山东大学（威海）
SHANDONG UNIVERSITY , WEIHAI

运筹学与数学建模课程论文

班　　级：数学与统计学院 18 数据科学班

学生姓名　　学号

许函嘉　　201800820045

赵呈亮　　201800820179

日　　期：　2020 年 12 月 30 日

得　　分：

关于最小割问题及其算法的研究与改进

摘要

本文针对图论最小割问题，深刻研究了其定义、实际应用以及现有的解决该问题的工作。其次本文讨论并总结了最小割问题在不同类型图中的问题性质，判断其为 P 或是 NP 问题，同时给出了相应理由或证明。另外针对无向图中的最小割问题，本文利用 *Python* 复现了 *Karger* 算法和 *Stoer-Wagner* 算法，并利用所给测试图验证了算法结果的正确性。最后本文进一步研究了这两个算法在适用范围和运算速度上的推广和改进，并提出了自己对算法的改进方案。

关键词：图论 最小割 NP 问题 Karger 算法 Stoer-Wagner 算法 改进优化

1. 最小割问题简介

1.1. 问题阐述

本文研究最小割问题。给定一个有n个顶点和m条(可能是加权的)边的无向图，我们希望将这些顶点划分成两个非空集合，以便最小化它们之间交叉的边的数量(或总权重)。更正式地说，图G的割$(A，B)$是将G的顶点划分成两个非空集A和B。如果v和w中的一个在A中，另一个在B中，则边$(v，w)$与切口$(A，B)$相交。切割的值是穿过切割的边的数量，或者在加权图中，是穿过切割的边的权重之和。最小割问题是求最小值的割。

在本文中，假设该图是连通的，否则问题就不具意义。我们还要求所有的边权重都是非负的，否则问题是由最大割问题的普通转换。另外我们将最小割问题与$s-t$最小割问题区分开来，在$s-t$最小割问题中，我们要求两个指定的顶点s和t位于割的相对两侧；在最小割问题中则没有这样的限制。特别是在未加权的图上，解决最小割问题有时被称为寻找图的连通性，即确定断开图必须移除的最小边数(或最小总边权重)。

具体假设以及不同情况将在问题一的求解中进一步详细说明。

1.2. 问题应用

最小割问题有许多应用，本文在此调查了主要一些应用场景。

1.2.1. 网络可靠性

确定网络连通性的问题经常出现在网络设计和网络可靠性的问题中[*Colbourn 1987*]。在具有随机边缘故障的网络中，网络最有可能以最小割被划分。例如，考虑一个无向图，其中每条边都以某种概率p断开，假设我们希望确定该图断开的概率。让f_k表示大小为k的边集的数目，去掉这些边会断开图。那么图断开概率为$f_k p_k (1-p)^{m-k}$如果p非常小，那么可以通过仅考虑k的值来精确地近似该值。因此，列举所有最小割变得很重要，甚至需要列举所有几乎最小割。在最近的应用中，这种枚举被用于全终端网络可靠性问题的完全多项式时间近似方案中。

1.2.2. 信息检索

在信息检索中，最小割已被用来识别超文本系统中的局部相关文档簇[*Botafogo 1993*]。如果超文本集合中的链接被视为图中的边，那么小的剪切对应于文档组，这些文档组之间几乎没有链接，因此可能是不相关的。

最小割问题出现在并行语言编译器的设计中。考虑一个我们试图在分布式内存机器上执行的并行程序。在该程序的对齐分布图中，顶点对应于程序操作，边对应于程序操作之间的数据流。当程序操作分布在处理器中时，连接不同处理器上节点的边被“切割”。这些切割边缘不好，因为它们表明需要处理器间通信。寻找程序操作的最佳布局需要在校准分布图中重复解决最小切割问题。

1.2.3. 组合优化问题

最小割问题在大规模组合优化中也起着重要的作用。目前，寻找大型旅行推销员问题精确解的最佳方法是基于切割平面的技术。切割平面算法通过反复生成线性不等式来寻找最优路径，这些不等式会切割掉多面体的不需要的部分，直到只剩下最优路径。最有用的不等式是由 *Dantzig* 等人首先提出的次优消去约束。识别子图消除约束的问题可以重新表述为在具有实值边权重的图中寻找最小割的问题。因此，旅行商问题的切割平面算法必须解决大量的最小切割问题。 *Padberg and Rinaldi* [1990]说，最小切割问题的解决方案是他们最先进的基于切割平面的算法的计算瓶颈。他们还报告说，最小割问题是许多其他基于割平面的组合问题算法的瓶颈，通过这些组合问题的解产生连通图。*Applegate* 等人[1995]进行了类似的观察，并指出一种可以寻找几乎所有的最小切割的算法会对组合优化问题产生巨大的帮助。

1.3. 相关工作

调查了关于最小割问题的几类主流算法，并简述了它们被提出时的算法原理。

1.3.1. 基于流程的方法

寻找最小割的第一个算法利用了$s-t$最小割和$s-t$最大流之间的对偶性。因为$s-t$最大流包含了每个$s-t$最小割，所以在给定最大流的情况下，很容易找到$s-t$最小割——例如，在最大流的剩余图中，从源可到达的所有顶点的集合形成了这种 s-t 最小割的一边。因此，$s-t$最大流算法可用于寻找$s-t$最小切割，并且最小化s和t的所有$\binom{n}{2}$个可能选择产生最小切割。*Gomory* 和 *Hu* [1961]提出了流等价树的概念，并观察到最小割集只能通过求解$n-1$个最大流问题来得到。

1.3.2. 无流量切割

Gabow [1995]开发的一种方法是基于最小割问题的拟阵特征。根据这个特征，图中的最小割等于图中可以找到的不相交有向生成树的最大数量。*Gabow* 的算法通过寻找这样一个最大的树填充来寻找最小割。虽然没有使用流，但是树是通过一系列增加的路径计算来构建的。*Gabow* 的算法不是直接计算最小割，而是计算一个类似流的结构，该结构包含了图的最小割。其中心思想是重复识别和收缩不在最小切割中的边，直到最小切割变得明显。它只适用于无向图，但它们可能是加权的。

1.3.3. 并行算法

最小割问题的并行算法也进行了探索，对于无向未加权图，*Khuller* 和 *Schieber* [1991]给出了一个算法，该算法使用cn^2个处理器在$O(c)$时间内找到c值的最小割。

2. 最小割/全局最小割问题定义及分类

2.1 问题假设

在本题中我们作出以下假设：

(1) 假设以下讨论中所提及的图皆为平面图或可以转化为平面图。

(2) 假设该图的顶点不存在权重。

(3) 假设所给图各边权重非负。

2.2 问题分析

关于 *Min-cut* 是否为 P 或 NP 问题的判定，通常需要考虑以下几种条件变化：

(1) 有终端或无终端

(2) 有向图或无向图

(3) 终端数（有终端时）或划分集合数（无终端时）k

(4) 有边权或无边权

当图无边权时，最小割问题等价于图的边缘连通性问题，即求目标割集中边的数量，因此可以将无边权情况等价地看作边权为“1”的有边权情况，即条件(4)可不做讨论。

当所给图无终端时，即为无向图，因此此时 *Min-cut* 问题通常指 *Global min-cut* 问题，因此在以下讨论中，对第一小问 *Min-cut* 的讨论将以有终端图为基础，对第二小问 *Global Min-cut* 的讨论将以无终端图且为无向图为基础。

综上所述，第一小问对 *Min-cut* 将讨论条件（2）和条件（3）的组合情况。第二小问 *Global Min-cut* 将讨论条件（3）的情况。

2.3. *N*或 *NP*问题的判定原则

在判定为 *P* 问题时，我们认为若现存一种或多种能在多项式时间内解决该问题的算法，即可说明其为 *P* 问题。反之若不存在这样的算法，却存在能在多项式时间内验证得出一个正确解的例子，则认为它暂时为 *NP* 问题。更复杂的情况如 *NP-hard*、*NP-complete* 则参考已有文献对问题类型进一步判定。在此则不对 *P*、*NP*、*NP-hard*、*NP-complete* 等定义进行赘述。

2.4. 最小割问题的定义和分类

2.4.1. 问题定义

给定一个**平面**图$G = (V, E)$，其中每条边都具有**非负权重**（容量）c_e，以及一个包含k ($k \in \{2, 3, \ldots, |V|\}$)个的顶点的子集$T \subset V$（T 中的点称为终端 Terminal）。最小割问题为寻找到一组边集$C \subset E$，使得在图 $G = (V, E - C)$ 中T内任意不同两点之间没有边相连，并使得 C 中边权和最小，即找到$\sum_{e \in C} c_e$ 的最小值。

2.4.2. 问题分类

(1) 终端数 *k=2*（特殊情形 *s-t cut*）时：

① 有向图时（此时终端一般分别称为“源节点”和“汇节点”）：

结论：P问题

理由：此情况时最小割问题可以转化为图最大流问题。在定向加权流动网络中），最小切口将源顶点和汇点顶点分开，并使从切口源端指向切口汇点侧的边上的总权重最小。根据最大流最小割定理，此割的权重等于在给定网络中可以从源发送到汇的最大流量。最大流问题可以在多项式时间内解决[1]，因此此情况下最小割问题为 P 问题。

特别地：当最大流问题具有析取约束时，问题的性质会变得更复杂。当问题具有负析取约束时，即表示某对边不能同时具有非零流量，此时即使对于简单的网络，问题会变为强的 NP-hard 问题。当问题具有正析取约束时，即在固定的一对边中至少有一个必须具有非零流量，此时如果允许边具有分数流，则问题是 P 问题，但当流量必须为整数时，问题可能是强的 NP-hard 问题。

② 无向图时：

结论：P问题

理由：此情况时最小割问题可以转换为构造图的最小割树（*Gomory-Hu tree*）问题。构造最小割树的原理为：在当前点集随意选取两个点u,v，在原图上跑出他们之间的最小割，然后就在树上连一条从 u 到 v, 权值为$\lambda(u,v)$的边。然后找出u,v分属的两个点集，对这两个点集递归进行操作。当点集中的点只剩一个的时候停止递归。由最小割树定义可知，树上去掉$(s-t)$后产生的两个集合恰好是原图上$(s-t)$的最小割把原图分成的两个集合，且构造最小割树的时间复杂度为$O(n^3m)$，因此此情况下最小割问题为 P 问题。

(2) 终端数 $k>2$（一般情形）时：

① 有向图时（此时终端一般分别称为“源节点”和“汇节点”）：

结论：*NP-hard* 问题

理由：*J. Naor* 和 *L. Zosin* 的研究中[2]指出此情况下的最小割问题为 *NP-hard* 且 *max SNP-hard* 问题。同时给出了时间近似比为 2 的多项式时间近似算法。

② 无向图时：

结论：*NP-complete* 问题

理由：此情况可以看作第二小问中划分集合数 $k>2$ 的特殊情况，在其基础上要求 k 个互不相交的集合将 k 个给定顶点一一划分开来。在 *Huzur Saran* 和 *Vijay V. Vazirani* 的研究中[3]，指出此情况下最小割问题无法在多项式时间内

求解，为 *NPC* 问题。*Saran* 和 *Vazirani*[4]对此情况基于线性规划方法给出了一个时间近似比为$(2-2/k)$的多项式时间近似算法。

2.5. 全局最小割问题的定义和分类

2.5.1. 问题定义

给定一个**无向平面**图$G=(V,E)$，其中每条边都具有**非负权重**（容量）c_e，以及一个正整数k（$k\in\{2,3,\dots,|V|\}$）。最小割问题为寻找到一组边集$C\subset E$，使得V被划分为不相交的k个非空集合，并使得C中边权和最小，即找到$\sum_{e\in C}c_e$的最小值。

2.5.2. 问题分类

(1) 划分集合数 *k=2*（特殊情形 *S-T cut*）时：

结论：*P* 问题

理由：

1. 有权图中，可通过 *Stoer-Wagner* 算法在多项式时间内求解，其时间复杂度为$O(mn+n^2logn)$）。算法基本思想：通过合并最密集的顶点来缩小图的规模，直到图仅包含两个组合顶点集。在每一阶段都会找到一个$s-t$最小割，其中$s-t$两点任意，最终全局最小割即为最小权重的一次割。
2. 无权图中，可特别地通过 *Karger-stein* 算法在多项式时间内求解，其时间复杂度为$O(n^2ln^3n)$。算法基本思想：在图中随机取一条边，将边的两个端点合并，同时消除由于合并而形成自环的边，然后重复此步骤直到图中仅剩两个点，将最终两点之间的边作为目标最小割。这样一次运算的复杂度为$O(m)$，这样随机的过程返回的结果是不确定的，找到的割并不一定是最小的，事实上可以证明，一次运行找到最小割的概率最低为$1/(n/2)$，那么，将上述算法重复执行$(n/2)lnn$次，我们可以以低于的$1/n$的失败概率获得最小割。

(2) 划分集合数 *k>2*（一般情形）时：

结论：若 *k* 值固定，则为 *P* 问题；若 *k* 值不固定且作为输入值，则为 *NP-complete* 问题

理由：

1. *Goldschmidt* 和 *Hochbaum* 在论文中[4]证明了当 *k* 固定时，它可以在多项式时间内被解决，其算法时间复杂度为$O(|V|^{k^2})$。

2. 同时论文里指出，当k不固定（作为输入值）时，它是一个 *NP-complete* 问题。无法证明存在多项式时间求解的算法，但存在可在多项式时间内解出最小割的例子[4]。 *Saran* 和 *Vazirani* [5]对此情况基于线性规划方法给出了一个时间近似比为$(2-2/k)$的多项式时间近似算法。

3. Karger 算法实现

3.1. 算法介绍

Karger 算法是一种用于计算无向无权连通图的最小割的随机算法。它由 David Karger 发明，于 1993 年首次提出。

该算法的思想基于边收缩的概念，在无向图$G=(V,E)$中，边收缩的过程合并了节点u和v，将图的节点总数减少一。所有u和v的相邻点被“重新连接”到合并的节点，从而有效地产生了多重图。Karger 的基本算法迭代地收缩随机选择的边缘，直到只剩下两个节点为止。这些节点代表原始图中的切割。通过对该基本算法进行足够多次的迭代，可以高概率找到最小剪切。 所以通过重复收缩算法，我们能以极大的概率找到图的全局最小割。

3.1.1. 算法伪代码

Karger Algorithm :

Input：无向无权图 G

Output： 1.图 G 的全局最小割

2.最小割值

Solution ： 初始化 最小割$\leftarrow \emptyset$ 最小割值$\leftarrow +\infty$

For 每一次循环：

Do 1. 当前割、割值 $\leftarrow$ Contraction(G)

2. If 当前割值 $<$ 最小割值

最小割值 $\leftarrow$ 当前割值

最小割 $\leftarrow$ 当前割

Return 最小割、最小割值

其中 收缩算法的流程如下

3.1.2. 收缩算法伪代码

Function Contraction(G):

Input： 无向无权图$G = (V, E)$

Output： 1.图G一种割

2.割值

Solution ： While $|V| > 2$

Do 1. 随机在G的边E中选取边(u, v)

2. 收缩边(u, v) 合并点u, v

添加新点w 代替u, v

保留点u, v的边，将边的端点更新为w

保留平行边，删除 Self-Loop

$V = \{v_1, v_2\}$

Return 所有被收缩至v_1, v_2的端点、v_1, v_2间平行边个数

3.2. python 实现过程

运行环境 ： 1.python3

2.NetworkX（仅使用基础功能）

因为使用 NetworkX 库绘图时 无法绘制平行边，所以在本程序中，为了方便画图表示。用图中边的权重代表平行边数 点的名称为字符串表示，方便融合时新点的命名与最后查看割集的划分。

因为使用边的权重代替平行边数，所以程序中的图中的边数将少于实际边数。但这不会对最终结果产生影响。 因为在 Karger 算法中，若随机选取两点平行边中的一条进行收缩，则这两点间的其余平行边就会变为Self-loop 也将被去除，而其他边不会受到影响。所以两点之间的平行边若任意一条被选中，其结果都是这两点间的所有边都被去除，而在程序中，若选择到两点之间一条带权重的边，那么这条边也会被去除。所以这两者的效果都为两点之间的所有连接都被去除，并且与其他的连接不变。所以用权重值代替平行边个数并不会影响结果。

3.2.1. 辅助函数 1 read_txt_to_graph(path)

功能： 读取表示图 txt 文件，将其转化为 NX 包的图对象。其中点的表示形式为字符串。其中所需 txt 文件格式：每行代表一条边,两个数字为边的两个端点

输入： 文件路径

返回： NetworkX 图对象

代码如下：

```
def read_txt_to_graph(path):
   # 创建图
    G=nx.Graph()
    with open(path, "r") as f:
        data = f.readlines()

    edges=[]
    for items in data:
        # 处理字符串
        point=items.split(" ")
        edges.append((point[0],point[1].replace("\n",""),1))

    #在图中添加边
    G.add_weighted_edges_from(edges)
    print("文件导入完成")
```

3.2.2. 辅助函数 2 Merge(G,node1,node2):

功能： 合并给定的两点

具体处理方式如下：

1. 添加新点 代替要合并的两点
2. 将新点与两点的相邻点连接，权值不变 若为公共相邻点，权值相加。忽略两待合并点之间的边（保证无 Self-loop）
3. 在 G 中将 node1 node2 删除

输入 1： 图 G

输入 2： 待合并点 node1

输入 3： 待合并点 node1

返回： 点 1、2 合并后的图 G

代码如下：

```
def Merge(G,node1,node2):
    # 将 两个点合并后新的点 加入图中 名称继承自两点的名称
    new_node = str(node1) + "_" + str(node2) #字符串相连
    G.add_node(new_node)
    # 在node1的相邻点中遍历
    for w, e in G[node1].items():
        #忽略 node1 与 node2 之间的连接
        if w != node2:
            if w not in G[node2]:      #非公共相邻点，添加新边 权值不变
                G.add_edge(new_node, w, weight=e["weight"])
            else:                      #公共相邻点 ，添加新边
                G.add_edge(new_node, w, weight=e["weight"])
    # 在node2 的相邻点中便利
    for w, e in G[node2].items():
        if w != node1:
            if w not in G[node1]:  #
                G.add_edge(new_node, w, weight=e["weight"])
            else:   #对于公共相邻点，权值相加
                G[new_node][w]["weight"] += e["weight"]

    G.remove_nodes_from([node1, node2])
    return G
```

3.2.3. 辅助函数 3 Contraction(G)：

此函数为函数3.1.2的python实现

功能：求图G的一种割

输入：图G

返回：该种割及割值

代码如下：

```
def Contraction(G):
    #新建一个图对象进行操作
    new=G.copy()
    while new.number_of_nodes()>2:
        #随机选取一条边
        u,v= random.choice(list(new.edges()))
        #合并这条边的两段点
        new = Merge(new,u,v)
    cut_value=list(new.edges(data=True))[0][-1]["weight"]
    node_list=list(new.nodes())
    cut_set=   {" 子 点 集 1":node_list[0].split("_")," 子 点 集
    2":node_list[1].split("_")}#得到割集 点的划分（以字典的形式）
    #仅有两个点，这两个点的名称中含有割集的划分
    return cut_value,cut_set
```

3.2.3. 算法函数

功能：实现 Karger 算法求无向无权图的 最小割 及最小割值

输入：图 G:

返回：最小割值及割集

```
def Karger(G):
    iter=100# 迭代次数
    min_cut=10000
    min_cut_set={}
    for i in range(iter):
        cur_cut,cur_cut_set=Contraction(G)
        #print(c)
        if cur_cut< min_cut:
            min_cut=cur_cut
            min_cut_set=cur_cut_set
    print("图的最小割为")
    print(min_cut_set)
    print("求得最小割值为：%i" % min_cut)
    return min_cut_set, min_cut
```

通过构建相关辅助函数，我们最终使用 python 实现了 Karger 算法

4. Stoer-wagner 算法实现

4.1. 算法介绍

在图论中，*Stoer-Wagner* 算法是一种递归算法，用于解决具有非负权重的无向加权图中的最小割问题。它是由 *Mechthild Stoer* 和 *Frank Wagner* 于 1995 年提出的。该算法的基本思想是通过合并的顶点来缩小图，在每个阶段，算法都会找到包含点s,t的最小割，然后算法合并了顶点s,t。直到图仅包含两个点。

Stoer-Wagner 算法基于下面的事实对于图中的任意两点 s,t要么属于一个集合，要么分属于两个集合。如果是前者，我们在这一次最小割的计算就是最优解了，如果是后者，不难得到的是合并这两个点不影响答案。

4.1.1. 求阶段最小割

MinimumCutPhase(G, a)：

Input：非负权重连通图$G = (V, E)$ 起始点a

Output： 1.图G的$s-t$最小割

2.点s,t

Solution ： 初始化 点集$A \leftarrow \{a\}$

While $A \neq V$：

Do 1. 将与点集A连接最紧的点加入A

End

保存当前割 与 割值

合并两个最后 2 个加入A的点s,t

Return 割集$(V-\{t\},\{t\})$, 割值

4.1.2. Stoer-Wagner 算法

Stoer-Wagner Algorithm (G) :

Input：非负权重连通图$G = (V, E)$ 起始点a

Output： 1.图G的$s-t$最小割

2.点s, t

Solution ： While $|V| > 2$：

Do 1.当前割、割值 ⟵ **MinimumCutPhase(G, a)**

2. If 当前割值 ＜ 最小割值

最小割值 ⟵ 当前割值

最小割 ⟵ 当前割

End

Return 最小割、最小割值

其中，在每一次循环中，起始点 a 可以任意选取。

4.2. Python 实现

4.2.1. 辅助函数 1 read_txt_to_graph(path)

同 3.2.1

4.2.2. 辅助函数 2 Merge(G,node1,node2)：

功能：合并给定的两点

具体处理方式如下：

1. 添加新点 代替要合并的两点
2. 将新点与两点的相邻点连接，权值不变 若为公共相邻点，权值相加。忽略两待合并点之间的边（保证无 Self-loop）
3. 在 G 中将 node1 node2 删除

输入 1： 图 G

输入 2： 待合并点 node1

输入 3： 待合并点 node1

返回： 点 1、2 合并后的图 G

代码如下：

```
def Merge(G,node1,node2):
    # 将 两个点合并后新的点 加入图中 名称继承自两点的名称
    new_node = str(node1) + "_" + str(node2) #字符串相连
    G.add_node(new_node)
    # 在node1的相邻点中遍历
    for w, e in G[node1].items():
        #忽略 node1 与 node2 之间的连接
        if w != node2:
            if w not in G[node2]:      #非公共相邻点，添加新边 权值不变
                G.add_edge(new_node, w, weight=e["weight"])
            else:                      #公共相邻点 ，添加新边
                G.add_edge(new_node, w, weight=e["weight"])
    # 在node2 的相邻点中便利
    for w, e in G[node2].items():
        if w != node1:
            if w not in G[node1]:  #
                G.add_edge(new_node, w, weight=e["weight"])
            else:   #对于公共相邻点，权值相加
                G[new_node][w]["weight"] += e["weight"]

    G.remove_nodes_from([node1, node2])
    return G
```

4.2.3. **辅助函数** 3 MinimumCutPhase(G)：

此函数复现参考文献中的伪代码

功能：求一种$s-t$最小割

输入：图G

返回：$s-t$最小割值，及点s,t

代码如下：

```
def MinimumCutPhase(G):

    node_list = list(G.nodes())
    PQ={}
    for node in node_list:
        PQ[node]=0
    s=None
    t=None
    while PQ!={}:
        max_key = max(zip(PQ.values(), PQ.keys()))
        u=max_key[1]
        del [PQ[u]]
        s=t
        t=u
        for v,e in G[u].items():
            if v in PQ.keys():
                weight=PQ[v]+e["weight"]
                PQ.update({v:weight})
    cut = 0
    for _, edge in G[t].items():
        cut += edge["weight"]
    return cut,s,t
```

3.2.4. 算法函数 Stoer_Wagner

功能：实现 *Karger* 算法求非负权重无向图的 最小割 及最小割值

输入：图 G:

返回：最小割值 及割集:

```
def Stoer_Wagner(G):

    min_cut = 1000000
    phase = 1
    last_node=None
    nodes=list(G.nodes())
    new_G=G.copy()# 复制一份图G用于之后操作（不改变图G本身）


    while True:
        print("在第%i阶段"% phase)
        cut, s, t = MinimumCutPhase(new_G)
        new_G = Merge( new_G, s, t)

        #判断图中仅有两个点时退出循环
        if len(new_G.nodes()) ==2:
            break
        phase += 1
        print("合并后图中点的个数为：%i" % nx.number_of_nodes(new_G))

        if cut < min_cut:
            min_cut = cut
            last_node=t
    partition1=last_node.split("_")
    partition2=list(set(nodes) - set(partition1))
    cut_set= {"子点集1": partition1,"子点集2": partition2}
    print("图的最小割的值为: %i" % min_cut)
    print("图的最小割为")
    print(cut_set)
    return cut_set,cut
```

5. 数据测试

运行环境：python3.7、Anaconda3、NetworkX2.5

硬件环境：CPU：6-Core Intel Core i7

5.1. 数据 1

文件名称：Example.txt

图中点数：10

图中边数：23

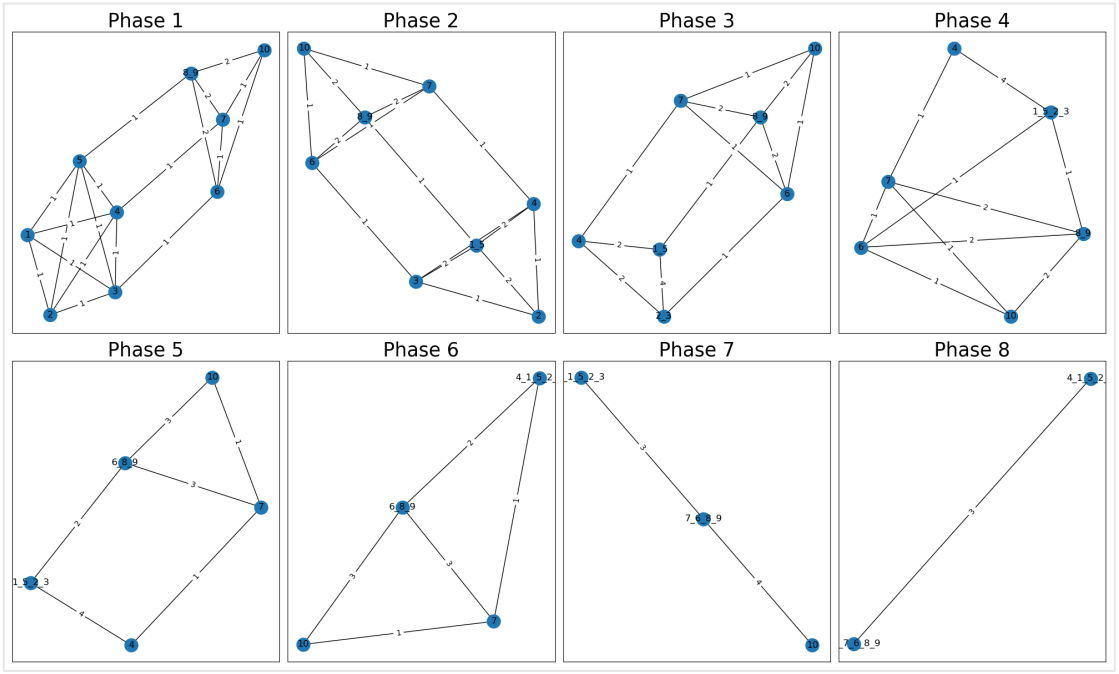
5.1.1. Karger 算法结果

最小割值：3

最小割：{1,2,3,4,5},{6,7,8,9,10}(点编号)

100 次循环运行时间：22.124958038330078 s

下图为一次循环的过程：



其中，边的权值代表两端点间平行边个数。最终分割如下图。

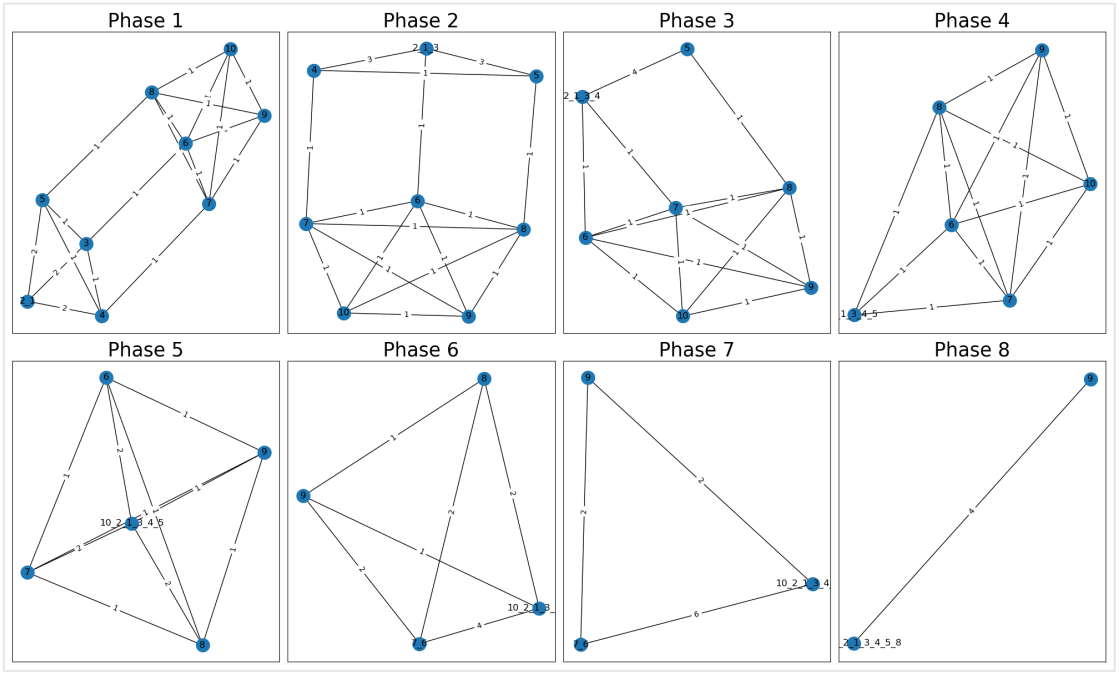
5.1.1. Stoer-Wagner 算法结果

最小割值：3

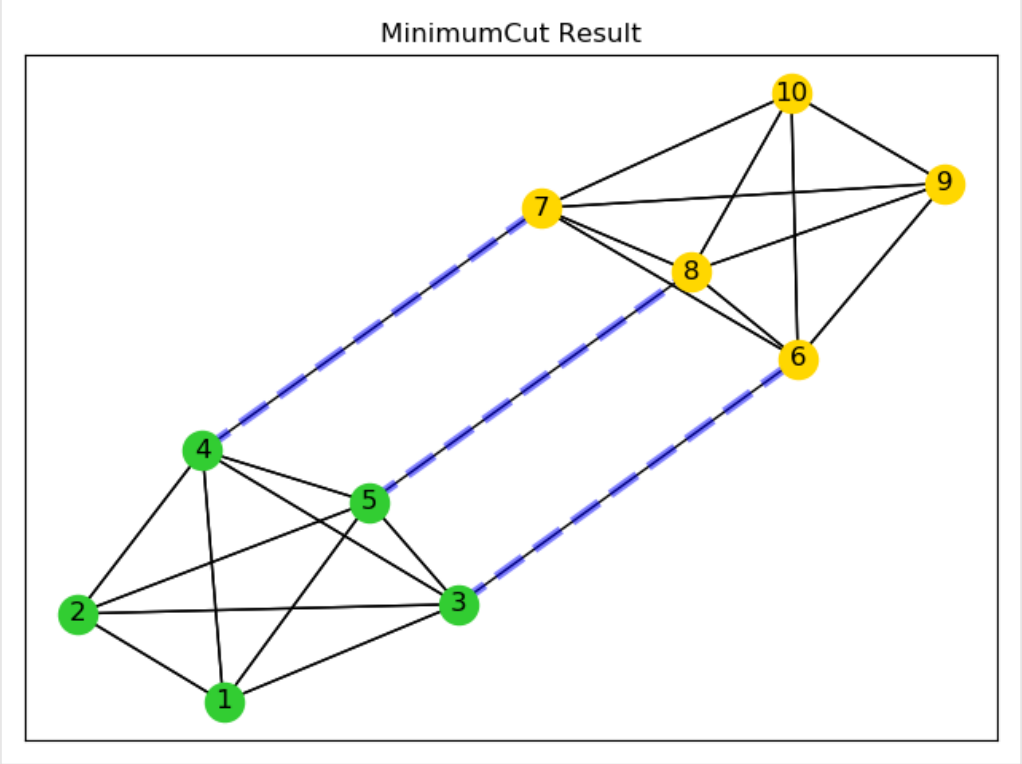
最小割：{1,2,3,4,5},{6,7,8,9,10}(点编号)

运行时间：0.0005450248718261719 s

其中，*Stoer-Wagner* 算法中每一步 *MinimumCutPhase* 的结果如下图所示：



Karger 算法与 Stoer-Wagner 算法的出相同结果



5.2. 数据 2

文件名称：Crime_Gcc.txt

图中点数：754

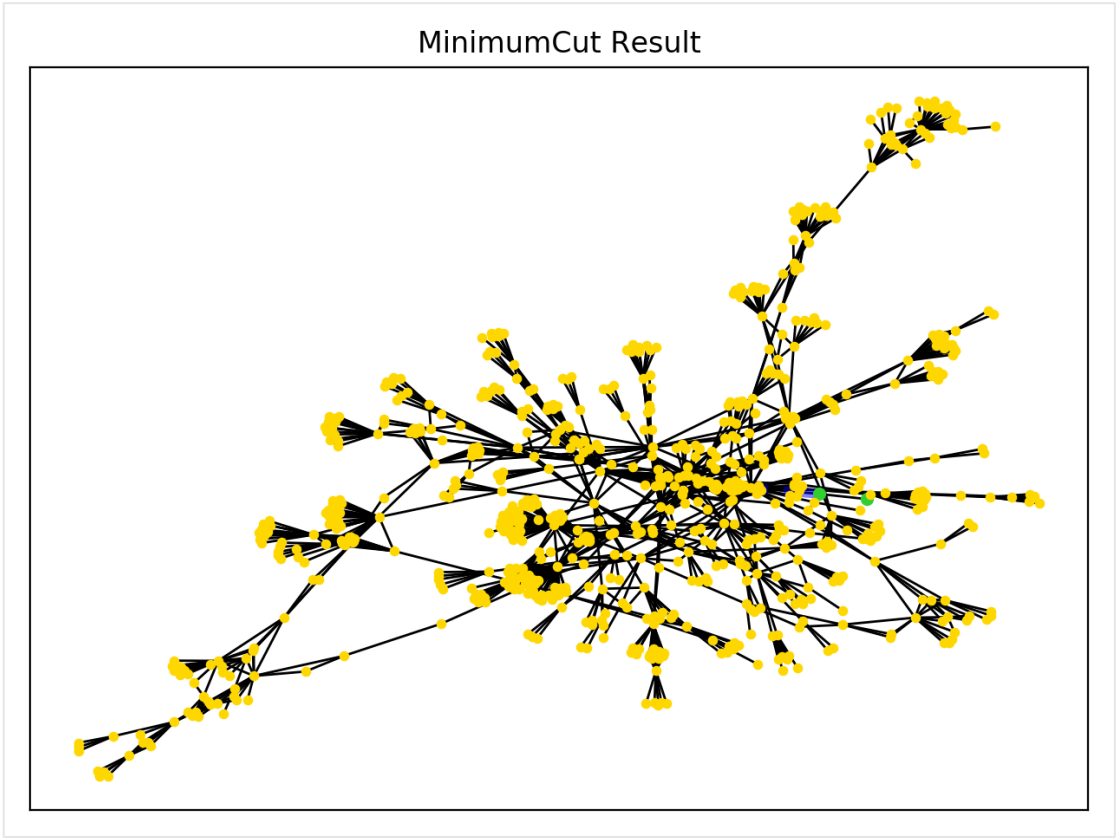
图中边数：2127

5.2.1. Karger 算法结果

最小割值：1

最小割：{180，269},{其余点}

100 次循环运行时间：24.582146883010864 s

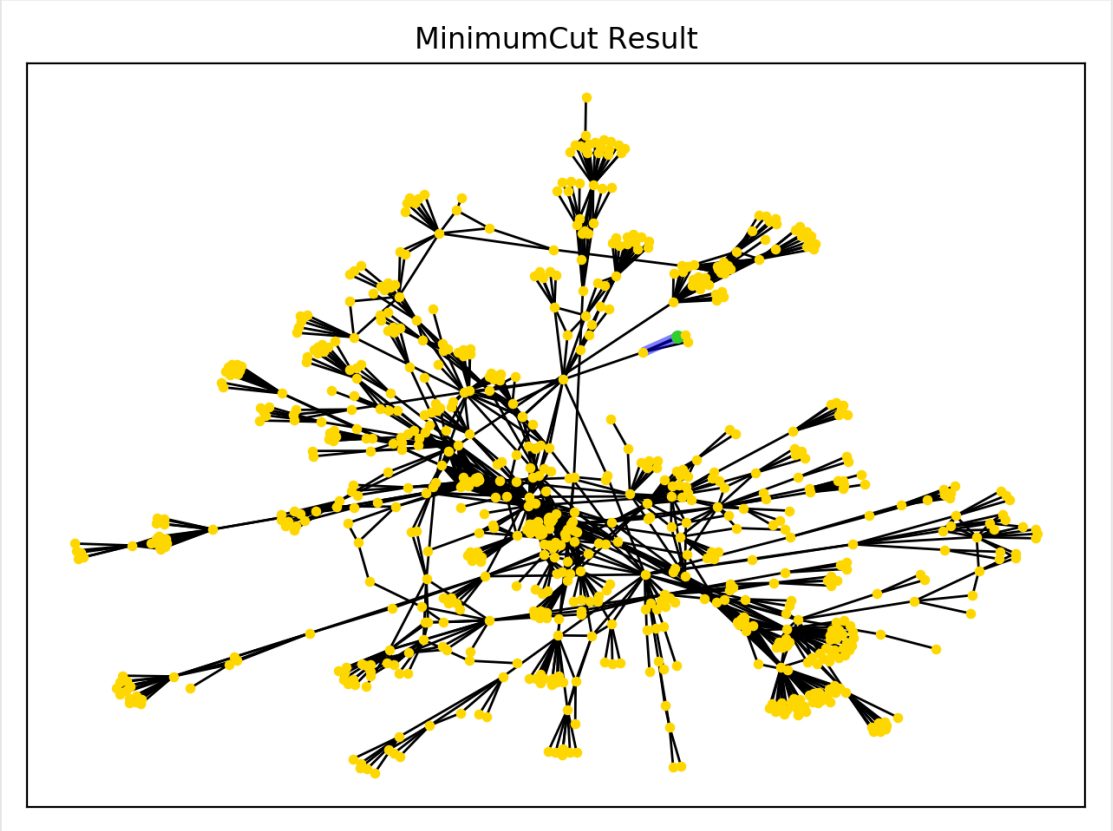


5.2.1. Stoer-Wagner 算法结果

最小割值：1

最小割：{629},{其余点}

运行时间：7.167319059371948 s



5.3. 数据 3

文件名称：BenchmarkNetwork.txt

图中点数：83

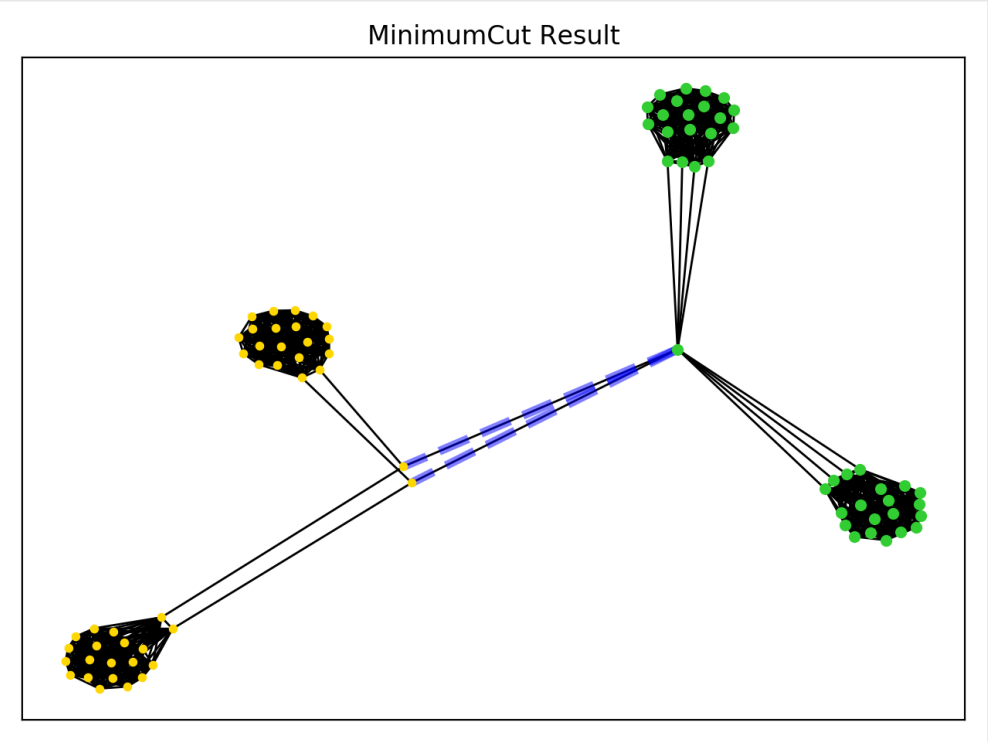
图中边数：774

5.3.1. Karger 算法结果

最小割值：2

最小割：{81, 24, 34, 22, 29, 39, 26, 28, 27, 38, 36, 40, 47, 31,32, 21, 25, 23, 35, 30, 33, 14, 12, 1, 3, 5, 19, 16, 2, 8, 9, 15, 11, 17, 10, 13, 6, 7, 20, 4, 18]},{其余点}

100 次循环运行时间：1.2072358131408691 s

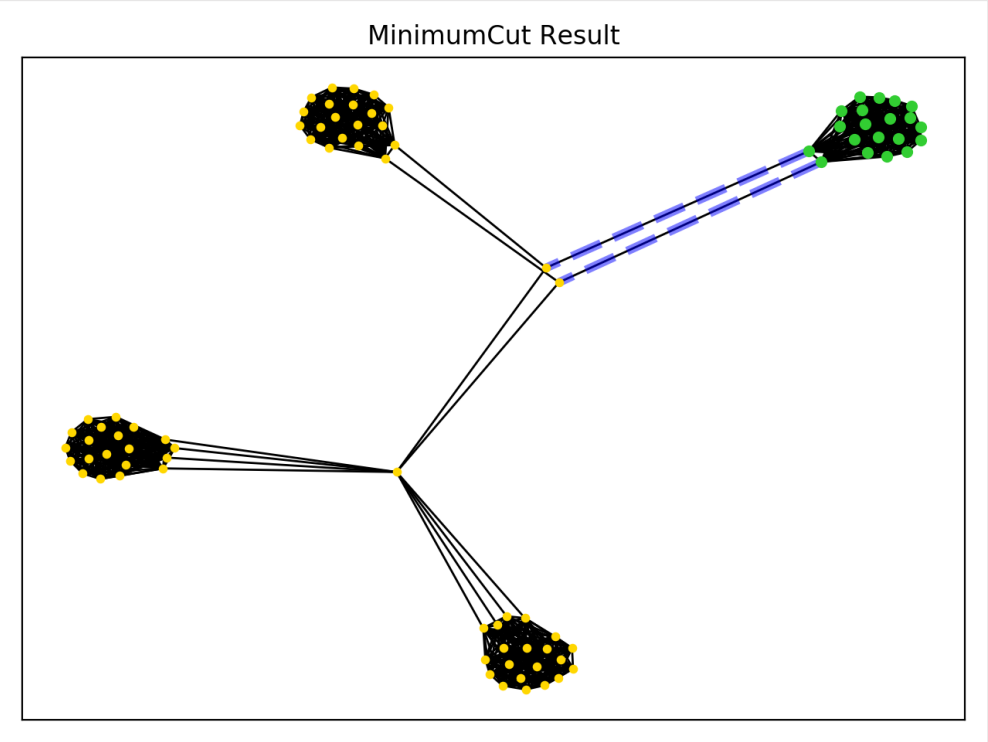


5.3.2. Stoer-Wagner 算法结果

最小割值：2

最小割：{58, 47, 46, 45, 44, 43, 41, 42, 60, 59, 57, 56, 55, 54, 53, 52, 51, 50, 49, 48},{其余点}

运行时间：0.049189090728759766 s



5.4. 数据 4

文件名称：Corruption_Gcc.txt

图中点数：307

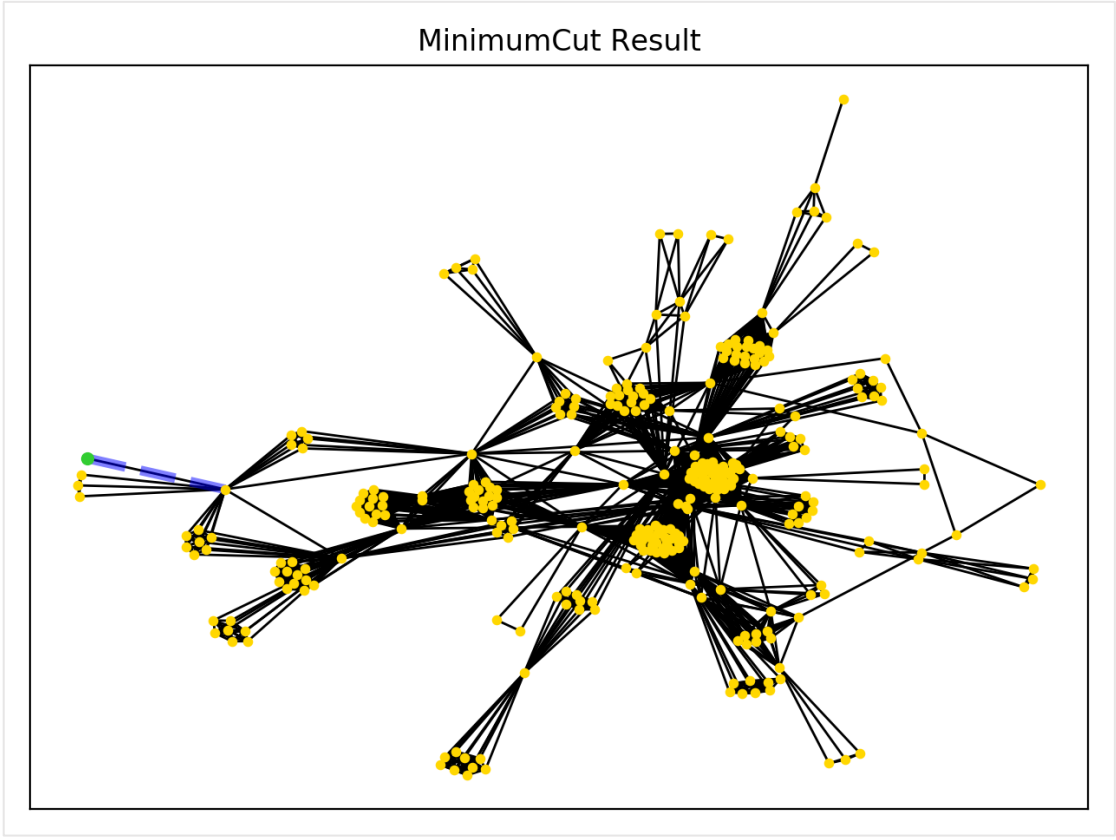
图中边数：3281

5.4.1. Karger 算法结果

最小割值：1

最小割：{309},{其余点}

100 次循环运行时间：10.05153775215149 s

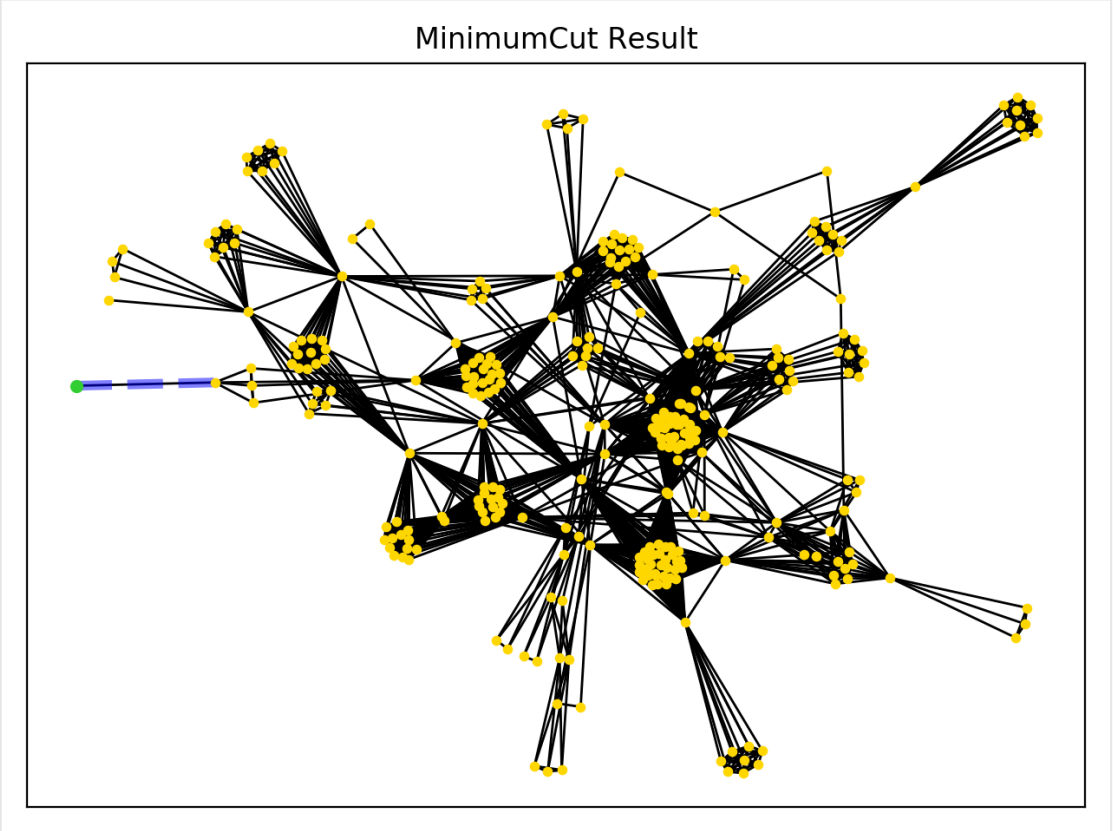


5.4.2. Stoer-Wagner 算法结果

最小割值：2

最小割：{19},{其余点}

运行时间：1.2400200366973877 s



5.5. 数据 5

文件名称：PPI_gcc.txt

图中点数：2224

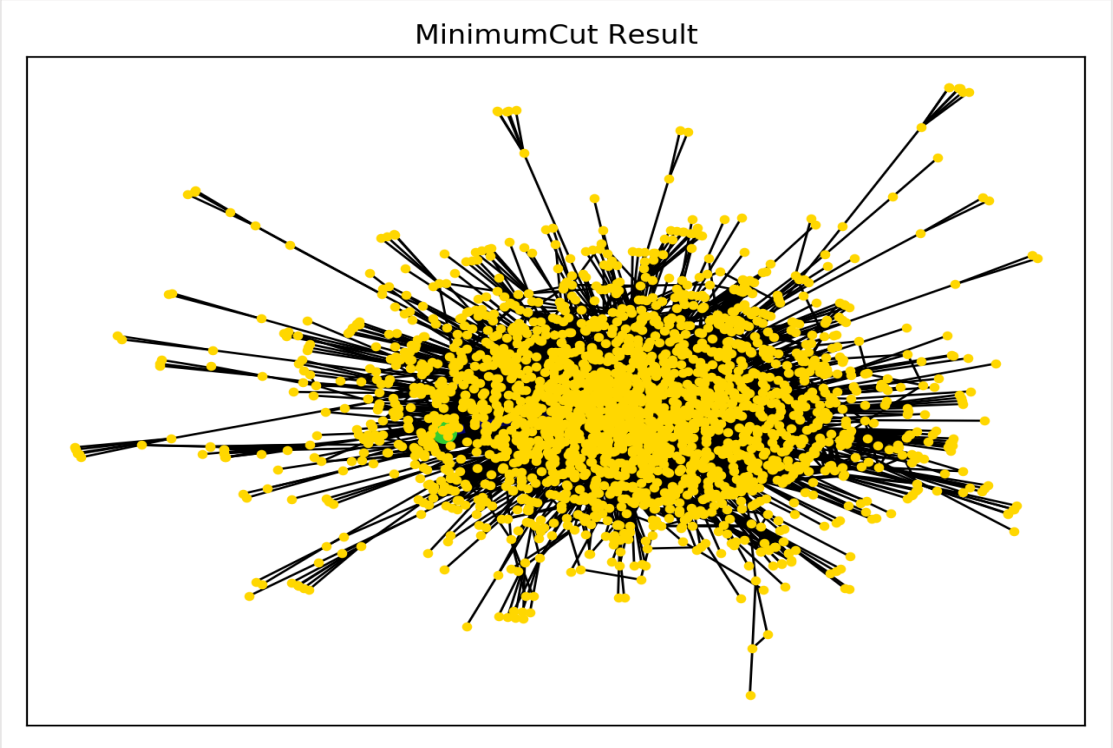
图中边数：6609

5.5.1. Karger 算法结果

最小割值：1

最小割：{2105},{其余点}

10 次循环运行时间：26.133662939071655 s

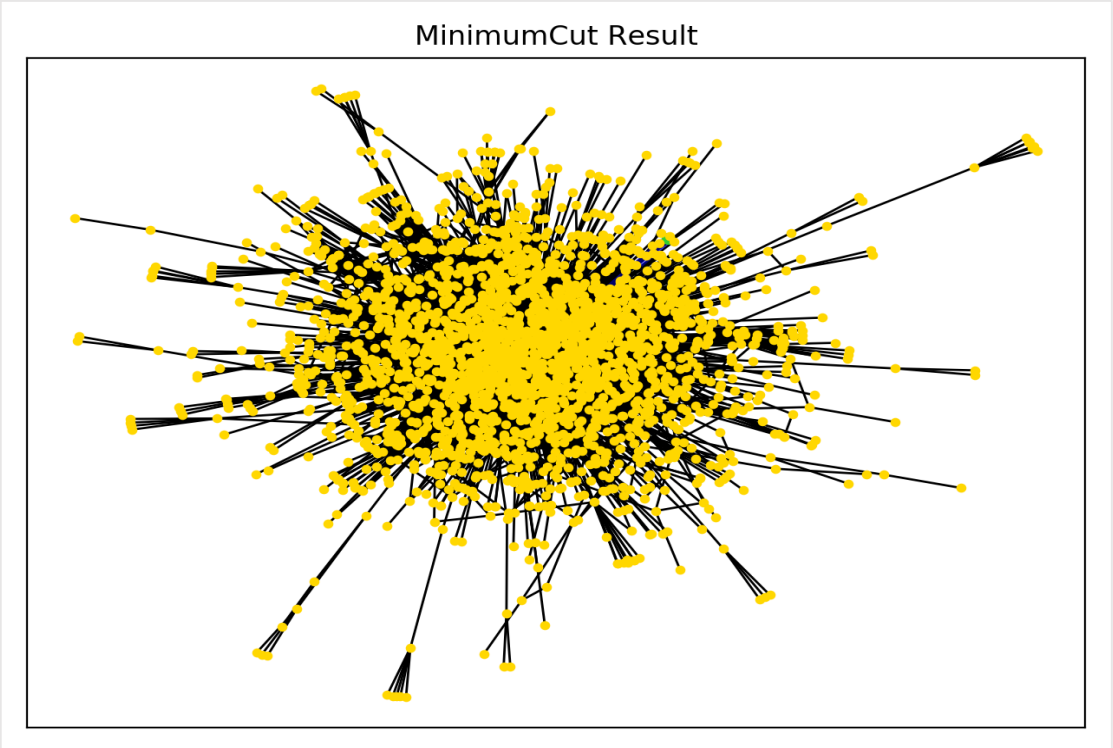


5.5.2. Stoer-Wagner 算法结果

最小割值：1

最小割：{1000},{其余点}

运行时间：125.28080677986145 s



5.6. 数据 6

文件名称：RodeEU_gcc.txt

图中点数：1039

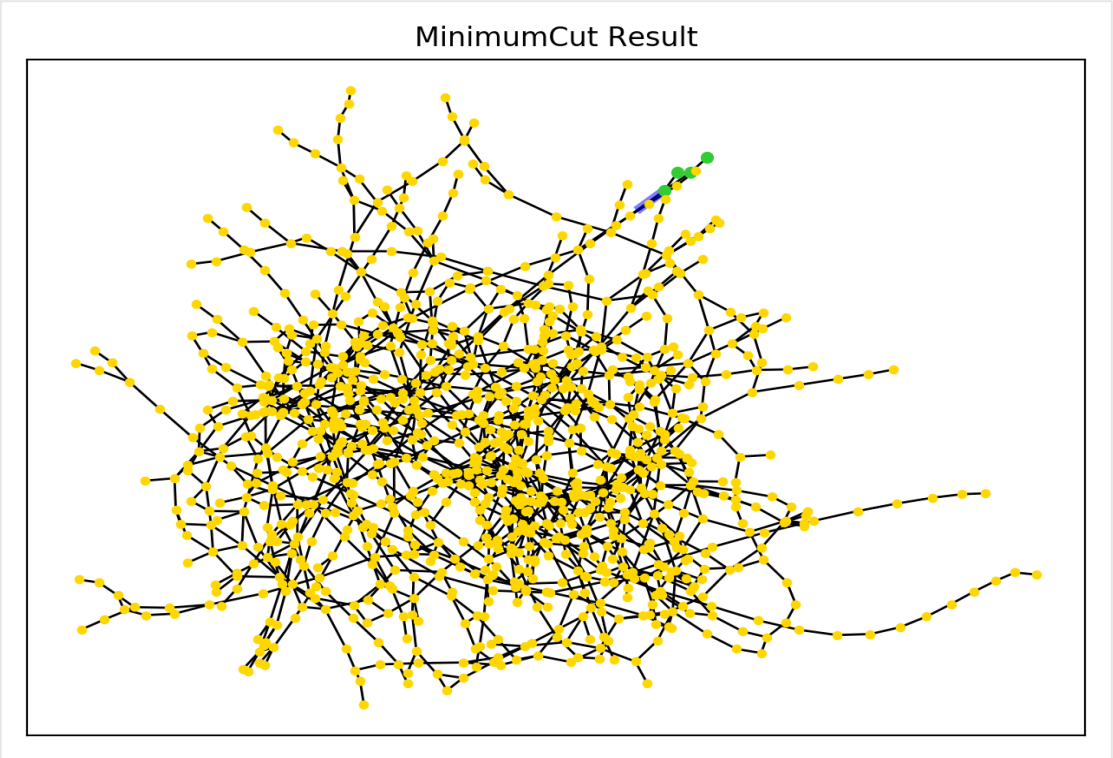
图中边数：1350

5.6.1. Karger 算法结果

最小割值：1

最小割：{ 587, 588, 585, 586 },{其余点}

100 次循环运行时间：30.754778146743774 s

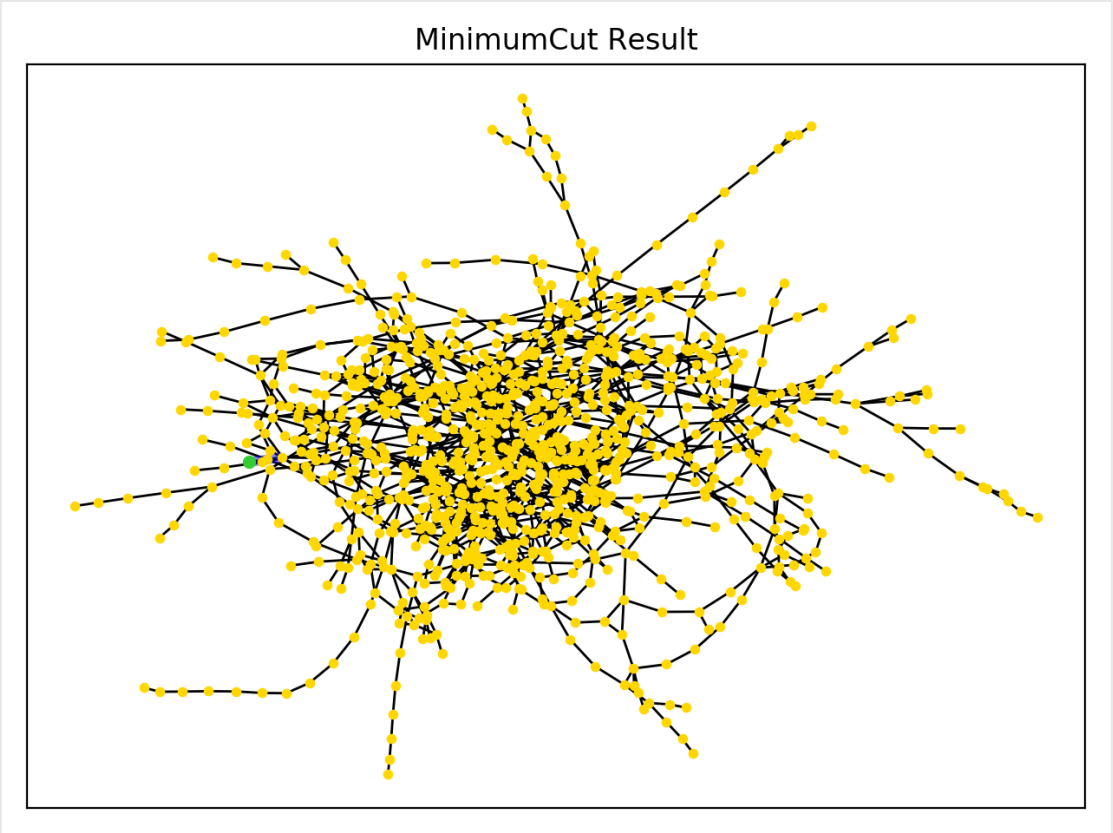


5.6.2. Stoer-Wagner 算法结果

最小割值：1

最小割：{1001},{其余点}

运行时间：14.02881383895874 s



5. 算法改进和推广

5.1. 已有推广和改进

由于最小割问题可以应用于许多问题变形，因此针对不同的问题情形 *Karger* 算法有很多细微的改进调整，对于算法运行时间的优化也存在大量的优化算法，很难逐一列举。因此本部分着重调查了近几年 Karger 算法和 *Stoer-Wagner* 算法具有较大创新意义的应用推广和算法改进，并加以概括阐述。

5.1.1. Karger 算法

(1) 针对算法本身优化

继 *Karger* 算法提出后，*Karger* 和 *Stein* 进一步改进了该算法，提出新的 *Karger-Stein* 算法，该算法在保持算法复杂度几乎不变的基础上，进一步将误差概率改进至$O(1/n)$。

(2) 针对网络可靠性问题

Karger-stein 算法提出后，Karger 进一步提出了一个完全多项式时间的算法[6]［*SIAM Journal on Computing*, 1999］，用于求解当给定每条边独立地以概率 p 被去除时，图变为非连通状态的概率。该算法运行时间为$n^{5+o(1)}\epsilon^{-3}$，其中ϵ为相对误差。该算法对判定网络可靠性具有重要意义。

针对同样问题，在 *Karger* 提出的算法基础上，*DAVID G. HARRIS* 和 *ARAVIND SRINIVASAN*[2017]描述了一种可以同时影响所有原算法界限的新图形参数[7]，这一改进使我们可以获得对割结构的更准确的估计，并将算法运行时间提升至$n^{3+o(1)}\epsilon^{-2}$。其改进过程也同时验证了 *Karger* 和 *Tai* 的实验结果[8]（*Proc. ACM-SIAM Symposium on Discrete Algorithms*, 1997）。

(3) 针对超图情形

Kyle Fox[*SIAM*,2019]等人首次提出 *Karger* 算法在超图情形下的应用推广[9]，提出了可以解决超图中 *Minimum k-cut* 的算法，其时间复杂度为$O(mn^{2k-2})$。此算法是作为超图上的无根分枝随机收缩技术的实例而获得的，它扩展了 *Karger* 和 *Stein* 关于图中递归收缩的工作。除了超图情形，该算法和结果也可以推广到边图上的最小对冲割和最小对冲 k 割问题。

David P. Williamson 在此基础上进一步考虑了算法的变形并做了进一步优化[10]。

(4) 针对算法运行速度

给定一个加权图 G 及其生成树 T，求 T 中最多包含两条边的割中的最小割。这个问题是 *Karger* 算法中的一个重要子程序，也是改进 Karger 算法的重要因素。

针对此问题[11]*Sagnik Mukhopadhyay* 和 *Danupon Nanongkai*[2020]针对这个问题提出了一种新的方法，从而产生加权图的随机最小割算法，其时间复杂度为$O(m\frac{log^2n}{loglogn}+nlog^6n)$。以往的算法改进通常基于简单图的强假设，即图无权重且无平行边［*Henzinger, Rao, Wang, SODA'17; Ghaffffari, Nowicki, Thorup, SODA'20*］。该算法在降低算法复杂度的同时改善了 *Karger* 算法的应用界限，仅要求图不是过度紧密或稀疏，而不要求上述假设。特别地，对可能有平行边的无权图，算法可进一步改进至时间复杂度为$O(m\frac{log^{1.5}n}{loglogn}+nlog^6n)$。

5.1.2. Stoer-Wagner 算法

(1) 针对算法本身优化

Brinkmerier[2016]通过尽可能在每个阶段合并一对以上的节点，改进了*Stoer-Wagner*算法[13]。这种改进将算法的运行时间减少到 $O(n^2 \cdot (max(log(n), min(m/n, \delta_G/\varepsilon))$，其中 δ 是最小节点度，ε 是最小边权重。

论文中具体提出了算法的两个改进，他在无向图的顶点上计算一个最大邻接顺序，然后按这个顺序收缩最后两个顶点。重复 n-1 次。对于实数加权边情况，第一个变化是通过尽可能收缩一对以上的顶点来减少平均运行时间，将实权边的渐进最漫运行时间减少到$max(log(n), min(m/n, \delta/\varepsilon)$。特殊地，对于整数加权边情况，第二个变化是允许额外放宽应用的最大邻接顺序，此改进对于具有非负整数权重的无向图，可以进一步将算法运行时间减少至$O(\delta_G n^2)$。这比起 *Nagamochi* 和 *Ibaraki* 得到的$O(Mn + n^2)$有了提升。具体证明见[13]。

(2) 针对云数据分区问题

分区技术是在云中实现扩展架构和支持多节点数据放置的关键步骤。

Xiaona Li 等人[2016]提出了一种适用于 SaaS(软件即服务)应用的多租户数据划分模型和算法[14]。它解决了现有云数据管理带来的数据分区会产生大量分布式事务的问题。由相同事务访问的一个租户的定制表可以基于相关性组算法构造为最小的相关性粒度.然后我们构造了一个抽象图，其中组是基本单元，事务访问情况为权重。通过提出一种基于 *Stoer-Wagner* 算法和相关性组的多租户数据分割算法，确保了跨分区的分布式事务数量最小化，从而得到以组为粒度的多租户划分。

(3) 针对计算机图形学的三维实体转换问题

边界表示法（*Boundary Representation*，*B-Rep*）和构造实体表示法（*Constructive Solid Modeling*，*CSG*）是 2 种应用最广泛的三维实体表示方法，研究 *B-Rep* 模型至 *CSG* 模型的转换算法具有重要的理论意义和应用价值。

为了增强转换所得 CSG 模型的可读性，罗月童等人首次提出利用面壳封闭技术改 *B-Rep* 至 *CSG* 转换算法[15]。*B-Rep* 和 *CSG* 转换包括生成基本体元和构建 *CSG* 树。基于面壳封闭的 *B-Rep* 模型分解算法能生成基本体元。提出构建 *CSG* 树的算法，并提出 *VRG* 边权重及割容量的计算方法，将构建 *CSG* 树的问题转换为求解最小割问题，然后基于改进的 *Stoer-Wagner* 最小割算法实现从 *VRG* 至 *CSG* 树的转换，最终实现 *B-Rep* 和 *CSG* 转换。

5.2 自主推广和改进

5.2.1. Karger 算法

我们知道 *Karger* 算法中，若给定边数为m，点数为n的图，P_n为单次收缩后仍存在特定割的概率。由关系式$P_n \geq (1-\frac{2}{n})P_{n-1}$和初始值$P_2 = 1$，关系式可递推为：

$$P_n \geq \prod_{i=0}^{n-3}\left(1-\frac{2}{n-i}\right) = \prod_{i=0}^{n-3}\frac{n-i-2}{n-i} = \frac{n-2}{n}\cdot\frac{n-3}{n-1}\cdot\frac{n-4}{n-2}\cdots\frac{2}{4}\cdot\frac{1}{3} = \binom{n}{2}^{-1}$$

P_n概率最小为$1/\binom{n}{2}=O(\frac{1}{n^2})$。这个正确概率看似不高，但我们运行算法多次。在每次运行中，没得到特定最小割的概率$P = 1-\frac{1}{n^2}$。那么运行 T 次没得到的概率$P_m = (1-\frac{1}{n^2})^T$。当$T = n^2$，失败的概率小于$\frac{1}{e}$。当$T = n^2 ln(n)$时，失败的概率更是不到$\frac{1}{n}$，考虑到一般图的节点数量，此失败概率可以忽略不计。而对于单次收缩过程，由于我们把一个有 n 个顶点的图收缩成只有 2 个顶点的图。通过使用邻接矩阵来表示图，每条边的收缩需要$O(n)$时间，而我们需要收缩$n-2$次，因此单次收缩过程需要$O(n^2)$时间。因此整个算法在$T = n^2$情况下复杂度为$O(n^4)$。

以下我们通过结合 *Karger* 算法和最大流算法，可以在保持算法复杂度不变的基础上降低失败概率。本文中我们考虑假设当$T = n^2$时的情景。原算法复杂度为$O(n^4)$，失败概率约为$\frac{1}{e}$。本文算法进一步将失败概率降低至$(\frac{1}{e})^n$。

我们假设每次收缩过程当收缩到剩下 r 个点时停止（原 Karger 算法中 t=2）此时我们重新得到关于P_n的递推关系式为：

$$P_n \geq \prod_{i=0}^{n-r-1}\frac{n-i-2}{n-i} = \frac{r(r-1)}{n(n-1)} = O(\frac{r^2}{n^2})$$

(1) 失败概率

我们令$r = \sqrt{n}$，此时可以得到P_n概率最小为$O(\frac{1}{n})$。当$T = n^2$时，失败的概率$P_T = (1-\frac{1}{n})^T = ((1-\frac{1}{n})^n)^n = (\frac{1}{e})^n < \frac{1}{e}$。比原算法有显著降低。

(2) 算法复杂度

每次随机收缩过程收缩$n-r$个点，此过程需要$O((n-r)^2)=O\left((n-\sqrt{n})^2\right)=O(n^2)$时间，此过程与原 *Karger* 算法一致。

对剩下r个点我们采取决定性的最大流算法（*Edmonds-Karp* 实现的增强路径算法，假设图有r个点，对确定的两点算法复杂度为$O(r^3)$）。通过最小割最大流定理，我们得到了 s-t 最小割作为副产品的算法。解决图最小割问题的原始想法是枚举所有可能的 s-t 对。然而这是多余的，因为不失一般性，节点 s（前$n-r$个点收缩而成的点）属于切割的一个分区，因此我们只需要通过遍历剩下的顶点上作为节点 t。因此，需要重复$r-1$次该算法，总的时间复杂度为$O\left(r^3(r-1)\right)=O(r^4)=O\left(\sqrt{n}^4\right)=O(n^2)$。这与前$n-r$个点所耗费的时间一致，进一步说明了取$r=\sqrt{n}$的合理性。

因此整个算法过程所需时间为$O(n^2\cdot n^2)=O(n^4)$,这与原 *Karger* 算法保持一致。根据上述证明，同理我们也可以说明若选择保持失败概率不变，我们可以反之降低算法的运行时间，证明略。

5.2.2. Stoer-Wagner 算法

Stoer_Wagner 是为了解决在非负权重无向图中的最小割问题。算法在无向图中计算一个$s-t$最小割，之后将点$s-t$合并。重复 n-1 次后就获得了图的全局最小割。在每次计算$s-t$最小割后，只进行一次合并。而如果可以进行多次合并，所需的循环次数就会变少。算法的运行时间就会加快。

为了实现这一想法，我们需要改进 *Stoer_Wagner* 算法在求$s-t$最小割是的更新策略，也就是不断求与点集连接最密集的点，直到图中最后一个点。我们希望可以不重复寻找至到最后一个点，而是引入一个阈值τ，当某一点与当前点集的连接权重和 大于该阈值τ时，就停止寻找，合并这个点与上一个加入点集的点（类似 *Stoer_Wagner* 中合并最后加入点集的两点）。 对于图中的所有点，其边的权重和称为这个点的度。$deg(v)=\sum_{u\in V}w(u,v)$

阈值τ设为图中点的度的最小值。 当图中点数小于 2 时τ即为该图的全局最小割值。该阈值表述该算法流程图如下

Improved Stoer Wagner Algorithm :

Input：非负无向图 G

Output： 1.图 G 的全局最小割

2.最小割值

Solution ： 初始化 $cut \longleftarrow \emptyset \ \tau \longleftarrow +\infty$

While $|V| > 2$：

Do 1. $v = argmin\{deg(p)|\, p \in V\}$

$cut = [v]$

2. 初始化点集$A \longleftarrow \{V$ *中任意一点*$\}$

While $A \neq V$：

Do 1. 将与点集A连接最紧的点s加入A

2. If $\sum_{u \in A} w(u,s) > \tau$

合并点s 和上一个加入 A 的点t

End

End

Return cut、τ

该改进算法与 *Stoer_Wagner* 不同之处在于尽可能在一次循环内合并一对以上的点。为了达到这一目的我们引入阈值τ= $min\{deg(p)|\, p \in V\}$ ，因为图的全局最小割不可能大于τ。所以如果在寻找连接最紧密的点时，某一点与当前点集的连接权重和大于阈值时。合并该点与上一个加入点集的点。这个操作并不会增加阈值τ的值。所以最终当图中仅有 2 个点时，此时的阈值τ为全局最小割值。同时这个操作减少了循环次数。加快了算法的运行速度。

参考文献

[1] A. V. Goldberg and R. E. Tarjan, A New Approach to the Maximum-Flow Problem, *J. Assoc. Comput.*Mach*., 35 (1988), 921－940.

[2] J. Naor and L. Zosin, A 2-Approximation Algorithm for the Directed Multiway Cut Problem, *Proc* 38*th*IEEE FOCS*, 1997, pp. 548－553.

[3] Saran, H.; Vazirani, V. (1991), Finding k-cuts within twice the optimal, *Proc. 32nd Ann. IEEE Symp. on Foundations of Comput. Sci*, IEEE Computer Society, pp. 743－751

[4] O. Goldschmidt and D. S. Hochbaum, A Polynomial Algorithm for the *k*-Cut Problem for Fixed *k*,Math*. *Oper. Res.*, 19 (1994), 24－37.

[5] M. R. Henzinger, P. Klein, S. Rao, and S. Subramanian, Faster Shortest-Path Algorithms for Planar Graphs, *J. Comput. System Sci.*, 55 (1997), 3－23.

[6] Karger, D.: A Randomized Fully Polynomial Time Approximation Scheme for the All Terminal Network Reliability Problem. SIAM Journal on Computing 29-2, 492-514 (1999)

[7] DAVID G. HARRIS, ARAVIND SRINIVASAN, IMPROVED BOUNDS AND ALGORITHMS FOR GRAPH CUTS AND NETWORK RELIABILITY.

[8] Karger, D., Tai, P.: Implementing a Fully Polynomial Time Approximation Scheme for All Terminal Network Reliability. Proceedings of the ACM-SIAM Symposium on Discrete Algorithms, 334-343 (1997).

[9] K. Fox, D. Panigrahi, and F. Zhang. Minimum cut and minimum k-cut in hypergraphs via branching contraction. In Proceedings of the 30th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 881－896, 2019.

[10] David P. Williamson, Recursive Random Contraction Revisited.

[11] Sagnik Mukhopadhyay, Danupon Nanongkai, Weighted Min-Cut: Sequential, Cut-Query and Streaming Algorithms.

[12] Xiaona Li • Junli Zhao • Yumei Ma • Pingping Wang • Hongyi Sun • Yi Tang, A partition model and strategy based on the Stoer-Wagner algorithm for SaaS multi-tenant data, Soft Comput (2017) 21:6121-6132 DOI 10.1007/s00500-016-2169-z.

[13] M. Brinkmeier, A simple and fast min-cut algorithm, Theory of Computing Systems 41 (2) (2007) 369-380.

[14] Jiaxiao Zheng, Wenjun Xu, Gaofei Sun, Xiaohua Tian, Xinbing Wang, Hybrid Channel Assignment in Multi-hop Multi-radio Cognitive Ad Hoc Network, IEEE ICC 2013 - Ad-hoc and Sensor Networking Symposium.

[15] 罗月童,樊晓菁,俞盛朋,王寒冰,周俊,龙鹏程,FDS 团队,基于面壳封闭的 B-Rep 至 CSG 转换算法.