ReadMe of the Code of iDistance

Beta version 1

Rui Zhang

rui@csse.unimelb.edu.au

http://www.csse.unimelb.edu.au/∼rui

Department of Computer Science
and Software Engineering,
The University of Melbourne,
Carlton, Victoria, Australia, 3053
2006

1 Paper related to this code

Rui Zhang, Panos Kalnis, Beng Chin Ooi, Kian-Lee Tan: **Generalized Multi-dimensional Data Mapping and Query Processing**, *ACM Transactions on Data Base Systems (TODS)*, 30(2), 364-397, 2005.

2 Copyright

3 Implementation Notes

Cui Yu has contributed to an older version of the code when she was in National University of Singapore.

4 How to Use the Code

4.1 Compilation

The code was compiled on Fedora 2 and can be compiled on most Linux and Unix systems. Use the command "make" to compile.

4.2 Follow the following steps to run the code

Please follow the steps below to run the code:

1. Suppose the dimensionality is 30, then change file "btree.h": #define D 30;
2. Suppose the number of reference points is 64, then change file "btree.h": #define ClusterNumber 64;

3. Data file and query file are both binary files which consist of float points sequentially. Change the variable "fp_data" in file "knn.c" to point to the data file and "fp_query" to point to the query file. You can search "define data file here" and "define query file here" to locate where they are defined.

4. Reference points are stored as a text file called "reference" in the directory. In file "knn.c", the variable "fp_reference" points to this file of reference points. How to choose reference points? If the dataset is uniformly distributed, reference points are also uniform. If the dataset is skewed or clustered, first find cluster centers, these centers are used as reference points. We have used 64 as the default number of reference points. However, larger numbers could be even better. Note that different choice of reference points may result in different performance. k-means clustering is suggested to find the cluster centers.

5. The value "K" in KNN (that is, the number of nearest neighbors requested) is set in file "knn.c". For example, #define K 10. By default K is 10.

6. The number of queries is set in file "knn.c". For example, #define NOQUERY 200. By default NOQUERY is 200.

7. After setting all the above, run "make" to compile the code. The executable is named "Idist".

8. Build the index by running "Idist b".

9. When building index is completed, some information like the following appears:

   ---

   idistance>nn b Root of b+ tree is created Splitting at root level ********** Note: Offset of new root : 0offset of new root2109440 The tree is set up. Root: 2109440 Left most leaf node: 8192 Max_top of Stack : 4

   btreeinfo done

   ---

10. Run the KNN search by "Idist", the results are written in the text file "result". The average page access number and query processing time are shown at the end of the file.

5 Optimization Notes

1. Internal nodes of the B^+-tree are load into memory before the queries. The internal nodes are less than 0.3% of the whole index, which can fit into memory. The time for loading is counted as running time. To test the effect of buffer, the code needs to be touched up.

2. Nodes of the B^+-tree can be compacted by the function compact( ) to a higher node utilization rate, which is set by the constant "compactrate" in file "btree.h". E.g. "#define compactrate 0.9" means the utilization rate is 0.9. Another way to achieve high utilization is by using a B*-tree style insert/delete, which is not implemented in this B^+-tree yet. If compactrate is set less than 0, no compaction would be conducted.

3. For KNN search, the initial search radius and the increasing amount of the search radius are set by two parameters "r0" and "dr" in the file "btree.h". Now r0 is set as $0.5 \cdot Mindist$, where Mindist is the distance between the query and the nearest reference point. And dr=0.005. r0 may be optimized by estimating the final search radius and dr can be set accordingly, such as 5% of r0.