大作业: k-VCC

论文：Enumerating k-Vertex Connected Components in Large Graphs

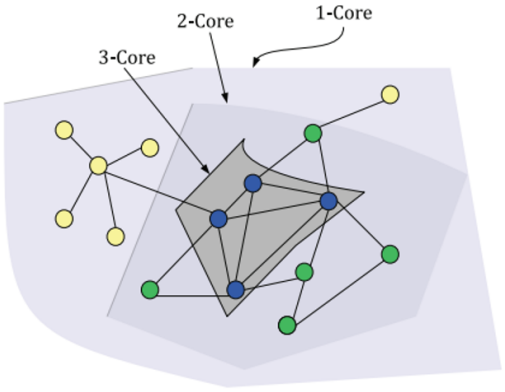
1. Introdution

1.1. K-Vertex Connected Component(K-VCC)的定义

无向图G中的极大子图G'，满足G'是一个k-点连通图(至少去掉k个顶点才能破坏连通性，或者说G'的最小点割集大小为k)

1.2. k-VCC的性质

- k-VCC一定在k-core中
  k-core定义为：满足子图内所有点度数$>=k$的极大子图



  k-core有两个性质：
  - 1. 大core一定被包含在小core内
  - 2. core的划分是唯一的
- 不同k-VCC至多相交于$k-1$个点

1.3. 大作业任务：

给定一个无向图，找到图中所有的k-VCC

- 朴素算法(60分)
  - Task1: 给定无向连通图G以及源点、汇点，计算最小点割(20分)
  - Task2: 给定无向连通图G，求全局最小点割(20分)
  - Task3: 给定无向图G，给出找到图中所有k-VCC的朴素算法(20分)

- 简单优化(40分)
  见 3. 算法优化

2. 朴素算法

2.1. 无向图局部最小点割

提示：

- 对于求给定源点和汇点的最小点割，如果直接使用最大流最小割定理求出来的是最小边割，如何转化为点割?
- 如何在残量网络中搜索找到割边，从而得到对应的割点?

2.2. 无向图全局最小点割

基本思路如下：
任意选取一个源点 u(一般是选择度数最小的点)

- Case1: $u \notin S$，计算 u 与其他所有点之间最小割
- Case2: $u \in S$，计算 u 的所有邻居点对之间最小割
  (下图中 V 为全体点集；$N(u)$ 表示 u 的所有邻居节点；可以忽略SC；在这一阶段不对 k 作限制；LOC-CUT的求法使用2.1.中实现的算法即可)

Algorithm 2 GLOBAL-CUT(G, k)

Input: a graph G and an integer k;
Output: a vertex cut with fewer than k vertices;

compute a sparse certification $\mathcal{SC}$ of G;
select a source vertex u with the minimum degree;
construct the directed flow graph $\overline{\mathcal{SC}}$ of $\mathcal{SC}$;
for all $v \in V$ **do**
  $\mathcal{S} \leftarrow$ LOC-CUT$(u, v, \overline{\mathcal{SC}}, \mathcal{SC})$; step1: u与其他点之间是否有割集
  if $\mathcal{S} \neq \emptyset$ **then return** $\mathcal{S}$;
for all $v_a \in N(u)$ **do**
  for all $v_b \in N(u)$ **do**
    $\mathcal{S} \leftarrow$ LOC-CUT$(v_a, v_b, \overline{\mathcal{SC}}, \mathcal{SC})$; step2: u的邻居相互之间是否有割集
    if $\mathcal{S} \neq \emptyset$ **then return** $\mathcal{S}$;
return $\emptyset$;

Procedure LOC-CUT$(u, v, \overline{G}, G)$ 返回u,v之间小于k的点割集
if $v \in N(u)$ or $v = u$ **then return** $\emptyset$;
$\lambda \leftarrow$ calculate the maximum flow from u to v in $\overline{G}$;
if $\lambda \geq k$ **then return** $\emptyset$;
compute the minimum edge cut in $\overline{G}$;
return the corresponding vertex cut in G;

2.3. k-VCC朴素算法

Algorithm 1 KVCC-ENUM(G, k)

Input: a graph G and an integer k;
Output: all k-vertex connected components;

1: $VCC_k(G) \leftarrow \emptyset$;
2: **while** $\exists u : d(u) < k$ **do** remove u and incident edges;
3: identify connected components $\mathcal{G} = \{G_1, G_2, ..., G_t\}$ in G;
4: **for all** connected component $G_i \in \mathcal{G}$ **do**
5: $\quad \mathcal{S} \leftarrow$ GLOBAL-CUT(G_i, k);
6: $\quad$ **if** $\mathcal{S} = \emptyset$ **then**
7: $\quad\quad VCC_k(G) \leftarrow VCC_k(G) \cup \{G_i\}$;
8: $\quad$ **else**
9: $\quad\quad \mathcal{G}_i \leftarrow$ OVERLAP-PARTITION$(G_i, \mathcal{S})$; gi={Gi1, Gi2, ...}
10: $\quad\quad$ **for all** $G_i^j \in \mathcal{G}_i$ **do**
11: $\quad\quad\quad VCC_k(G) \leftarrow VCC_k(G) \cup$ KVCC-ENUM(G_i^j, k);
12: **return** $VCC_k(G)$;
13: **Procedure** OVERLAP-PARTITION(Graph G, Vertex Cut $\mathcal{S}$)
14: $\mathcal{G} \leftarrow \emptyset$;
15: $G' \leftarrow G[V(G) \setminus S]$;
16: **for all** connected component G'_i in G' **do**
17: $\quad \mathcal{G} \leftarrow \mathcal{G} \cup \{G[V(G'_i) \cup \mathcal{S}]\}$;
18: **return** $\mathcal{G}$;

2.3.1. 找出图中所有k-core，删除不在k-core内的所有点

通过寻找k-core将图划分为若干部分，对每一部分单独处理，寻找k-VCC(对应上图算法中第2、3行)
方法：类似拓扑排序，不断删去度最小的点，直到当前度最小的顶点度数$>= k$

Algorithm 1: *CoreDecomposition*$(G(V, E))$

```
c ← 0;
while V ≠ ∅ do
    u ← arg min_{v∈V} deg(v);
    c ← max(deg(u), c);
    core(u) ← c, V ← V \ u;
    foreach v ∈ N(u) do deg(v) ← deg(v) − 1 ;
return core(u) for all u;
```

2.3.2. 分治求解k-VCC

不断找 $< k$ 的全局最小割 s，把当前子图用 s 分成若干部分(称为overlap-partition，如下图中灰色节点)，再对分开的子图做同样的处理，直到子图中最小割 $>= k$，则当前子图是一个k-VCC

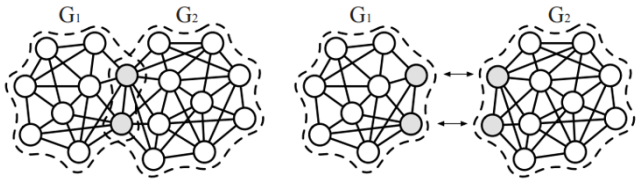


Fig. 2: An example of overlapped graph partition.

3. 算法优化

3.1. 优化1：Neighbor Sweep using Side-Vertex

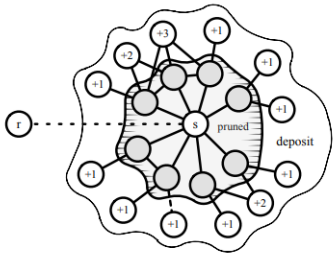
定义side-vertex: u 是 side-vertex 等价于不存在一个小于 k 的点割集 s 使得 $u \in s$

- 定理1: 如果$a \equiv b, b \equiv c$ 且 b 是side-vertex, 则 $a \equiv c$ ($a \equiv b$ 表示 a 和 b 之间不存在小于 k 的点割，或者说 a 和 b 之间有 k 条顶点不相交的路径)
- 定理2：如果 u 的任意两个邻居 v, v'满足$|N(v) \bigcap N(v')| >= k$ 或 $(v, v') \in E$ (E 为边集)，则 u 一定是 side-vertex 且称为 strong side-vertex(为side-vertex提供一个便于判定的充分条件)
- 定理3：如果$(v, w) \in E$，则一定有 $v \equiv w$

根据上述3定理可以得出以下优化：
(Neighbor Sweep Rule 1) 设 u 是源点，如果(1) $u \equiv v$;(2) v是strong side-vertex;(3) w 是 v 的邻居，则 w 可以不用检验(与 u 之间是否有 $< k$ 的割集)

3.2. 优化2：Neighbor Sweep using Vertex Deposit

- 定理4：u 是源点，如果 v 的邻居中有 $>= k$ 个顶点 w 满足 $u \equiv w$，则 $u \equiv v$
- 定义 $deposit(v)$ 表示 v 的邻居中满足 $u \equiv w$ 的顶点个数，则根据定理4，如果 $deposit(v) >= k$，则有$u \equiv v$



(a) 2-hop deposit

由此得到第2个优化策略：
(Neighbor Sweep Rule 2) 给定源点 u，如果 $deposit(v) >= k$，则 v 不用检验(与 u 之间是否有 $< k$ 的割集)
具体操作: 在优化1中清除strong side-vertex的邻居时，把源点的2-hop deposit数值+1

3.3. 具体优化算法

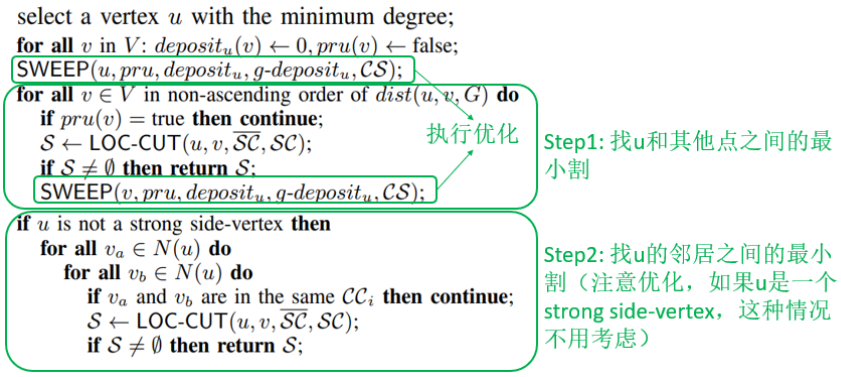
Algorithm 4 SWEEP$(v, pru, deposit_u, g\text{-}deposit_u, \mathcal{CS})$

1: $pru(v) \leftarrow$ true;
2: **for all** $w \in N(v)$ s.t. $pru(w) =$ false **do**
3: $deposit_u(w)$++;
4: **if** v is a strong side-vertex **or** $deposit_u(w) \geq k$ **then**
5: SWEEP$(w, pru, deposit_u, g\text{-}deposit_u, \mathcal{CS})$;

上图是基于上述两条优化的算法(不用关注最后2个参数)
给定源点u，当判断出$v \equiv u$，调用 `sweep(v, pru, deposit)`

(pru是一个初始化全false的bool型数组，记录每个点是否被访问过；deposit是一个初始化全0的int型数组，记录每个点的deposit值)

求解全局最小割的优化如下(忽略图中Step2有关CC的那一行)：



另外在overlap-partition步骤后，在原图 G 通过全局最小割 s 划分的子图 G_i 中，不必对G_i 中所有的点检查 strong side-vertex，只有同时满足以下两点的点才可能是 G_i 的 strong side-vertex：1) u 在 G 中是 strong side-vertex; 2) $N(u) \cap s \neq \emptyset$

4. 测试数据与输出

对于最小割部分的测试，请自行构造数据进行验证(小数据即可)

数据集：**注意：部分数据集中的边为有向边，使用时需要注意去重**

- DBLP
  http://snap.stanford.edu/data/com-DBLP.html
  建议测试 k=20,25,30,35,40
- web-Google
  http://snap.stanford.edu/data/web-Google.html
  建议测试 k=25,30,35,40
- web-stanford
  http://snap.stanford.edu/data/web-Stanford.html
  建议测试 k=40

参考结果如下：

Dataset	k-VCC num	Dataset	k-VCC num
DBLP 20	110	Google 25	33
DBLP 25	57	Google 30	11

Dataset	k-VCC num	Dataset	k-VCC num
DBLP 30	27	Google 35	3
DBLP 35	18	Google 40	3
DBLP 40	10	Stanford 40	11

输出时先输出kvcc个数，然后按照kvcc大小从小到大输出每个kvcc中的所有节点，节点编号也按照从小到大输出，可以参考下图(为DBLP k=40的部分结果)：

```
k = 40, KVCC num = 10
Node num :43
2488 7344 28215 32952 41537 43784 53062 56493 72133 75361 85941 103979 116360 195654 209954 214953 219452 219857
227877 242903 255491 262520 271908 271909 275900 289806 289807 303482 318399 318400 323353 323354 323355 323356
323357 323358 328299 328300 328301 328302 328303 328304 336906

Node num :44
34263 45634 99125 145932 206096 208720 212767 231527 256418 270521 301788 310618 314476 318279 323701 323702
323703 323793 344544 351271 361798 361799 361800 361801 361802 361803 361804 361805 361806 361847 361848 361849
361850 361851 361852 361853 361854 361855 361856 361857 361858 361859 362443 364863

Node num :44
34074 42596 89987 92793 326844 359180 359181 359182 359183 359184 359328 359329 359330 359331 359332 359333 359334
359450 359451 359452 359453 359801 359802 359803 359804 359805 359806 359807 359808 360076 360077 360078 360079
360080 360081 360082 360083 360084 360085 360086 374560 374743 393845 393846

Node num :48
18285 25313 29736 51518 63460 84521 97287 101816 109015 130348 130860 134547 137888 143086 143511 151124 156801
182017 188149 194892 217579 238983 257945 261254 264566 268236 270037 282859 282860 284646 291955 292522 321615
321616 326939 341019 362437 362438 362439 364439 373101 373102 373103 373104 375422 375423 375424 375425
```

5. 提交要求

- 所有源代码(一个文件夹)
  要求朴素算法(60分)中的三个Task各自需要提交一份代码；优化部分(40分)需提交一份完整的优化kvcc算法代码，即如果完整地完成了实验，需要提交至少4份代码
- 所有测试结果(一个文件夹)
  输出要求见 4.测试数据与输出
- 实验报告(pdf格式)
  需要包含以下内容：
  - 实现思路与算法细节讲解，必要时可以结合代码或论文原文，要求代码必须有注释
  - 不同数据集/k取值的运行时间
- **严禁抄袭！！！**