K-VCC 实验报告

------2020200982 闫世杰

实现了全局 K-VCC 的查找及优化

图结构：

```
/*------struct------*/
struct TUG{
    map< unsigned, vector<unsigned> > E; //store edge
    set<unsigned> V; //store vnode

};
typedef pair<unsigned, unsigned> Edge; //egde
typedef pair<unsigned, unsigned> Pointer_Pair; //point pair
```

图函数：

```
/*------some basic graph function------*/
//delete the edge(E) from the graph(G)
void DelE(TUG&G, Edge E);

//delete the vnode(id) from the graph(G)
void DelNID(TUG&G, unsigned id);

//get the min degree vnode of the graph(G)
unsigned Getmin(TUG G);

//use V_set(subV) make a subGraph of graph(G)
TUG SG(TUG G, set<unsigned> SubV);

//DFS to get V_set in the same connect graph
set<unsigned> Get_V(TUG G, unsigned s);

//reverse the edge
void GR(TUG&G, vector<unsigned>path);

//construct the direct_flow_graph of G
TUG U2D(TUG G);
```

1. 点对间 Min_Cut 的计算

根据 Edmond-Karp 算法实现点对间的最小割计算

```cpp
int GetMinCut(TUG G, unsigned s, unsigned t, unsigned maxflow, vector<Edge>&cut){
//calculate the maximum flow and min_cut_set(store int (set)cut) from s to t
    unsigned flow;
    auto Es = GetMaxFlow(G,s,t,flow); //calculate the maxflow of <s,t> in the graph
    if(flow==0) return 1;//get the cut,stop working
    vector<Edge> CutCond;
    for( auto i: Es ) {
        auto e = i.first; auto f = i.second;
        if( f!=1 || e.first==s || e.second==t || e.first%2 || e.second!=e.first+1 )
            continue;
        CutCond.push_back(e);
    }
    while( !CutCond.empty() ) { //delete condidate edge and recursively GetMinCut
        auto e = CutCond.back();  CutCond.pop_back();
        cut.push_back(e);
        if( cut.size() > maxflow ) {
            cut.pop_back();
            break;
        }
        TUG Gc = G; DelE(Gc, e);
        if(GetMinCut(Gc, s, t, maxflow, cut)) //recursively GetMinCut
            return 1;
    }
    cut.pop_back();
    return 0;
}
```

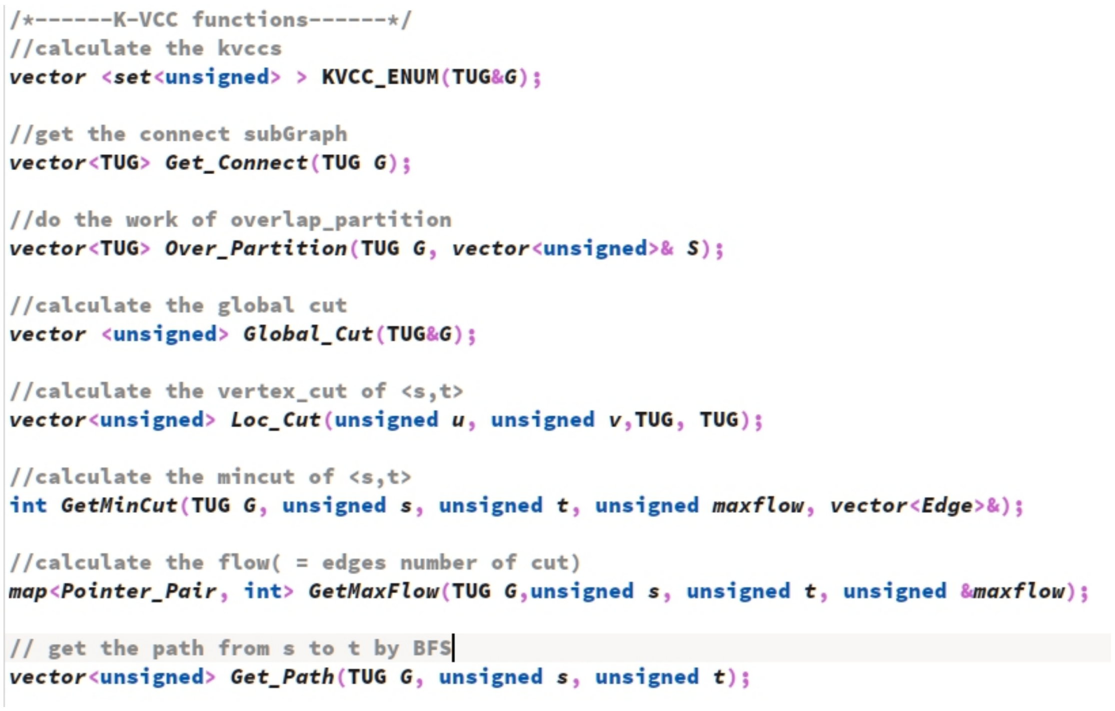
2. 全图 Min_Cut 的计算:实现 ppt 中的 Algorithm2 即可

```cpp
vector <unsigned> Global_Cut(TUG&G){ //calculate the global cut
    TUG TG = U2D(G); //construct direct flow grap of G
    unsigned u = Getmin(G); //select u with the minimum degree
    int *pru = new int [maxid+1];
    int *deposit = new int[maxid+1];
    for( int i = 0; i <= maxid; i++ )
        pru[i] = deposit[i] = 0;

    SWEEP(u, pru, deposit, G);
    //DFS get other node in the connect graph in a non_ascending order
    set<unsigned>  Vs = Get_V(G, u);
    //step1: find the min_cut of <u,others>
    for( auto v:Vs ){
        if( pru[v] )
            continue;
        auto S = Loc_Cut(u, v, TG, G);
        if( !S.empty() ) return S;
        SWEEP(v, pru, deposit, G);
    }
    //step2: find the min_cut of <u,neighbors>
    if( !Strong_Side_Vertex(G, u) )
        for( auto i = G.E[u].begin(); i != G.E[u].end(); i++ )
            for( auto j = G.E[u].begin(); j != G.E[u].end(); j++ ) {
                if( i == j ) continue;
                auto S = Loc_Cut(*i, *j, TG, G);
                if( !S.empty() )
                    return S;
            }
    return {};
}
```

3. 全图 K-VCC 的计算

根据 Algorithm1,调用已实现的 Global_Cut 即可

所使用函数如下(具体实现见 KVCC.cpp)

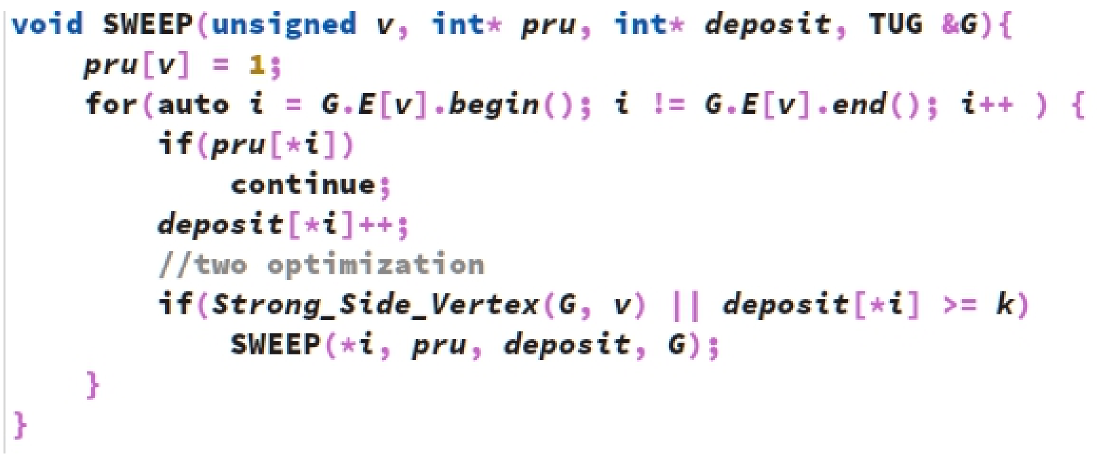


```cpp
/*------K-VCC functions------*/
//calculate the kvccs
vector <set<unsigned> > KVCC_ENUM(TUG&G);

//get the connect subGraph
vector<TUG> Get_Connect(TUG G);

//do the work of overlap_partition
vector<TUG> Over_Partition(TUG G, vector<unsigned>& S);

//calculate the global cut
vector <unsigned> Global_Cut(TUG&G);

//calculate the vertex_cut of <s,t>
vector<unsigned> Loc_Cut(unsigned u, unsigned v,TUG, TUG);

//calculate the mincut of <s,t>
int GetMinCut(TUG G, unsigned s, unsigned t, unsigned maxflow, vector<Edge>&);

//calculate the flow( = edges number of cut)
map<Pointer_Pair, int> GetMaxFlow(TUG G,unsigned s, unsigned t, unsigned &maxflow);

// get the path from s to t by BFS
vector<unsigned> Get_Path(TUG G, unsigned s, unsigned t);
```

4. 优化

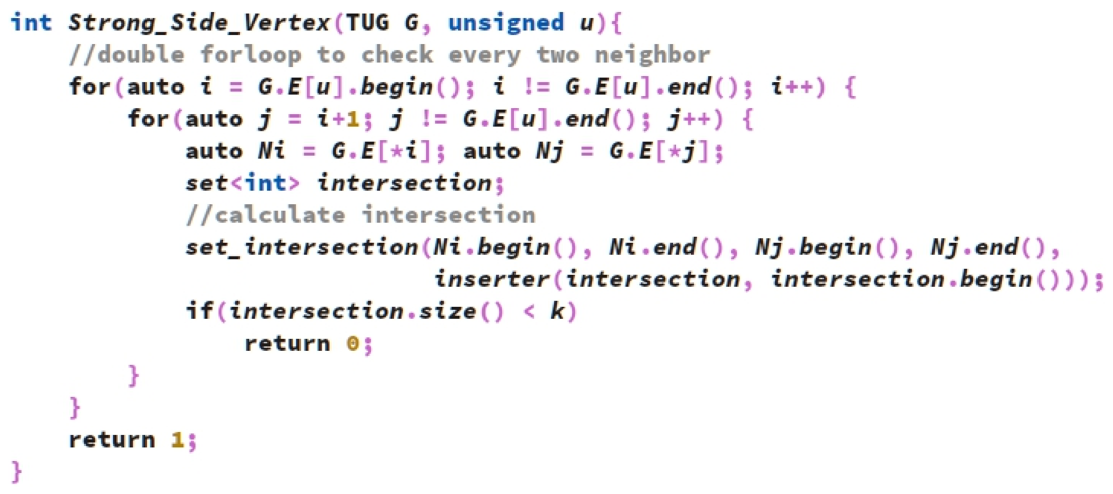
根据 ppt 中给出的算法

1. 实现 SWEEP(u,pru,deposit)



```cpp
void SWEEP(unsigned v, int* pru, int* deposit, TUG &G){
    pru[v] = 1;
    for(auto i = G.E[v].begin(); i != G.E[v].end(); i++ ) {
        if(pru[*i])
            continue;
        deposit[*i]++;
        //two optimization
        if(Strong_Side_Vertex(G, v) || deposit[*i] >= k)
            SWEEP(*i, pru, deposit, G);
    }
}
```

2. 实现 Strong Side_Vertex 的判断



```
int Strong_Side_Vertex(TUG G, unsigned u){
    //double forloop to check every two neighbor
    for(auto i = G.E[u].begin(); i != G.E[u].end(); i++) {
        for(auto j = i+1; j != G.E[u].end(); j++) {
            auto Ni = G.E[*i]; auto Nj = G.E[*j];
            set<int> intersection;
            //calculate intersection
            set_intersection(Ni.begin(), Ni.end(), Nj.begin(), Nj.end(),
                             inserter(intersection, intersection.begin()));
            if(intersection.size() < k)
                return 0;
        }
    }
    return 1;
}
```

虽然进行了优化，但是由于 Edmond-Karp 算法的复杂度较高，整个程序的运行复杂度也很高，大图难以在短时间内完成

所有函数定义在 KVCC.h 中，实现在 KVCC.cpp 中

在 main 函数中修改 path 可执行不同文件

(个人设置为 MiniDataSet.txt，较大图无法在短时间内完成)

g++ KVCC.cpp -o KVCC

生成可执行文件 KVCC，执行 KVCC，输入 k 值即可

在图 MiniDataSet 上运行 k=2~8 得到的结果与答案相同

个人运行出的结果保存在 Output_for_MiniData 中，

可视化结果在 Picture 中