Road Map

2017CSM1001,2017CSM1004

October 2017

1 Introduction

It is a client server project for querying a road map stored in a data base system. The server is maintaining information about the road map of a city (modeled as a graph) in disk pages simulated as files in our program. Clients would be query to know k-neighbors of any particular point.
Here multiple clients allow for query.
INPUT : Source node(S),Number of neighbors (K).
OUTPUT : The first K node ids found in the graph representation of the road map.

2 Problem with heuristic searching

Since server having huge collection of data while a client will query for a part of graph so for such kind of query search through all graph will be unnecessary resoucese utilization.

Client are allow to query simultaneously and might be their interest region independent to each other so for fast query search for each query independent.

So, What is solution for such kind of application ?
Divide graph into subgraph i.e. Partition of Graph

Okey, Why should we worry about it ?
Graph partition is NP-Hard Problem...

What!!!, how will we then partition of graph ?
Using our basic concept of partition
⇒ uniform Grid partition
⇒ BFS partition

What about it's performance and accuracy?
Simply, I don't know.....

3 Data Structure

- Adjancency List for maintaing the connectivity between the nodes of the graph.
- Socket Programming for handling Clients and Server.

3.1 Partitioning Method

- Uniform Grid
- BFS Grid

3.1.1 Data Structure for Uniform Grid file partition

Structure Index_Root : maintain information about Grid

```
struct Index_Root
{
        latitude
        lognitude
        Index_Root → next
}
```

Structure File_Root : maintain information about file and points lies inside grid and those crosses boundary having file name inside its neighbour stored.

```
struct File_Root
{
        latitude
        lognitude
        Is_Inside
        visited
        next_File_name
}
```

Two **Queues** are maintain:

- **Inside_Grid:** Used to perform BFS for points lies inside Grid
- **Crossing_Grid:** Used to maintain those nodes going out of grid

4 Graph Partition

INPUT: $\langle file1 \quad file2 \rangle$
OUTPUT: $\langle partition \quad graph \rangle$

Algorithm 1 Partitioning Data-Set using Uniform Grid approach

1: Find the smallest latitude and longitude(most lower x and most lower y)
2: From minimum point, calculate the lower left and upper right co-ordinate of partitioning the graph.
3: **while** traverse node inside the grid **do**
4: those points lies inside grid are mark as inside and kept in same file
5: those points crossing boundary are mark as outside and PUSH into **Crossing_Grid** Queue
6: **if** all points inside visited **then**
7: POP points from **Crossing_Grid** Queue and calculate mininum and maximum coordinate of grid inside this point lies as follow:
$LowerLeftLatitude = \lfloor \frac{Latitude}{Grid_{length}} \rfloor + Grid_{length}$,
$LowerLeftLongitude = \lfloor \frac{Longitude}{Grid_{height}} \rfloor + Grid_{height}$,
$UpperRightLatitude = LowerLeftLatitude + Grid_{length}$
$UpperRightLongitude = LowerLeftLongitude + Grid_{height}$
8: **end if**
9: Repeat untill all points visited
10: **end while**

Algorithm 2 Partition Data-Set using BFS approach

1: Sort Points in Decreasing Order of Degree
2: **while** not Visited **do**
3: Pick element with highest Degree and cover maximum unvisited node
4: apply one level BFS and maintain all visited in same file with root node where BFS start
5: Keep only those points those are not visited and connect with all previous visited points with new file points maintaining connection exactly same as in original Graph have.
6: Repeat Untill all points visited
7: **end while**

5 Query

INPUT: $\langle (latitude,\ longitude)\quad k \rangle$
OUTPUT: $\langle list\ \ of points,\ \ k - points \rangle$

Algorithm 3 Grid_Query(Point, k)

1: For given input value of (latitude, longitude) find Grid coordinates in between point lies
2: After find grid we have file name where its points are stored within these sub-graph find exact node of given(latitude, longitude)
3: Now apply BFS to get k-neighbours points
4: **while** $(no\_of\_points < k)$ **do**
5: Points lies either inside grid or crosses boundary
6: Points those lies inside grid belongs to same file and traverse through all node
7: Points those crosses boundary are in another file "name" associated in data structure and jump to this file name
8: Repeat [4-7] until k-points discovered
9: **end while**

Algorithm 4 BFS_Query((Point, k)

1: For given input value of (latitude, longitude) find into partition sub-graph it belongs to
2: **if** points belogns to one of **Root_node** **then**
3: **while** $(no\_of\_points < k)$ **do**
4: **if** points lies in textbfRoot_node **then**
5: count all point inside **Root_node** as neighbour
6: **else if** point belongs outside **Root_node then**
7: this node holding its **Root_node** and will be neighbour for this node
8: **end if**
9: **end while**
10: **else**
11: for all **Root_node** choose a root, that having smallest Euclidean distance with (latitude, longitude)
12: its **Root_node** must be neighbour of current point
13: **if** current point having other connection from other **Root_node then**
14: goto step[11]
15: **else**
16: call BFS_Query(**Root_node**, k)
17: **end if**
18: **end if**

6 Pros and Cons

- In Grid Partition Searching for a Grid for given point will be better than BFS partition.
- But, Range Query in Grid Partition will be slower than BFS partition.