随机化算法简介*

周源
清华大学计算机科学与技术系

2009年1月17日

1 为什么要讲随机算法

与竞赛相关的目的：

- 试图给竞赛中常用的“随机调整”等方法以理论证明：它们不一定只是不会解题时“无奈的选择”，也许就是一个简单、正确、高效的算法。
- 增强理论修养。

与算法设计相关的目的：

- 有时随机化可以帮助我们避免最坏情况（如随机化的快速排序）。
- 有时随机化可以帮助我们解决一些我们不知道是否存在高效确定性算法的问题（如质数判定）。

在这个短短的讲座中，我们将涉及

- 利用随机化设计算法的技巧。
- 分析随机算法的技巧。
- 一些简单、漂亮的随机化算法。

2 概率基础

定义 2.1 (概率和期望的符号) 首先定义两个基本的记号如下，

- 使用$\mathbb{P}[A]$ 表示事件A 发生的概率。
- 使用$\mathbb{E}[X]$ 表示随机变量X 的数学期望。

*全国信息学奥林匹克冬令营讲义

注释 2.2 在离散的概率空间Ω中，

$$\mathbb{E}[X] = \sum_{x\in\Omega} x\mathbb{P}[X = x]$$

如投掷一个普通的6面骰子，骰子朝向上的面上的数字X的数学期望是

$$\mathbb{E}[X] = \frac{1+2+3+4+5+6}{6} = 3.5$$

注释 2.3 在连续的概率空间Ω中，

$$\mathbb{E}[X] = \int_{x\in\Omega} x\mathrm{d}\mathbb{P}[X = x]$$

如随机变量X 以均匀的概率出现在$[0, 100]$ 中，则

$$\begin{aligned}\mathbb{E}[X] &= \int_0^{100} x\mathrm{d}\Big(\frac{x}{100}\Big)\\ &= 50\end{aligned}$$

定理 2.4 (Union Bound) 令$A_1, A_2, \cdots, A_n$是一列事件，那么它们中至少有一个发生的概率不大于它们中每一个单独发生的概率之和。即

$$\mathbb{P}[\cup_{i=1}^n A_i] \le \sum_{i=1}^n \mathbb{P}[A_i]$$

定理 2.5 (期望的线性性) 对于任意随机变量X和Y，任意常数a和b，有，

- $\mathbb{E}[aX + b] = a\mathbb{E}[X] + b$,
- $\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$。

以上两个定理略去证明。

定义 2.6 (随机变量的方差和标准差) 定义

$$\mathrm{Var}[X] = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - (\mathbb{E}[X])^2 \ge 0$$

为随机变量X 的方差。同时定义

$$\sigma = \sqrt{\mathrm{Var}[X]}$$

为随机变量X 的标准差。

定理 2.7 (Markov (马尔科夫) 不等式) 如果随机变量$X \ge 0$，则对于任意$a > 0$，有，

$$\mathbb{P}[X \ge a] \le \frac{\mathbb{E}[X]}{a}$$

证明(定理2.7) 反设$\mathbb{P}[X \geq a] > \frac{\mathbb{E}[X]}{a}$，则

$$\begin{aligned}
\mathbb{E}[X] &= \sum_x x\mathbb{P}[X = x] \\
&\geq \sum_{x \geq a} x\mathbb{P}[X = x] \\
&\geq a\sum_{x \geq a} \mathbb{P}[X = x] \\
&= a\mathbb{P}[X \geq a] \\
&> a \cdot \frac{\mathbb{E}[X]}{a} \\
&= \mathbb{E}[X]
\end{aligned}$$

矛盾。 ■

定理 2.8 (Chebyshev (契比雪夫) 不等式) 对于任何随机变量X和常数$a > 0$，有

$$\mathbb{P}[|X - \mathbb{E}[X]| \geq a] \leq \frac{\mathrm{Var}[X]}{a^2}$$

证明(定理2.8)

$$\begin{aligned}
\mathbb{P}[|X - \mathbb{E}[X]| \geq a] &= \mathbb{P}[(X - \mathbb{E}[X])^2 \geq a^2] \\
&\leq \frac{\mathbb{E}[(X - \mathbb{E}[X])^2]}{a^2} \\
&= \frac{\mathrm{Var}[X]}{a^2}
\end{aligned}$$

其中的“$\leq$”是利用了Markov 不等式(定理2.7)。 ■

3 最大割：近似算法

定义 3.1 (割和割的容量) 给定一个加（正）权无向图$G = (V, E, w)$，任何一个关于顶点集合V的划分$[A, B]$ *($V = A \cup B, B = V - A, A \neq \emptyset, B \neq \emptyset$)*称为该图的一个割。

对于任何$[A, B]$，定义割的容量

$$C[A, B] = \sum_{(u,v) \in E, u \in A, v \in B} w(u, v)$$

定义 3.2 (最大割) 定义G中的最大割为所有割$[A, B]$中容量最大者。即

$$\mathrm{MaxCut}(G) = \max_{[A,B]}\{C[A, B]\}$$

注释 3.3 *最大割问题是NP-Hard, 即只要$P \neq NP$, 没有有效的多项式时间算法解决最大割问题。*

下面我们要给出一个简单的随机（多项式时间）算法，保证得到容量至少是最大割的一半的割。即假设C^*是最大割的容量，而C是算法返回的割的容量。那么

$$C \geq \frac{1}{2}C^*$$

也称这是一个最大割的2-近似算法。

算法：随机返回一个割。即假设输出的割是$[A, B]$，对于图中的每一个顶点，以1/2的概率将其放入A中，以另外1/2的概率将其放入B中。

定理 3.4

$$\mathbb{E}[C[A, B]] = \frac{1}{2}\sum_{e \in E} w(e) \geq \frac{1}{2}C^*$$

证明 (定理3.4) 根据定义3.1，

$$\begin{aligned} C[A, B] &= \sum_{(u,v)\in E, u\in A, v\in B} w(u, v) \\ &= \sum_{(u,v)\in E} w(u, v)\mathbf{1}_{u\in A, v\in B} \end{aligned}$$

再根据期望的线性性(定理2.5)，

$$\begin{aligned} \mathbb{E}[C[A, B]] &= \sum_{(u,v)\in E, u\in A, v\in B} w(u, v)\mathbb{P}[u \in A, v \in B] \\ &= \sum_{(u,v)\in E, u\in A, v\in B} w(u, v) \cdot \frac{1}{2} \\ &= \frac{1}{2}\sum_{e\in E} w(e) \\ &\geq \frac{1}{2}C^* \end{aligned}$$

■

但是运行一次本算法，并不能保证一定能够得到一个2-近似的解。我们将算法改写如下。

保证得到2-近似解的算法：不停的运行原先的最大割算法，每次得到一个割$[A, B]$，判断是否有$C[A, B] \geq \frac{1}{2}\sum_{e\in E} w(e)$，如果满足，则返回$[A, B]$，否则继续循环。

根据Markov 不等式(定理2.7)，

$$\begin{aligned}\mathbb{P}[C[A,B] < \frac{1}{2}\sum_{e\in E} w(e)] &= \mathbb{P}[\sum_{e\in E} w(e) - C[A,B] > \frac{1}{2}\sum_{e\in E} w(e)] \\ &\leq \frac{\mathbb{E}[\sum_{e\in E} w(e) - C[A,B]]}{\frac{1}{2}\sum_{e\in E}} \\ &= \frac{1}{2}\end{aligned}$$

即每执行一次循环体，有至少1/2的概率返回。这样算法期望执行2次循环体。而循环体内时间复杂度为$O(m)$。这样算法的期望时间复杂度也是$O(m)$。

4 随机化的快速排序的时间复杂度

快速排序：需要将$S_1, S_2, \cdots, S_n$进行排序。

- 假设在每次快速扫描(Quick Pass) 过程中，都能取到中位数作为分裂点，则快速排序的时间复杂度

$$T(n) = 2T(n/2) + O(n) = O(n\log n)$$

- 精确的取到中位数很难，但是“几乎精确”的中位数也是可以的。
- 随机的取一个元素，“很可能”这个元素和中位数相差不远。

分析：
希望证明期望的比较次数比C小，于是，

- $X_{ij} = 1$ 当S_i 和S_j有过比较（显然最多只会比一次）。
- 根据**定理2.5 (期望的线性性)**，$\mathbb{E}[C] = \sum \mathbb{E}[X_{ij}]$。
- 令$\mathbb{E}[X_{ij}] = p_{ij}$。
- 考虑最小的同时包含S_i和S_j的递归过程（只有在这个过程中才有可能将S_i和S_j进行比较)。
- S_i和S_j被比较当且仅当分裂点是S_i或者S_j。
- 令P_i为S_i在排好序的数列中的位置，则这个递归过程中包含至少$|P_i - P_j| + 1$个数字。
- p_{ij} 最多是$2/(|P_i - P_j| + 1)$。

- 于是，

$$\begin{aligned}
\sum_{1\le i<j\le n} p_{ij} &\le \sum_{1\le i<j\le n} \frac{2}{|P_i - P_j| + 1} \\
&= \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{2}{j-i+1} \\
&= \sum_{i=1}^{n} \sum_{k=1}^{n-i+1} \frac{2}{k} \\
&\le 2\sum_{i=1}^{n} \sum_{k=1}^{n} \frac{1}{k} \\
&= 2nH_n \\
&= 2n\ln n + O(n) \\
&= O(n\log n)
\end{aligned}$$

这里，$H_n = 1 + 1/2 + 1/3 + \cdots + 1/n = \ln n + O(1)$。

注意：

- 这个结果对于任何一个输入序列成立，但对于随机的输入序列不一定成立(除非随机输入序列与分裂点的选择互相“独立”)。
- 这个结果仅仅保证算法在大部分情况下运行速度很快($O(n\log n)$)，但仍然有非常慢($\Omega(n^2)$) 的可能。

我们还可以证明对于任何一个输入序列，算法运行速度很快的概率“非常大”。

引理 4.1 *对于足够大的n(如$n \ge 20$)，*

$$\mathrm{Var}[C] \le n^2$$

由引理4.1和Chebyshev 不等式(引理2.8)我们可以得到，

$$\begin{aligned}
\mathbb{P}[C \ge 4n\ln n + O(n)] &= \mathbb{P}[C \ge 2\mathbb{E}[C]] \\
&\le \frac{\mathrm{Var}[C]}{\mathbb{E}[C]^2} \\
&\le \frac{n^2}{(2n\ln n + O(n))^2} \\
&= \Omega\Big(\frac{1}{\ln^2 n}\Big)
\end{aligned}$$

引理4.1的更精确形式为

$$\begin{aligned}
\mathrm{Var}[C] &= 7n^2 - 4(n+1)^2 H_n^{(2)} - 2(n+1)H_n + 13n \\
&= \Big(7 - \frac{2}{3}\pi\Big)n^2 - 2n\ln n + O(n)
\end{aligned}$$

该结论参见[1]和[2]。

在1989年，Hennequin [3]利用Chebyshev 不等式的四次矩形式，证明了

$$\mathbb{P}[C \geq 4n\ln n + O(n)] = O\Big(\frac{1}{\ln^4 n}\Big)$$

接着，McDiarmid和Hayward[4]在1992年证明了，

$$\mathbb{P}[C \geq 4n\ln n + O(n)] = n^{-2(\ln\ln n + O(\ln\ln\ln n))}$$

将这个界做出了数量级上的改进。

5 稳定婚姻问题(Stable Marriage Problem)

问题定义：有n位男生和n位女生，每位男生都有一个他对所有女生按喜欢程度排序的列表。对应的，每位女生也有这样一张关于所有男生的列表。比如这个问题的一个实例可能是，对于$n = 4$，有下述列表，

$$\begin{array}{llll} A: abcd & B: bacd & C: adcb & D: dcab \\ a: ABCD & b: DCBA & c: ABCD & d: CDAB \end{array}$$

这里大写字母表示女生，小写字母表示男生。考虑这样一个婚姻安排$M = \{A-a, B-b, C-c, D-d\}$。注意到相比$M$的安排，$C-d$更愿意结合在一起：$C$会抛弃$c$选择$d$，而$d$也会抛弃$D$选择$C$。于是如同$M$这样的婚姻安排是不稳定的，因为$C-d$会一起私奔。

问题 5.1 *给定所有男生和女生的排序列表，是否存在一个稳定的婚姻安排？*

求婚算法：每位单身的男生按照自己的喜爱程度从高到底依次向女生求婚。女生可以选择拒绝：因为她已经和更喜欢的人在一起了；或者是接受求婚：如果女生现在不是单身，则抛弃现任配偶。

算法的正确性：

引理 5.2 *求婚算法在结束时一定能够返回一个稳定的婚姻安排。*

证明 (引理5.2) 以下证明算法结束时，一定会返回一个所有女生和所有男生之间的完全匹配M。

- 女生一旦脱离单身状态，则再也不会单身。
- 算法结束时，不会有女生X仍然是单身：否则一定会有单身的男生y，而该男生y会向所有的女生求婚，包括X；由于X单身，她一定会接受求婚摆脱单身，从而矛盾。

下面证明该匹配M一定是稳定的。

- 对于每一位女生来说，与她配对的男生（在她自己的喜爱列表中）名次一定是越来越高的。
- 对于M来说，不存在任何一对$X - y \notin M$愿意私奔。这是因为：
  - 如果y已经向X求婚过，那么y在X喜爱列表中的名次一定低于X的现任配偶；
  - 如果y还没有向X求婚，那么y一定更喜欢他的现任配偶，而不是X。

■

容易看出，算法时间复杂度为$O(n^2)$，即整个算法中最坏会有$\Omega(n^2)$次求婚过程。

5.1 平均时间复杂度分析

假设每位女生事先确定好自己的列表，而每位男生的列表是随机确定的（即每位男生按照一个随机的顺序求婚）。那么求婚算法的期望求婚次数是多少？令T_P是为这个数目。

看来$\mathbb{E}[T_P]$不是很好分析。我们再定义一个“健忘”算法，每位男生每次从全体女生中随机选取一位女生求婚（即使有些人可能已经求婚过）。令T_A是健忘算法的求婚次数。

事实 5.3

$$\forall z \in \mathbb{R}, \mathbb{P}[T_A > z] \geq \mathbb{P}[T_P > z]$$

由此可知，T_A在任何情况下都是T_P的上界，任何我们得到的T_A的上界，也是T_P的上界。

当每一位女生都有配偶的时候，算法结束。因此在每一个单身男生求婚时，若他选择了一个单身的女生，该女生一定接受，而算法就向结束迈进了一步。换句话说，T_A与下述奖券收集者问题中的X分布是一样的。

奖券收集者问题：公司发行n种类型的奖券，购物者每购买该公司的一份产品，就会获得一张奖券，该奖券是n种奖卷中的随机的一种。假设购物者需要购买X份产品，才能集齐所有n种奖券。则

定理 5.4 *对于任何常数$c \geq 0$,*

$$\mathbb{P}[X > (c+1)n\ln n] \leq \frac{1}{n^c}$$

推论 5.5 *对于任何常数$c \geq 0$,*

$$\mathbb{P}[T_P > (c+1)n\ln n] \leq \mathbb{P}[T_A > (c+1)n\ln n] \leq \frac{1}{n^c}$$

例子 5.6 *对于$c = 3$,*

$$\mathbb{P}[T_P > 4n\ln n] \leq \frac{1}{n^3}$$

也就是说在随机输入的前提下，求婚算法求婚次数超过$4n\ln n$的概率不超过$1/n^3$。

5.2 奖券收集者问题

证明 (定理5.4) 令事件Z_i^r为在前r次获得的奖券中都没有第i种奖券。显然，

$$\mathbb{P}[Z_i^r] = \left(1 - \frac{1}{n}\right)^r \leq e^{-r/n}$$

于是对于$r = \beta n\ln n$，我们有$\mathbb{P}[Z_i^r] \leq e^{-(\beta n\ln n)/n} = n^{-\beta}$。即，

$$\mathbb{P}[X > \beta n\ln n] \leq \mathbb{P}[\cup_i Z_i^{\beta n\ln n}] \leq n \cdot n^{-\beta} = n^{1-\beta}$$

这里，我们使用了Union Bound (定理2.4)。

取$\beta = (c+1)$就可以得到定理5.4的结论。

■

其它分析方法： 令$C_i \in \{1, 2, \cdots, n\}$为购买第$i$份产品时获得的奖券种类。如果$C_j$和前$(j-1)$第$j$次购买是成功的。令$X_i$为第$i$次成功之后到第$(i+1)$次成功购买的次数。显然，

$$X = \sum_{i=0}^{n-1} X_i$$

在第i次成功之后，有$(n-i)$种奖券还没有获得，因此每买一份产品的成功概率是$(n-i)/n$，从而

$$\mathbb{E}[X_i] = \sum_{k=1}^{\infty} k \cdot \left(\frac{i}{n}\right)^{k-1} \left(\frac{n-i}{n}\right) = \frac{n}{n-i}$$
$$\mathbb{E}[X_i^2] = \sum_{k=1}^{\infty} k^2 \cdot \left(\frac{i}{n}\right)^{k-1} \left(\frac{n-i}{n}\right) = \frac{n(n+i)}{(n-i)^2}$$

根据期望线性性(引理2.5)，

$$\begin{aligned}
\mathbb{E}[X] &= \sum_{i=0}^{n-1} \mathbb{E}[X_i] \\
&= \sum_{i=0}^{n-1} \frac{n}{n-i} \\
&= nH_n \\
&\leq n \ln n + O(n)
\end{aligned}$$

这里，$H_n = 1 + 1/2 + 1/3 + \cdots + 1/n = \ln n + O(1)$。

另外，对于任何$i \neq j$，显然X_i与X_j相互独立：$\mathbb{E}[X_i X_j] = \mathbb{E}[X_i]\mathbb{E}[X_j]$。于是，

$$\begin{aligned}
\mathrm{Var}[X] &= \mathbb{E}[X^2] - \mathbb{E}[X]^2 \\
&= \mathbb{E}\Big[\Big(\sum_{i=0}^{n-1} X_i\Big)^2\Big] - \Big(\sum_{i=0}^{n-1} \mathbb{E}[X_i]\Big)^2 \\
&= \sum_{i=0}^{n-1} (\mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2) \\
&= \sum_{i=0}^{n-1} \Big[\frac{n(n+i)}{(n-i)^2} - \Big(\frac{n}{n-i}\Big)^2\Big] \\
&= \sum_{i=0}^{n-1} \frac{ni}{(n-i)^2} \\
&= n\sum_{i=0}^{n-1} \frac{n-(n-i)}{(n-i)^2} \\
&= n^2 \sum_{i=0}^{n-1} \frac{1}{(n-i)^2} - n\sum_{i=0}^{n-1} \frac{1}{n-i} \\
&= n^2 H_n^{(2)} - nH_n \\
&\leq \frac{\pi}{6} n^2
\end{aligned}$$

这里，$H_n^{(2)} = 1 + 1/4 + 1/9 + \cdots + 1/n^2 \leq \pi/6$。

使用Chebyshev 不等式(定理2.8)，有

$$\mathbb{P}[X > n\ln n + cn\ln n] \leq \frac{\frac{\pi}{6}n^2}{c^2(n\ln n)^2} = \frac{\pi}{6(c\ln n)^2}$$

当然这样比定理5.4的界要差不少。

6 将一个可三染色图分成两个不含三角形的子图

问题定义：若一个无向图可以三染色，则我们可以将其顶点集合V分成三个不相交的子集：$V = X \cup Y \cup Z$，而每一个子集在原图中都是一个独立集(即原图中没有一条边的两个顶点同时存在于一个子集中)。由于判定图是否可以三染色的问题是NP-Complete，因此没有有效算法可以将可三染色图分成三个不相交的独立集。

那么换一个简单一点的问题：是否可以将一个可三染色图的顶点集合分成两个不相交的子集：$V = A \cup B$，使得没有原图中的三角形的三个顶点同时存在于一个子集中。

算法：首先随机的将V分成两个子集A和B。对于一个不合法的三角形（三个顶点同时存在于A或B）中，随机选择其中一个顶点，放到对面的子集中。这样不断的循环下去，直到没有不合法的三角形。

时间复杂度分析：以下分析选自[5]：

- 原图可三染色：存在V的划分$V = X \cup Y \cup Z$，使得原图中的任何三角形，三个顶点分别落在X, Y和Z中。

- 令$c = |X| + |Y|$，则$c \le n$。
- 假设算法初始时随机划分为$V = A \cup B$，令$N(A,B) = |A \cap X| + |B \cap Y|$，显然$0 \le N(A,B) \le c$。
- 若$N(A,B) = 0$或$N(A,B) = c$，则(A,B)是一个合法的划分：没有原图中的三角形的三个顶点同时在A或B中。
- 在算法的循环中$0 < N(A,B) < c$。
- 对于任何一个不合法的三角形，算法在三个顶点v_1, v_2, v_3中选择一个随机的点v放到对面的子集中：从划分(A,B)到(A',B')，考虑$N(A',B') - N(A,B)$：

$$N(A',B') - N(A,B) = \begin{cases} +1 & \text{当} v \in A \cap Y \text{ 或} v \in B \cap X \\ -1 & \text{当} v \in A \cap X \text{ 或} v \in B \cap Y \\ 0 & \text{当} v \in Z \end{cases}$$

- 由于v_1, v_2, v_3分别落在X, Y, Z中，于是v以相同的概率（各1/3）落在X, Y, Z中，以上三项的概率都是1/3。
- 算法的执行相当于在$[0, c]$的一条线段上进行随机行走(random work)：从一个整数点出发，结束点在0和c，在行走中的每一步，各以1/3的概率向左走一步(-1)，向右走一步(+1)和原地不动(0)。

引理 6.1 *在上述随机行走中，期望行走的步数不超过$3c^2/8$。*

证明(引理6.1) 令随机变量$X_i (0 \le i \le c)$ 为从整数点i 出发行走的步数。显然

$$\mathbb{E}[X_0] = \mathbb{E}[X_c] = 0$$

对于$0 < i < c$，有

$$\begin{aligned} \mathbb{E}[X_i] &= 1 + \frac{1}{3}(\mathbb{E}[X_{i-1}] + \mathbb{E}[X_i] + \mathbb{E}[X_{i+1}]) \\ \mathbb{E}[X_{i+1}] - \mathbb{E}[X_i] &= \mathbb{E}[X_i] - \mathbb{E}[X_{i-1}] - 3 \end{aligned}$$

设$\Delta = \mathbb{E}[X_1] - \mathbb{E}[X_0]$，则对于$0 \le i \le c$,

$$\mathbb{E}[X_i] = -\frac{3}{2}i(i-1) + i\Delta$$

由$\mathbb{E}[X_c] = 0$知

$$\begin{aligned} -\frac{3}{2}c(c-1) + c\Delta &= 0 \\ \Delta &= \frac{3}{2}(c-1) \end{aligned}$$

于是

$$\begin{aligned}\mathbb{E}[X_i] &= \frac{3}{2}i(c-i) \\ &\leq \frac{3}{8}c^2\end{aligned}$$

■

定理 6.2 上述算法的期望迭代次数为$O(n^2)$。

7 最小割

定义 7.1 (割，$s-t$割和割的容量) *(参见定义3.1)*给定一个加(正)权无向图$G=(V,E,w)$，任何一个关于顶点集合V的划分$[A,B]$ *($V=A\cup B, B=V-A, A\neq\emptyset, B\neq\emptyset$)*称为该图的一个割。

对于图中两个不同的顶点$s,t\in V$，若$[A,B]$满足$s\in A, t\in B$，则称$[A,B]$是一个$s-t$割。

对于任何$[A,B]$，定义割的容量

$$C[A,B]=\sum_{(u,v)\in E, u\in A, v\in B} w(u,v)$$

定义 7.2 (最小割，最小$s-t$割) 定义G中的最小割为所有割$[A,B]$中容量最小者。给定不同的$s,t\in V$，定义G中的最小$s-t$割为所有$s-t$割中的容量最小者。即

$$\begin{aligned}\mathrm{MinCut}(G) &= \min_{[A,B]}\{C[A,B]\} \\ \mathrm{MinCut}_{s-t}(G) &= \min_{[A,B], s\in A, t\in B}\{C[A,B]\}\end{aligned}$$

引理 7.3

$$\mathrm{MinCut}(G)=\min_{s\neq t\in G}\{\mathrm{MinCut}_{s-t}(G)\}$$

注释 7.4 由最大流最小割定理，最小$s-t$割可以通过求在G中从s到t的最大流得到。而根据引理*7.3*，G最小割可以通过枚举s,t，进而计算最小$s-t$割得出。不过这样需要计算$\Theta(n^2)$次最大流。

在下本节中，我们将介绍一种简单的随机算法，时间复杂度仅为$O(n^2\log^3 n)$。

为了描述简单，我们仅仅讨论无权图（每条边的权值都是1：$w\equiv 1$）上的最小割。显然，任何无权图上的算法都可以很容易被应用到加权图上。

简单算法：

(1) 在G 中随机选择一条边(u,v)。

(2) 收缩(u,v)，保留重边，但删除自环。即如果图中存在两条边(u,y)和(v,y)，在收缩后这两条边都被保留。

(3) 如果图中还有多于2个顶点，返回(1)进行循环，否则输出两个点所代表的集合，作为最小割$[A, B]$。

等效算法：

(1) 将所有边随机排序。

(2) 按照这个顺序，使用Kruskal（克鲁斯卡尔）算法求出一棵生成树。

(3) 去掉最后加上的那条边，输出两个连通分量，作为最小割$[A, B]$。

引理 7.5 *简单算法有$\Omega(1/n^2)$的概率输出正确的最小割。*

证明 (引理7.5) 分析如下：

- 假设原图的最小割为$[A_0, B_0]$，那么在算法中，只要不选择跨越$[A_0, B_0]$的边进行收缩，$[A_0, B_0]$将一直是一个合法的割。
- 在算法的循环过程中，新图的任何一个割都一定是原图的割（把新图中的点展开即可）。因而在算法执行过程中，图中的最小割容量一定不会减少。
- 假设原图中最小割的容量是k，那么在算法执行过程中，新图的最小割容量至少也是k。于是新图中每一个点的度至少是k，假设这是第$i(1 \le i < n-1)$次循环，新图中有$(n-i+1)$个顶点，那么图中至少有$(n-i+1)k/2$条边。

于是，确定原图中的一个最小割$C[A_0, B_0] = k$，在算法循环的第i轮中，$[A_0, B_0]$“存活”下来的概率至少是

$$1 - \frac{k}{(n-i+1)k/2} = \frac{n-i-1}{n-i+1}$$

那么在算法结束时（第$(n-1)$轮后），$[A_0, B_0]$作为输出的概率至少是

$$\begin{aligned} & \prod_{i=1}^{n-2} \frac{n-i-1}{n-i+1} \\ = & \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \cdots \cdot \frac{2}{4} \cdot \frac{1}{3} \\ = & \frac{2}{n(n-1)} \end{aligned}$$

■

一个简单的推论：一个图中最多有$n(n-1)/2$个不同的最小割。

定理 7.6 *将简单算法重复运行$cn^2 \ln n$次，有至少$1 - \frac{1}{n^{2c}}$的概率我们可以得到最小割。*

证明 (定理7.6) 运行一次简单算法失败的概率最多是$1 - \frac{2}{n^2}$，那么运行$cn^2 \ln n$次该算法，失败的概率最多为

$$\begin{aligned} & \left(1 - \frac{2}{n^2}\right)^{cn^2 \ln n} \\ \le & e^{-2c \ln n} \\ = & \frac{1}{n^{2c}} \end{aligned}$$

于是有至少$1-\frac{1}{n^{2c}}$的概率我们可以得到最小割。这个算法的时间复杂度为$O(n^4 \log n)$。 ■

加速算法： 该算法由D. Karger和C. Stein在[6]中给出，时间复杂度为$O(n^2 \log^3 n)$，后来Karger由此改进在[7]中给出了一个$O(m \log^3 n)$的算法。

观察 7.7 *相比后期，在算法执行的前期，简单算法更不容易丢掉最小割。*

在执行了$(1-1/\sqrt{2})n$轮循环以后，算法没有丢掉最小割的概率至少是

$$\begin{aligned} & \prod_{i=1}^{(1-1/\sqrt{2})n} \frac{n-i-1}{n-i+1} \\ = & \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \cdots \cdot \frac{n/\sqrt{2}}{n/\sqrt{2}+2} \cdot \frac{n/\sqrt{2}-1}{n/\sqrt{2}+1} \\ = & \frac{n^2/2-n/\sqrt{2}}{n(n-1)} \\ \approx & \frac{1}{2} \end{aligned}$$

这个事实提示我们：我们应该着重重复执行算法的后期，而不是前期。设计算法如下：
对于一个规模为n的图G，

(1) 进行$(1-1/\sqrt{2})n$轮简单算法中的循环，得到一个规模为$n/\sqrt{2}$的图G'。

(2) 重复2次：使用本算法递归计算G'的最小割，取较优者输出。

算法执行完后，会构造出一棵深度为$2\log_2 n$的满二叉搜索树，有n^2个叶子结点。
算法时间复杂度分析：

$$\begin{aligned} T(n) &= n^2 + 2T(n/\sqrt{2}) \\ &= n^2 + n^2 + 4T(n/2) \\ &= n^2 + n^2 + n^2 + 8T(n/2\sqrt{(2)}) \\ &= kn^2 + 2^k T(n/\sqrt{2}^k) \\ &= O(n^2 \log n) \end{aligned}$$

算法成功概率分析： 算法成功的概率相当于在搜索树上，每条边都以1/2的概率存在时，从根结点到可以到达至少一个叶结点的概率。

引理 7.8 *令$P(d)$为在这样一个层数为d的树上从根结点可以到达至少一个叶结点的概率，则*

$$P(d) \geq \frac{1}{d}$$

证明 (引理7.8) 施归纳法于d。
归纳奠基： 当$d=1$是，树中只有一个结点：既是根结点也是叶结点，显然$P(d)=1 \geq 1/d$。

归纳过程： 假设对于$d = k \geq 1$时引理成立，那么

$$\begin{aligned} P(k+1) &= 1 - \left(1 - \frac{1}{2}P(k)\right)^2 \\ &= P(k) - \frac{1}{4}P(k)^2 \\ &\geq \frac{1}{k} - \frac{1}{4k^2} \\ &> \frac{1}{k} - \frac{1}{k(k+1)} \\ &= \frac{1}{k+1} \end{aligned}$$

■

于是算法成功的概率至少为$P(2\log_2 n) = \Omega(1/\log n)$。

定理 7.9 将上述算法运行$O(\log^2 n)$次，消耗时间$O(n^2 \log^3 n)$，以$1 - \frac{1}{n^2}$的概率得到最小割。

References

[1] Donald Knuth. *The Art of Computer Programming*. Addison-Wesley, first edition, 1973.

[2] Robert Sedgewick. *Quicksort*. Garland Publishing Inc., first edition, 1980.

[3] P. Hennequin. Combinatorial analysis of quicksort algorithm. *Theoretical Informatics and Applications*, 23(3):317–333, 1989.

[4] Colin McDiarmid, Ryan Hayward, Strong Concentration for Quicksort, *Proc. of the 3rd SODA*, 414-421, 1992.

[5] Colin McDiarmid, A random recolouring method for graphs and hypergraphs, *Combinatorics, Probability and Computing*, 2:363–365, 1993.

[6] D. Karger and C. Stein, An $\tilde{O}(n^2)$ algorithm for minimum cuts, *Proc. of the 25th STOC*, 757-765, 1993.

[7] D. Karger, Minimum cuts in near-linear time, *Proc. of the 28th STOC*, 56-63, 1996.