbert trees to support efficient approximate search.

The source code of the Hilbert forest, as used in the challenge, is publicly available [1], and the official challenge results are accessible on GitHub [3]. We briefly describe the core ideas behind the Hilbert forest, summarize its application to the challenge tasks, and report the achieved performance. While a more detailed technical analysis will be presented in future work, this paper serves as a record of our participation and the outcomes of the challenge.

2 Hilbert Sort and the Hilbert Forest

Hilbert sort refers to reordering a set of points in high-dimensional space in the order along a Hilbert space-filling curve (hereafter, Hilbert order). Using a previously developed algorithm [2], we can efficiently sort n points in average $O(n \log n)$ time, independent of dimensionality. Since the Hilbert order defines a total ordering, binary search applies to the sorted sequence. Utilizing the idea of the fast Hilbert sort algorithm, we can locate the point in a set of n points that is closest to a given point p in Hilbert order in average $O(\log n)$ time.

A *Hilbert tree* is a binary search tree built with a point sequence ordered by Hilbert sort. It can be constructed in average $O(n \log n)$ time.

Points that are close in Hilbert order tend to be close in the original space as well, although the converse is not always true. However, due to the super-exponential increase in the number of possible Hilbert curves in higher dimensions, any pair of spatially close points is likely to be adjacent in at least one such Hilbert order.

A *Hilbert forest* is an index structure composed of multiple Hilbert trees, enabling approximate nearest neighbor search. Using more trees improves recall but increases both memory usage and computation time.

3 Task 1: Approximate Neighbor Search on PUBMED23

Task 1 deals with the PUBMED23 dataset, consisting of 23 million vectors in 384 dimensions. The goal is to minimize the query throughput time for 10,000 queries under the constraint of RAM 16GB and 8 CPU cores, while keeping recall@30 > 0.7.

In this section, we describe the index structure, search method, and preliminary experimental results obtained in the development environment.

3.1 Index Structure and Search Method

The index employs a two-stage filtering approach: coarse candidate selection using a Hilbert Forest, followed by further fine filtering with sketches. Final candidate selection is performed based on vector distance.

A full binary search tree constructed on the entire dataset would exceed 400MB per Hilbert tree. To stay within the 16GB RAM limit and leave room for sketches and quantized vectors used in later stages, we compressed each tree to about 76MB by replacing subtrees containing about 100 points with leaf nodes. The number of trees in the forest is at most 160.

To accelerate the filtering step, we avoid processing each query independently. Instead, we simulate the Hilbert sort for all queries collectively and compute their positions in Hilbert order. One Hilbert order is selected as the master order, and all vectors and sketches are rearranged in memory accordingly. This improves memory access locality in the second filtering stage. Additionally, we optionally include several neighbors around the selected candidates in the master order to improve recall.

For the second-stage filtering, we used 384-bit sketches, striking a balance between discriminative power and memory usage. The total size of the sketches is approximately 1.1 GB (23M $\times$ 384 bits). Hamming distance was used as the selection criterion.

The original vectors are represented by 32-bit float values, which amount to approximately 36 GB, and cannot be fully loaded into RAM. Therefore, we quantized the vectors to 4-bit integers. To reduce memory usage further, one bit is shared with the sketch, compressing the combined memory footprint of sketches and quantized vectors to about 4.5 GB. For accuracy, queries are not quantized during the final distance computation.

Key Hyperparameters

- Number of trees in the Hilbert Forest: n
- Number of candidates extracted per query from each tree: k_1
- Number of candidates selected per query by sketches: k_2
- Number of additional neighbors from the master order per candidate: h

The total number of candidates selected by the forest (including duplicates) for each query is $n \times k_1$. The number of candidates examined in the second-stage filtering is approximately $k_2 \times (2h + 1)$ per query.

We summarize the entire procedure of our approximate k-NN search method using the Hilbert forest in Algorithm 1.

3.2 Experimental Results

The experiments were conducted on a development machine with a Ryzen 9 3950X CPU and 128GB RAM. The actual runs were performed in a container limited to 8 CPU cores and 16GB RAM using WSL2 on Windows 11.

Index construction times were as follows:

- Hilbert Forest (160 trees): 38m56s
- Sketch generation: 5s
- Vector quantization: 2m32s
- Master sorting: 14s

Total preprocessing time: 2594 seconds (approximately 43 minutes)

Results for 16 hyperparameter combinations are summarized in Table 1.

Algorithm 1: Approximate k-NN Search Using Hilbert Forest

```
Input: Queries $q[0], \dots, q[Q-1]$
Output: Result sets $S[0], \dots, q[Q-1]$
// Preprocessing steps
Perform vector quantization of the dataset;
Generate binary sketches for all vectors;
Create a master order using Hilbert sorting and adjust memory layout for
  vectors and sketches;
Build $n$ Hilbert trees with randomized axis orders;
// Initialize candidate sets for each query
for $i \leftarrow 0$ to $Q-1$ do
    $C_1[i] \leftarrow \emptyset$;
// Use Hilbert forest to collect candidates
for $t \leftarrow 0$ to $ntrees - 1$ do
    simulate Hilbert sort of $q[0]$ to $q[Q-1]$ using $tree[t]$;
    for $i \leftarrow 0$ to $Q-1$ do
        extract $k_1$ candidates near $q[i]$'s position and add to $C_1[i]$;
// Filter candidates using Hamming distance of sketches
for $i \leftarrow 0$ to $Q-1$ do
    $C_2[i] \leftarrow$ select top-$k_2$ candidates from $C_1[i]$ using Hamming distance;
// Optionally expand and select final results using full vector distance
for $i \leftarrow 0$ to $Q-1$ do
    expand $C_2[i]$ using the master order (e.g., include $h$ neighbors on each side
      of each candidate);
    $S[i] \leftarrow$ select top-$k$ candidates from $C_2[i]$ using vector distance;
```

4 Task 2: k-Nearest Neighbor Graph Construction on GOOAQ

Task 2 targets the GOOAQ dataset, which consists of 3 million vectors of 384 dimensions. The goal is to construct a k-nearest neighbor (k-NN) graph for all data points under the constraint of 16GB RAM and 8 CPU cores, while achieving recall@15 > 0.8. The primary evaluation criterion is the total construction time.

4.1 Hilbert Forest-Based Graph Construction

Since every data point is also a query in this task, a Hilbert-ordered sequence can be used to select neighboring candidates based on proximity in the sorted order. Therefore, unlike in Task 1, the binary search tree component of each Hilbert tree is not needed. Once candidates are extracted using a Hilbert order, the sorted sequence can be discarded. Even when using multiple Hilbert orders, memory consumption remains constant, with only the computation time increasing.

Table 1. Recall and Search Time for Task1

n	k_1	k_2	h	recall (%)	time (sec)
160	1420	370	2	72.9	54.9
160	1420	360	2	72.8	18.0
160	1300	350	2	71.9	17.9
160	1300	340	2	71.8	16.1
160	1200	330	2	71.0	14.9
160	1200	320	2	70.9	17.3
160	1100	310	2	70.0	13.6
160	1100	300	2	69.9	15.7
120	4000	1000	2	79.1	29.4
120	3200	1000	2	77.2	25.8
120	2800	1000	2	76.0	23.6
120	2400	1000	2	74.5	23.8
120	2000	1000	2	72.6	19.4
120	1800	1000	2	71.5	18.3
120	1600	1000	2	70.3	19.6
120	1600	800	2	70.0	15.6

Key Hyperparameters

- Number of Hilbert sorts used: n
- Number of candidates extracted per point from each sort: k_1
- Number of candidates selected per point by sketches: k_2

We give the outline of the graph construction in Algorithm 2.

4.2 Experimental Results

The Table 2 below summarizes performance under different numbers of Hilbert sorts:

A minimum recall of 80% can be achieved in 74 seconds. Moreover, a graph with 95% recall can be constructed in just 5 minutes and 30 seconds.

5 Conclusion

For Task 1, due to memory limitations, it was difficult to increase the number of trees in the Hilbert forest, and thus achieving very fast search performance was not possible. However, it has been confirmed that under conditions with sufficient memory, significantly faster searches can be realized. In contrast, for Task 2, since it is not necessary to keep all Hilbert forest trees resident in memory, and a single Hilbert order sequence can be used to select candidates from the entire dataset, very fast processing was achieved.

Algorithm 2: Approximate k-NN Graph Construction for Task 2

```
// Preprocessing steps
Vector quantization;
Generate binary sketches for all vectors;
Input: Dataset of size $N$
Output: Approximate $k$-NN graph $S$.
// Initialize candidate sets
for $i \leftarrow 0$ to $N-1$ do
    $C_1[i] \leftarrow \emptyset$
// First-stage filtering (using Hilbert sorts)
for $t \leftarrow 0$ to $n-1$ do
    randomly permute coordinate axes;
    compute Hilbert order of all points;
    for $i \leftarrow 0$ to $N-1$ do
        $j \leftarrow$ index of the $i$-th point in the Hilbert order;
        extract $k_1$ neighbors around position $i$ in the order;
        append their indices to $C_1[j]$;
// Second-stage filtering (using sketches)
for $i \leftarrow 0$ to $N-1$ do
    $C_2[i] \leftarrow$ select top-$k_2$ candidates from $C_1[i]$ using Hamming distance;
// Final selection (using vector distances)
for $i \leftarrow 0$ to $N-1$ do
    $S[i] \leftarrow$ select top-15 from $C_2[i]$ using vector distance;
```

The approximate nearest neighbor search method using the proposed Hilbert forest, although still in the early stages of development, shows promise in providing an effective indexing structure and search strategy for high-dimensional data. In particular, the combination of multiple Hilbert orders leveraging the properties of space-filling curves is meaningful in achieving a balance between recall and speed.

Going forward, we aim to further develop this method into a practical high-dimensional indexing technique by applying it to more diverse datasets and constrained environments, optimizing parameters at each stage, and conducting comparative evaluations with other filtering methods.

Table 2. Construction Performance for Task2

Time (sec)	Recall (%)	n	k_1	k_2
74	80.5	80	96	60
109	85.5	112	106	75
164	90.5	160	130	100
330	95.5	280	168	150
856	98.5	720	170	300

References

1. Imamura, Y.: Implementation of hilbert forest for sisap'25 indexing challenge task1 & task2. https://github.com/colun/hforest, accessed: 2025-08-23
2. Imamura, Y., Shinohara, T., Hirata, K., Kuboyama, T.: Fast hilbert sort algorithm without using hilbert indices. In: Proc. SISAP 2016. pp. 259–267. LNCS 9939, Springer (2016)
3. SISAP Challenge Organizers: Sisap indexing challenge 2025 results. https://github.com/sisap-challenges/challenge2025/tree/results, accessed: 2025-08-23
4. Tellez, E.S., Chavez, E., Aumüller, M., Mic, V.: Overview of the sisap 2025 indexing challenge. In: Proc. SISAP 2025. LNCS 16134, Springer (2025)