运筹学通论I

胡晓东

应用数学研究所

中国科学院数学与系统科学研究院

http://www.amt.ac.cn/member/huxiaodong/index.html

5. 组合优化 - 算法设计技巧

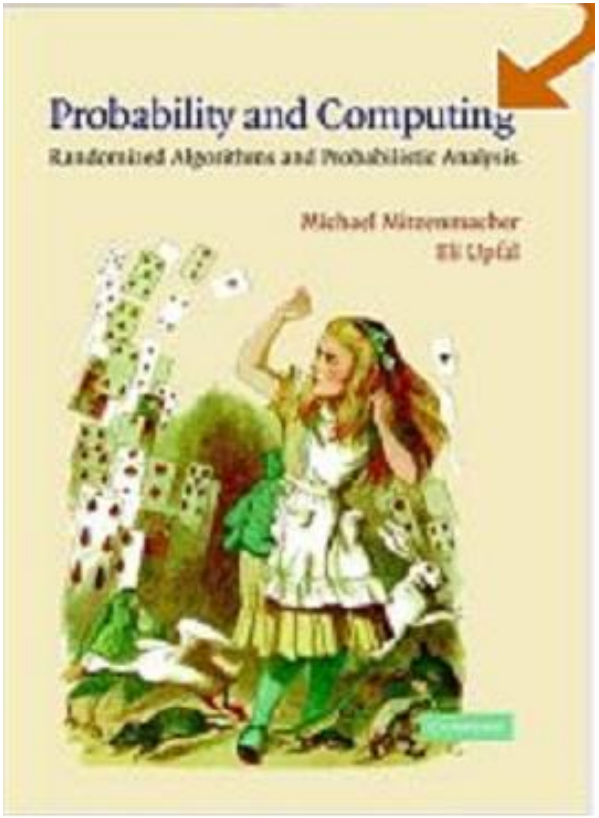
精确算法

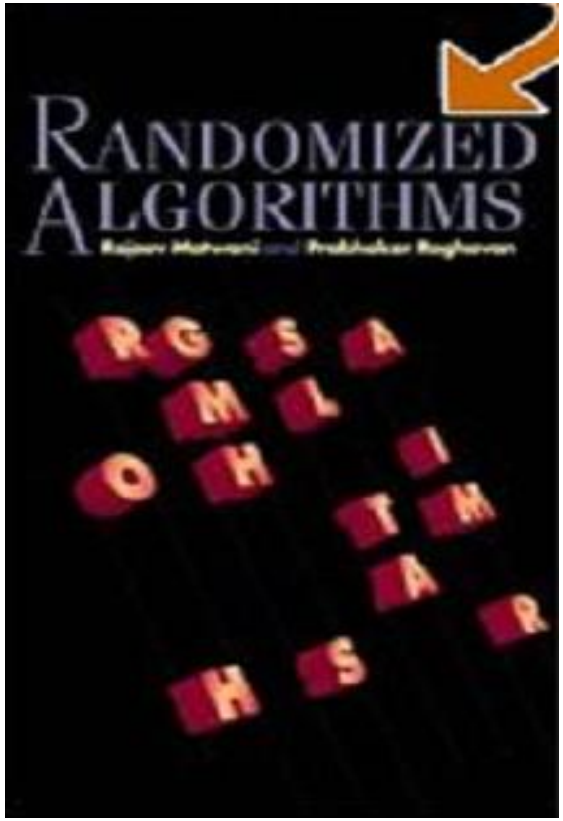
分而治之 (搜索、排大小序、旅行商)
动态规划 (最短路、三角剖分、背包)
分支定界 (整数线性规划、旅行商、工件排序)

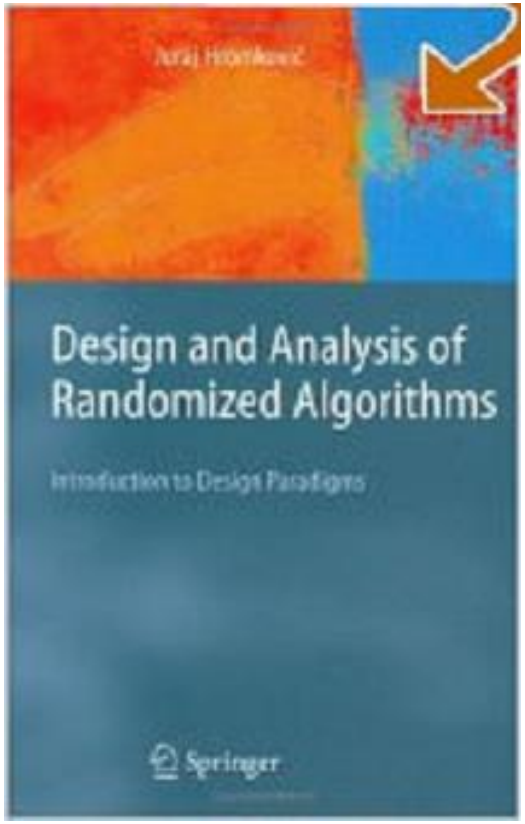
近似算法或启发式算法

贪婪策略 (最小生成树、最大可满足、背包、
顶点覆盖、独立集、旅行商)
局部搜索(最大匹配、旅行商、最大割)
序贯法 (工件排序、装箱、顶点着色)
整数规划法 (顶点覆盖、最大可满足、最大割)
随机方法 (最小割、最大可满足、顶点覆盖)
在线算法 (页面调度、k-服务器、工件排序、装箱)
不可近似 (最大团、背包、旅行商、装箱、连通控制集等)

5.8 随机算法


Probability and Computing
Randomized Algorithms and Probabilistic Analysis
Michael Mitzenmacher
Eli Upfal


RANDOMIZED
ALGORITHMS


Design and Analysis of
Randomized Algorithms
Springer

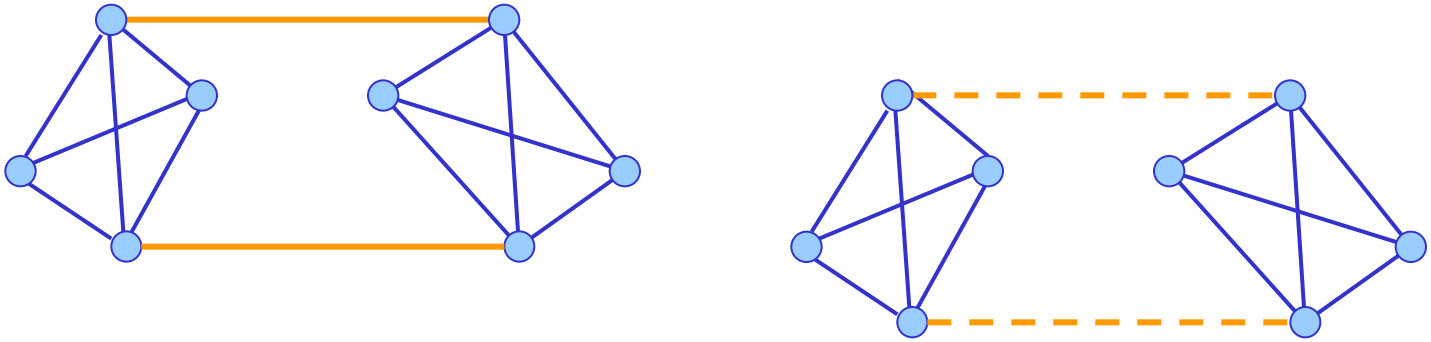
5.8 随机算法（续一）

近二十年里，在组合优化方向形成了一个非常热的研究课题：随机算法。一个主要的原因是，在处理许多组合优化问题时，与确定型算法相比，随机算法或者更加简单（容易设计），或者更加快速。

一个随机算法在执行一些指令是，需要做出随机性的选择。注意，因为算法进行了随机性的操作，所以即使对问题的同一个实例，运行随机算法若干次，有可能产生的解是不一样的。这样一个随机算法生成的解的目标函数值是一个随机变量，因而当我们估计该值的期望值时，我们不得不考虑随机算法所有可能的运行结果的平均性能。

5.8 最小割

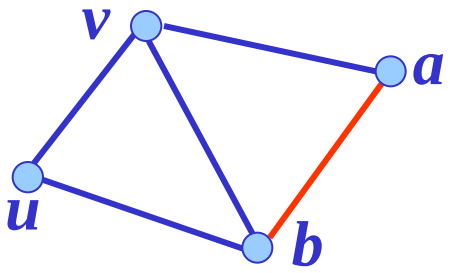
我们首先考虑**最小割问题**。给定一个连通图 $G=(V, E)$，其中 G 每一对顶点之间可能存在多重边。我们称一个子集 $C \setminus E$ 是一个**割**如果将集合 C 中的边从图 G 中去掉会造成剩下的图变成非连通。目标是如何找到一个含有最少边的割（若两个顶点之间若存在重边，则按照重数计算边数）。

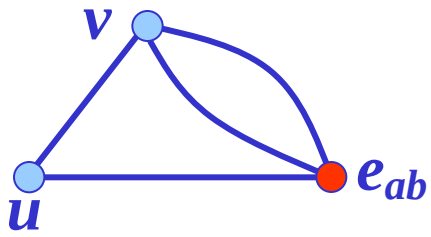


显然，图 G 最小割含有的边的条数不会超过图 G 中顶点的最小度数，通常称其为图 G 的**边连通度**，这是因为要使得图 G 变得非连通，最少要从其中去掉这么多条边。

5.8 最小割（续一）

求解最小割问题有一个非常简单的随机算法，它主要是利用对边进行的**收缩**运算。给定一个（可能含有重边的）图$G = (V, E)$，**收缩**其中的一条边 $e = (a, b)$ 是将该边的两个端点 a 和 b 用一个新的顶点 e_{ab} 代替，而对每一个顶点 $v \in V \setminus \{a, b\}$，将边 (a, v) 或者 (b, v) 用新的边 (e_{ab}, v) 代替，图中的其他边和顶点皆保持不变。图中所有的重边都需要保留。我们将对边进行收缩过的图记为$G/(a, b)$。

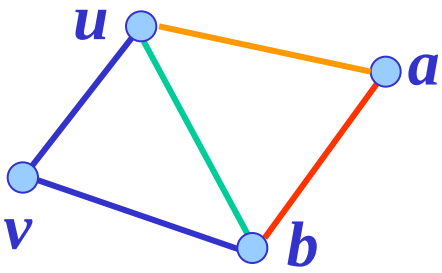


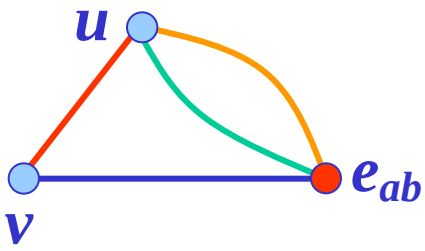


练习 对图 G 中任意一条边进行收缩不会减少其最小割含有的边的条数。

5.8 最小割（续二）

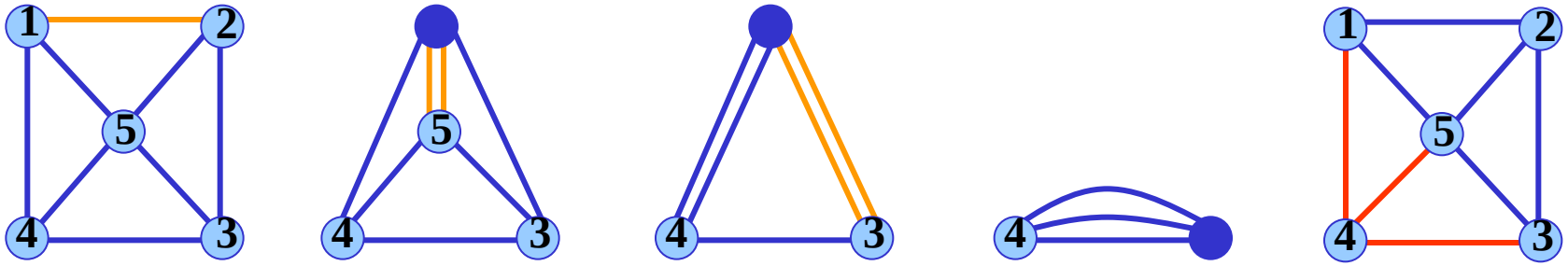
该算法的步骤如下：从边集 $E(G)$ 中以等概率任取一条边 (a, b)，将这条边收缩得到一个新图 $G/(a, b)$。如果一个（新产生的）顶点对之间有不只一条边，那么将它们全部都保留。去掉合并的顶点之间的边，这样就不会产生任何环（边的两个端点是同一个顶点）。 对图 $G/(a, b)$ 重复进行边收缩操作，直到图 $G/(a, b)$ 中仅存两个顶点；此时，这两个顶点之间的边就构成了图 G 的一个割（这里我们可能需要将这些边中的若干条边回溯的原图 G 中相应的边）。



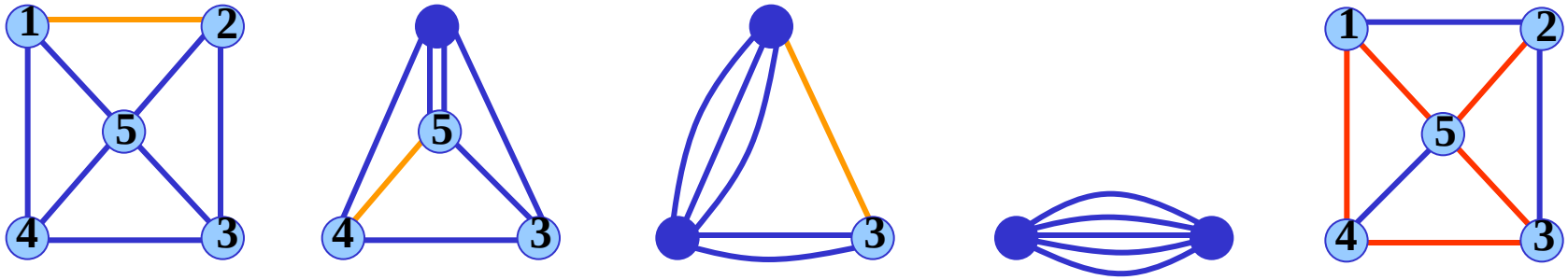


5.8 最小割（续三）

这个算法找到的割是否为最小割可能取决于对那些边进行收缩。如下图产生的割是最小的。



而如下图产生的割就不是最小的。然而，它是一个极小割（去掉任意一条边，所剩下的边就构不成一个割）。



5.8 最小割（续四）

定理 1 任给最小割问题的一个实例，应用随机算法可找到最小割的概率至少是 $2n^{-2}$。

现在假定，我们随机独立地运行 $n^2/2$ 次这个随机算法，那么在这 $n^2/2$ 次运行中没有找到最小割的概率最多是

$$\left(1-\frac{2}{n^2}\right)^{n^2/2}<\frac{1}{e}.$$

事实上，如果我们再运行更多次这个随机算法，那么找不到最小割的概率可以任意地小，代价是增加了运算时间。也就是说，我们在多次运行随机算法以后，产生的所有割中找出最小的一个割，它很有可能就是原图 G 的一个最小割。

5.8 最大权可满足

我们下面再回过头来讨论一下最大权可满足问题：给定一组子句，每一个子句赋有一个正权值，目标是找到布尔变量的一个赋值使得满足的子句的权值之和最大。

问　题: **最大权可满足**

实　例：n 个布尔变量 $X=\{x_1, \ldots, x_n\}$ 的 m 个合取子句 $C=\{c_1, \ldots, c_m\}$，每一个子句 c_i 有一个权值 w_i

可行解：布尔变量赋值 $f: X \rightarrow \{真, 伪\}$。

目　标：C 中被满足的子句的权值之和最大

在前面的讨论中我们给出了求解该问题的一个基于整数规划的算法。下面我们首先介绍一个求解该问题的非常简单的随机算法，然而证明将这两个算法相结合可以设计一个新的算法，它可以产生一个更好的赋值。

5.8 最大权可满足（续一）

求解最大权可满足问题的一个极其简单的算法是：将 X 中的每一个布尔变量随机地赋值“真”或者“伪”。很显然，在最坏情形下，该算法给出的赋值可能是一个非常坏的赋值。然而，下述定理说明，在一般情况（即平均意义）下，该算法给出的赋值并不是一个非常坏的赋值。

定理 2 任给最大权可满足问题的一个实例，若其中每一个子句最少含有 k 个字（布尔变量或布尔变量的非），算法所产生的赋值所满足的子句的权值之和期望值至少是最优赋值所能满足的子句的权值的 $(1-2^{-k})$ 倍。

证明. (练习) 注意事实，含有 k 个字的子句未被算法的赋值所满足的概率是 2^{-k}， 而含有至少 k 个字的子句被算法的赋值所满足的概率至少是 $1-2^{-k}$。

5.8 最大权可满足（续二）

设 $C_{SR}(I)$ 为算法所给出的布尔变量赋值所能满足的子句的权值之和期望值。则对所有子句求和即得

$$EXP[c_{SR}(I)] \geq \sum_{i=1}^{m} (1 - 2^{-k})\, w(c_i) = (1 - 2^{-k}) \sum_{i=1}^{m} w(c_i)$$

因为该问题的最优值最多为 $w(c_1) + w(c_2) + \ldots + w(c_m)$，所以定理成立。

注意，当每一个子句至少含有 2 个字时，这个简单的随机算法可以给出一个不错的解，这是因为 $k \geq 2$ 当时，我们有 $1 - 2^{-k} \geq 3/4$；另一方面，回忆一下，当每一个子句至多含有 2 个字时，随机取整算法的性能比是 3/4。因而，我们可以设计一个简单的算法：分别运行这两个算法，从得到的两个赋值中选取一个目标函数值更优的赋值方法。

5.8 最大权可满足（续三）

定理 3 任给最大权可满足问题的一个实例 **I**，将简单随机算法和随机取整算法输出的目标函数值分别设为 $c_{SR}(I)$ 和 $c_{RR}(I)$。则 $\max\{c_{SR}(I), c_{RR}(I)\} \geq 3c_{opt}(I)/4$。

证明. 要证明定理的结论只需证明

$$\frac{c_{SR}(I) + c_{RR}(I)}{2} \geq \frac{3}{4} c_{opt}^{LP}(I) \quad \text{因为 } c_{opt}^{LP}(I) \geq c_{opt}(I).$$

现用 C_k 表示恰好有 k 个字的子句的集合。注意，每一个子句 $c_j \in C_k$ 被简单随机算法给出的赋值所满足的概率为 $1 - 2^{-k}$。令 $\beta_k = 1 - 2^{-k}$，注意对每一个指标 j 有 $0 \leq z^* \leq 1$。则有

$$c_{SR}(I) \geq \sum_{k \geq 1} \sum_{c_j \in C_k} \beta_k w_j \geq \sum_{k \geq 1} \sum_{c_j \in C_k} \beta_k w_j z_j^*$$

5.8 最大权可满足（续四）

此外，再由前面定理，可得

$$c_{\mathrm{RR}}(\mathrm{I}) \geq \sum_{k\geq 1} \sum_{c_j \in C_k} \alpha_k w_j z_j^*$$

现将上述两个不等式相加，即得

$$\frac{c_{\mathrm{SR}}(\mathrm{I})+c_{\mathrm{RR}}(\mathrm{I})}{2} \geq \sum_{k\geq 1} \sum_{c_j \in C_k} \left(\frac{\beta_k + \alpha_k}{2}\right) w_j z_j^*$$

另外，再注意到 $\beta_1 + \alpha_1 = \beta_2 + \alpha_2 = 3/2$，对每一个 $k \geq 3$ 都有 $\beta_k + \alpha_k \geq 7/8 + 1 - 1/e \geq 3/2$。因而由上式可得

$$\frac{c_{\mathrm{SR}}(\mathrm{I})+c_{\mathrm{RR}}(\mathrm{I})}{2} \geq \sum_{k\geq 1} \sum_{c_j \in C_k} \frac{3}{4} w_j z_j^* = \frac{3}{4} c_{\mathrm{opt}}^{\mathrm{LP}}(\mathrm{I}).$$

事实上，并不需要分别独立地运行这两个算法，然后找出其中的一个更好的赋值；而只要以概率 1/2 运行两个算法中的一个，得到的赋值的目标函数的期望值是一样的。

5.8 最小顶点覆盖

在本节中我们再讨论**最小权顶点覆盖问题**。回忆一下，在前面第4节，我们给出了求解**最小顶点覆盖问题**（不带权）的两个简单的算法，一个是应用贪婪策略，而另一个是基于极大匹配；此外，在第7节我们还给出了求解最小权顶点覆盖问题的两个随机算法。下面我们给出求解该问题的另外一个随机方法。

问　题: **最小权顶点覆盖**

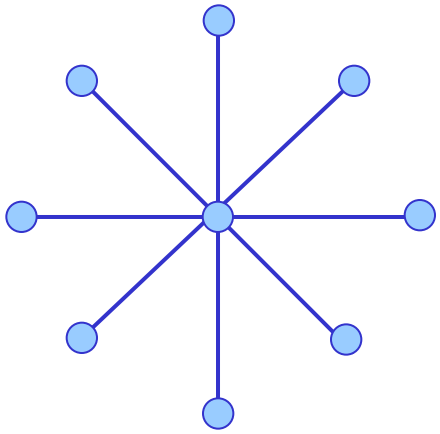
实　例: 图 $G=(V, E)$ 及顶点 $v_i \in V$ 的权值 w_i 。

可行解: 顶点覆盖 $C \subseteq V$。

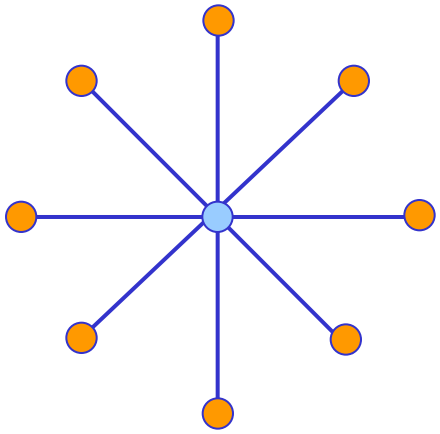
目　标: 最小化覆盖 C 中的顶点的权值之和 $||C||=\sum_{v_i \in C} w_i$

5.8 最小顶点覆盖（续一）

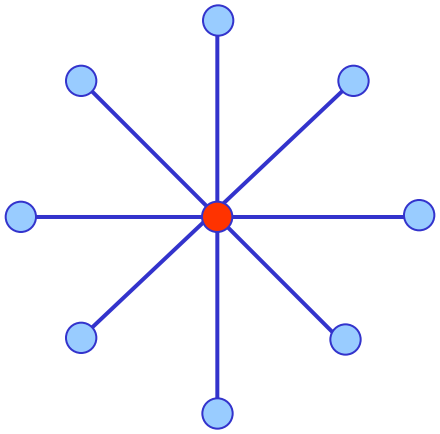
如同我们在设计求解**最大权可满足问题**的随机算法时那样，求解**最小（权）顶点覆盖问题**的一个简单的随机算法即是，将每条边的两个端点中的一个以（相同的）概率 **1/2** 放入顶点覆盖中。



问题的一个实例



随机算法产生的顶点覆盖



最优顶点覆盖

5.8 最小顶点覆盖（续二）

定理 4 任给一个图 $G=(V, E)$，简单随机算法输出的图 G 的顶点覆盖所含顶点个数的期望值不超过图 G 的最优顶点覆盖所含顶点个数的 2倍。

证明. 首先，让我们固定图 $G=(V, E)$，将其最优顶点覆盖记为 $C_{opt} \subset V$。然后，我们称一次抛硬币的结果是“好的”如果它使得顶点 $v \in C_{opt}$ 被放入了集合 C 中。因为每一条边至少有一个端点属于集合 C_{opt} 中，一次抛硬币的结果是“好的”概率至少是 1/2。

然而，抛硬币是“好的”结果的次数不会超过 $|C_{opt}|$，否则 集合 C_{opt} 中的所有顶点也都属于集合 C，因而图G 中的每一条边都被 C 中的顶点覆盖了。由此可知， 需要抛硬币的次数的期望值不会超过 $2|C_{opt}|$。

5.8 最小顶点覆盖（续三）

下面我们讨论一下如何将上述这个非常简单的求**最小顶点覆盖问题**的随机算法进行推广，使得它可以处理**最小权顶点覆盖问题**。基本思想还是一样的，惟一的区别在于，我们每一次抛的都是有偏差硬币。我们选取每一条边的两个端点中的一个的概率是与其权值成反比的。

ALGORITHM 13.5 *Randomized Algorithm*

$C := \emptyset$.
while $E \neq \emptyset$ **do begin**
    pick $e = (u, v) \in V$;
    randomly choose x from $\{u, v\}$ with $Prob[x = u] = \frac{w(v)}{w(u)+w(v)}$;
    $C := C \cup \{x\}$;
    $E := E \setminus \{e \,|\, e = (x, y)\}$.
end-while
return C.

5.8 最小顶点覆盖（续四）

注意，上述算法在利用了贪婪算法的优点的同时，也避免了在每一次都持续地选错端点。换言之，度数高的顶点被选入顶点覆盖的集合更大，在每一次抛硬币时，结果会偏向选取顶点权值小的端点。因而，一个顶点的权值与它当前的度数之比决定了该顶点被选入顶点覆盖的机会大小。

定理 5 任给一个顶点赋权图 ***G***，推广的随机算法可输出图 ***G*** 的一个顶点覆盖， 其权值的期望值最多不超过图 ***G*** 的最优顶点覆盖的权值的 **2** 倍。

- 相比确定性算法，随机算法的最显著特征就是，它们的结构简单，而且耗时少。
- 随机算法的主要缺点就是，它们所输出解的好坏具有随机性。不过，可以通过去随机化克服这个缺点。

5.8 总结

随机算法通常可以分为如下两种类型：

- **Las Vegas 算法**: 算法每次总可以输出所求解问题的一个可行解，只是每次输出解所需要的时间有可能不同。

- **Monte Carlo 算法**: 算法有时可能输出的不是所求解问题的一个可行解。然而，我们可以估计出发生这种情况的概率的上界。如果我们能以独立地随机方式，重复运行该类算法，输出的不是所求解问题的可行解的概率可以任意的小，当然重复运行算法需要花费更多的时间。

最后注意， 一方面，一个**Monte Carlo** 算法可以由一个 **Las Vegas** 算法生成（出错概率为 **0**）；而另一方面，**Las Vegas** 算法也可由一个 **Monte Carlo** 算法生成。