



2015 级《数据结构》第 2 次作业

最短路径与社区识别

专业名称： 地理信息科学(遥感与 GIS)

学 号： 15303082

姓 名： 刘圣杰

联系方式： stop68@foxmail.com

指导老师： 李秋萍

完成时间： 2017/7/9

1.	实验内容.....	1
2.	实验软硬件环境.....	1
3.	模块/函数说明.....	1
3.1.	存放数据 Class	1
3.2.	Class Manager ， 最短路径与边介数	2
3.2.1.	读取结点/路网数据.....	2
3.2.2.	初始化图.....	2
3.2.3.	近似大地线重设道路长度.....	2
3.2.4.	两点最短路径.....	3
3.2.5.	最短路径与边介数.....	3
3.3.	社区识别	3
3.3.1.	Modularity 模块度计算.....	3
3.3.2.	真实距离标签传播算法.....	4
4.	实验结果.....	5
4.1.	实验流程	5
4.1.1.	两点最短路径.....	6
4.1.2.	全图最短路径与边介数.....	7
4.1.3.	社区识别.....	7
4.2.	核心函数运行时间	9
4.3.	讨论与算法对比(标签传播算法 VS Fast unfolding)	9
5.	总结.....	10
6.	附录.....	12
6.1.	附录 1 <边介数可视化>.....	12
6.2.	附录 2 <4.4km 社区识别>.....	13
6.3.	附录 3 <8.4km 社区识别>.....	14
6.4.	附录 4 <14km 社区识别>.....	15
6.5.	附录 5 <20km 社区识别>.....	16

1. 实验内容

构建一个有向图：

- a) 用户可任意选择两个节点计算最短路径
- b) 计算边介数
- c) 识别社区结构（标签传播算法）

2. 实验软硬件环境

windows 7 SP1 x64

Codeblocks 16.01(TDM-GCC 4.9.2, 32bit, SJLJ)

Intel i3-2350m, 8G

C++

3. 模块/函数说明

3.1. 存放数据 Class

Class Node: 存结点

Class Road: 存道路

Class Graph: 邻接表存路网，Graph 表头结点，存有指针 Destination*

Class Destination: 邻接表表结点，数据成员：

```
int ID; // 道路末结点 ID
double length; // 道路长度
int roadID; // 对应道路 ID
Destination* next;
```

3.2. Class Manager ， 最短路径与边介数

```
Node* nodeArray[NODE_SIZE];  
Road* pRoadHead;  
Road* ptrCurrentRoad;
```

分别存 node、road 头结点，ptrCurrentRoad 用于遍历路网

3.2.1. 读取结点/路网数据

```
int readFileRoad(char* );  
int readFileNodeArray(char* );
```

链表存道路，数组存结点。若文件不存在，return(-2)退出。若已读取道路，return(-1)退出，防止原链表无法获取内存泄漏。

3.2.2. 初始化图

```
int searchForNode(double,double);  
int initializeGraph(Graph*);
```

后者内有数组存 58 个非法道路 ID，放弃这些路。遍历道路，调用前者匹配结点（返回结点 ID）；new 一个表结点(Destination)并初始化；若匹配结点失败前者 return(-1)放弃这条路。

若 oneway==0，反向同操作。

3.2.3. 近似大地线重设道路长度

```
void setRoadLength();  
double getPtsDist(double lat1, double lng1, double lat2, double lng2);
```

后者返回道路 length (m)，前者遍历道路，调用 Road::setRoadLength()重设长度。

3.2.4. 两点最短路径

```
int shortestPath(int source, int destination, Graph* graph);
```

计算两点最短路径，打印。数组存储头结点，邻接表，未进行优先队列（二叉堆，斐波纳契堆）优化，时间复杂度 $O(n^2)$ 。

@return 0, 成功找到最短路径
@return 1, 起点无法到达任何点
@return 2, 起点到终点无道路连接
@return -1, 起点或终点超出范围
@return -2, 起点和终点相同

3.2.5. 最短路径与边介数

```
int calculateBetweenness(Graph* graph);
```

计算所有最短路径，求边介数，输出<result.txt>到当前目录。边介数可视化见附录 1。时间复杂度 $O(n^3)$ 。

3.3. 社区识别

3.3.1. Modularity 模块度计算

```
double calculateModularity(Graph* graph, int communityDistance);
```

Fig 1. <Modularity 公式>

$$Q = \sum_i (e_{ii} - a_i^2) = \sum_i e_{ii} - \sum_i a_i^2$$

使用标签传播算法识别社区，计算 modularity，打印。

使用公式求 modularity(Q)。程序中对公式进行变形，使之适应有向图。

3.3.2. 真实距离标签传播算法

```
double calculateModularity(Graph* graph, int communityDistance);  
int detectCommunity(Graph* graph, int x);
```

采用真实距离标签传播算法社区识别，需要传入一个最大社区距离限制。由于网络为真实道路，社区应有合理大小。

随机选取一个点（该点不应为不可到达任何点的点，该点最好为只能出发不能到达的点），程序中使用 `node[0]`（该点只能出发不能到达）。计算该点到全图最短距离，最短距离小于指定社区距离都归进一个社区。

然后，选取一个未被标记点，计算新社区。遇到已经被标记的点，计算被争夺点原社区平均 `node-to-node` 距离，与新社区平均 `node-to-node` 距离比较，小的社区争夺成功。

遍历结点到最后，由于每新生成一个社区都会对原有社区进行“腐蚀争夺”，无需二次循环即可完成社区识别。

直觉上，应对被争夺点使用 Dijkstra 算法计算平均 `node-to-node` 距离，但程序中只使用了 社区总道路长度/社区道路数 做简单处理。**反复思考，我不认为前者一定比后者优越，更不提效率上现有争夺算法更具优势。**

现有算法追求社区**紧凑度**，若社区越紧凑/路网越密集，争夺能力越强。像一个强核心对周围的辐射。但超过一定辐射半径后，不再拥有争夺能力。

最终输出<community.txt>文件记录每个 `node` 的社区分类。另见附录，有 4 个尺度的社区识别可视化。

4. 实验结果

4.1. 实验流程

启动程序，自动读取道路与结点（若失败退出程序，宏定义路径，表头已删除），输入数字控制程序。

1: 点到点最短路径计算

100: 计算全图 betweenness, 并输出文件<result.txt>

2: 计算指定 scale 的 modularity

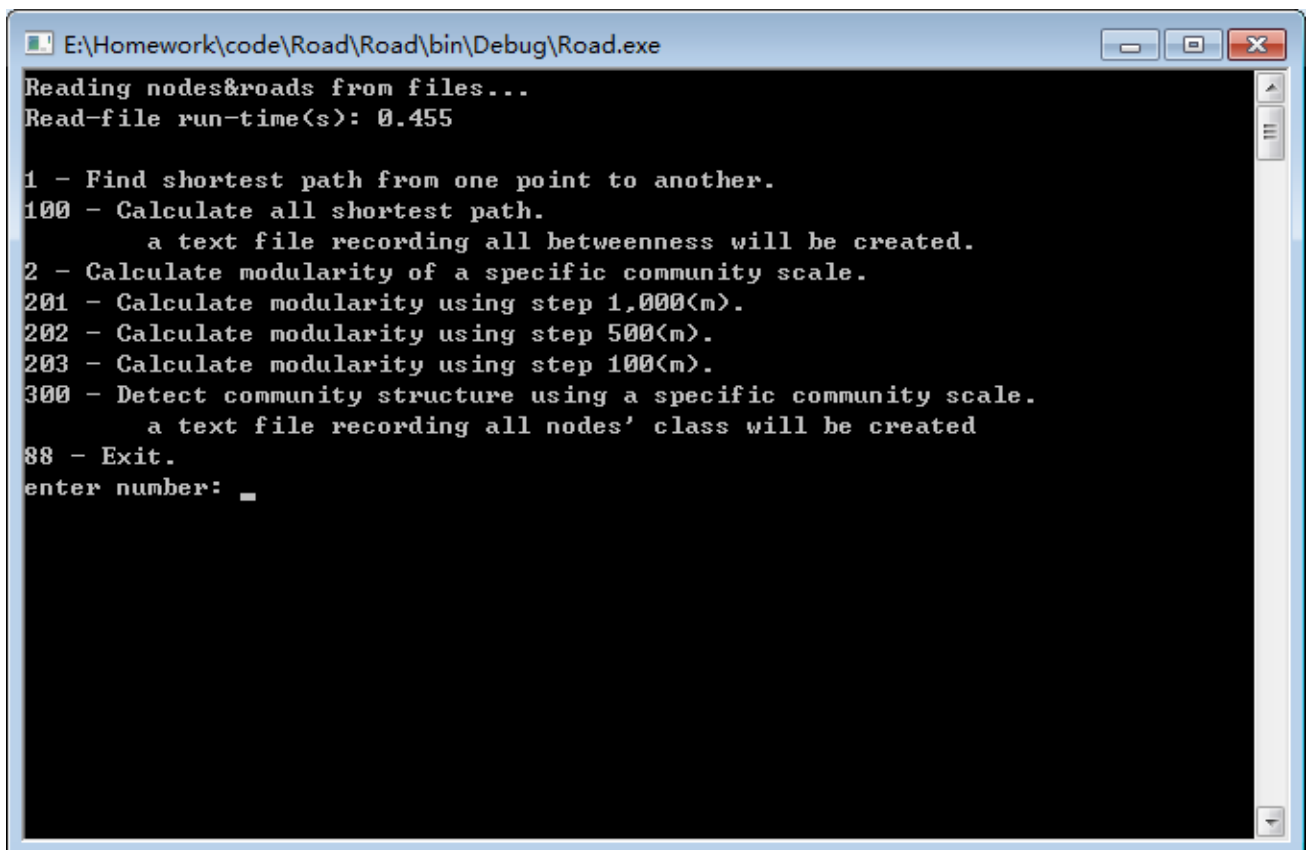
201: step 1000(m)计算指定最小 scale 到最大 scale 的 modularity

202: step 500(m) 计算指定最小 scale 到最大 scale 的 modularity

203: step 500(m) 计算指定最小 scale 到最大 scale 的 modularity

300: 社区识别并输出识别结果<community.txt>

Fig 2. <启动界面>

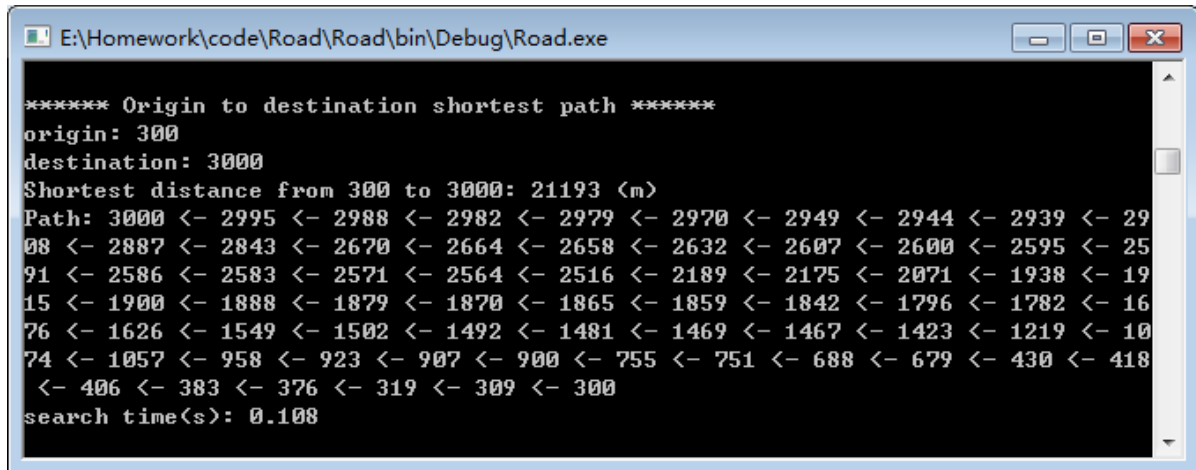


```
E:\Homework\code\Road\Road\bin\Debug\Road.exe
Reading nodes&roads from files...
Read-file run-time(s): 0.455

1 - Find shortest path from one point to another.
100 - Calculate all shortest path.
      a text file recording all betweenness will be created.
2 - Calculate modularity of a specific community scale.
201 - Calculate modularity using step 1,000(m).
202 - Calculate modularity using step 500(m).
203 - Calculate modularity using step 100(m).
300 - Detect community structure using a specific community scale.
      a text file recording all nodes' class will be created
88 - Exit.
enter number: _
```

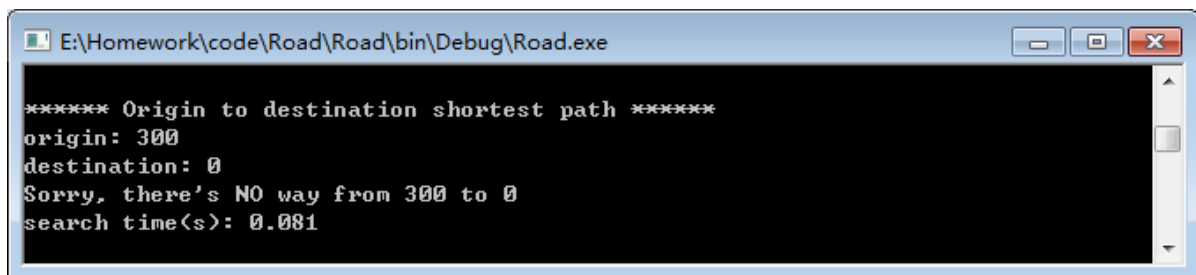
4.1.1. 两点最短路径

Fig 3. (a) <最短路径存在>



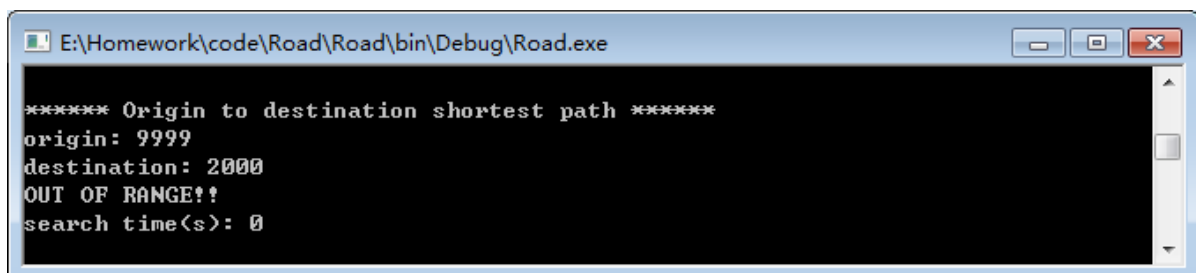
```
***** Origin to destination shortest path *****
origin: 300
destination: 3000
Shortest distance from 300 to 3000: 21193 (m)
Path: 3000 <- 2995 <- 2988 <- 2982 <- 2979 <- 2970 <- 2949 <- 2944 <- 2939 <- 2908 <- 2887 <- 2843 <- 2670 <- 2664 <- 2658 <- 2632 <- 2607 <- 2600 <- 2595 <- 2591 <- 2586 <- 2583 <- 2571 <- 2564 <- 2516 <- 2189 <- 2175 <- 2071 <- 1938 <- 1915 <- 1900 <- 1888 <- 1879 <- 1870 <- 1865 <- 1859 <- 1842 <- 1796 <- 1782 <- 1676 <- 1626 <- 1549 <- 1502 <- 1492 <- 1481 <- 1469 <- 1467 <- 1423 <- 1219 <- 1074 <- 1057 <- 958 <- 923 <- 907 <- 900 <- 755 <- 751 <- 688 <- 679 <- 430 <- 418 <- 406 <- 383 <- 376 <- 319 <- 309 <- 300
search time(s): 0.108
```

Fig 3. (b) <最短路径不存在>



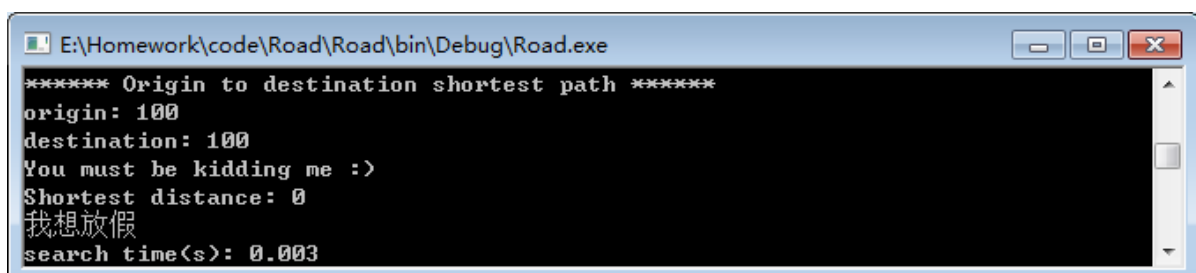
```
***** Origin to destination shortest path *****
origin: 300
destination: 0
Sorry, there's NO way from 300 to 0
search time(s): 0.081
```

Fig 3. (c) <输入超出范围>



```
***** Origin to destination shortest path *****
origin: 9999
destination: 2000
OUT OF RANGE!!
search time(s): 0
```

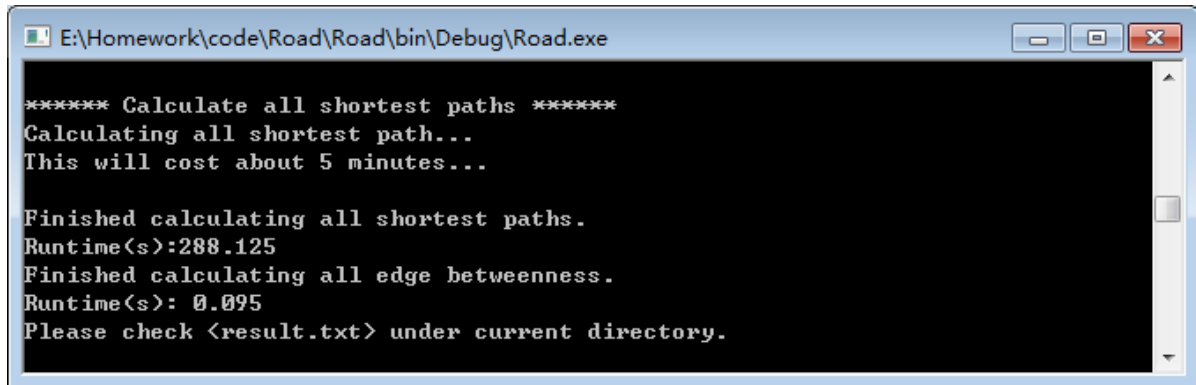
Fig 3. (d) <相同两点>



```
***** Origin to destination shortest path *****
origin: 100
destination: 100
You must be kidding me :>
Shortest distance: 0
我想放假
search time(s): 0.003
```


4.1.2. 全图最短路径与边介数

Fig 4.(a) <全图最短路径>

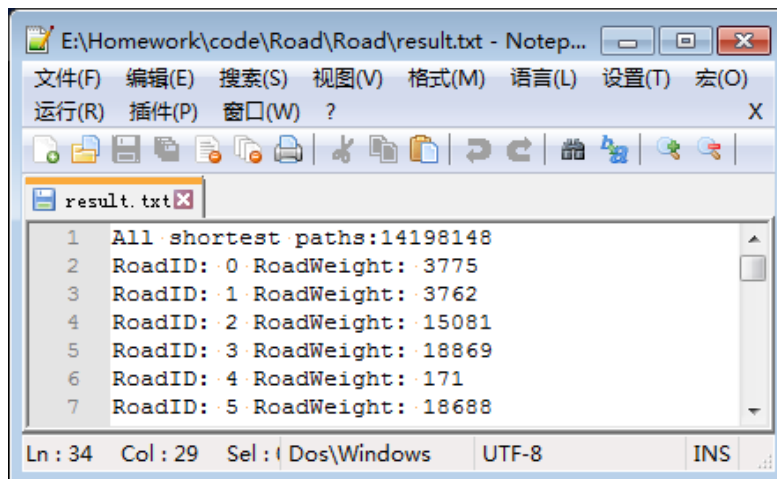


```
***** Calculate all shortest paths *****
Calculating all shortest path...
This will cost about 5 minutes...

Finished calculating all shortest paths.
Runtime(s):288.125
Finished calculating all edge betweenness.
Runtime(s): 0.095
Please check <result.txt> under current directory.
```

Fig 4.(b) <result.txt>

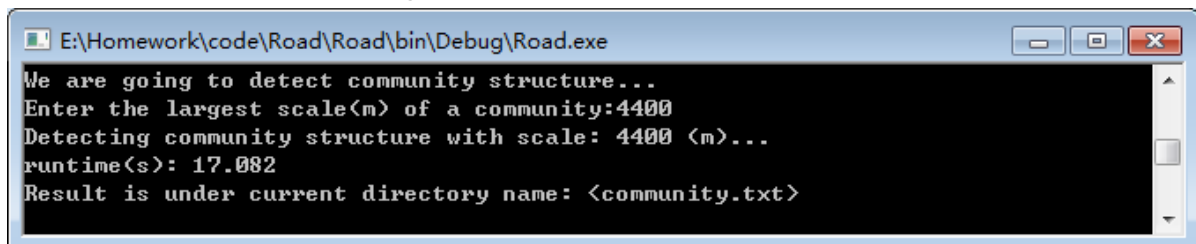
*边介数的可视化见附录 1.



```
1 All shortest paths:14198148
2 RoadID: 0 RoadWeight: 3775
3 RoadID: 1 RoadWeight: 3762
4 RoadID: 2 RoadWeight: 15081
5 RoadID: 3 RoadWeight: 18869
6 RoadID: 4 RoadWeight: 171
7 RoadID: 5 RoadWeight: 18688
```

4.1.3. 社区识别

Fig 5.(a) <4.4km 社区识别>



```
We are going to detect community structure...
Enter the largest scale(m) of a community:4400
Detecting community structure with scale: 4400 (m)...
runtime(s): 17.082
Result is under current directory name: <community.txt>
```

Fig 5.(b) <8.4km 社区识别>

```

E:\Homework\code\Road\Road\bin\Debug\Road.exe
We are going to detect community structure...
Enter the largest scale(m) of a community:8400
Detecting community structure with scale: 8400 <m>...
runtime(s): 9.919
Result is under current directory name: <community.txt>

```

Fig 5.(c) <8.4km 社区识别结果 community.txt>

```

E:\Homework\code\Road\Road\bin\Debug\comm...
文件(F) 编辑(E) 搜索(S) 视图(V) 格式(M) 语言(L) 设置(I) 宏(O)
运行(R) 插件(P) 窗口(W) ?
community.txt
3860 NodeID: 3859 Class: 96
3861 NodeID: 3860 Class: 96
3862 NodeID: 3861 Class: 115
3863 NodeID: 3862 Class: 116
3864 NodeID: 3863 Class: 103
3865 NodeID: 3864 Class: 114
3866 NodeID: 3865 Class: 103
3867 NodeID: 3866 Class: 96
3868 NodeID: 3867 Class: 110
Ln: 1 Col: 1 Sel: 0 | Dos\Windows UTF-8 INS

```

Fig 6.(a) <step1,000m 自 1,000m 到 30,000m Modularity 计算>

```

limitLength: 14000 class: 123 Modularity: 0.918508
limitLength: 15000 class: 120 Modularity: 0.894166
limitLength: 16000 class: 112 Modularity: 0.892049
limitLength: 17000 class: 110 Modularity: 0.908718
limitLength: 18000 class: 107 Modularity: 0.930811
limitLength: 19000 class: 106 Modularity: 0.934647
limitLength: 20000 class: 106 Modularity: 0.941659
limitLength: 21000 class: 106 Modularity: 0.939807
limitLength: 22000 class: 105 Modularity: 0.935441
limitLength: 23000 class: 105 Modularity: 0.930414
limitLength: 24000 class: 104 Modularity: 0.933192
limitLength: 25000 class: 104 Modularity: 0.93306
limitLength: 26000 class: 104 Modularity: 0.938881
limitLength: 27000 class: 103 Modularity: 0.92208
limitLength: 28000 class: 103 Modularity: 0.947348
limitLength: 29000 class: 103 Modularity: 0.962429
limitLength: 30000 class: 103 Modularity: 0.962958
runtime(s): 389.708
Result is under current directory name: <modularity_scale1000.txt>

```

Fig 6.(b) <step100m 自 3,100m 到 9,900m Modularity 计算>

```

limitLength: 8400 class: 149 Modularity: 0.868104
limitLength: 8500 class: 146 Modularity: 0.864797
limitLength: 8600 class: 146 Modularity: 0.861754
limitLength: 8700 class: 145 Modularity: 0.853287
limitLength: 8800 class: 144 Modularity: 0.851038
limitLength: 8900 class: 143 Modularity: 0.850377
limitLength: 9000 class: 142 Modularity: 0.852229
limitLength: 9100 class: 142 Modularity: 0.858579
limitLength: 9200 class: 140 Modularity: 0.861357
limitLength: 9300 class: 142 Modularity: 0.854214
limitLength: 9400 class: 142 Modularity: 0.855007
limitLength: 9500 class: 142 Modularity: 0.843101
limitLength: 9600 class: 141 Modularity: 0.847996
limitLength: 9700 class: 140 Modularity: 0.840455
limitLength: 9800 class: 139 Modularity: 0.834634
limitLength: 9900 class: 139 Modularity: 0.846011
runtime(s): 2164.36
Result is under current directory: <modularity_scale100.txt>

```

4.2. 核心函数运行时间

Table 1. <核心函数运行用时>

函数功能	函数名	用时(s)
读入文件	Manager:: readFileNameArray(char*) Manager:: readFileRoad(char*)	0.455
两点最短路径	Manager:: shortestPath(int, int, Graph*)	0.108
全图最短路径	Manager:: calculateBetweenness(Graph*)	288.125
道路边介数		0.095
模块度 step1,000 1,000-30,000(m)	double Manager:: calculateModularity(Graph* graph, int x)	389.7
模块度 step100 500-9,900(m)		13.0(avg)
		2194.4
		22.8(avg)
社区识别: 4,400m	int Manager::detectCommunity(Graph* graph, int x)	17.082
社区识别: 8,400m		9.919

4.3. 讨论与算法对比 (标签传播算法 VS Fast unfolding)

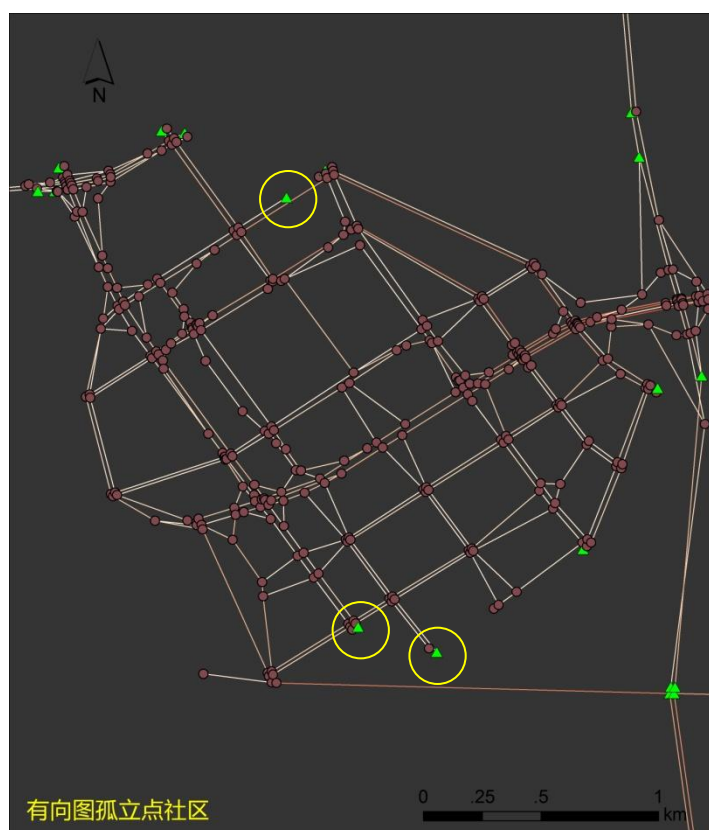
Table 2. <算法对比>

本文标签传播算法			Gephi 提供方法		
最大距离	社区数量	Modularity	社区数量	使用方法	
4.4km	253	0.864	0.841	254	Fast unfolding
8.4km	149	0.868	0.895	148	
14km	123	0.919	0.906	122	
20km	106	0.942	0.910	107	

使用 Gephi 0.9.1 计算 modularity (有向图, 无权重), 与本文标签传播算法较优解比较 (Fig. 9)。GN 算法太慢($O(V^3 * E)$)不考虑。

本文标签传播算法思路简单，不需二次循环，单次最差退化为全图 Dijkstra($O(V^3)$, 距离极小时)，通过变步长合理确定社区最大距离，modularity 整体优于 Fast unfolding。问题是，有向图中，有些点只出不进。若相连社区已标记，而该点直接连接点的边长 > 已连接社区 average node-to-node 距离，会出现孤立点社区的情况 (Fig 7.); 但这种现象在 Fast unfolding 也存在 (Fig 8.)。

Fig 7. <有向图孤立点社区>



原猜想 2km 最优，故跳 1km。后发现不对，先跳 2km 到 50km 直到 modularity 稳定，再对 3-10km 跳 500m 加密。最后狠下心跳 100m(微笑)。对于真实社区，500m 是足够小的尺度，如此所得 4.5/8.5km 与现有结果几无差别。

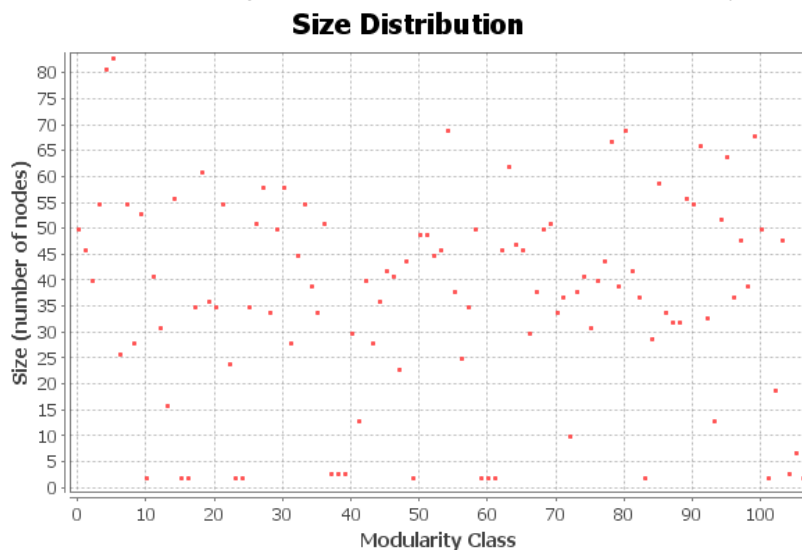
5. 总结

原设想社区步行可达，直径 2km。结果显示小尺度社区 modularity 较优解 0.864 在 4.4km 处，应理解为从中心到周边 2km 半径（近似真实街道或 TOD 小镇）。

在 4.4km-10km 间 modularity 仅上下波动，可能是城市在该范围联系紧密，南沙番禺各点在 4.4-10km 间联系紧密（中小型城市常见大小），这部分峰值 0.868(8.4km)。另一峰值 0.919(14km)，可能刚好到达各自最优半径。最后一个峰

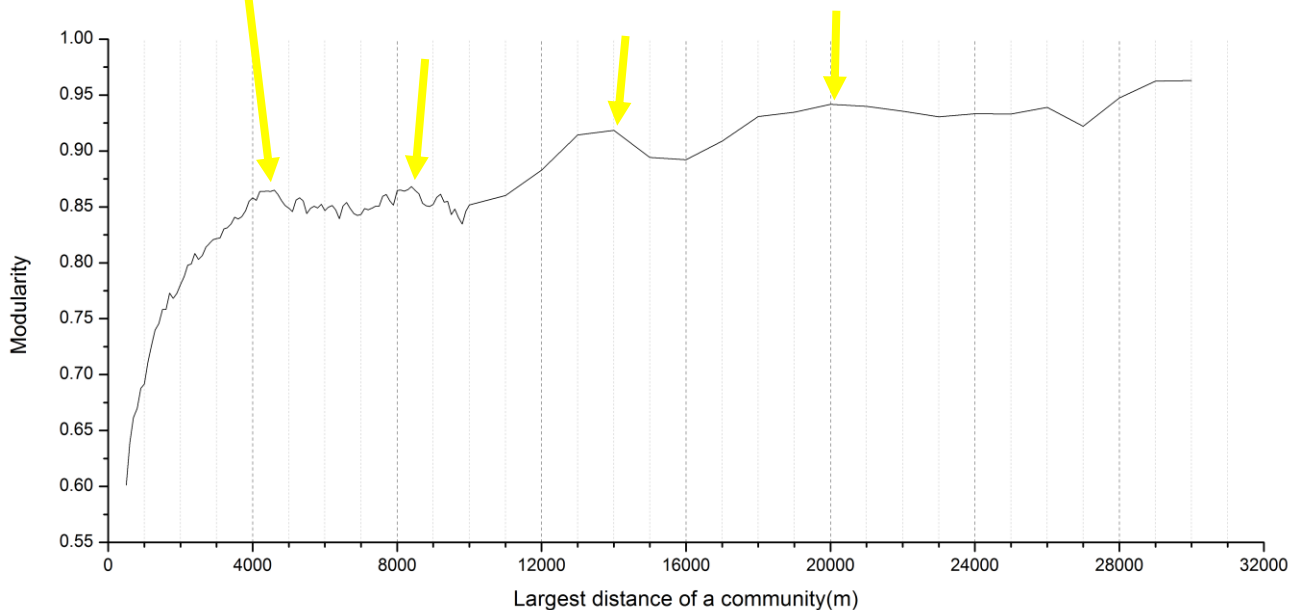
值 0.942(20km)接近全图最大值（全图 modularity=0.988，有少量不与主路网连接的点），已知番禺与南沙中心直线距离>20km，可能是 20km 为以番禺为中心的最大辐射半径。

Fig 8. <Fast unfolding 也存在孤立点社区(modularity=0.910)>



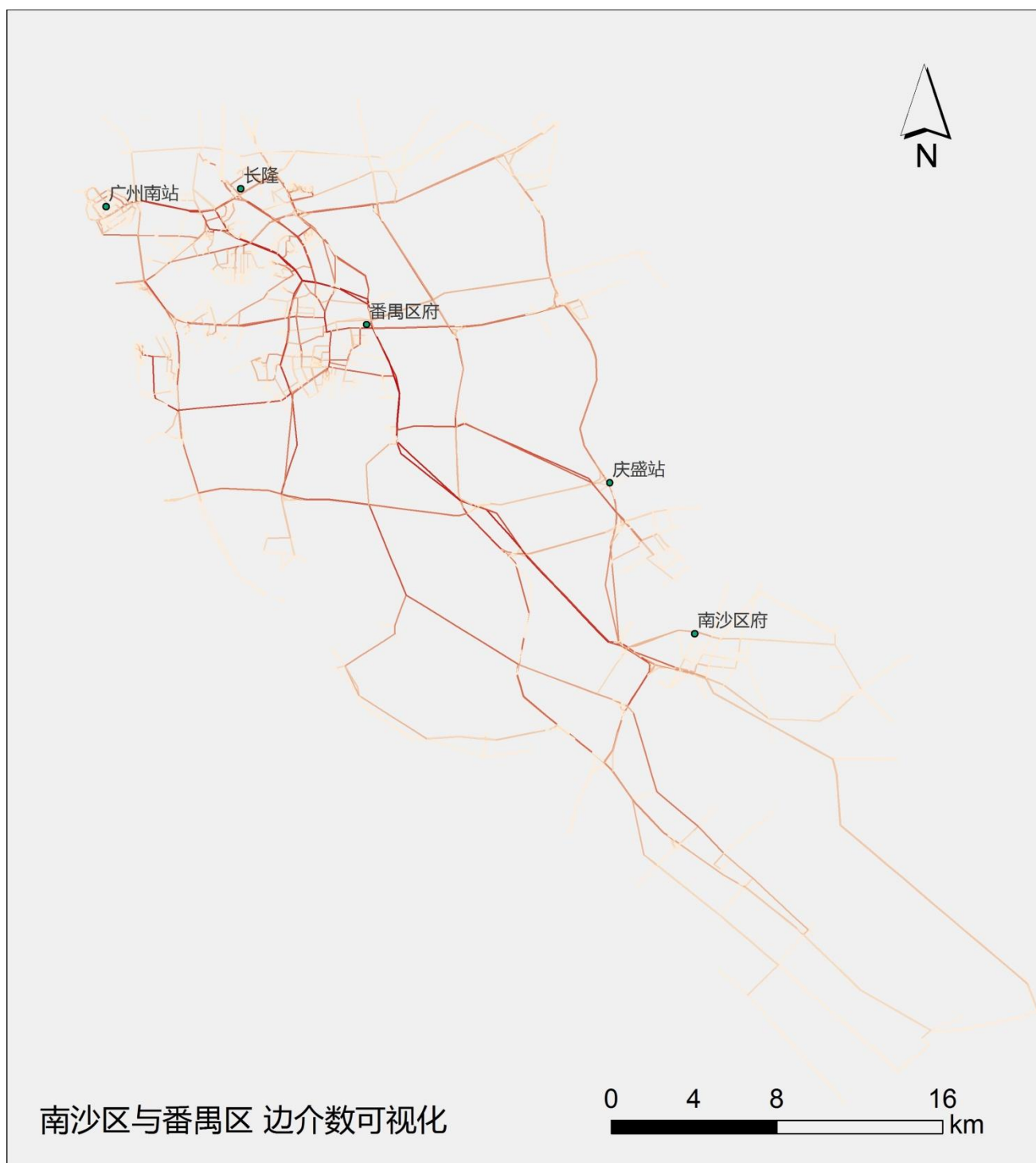
通过可视化（附录），比较 modularity，倾向使用 4.4km 结果作为小尺度社区（街道级），14km 结果作为大尺度社区（亚县区级）。

Fig 9. <modularity 与社区最大距离关系>

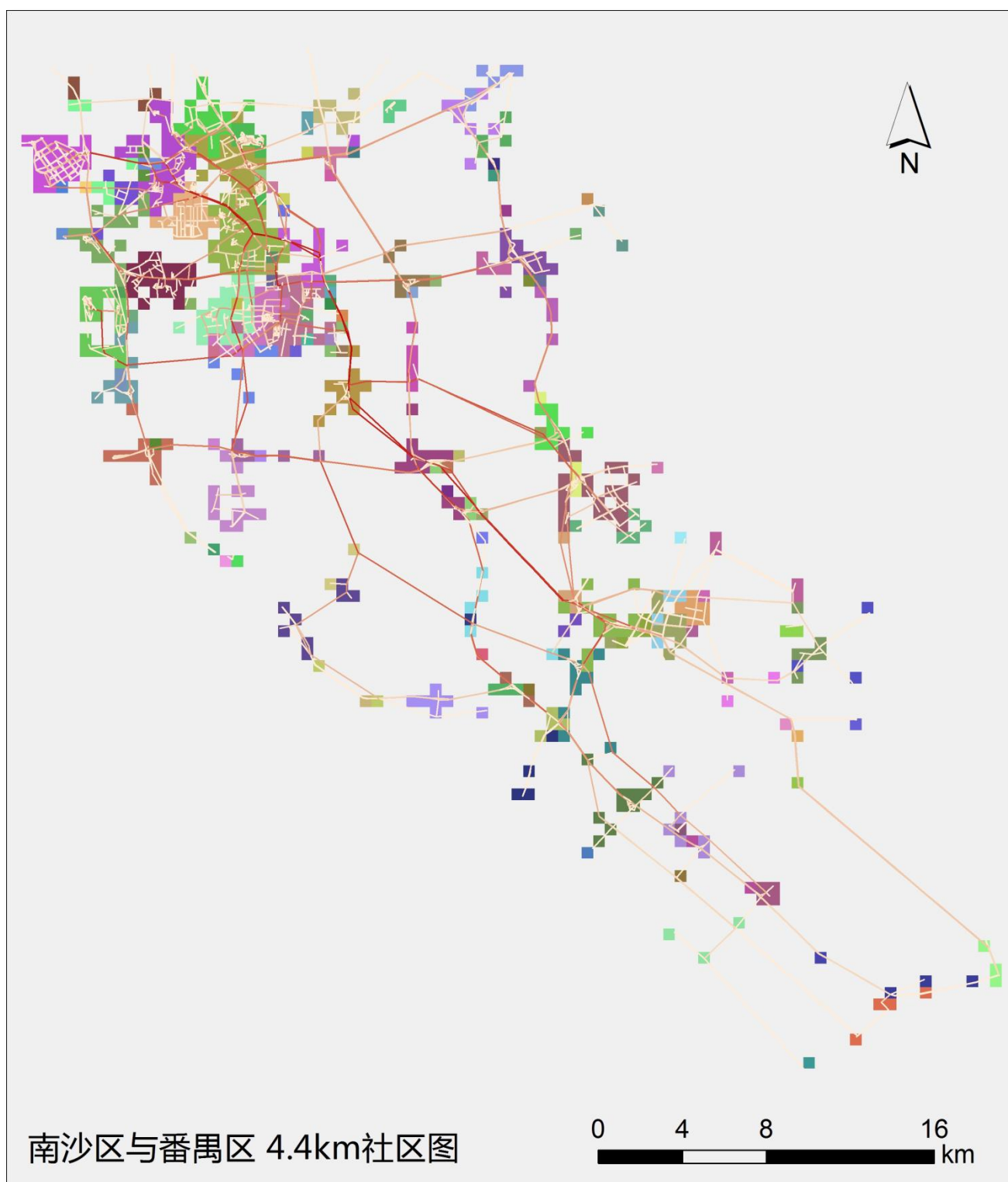


6. 附录

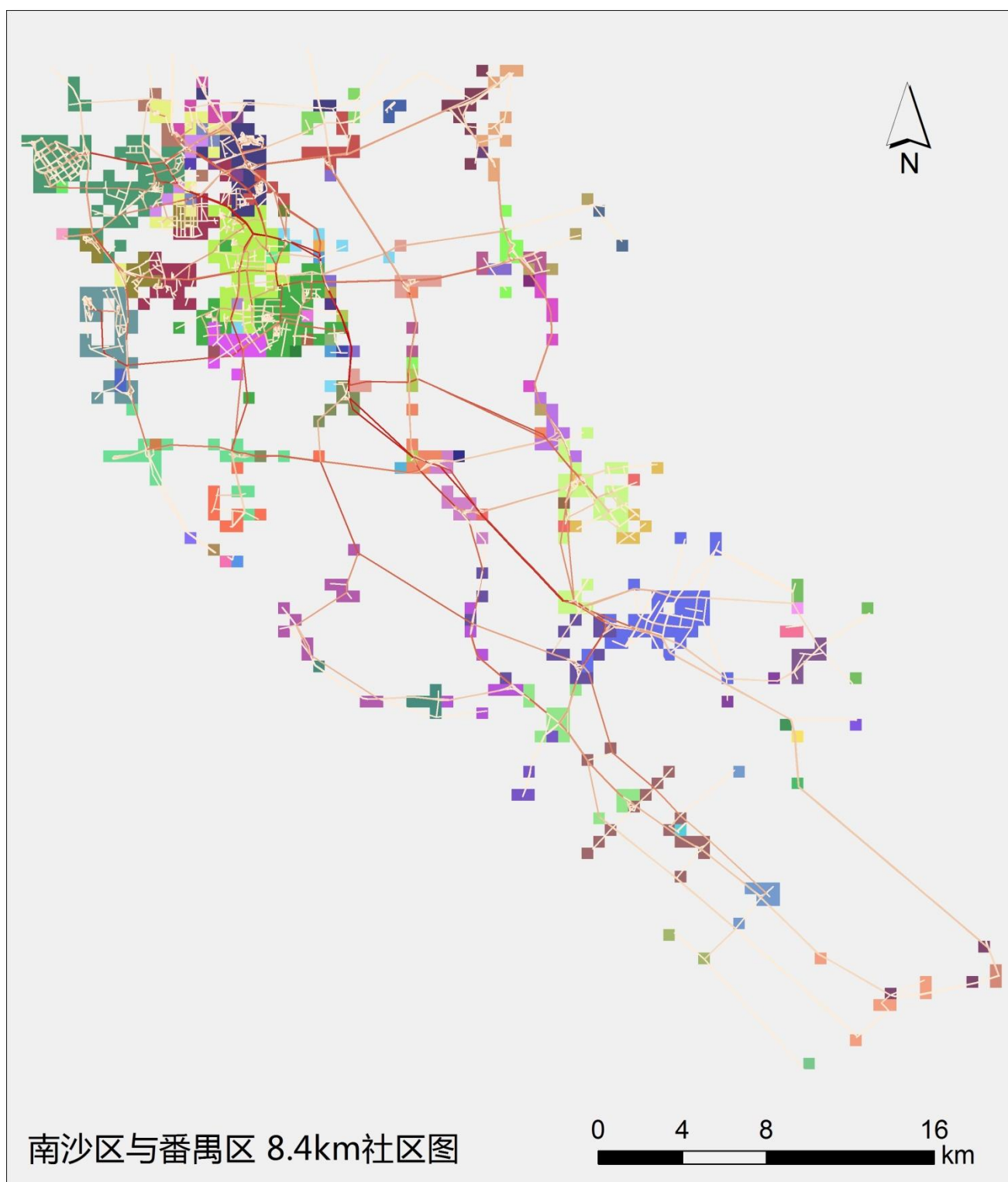
6.1. <边介数可视化>



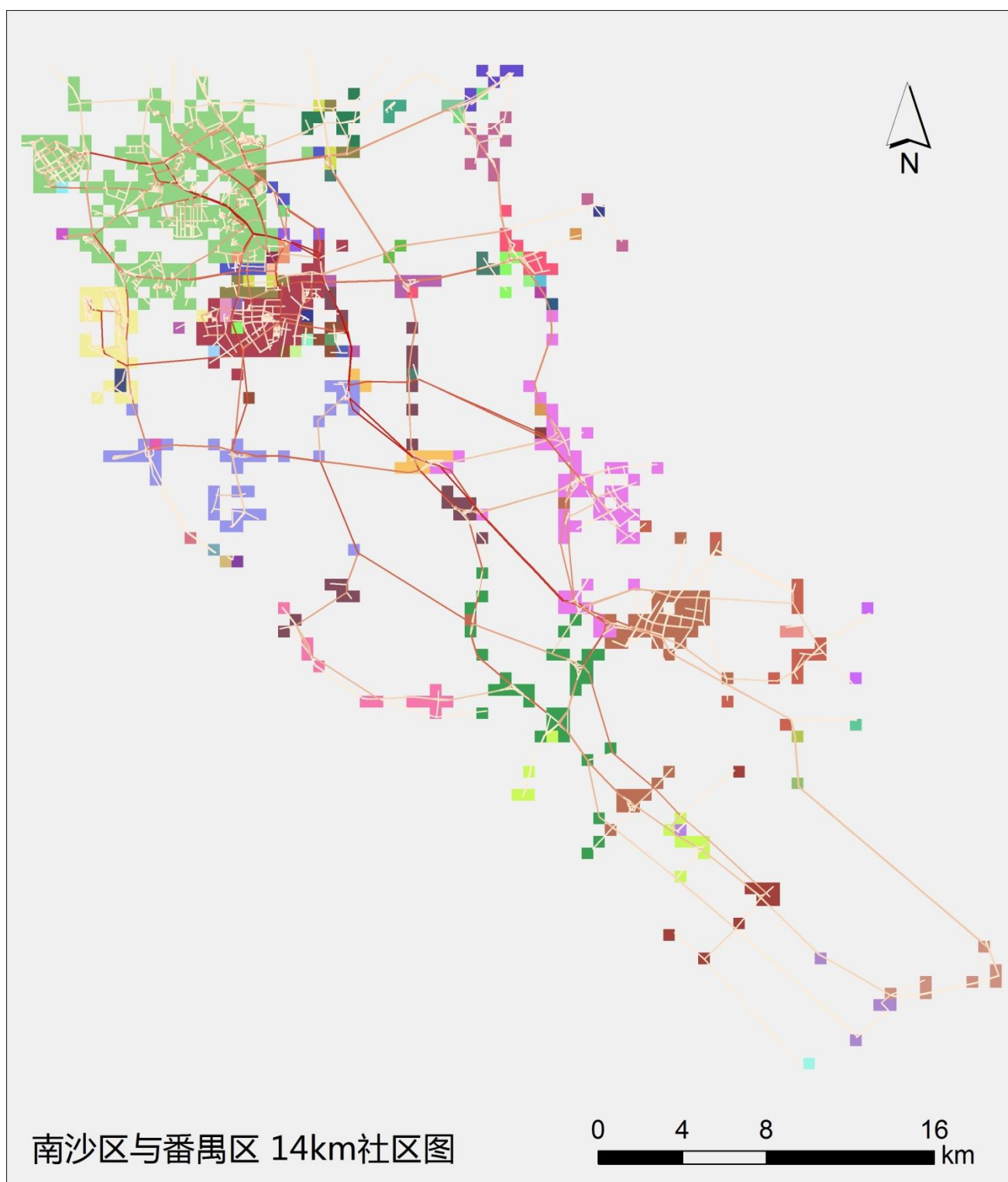
6. 2. <4. 4km 社区>



6.3. <8.4km 社区>



6. 4. <14km 社区>



6. 5. <20km 社区>

