测试文件调用说明

(gtree, road, silc)

2016 年 10 月 26 日

目录

总体说明

文件夹包含 gtree, road, silc 三种算法的 knn 测试与索引树生成代码。
1.Gtree 为 gtree 对应的 knn 代码。
2.gtree_new_p2p 是对 gtree 两点路算法返回路线的修正。
3.road 来自于算法作者，并做了一定的测试完善。
4.Silc 为根据文章算法生成的测试代码。

一、Gtree（gtree 文件夹）

1.1 编译说明

构建 gtree 索引结构：

g++ -std=c++0x -O2 gtree_build.cpp -L/usr/local/lib/ -lmetis -o gtree_build

构建 knn query 测试

g++ -std=c++0x -O2 gtree_query.cpp -L/usr/local/lib/ -lmetis -o gtree_query

1.2 索引树构建说明

构建树输入：**$./**gtree_build graph file(.cnode, .cedge)

输入文件为纯文本格式，输入文件为纯文本格式，分为两个文件：**edge,node**。Edge 表示边文件，Node 表示节点文件。

edge 文件每行 4 列，edge 编号　起　止　weight

node 文件每行 3 列，node 编号　经纬度

举例：

$ head col.cedge

0 0 1 0.012406

1 2 3 0.001082

2 2 4 0.003068

3 5 6 0.000487

4 6 7 0.000449

5 8 9 0.000350

6 9 10 0.000603

举例：

$ head col.cnode

0 -103.992898 39.971728

1 -103.978333 39.971957

2 -103.890907 40.001074

3 -103.889636 40.001080

4 -103.891479 39.998348

5 -105.044258 39.820320

6 -105.043720 39.820174

输出：

GTree index(.gtree)

GTree branch paths(.gpath)

GTree distance matrix(.mind)

1.3 gtree knn 调用

gtree_query　graph file(.cnode, .cedge)

GTree index(.gtree)

GTree branch paths(.gpath)

GTree distance matrix(.mind)

快速调用方法：

make gtree_query

./gtree_query

1.4 注意事项与依赖库：

1.图的划分需要 metis 库，具体下载与 guide 网址：
http://glaros.dtc.umn.edu/gkhome/metis/metis/overview
2.图的结构与起止编号，需参考示例文件进行。
3.图要求是连通图，不连通图无法利用 metis 划分，会报错。

二、Gtree 两点路修正版（gtree_new_p2p 文件夹）

2.1 输入文件说明

输入文件为纯文本格式，第一行两个整数 n,m 表示点数和边数，接下来 m 行每行三个整数 U,V,C 表示 U->V 有一条长度为 C 的边。路网文件可以为有向图，也可为无向图。具体设置参数在后面详述。

```
举例：
1089933 2545844   //表示路网图有 1089933 个点，2545844 条边。
0 1 655           // 0-> 1 路网距离为 655， 共 m 行
0 80 1393
0 81 1426
1 0 655
```

2.2 索引树生成与两点路查询函数

与 GTree 相同，G^+Tree需要对路网文件进行处理，并建立索引结构，以便后面查询使用。在本程序文件中，对单独的树结构构建时采取每次重新构建的方法。实际应用中可以将 save()和 load()函数分开，就不需要每次重新建立索引树。

2.2.1 路网图的G^+Tree索引构建

G^+Tree建立过程，主要使用了如下函数：

- void init() //初始化
  其中，int Additional_Memory //表示全连接矩阵规模，设置值越大两点路速度越快，按理论复杂度不超限，设置为$2 \times V \times \log_2 V$，如果内存大，可以调大该参数。
- void read() //读取 const char Edge_File[] 路网图边权值文件
- void tree.build() //建立 gptree

2.2.2 G^+Tree的保存于装载

G^+Tree可以不需要每次都单独构建树结构索引，使用的时候，先用 save（）生成索引结构，可以在之后的最短路查询中直接 load（）。

- void save() //将 gptree 存储如文件"GP_Tree.data"
- void load() //从"GP_Tree.data"文件读取数据，使用时可替代上面的 tree.build()

2.2.3 G^+Tree的图上两点最短路查询

根据不同情况设计了两点最短路不同调用方法：

- `int search(int S, int T) //`查询路网点 S、T 之间的最短路，返回值为距离
- `int search_catch(int S,int T) //`带 cache 的两点最短路，速度慢于 search，频繁查询速度会提高
- `int find_path(int S,int T,vector<int> &order)//`返回 S-T 最短路长度，并将沿途经过的结点存储到 order 数组中

2.3 相关参数说明

为了保证性能及适应不同路网结构，制定了较灵活的参数，大部分在源文件中有说明，以下是比较重要的参数。

- `const bool Optimization_G_tree_Search` 是否使用全连接矩阵。使用全连接矩阵能够提高两点路查询速度，但建树时间会增加，该矩阵大小为 `Additional_Memory` 个 `int`。
- `Edge_File` 为边权文件，`Node_File` 为节点位置文件。（`Node_File` 为欧几里得剪裁等提供支持，但在两点路中不需要此文件）
- `Partition_Part` 表示 gptree 分叉数，对应论文中 fallout
- `Naive_Split_Limit` 表示叶子节点最大规模+1，对应论文中 τ+1
- `RevE` `false` 代表有向图，`true` 代表无向图读入边复制反向一条边。目前测试均使用有向图。

2.4 关于路网结构的补充

使用时，不同路网起始点编号不同（如有的是 0，有的是 1），如对应路网图 NW.edge 时，起点为 0，应该将 2383 行：

```
if(RevE==false)G.add_D(j-1,k-1,l);//单向边
```

改为

```
if(RevE==false)G.add_D(j,k,l);//单向边
```

三、ROAD（Road 文件夹）

3.1 编译说明

直接调用 make 文件

3.2 索引树构建说明

索引树生成方法：
./hiergraphloader -n BJ_washed.cnode -e BJ_washed.cedge -t 4 -l 8 -h BJ_washed_roadtree_t4_l8.idx
t 和 l 是 ROAD 原文的参数，分别表示分支和层数
注：如果生成索引文件过短，需保证边的起点编号大于终点编号

输入文件为纯文本格式，输入文件为纯文本格式，分为两个文件：edge,node。Edge 表示边文件，Node 表示节点文件。

对于 edge 文件每行 4 列，edge 编号 起 止 weight

对于 node 文件每行 3 列，node 编号 经纬度

举例：
$ head col.cedge
0 0 1 0.012406
1 2 3 0.001082
2 2 4 0.003068
3 5 6 0.000487
4 6 7 0.000449
5 8 9 0.000350
6 9 10 0.000603

举例：
$ head col.cnode
0 -103.992898 39.971728
1 -103.978333 39.971957
2 -103.890907 40.001074
3 -103.889636 40.001080
4 -103.891479 39.998348
5 -105.044258 39.820320
6 -105.043720 39.820174

3.3 road knn 测试说明

hiernn_gtree -h BJ_roadtree_t4_l8.idx -x testFile
分别表示为 -h 路网文件 -x 需要测试文件
Road 测试利用对外部文件的读取

testFile 文件格式：
第一行： num_obj // object 数目
接下来 num_obj 行： Obje_id //表示的是 vertex 的点的 id
query 数目： num_query //query 的数目
接下来 num_query 行 query_id //表示的是 vertex 的点

1.4 注意事项：

为 gtree 统一测试专门编写了新的测试文件，如果编译不正常，直接运行
$ make hiernn_gtree

四、silc（silc 文件夹）

4.1 编译说明

索引文件构建
g++ -O3 silc.cpp –o silc
knn 测试构建
g++ -O3 silc_knn.cpp –o silc_knn

4.2 索引树构建说明

调用规则 ./silc XX.node XX.edge 1 XX.morton 1

输入文件为纯文本格式，输入文件为纯文本格式，分为两个文件：edge,node。
Edge 表示边文件，Node 表示节点文件。
对于 edge 文件每行 4 列，edge 编号 起 止 weight
对于 node 文件每行 3 列，node 编号 经纬度

举例：
$ head col.cedge
0 0 1 0.012406
1 2 3 0.001082
2 2 4 0.003068
3 5 6 0.000487
4 6 7 0.000449
5 8 9 0.000350
6 9 10 0.000603

举例：
$ head col.cnode
0 -103.992898 39.971728
1 -103.978333 39.971957
2 -103.890907 40.001074
3 -103.889636 40.001080
4 -103.891479 39.998348
5 -105.044258 39.820320
6 -105.043720 39.820174

4.3 silc knn 测试说明

测试代码：silc_knn.cpp
./silc_knn ./data/col.cnode ./data/col.cedge 10000 $MORTON testFile

testFile 文件格式：
第一行： num_obj // object 数目
接下来 num_obj 行： Obje_id //表示的是 vertex 的点的 id
query 数目： num_query //query 的数目
接下来 num_query 行 query_id //表示的是 vertex 的点