Approximate Nearest Neighbor Search

Weijie Zhao

11/07/2023

HW 4: Approximate Nearest Neighbor Search

- Find one sufficiently similar base vector for each query
- The first line of the input file contains 4 numbers: D N M T
  - D is the number of dimensions
  - N is the number of base vectors
  - M is the number of queries
  - T is the target similarity for each query.
- The following N lines are base vectors. Then M lines of queries.
- For each query, output one index (zero-based) for base vector.
- For at least 50% queries, the similarity should be at least T.

HW 4: Approximate Nearest Neighbor Search

- No 3rd party code is allowed.
- 10 test cases. Each case weights 1 pt.
- The compilation is considered failed if it does not finish in 5 minute.
- A test case is considered incorrect if it does not finish in 3 minutes.
- Correct GPU solutions will get 5 pts bonus.
- The summation of the execution time across 10 cases will be used to rank correct solutions.

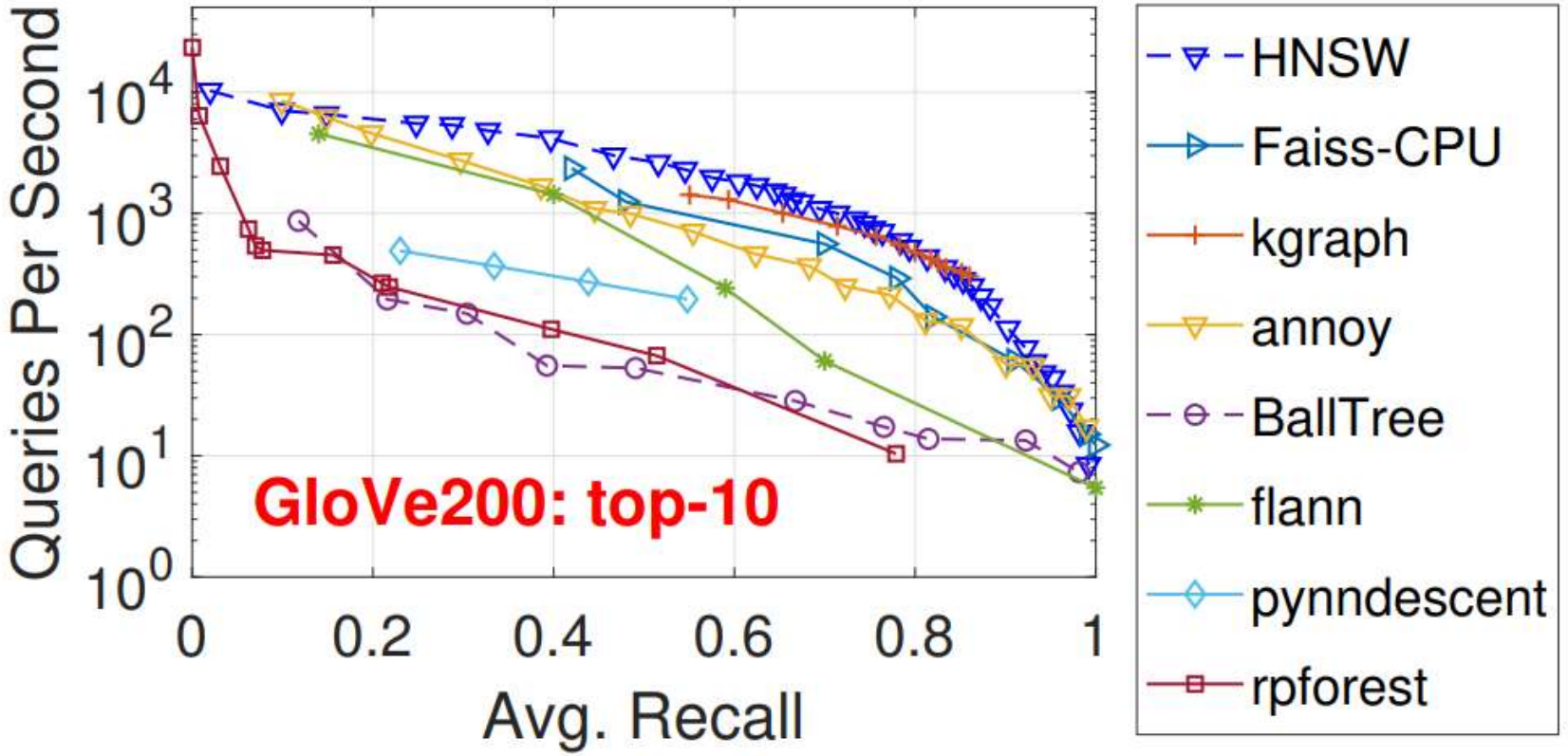
Testing Environment

- ssh yourusername@granger.cs.rit.edu
- Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
- 48 threads in total (2 sockets, 12 cores per socket, 2 threads per core)
- 251 GB memory
- GPU: Tesla P4

- Testing limit:
  - 8 threads taskset -c
  - 2 GPU

Approximate Nearest Neighbor Search

- Tree
- Hashing
- Quantization
- Graph

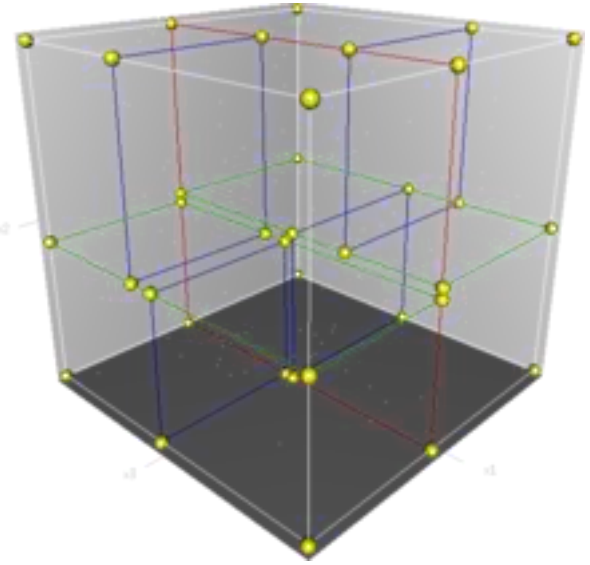


K-D Tree

```
function kdtree (list of points pointList, int depth)
{
    // Select axis based on depth so that axis cycles through all valid values
    var int axis := depth mod k;

    // Sort point list and choose median as pivot element
    select median by axis from pointList;

    // Create node and construct subtree
    node.location := median;
    node.leftChild := kdtree(points in pointList before median, depth+1);
    node.rightChild := kdtree(points in pointList after median, depth+1);
    return node;
}
```



https://en.wikipedia.org/wiki/K-d_tree

Locality-Sensitive Hashing (LSH)

- Random Projection
- MinHash
- Consistent Weighted Sampling

Algorithm 1 Generalized consistent weighted sampling (GCWS) for hashing the pGMM kernel.

Input: Data vector u_i ($i = 1$ to D)

Generate vector $\tilde{u}$ in $2D$-dim by (1).

For i from 1 to $2D$

$r_i \sim Gamma(2,1)$, $c_i \sim Gamma(2,1)$, $\beta_i \sim Uniform(0,1)$

$t_i \leftarrow \lfloor p\frac{\log \tilde{u}_i}{r_i} + \beta_i \rfloor$, $a_i \leftarrow \log(c_i) - r_i(t_i + 1 - \beta_i)$

End For

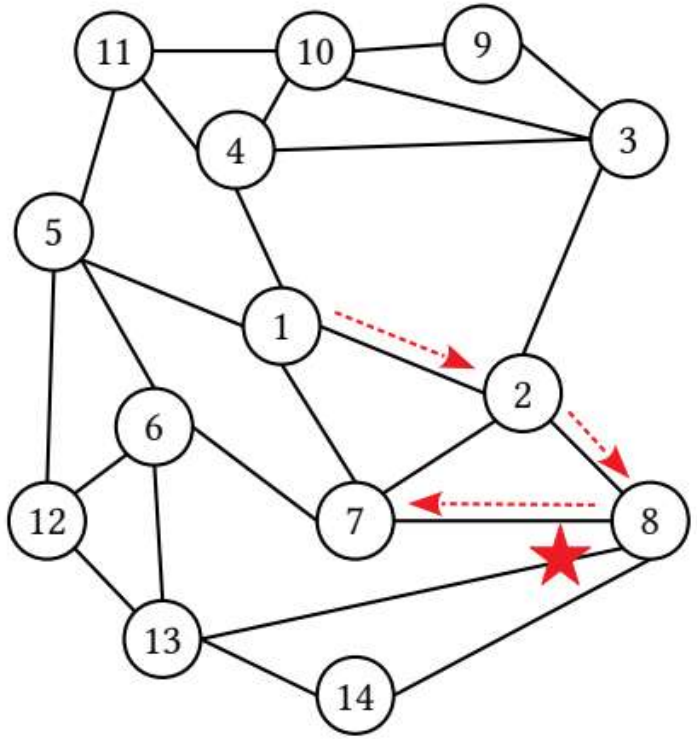
Output: $i^* \leftarrow arg\min_i \ a_i, \qquad t^* \leftarrow t_{i^*}$

Quantization

- Map high dimensional vector to low dimensional integer
- K means
- Product Quantization (PQ)

Graph-Based ANN

- Hierarchical Small World Graph (HNSW)
- Search ON Graph (SONG)
- BipartitE Graph Indices (BEGIN)



Initialization	*q:*	1
	topk:	∅
	visited:	1
Iteration 1	*q:*	2 7 4 5
	topk:	1
	visited:	1 2 4 5 7
Iteration 2	*q:*	8 7 3 4 5
	topk:	1 2
	visited:	1 2 3 4 5 7 8
Iteration 3	*q:*	7 3 4 5 14 13
	topk:	1 2 8
	visited:	1 2 3 4 5 7 8 13 14
Iteration 4	*q:*	3 4 5 6 14 13
	topk:	7 2 8
	visited:	1 2 3 4 5 6 7 8 13 14