18.433 组合最优化

最小割

October 2 *授课教师: Santosh Vempala*

寻找一个图的最小割是一个很有意思的问题，而且在某些领域中有很多应用，如网络设计。这个问题有两种不同形式：(1) 找边的最小集合，若其被删去则特定的顶点s和t变为不连通（我们把这种形式称为*最小 s－t 割问题*），(2) 找边的最小集合，若其被删去则图变得不连通（我们把这种形式称为*最小割问题*）。这两个问题均适用于无向和有向图。

有向图中的最小 s－t 割问题可以利用最大流－最小割定理解决，即图中的最大流等于最小割。实际上，我们可以通过选择在剩余图 Res(G)中由s经有向路可达的那些顶点作为S，而G中其它的顶点作为$\overline{S}$来得到一个最小割$(S,\overline{S})$。上述的算法也可以解决无向图中的最小 s－t 割问题（具体解答留做练习）。

解决最小割问题的一个很自然的方法是选择任意一对顶点作为s和t，然后利用求最小 s－t 割的算法。这个算法的执行时间是最大流算法的$\binom{n}{2}$倍。考虑到每个最小割都把一个固定的顶点s和至少一个其它的顶点t分离开（有$n-1$种方法来选择t），我们可以把算法化简到最大流算法的$n-1$倍。

下面，我们给出求无向图中最小割的一个随机方法，并仔细地来分析它的算法执行时间。

随机最小割算法：

1、 当图中存在两个以上的顶点时
   (a) 随便选择一条边e。
   (b) 把边e收缩为一个顶点，得到一个重图（保留重边）。

2、 在两个剩余的超顶点之间的边记录为一个最小割。

注意到收缩一条边的执行时间不超过$O(n)$，主循环迭代的次数为$n-2$次。因此全部的执行时间不超过$O(n^2)$。在这一讲的后面，我们将利用某些技巧把算法执行时间从$O(n^2)$降低到$O(m)$（m是边的条数）。

我们现在来计算数据传送一个最小割的概率。我们知道一个图的割集的数目是指数型的(2^n)，因此从一个随机进程中得到一个最小割的概率可能非常低，即$\frac{1}{2^n}$。但是，引理 1 将说明在我们的算法中概率并不是那么小。

引理 1. 算法中得到某个特定最小割的概率不低于 $\frac{1}{\binom{n}{2}} \approx \frac{2}{n^2}$。

证明. 设最小割 $C = (S, \overline{S})$ 有 c 条边。那么图中每一个顶点的度数至少为 c 且至少有 $\frac{nc}{2}$ 条边。如果上述算法没有选择 C 中的任一边，那么最后的割集将为 C 。

$$Prob(picking\ an\ edge\ from\ C) \le \frac{c}{\frac{nc}{2}} = \frac{2}{n}$$

因此

$$Prob(\text{不选 } C \text{ 中的任一边}) \quad \ge 1 - \frac{2}{n}$$

同理可知，收缩完第一条边以后不选 C 中任一边的概率为 $1 - \frac{2}{n-1}$ ，等等。所以，

$$Prob(\text{找到割 } C\ ) \ge (1 - \frac{2}{n}) \cdot (1 - \frac{2}{n-1}) \cdots (1 - \frac{2}{3}) = \frac{1}{\binom{n}{2}}. \qquad \square$$

所以，随机算法成功的概率不低于 $\frac{1}{\binom{n}{2}}$。增加迭代次数和选择最好的割作为最终结果可以增加成功的概率。我们有：

$$\begin{aligned} Prob(succeed\ in\ k\ attempts) &= 1 - Prob(fail\ in\ all\ attempts) \\ &= 1 - Prob(F_1)Prob(F_2)\cdots Prob(F_k) \\ &= 1 - Prob(F_1)^k \\ &= 1 - (1 - \frac{1}{\binom{n}{2}})^k \end{aligned}$$

例如，算法在经过 $k = \binom{n}{2}$ 次迭代后，成功的概率不低于 $1 - \frac{1}{e}$ ；经过 $k = 2\binom{n}{2}\ln n$ 次迭代后，成功的概率不低于 $1 - \frac{1}{n^2}$ 。

值得一提的是利用引理 1，我们可以得出一个图最多有 $\binom{n}{2}$ 个最小割。实际上，每个最小割都可以以不低于 $\frac{1}{\binom{n}{2}}$ 的概率通过上面的算法唯一的获得，所以最小割的数目低于上述的上界。

如前所述，这个算法的执行时间为 $O(n^2)$，接下来我们将其改进为 $O(m)$。设 $e_1, e_2, \cdots, e_m$ 是所有边的一个随机排列，按这个顺序收缩每条边，直到图中只剩下两个顶点。这个算法与第一个算法的输出相同，但是，考虑这种二进制搜索方法：在 $O(m)$内

首先收缩边 $e_1, e_2, \cdots, e_{\lceil \frac{m}{2} \rceil}$，如果收缩后剩下两个顶点，那么这两个顶点间的边记为最小割；如果只剩下一个顶点，那么我们在这一半边上进行递归；否则我们在剩下的一半边上继续运行该算法。这样算法的执行时间不超过 $O(m \log m)$。但是，算法仍然有可以改进的地方。如果剩余顶点的个数多于两个，那么得到的新图 G' 最多有 $\lfloor \frac{m}{2} \rfloor$ 条边（因为我们已经收缩了前一半边）。如果剩余顶点的个数为 1，那么我们可以放弃后一半边。在这两种情况下，都最多有 $\frac{m}{2}$ 条边。因此，第二次迭代所需的时间正比于 $m/2$。类似地，第三次迭代所需时间正比于 $m/4$，等等。所以，总时间为 $O(m)$。