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 数科班

学生姓名　　学号

尹爱华　　201800820052

唐炜权　　201800810075

王郡浩　　201800810082

日　　期：　2020 年 12 月 29 日

得　　分：

全局最小割及其算法

1. 摘要

本文研究了研究了正权重有向图、正权重无向图、含有负权重的部分图的最小割和全局最小割问题以及最小 k 割和全局最小 k 割的时间复杂度。本文编写总结了 Karger 算法和 Stoer-Wagner 算法的思想和步骤，并编写 python 程序求解了题目中 6 组数据的全局最小割。为了更快速的求解全局最小割问题，我们对 Karger 算法和 Stoer-Wagner 算法的改进方法进行了总结。类比其改进思想，针对 Karger 算法，我们通过预处理大致确定最小割的范围进一步优化了算法；针对 Stoer-Wagner 算法，我们创新性的使用 Quake-Heaps 存储节点，降低了算法的时间复杂度，也使得算法运作更为简单。

2. 引言

图 G 的全局最小割是指把顶点集 V 分为两个子集 S 和 T，使得 S 到 T 的权重之和最小。全局最小割问题是图形优化的一个基本问题，由于其在交通、运输、经济、图像分割等方面中均有重要应用，因此受到了研究界的广泛关注。

全局最小割问题能否在多项式时间内解决决定了其解决实际问题的效率。McCormick 的研究表明正权重无向图中的最小割问题是 p 问题 0，Goldschmidt 则证明了全局最小 k 割问题是 NP-Hard 问题[2]。因此，在不同的图 G 中，不同定义下，最小割和全局最小割问题具有不同的时间复杂度。因此，研究能否有效解决以及在什么条件下可以有效解决最小割和全局最小割问题具有重要意义。

在本文中，我们研究了正权重有向图、正权重无向图、含有负权重的部分图的最小割和全局最小割问题的复杂度。在实际问题中，割集数量大于 2 的情形有较为广泛的应用，因此，本文也讨论了最小 k 割和全局最小 k 割的时间复杂度。

Karger 算法和 Stoer-Wagner 算法是加权最小割问题的两个开创性算法。Karger 算法在$O(n^2 mlogn)$时间内以高概率在含有 m 条边和 n 个点的图中产生全局最小割；Stoer-Wagner 算法可以在$O(nm + n^2 logn)$时间内不断减小图中点的数量产生全局最小割。在本文中，我们编写 python 代码实现了这两个算法并解决了题目中 6 个图的全局最小割问题(结果见图 2、图 3)。

为了更快速的求解全局最小割问题，我们对这两个算法进行了改进。针对 Karger 算法，从边收缩角度进行优化，我们使用 Karger-Stein 算法在$O(n^2 log^3 n)$时间内求解了题目中的 6 个全局最小割，并与 Karger 算法的运行时间进行了对比(见表 3)；从树集的角度进行优化，我们总结了利用轻重链剖分简化 Karger 算法的相关算法。针对 Stoer-Wagner 算法，我们利用斐波那契堆存储节点，通过提升每一阶段最小割算法的效率，提升整个算法的效率。类比这些改进思想，针

对 Karger 算法我们通过预处理大致确定最小割的范围进一步优化了算法；针对 Stoer-Wagner 算法我们使用 Quake-Heaps 存储节点，降低了算法的时间复杂度，也使得算法运作更为简单。

本文的结构如下：在第 3 节中，我们研究了不同图 G 中和不同定义下的最小割及全局最小割问题的时间复杂度。在第 4 节中，我们首先总结了 Karger 算法和 Stoer-Wagner 算法的思想和步骤，接着运用这两种算法编程求解了题目中的 6 组数据，最后，我们总结了这两个算法在运用中的特点。在第 5 节中，我们从不同角度总结了这两种算法的改进方法，并类比提出了一种新的改进方法。

3. 最小割及全局最小割问题的时间复杂度

交通、运输、经济、图像分割等领域的很多问题都可以转化成最小割和全局最小割问题，因此研究能否有效解决以及在什么条件下可以有效解决该问题具有重要意义。

因此，本文给出了最小割以及全局最小割的定义，并对不同条件下的最小割和全局最小割问题的时间复杂度进行了讨论和证明。

3.1 定义

3.1.1 最小割

设图$G=(V,E)$，其中，V 是顶点集，E 是边集。设 V 中点的个数为 n，E 中边的条数为 m，s 为源点，t 为收点。c(u,v)表示从点 u 到 v 的边的容量。将图 G 的顶点集分为两个集合 S 和 T，且满足 s∈S,，t∈T,，S∩T=∅，S∪T=V，使得$\sum_{u\in S}\sum_{v\in T}c(u,v)$最小的一种分割称为 s-t 最小割，简称为最小割，记为 C(S,T)。

3.1.2 全局最小割

设图 G=（V，E），其中，V 是顶点集，E 是边集。设 V 中点的个数为 n，E 中边的条数为 m。c(u,v)表示从点 u 到 v 的边的容量。将图 G 的顶点集 V 分为两个集合 S 和 T，且满足 S∩T=∅，S∪T=V，使得$\sum_{u\in S}\sum_{v\in T}c(u,v)$最小的一种分割方式称为全局最小最小割,记为 C(S,T)。

下面，本文将讨论 G 为正权重有向图、正权重无向图、含有负权重的图时的最小割和全局最小割的时间复杂度问题，以及拓展定义下的最小 k 割和全局最小 k 割的时间复杂度。

3.2 正权重图

3.2.1 有向图最小割

- 结论：*在正权重有向图中，最小割问题是 P 问题。*

证明：

(1) 定理 1：在正权重有向图中，已知从 s 到 t 的最大流，可以在$O(n^2)$时间内找到该图的 s-t 最小割。

证明：

① 存在性：当达到最大流时，根据增广路定理，残留网络中没有从 s 到 t 的路径，否则流量还能继续增大。把源点 s 能到的点集设为 S，剩余的点集设为 T，则构造出了一个割集 C(S,T)。S 到 T 的边必然满流，否则在残留网络中存在边(u,v)，u∈S, v∈T，进而 s 能到 v，即 v∈S，这与 v∈T 矛盾。这些满流边的流量和即为当前最大流，把这些满流边作为割，即构造出了一个和最大流相等的割。

② 复杂度：根据已知的最大流，可以在$\boldsymbol{O}(m)$时间内求出残留网络。在残留网络中，使用 DFS 算法求解 s 的可达点，需要$\boldsymbol{O}(n^2)$的时间。所以根据最大流可以在$\boldsymbol{O}(n^2+m)$时间内求得最小割。

因此，在有向图中，可以通过求解最大流求解最小割。

下面，以最大流的求解算法 Dinic 算法为例分析最小割问题的复杂度。

(2) 最大流算法复杂度：

1) Dinic 算法流程图如下图所示：

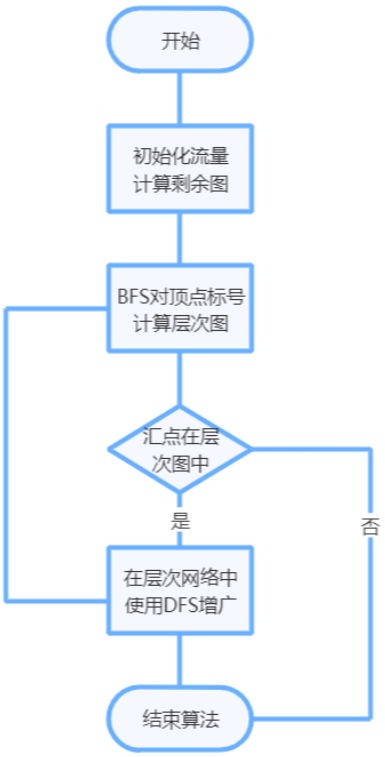

图 1：Dinic 算法流程图

2) Dinic 算法复杂度分析：Dinic 算法中 DFS 过程首先修改增广路上边的流量。至多会增广 m 次，一条增广路的长度至多是 n，所以复杂度是$\boldsymbol{O}(mn)$；其次 DFS 遍历寻找增广路失败时经过的边。若从某条边出发找最短路失败了，该算法不会再走这条边，所以这一部分的复杂度是$\boldsymbol{O}(m)$。因此 DFS 增广的复杂度是$\boldsymbol{O}(mn)$。由于每一次重建分层图，残留网络上从 s 到 t 的最短路长度一定会增加，所以至多重建(n-1)次图。因此 Dinic 运行时间上界为$\boldsymbol{O}(n^2m)$。

(3) 由(1)和(2)可知，根据最大流算法可以在$\boldsymbol{O}(n^2m)+\boldsymbol{O}(n^2+m)=$

$\boldsymbol{O}(n^2m+n^2+m)$时间内求出最小割。

因此，在该定义下最小割问题是 P 问题。

3.2.2 有向图全局最小割

➢ 结论：*在正权重有向图中，全局最小割问题是 P 问题。*

证明：未知源点和收点，全局最小割的割值等于∀s ∈ V，∀t ∈ V，s-t 最小割的最小值。固定∀点 s，则最小割为剩余(n-1)个点与 s 点的最小割的最小值。

由于最小割的求解复杂度为$\boldsymbol{O}(n^2m+n^2+m)$，所以，求解剩余(n-1)个点与 s 点的最小割共需要$\boldsymbol{O}((n^2m+n^2+m)(n-1))=\boldsymbol{O}(n^3m-n^2m+n^3-n^2+nm-m)$的时间。

因此，在该定义下全局最小割问题是 P 问题。

3.2.3 无向图最小割

➢ 结论：*在正权重无向图中，最小割问题是 P 问题。*

证明：本文将无向图最小割转化成有向图最小割证明其复杂度。

对∀无向图 G=(V,E)，本文按照如下方式构造出其对应的唯一的有向图 G'=(V',E')：

① V' =V

② 若在图 G 中，c(u,v)>0，则在 G'中，令 c(u',v')= c(v',u')= c(u,v)

因为 C(S',T')为 G'中 s'-t'的最小割，所以 s'∈S'，t'∈T'。因为 V'=V，所以 s∈S，t∈ T。因此，C(S,T)为 G 中的 s-t 的最小割。所以，若 C(S',T')为 G'中 s'-t'的最小割，G 中 S 与 S'对应，T 与 T'对应，则 C(S,T)为 G 中的 s-t 的最小割。

因此，无向图的 s-t 最小割，可以等价转化为有向图的 s-t 最小割。因为有向图的最小割问题时间复杂度问题为$\boldsymbol{O}(n^2m+n^2+m)$，所以无向图的最小割问题的时间复杂度问题为$\boldsymbol{O}(n^2m+n^2+m)$。

因此，在该定义下，最小割问题是 P 问题。

3.2.4 无向图全局最小割

➢ 结论：*在正权重无向图中，全局最小割问题是 P 问题。*

证明：未知源点和收点，全局最小割的割值等于∀s ∈ V，∀t ∈ V，s-t 最小割的最小值。固定∀点 s，则最小割为剩余(n-1)个点与 s 点的最小割的最小值。

由于无向图最小割的求解复杂度为$\boldsymbol{O}(n^2m+n^2+m)$，所以，求解剩余(n-1)个点与 s 点的最小割共需要$\boldsymbol{O}((n^2m+n^2+m)(n-1))=\boldsymbol{O}(n^3m-n^2m+n^3-n^2+nm-m)$的时间。

因此，在该定义下全局最小割问题是 P 问题。

3.3 含有负权重的图

由于含有负权重的图的最小割和全局最小割问题较为复杂，因此，本文只选取几种特殊图研究其最小割和全局最小割的复杂度

3.3.1 s-t 负的加权图的最小割

E_w^+是图中所有正边的集合，E_w^-是图中所有负边的集合。s 为源点，t 为收点。如果E_w^-是集合 C({s}，V/{s})∪C({t}，V/{t})的子集，则称该图为 s-t 负的。

- 结论：*在 s-t 负的加权图中，最小割问题是 P 问题。*

证明：带有负权重的最小割问题可以利用 s-t 负的加权图的性质转化成正权重的最小割问题解决。

通过添加权重为 0 的边，可以不失一般性地假定 E 包含所有 i∈V\{s，t}的边(s，i)和(i，t)。由于每个 s-t 割正好包含所有 i∈V\{s，t}的每对边(s，i)和(i，t)中的一个，因此我们可以向 c(s,i)和 c(i,t)添加一个任意常数，使它们均为非负数。类似地，如果(s，t)∈E 并且 c(s,t) <0，我们可以向 c(s,t)添加一个任意常数以使其为非负数。所得的加权图具有非负权重，因此可以在多项式时间内找到其最小割。

因此，在该定义下最小割问题是 P 问题

3.3.2 近似正权图的全局最小割

如果边集E_w^-可以被至多$\boldsymbol{O}(logn)$个点覆盖，则该加权图定义为近似正权图

- 结论：*在近似正权图中，全局最小割问题是 P 问题。*

证明：由定理 3：在近似正权图中，最小割问题可以在多项式时间内解决[1]。

因此，在该定义下，全局最小割问题是 P 问题。

3.4 拓展定义——k 割问题

3.4.1 最小 k 割问题

- 定义

给定图 G=（V,E），一个集合$S = \{s_1, s_2, \dots\dots, s_k\}$是 k 个给定的顶点，每条边 e∈E 存在正权重 c(e),寻找一个权重和最小的边集 E'∈E，使得移除 E'后 S 中的点互不连通。该边集成为最小 k 割。

- 结论：*最小 k 割问题是 NP-Hard 问题。*

证明：定理 4：如果 k 不固定，平面图的最小 k 割问题是 NP-Hard 问题；定理 5：对于任意固定的 k≥3，对于任意图，最小 k 割问题是 NP-Hard 问题[2]。

所以，在该定义下最小 k 割问题是 NP-Hard 问题。

3.4.2 全局最小 k 割问题

3.4.2.1 固定 k 下的全局最小 k 割问题

- 结论：*固定 k 下的全局最小 k 割问题是 P 问题。*

证明：Goldschmidt 在 1994 年提出的 k-cut algorithm 可以在$\boldsymbol{O}\left(n^{k^2}\right)$时间内解决全局最小割问题[2]。因此，在 k 固定的条件下。全局最小割问题可以在多项式时间内解决。

因此，在该定义下，全局最小割问题是 P 问题。

3.4.2.2 任意 k 下的全局最小 k 割问题

➢ 结论：*任意 k 下的全局最小 k 割问题是 NP-Complete 问题。*

证明：最大团决策问题定义为：给定一个图 G=（V,E）和一个正整数 M，G 的最大完全子图是否恰好包含 M 个顶点。

以（0,1）为权重的 k 割问题等价于将 V 划分为 k 个非空分量，以使两端在同一分量中的边的数量最大。一个简单的计算表明，如果 G 有一个大小为 M = |V| - (k - 1)的团，那么 k 割问题的任意解决方案都将以这个团作为其割集之一。因此，任意 k 的最大团问题可以规约成任意 k 的最小割问题。因为对于任意 k，最大团问题都是 NP-Complete 问题[2]，所以对于任意 k，最小 k 割问题是 NP-Complete 问题

因此，在该定义下，全局最小割问题是 NP-Complete 问题。

3.5 总结

表 1：最小割和全局最小割 pVSnp 总结

正权重有向图		正权重无向图		含负权重的图		K 割		
最小割	全局最小割	最小割	全局最小割	s-t 负最小割	近似正权图全局	最小割	固定 k 全局	任意 k 全局
P	P	P	P	P	P	NPH	P	NPC

4. 全局最小割算法

从上面的证明中可以看出，全局最小割时间复杂度更高。同时，它在在实际应用中更广泛。所以研究设计有效算法解决全局最小割问题有重要意义。下面，本文将介绍、实现并应用两种常见的全局最小割算法。

4.1 Karger 算法

4.1.1 算法思想

如果算法识别出不跨越最小割的边，那么就可以将这个边收缩，这不会影响最小割的结果。当非最小割边都被收缩后，剩下的边就是全局最小割。

4.1.2 算法流程图

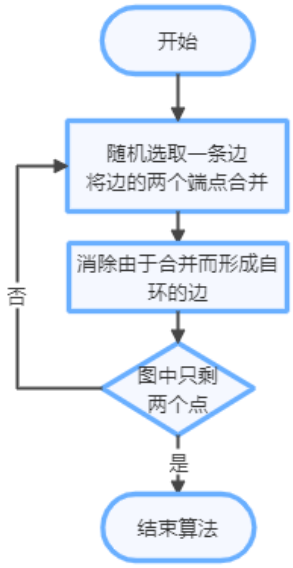

图 2：Karger 算法流程图

4.1.3 算法实现

本文用 Python 实现了该 Karger 算法，具体代码见附录 1。

4.1.4 算法应用

本文使用 python 代码求解题目中的 6 组数据，结果如下表所示：

表 2：Karger 算法求解结果

数据	测试集	Bench	Corruption	Crime	RodeEU	PPI
点数	10	83	309	754	1039	2224
边数	23	774	3281	2127	1350	6609
割值	3	2	1	1	1	1
S	{6——10}	{61——80}	{309}	{349}	{981}	{314}

4.1.5 算法特点

在算法的应用中，本文总结出 Karger 算法的如下特点：

1. 算法简单，编程实现容易。
2. 网络规模简单时，算法的运行速度较快。
3. 网络规模庞大时，算法的运行时间明显增加。

4.2 Stoer-Wagner 算法

4.2.1 算法思想

如果找到了一个图 G 的 s-t 割，这个 s-t 割要么是 G 的全局最小割，要么不是。如果是全局最小割，那么就得到了问题的解；如果不是，那么 s 和 t 在全局最小割中一定属于同一个集合，这样 G/{s,t}的最小割就是 G 的全局最小割，所以可以按照该思路递归求解全局最小。

4.2.2 算法步骤

■ Karger 算法求解最小割：

1. 设最小割 cut=INF，任选一个点 s 到集合 A 中，定义 W(A，p)为 A 中的所有点到 A 外一点 p 的权总和。
2. 对刚才选定的 s，更新 W(A,p)。
3. 选出 A 外一点 p，且 W(A,p)最大的作为新的 s，若 A!=G(V)，则继续 2.
4. 把最后进入 A 的两点记为 s 和 t，用 W(A,t)更新 cut.
5. 新建顶点 u，边权 w(u，v)=w(s，v)+w(t，v)，删除顶点 s 和 t，以及与它们相连的边.
6. 若|V|!=1 则继续 1.

4.2.3 算法实现

本文用 Python 实现了该 Stoer-Wagner 算法，具体代码见附录 2。

4.2.4 算法应用

本文使用 python 代码求解题目中的 6 组数据，结果如下表所示：

表 3：Stoer-Wagner 算法求解结果

数据	测试集	Benchmark	Corruption	Crime	RodeEU	PPI
点数	10	83	309	754	1039	2224
边数	23	774	3281	2127	1350	6609
割值	3	2	1	1	1	1
S	{6——10}	{61——80}	{309}	{753}	{48}	{314}

4.2.5 算法特点

在算法的应用中，本文总结出 Stoer-Wagner 算法的如下特点：

1. 网络规模简单时，算法的运行速度较快。
2. 网络规模庞大时，算法的运行速度受到很大限制。

5. 算法改进与推广

5.1 Krager 算法

Karger 算法每一次的运行复杂度为$O(m)$,而每一次运行找到最小割的概率为$\frac{2}{n(n-1)}\sim O(n^2)$，运行 Karger 算法$n^2logn$次，所以算法总的复杂度为$O(n^2mlogn)$。Karger 算法的改进主要从通过边收缩和树集降低其时间复杂度。

5.1.1 边收缩优化角度——Karger-Stein 算法

5.1.1.1 改进思想

1) 每次迭代中，同时运行两个进程，然后选择结果更佳的一个。设成功的概率为 p，则不成功的概率$(1-p)$ 减少为$(1-p)^2$，这样很大程度上减小了每一次迭代不成功的概率。
2) 考虑到在 Karger 算法边收缩的过程中，在前期选中正确的边（集合内的边）的概率要比后期高，所以我们在 Karger-Stein 算法中，尝试提前停下来

5.1.1.2 算法步骤

■ Karger-Stein 算法求解最小割：KargerStein(G)

输入：具有 n 个顶点的图 G
如果 n≥6：
执行两次并输出两次中的最好结果：
使用 Karger 最小割算法收缩边至剩余$\frac{n}{\sqrt{2}}+1$个顶点，将此时的图记为 G’；
递归 KargerStein(G’)

5.1.1.3 算法复杂度分析

从算法步骤可以看出，算法运行一次的递推公式为：$T(n)=2T\left(\frac{n}{\sqrt{2}}\right)+O(n^2)$，可以求得$T(n)=O(n^2logn)$。概率的递推公式为$P(n)\geq 1-(1-\frac{1}{2}P(\frac{n}{\sqrt{2}}))^2$，可以求得$P(n)=\Omega(\frac{1}{logn})$。因此，运行$O(log^2n)$次算法，成功的概率至少是$1-\frac{1}{poly(n)}$。因此，Karger-Stein 算法的复杂度为$O(log^2n*n^2logn)=O(n^2log^3n)$。

5.1.1.4 算法应用

本文将 Karger-Stein 算法与 Karger 算法应用于题目中的六组数据，具体代码见附录 3。两种算法的运行时间如下表所示：

表 4：Karger-Stein 算法与 Karger 算法性能对比

数据	测试集	Benchma	Corrup	Crime	RodeEU	PPI
点数	10	83	309	754	1039	2224
边数	23	774	3281	2127	1350	6609
Karger	0.00099659s	0.34009s	5.87129s	10.88591s	12.2961s	137.67671s
K-Stein	0.00199175s	0.29325s	5.55514s	11.07038s	11.48328s	132.1674s

测试系统配置：Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.71GHz

5.1.1.4 算法效果

可以看出，当网络规模较小时，Karger-Stein 算法运行速度比 Karger 算法慢；但是，当网络规模较大时，Karger-Stein 算法速度较 Karger 算法有了明显提升。因此，对于大规模网络，Karger-Stein 算法比 Karger 算法更有优势。

5.1.2 树集优化角度

5.1.2.1 相关定义

- **权重树集**是生成树组成的集合，每一个生成树都有一个非负的权重使得对任意一条给定的 G 中的边，包含这条边的所有生成树的权重之和不大于这条边的权重。权重树集的权重是其中所有生成树的权重之和。
- **轻重链剖分：**指一种对树进行划分的算法，它先通过轻重边剖分将树分为多条链，保证每个点属于且只属于一条链，然后再通过数据结构（树状数组、BST、SPLAY、线段树等）来维护每一条链。
- **k-respects：**假设 T 为 G 的一个生成树，我们说 G 的一个割 k-respects T 如果这个割最多割掉 T 中的 k 个边。
  以下这种优化算法正是通过轻重链剖分简化了 Karge 算法

5.1.2.1 算法步骤

■ 算法 1：从图 G 中得到一个权重至少为 0.4c 的权重树集，其中，c 是 G 的最小割的值。

G 为有 m 条边和 n 个点的图：

1.对于所有边 e，初始化$l(e) <— 0$.多集P为空集，W 为 0。

2. 重复以下操作：
   a) 找出相对于$l(.)$的最小生成树 T。
   b) 对于所有 e∈T,$l(e) <— l(e) + 1/(75lnm)$。若$l(e)>1$，返回 W,P
   c) $W <— W + 1/(75lnm)$
   d) 把 T 加入到 P 集合中

这个算法的时间复杂度是$O(mclogn)$。

■ 算法 2: 从上一个算法获得的 P 中，采样得到$O(logn)$个生成树

令 d 表示成功概率的指数$1 = \frac{1}{n^d}$。$b = 3 * 6^2(d+2)lnn$。

1. 通过首先归一化 G 的边缘权重，使最小的非零边缘权重具有权重 1，然后将每个边缘权重乘以 100 并舍入为最接近的整数，从 G 形成图形 G'。令 U 为 G'的最小切割大小的上限。
2. 初始化 c'〈—U。重复如下操作：
   a) 通过以下方式构造 H：对于 G'的每个边 e，让 e 具有从二项式分布中抽取的 H 中的权重，概率为$\mathrm{p} = \min(\mathrm{b}/\mathrm{c}', 1)$，并且 G'中 e 的权重为试验次数。将 H 中任何边缘的重量限制为[7/6 • 12b]。
   b) 在 H 上运行算法 1，将权重 w 的边视为 w 个平行边。 有以下三种情况：

i. 如果 p = 1，则将 P 设置为返回的集，然后跳至步骤 3。
ii. 如果返回的包装重量为 24b / 70 或更大，则设置 c'←c'/ 6 并重复步骤 2a 和 2b，将 P 设置为返回的集，然后继续执行步骤 3。
iii. 其他情况下，重复 2a 和 2b，并令 c'←c'/ 2。

这个算法的时间复杂度为$O(mlog^3n)$，并且图 G 的最小割在很大概率上至少 2-respects 本算法得到的生成树中的一个。

■ 算法 3：2-respects T 的最小割

1. 按照顺序排列边，把它们标记为$e_1, e_2, \ldots\ldots, e_{n-1}$。
2. 对于每个非树边 uv，标记每个 i 以使e_i在 T 的 uv 路径上，而$e_i + 1$ 不在 T 的 uv 路径上，反之亦然。标示边缘e_1是否在 T 中的 uv 路径上。
3. 在 T 上初始化数据结构，以使边缘e_j的权重等于其在 T 中的权重。
4. 将索引 i 从 1 迭代到 n-1。通过步骤 2 中的计算，在迭代 i 时，在数据结构中保持以下不变式。
   a) 当边e_i在 T 中的 uv 路径上时，将非树边 uv 的权重添加到 T 中 uv 路径的所有边上。
   b) 当边e_i离开 T 中的 uv 路径时，将非树边 uv 的权重添加到 T 中 uv 路径上的所有边上。

   每次 i 递增时，更新权重后，将∞加到边e_i上，执行 QueryMinimum()，然后从边 e 减去∞。每次迭代中找到的最小割的值是 QueryMinimum()的结果加上e_i的权重。
5. 当我们认为边e_i不在数据结构中所有非树边 uv 的路径上时，以 QueryMinimum()的结果返回在步骤 4 中找到的最小割的最小值。

这个算法可以在$O(mlog^2n)$时间内找到图 G 的 2-respects T 的最小割。

5.1.2.3 算法复杂度分析

算法 2 获得了$O(logn)$个生成树，并且图 G 的最小割在很大概率上至少 2-respects 本算法得到的生成树中的一个。又因为对前面那些树中的任一个树 T，算法 3 可以在$O(mlog^2n)$时间内找到图 G 的 2-respects T 的最小割。因此，该算法遍历$O(logn)$个生成树，以每个树$O(mlog^2n)$的复杂度找到 2-respects 这棵树的最小割。从而算法总的时间复杂度为$O(log^3n)$。

5.1.3 团队改进

我们在 Karger-Stein 的基础上做进一步的优化

5.1.3.1 预处理

（1） 首先我们遍历图中的每个结点，记录下度（结点的边的条数）最小的节点，把最小节点的度记为 m。
（2） 根据 m 的值，分以下三种情况：
① 如果最小割 c=m，那么度为 m 的结点的边即为最小割；
② 如果最小割 c≠m，那么最小割 c 必定满足 1≤c<m，这样我们有了最小割 c 的取值范围。

现在我们就针对 c≠m 的情况来对算法进行优化（其他两种情况在预处理时便可以求得最小割，所以这里不再讨论）

5.1.3.2 对 Karger-Stein 的改进

Karger-Stein 算法尝试提前终止缩边以降低缩到割边的概率，然后寻找终止后的规模较小的图的全局最小割，缩边终点的计算方式如下：

（1） 设我们在只剩l个节点时停下。因为我们每一步都同时跑两个进程，所以我们希望停下时成功的概率要在$\frac{1}{2}$左右。

（2） 由 Karger 算法中关于成功概率的推导（参考 1）可以知道，在n个结点时开始，在l个节点时停止，其成功的概率

$P_l = P(e_1, \ldots\ldots, e_l \notin \delta) = \frac{n-2}{n} \times \frac{n-3}{n-1} \times \frac{n-2}{n} \times \ldots\ldots \times \frac{l-1}{l+1} = \frac{l(l-2)}{n(n-1)} \geq \frac{1}{2}$，其中 δ 是最小割中跨越两个集合的边集，$e_i$是第 i 次迭代选中的边。

（3） 解得$l = \frac{n}{\sqrt{2}} + 1$。

所以，在$l = \frac{n}{\sqrt{2}} + 1$时停止缩边可以保证每一步成功的概率超过$\frac{1}{2}$，因为一共有两个进程，所以我们就可以期望得到一个没有缩到全局最小割的边的图。

我们已经有了最小割 c 的取值范围，在此基础上，我们对缩边终点进行调整来实现对算法的优化。

我们假设图的总边数为 M，当缩边个数超过 M-m+1 时，下一次缩边必定会把割边给缩掉，所以我们增加一个终止条件：缩边总数不超过 M-m+1，从而算法相应的改为：

■ Karger-Stein 的优化算法求解最小割：KargerSteinOpt(G)

输入：具有 n 个顶点的图 G
如果 n≥6：
　执行两次并输出两次中的最好结果：
　使用 Karger 最小割算法收缩边至剩余$\frac{n}{\sqrt{2}} + 1$个顶点**或者当缩边个数超过 M-m+1 时**将此时的图记为 G’；
　递归 KargerSteinOpt(G’)

5.2 Stoer-Wagner 算法

Stoer-Wagner 算法的复杂度为 n-1 次阶段性的最小割算法的复杂度之和，而阶段性的最小割算法的复杂度为$O(m + nlogn)$，所以 Stoer-Wagner 算法的复杂度为$O(nm + n^2logn)$。

5.2.1 改进思想

SW 算法由若干阶段性的最小割算法构成，因此，可以通过提升每一个小阶段的效率，进一步提升全局算法的表现。

在每一个小阶段的执行过程中，所有不在 A 中的结点都会根据其值保存在一个优先级队列中。一个结点 v 的值是连接它和当前 A 的边的权重之和，即 W(A, v)。每当一个结点 v 被添加到 A 算法便对优先级队列进行更新：从队列中删除

结点 v；对每一个不在 A 中的结点 u，根据 W(A,u)的值相应地调整 u 在优先级队列中的值。

因此，每当融合一条边，算法进行的操作是：

- 提取最大值即在所有结点中，找到与 A 之间的边权重最大的那个节点：复杂度为 O(n)。
- 增加键值即根据新加入的节点和边增加 A 的边的键值：复杂度为 O(m)。

5.2.2 已有算法改进

5.2.2.1 斐波那契堆

斐波那契堆同二项堆一样，也是一种可合并堆。与二项堆一样，斐波那契堆是由一组最小堆有序树构成，但堆中的树并不一定是二项树。与二项堆中树都是有序的不同，斐波那契堆中的树都是有根而无序的。

5.2.2.2 复杂度分析

通过斐波那契堆存储节点，可以将提取最大值和增加键值的复杂度分别降到$O(logn)$和$O(1)$，从而算法总的复杂度降到$O(m+nlogn)$.

5.2.3 团队改进

我们利用 Quake-Heaps 来取代斐波那契堆。Quake-Heaps 在操作的复杂度上和斐波那契堆相同，不过其运作更为简单。在保持平衡性上，Quake-Heaps 的思想主要是“懒惰”，即只有在堆得结构变“坏”之后才将结构重建。

5.2.3.1 Quake-Heaps

Quake-Heaps 是在斐波那契堆的基础上提出的一种数据结构，结构与斐波那契堆相似，都是一些有根无序树的集合。Quake-Heaps 在操作上更为“懒惰”，比如说在减少键值时省去了“cascading”操作，同时结构的重建也只有在堆的结构变“坏”时才进行。在各种操作的时间复杂度上，Quake-Heaps 与斐波那契堆是相同的，不过 Quake-Heaps 相对斐波那契堆更简单、更容易理解。

5.2.3.2 Quake-Heaps 算法

■ Quake-Heaps 算法：

```
insert(x)：
    创建一个包含{x}的新树
decrease-key(x, k):
    切掉根于存储 x 的最高节点的子树
    把 x 的值改为 k
delete-min():
    x<——所有根的最小值
    删除存储 x 的节点的路径
    当有两棵相同高度的树时：
        链接两棵树
```

如果对于某个 i，$n_{i+1} > an_i$:
设 i 为这样的指数中的最小值
删除所有高度＞i 的节点
返回 x

6. 参考文献

[1] S.T. McCormick, M.R. Rao, G. Rinaldi. WHEN IS MIN CUT WITH NEGATIVE EDGES EASY TO SOLVE? EASY AND DIFFICULT OBJECTIVE FUNCTIONS FOR MAX CUT [J]. CONSIGLIO NAZIONALE DELLE RICERCHE, 2000, R.524:1128–3378

[2] OLIVIER GOLDSCHMIDT AND DORIT S. HOCHBAUM. A POLYNOMIAL ALGORITHM FOR THE k-CUT PROBLEM FOR FIXED k. : Mathematics of Operations Research , Feb., 1994, Vol. 19, No. 1 (Feb., 1994), pp. 24-37

[3] Nalin Bhardwaj, Antonio J. Molina Lovett , and Bryce Sandlund, “A Simple Algorithm for Minimum Cuts in Near-Linear Time”, arXiv e-prints, 2019.

[4] David R. Karger. 1993. Global min-cuts in RNC, and other ramifications of a simple min-cut algorithm. In Proceedings of the fourth annual ACM-SIAM symposium on Discrete algorithms (SODA '93). Society for Industrial and Applied Mathematics, USA, 21–30.

[5] David R. Karger and Clifford Stein. 1996. A new approach to the minimum cut problem. J. ACM 43, 4 (July 1996), 601–640.

[6] Michael L. Fredman and Robert Endre Tarjan. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM 34, 3 (July 1987), 596–615.

[7] Mechthild Stoer and Frank Wagner. 1997. A simple min-cut algorithm. J. ACM 44, 4 (July 1997), 585–591.

[8] Chan T.M. (2013) Quake Heaps: A Simple Alternative to Fibonacci Heaps. In: Brodnik A., López-Ortiz A., Raman V., Viola A. (eds) Space-Efficient Data Structures, Streams, and Algorithms. Lecture Notes in Computer Science, vol 8066. Springer, Berlin.

7. 附录

附录 1：Karger 算法代码

```
import random
import copy

def ranContract(v, e):
    connect = []
    while len(v)>2:
        [a, b] = e.pop(random.randrange(0, len(e)))
        connect.append([a, b])
        v.remove(a)
        new = []
        for i in range(len(e)):
```

```
            if e[i][0] == a:
                e[i][0] = b
            elif e[i][1] == a:
                e[i][1] = b
            if e[i][0] != e[i][1]:
                new.append(e[i])
        e = new
    return(len(e), connect, v)

def findE(V1, e):
    connectToV1 = []
    new = []
    cur = [V1]
    while cur!=[]:
        for i in cur:
            for j in range(len(e)):
                if e[j][0]==i:
                    new.append(e[j][1])
                if e[j][1]==i:
                    new.append(e[j][0])
            e = [g for g in e if g[0] != i and g[1] != i]
        connectToV1+=new
        cur = new
        new = []
    return connectToV1
if __name__ == '__main__':
    file = open('kargerMinCut.txt')
    f = list(file)
    edges = []
    vertices = []
    for i in range(len(f)):
        dat = f[i].split()
        vertices.append(int(dat[0]))
        # 200个点
        for j in range(1, len(dat)):
            if [int(dat[j]), int(dat[0])] not in edges:
                edges.append([int(dat[0]), int(dat[j])])
    # print(len(vertices), len(edges))
    min = 100000
    for i in range(200):
        v = copy.deepcopy(vertices)
        e = copy.deepcopy(edges)
        r, g, b = ranContract(v, e)
        if min>r:
```

```
            min=r
            con = findE(b[1], g)
    print("================Karger==============")
    print("the min-cut is ",min)
    print("the min-cut contracted to V1 is \n",con)
```

附录 2：Stoer-Wagner 算法代码

```
import time
import copy

'''''
函数：loaddata
作用：从 txt 文件中读取结点和边的数据
输入：数据路径
输出：结点列表 vertices，边列表 edges
'''
def loaddata(path):
    print("====================="+path+"====================")
    file = open(path)
    f = file.readlines()
    edges = []
    vertices = []
    for i in range(len(f)):
        dat = f[i].split()
        if [int(dat[0]), int(dat[1])] not in edges and [int(dat[1]), int(dat[0])] not in edges:
            edges.append([int(dat[0]), int(dat[1])])
    for edge in edges:
        vertices.append(edge[0])
        vertices.append(edge[1])
    vertices = list(set(vertices))
    return vertices, edges


'''''
函数：stMinCut
作用：寻找 s-t 最小割
输入图：结点列表 vs，边列表 es，结点总数 num_v
输出：s-t 割[[s],A]，s-t 割值 A_ajac[s]，倒数第二个结点 t
'''
def stMinCut(vs,es,num_v):
    ves = copy.deepcopy(vs)
    eds = copy.deepcopy(es)
```

```
    Anode = 1
    A = [Anode]
    A_ajac = [0 for v in range(num_v)]
    A_ajac.append(0)    #转换为 1 起点

    # 初始化 A_ajac(表示与 A 集合相连的紧密程度)
    for e in es:
        if e[0] == Anode:
            A_ajac[e[1]] += 1
        if e[1] == Anode:
            A_ajac[e[0]] += 1

    s = A[0]
    t = None
    while len(vs) > 1:
        maxi = -float('inf')
        nxt = None
        for a in range(1,len(A_ajac)):
            if A_ajac[a] > maxi and a not in A:
                nxt = a
                maxi = A_ajac[a]
        t = s
        s = nxt
        # 把 s 合进 A 中
        A.append(s)
        # 删除 s 点
        vs.remove(s)
        # 删除边、继承边
        new_eds = []
        for e in es:
            # 删除边
            if e == [Anode,s] or e == [s,Anode]:
                continue
            elif e[0] == s:
                new_eds.append([Anode,e[1]])
                A_ajac[e[1]] += 1
            elif e[1] == s:
                new_eds.append([e[0],Anode])
                A_ajac[e[0]] += 1
            else:
                new_eds.append(e)
        es = new_eds

    return [[s],A],A_ajac[s],t
```

```
80.
81. '''''
82. 函数：contract
83. 作用：将 s 结点合并到 t 结点中
84. 输入图：结点列表 vs，边列表 es，结点 s，结点 t
85. 输出：合并之后的结点列表 vs，合并之后的边列表 es
86. '''
87. def contract(vs,es,t,s):
88.     # 删除 s 点
89.     vs.remove(s)
90.     # 删除边、继承边
91.     new_eds = []
92.     for e in es:
93.         # 删除边
94.         if e == [t,s] or e == [s,t]:
95.             continue
96.         elif e[0] == s:
97.             new_eds.append([t,e[1]])
98.         elif e[1] == s:
99.             new_eds.append([e[0],t])
100.         else:
101.             new_eds.append(e)
102.     es = new_eds
103.
104.     return vs,es
105.
106. '''''
107. 函数：globalMinCut
108. 作用：求图的全局最小割
109. 输入图：结点列表 vs，边列表 es，结点总数 num_v
110. 输出：全局最小割集合 cut，全局最小割值 c
111. '''
112. def globalMinCut(vs,es,num_v):
113.     ves = copy.deepcopy(vs)
114.     eds = copy.deepcopy(es)
115.
116.     if len(vs) == 2:
117.         cut = 0
118.         for e in es:
119.             if e == [vs[0],vs[1]] or e == [vs[1],vs[0]]:
120.                 cut += 1
121.         return [[vs[0]],[vs[1]]], cut
122.     else:
123.         cut1, c1, t = stMinCut(ves,eds,num_v)
```

```
            s = cut1[0][0]
            vs,es = contract(vs,es,t,s)
            cut2, c2 = globalMinCut(vs,es,num_v)
            for cut in cut2:
                if t in cut:
                    cut.append(s)
            if c1 <= c2:
                return cut1, c1
            else:
                return cut2, c2


if __name__=='__main__':
    fileList = ['./data/BenchmarkNetwork.txt','./data/test1.txt','./data/Corruption_Gcc.txt', './data/Crime_Gcc.txt', './data/PPI_gcc.txt', './data/RodeEU_gcc.txt']
    for path in fileList:
        start_time = time.time()
        vertices, edges = loaddata(path)
        print('vertices: ',len(vertices),'edges: ',len(edges))
        minCutSet, minCut = globalMinCut(vertices,edges,len(vertices))
        print("the min-cut is ",minCut)
        minCutSet[0].sort()
        print("the nodes linked to V1 is ",minCutSet[0])
        time_dura = time.time() - start_time
        print("time duration : ",time_dura)
```

附录 3：Karger-Stein 代码

```
import random
import math
import copy
import time
import networkx as nx
import matplotlib.pyplot as plt

def drawNet(v,e):
    G = nx.Graph()
    G.add_nodes_from(v)
    G.add_edges_from(e)
    nx.draw_spectral(G)
    plt.show()

def ranContract(v, e, terminal):
```

```
    connect = []
    while len(v)>terminal:
        [a, b] = e.pop(random.randrange(0, len(e)))
        connect.append([a, b])
        v.remove(a)
        new = []
        for i in range(len(e)):
            if e[i][0] == a:
                e[i][0] = b
            elif e[i][1] == a:
                e[i][1] = b
            if e[i][0] != e[i][1]:
                new.append(e[i])
        e = new
    return (e, connect, v)

def findE(V1, e):
    connectToV1 = []
    new = []
    cur = [V1]
    while cur!=[]:
        for i in cur:
            for j in range(len(e)):
                if e[j][0]==i:
                    new.append(e[j][1])
                if e[j][1]==i:
                    new.append(e[j][0])
            e = [g for g in e if g[0] != i and g[1] != i]
        connectToV1+=new
        cur = new
        new = []
    return connectToV1

def loaddata(path):
    print("====================="+path+"====================")
    file = open(path)
    f = file.readlines()
    edges = []
    vertices = []
    for i in range(len(f)):
        dat = f[i].split()
        for i in dat:
            if int(i) not in vertices:
                vertices.append(int(i))
```

```
60.         if [int(dat[0]), int(dat[1])] not in edges and [int(dat[1]), int(d
    at[0])] not in edges:
61.             edges.append([int(dat[0]), int(dat[1])])
62.     print("vertices:", len(vertices), "edges:", len(edges))
63.     return vertices, edges
64.
65. def karger_stein(v,e):
66.     if len(v) >= 6:
67.         t = int(len(v)/math.sqrt(2)) + 1
68.         e_1, c_1, v_1 = ranContract(v, e, t)
69.         e_2, c_2, v_2 = ranContract(v, e, t)
70.         if len(e_1) < len(e_2):
71.             r,g,b = karger_stein(v_1, e_1)
72.         else:
73.             r,g,b = karger_stein(v_2, e_2)
74.         return r,g,b
75.     else:
76.         r,g,b = ranContract(v,e,2)
77.         return r,g,b
78.
79. if __name__ == '__main__':
80.     fileList = ['./data/BenchmarkNetwork.txt','./data/test1.txt','./data/C
    orruption_Gcc.txt', './data/Crime_Gcc.txt', './data/PPI_gcc.txt', './data/
    RodeEU_gcc.txt']
81.     times_for_k = []
82.     times_for_kg = []
83.     for fi in fileList:
84.         vertices, edges = loaddata(fi)
85.         drawNet(vertices, edges)
86.         numVer = len(vertices)
87.         min_ = 100000
88.         start_time = time.time()
89.         n = int(math.log(numVer)*math.log(numVer))
90.         for i in range(n):
91.             v = copy.deepcopy(vertices)
92.             e = copy.deepcopy(edges)
93.             r, g, b = ranContract(v, e, 2)
94.             if min_>len(r):
95.                 min_=len(r)
96.                 con = findE(b[1], g)
97.         time_dura = time.time() - start_time
98.         times_for_k.append(time_dura)
99.         print("time duration for karge : ",time_dura)
100.         print("the min-cut is ",min_)
```

```
101.        print("the min-cut contracted to V1 is \n",con)
102.
103.        min_ = 100000
104.        start_time = time.time()
105.        # n* (n-1) /2
106.        for i in range(n):
107.            v = copy.deepcopy(vertices)
108.            e = copy.deepcopy(edges)
109.            r, g, b = karger_stein(v, e)
110.            if min_>len(r):
111.                min_=len(r)
112.                con = findE(b[1], g)
113.        time_dura = time.time() - start_time
114.        times_for_kg.append(time_dura)
115.        print("time duration for karge-stein : ",time_dura)
116.        print("the min-cut is ",min_)
117.        print("the min-cut contracted to V1 is \n",con)
```