算法设计：k-VCC的查找

石依凡 2020202264

目录

1 问题概览

k-VCC (k-Vertex Connected Component) 指无向图G中的一个极大子图G'，满足G'是一个k-点连通图，即至少删去k个顶点才能破坏其连通性，或者说G'的最小点割集的基数为k。

k-VCC在很多领域均有着广泛的应用：在社交网络中，计算出社交关系图的所有k-VCCs可以帮助确定社交网络中高度相关的社群用户，为广告和推荐系统提供宝贵的数据支撑；在共同作者网络中，确定k-VCCs可以帮助人们确定网络中的研究小组。

本次实验的目的，即为查找无向图网络中的所有k-VCC。

2 实验完成度

本实验完成了对中、小型数据集的全部k-VCC查找，并根据论文的内容实现了简单优化。

程序对于超大数据集无法在可接受的时间完成搜索。

3 实验说明

本实验采用C++程序设计语言编写，使用面向对象的思想，设计类`Graph`和相关方法，实现k-VCC的搜索。程序主要部分有：

- 使用最大流算法（*Edmond-Karp Algorithm*）搜索有向图从给定源点到汇点的最大流。
- 使用深度优先搜素，根据最大流算法搜索有向图的最小边割集。
- 使用无向图拆点边有向图的方法，把有向图的最小边割集转化为原图的最小点割集。
- 使用给定的算法，求解全图的最小割。
- 使用k-VCC的性质，求出全图的所有k-VCCs。
- 使用优化算法，对求解过程进行优化。

本程序使用到的STL库有：

```cpp
#include <vector>
#include <map>
#include <string>
#include <iostream>
#include <fstream>
#include <sstream>
#include <set>
#include <queue>
#include <stack>
#include <cstring>
#include <algorithm>
```

程序关于类`Graph`的设计如下：

```cpp
class Graph
{
public:
    // Construction Methods
    Graph() = default;
    Graph(const Graph & other) = default;
    Graph(const string & path);
    void setK(int s);
```

```
9
10      // Normal Methods
11      void addEdge(int from, int to);
12      void delEdge(int from, int to);
13      void remove(int v);
14      void remove(const set <int>& v);
15      Graph subGraph(const set <int> & v);
16      static vector <Graph> connectComponents(Graph g);
17      Graph genDir();
18      set <int> vertex();
19      set <int> canAccess(int source);
20
21      // Max Flow Methods
22      pair <int, set <Edge> > minEdgeCut(int u, int v);
23      vector <int> BFSFindPath(int u, int v);
24      void reverse(int u, int v);
25
26      // K-VCC Methods
27      set <int> locCut(int u, int v);
28      set <int> globalCut();
29      set <set<int> > kvcc();
30      static set<set<int> > kvccEnum(Graph g, int k);
31      static vector <Graph> overlapPartition(Graph G, const set
    <int>& S);
32
33      // Reduction Methods
34      bool isStrongVertex(int u);
35      void sweep(int v, bool * pru, int * deposit);
36      priority_queue <Node> getDistance(int u);
37
38  private:
39      map<int, set <int> > edges;
40      int k;
41      int degree = -1;
42  };
```

程序使用了两个全局函数：

```
bool isin(int u, const set <int>& v);
void report(string path, const set<set<int> > & result);
```

程序自定义了类型`Edge`和结构体`Node`如下：

```
typedef pair<int, int> Edge;
struct Node
{
    int v;
    int dis;
    bool operator<(const Node & other) const
    {
        if(dis != other.dis)
            return dis < other.dis;
        else
            return v < other.v;
    }
};
```

4 测试结果

对于给定的数据集（MiniDataSet），运行程序（注意，应事先删除第一行图的总体信息），得出k取2、3、5、7时的结果如下：

当$k = 2$时：

```
VCC Num: 4

K-VCC(No. 1): node_nums = 47
Node_ID: 0
Node_ID: 2
Node_ID: 3
Node_ID: 5
Node_ID: 6
Node_ID: 8
Node_ID: 9
Node_ID: 11
```

```
Node_ID: 14
Node_ID: 15
Node_ID: 16
Node_ID: 17
Node_ID: 19
Node_ID: 21
Node_ID: 22
Node_ID: 23
Node_ID: 26
Node_ID: 27
Node_ID: 29
Node_ID: 32
Node_ID: 36
Node_ID: 37
Node_ID: 40
Node_ID: 41
Node_ID: 42
Node_ID: 43
Node_ID: 44
Node_ID: 46
Node_ID: 47
Node_ID: 48
Node_ID: 50
Node_ID: 51
Node_ID: 52
Node_ID: 53
Node_ID: 55
Node_ID: 57
Node_ID: 58
Node_ID: 60
Node_ID: 64
Node_ID: 65
Node_ID: 67
Node_ID: 69
Node_ID: 71
Node_ID: 72
Node_ID: 73
Node_ID: 76
Node_ID: 78
```

```
51
52 K-VCC(No. 2): node_nums = 16
53 Node_ID: 1
54 Node_ID: 4
55 Node_ID: 10
56 Node_ID: 12
57 Node_ID: 28
58 Node_ID: 30
59 Node_ID: 31
60 Node_ID: 35
61 Node_ID: 50
62 Node_ID: 54
63 Node_ID: 62
64 Node_ID: 63
65 Node_ID: 66
66 Node_ID: 68
67 Node_ID: 70
68 Node_ID: 77
69
70 K-VCC(No. 3): node_nums = 5
71 Node_ID: 18
72 Node_ID: 19
73 Node_ID: 20
74 Node_ID: 39
75 Node_ID: 61
76
77 K-VCC(No. 4): node_nums = 6
78 Node_ID: 25
79 Node_ID: 33
80 Node_ID: 38
81 Node_ID: 67
82 Node_ID: 74
83 Node_ID: 75
```

当$k = 3$时：

```
1 # k = 3
2 VCC Num: 5
```

```
3
4  K-VCC(No. 1): node_nums = 15
5  Node_ID: 0
6  Node_ID: 2
7  Node_ID: 3
8  Node_ID: 5
9  Node_ID: 6
10 Node_ID: 11
11 Node_ID: 15
12 Node_ID: 17
13 Node_ID: 23
14 Node_ID: 29
15 Node_ID: 42
16 Node_ID: 44
17 Node_ID: 50
18 Node_ID: 55
19 Node_ID: 60
20
21 K-VCC(No. 2): node_nums = 15
22 Node_ID: 1
23 Node_ID: 4
24 Node_ID: 10
25 Node_ID: 12
26 Node_ID: 28
27 Node_ID: 30
28 Node_ID: 31
29 Node_ID: 35
30 Node_ID: 54
31 Node_ID: 62
32 Node_ID: 63
33 Node_ID: 66
34 Node_ID: 68
35 Node_ID: 70
36 Node_ID: 77
37
38 K-VCC(No. 3): node_nums = 30
39 Node_ID: 8
40 Node_ID: 9
41 Node_ID: 14
```

```
Node_ID: 16
Node_ID: 21
Node_ID: 22
Node_ID: 23
Node_ID: 26
Node_ID: 27
Node_ID: 32
Node_ID: 36
Node_ID: 37
Node_ID: 40
Node_ID: 41
Node_ID: 43
Node_ID: 46
Node_ID: 47
Node_ID: 48
Node_ID: 51
Node_ID: 52
Node_ID: 53
Node_ID: 57
Node_ID: 58
Node_ID: 64
Node_ID: 65
Node_ID: 69
Node_ID: 71
Node_ID: 72
Node_ID: 76
Node_ID: 78

K-VCC(No. 4): node_nums = 4
Node_ID: 18
Node_ID: 20
Node_ID: 39
Node_ID: 61

K-VCC(No. 5): node_nums = 6
Node_ID: 25
Node_ID: 33
Node_ID: 38
Node_ID: 67
```

```
Node_ID: 74
Node_ID: 75
```

当$k = 5$时:

```
# k = 5
VCC Num: 4

K-VCC(No. 1): node_nums = 6
Node_ID: 2
Node_ID: 3
Node_ID: 6
Node_ID: 15
Node_ID: 42
Node_ID: 60

K-VCC(No. 2): node_nums = 6
Node_ID: 4
Node_ID: 30
Node_ID: 54
Node_ID: 62
Node_ID: 66
Node_ID: 68

K-VCC(No. 3): node_nums = 19
Node_ID: 9
Node_ID: 14
Node_ID: 21
Node_ID: 22
Node_ID: 32
Node_ID: 36
Node_ID: 37
Node_ID: 40
Node_ID: 41
Node_ID: 43
Node_ID: 47
Node_ID: 52
Node_ID: 58
```

```
Node_ID: 64
Node_ID: 69
Node_ID: 71
Node_ID: 72
Node_ID: 76
Node_ID: 78

K-VCC(No. 4): node_nums = 6
Node_ID: 27
Node_ID: 46
Node_ID: 48
Node_ID: 51
Node_ID: 53
Node_ID: 57
```

当$k = 7$时:

```
# k = 7
VCC Num: 1

K-VCC(No. 1): node_nums = 15
Node_ID: 9
Node_ID: 32
Node_ID: 36
Node_ID: 37
Node_ID: 40
Node_ID: 41
Node_ID: 43
Node_ID: 47
Node_ID: 52
Node_ID: 58
Node_ID: 64
Node_ID: 69
Node_ID: 72
Node_ID: 76
Node_ID: 78
```

附录A 测试数据集

MiniDataSet的点对表示如下：

```
0 5
0 11
0 17
0 23
0 29
0 44
0 55
1 30
1 31
1 35
1 54
1 62
1 63
1 68
1 77
2 3
2 6
2 15
2 42
2 60
3 6
3 15
3 42
3 50
3 60
4 10
4 30
4 54
4 62
4 66
4 68
4 70
5 11
5 17
5 23
```

```
5 29
5 44
5 55
6 11
6 15
6 42
6 60
7 45
8 23
8 27
8 44
8 51
8 53
9 32
9 36
9 37
9 40
9 41
9 43
9 47
9 58
9 64
9 72
9 76
9 78
10 66
10 68
10 70
11 15
11 55
11 60
12 28
12 31
12 35
12 50
12 63
12 77
13 33
13 34
```

```
14 21
14 22
14 32
14 41
14 47
14 71
14 73
15 42
15 50
15 60
16 26
16 36
16 40
16 65
16 67
17 23
17 29
17 55
18 19
18 20
18 39
18 61
19 22
19 39
19 73
20 39
20 61
21 22
21 32
21 41
21 47
21 49
21 71
21 72
22 32
22 47
22 71
22 73
22 78
```

```
23 27
23 29
23 44
23 53
24 49
25 33
25 74
25 75
26 36
26 40
26 48
26 57
26 65
26 67
27 46
27 48
27 51
27 53
27 57
28 31
28 50
28 63
29 55
30 54
30 62
30 63
30 66
30 68
30 77
31 35
31 63
31 77
32 37
32 41
32 43
32 47
32 64
32 71
32 72
```

```
32 76
32 78
33 38
33 67
33 74
35 63
35 77
36 37
36 40
36 43
36 46
36 48
36 52
36 57
36 64
36 65
36 69
36 72
37 40
37 41
37 43
37 47
37 52
37 58
37 64
37 72
37 76
37 78
38 67
38 74
38 75
39 61
40 43
40 46
40 48
40 52
40 64
40 65
40 69
```

```
40 72
40 76
41 43
41 47
41 52
41 64
41 71
41 72
41 76
41 78
42 44
42 50
42 60
43 47
43 52
43 58
43 64
43 69
43 71
43 72
43 76
43 78
46 48
46 51
46 53
46 57
47 58
47 64
47 71
47 72
47 76
47 78
48 51
48 53
48 57
51 53
51 57
52 64
52 65
```

```
52 69
52 72
52 76
53 57
54 62
54 66
54 68
54 70
55 60
56 59
58 64
58 72
58 78
62 66
62 68
62 77
63 77
64 69
64 72
64 76
64 78
65 69
66 68
66 70
67 75
68 70
69 72
69 76
71 72
71 76
72 76
72 78
74 75
```