深圳大学研究生答题纸

专业 计算机科学与技术 成绩

课程名称、代码 并行算法 2703025 年级 2018 级

姓名 杨树 学号 1800271003 时间 2018 年 12 月

深圳大学研究生课程论文

题目 基于多线程的图割问题的算法设计与实现 成绩

专业 计算机科学与技术 课程名称、代码 并行算法 2703025

年　级 2018 级 姓　名 杨 树

学　号 1800271003 时　间 2018 年 12 月 27 日星期四

任课教师 廖 好

1. 背景介绍

1.1 问题描述

令$G=(V,E)$是一个有n个顶点和m条边的无向连通图，即$|V|=n$且$|E|=m$。

定义1 图G的**割**(cut)指的是将顶点集V划分成两个集合S和T，并且$T=V(G)\backslash S$，这个划分是通过对边的切割来实现的，这些被切割的边的集合为

$$(S,T)=\{uv\,|\,u\in S,v\in T,S\cap T=V(G),uv\in E(G)\}$$

其中集合S和集合T均非空，即$S\neq\varnothing$且$T\neq\varnothing$。

最小割$(Min-Cut)$则是指集合(S,T)中的元素个数最小化，如果图G是带权图，则是指集合(S,T)中的元素带有权值的和最小化。我们约定，不带权的无向连通图中的割的代价是集合(S,T)中的元素的个数；带权的无向连通图中的割的代价是集合(S,T)中的元素带有权值的和。求最小割也就相当于最小化割的代价，将这个最优解记为(S^*,T^*)。

如下图 1 和图 2 所示，图 1 展示的是不带权重的无向连通图的最小割，此时割的代价为 1；图 2 展示的是带权重的无向连通图的最小割，此时割的代价为 15。

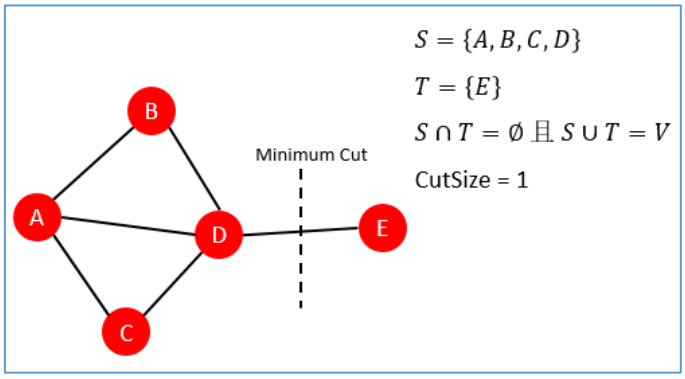

图 1　不带权重的无向连通图的最小割

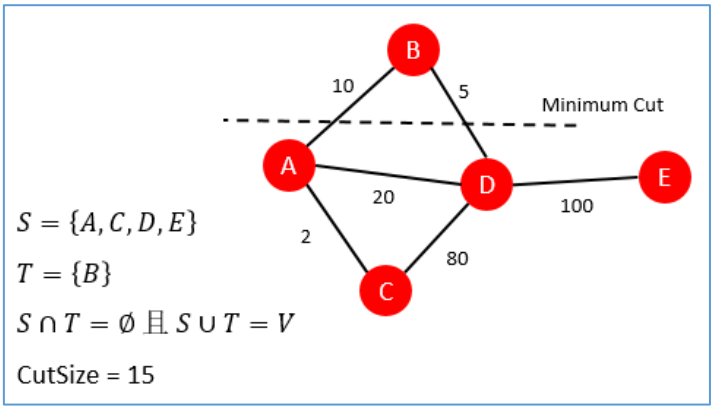

图 2　带权重的无向连通图的最小割

Picard 和 Queyranne 于 1982 年介绍了最小割算法的一些应用[6]。文中介绍了最小割在组合优化问题及其各种变体的问题上的应用，包括线性和非线性的整数规划、图论、排序和调度等问题。除了 Picard 和 Queyranne 的介绍之外，图的最小割问题还有很多其他方面的应用。

图的最小割问题在网络设计和网络可靠性问题中经常出现确定网络连接性的问题[7]。在信息检索领域，最小割已被用于识别超文本系统中相关文档的聚类[8]。图的最小割在大规模组合优化（large-scale combinatorial optimization）中也起着重要作用。

1.2 研究现状

到目前为止，有几种不同的方法用于求解图的最小割这一问题。早期用于求解图的最小割问题主要是靠它的对偶问题——最大流问题[1]。Ford 和 Fulkerson 给出了最大流问题和最小割问题可相互转化的定理及相关的证明[1]。

在给定源点（source）和汇点（sink）的前提下，有许多用于求解最大流问题的算法，其中比较好的一种算法是 Goldberg 和 Tarjan 于 1988 年提出的推送-重贴标签算法[2]，该算法的时间复杂度为$\mathrm{O}\left(n^2m\right)$（$n$ 是图G 的顶点数，m 是图G 的边的数目）。要使用最大流算法，就要给出源点和汇点。而对于源点和汇点的选择，可以遍历所有的顶点对，这个组合数为$\binom{n}{2}=\frac{n\left(n-1\right)}{2}$，所以通过最大流算法来求解最小割的算法时间复杂度为$\mathrm{O}\left(n^4m\right)$。如果图$G$ 是稠密的，即有$m\approx n^2$，那么通过最大流算法来求解最小割的算法时间复杂度为$\mathrm{O}\left(n^6\right)$。

除了上边介绍的利用基于最大流的算法来求解最小割问题，还有另外一类算法，这些算法不需要将最小割问题转化成最大流问题而是直接进行求解。比较著名的有 Karger 于 1993 年提出的基于边的收缩的算法[3]。几年之后在 Karger 算法的基础上，Karger 和 Stein 合作对该算法进行了改进提出了 Karger-Stein 算法[4]。

2. Karger 算法

Karger 于 1993 年发表论文提出了一种新的算法来求解最小割问题[3]。Karger 算法是 David Karger 在斯坦福大学读博的时候创造出来的。Karger 算法是一种求解无向连通图G 的最小割的随机算法（Randomized Algorithm）。

该算法的思想基于无向图中的边的收缩这一概念。Karger 算法通过不断迭代地收缩随机选择的边，即把两个有边相连的顶点合并成一个超顶点，直到只剩下两个超顶点；这两个超顶点代表原来图中的一个切割。通过迭代该基本算法足够的次数，可以高概率地找到图的最小割。因此，它可以被视为蒙特卡罗方法，是一种随机算法。

2.1 收缩算法

收缩算法在原论文[1]中的用语是“The Contraction Algorithm”，该算法也是 Karger 算法的基础，下边给出“边的收缩”的定义。

定义3 在图G中，顶点u、v之间的边e的**收缩（Contraction）**是指用单个超顶点来代替顶点u和v，原来连接到u或v的边均连接到该超顶点上（边e除外，边e从图中删除）。边e收缩之后的图的边的集合变成$E(G)\backslash e$，比原来的图G少了一条边。

如下图 3 所示，对顶点 BD 之间的边进行收缩。

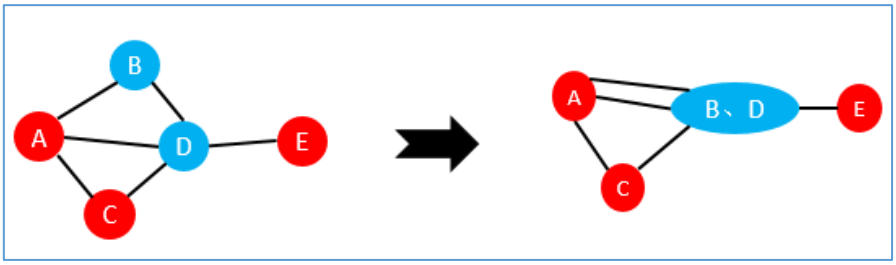

图 3　边 BD 的收缩

Karger 则是利用了图中边的收缩这一概念来实现割的求解。具体的收缩算法的伪代码如下表 1 所示。

表 1　收缩算法

```
MinCut ( G )：
    While |V| > 2：
        随机选择一条边；
        收缩这条边（即合并边两端的顶点为超顶点）；
    End While
```

如下图 4 所示，展示了表 1 中收缩算法的具体过程。在每次循环过程中，首先随机选择一条边，将这条边收缩、边两端的顶点进行合并。当算法执行到图中只有两个超顶点时，算法终止。如果边不带权重的话，那么此时连接两个超顶点的边的数量即为割的代价；如果带有权重的话，此时连接两个超顶点的边的权重之和即为割的代价。图 4 所示的算法执行完成时割的代价为连接$\{A,B,C,D\}$和$\{E\}$两个顶点集合的边的数量，即为 1。

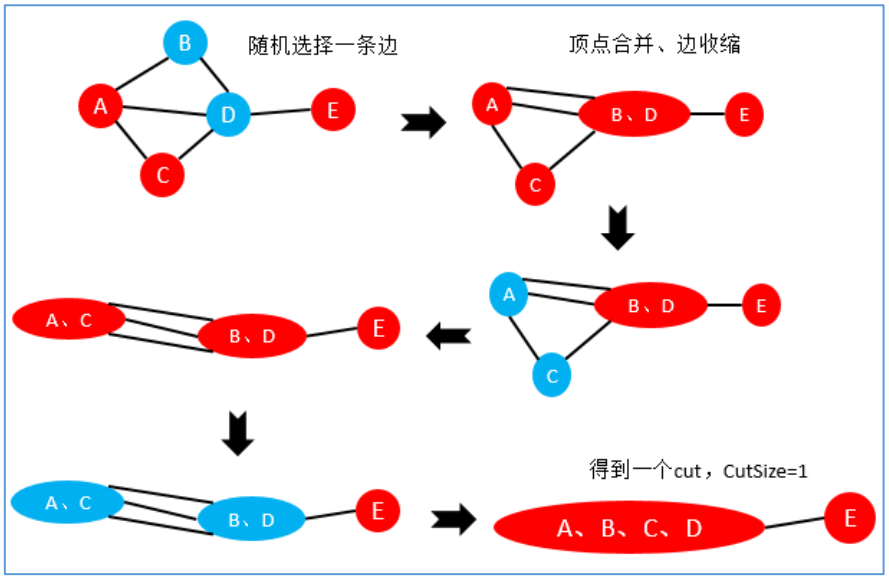

图 4　收缩算法的执行

不难看出，表 1 中的算法经过$n-2$次循环即可运行完成（n 是图G 的顶点个数）。由于这个算法是随机算法，所以不保证执行得到的结果一定是我们要求的最小割。但是我们如果将该算法重复执行一定的次数，就可以将算法失败的概率降到很小，这样我们就能以高概率来求得图G 的最小割。另外，不难得到，表 1 中的收缩算法的时间复杂度为$\mathrm{O}\left(n^2\right)$。因为每次合并顶点时，要遍历所有与这两个顶点相连的边并将这些边连接到合并后的超顶点上，而算法总共要循环$n-2$次，表 1 中的收缩算法的时间复杂度为$\mathrm{O}\left(n^2\right)$。

2.2 算法分析

下边通过给出几个引理及相关的证明来对 Karger 算法的时间复杂度进行分析。

引理 1 对于图G（有n 个顶点和m 条边）来说，随机选取一条边，这条边是属于集合$\left(S^*,T^*\right)$的概率至多为$\frac{2}{n}$。

证明：若集合$\left(S^*,T^*\right)$中的元素个数为k，则图G 中的边的数量至少为$\frac{kn}{2}$。否则利用鸽巢原理，总存在一个顶点v，与之相连的边的数量小于k。那么只需要令$S=\{v\}$且有$T=V(G)\setminus S$，则此时集合(S,T)中的元素个数小于k，这与集合$\left(S^*,T^*\right)$中的元素个数为k矛盾，所以若集合$\left(S^*,T^*\right)$中的元素个数为k，则图G 中的边的数量至少为$\frac{kn}{2}$。此时若

随机选取一条边，则这条边是属于集合$\left(S^*,T^*\right)$的概率小于$\frac{k}{kn/2}=\frac{2}{n}$，得证。

引理 2 对于图G（有n个顶点和m条边）来说，表 1 中的收缩算法执行完成时找到的割是最小割的概率$P\geq\frac{2}{n(n-1)}$。

证明：通过引理 1 可知，对于顶点数为n的图G，随机选取一条边，这条边属于集合$\left(S^*,T^*\right)$的概率至多为$\frac{2}{n}$。那么收缩算法执行完成时找到的割是最小割的概率为

$$P\geq\left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\cdots\left(1-\frac{2}{3}\right)=\frac{n-2}{n}\frac{n-3}{n-1}\cdots\frac{1}{3}=\frac{2}{n(n-1)}$$

得证。

引理 3 对于图G（有n个顶点和m条边）来说，将表 1 中的收缩算法重复执行$\binom{n}{2}$次，可以将表 1 中的收缩算法失败（即得到的不是图G的全局最小割）的概率降至常数$\frac{1}{e}$以下；将表 1 中的收缩算法重复执行$\binom{n}{2}\ln n$次，即可将表 1 中的收缩算法失败的概率降至$\frac{1}{n}$以下。

证明：通过引理 2 可知，对于顶点数为n的图G，表 1 中的收缩算法执行完成时找到的割是最小割的概率$P\geq\frac{2}{n(n-1)}$，则找到的不是最小割的概率小于$1-\frac{2}{n(n-1)}$。重复执行表 1 中的收缩算法$\binom{n}{2}$次，那么每次都找不到最小割的概率为

$$\left(1-\frac{2}{n(n-1)}\right)^{\binom{n}{2}}=\left(1-\frac{2}{n(n-1)}\right)^{\frac{n(n-1)}{2}}\leq e^{-1}=\frac{1}{e}$$

重复执行表 1 中的收缩算法$\binom{n}{2}\ln n$次，那么每次都找不到最小割的概率为

$$\left(1-\frac{2}{n(n-1)}\right)^{\binom{n}{2}\ln n}=\left(1-\frac{2}{n(n-1)}\right)^{\frac{n(n-1)}{2}\ln n}\leq e^{-\ln n}=\frac{1}{n}$$

所以重复执行$\binom{n}{2}$次，算法失败的概率小于常数$\frac{1}{e}$；重复执行$\binom{n}{2}\ln n$次，算法失败的概率

小于$\frac{1}{n}$，得证。

通过引理 3 可知，将表 1 中的收缩算法重复执行$\binom{n}{2}\ln n$次，即可将表 1 中的收缩算法失败的概率降至$\frac{1}{n}$以下，这样可以高概率地得到图G的最小割。又因为表 1 中的收缩算法的时间复杂度为$\mathrm{O}\left(n^2\right)$，所以总的时间复杂度为$\mathrm{O}\left(n^4 \ln n\right)$。

3. Karger-Stein 算法

David Karger 和 Clifford Stein 对 Karger 算法做了进一步的改进创造了 Karger-Stein 算法并于 1996 年将该算法发表[4]。

3.1 递归的收缩算法

递归的收缩算法在原论文中的用语是“Recursive Contraction Algorithm”。这个算法和表 1 中的收缩算法最大的区别是该算法执行边的收缩，直到图G中只剩下$\left\lceil 1+n/\sqrt{2} \right\rceil$个超顶点。在正式介绍 Karger-Stein 算法之前，先仿照引理 2 来给出下边的引理 4。

引理 4 对于图G（有n个顶点和m条边）来说，表 1 中的收缩算法执行直到图G中只剩下$\left\lceil 1+\frac{n}{\sqrt{2}} \right\rceil$个超顶点时，即算法循环了$l=n-\left\lceil 1+\frac{n}{\sqrt{2}} \right\rceil$次，找到的边均不属于集合$(S^*,T^*)$（即不属于最小割所切断的边）的概率$P \geq \frac{(n-l)(n-l-1)}{n(n-1)}$。

证明：对于图G（有n个顶点和m条边）来说，表 1 中的收缩算法执行直到图G中只剩下$\left\lceil 1+\frac{n}{\sqrt{2}} \right\rceil$个超顶点时，算法循环了$l=n-\left\lceil 1+\frac{n}{\sqrt{2}} \right\rceil$次。所以有

$$P \geq \prod_{i=0}^{l-1}\left(1-\frac{2}{n-i}\right) \geq \left(1-\frac{2}{n}\right)\left(1-\frac{2}{n-1}\right)\cdots\left(1-\frac{2}{n-l+1}\right) = \frac{n-2}{n}\frac{n-3}{n-1}\cdots = \frac{(n-l)(n-l-1)}{n(n-1)}$$

得证。

Karger 和 Stein 指出，当图G利用表 1 中的算法执行时，随着图中顶点数越来越少，随机选取的边属于集合(S^*,T^*)的概率会逐渐增大，借助于引理 1 可知，即

$\frac{2}{n-i}\left(0 \le i \le n-3\right)$随着$i$的增大而增大。所以Karger和Stein对表1中的算法做出了改进，给出了如下表2所示的改进的收缩算法。

表2　改进的收缩算法

```
MinCut ( G，t )：
    While |V| > t：
        随机选择一条边；
        收缩这条边（即合并边两端的顶点为超顶点）；
    End While
    Return G
```

对于表2所示的改进的收缩算法，当$t=\frac{n}{\sqrt{2}}$时，算法循环了$l=n-\left\lceil 1+\frac{n}{\sqrt{2}}\right\rceil$次，找到的边均不属于集合$(S^*,T^*)$（即不属于最小割所切断的边）的概率大于$\frac{1}{2}$。

引理5 对于图G（有n个顶点和m条边）来说，当$t=\frac{n}{\sqrt{2}}$时，表2中的改进的收缩算法执行结束时找到的边均不属于集合(S^*,T^*)的概率大于$\frac{1}{2}$。

证明：对于图G来说，当$t=\frac{n}{\sqrt{2}}$时，表2中的改进的收缩算法执行结束时，算法共循环了$l=n-\left\lceil 1+\frac{n}{\sqrt{2}}\right\rceil$次，通过引理4可知找到的边均不属于集合$(S^*,T^*)$的概率为$P\ge\frac{(n-l)(n-l-1)}{n(n-1)}$，所以有

$$P\ge\frac{(n-l)(n-l-1)}{n(n-1)}=\frac{\left(\left\lceil 1+n/\sqrt{2}\right\rceil\right)\left(\left\lceil 1+n/\sqrt{2}\right\rceil-1\right)}{n(n-1)}$$

$$=\frac{1}{2}\times\frac{\left(\left\lceil \sqrt{2}+n\right\rceil\right)\left(\left\lceil \sqrt{2}+n\right\rceil-1\right)}{n(n-1)}>\frac{1}{2}$$

即找到的边均不属于集合(S^*,T^*)的概率大于$\frac{1}{2}$，得证。

下边给出递归的收缩算法的伪代码。

表 3　递归的收缩算法

```
FastMinCut ( G )：
    If |V| < 6：
        Return MinCut(G, 2)
    Else
        t = 1 + n / √2
        Graph_1 = MinCut(G, t)
        Graph_2 = MinCut(G, t)
        Return min( FastMinCut(Graph_1), FastMinCut(Graph_1) )
```

3.2 算法分析

下边通过给出几个引理及相关的证明来对 Karger-Stein 算法的时间复杂度进行分析。

引理 6 对于图G（有n个顶点和m条边）来说，表 3 所示的递归的收缩算法 FastMinCut 的时间复杂度为$\mathrm{O}\left(n^2 \log n\right)$。

证明：令算法 FastMinCut 的时间复杂度为$T\left(n\right)$。对于图G来说，算法 FastMinCut 两次调用 MinCut(G, t) 的时间开销为$\mathrm{O}\left(n^2\right)$。在得到两个函数调用返回的结果后，算法执行两个递归调用，所以有

$$T\left(n\right)=\mathrm{O}\left(n^2\right)+2T\left(\frac{n}{\sqrt{2}}\right)$$

求解该递归公式，即可得到算法 FastMinCut 的时间复杂度为$\mathrm{O}\left(n^2 \log n\right)$，得证。

引理 7 对于图G（有n个顶点和m条边）来说，表 3 所示的递归的收缩算法 FastMinCut 执行完成时找到的割是最小割的概率为$\Omega\left(\frac{1}{\log n}\right)$。

证明：令$P\left(n\right)$表示对于有n个顶点的图G来说算法成功的概率。首先来看表 3 中所示的图 Graph_1。由引理 5 可知，通过调用 MinCut(G, t) 得到 Graph_1 的过程中没有选取属于集合(S^*,T^*)的边的概率大于$\frac{1}{2}$。之后要将图 Graph_1 传入 FastMinCut 算法递归调用，这部分算法的成功的概率$P\left(\frac{n}{\sqrt{2}}\right)$，所以通过图 Graph_1 找到的割是最小割的概率大于$\frac{1}{2}P\left(\frac{n}{\sqrt{2}}\right)$，则通过图 Graph_1 找到的割不是最小割（算法失败）的概率不大于$1-\frac{1}{2}P\left(\frac{n}{\sqrt{2}}\right)$。

所以通过图 Graph_1 和图 Graph_2 都没有找到最小割的概率小于$\left(1-\frac{1}{2}P\left(\frac{n}{\sqrt{2}}\right)\right)^2$。所以 FastMinCut 算法成功的概率为

$$P(n)\geq 1-\left(1-\frac{1}{2}P\left(\frac{n}{\sqrt{2}}\right)\right)^2=P\left(\frac{n}{\sqrt{2}}\right)-\frac{1}{4}\left(P\left(\frac{n}{\sqrt{2}}\right)\right)^2$$

下边需要对这个递归公式进行求解。在正式求解之前先进行变量的替换，引入记号 p_k，令 p_k 表示这个递归的第 k 层的概率，并且这个递归树的叶子为第 0 层。当 $n\leq 6$ 时，由引理 2 可知有 $p_0=\frac{1}{15}$；当 $n\geq 7$ 时，有 $p_{k+1}\geq p_k-\frac{1}{4}p_k^2$。记 $z_k=\frac{4}{p_k}-1$，则 $p_k=\frac{4}{z_k+1}$。将其带入 $p_{k+1}\geq p_k-\frac{1}{4}p_k^2$ 则有 $z_0=59$ 和 $z_{k+1}=z_k+1+\frac{1}{z_k}$。考虑最坏的情况（概率最小的时候，即不等式取等号）继续推导有

$$\begin{cases} z_0=59 \\ z_{k+1}=z_k+1+\frac{1}{z_k} \end{cases}\Rightarrow\begin{cases} z_k>z_{k-1}+1>k \\ z_k<z_{k-1}+2<59+2k \end{cases}$$

即 $k<z_k<59+2k$。所以 $z_k=\Theta(k)$，$p_k=\frac{4}{z_k+1}=\Theta\left(\frac{1}{k}\right)$。注意到此时 k 表示的是递归树的深度，所以 $k=\log_{\sqrt{2}}n=2\log_2 n$，故有 $p_k=\Theta\left(\frac{1}{\log n}\right)$。由于这个推导考虑的是最坏的情况，所以有 $P(n)\geq p_k=\Theta\left(\frac{1}{\log n}\right)$。所以表 3 所示的递归的收缩算法 FastMinCut 执行完成时找到的割是最小割的概率为 $\Omega\left(\frac{1}{\log n}\right)$（指出了概率的下界）。得证。

根据引理 7 给出的概率下界，可以得到，将算法 FastMinCut 重复执行 $O\left(\log^2 n\right)$ 次时，算法 FastMinCut 失败的概率小于 $O\left(\frac{1}{n}\right)$。

引理 8 对于图 G（有 n 个顶点和 m 条边）来说，表 3 所示的算法 FastMinCut 执行的时间复杂度为 $O\left(n^2\log^3 n\right)$ 时，可以高概率地找到最小割。

证明：由引理 7 及其证明可以知道 $P(n)=O\left(\frac{1}{\log n}\right)$。因此将该算法重复运行

$O\left(\log^2 n\right)$次，算法失败（找不到最小割）的概率为

$$\left(1-P\left(n\right)\right)^{O\left(\log^2 n\right)} \le \left(1-\frac{c}{\log n}\right)^{\frac{3\log^2 n}{c}} \le e^{-3\log n} = \frac{1}{n^3}$$

其中c是常数。在有着n个顶点的图中，最多有$\binom{n}{2}$个最小割[5]。综合以上，算法找不到所有最小割的概率为$\left(1-P\left(n\right)\right)^{O\left(\log^2 n\right)} \le \binom{n}{2}\frac{1}{n^3} = O\left(\frac{1}{n}\right)$。根据引理 6，表 3 所示的算法 FastMinCut 的时间复杂度为$O\left(n^2 \log n\right)$，所以总的时间复杂度为$O\left(n^2 \log^3 n\right)$，得证。

4. Linux 下的多线程实现

通过利用多线程，来并行化执行图的最小割算法，从而加快找到最小割的速度。如下图 5 所示为四线程执行 Karger 算法的示意图。

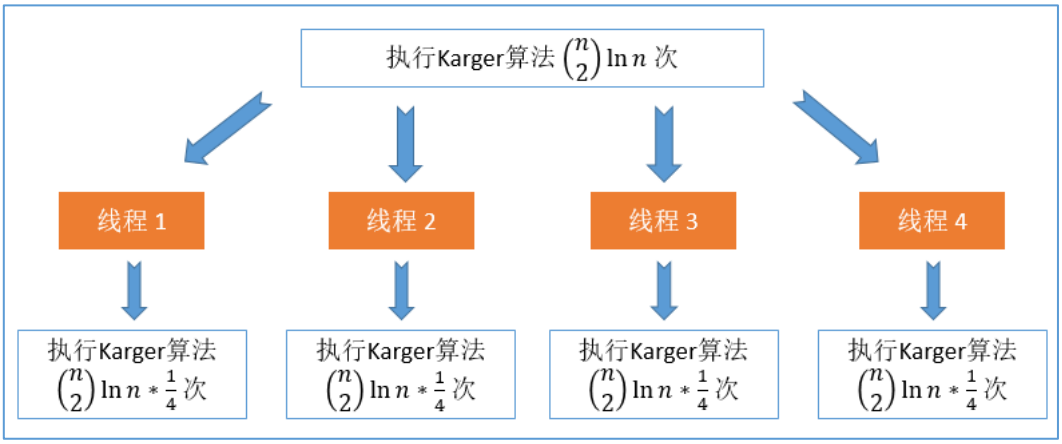

图 5　四线程执行 Karger 算法

注意到 FastMinCut 算法中连续两次调用 MinCut(G, t)算法，得到的 graph_1 和 graph_2 相互独立，所以可以使用多线程并行执行，从而加快执行速度，如图 6 所示。图 7 展示的两个线程执行 FastMinCut 算法的示意图。

```
FastMinCut (G) {
   If |V| < 6
      return minCut (graph, 2)
   Else
      t = 1 + |V|/√2
      graph_1 = minCut (graph, t)
      graph_2 = minCut (graph, t)
      return min (FastMinCut(graph_1), FastMinCut(graph_2))
}
```

图 6　FastMinCut 算法

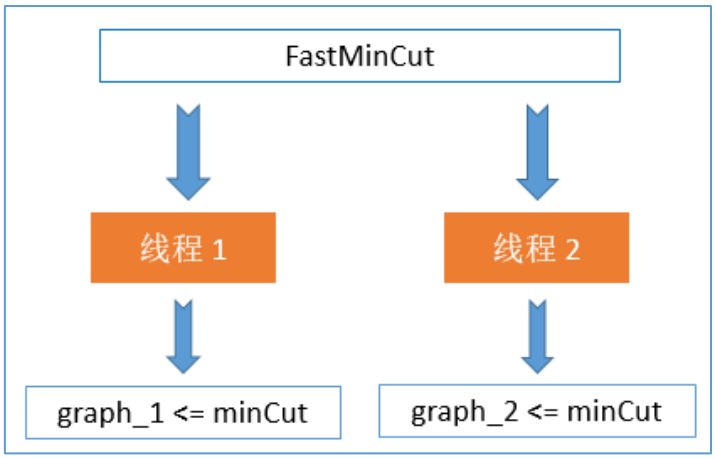

图 7　两线程执行 FastMinCut 算法

下边图 8 展示的是我定义的 C++类 Graph，类中实现了许多方法。count_vertices 函数用于记录图中的超顶点的数目，count_edges 函数用于统计任意时刻的图中边的数量，count_Weight 函数用于统计任意时刻的图中边的权值和。merge_vertices 函数用于收缩边、合并边两端的顶点。calculateVerSet 则用于计算程序结束后的两个集合的顶点编号，即图的最小割所分成的两个顶点集合，并将它们存放到 std::set<size_t>类型的两个集合中。

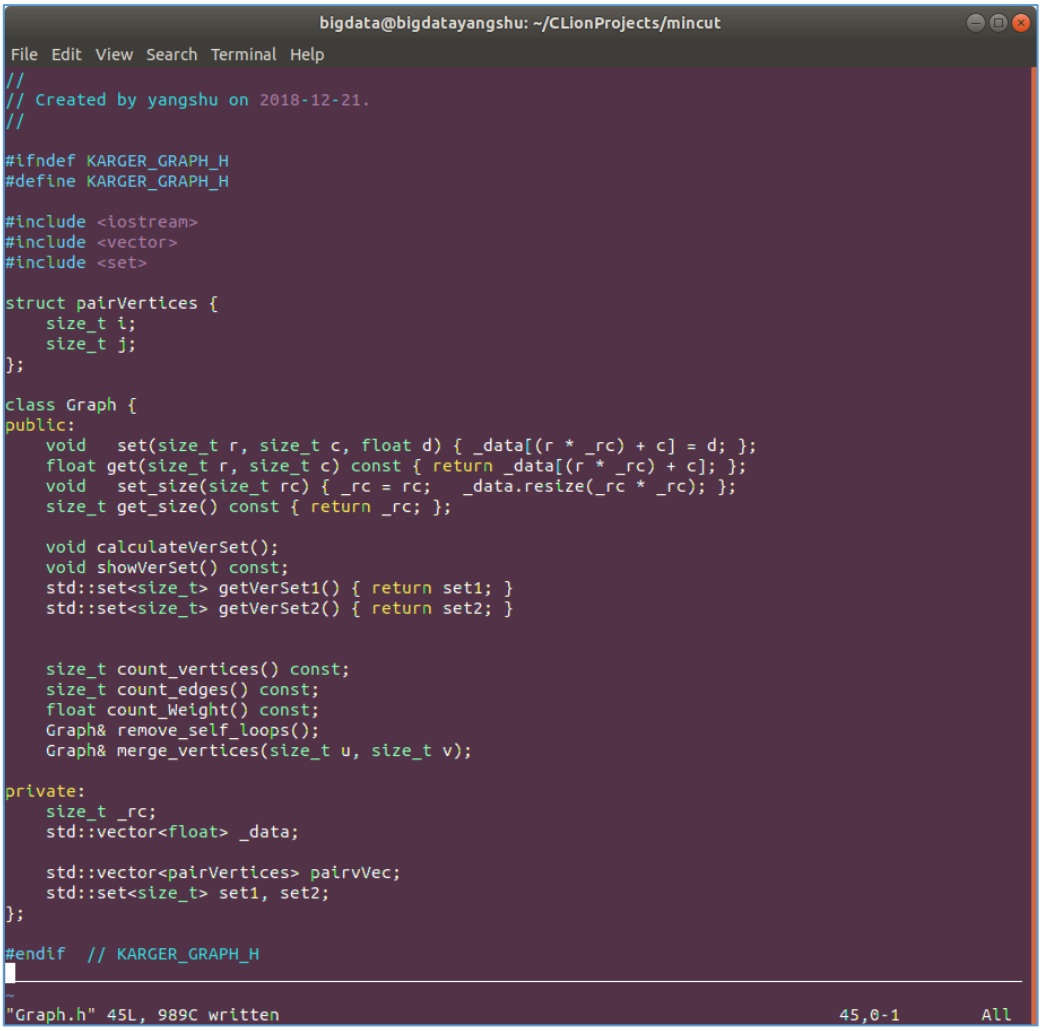

图 8　Graph 类的变量及方法

5. 实验和验证

5.1 实验环境

操作系统：Ubuntu 18.04
硬件环境：Core i5

5.2 实验设置

实验所测试的图是有 100 个节点 4950 条边的图。之所以采用这个规模的图，是因为 Karger-Stein 算法在递归的过程中，需要大量的内存空间，而电脑的存储空间有限，规模再大就很容易造成程序中断，无法进行测试。而原论文中也提到，该算法的空间复杂度较高，故本次实验采用了这个规模的图。

如下图 8 所示，即为程序运行时的内存占用情况。

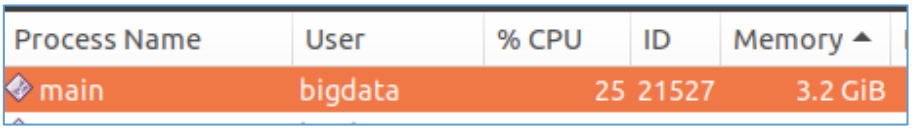

图 8　资源占用情况

5.3 实验结果

如图 9 所示，为程序运行结果示意图。

```
bigdata@bigdatayangshu:~/CLionProjects/mincut$ g++ -pthread -o main main.cpp Graph.h Graph.cpp
bigdata@bigdatayangshu:~/CLionProjects/mincut$ ./main
Input vertex count: 100
Input edge count: 4950
Already ...
Karger Algorithm Thread: 48.3735
Karger Algorithm: 155.155
KargerStein Algorithm Thread: 27.4016
KargerStein Algorithm: 85.0143
Done ...
bigdata@bigdatayangshu:~/CLionProjects/mincut$
```

图 9　程序运行结果示意图

将该程序运行 20 次得到表 4 所示的实验数据。

表 4　运行 20 次实验数据

	Karger(thread)	**Karger**	**KargerStein(thread)**	**KargerStein**
	48.3914	154.897	26.7138	86.128
	48.0125	154.902	27.1108	85.0374
	48.1394	153.836	26.8639	85.3036
	48.558	154.806	26.6991	85.0186
	47.9599	153.643	27.3868	85.2816
	48.2641	154.674	27.3889	84.9608
	48.1698	153.855	26.9874	85.3864
	48.2732	155.21	27.1037	84.9821
	48.175	153.6	26.9625	85.3934
	48.6327	153.737	27.0503	85.3875
	48.2089	154.323	27.3572	84.9756
	48.2128	153.756	27.1094	85.518
	48.3887	154.947	27.3188	85.0701
	48.1313	153.926	26.6293	85.4105
	47.8396	154.483	26.6344	85.0946
	48.3967	154.714	27.4081	85.198
	48.1814	155.186	26.9225	85.0852
	47.9961	154.818	27.1196	85.0964
	48.3203	153.799	26.7176	85.5111
	48.1705	154.186	26.7616	85.3008
平均值	**48.22112**	**154.3649**	**27.01229**	**85.25699**

6. 结论与收获

从表 4 中可以看出 KargerStein 算法要比 Karger 算法快，这与我们前边分析的时间复杂度的结果是一致的（KargerStein 的时间复杂度比 Karger 算法小）。使用多线程实现之后，两种算法都有不同程度的加速。

经过本学期的并行算法的学习，不仅对并行计算机有了一定的了解，而且对并行编程方面也学到了很多，另外，也是上了这门课程，让我对 NP 和 NPC 这类易混淆的概念有了清楚的了解。也是利用这门课的入门介绍，我自学了 MapReduce 以及 PThread 的使用，收获颇丰。在以后的科研当中，我会充分利用所学知识对算法进行优化，毕竟并行计算要比串行快很多。

参考文献

[1]. Lester R Ford and Delbert R Fulkerson. Maximal flow through a network. Canadian journal of Mathematics, 8(3):399-404, 1956.

[2]. Andrew V Goldberg and Robert E Tarjan. A new approach to the maximum-flow problem. Journal of the ACM (JACM), 35(4):921-940, 1988.

[3]. David R Karger. Global min-cuts in rnc, and other ramifications of a simple min-out algorithm. In Proceedings of the fourth Annual ACM-SIAM Symposium on Discrete Algorithms, 1993:21-30.

[4]. David R Karger and Clifford Stein. A new approach to the minimum cut problem. Journal of the ACM (JACM), 43(4):601-640, 1996.

[5]. Efim A. Dinitz, A. V. Karzanov, and Micael V. Lomonosov. On the structure of a family of minimum weighted cuts in a graph. In A. A. Fridman, editor, Studies in Discrete Optimization, pages 290-306. Nauka Publishers, 1976.

[6]. Jean C Picard and Maurice Queyranne. Selected applications of minimum cuts in networks. INFOR: Information Systems and Operational Research, 20:394-422, 1982.

[7]. Charles J. Colbourn. The Combinatorics of Network Reliability. New York: Oxford University Press, 1987.

[8]. Rodrigo A. Botafogo. Cluster analysis for hypertext systems. In Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 116-125, 1993.

[9]. https://en.wikipedia.org/wiki/Cut_(graph_theory)

[10]. https://en.wikipedia.org/wiki/Flow_network