Course Project: NN Search on k-NN Graph and the Diversified k-NN Graph

Lecturer: Dr. Wan-Lei Zhao

December 25, 2024

1 Problems

- You are required to implement NN-Descent search algorithm that is largely described in [1]. In addition, in order to boost the performance of NN-Descent search, you are required to implement the graph diversification [2]. You are already provided with most of the codes for the project. You only need to digest the provided codes and fill the missing parts. Specifically, you are required to fulfill the following steps.

  1. Fill the missing part in "nnsearch.hpp" and "graphdiverse.hpp";
  2. Perform nearest neighbor search by calling member function "nnSearch()" in class **NNSearch**
  3. Evaluate the search performance on two datasets. You should conduct the NN search on the provided k-NN graphs and the diversified k-NN graphs. You can get the diversified k-NN graph after you apply graph diversification on the provided k-NN graph.
  4. Display the performance figure like Fig. 1 for each dataset ang give comments on the performance.

In the provided, the part that needs to be completed by you has been hinted by comments as follows. You are NOT suggested to modify the other codes unless you fully understand how they work.

```
/**
filling your code here
 **/
```

2 Checklists of Available Materials

1. C++ codes

   - Load data, found in class "**IOManager**" in file "iomanager.hpp"
   - Call NNSearch and produce QPS (queries per second) evaluation, found in function "searchRecall()" in file "dosearch.cpp"
   - Nearest neigbhor search algorithm, found in class "**NNSearch**" in "nnsearch.hpp"
   - Graph diversification code, found in class "**GraphDiverse**" in "graphdiverse.hpp"

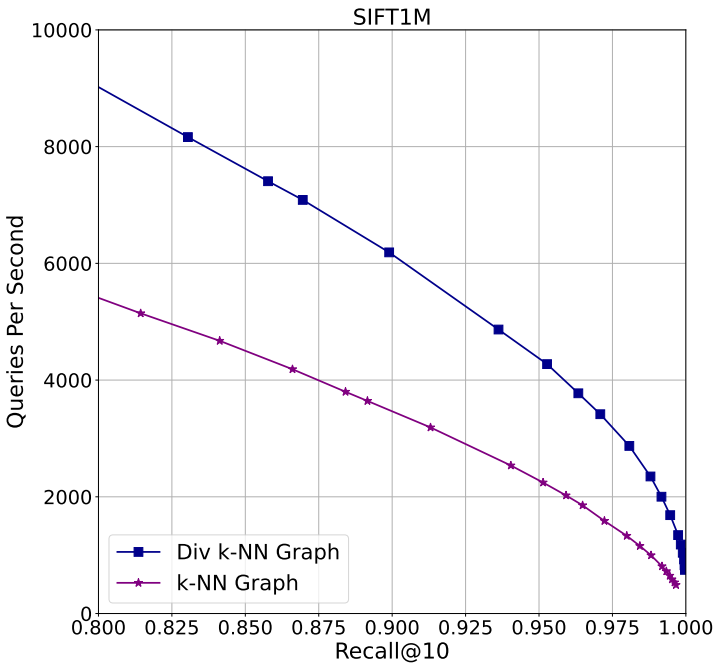

Figure 1: An illustration of search performance (QPS) on SIFT1M.

- Distance functions, found in class "**Metrics**" in file "metrics.hpp"
- A cmake file that is used to build the project "CMakeLists.txt"

2. Datasets

   - Two million-level datasets, including vectors file (".fvecs"), query files (".fvecs"), k-NN graph files (".ivecs") and the search ground-truth (".ivecs")
   - The links to download the datasets can be found from the attached text file "links2data.txt"

3 Suggestions on the project

You should fill the missing part in member function nnSearch() in class **NNSearch**. Once it is done, you can try to run the NN search by calling "searchRecall" in file "doseach.cpp". You should be able to produce the performance curve in purple color in Fig. 1.

You can proceed to filling the missing part in member function "triagDiverse()" in file "graphdiverse.hpp". The basic idea is to remove some of the neighbors from each neighborhood list. The scheme of triangle graph diversification is illustrated in Fig. 2. As shown in the figure, for each NN list, we check the triangle relationship with respect to the host node A. The checking starts from the second nearest neighbor of A[1]. If it is occluded by any closer neighbors to A, it should be removed. The diversification rule can be formulated as

$$\begin{cases} d(A, x_1) < d(A, x_2) \\ d(x_1, x_2) < d(A, x_1) \end{cases} . \tag{1}$$

Any neighbor x_2 is viewed as being occluded by x_1 when it satisfies the above rule, and should be removed from the list of k-nearest neighbors.

Moreover, when this diversification is done for each k-NN list, we should append the reverse neighbors of each node. Before we join the reverse neighbors in, they must also be under the same diversification by the above rule.

[1]The first neigbhor is not occluded by any other node, so it should be kept all the time

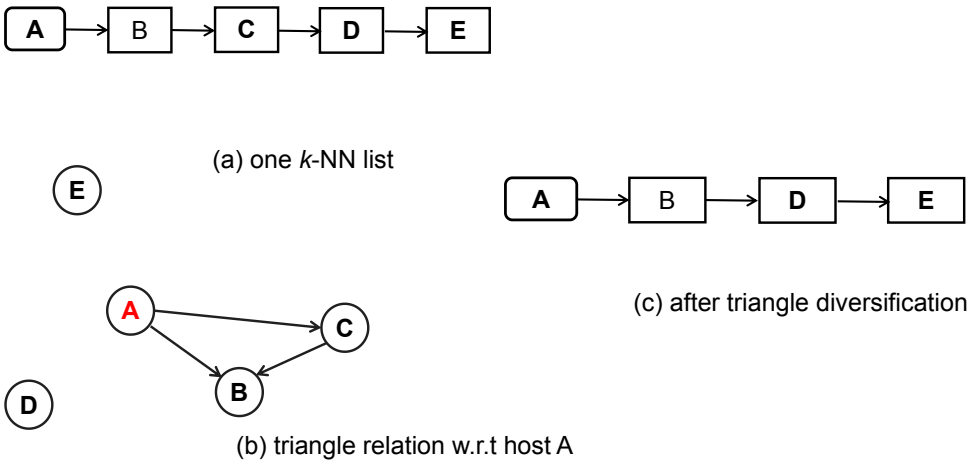

Figure 2: The idea of triangle graph diversification. Notice that neighbors **D** and **E** are not occluded by any other neighbors in the neighborhood. While neighbor **C** is occluded by neighbor **B**.

It is suggested to fulfill this project in two steps. In the first step, you complete the nnSearch() in class **NNSearch**. Then try to run the NN search with the provided k-NN graphs. You can call the compiled project from the commandline via following command given your data have been placed into directory "/home/user/datasets/".

```
./nns -q /home/user/datasets/sift1m_query.fvecs -i /home/user/datasets
   /sift1m_dynnd_k64.ivecs -gt /home/user/datasets/sift1m_gt.ivecs -c
   /home/user/datasets/sift1m_base.fvecs
```

Listing 1: commandline to run the NN search

In the second step, you can try to complete the triagDiverse() function in class **GraphDiverse**. Given you have completed it, you should uncomment the following codes in main() function in file "dosearch.cpp" to test it.

```
    callGraphDiverse();
    return 0;
```

Listing 2: code to call graph diversification

The graph diversification produces a diversified graph, and typically named as "xx_dynnd_div_k64.ivecs" You can modify this in function test() in the class. You can now switch back to run the NN search code. The commandline looks as follows given the diversificed graph is saved under directory "/home/user/datasets/".

```
./nns -q /home/user/datasets/sift1m_query.fvecs -i /home/user/datasets
   /sift1m_dynnd_div_k64.ivecs -gt /home/user/datasets/sift1m_gt.ivecs
    -c /home/user/datasets/sift1m_base.fvecs
```

Listing 3: commandline to run the NN search with the diversified graph

4 Requirements for your Submission

You are required to submit all your source codes (".hpp" and ".h") as well as a "Makefile" as a zipped package (e.g., studnumb.zip). Before your submission, please make sure they can be compiled soundly under Linux. It is strongly recommended that you work with GCC C++

compiler, either you are with Windows or Linux. The g++ version is assumed to be **5.4** or higher.

Along with the source code, you are required to submit a report written in English. Basically, there should be four sections. In the first section, the general issue that you are going to address should be reviewed. In the second section, please elaborate how you implement the NN-Descent and graph diversification. In the section followed, please present the performance of your approach on these two datasets, namely SIFT1M and DEEP1M. You are already provided with the codes to evaluate the search performance. Please run the search code in single thread. You should finally plot out figure with Python as Fig. 1. Finally, please conclude this project with your insights on this issue and this algorithm. The total marks for this project is 30 points. Twenty points are set aside for the implementation. Another 10 points are set for the report. Any forms of plagiarism are PROHIBITED.

1. Submission due: 2024/Jan./25, 24:00pm
2. Submission should include the source codes that can be compiled smoothly under Ubuntu, and a PDF report

References

[1] K. Hajebi, Y. Abbasi-Yadkori, H. Shahbazi, and H. Zhang, "Fast approximate nearest-neighbor search with k-nearest neighbor graph," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence*, 2011, pp. 1312–1317.

[2] W.-L. Zhao, H. Wang, and C.-W. Ngo, "Approximate k-NN graph construction: A generic online approach," *IEEE Transactions on Multimedia*, vol. 24, pp. 1909–1921, 2022.