Double Filtering Using Short and Long Quantized Projections

— Memory-Efficient Online k-NN Search for Large-Scale Datasets —

(beyond PUBMED23 in the Indexing Challenge)

N. Higuchi[1], Y. Imamura[2], T. Shinohara[3] K. Hirata[3] T. Kuboyama[4]

1. Sojo Univ. 2. Third Inc. 3. Kyushu Inst. Tech. 4. Gakushuin Univ.

Key Features of Our Method

1. Double Filtering
   - Two types of index: short and long projections
   - 1st stage by fast filtering by enumeration
2. Asymmetric distance between query and projection
   - Better than Hamming distance
3. Memory-efficient for large datasets

Results for Recall@30 = 70%

					Filtering Cost (ms/q)		Latency (ms/q)		
w	dim	qbit	k_{1st}	k_{2nd}	1^{st}	2^{nd}	avg	std	$avg^{\dagger}$
18	192	1	180,000	600	0.03	0.49	0.64	0.09	1.05
20	192	1	150,000	400	0.06	0.48	0.66	0.10	1.12

very small

stable

w: sketch width
dim, qbit: QSMAP dimension and quantization
k_{1st}, k_{2nd}: #candidates of 1^{st} and 2^{nd} filtering
1^{st}, 2^{nd}: filtering cost (ms/q)
avg, std: latency (ms/q), $avg^{\dagger}$: reported by SISAP

Experimental Environment: Docker container with 16GB RAM and 8 CPU, running on Windows 11 WSL, Ryzen 9 3950X CPU, 64 GB RAM

Filtering by Sketch Enumeration

```
// q: query, k': number of candidates
// PriorityOrder(q): sketch enumerator
// σ: sketch transformation
// σ⁻¹(ς): { x ∈ DS | σ(x) = ς }, the set of points whose sketch is ς
FilteringBySketchEnumeration(q, k')
        C := ∅;
        for ς in PriorityOrder(q)
                C := C ∪ σ⁻¹(ς);
                if |C| ≥ k' break;
        return C;
```

k-NN Search by Double Filtering

Two Types of Index: Short and Long
Short = Binary Sketches
Long = Quantized Projections of S-Map (QSMAP)

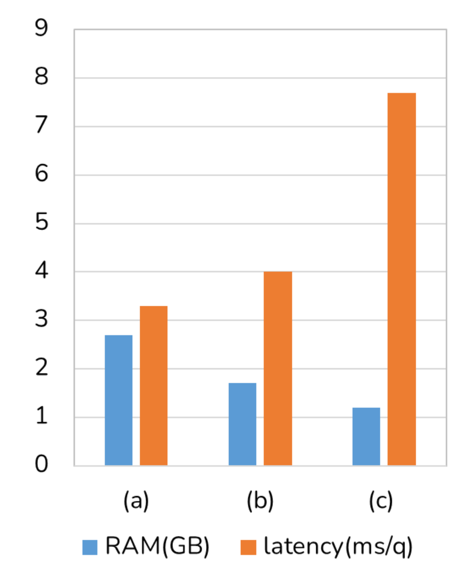
Double Filtering

Full Dataset ($n > k_{1st} > k_{2nd} > k$)
[Coarse Filtering by Sketch Enumeration]
> k_{1st} Candidates
[Fine Filtering by QSMAP]
> k_{2nd} Candidates
[Reranking by Exact Distance]
> kNN Answer

Results on Smaller RAM

Index (Bucket + QSMAP) is very small
Dataset on SSD
→ Double-Filtering can run on smaller RAM < 3 GB
Sketch width = 20-bit, Recall@30 = 70%

	dim	qbit	k_{1st}	k_{2nd}	latency	reranking	RAM(GB)
	384	2	300,000	30	4.0	1.5	2.7
(a)	384	2	160,000	40	3.3	1.8	2.7
(b)	384	1	260,000	60	4.0	2.7	1.7
(c)	192	1	500,000	150	7.7	6.1	1.2



Why Fast ?

- Average bucket size $|\sigma^{-1}(\varsigma)|$ is greater than 1
  - Typically ≥ 10 (e.g., ~23 for PUBMED23 with 20-bit sketches)
- To collect top-k' candidates:
  - Expected sketches to enumerate ≈ k' / (average bucket size)
  - Much smaller than the total number of sketches
- Result:
  - Only a small fraction of sketches are needed
  - Explains why the 1st-stage filtering cost is negligible

Index Construction & Total Memory Layout

Index Construction

Sketch
Width(w): 18-bit or 20-bit
(A) Bucket Table (keyed by sketches)
QSMAP
192 Dimensions
1-bit Quantization
(B) Array Sorted in Sketch Order
Memory Requirement
(A): $2^w \times$ int = 1 MB or 4 MB
(B): 192 × 1 × 23 M (bits) = 552 MB

Total Memory Layout

Index Part
= (A) + (B) < 1 GB

Dataset
Float (32-bit) Vectors = 30 GB
→ Char (8-bit) Vectors = 8.8 GB
(with negligible effect on accuracy)

Our Implementation
Index + Dataset < 9.8 GB < 16 GB

Fast Reranking on RAM !

Feature Directions

1. Optimal pivot selection for quantized projections
2. Application to other large-scale datasets (e.g., DEEP1B)
3. Similarity search in non-Euclidean or coordinate-free spaces

Filtering by Narrow Sketch Enumeration

- Problem: Full data scan is costly (n = 23M)
- Enumeration method (example for recall@30 = 70%):
  - Required candidates k' = 150,000
    → Expected sketches to enumerate ≈ 150,000 / 23 ≈ 6,522
  - Candidates ≈ 0.65% of full dataset → much smaller than n
- Even with narrow sketches:
  - Enumeration efficiently retrieves a small set of candidates