CS 598CSC：近似算法	讲座日期：2011年1月19日
讲师：Chandra Chekuri	记录员：Nitish Korula（2009年）

讲师于2011年编辑的笔记。

1 引言/行政事务

- 课程网站：http://www.cs.uiuc.edu/class/sp11/cs598csc/。
- 推荐书籍：
  - 《近似算法设计》*David Shmoys*和*David Williamson*著，剑桥大学出版社，2011年出版。 可在 http://www.designofapproxalgs.com/ 上免费获取。
  - 《近似算法》*Vijay Vazirani*著，*Springer-Verlag*出版，2004年。
- 6个作业集，前3个必做。 后3个作业集可以用项目替代。

课程目标

1. 要认识到不可解决问题并非都相同。 **NP**优化问题，在精确可解性方面相同，但在近似解的角度上可能非常不同。. 这解释了为什么在实践中，某些优化问题（如背包问题）很容易，而其他问题（如团问题）非常困难。

2. 通过一些基本问题，学习设计和分析近似算法的技术。

3. 构建一个可以用来解决各种问题的广泛适用的算法/启发式工具包。

4. 理解优化问题之间的约简，并培养将新问题与已知问题相关联的能力。

复杂度类 **P**包含可以在多项式时间内解决的问题集合。
从理论角度来看，这描述了可处理问题的类别，即可以高效解决的问题。 类 **NP**是可以在非确定性多项式时间内解决的问题集合，或者等价地说，可以在多项式时间内验证解的问题。

NP包含许多在实践中经常出现的有趣问题，但有充分的理由相信 $\mathbf{P} = \mathbf{NP}$。 也就是说，解决 **NP**优化问题的算法很可能不存在，因此我们经常采用启发式方法来解决这些问题。

启发式方法包括回溯搜索及其变体、数学规划方法、局部搜索、遗传算法、禁忌搜索、模拟退火等。有些方法保证找到最优解，尽管可能需要指数时间；而其他方法保证在多项式时间内运行，尽管可能无法返回最优解。 近似算法属于后一类；然而，尽管它们不能找到最优解，我们可以对找到的解的质量给出保证。

近似比

要对解的质量给出保证，首先必须定义我们所说的解的质量。 我们将在下一讲中更仔细地讨论这个问题；现在，请注意每个优化问题的每个实例都有一组可行解。 我们考虑的优化问题具有一个目标函数，它为每个实例的每个可行解分配一个（实数/有理数）数值。 目标是找到具有最小目标函数值或最大目标函数值的可行解。 前者是最小化问题，后者是最大化问题。

对于每个问题实例 I，让 $\mathrm{OPT}(I)$表示最优解的值。 我们说算法 $\mathcal{A}$是问题的 α-近似算法，如果对于每个实例 I， $\mathcal{A}$返回的可行解的值在 $\mathrm{OPT}(I)$的乘法因子 α内。 等价地，我们说 $\mathcal{A}$是一个近似算法，其近似比率为 α。 对于最小化问题，我们有 $\alpha \geq 1$，对于最大化问题，我们有 $\alpha \leq 1$。 然而，在文献中，我们经常会发现对于最大化问题有一个不同的约定，即如果 $\mathcal{A}$返回的可行解的值至少为 $\alpha \cdot \mathrm{OPT}(I)$，则称 $\mathcal{A}$是一个 α-近似算法；使用这个约定的原因是为了使最小化和最大化问题的近似比率都大于等于1。 在这门课程中，对于最小化问题，我们将大部分时间使用 $\alpha \geq 1$的约定，对于最大化问题，我们将使用 $\alpha \leq 1$的约定。

备注：

1. 对于最小化问题的算法，近似比是算法返回的解的值与最优解之间的比值的最大值（或上确界），在问题的所有实例上。 因此，它是算法的最坏情况性能的一个界限。

2. 近似比 α可以依赖于实例 I的大小，因此严格来说，应该写作 $\alpha(|I|)$。

3. 一个自然的问题是近似比是否应该以加法的方式定义。
   例如，对于最小化问题，如果一个算法输出的可行解的值对于所有的I都不超过 $\mathrm{OPT}(I) + \alpha$，则该算法具有 α-近似。 这是一个有效的定义，在某些情况下更相关。 然而，对于许多 **NP** 问题来说，很容易证明由于缩放问题，无法获得任何有趣的加法近似（除非当然 $P = NP$）。 我们将在后面的例子中说明这一点。

近似方法的优点和缺点：

近似方法的一些优点包括：

1. 它解释了为什么问题的难度会有很大的变化。
2. 对问题和问题实例的分析可以区分简单情况和困难情况。
3. 在许多方面，最坏情况比率是稳健的。 它允许在问题之间进行简化。
4. 算法思想/工具在开发启发式算法方面非常有价值，包括许多实用且有效的算法。

作为额外的奖励，许多思想都很美丽和复杂，并且与数学和计算机科学的其他领域有关。

缺点包括：

1. 过于关注最坏情况的度量可能忽视了实际可行或平均表现良好的算法或启发式算法。

2. 与整数规划等问题不同，运行时间和解的质量通常没有增量/连续的权衡。

3. 近似算法通常只适用于清晰陈述的问题。

4. 该框架不适用于决策问题或无法近似的问题。

近似算法作为一种广泛的视角

近似算法的使用不仅限于 **NP**-难优化问题。
通常情况下，近似算法的思想可以用来解决许多问题，如果要找到精确解，将需要太多的资源。

我们经常关注的一种资源是时间。 精确解决 **NP**-难问题（据我们所知）将需要指数时间，因此我们可能希望使用近似算法。 然而，对于大数据集，即使是多项式运行时间有时也是不可接受的。
例如，在一般图中，已知的最佳精确算法对于匹配问题需要 $O(n^3)$ 时间；对于大图来说，这可能是不切实际的。相比之下，一个简单的贪婪算法只需要接近线性时间，并输出至少是最大匹配一半的基数的匹配；此外，还有随机的次线性时间算法。

另一个经常受限的资源是空间。 在数据流/流算法领域，我们通常只允许一次性读取输入，并给出少量的额外存储空间。 考虑一个希望计算通过它的数据包统计信息的网络交换机。精确计算平均数据包长度很容易，但无法精确计算中位数长度。 令人惊讶的是，许多统计量可以近似计算。

其他资源包括程序员的时间（对于匹配问题，精确算法可能比返回近似解的算法复杂得多），或通信需求（例如，如果计算发生在多个位置之间）。

2 Steiner树问题

在$S\text{TEINER}$树问题中，输入是一个图G（V，E），以及一组终端$S \subseteq V$，以及每条边$e \in E$的成本$c(e)$。 目标是找到连接所有终端的最小成本树，其中子图的成本是其边的成本之和。

STEINER树问题是**NP**-难的，也是**APX**-难的[2]。 后者意味着存在一个常数$\delta > 1$，使得在比率小于δ的情况下近似解决方案是**NP**-难的；目前已知STEINER树问题在$95/94$的比率内是难以近似的[5]。

以Jakob Steiner命名的Steiner树问题的变体已经被Fermat、Weber和其他人研究了几个世纪。 课程教材的封面上有一封高斯写给Schumacher关于Steiner树问题的信的复制品。

注意：如果$|S| = 2$（即只有2个终端），最优Steiner树就是这两个终端之间的最短路径。 如果$S = V$（即所有顶点都是终端），最优解就是输入图的最小生成树。 在这两种情况下，问题可以在多项式时间内精确解决。

问题：你能找到一个有效的算法来准确解决带有3个终端的Steiner树问题吗？ 如果 $|S|$是一个固定常数，情况如何？

观察到，在图 G上解决Steiner树问题足够了，它在下面的度量完成图 G上解决它。（为什么是这样？）

定义：给定一个带有边成本的连通图 $G(V, E)$，度量完成图 $H(V, E')$是一个完全图，对于每个$u, v \in V$， H中的边 uv的成本是 G中从 u到 v的最短路径的成本。

具有边成本的图 H是度量的，因为边成本满足三角不等式：
$\forall u, v, w, \ \ cost(uv) \leq cost(uw) + cost(wv)$。

图1：左边是一个图。 右边是它的度量完成图，其中新的边和修改后的边缘成本用红色表示。

现在我们来看看两种近似算法来解决Steiner Tree问题。

2.1 最小生成树算法

以下算法的近似比率为$(2 - 2/|S|)$，来源于[12]：

SteinerMST$(G(V, E), S \subseteq V)$:
令 $H(V, E')$ ←图 G的度量完成图。
令 $T \leftarrow H[S]$的最小生成树。
输出 T。

（这里，我们使用 $H[S]$ 表示由终端集合 S引起的子图 H。）

以下引理对于算法 SteinerMST的分析至关重要。

引理1 对于Steiner Tree的任何实例 I，令 H表示图的度量完成图， S表示终端集合。 在 $H[S]$（由终端引起的图）中存在一棵生成树，其成本最多为 $2(1 - \frac{1}{|S|})$OPT，其中 OPT是实例 I的最优解的成本。

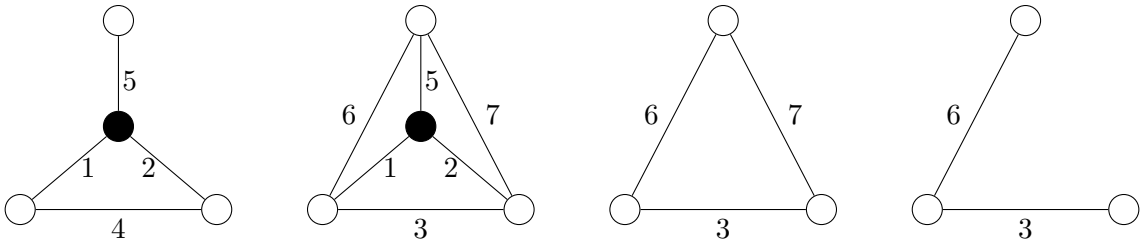

图2：说明STEINER树的最小生成树启发式算法

在我们证明引理之前，我们注意到如果存在成本不超过$2(1-\frac{1}{|S|})$OPT的H[S]中的某个生成树，则最小生成树的成本不超过此成本。因此，引理1意味着算法 STEINERMST是Steiner树问题的$2(1-\frac{1}{|S|})$-近似算法。

引理1的证明。 让 T^*表示给定实例 H中的最优解，其成本为$c(T^*)$。. 将 T^*的所有边加倍，得到一个欧拉图，并固定一个欧拉回路 W 来表示这个图。（见上图3。）现在，通过缩短 W 的边来得到一个遍历 T^*中顶点的回路 W'，其中每个顶点恰好被访问一次。 再次缩短 W'的边以消除所有非终端节点；这将得到一个遍历每个终端节点恰好一次的路径 W''。

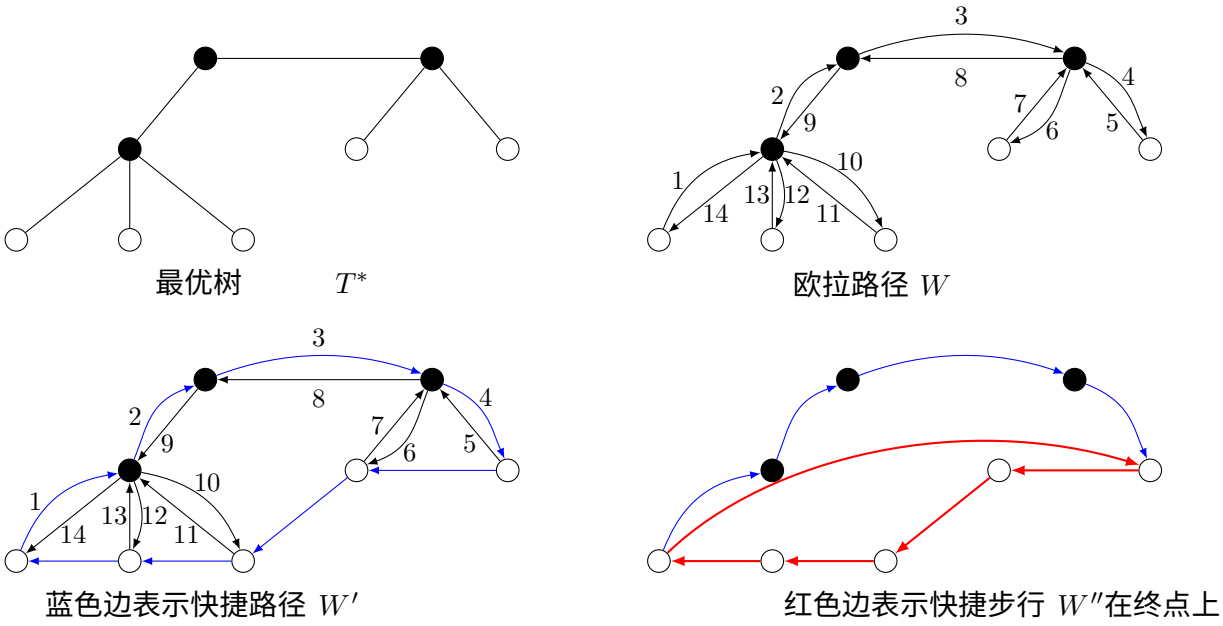

图3：加倍边 T^*并进行快捷操作可得到终点上的低成本生成树。

很容易看出 $c(W'') \leq c(W') \leq c(W) = 2c(T^*)$，其中不等式来自于快捷操作只能减少路径长度的事实。 （回想一下，我们在度量完备空间 H中工作。）现在，删除 W''中最重的边，以获得一个经过所有终点的路径在 S中，成本最多为（1-|S|）c(W'')。这条路径是终点的生成树，只包含终点；因此，在H[S]中存在一棵生成树，其成本至多为$(1-\frac{1}{|S|})c(T^*)$.

□

一个紧密的例子：下面的例子（图4）表明这个分析是紧密的；存在着Steiner Tree的实例，其中 SteinerMST算法找到了一棵成本为$2(1-\frac{1}{S})$OPT的树。 –
在这里，每对终端之间由成本为2的边连接，每个终端与中心非终端之间由成本为1的边连接。 最优树是一个包含中心非终端的星形树，与所有终端之间有边；它的成本为 $|S|$。 然而，在 $H[S]$中，唯一的树是由成本为2的 $|S|-1$条边构成的；它们的成本为$2(|S|-1)$。

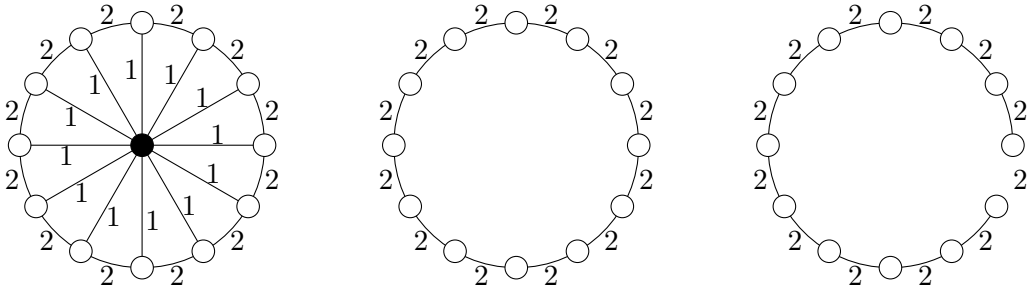

图 G；未显示所有边 $H[S]$；未显示所有边。 H的最小生成树[S]。图4：SteinerMST算法的一个紧密示例

2.2 贪心/在线算法

我们现在描述另一个简单的Steiner树问题算法，由[9]提出：

```
GreedySteiner(G(V, E), S):
让 {s_1, s_2, ..., s_|S|}是终端的任意排序。
令T ← {s_1}对
于(i从2到|S|):
    令P_i是G中从s_i到T的最短路径。将P_i添加到T
    中。
```

Gree⌈ySteiner是一个⌈ $\log_2|S|$E-近似算法；在这里，我们证明了一个稍微弱一些的结果。

定理2 算法G**ree**dy$_S$t**e**$_i$N**er**的近似比为$2H_{|S|}\approx 2\ l\mathrm{N}_{|S|}$，其中$H_i=\sum_{j=1}^{i} 1/j$表示第 i个调和数。

请注意，这是一种在线算法；终端按任意顺序考虑，并且当考虑一个终端时，它会立即连接到现有的树上。 因此，即使算法不能一次看到整个输入，而是逐个揭示终端，并且算法必须在每个阶段产生一棵斯坦纳树，贪婪斯坦纳算法的输出树的成本也不会超过最优树成本的$O(l\log|S|)$倍。

为了证明定理$_2$，我们引入一些符号。 让$c(i)$表示在第i次迭代中用于连接终端$s(i)$到已有树的路径P_i的成本。 显然，树的总成本是$\sum_{i=1}^{|S|} c(i)$。 现在，让$\{i_1, i_2, \ldots i^{|S|}\}$是$\{_1, _2, \ldots ^{|S|}\}$的一个排列，使得$c(i_1) \geq c(i_2) \geq \ldots \geq c(i^{|S|})$。（也就是说，按照算法考虑终端时所支付的成本降序重新标记终端。）

声明 **3** 对于所有 j，成本 $c(i_j)$最多为 $2\mathrm{OPT}/j$，其中 OPT是给定实例的最优解的成本。

证明：假设这不是真的，通过反证法；由于 s_i 是连接成本第 j高的终端，必须有 j个终端每个支付超过 $2\mathrm{OPT}/j$来连接到存在的树。令 $S' = \{s_{i_1}, s_{i_2}, \ldots s_{i_j}\}$表示这组终端。我们证明 $S' \cup \{s_1\}$中的任意两个终端之间的距离不超过 $2\mathrm{OPT}/j$。如果存在一对终端x, y在这个距离内，其中一个终端（假设是 y）必须比另一个终端更晚被算法考虑。但是，连接 y到已经存在的树（包括 x）的成本必须最多为 $2\mathrm{OPT}/j$，这与前提矛盾。

因此，在 $S' \cup \{s_1\}$中，任意两个终端之间的最小距离必须大于$2\mathrm{OPT}/j$。 由于这些终端中的任何MST必须有 j条边，所以MST的成本必须大于$2\mathrm{OPT}$。 但是终端子集的MST成本不能超过$2\mathrm{OPT}$，正如引理1的证明中所讨论的那样。 因此，我们得到了一个矛盾。

□

鉴于这一论断，证明定理2是很容易的。

$$\sum_{i=1}^{|S|} c(i) = \sum_{j=1}^{|S|} c(i_j) \leq \sum_{j=1}^{|S|} \frac{2\mathrm{OPT}}{j} = 2\mathrm{OPT} \sum_{j=1}^{|S|} \frac{1}{j} = 2H_{|S|} \cdot \mathrm{OPT}。$$

问题：给出一个图和终端的排序，使得贪婪算法的输出是$\Omega(\log |S|)\mathrm{OPT}$的一个例子。

备注：我们再次强调，上述分析适用于终端的任何排序。 一个自然的变体可能是自适应地对终端进行排序，以便在每次迭代 i中，算法选择已经存在的树 T中最接近的终端i。

你看到这与使用Prim算法的MST启发式方法等效吗？ 这
说明了在启发式方法的设计和分析中需要小心。

Steiner树的其他结果

使用MST启发式方法的2近似算法不是目前已知的Steiner树问题的最佳近似算法。 关于这个问题的其他结果如下所示。

1. 第一个获得比2更好比率的算法是由Alexander Zelikovsky提出的[13]；该算法的近似比率为$11/6 \approx 1.83$。 直到最近，在一般图中已知的最佳近似比率是$1 + \frac{\ln 3}{2} \approx 1.55$；这是通过[11]的算法实现的，它将MST启发式与局部搜索相结合。最近，Byrka等人提出了一种近似比为1.39 [4]的算法，目前这是该问题已知的最佳算法。

2. Steiner树的双向割LP松弛问题由[7]提出；它的整数间隙最多为$2(1 - \frac{1}{|S|})$，但有猜想认为间隙更小。目前还没有已知的算法利用这个LP松弛问题来获得比SteinerMST算法更好的近似比。虽然真正的整数间隙未知，但由于[8, 10]的例子表明它至少为$8/7 \approx 1.14$。

3. 对于许多应用程序，顶点可以被建模为平面上的点，它们之间的距离就是欧几里得距离。 基于最小生成树的算法在这种情况下表现得相当好；它的近似比率为$2/\sqrt{3} \approx 1.15$ [6]。 一个实现这个界限的例子是一个等边三角形的三个角上的三个点，边长为1；最小生成树启发式算法输出的树的成本为2（三角形的两条边），而最优解是将这三个点连接到一个斯坦纳顶点，该顶点是三角形的外心。 对于平面上的实例（或任何小维度的欧几里得空间），对于任何$\epsilon > 0$，存在一个$1 + \epsilon$-近似算法，其运行时间为多项式时间[1]。 对于平面图[3]和更一般的有界亏格图，也已经知道这样的近似方案。

参考文献

[1] S. Arora. 多项式时间近似方案用于欧几里得TSP和其他几何问题。A*CM*杂志45(5): 753–782, 1998.

[2] M. Bern和P. Plassmann. 边长为1和2的Steiner问题。 信息处理信件32, 171–176, 1989.

[3] G. Borradaile，C. Mathieu和P. Klein. 平面图中Steiner树的多项式时间近似方案。第十八届ACM-S*I*AM离散算法年会论文集（S*ODA*）, 2007, pp. 1285–1294.

[4] J. Byrka，F. Grandoni，T. Rothvoss和L. Sanita. 用于Steiner树的改进的基于LP的近似算法。A*CM*理论计算机科学研讨会（ST*OC*）, 583–592, 2010.

[5] M. Chlebik和J. Chlebikova. 图上Steiner树问题的近似难度. 第*8*届斯堪的纳维亚算法理论研讨会, Springer-Verlag, 170–179,2002年.

[6] D. Z. Du和F. K. Hwang. 关于Steiner比例的Gilbert-Pollak猜想的证明. *Algorithmica*, 7: 121–135, 1992年.

[7] J. Edmonds. 最优分支.国家标准研究所研究报告*, B*部分*,* 71:233–240*,* 1967年.

[8] M. Goemans. 未发表手稿, 1996年.

[9] M. Imaze和B.M. Waxman. 动态Steiner树问题. *SIAM*离散数学杂志, 4(3): 369–184, 1991年.

[10] J. Konemann, D. Pritchard, and K. Tan. 用于Steiner树的基于分区的松弛算法。手稿，2007年。arXiv:0712.3568,http://arxiv.org/abs/0712.3568v1

[11] G. Robins和A. Zelikovsky. 图Steiner树近似的更紧密界限。S*I*A*M*离散数学杂志，19(1)：122–134，2005年。

[12] J. Takahashi和A. Matsuyama. 图中Steiner问题的近似解。*Math. Jap.*，24：573–577，1980年。

[13] A. Zelikovsky. 用于网络Steiner问题的11/6近似算法。 Algorithmica，9：463–470，1993年。

CS 598CSC：近似算法	讲座日期：2011年1月21日
讲师：Chandra Chekuri	记录员：Sungjin Im（2009年）

讲师于2011年编辑的笔记。

在上一堂课中，我们快速概述了近似算法的几个基本方面。 我们还讨论了Steiner树问题的近似（离线和在线）算法。在本讲座中，我们将探讨另一个重要问题-旅行商问题（TSP）。

1 旅行商问题（TSP）

1.1 无向图中的TSP

在旅行商问题（TSP）中，我们给定一个无向图 $G=$（V，E）和每条边$e \in E$的成本 c（ e） >0。 我们的目标是找到一条具有最小成本的哈密顿回路。 如果一个循环中的每个顶点 V恰好访问一次，则称其为哈密顿回路。

TSP被认为是NP难的。此外，除非 $P=NP$，否则我们不能希望找到一个好的近似算法。 这是因为如果我们能够在多项式时间内给出TSP的一个好的近似解，那么我们就可以在多项式时间内精确解决NP完全的哈密顿回路问题（HAM），除非 $P=NP$。 回想一下，HAM是一个决策问题：给定一个图 $G=(V,E)$， G是否有一个哈密顿回路?

定理1（[3]）：让 $\alpha : \mathbb{N} \to \mathbb{N}$是一个多项式时间可计算的函数。 除非 $P=NP$，否则没有多项式时间算法能够在每个**T***SP*实例 I上输出一个成本不超过$\alpha(|I|) \cdot \mathrm{OPT}(I)$的解。

证明：为了推导矛盾，假设我们有一个近似算法 $\mathcal{A}$用于TS**P**，其近似比为 $\alpha(|I|)$。 我们通过展示使用 $\mathcal{A}$，我们可以在多项式时间内精确解决HAM来得到一个矛盾。 设 $G=(V,E)$是给定的HAM实例。我们创建一个新的图 $H=(V,E')$，对于每个$e \in E'$，其成本 $c(\ e)$为1如果$e \in E$，否则 $c(\ e) = B$，其中 $B = n\alpha(n)+2$且 $n = |V|$。 注意，这是一个多项式时间归约，因为 α是一个多项式时间可计算的函数。

我们观察到如果 G有一个哈密顿回路， $\mathrm{OPT} = n$，否则 $\mathrm{OPT} \geq n-1+B \geq n\alpha(\ n)+1$。（这里， OPT表示 H中最优TSP解的成本。）注意当 G有一个哈密顿回路和没有哈密顿回路之间存在一个“间隙”。因此，如果 $\mathcal{A}$的近似比率为 $\alpha(\ n)$，我们可以判断 G是否有哈密顿回路：只需在图 H上运行 $\mathcal{A}$；如果 $\mathcal{A}$返回一个成本不超过 $\alpha(\ n)\ n$的TSP路径，则输出 G有一个哈密顿回路，否则输出 G没有哈密顿回路。 我们将其留作一个练习，正式验证这将在多项式时间内解决HAM问题。

□

由于我们甚至不能近似解决一般的TSP问题，我们考虑更易处理的变体。

- 度量*TSP*：在度量*TSP*中，实例是一个完全图 $G=(V,E)$，其中 $c(e)$是 E上的成本，满足三角不等式，即 $c(uw) \leq c(uv)+c(vw)$，对于任意的$u,v,w \in V$。

- *TSP-R*：允许顶点重复的TSP。 输入是一个图 $G=(V, E)$，具有非负边缘成本，就像TSP一样。现在我们寻找一个最小成本的 $walk$，它至少访问每个顶点一次并返回到起始顶点。

练习：证明度量TSP的 α-近似意味着TSP-R的 α-近似，反之亦然。

我们在本节的其余部分将重点放在度量TSP上。 我们首先考虑一种自然的贪婪方法，即最近邻启发式算法（NNH）。

最近邻启发式算法$(G(V, E), c : E \to \mathcal{R}^+)$：
从任意顶点开始 s，
当存在未访问的顶点时
从当前顶点 u，前往最近的未访问顶点 v。
返回 s。

练习：

1. 证明NNH是一个 $O(\log n)$-近似算法。 (提示：回想一下贪婪斯坦纳树算法的$2H_{|S|}$-近似证明。)
2. NNH不是一个 $O(1)$-近似算法；你能找到一个例子来证明吗？ 事实上，NNH实现的近似比可以证明为$\Omega(\log n)$的下界。

对于TSP存在常数近似算法；我们现在考虑基于MST的算法。参见图1。

TSP-MST$(G(V, E), c : E \to \mathcal{R}^+)$:
计算图 G的MST T。
通过将T的边加倍，获得一个欧拉图$H = 2T$。$2T$的欧拉游览在G中给出一个游览。通过简化游览来获得一个哈密顿循环。

定理2 最小生成树启发式算法*(***T***SP-MS***T***)*是一个2-近似算法。

证明：我们有$c(T) = \sum$　　对于$_T$中的每条边e，$c_{(e_)} \leq \mathrm{OP}_T$，因为我们可以通过从最优哈密顿循环中删除任意边来获得G中的生成树，而$_T$是最小生成树。因此$c_{(H_)} = 2c_{(T)} \leq 2\mathrm{OP}_T$。 此外，简化只会减少成本。 □

我们观察到近似比例中的2倍损失是由于加倍边缘；我们这样做是为了获得一个欧拉游览。 但是，任何所有顶点度数都是偶数的图都是欧拉图，因此可以通过仅在T中的奇数度顶点之间添加边缘来获得欧拉游览。 Christofides启发式算法[2]利用这一点，将近似比例从2改进到$3/2$。

参见图2以获取快照。

克里斯托菲德算法（G（V，E），c：$E \to \mathcal{R}^+$）
：计算G的最小生成树T。
令S为T中奇数度的顶点。 （注意：$\lvert S\rvert$是偶数）在G中找到S的最小成本匹配M。将M添加到T以获得欧拉图H。计算H的欧拉回路。通过捷径将回路转化为哈密顿回路。

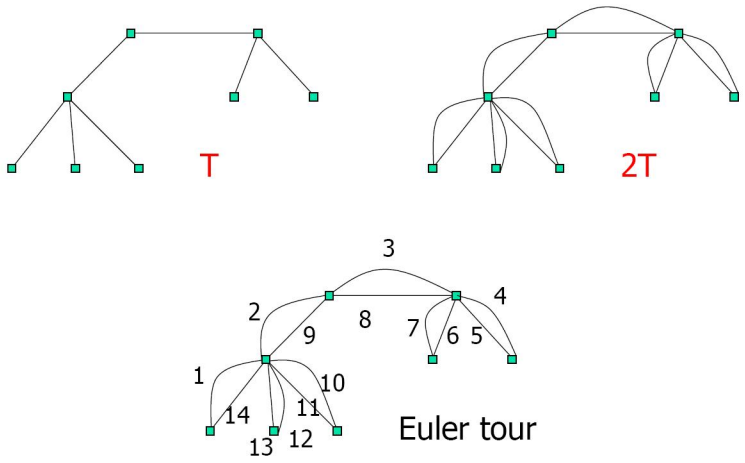

图1：基于最小生成树的启发式算法

定理3：克里斯托菲德算法是一个*1.5*近似算法。

证明：证明的主要部分是要证明$c(M) \leq 1.5\mathrm{OP}T$。 假设$c(M) \leq 1.5\mathrm{OP}T$。由于克里斯托菲德算法的解是通过在$H$上捷径欧拉回路得到的，其成本不超过$c(H) = c(T) + c(M) \leq 1.5\mathrm{OP}T$。 （参考引理2的证明，得到$c(T) \leq \mathrm{OP}T$。）因此，我们重点证明$c(M) \leq 1.5\mathrm{OP}T$。

令 F^* 为 G 中的最优旅行，其成本为 OPT；由于我们有一个度量实例，我们可以假设不失一般性地 F^* 是一个哈密顿环。 我们通过缩短与顶点 $V \setminus {}_S$ 相接触的 F^* 的部分，在图 $G[{}_S]$ 中获得一个哈密顿环 ${F_S}^*$。根据度量条件，$c({F_S}^*) \leq c(F^*) = \mathrm{OPT}$。 令 ${}_S = \{v_1, v_2, ..., {}_{|S|}\}$。 不失一般性地，${F_S}^*$ 按照顺序访问了 ${}_S$ 中的顶点 $v_1, v_2, ..., {}_{|S|}$。 回想一下，${}_{|S|}$ 是偶数。 注意，M_1 和 M_2 都是匹配，并且 $c(M_1) + c(M_2) = c({F_S}^*) \leq \mathrm{OPT}$。 我们可以假设不失一般性地 $c(M_1) \leq c(M_2)$。 然后我们有 $c(M_1) \leq 0.5 * \mathrm{OPT}$。 我们还知道 $c(M) \leq c(M_1)$，因为 M 是 $G[{}_S]$ 上的最小成本匹配。 因此，我们有 $c(M) \leq c(M_1) \leq 0.5 * \mathrm{OPT}$，这完成了证明。

□

注释：

1. 在实践中，局部搜索启发式算法被广泛使用，并且它们表现非常好。 一种流行的启发式算法 *2-O*pt是交换xy、*z*w对到x*z*、yw或xw、y*z*，如果它改善了路径。

2. 自从1976年发现Christofide*s*启发式算法以来，Metric-TSP没有任何改进。 改进Metric-TSP的近似比率为 ${}_2$ [3]仍然是一个重要的开放问题；有人猜测Held-Karp LP松弛[4]给出了一个$^3\,{}^4$的比率。 最近，Ove*i*s-Gharan、Sab*e*ri和Singh宣布了一个关于重要特殊情况的突破性进展，他们声称对于一些小但固定的$\delta > 0$，$c(e) = 1$对于每条边e，有一个$^3/_2$-δ的近似。

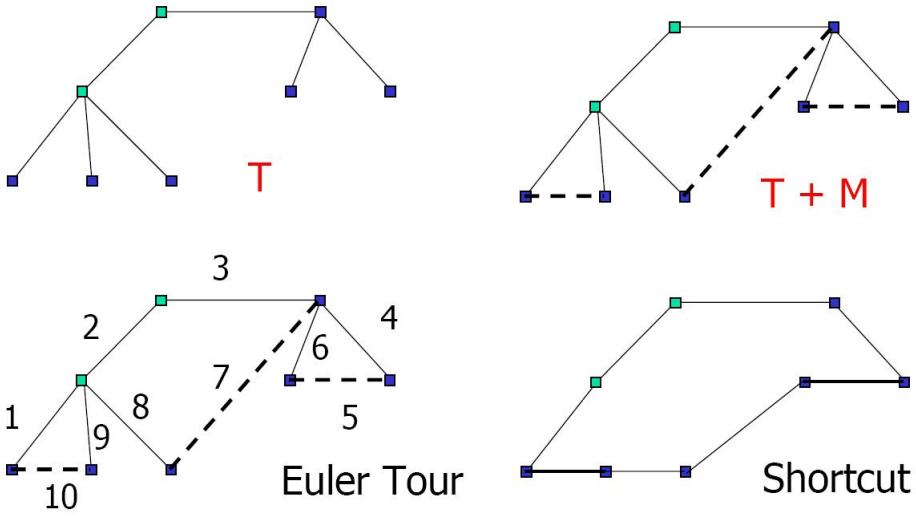

图2：克里斯托菲德启发式

1.2 有向图中的TSP

在这个小节中，我们考虑有向图中的TSP问题。 与无向TSP一样，我们需要放宽问题条件以获得任何正结果。 同样，允许每个顶点被多次访问等价于对所有的u、v、w，施加非对称三角不等式 $c(u,w) \leq c(u,v) + c(v,w)$。 这被称为非对称TSP（ATSP）问题。 我们给定一个有向图$G=(V,A)$，每个弧$a \in A$的成本 $c(\ a) >0$，并且我们的目标是找到一个访问所有顶点的闭合路径。 请注意，我们允许多次访问每个顶点，因为我们寻找的是一条路径，而不是一个循环。 关于一个有效的哈密顿路径的示例，请参见图3。

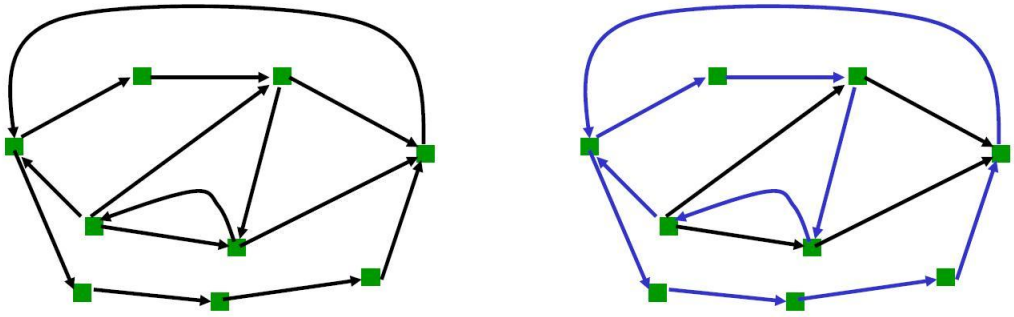

图3：一个有向图和一个有效的哈密顿路径

基于最小生成树的启发式算法在无向情况下没有有意义的推广到有向情况。这是因为边上的成本不对称。 因此，我们需要另一种方法。
循环缩减算法反复找到最小成本的循环覆盖并缩小循环，结合找到的循环覆盖。 回想一下，循环覆盖是覆盖所有顶点的一组不相交循环。 已知在多项式时间内可以找到最小成本的循环覆盖（见

作业0）。循环缩小算法实现了对数$_2n$的近似比率。

循环缩小算法（G（V，A），c：$A \to \mathcal{R}^+$）：
将G转换为完全图，并满足c（u，v）$+c$（v，w）$\geq c$（u，w）对于所有的u，v，w。如果$|V|=_1$，则输出由单个节点组成的平凡循环。找到一个最小成本的循环覆盖，其中包含循环C_1，...，C_k。 从每个C_i中选择一个任意的代理节点v_i。

递归地在 $G[\{v_1, \ldots\ldots\ldots\ldots\ldots, v_k\}]$上解决问题，得到解 C
$C' = C \cup C_1 \cup C_2 \ldots c_k$ 是一个欧拉图。
通过 C'快捷方式得到一个在 V 上的循环，并输出 C'。

有关循环缩小算法的快照，请参见图4。

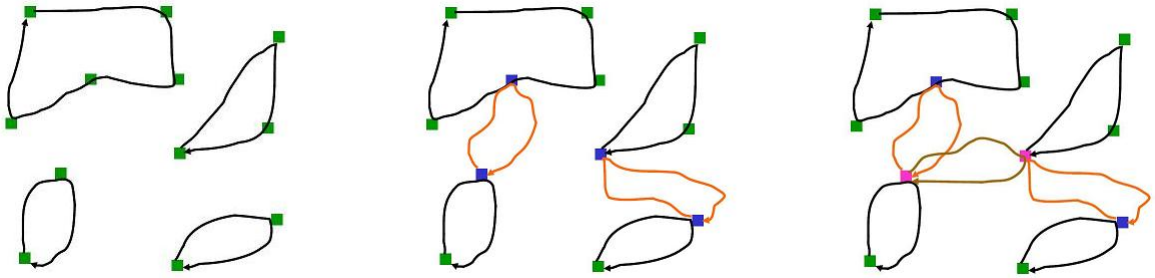

图4：循环缩小算法的快照。左边是一个循环覆盖 $\mathcal{C}_1$。 在中间，
蓝色顶点表示代理节点，并在代理节点上找到一个循环覆盖 $\mathcal{C}_2$。 右边，
粉色顶点是新的代理节点，并在新的代理节点上找到一个循环覆盖 $\mathcal{C}_3$。

引理**4**假设 G中边的成本满足非对称三角不等式。 那么对于任意的
$S \subseteq V$， $G[S]$中最优TSP路径的成本至多等于 G中最优TSP路径的成本。

证明：由于G满足三角不等式，所以在G中存在一个最优的TS**P**路径，它是一个哈密顿环C。 对于任意的$S \subseteq V$，可以通过对循环C进行简化来得到另一个只访问S并且成本不超过C的循环C'。

□

引理**5**：最小成本循环覆盖的成本不超过最优TSP路径的成本。

证明：最优TS**P**路径是一个循环覆盖。 □

定理**6**：循环缩小算法是A**T**SP的$\boldsymbol{lo}g_2n$近似算法。

证明：我们通过对G中的节点数n进行归纳来证明上述结论。 很容易看出，如果$n \leq 2$，则算法找到了一个最优解。 主要观察结果是循环覆盖中的循环数最多为$\lfloor n/2 \rfloor$；这是因为覆盖中的每个循环至少有2个节点，并且它们是不相交的。因此，$k \leq \lfloor n/2 \rfloor$。 令O**P**T$(S)$表示$G[S]$中最优解的成本。 根据引理4，我们有O**P**T$(S) \leq$O**P**T$(V)=$O**P**T对于所有$S \subseteq V$。 算法对代理节点$S=\{v_1,...,vk\}$进行递归。 注意，$|S|<n$，并且根据归纳法，递归调用输出的循环C的成本不超过$(\mathbf{log}2|S|)$O**P**T$(S) \leq (\mathbf{log}2|S|)$O**P**T。

该算法输出C'，其成本最多为C的成本加上在G中计算的循环覆盖的成本。 循环覆盖的成本最多为OPT（引理5）。 因此，C'的成本最多为$(\log_2|S|)\mathrm{OPT} + \mathrm{OPT} \leq (\log_2 n/_2)\mathrm{OPT} + \mathrm{OPT} \leq (\log_2 n)\mathrm{OPT}$；这完成了归纳证明。

□

注释：

1。 循环收缩算法的运行时间为$O(T(n))$，其中$T(n)$是找到最小成本循环覆盖的时间（为什么？）。 在作业0中，你有一个问题，将这个问题归约为在无向图中找到最小成本完美匹配的问题。 这可以在$O(n^{\wedge}3)$的时间内完成。 通过近似最小成本循环覆盖问题，可以将运行时间提高到$O(n^{\wedge}2)$；但会损失额外的常数因子。

2. 关于ATSP是否存在常数近似解已经是一个悬而未决的问题超过25年了。最近，Asadpour等人[1]使用一些非常新颖的思想和一个众所周知的LP松弛方法，获得了一个$O(\log n \,/\, \log\log n)$的近似解。

2 一些定义

2.1 NP优化问题

在本节中，我们介绍一些与近似算法相关的正式定义。 我们从优化问题的定义开始。 一个问题只是一个无限集合的实例。 设Π是一个优化问题。 Π可以是一个最小化或最大化问题。 Π的实例I是Σ^*的一个子集，其中Σ是一个有限的编码字母表。 对于每个实例I，有一组可行解$\mathcal{S}(I_)$。 我们将注意力限制在实数/有理数值优化问题上；在这些问题中，每个可行解$\mathcal{S}\in\mathcal{S}(I_)$都有一个值$\mathrm{val}(\mathcal{S}, I_)$。 对于一个最小化问题$\Pi$，目标是给定$I$，找到$\mathrm{OPT}(I_) = \min_{\mathcal{S}\in\mathcal{S}(I_)}\mathrm{val}(\mathcal{S}, I_)$。

现在让我们正式定义NP优化（NPO），它是与NP对应的优化问题类。

定义**7** Π 在NPO中

- 给定 $x \in \Sigma^*$，存在一个多项式时间算法来判断 x是否是 Π的有效实例。
  也就是说，我们可以高效地检查输入字符串是否格式良好。 这是一个基本要求，通常没有明确说明。

  也就是说，解的大小多项式地与输入大小相关。

- 存在一个多项式时间决策过程，对于每个 I和 $S \in \Sigma^*$，决定 S是否属于$\mathcal{S}$（I）。
  这是NP的关键属性；我们应该能够高效地验证解。

- $val(I, S)$是一个多项式时间可计算的函数。

我们观察到，对于一个最小化的NPO问题Π，存在一个相关的自然决策问题 $L(\Pi) = \{(I, B) : \mathrm{OPT}(I) \leq B\}$，即给定$\Pi$的实例 I和一个数字 B，最优值在 I上至多为 B吗？ 对于最大化问题Π，我们反转了定义中的不等式。

引理**8** $L(\Pi)$在$N\,P$中，如果 Π在NPO中。

2.2 相对近似

当Π是一个最小化问题时，回想一下，我们说一个近似算法 $\mathcal{A}$具有近似比 α当且仅当

- $\mathcal{A}$是一个多项式时间算法
- 对于Π的所有实例 I， $\mathcal{A}$产生一个可行解 $\mathcal{A}(I)$ 使得 $val(\mathcal{A}(I), I) \leq \alpha\, val\,(\mathrm{OPT}(I), I)$。（注意 $\alpha \geq 1$。）

最大化问题的近似算法类似地定义。 近似算法$\mathcal{A}$的近似比α定义为

- $\mathcal{A}$是一个多项式时间算法
- 对于问题Π的所有实例I，算法$\mathcal{A}$产生一个可行解$\mathcal{A}(I)$，使得$\mathrm{val}(\mathcal{A}(I), I) \geq \alpha\, \mathrm{val}(\mathrm{OPT}(I), I)$。（注意$\alpha \leq 1$。）

对于最大化问题，通常也会使用$1/\alpha$（必须≥ 1）作为近似比。

2.3 加法近似

注意上述所有定义都是关于相对近似的；也可以定义加法近似。 如果对于所有实例I，算法$\mathcal{A}$的值$\mathrm{val}(\mathcal{A}(I)) \leq \mathrm{OPT}(I) + \alpha$，则称$\mathcal{A}$为$\alpha$-加法近似算法。. 然而，大多数NPO问题不允许任何加法近似比，因为$\mathrm{OPT}(I)$具有缩放性质。

为了说明缩放性质，让我们考虑度量旅行商问题(Metric-TSP)。给定一个实例 I，让 I_β表示通过将所有边的成本增加 β倍得到的实例。 很容易观察到对于每个 $S \in \mathcal{S}(I) = \mathcal{S}(I_\beta)$， $val(S, I_\beta) = \beta\ val(S, I_\beta)$和 $\mathrm{OPT}(I_\beta) = \beta \mathrm{OPT}(I)$。 直观地，将边缩放 β倍会使值也按相同的因子 β缩放。 因此，通过选择足够大的 β，我们可以使加法近似（或误差）变得可以忽略不计。

引理**9**：度量旅行商问题(*Metric-TSP*)不具有任何多项式时间可计算的 α加法近似算法，除非 $P = NP$。

证明：为了简化起见，假设每条边的成本是整数。 为了推导矛盾，假设存在一个加法 α近似算法 $\mathcal{A}$用于度量-旅行商问题。给定 I，我们在 I_β上运行算法，并让 S成为解决方案，其中 $\beta = 2\alpha$。 我们声称 S是 I的最优解。 我们有$val(S, I) = val(S, I_\beta)/\beta \leq \mathrm{OPT}(I_\beta)/\beta + \alpha/\beta = \mathrm{OPT}(I) + 1/2$，因为 $\mathcal{A}$是 α-近似算法。
因此，我们得出结论 $\mathrm{OPT}(I) = val(S, I)$，因为 $\mathrm{OPT}(I) \leq val(S, I)$，而 $\mathrm{OPT}(I)$、 $val(S, I)$都是整数。 除非 $P = NP$，否则这是不可能的。 □

现在让我们考虑两个允许加法逼近的问题。 在平面图着色问题中，我们给定一个平面图 $G = (V, E)$。 我们被要求给定图 G的所有顶点着色，使得对于任意 $vw \in E$， v和 w具有不同的颜色。 目标是最小化不同颜色的数量。 已知决定一个平面图是否能够进行3-着色是NP-完全的，而任何平面图 G都可以使用4种颜色进行着色。 此外，我们可以高效地检查一个图是否是2-可着色的（即是否是二分图）。 因此，以下算法是平面图着色问题的1-加法逼近算法：如果图是二分图，则使用2种颜色进行着色；否则，使用4种颜色进行着色。

作为第二个例子，考虑边缘着色问题，我们被要求用最少的不同颜色给给定图 G的边缘着色，以便相邻的两条边没有不同的颜色。 根据Vizing定理[6]，我们知道可以用G的最大度数$\Delta(G)$或$\Delta(G)+1$个不同的颜色来给边缘着色，其中$\Delta(G)$是G的最大度数。 由于$\Delta(G)$是最小数量的一个显而易见的下界，我们可以说边缘着色问题允许1-可加近似。 请注意，已知决定精确最小数量是$\Delta(G)$还是$\Delta(G)+1$是NP完全的。

2.4 近似难度

现在我们转向近似难度。

定义**10**（近似阈值）给定一个最小化优化问题 Π，如果对于任何$\epsilon>0$， Π都有一个 $\alpha^*(\Pi)+\epsilon$近似解，但如果它有一个 $\alpha^*(\Pi)-\epsilon$近似解，则 $P=NP$。

如果$\alpha^*(\Pi)=1$，则意味着Π可以在多项式时间内求解。 许多NPO问题Π已知具有$\alpha^*(\Pi)>1$，假设$P=NP$。 我们可以说近似算法试图减小$\alpha^*(\Pi)$的上界，而近似难度试图增加$\alpha^*(\Pi)$的下界。

为了证明关于近似的NPO问题的难度结果，主要有两种方法：一种是通过从NP完全问题进行约简的直接方法，另一种是通过间隙约简的间接方法。 现在让我们快速看一个例子，使用从一个NP完全问题的约简。

在（度量）k中心问题中，我们给定一个无向图$G=(V, E)$和一个整数k。 我们被要求从V中选择一个称为中心的k个顶点的子集。 目标是最小化到中心的最大距离，即$\min_{S\subseteq V, |S|=k} \max_{v\in V} \operatorname{dist}(v, S)$，其中$\operatorname{dist}(v, S)=\min_{u\in S} \operatorname{dist}(u, v)$。

k-中心问题的近似阈值为2，因为存在一些2-近似算法用于k-中心问题，并且对于任何大于0的中心不存在$2-\epsilon$近似算法，除非$P=NP$。我们可以通过从支配集的决策版本进行约简来证明不可近似性：给定一个无向图$G=(V, E)$和一个整数k，G是否有一个大小不超过k的支配集？ 在G中，如果对于所有$v\in V$，$v\in S$或者v与S中的某个u相邻，则集合S被称为G的支配集。 已知支配集问题是NP完全的。

定理**11**（[5]）：除非$P=NP$，否则对于任何固定的NPk-中心问题不存在$2-\epsilon$近似算法。

证明：设I是一个由图$G=(V, E)$和整数k组成的支配集问题的实例。 我们创建一个实例I'，保持图G和k不变，成为k-center。 如果I有一个大小为k的支配集，那么$OPT(I')=1$，因为每个顶点都可以通过最多一次跳跃从支配集到达。否则，我们声称$OPT(I')\geq 2$。 这是因为如果$OPT(I')<2$，那么每个顶点必须在距离1内，这意味着证明$OPT(I')$的k-center是一个I的支配集。 因此，(2 - ε)近似算法可以用于解决支配集问题。这是不可能的，除非$P=NP$。

□

参考文献

[1] A. Asadpour, M. Goemans, A. Madry, S. Oveis Gharan, and A. Saberi. 一种 $O(\log n/\log\log n)$-近似算法用于非对称旅行商问题。*ACM-SIAM SODA*, 2010.

[2] N. Christofides. 旅行商问题的一种新启发式算法的最坏情况分析。技术报告，工业管理研究生院，*CMU* 1976年。

[3] S. Sahni和T.F. Gonzalez. P-Complete近似问题。*ACM*23: 555-565, 1976年。

[4] M. Held和R. M. Karp. 旅行商问题和最小生成树。运筹学18: 1138–1162, 1970年。

[5] W. L. Hsu和G.L. Nemhauser. 简单和困难的瓶颈位置问题。离散应用数学. 1:209-216, 1979年。

[6] D. West. 图论导论. 第2版. P*rentice* H*a*ll. 277-278, 2001.

CS 598CSC：近似算法	讲座日期：2011年1月26日
讲师：Chandra Chekuri	记录员：Md. Abul Hassan Samee（2009年）

讲师于2011年编辑的笔记。

1 引言

在本讲座中，我们讨论了两个密切相关的NP优化问题，即集合覆盖和最大覆盖。 集合覆盖是最早被分析近似算法的问题之一。. 从实际角度来看，这个问题也非常重要，因为该问题本身及其几个推广问题在许多应用领域中经常出现。 在本讲座中，我们将考虑三个集合覆盖的推广问题。 最后，我们简要讨论了如何用子模函数来表述集合覆盖问题。

2 集合覆盖和最大覆盖

2.1问题定义

在集合覆盖和最大覆盖问题中，我们的输入是一个包含n个元素的集合$\mathcal{U}$，以及一个包含m个子集的集合$\mathcal{S}=\{\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_m\}$在集合覆盖问题中，我们的目标是从$S$中选择尽可能少的子集，使它们的并集覆盖$\mathcal{U}$。 在最大覆盖问题中，输入还指定了一个整数$k \leq m$，我们的目标是从$S$中选择$k$个子集，使它们的并集具有最大的基数。 请注意，前者是一个最小化问题，而后者是一个最大化问题。 我们还可以考虑这些问题的加权版本，这将在以后的讲座中讨论。

2.2贪婪近似

集合覆盖和最大覆盖都被认为是**NP**-难问题。 这些问题的一种自然贪心近似算法如下所示。

贪心覆盖$(\mathcal{U}, \mathcal{S})$：
1：重复执行
2：　　选择覆盖未覆盖元素最多的集合
3：　　标记选择的集合中的元素为已覆盖
4：直到完成

对于集合覆盖问题，当集合$\mathcal{U}$中的所有元素都被覆盖时，算法贪心覆盖在第4行完成。 对于最大覆盖问题，当从集合$\mathcal{S}$中选择了恰好k个子集时，算法完成。

2.3 贪心覆盖的分析

定理1 贪心覆盖是最大覆盖的$1-(1-1/k)^k \geq (1-1/e) \simeq 0.632$近似算法，并且是集合覆盖的$(\ln n + 1)$近似算法。

根据Feige [1]的以下定理，它暗示了贪婪覆盖在近似比率方面是最好的。

定理**2**：除非$\boldsymbol{NP}\subseteq\boldsymbol{DTIME}(n^{O(log\ log\ n)})$，否则不存在$(1-(1))lnn$的近似解。 除非$\boldsymbol{P}=\boldsymbol{NP}$，对于任意固定的$\varepsilon>0$，不存在$(1-1_e-\varepsilon)$的近似解。

我们通过分别分析集合覆盖和最大覆盖的贪婪覆盖来证明定理1。 令OPT表示最大覆盖问题的最优解的值。 令x_i表示贪婪覆盖在第$_i$个集合中覆盖的新元素的数量。 同时，令$y_i = \sum_{ij}=1x_i$，$z_i = OPT$ - y_i。 注意，根据我们的符号表示，$y_0 = 0$，y_k是贪婪覆盖选择的元素数量，$z_0 = OPT$。

最大覆盖的分析

我们对于算法贪心覆盖在最大覆盖问题上有以下引理。

引理**3**：贪心覆盖是一种$1 - (1 - 1/^k)\text{^}k \geq 1 - 1/_e$的近似算法，用于最大覆盖问题。

我们首先证明以下两个命题。

命题**4**：$x_{i+1} \geq {}^{zi}.$

证明：在每一步中，贪心覆盖选择覆盖未覆盖元素最多的子集S_j。 由于最优解使用k个集合来覆盖OPT个元素，某个集合必须覆盖至少$1/k$比例的至少z_i个剩余未覆盖元素。因此，$x_{i+}1 \geq z_i.$

□

命题**5**：$z_{i+1} \leq (1 - {}_k\text{^}(-1))(_{i+1}) \cdot OPT$

证明：对于$i = 0$，该命题成立。 我们归纳地假设$zi \leq (1 - {}_k i\text{-}1) \cdot OPT$。那么

$$\begin{aligned} z_{i+1} &\leq z_i - x_{i+1} \\ &\leq z_i(1 - \frac{1}{k}) \text{ [使用命题4]} \\ &\leq (1 - \frac{1}{k})_{i+1} \cdot OPT。 \end{aligned}$$

□

引理**3**的证明。 根据命题5可知$z_k \leq (1 - {}_k\text{i-}1) \cdot {}_k \cdot OPT \leq OPT$因此，$y_k = OPT - z_k \geq (1 - 1/_e) \cdot OPT$。

□

集合覆盖的分析。

我们有以下引理。

引理**6**：贪婪覆盖是集合覆盖的$(\ln n + 1)$近似算法。

让 k^*表示SET COVER问题的最优解的值。 那么，对于 $k = k^*$的MAXIMUM COVERAGE问题的最优解，将覆盖集合$\mathcal{U}$中的所有 n个元素，并且 $z_{k^*} \leq {}^{n}{}_{e}$。 因此，在GREEDY COVER的前 k^*步之后，将有 ^{ne}个元素仍然未被覆盖。 类似地，在 $\mathrm{Gr}_{ee}\mathrm{dy}\ \mathrm{Cov}_{e}\mathrm{r}$的前$2 \cdot k^*$步之后，将有 ^{ne}个元素仍然未被覆盖。 $\frac{}{e^2}$ 元素将保持未覆盖状态。 这个简单的直觉使我们相信贪婪覆盖是集合覆盖问题的$(\ln n + 1)$近似算法。 下面给出了一个更简洁的证明。

引理**6**的证明。 由于 $z_i \leq (1 - \frac{1}{k^*})^i \cdot n$，在 $t = k^* \ln \frac{n}{k^*}$ 因此，在$_t$步之后，还剩下k^*个元素需要覆盖。 由于贪婪覆盖算法每一步至少选择一个元素，所以在选择最多$k^*\ln$个元素后，它覆盖了所有的元素。 $\frac{n}{k^*} + k^* \leq k^*(\ln n + 1)$个集合。 □

以下推论直接由引理6得出。

推论**7** 如果$|S_i| \leq d$，则贪婪覆盖算法是$Set\ \mathbf{Co}\mathrm{v}e\mathbf{r}$的$(\mathbf{l}^n\ d + 1)$近似算法。证明：由于$k^* \geq {}^{n}d$，所以$\mathbf{l}^n \quad - \quad \frac{n}{k^*} \quad \leq \ln d$。 然后根据引理6得出结论。 □

定理**1**的证明。这些结论直接由引理3和6得出。 □

当应用于**Set Cover**时，贪婪覆盖算法的一个紧密例子。

让我们考虑一个集合$\mathcal{U}$的n个元素，以及一个包含$k+_2$个子集$\{R_1, R_2, C_1, C_2, ..., Ck\}$的集合$\mathcal{S}$。 我们还假设$|C_i| = {}_{2i}$，并且$|R_1 \cap C_i| = |R_2 \cap C_i| = {}_{2i}{}^{-}{}_1\ ({}_1 \leq {}_i \leq k)$，如图$_1$所示。

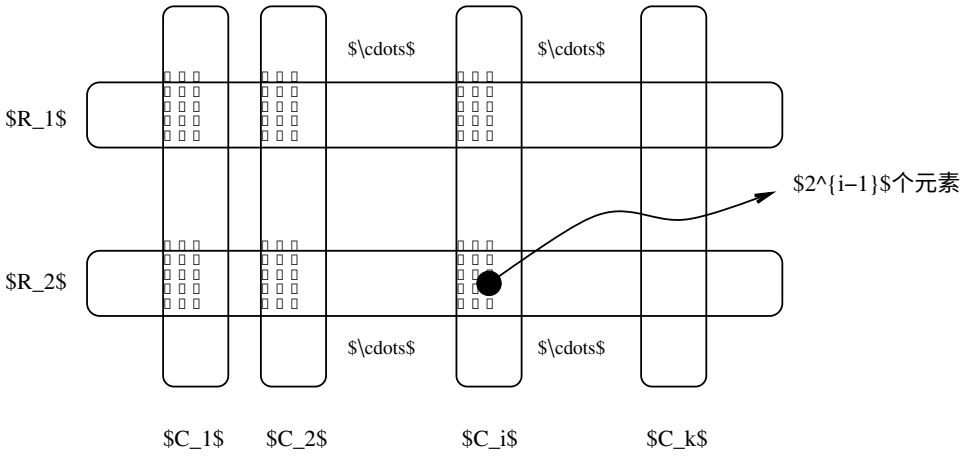

图1：当应用于集合覆盖时，贪婪覆盖的一个紧密示例

显然，最优解只包含两个集合，即R_1和R_2。因此，$OPT = {}_2$。
然而，贪婪覆盖将选择剩余的每个k个集合，即Ck，Ck-$_1$，...，C_1。 由于$n = 2 \cdot \sum(k\ {}_i = {}_0$-$_1)2\hat{}_i = 2 \cdot (2\hat{}k$-$_1)$，我们得到$k \approx \Omega(\log 2n)$。 因此，这个例子是紧密的。

练习：考虑集合覆盖问题的加权版本，其中给定一个权重函数w：$\mathcal{S} \to \mathcal{R}^+$，我们希望从$\mathcal{S}$中选择一个子集合$\mathcal{S}'$，使得$\cup_{X \in \mathcal{S}'} {}_X = \mathcal{U}$，并且$\sum_{X \in \mathcal{S}'} \mathrm{t}w(X)$是最小的。 证明贪婪启发式算法对于这个问题给出了一个$2 \cdot (\ln n + 1)$的近似解。

提示1：注意贪心算法从不选择成本超过OPT的集合。提示2：当贪心算法选择的集合的总成本首次超过OPT时，它已经覆盖了(1-1/e)的元素。

3 支配集和顶点覆盖

3.1 支配集

在图$G=(V, E)$中，支配集S是一个包含于V的集合，对于每个$u \in V$，要么$u \in S$，要么u的某个邻居$v \in S$。 在支配集问题中，我们的目标是找到一个最小的支配集G。

这个问题的一个自然贪心算法是迭代地选择具有最高度数的顶点。 可以证明，这个启发式算法给出了支配集问题的$(\ln n + 1)$或更准确地说，$(\ln (\Delta + 1) + 1)$的近似解。

练习：

1. 证明贪婪启发式算法对于支配集的近似保证。
2. 证明支配集是集合覆盖的特殊情况。
3. 证明集合覆盖可以以近似保持的方式减少为支配集。 更正式地，证明如果支配集在给定实例中有一个 $\alpha(n)$-近似，其中n是给定实例中的顶点数，则集合覆盖有一个 $(1 - o(1))\alpha(n)$-近似。

3.2 顶点覆盖

图 $G=(V, E)$ 的顶点覆盖是一个集合 $S \subseteq V$，使得对于每条边$(e \in E)$，至少一个端点在 S中。 在顶点覆盖问题中，我们的目标是找到一个最小的顶点覆盖。G。 在问题的加权版本中，给定一个权重函数 $w : V \rightarrow \mathcal{R}^+$，我们的目标是找到一个最小权重的顶点覆盖。 G。 问题的非加权版本也被称为基数顶点覆盖。

可以证明，贪婪覆盖算法可以给出顶点覆盖问题的加权和非加权版本的$O(\ln \Delta + 1)$近似解。

练习：

1. 证明顶点覆盖是集合覆盖的一个特例。
2. 构造一个例子，证明当贪婪覆盖算法应用于顶点覆盖问题时，给出的近似解是$\Omega(\log n)$。

3.2.1 更好的（常数）近似解顶点覆盖

基数顶点覆盖：以下是基数顶点覆盖问题的2近似算法。

匹配-顶点覆盖（G）：
1：$S \leftarrow \emptyset$
2：在G中计算一个最大匹配M
3：对于每条边（u, v）$\in M$，执行以下操作
4：　　将u和v都添加到S
5：输出S

定理8 匹配-顶点覆盖是一个2近似算法。

定理8的证明可以从两个简单的命题得出。

命题9：令OPT为最优解中顶点覆盖的大小。那么$OPT \geq |M|$。

证明：由于最优顶点覆盖必须包含M中每条边的至少一个端点，所以$O\mathbf{P}\ T \geq |M|$。

□

命题10：令$S(M)$=$\{u, v \mid (u, v) \in M\}$。 那么$S(M)$是一个顶点覆盖。

证明：如果$S(M)$不是一个顶点覆盖，那么一定存在一条边$e \in E$，使得它的两个端点都不在M中。但是，e可以被包含在M中，这与M的极大性矛盾。□ 定理8的证明：由于$S(M)$是一个顶点覆盖，命题9暗示着$|S(M)| = \square \cdot |M| \leq \square \cdot O\mathbf{P}\ \mathbf{T}$。

□

加权顶点覆盖：基于LP舍入或原始-对偶技术的加权顶点覆盖问题的2近似算法可以设计。 这些将在课程后面讲解。

3.2.2 带有小频率的集合覆盖

顶点覆盖是集合覆盖的一个实例，其中 $\mathcal{U}$中的每个元素最多在两个集合中（实际上，每个元素恰好在两个集合中）。 集合覆盖问题的这种特殊情况给出了一个2近似算法。 如果每个元素最多包含在三个集合中，会是什么情况？ 更一般地，给定一个集合覆盖的实例，对于每个 $e \in \mathcal{U}$，让 $f(e)$ 表示包含 e的集合的数量。 令 $f = \max_e f(e)$，我们称之为最大频率。

练习：给出一个 f-近似的集合覆盖算法，其中 f是一个元素的最大频率。提示：按照用于顶点覆盖的方法进行。

4贪婪近似算法的两个重要方面集合覆盖

4.1隐式实例的贪婪近似算法

事实证明，元素的宇宙$\mathcal{U}$和子集的集合$\mathcal{S}$在$\mathcal{S}$et Cover问题中不限于有限的或明确枚举的。 例如，一个问题可能需要用单位半径的圆盘覆盖平面上的有限点集。

有无限个这样的圆盘，但贪婪近似算法仍然可以应用。 对于这种隐式实例，如果我们可以访问一个oracle，在每次迭代中选择具有最佳密度的集合。 然而，oracle并不总是能够选择最优的集合。 在这种情况下，它可能需要进行近似选择。 如果在每次迭代中，oracle选择一个集合S，使得density$(S) \geq \alpha \cdot O$ptimal Density，其中$\alpha > 1$，则称oracle为α-近似oracle。 density(S)表示集合S的密度，Optimal Density表示最优密度。

练习：证明贪婪近似算法的近似保证是α-近似

对于集合覆盖问题，使用预言机的近似保证是$\alpha(\ln n + 1)$，对于最大覆盖问题是$(1 - e^{\wedge}\alpha)$ 对于最大覆盖问题，使用预言机的近

4.2 子模函数的贪婪近似

从更一般的角度来看，贪婪近似适用于任何子模集函数。 给定一个有限集合E，一个函数f: $2E \rightarrow \mathcal{R}^{+}$是子模的，如果对于所有的$A, B \subseteq E$，有$f(A)$ + $f(B) \geq f(A \cup B)$ + $f(A \cap B)$。 或者，f是子模函数，如果对于所有的$i \in E$和$A \subset B$，有$f(A^{+}i)$ - $f(A) \geq f(B^{+}i)$ - $f(B)$。 这个第二个特征是由子模函数的边际效用递减性质引起的。 直观上，将元素i添加到集合A将至少与将其添加到（更大的）集合B中一样有帮助，其中$A \subset B$。

练习：证明子模函数的两种特征是等价的。

如果对于所有的$i \in E$和$A \subseteq E$，子模函数 $f(\cdot)$ 是单调的，那么对于所有的 $i \in$E和$A \subseteq$E，有 $f($ $A+$i$) \geq f($A$)$。 我们假设 $f(\emptyset) = 0$。 子模集函数在许多实际领域中出现，包括组合优化、概率和几何学。 例如，一个拟阵的秩函数、有向或无向图中的割集大小、一组事件不同时发生的概率、随机变量的熵等。在接下来的内容中，我们将展示Set Cover和Maximum Coverage问题可以很容易地用子模集函数来表达。

练习。 假设我们有一个宇宙 $\mathcal{U}$和一个集合 $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}of\mathcal{U}$的子集。. 现在如果我们取 N $= \{1, 2, \ldots, m\}$， f: $2^{N} \rightarrow \mathcal{R}^{+}$，并定义 $f(A) =| \cup_{i \in A} S_i|$对于 $A \subseteq E$，那么证明函数 f是子模的。

4.2.1 子模集覆盖

当以子模集函数的形式表达时，集合覆盖问题如下。
给定一个单调子模集函数 f（其值由一个预言机计算），在 $N = \{1, 2, ..., m\}$ 上找到最小的集合 $S \subseteq N$，使得 $f(S) = f(N)$。 我们之前的贪婪近似算法可以应用于这个表述，如下所示。

贪婪子模集（f, N）：
1: $S \leftarrow \emptyset$
2: 当 $f(S) = f(N)$ 时
3: 找到 i 使得 $f(S + i)$ - $f(S)$ 最大化
4: $S \leftarrow S \cup \{i\}$

练习：

1. 证明贪婪算法是 SUBmodUlar Set Cover 的 $1 + \ln(f(N))$ 近似算法。

2. 证明贪婪算法是SUBMODULAR SET COVER问题的1 + ln (maxif(i)) 近似解。

4.2.2 SUBMODULAR MAXIMUM COVERAGE

通过将MAXIMUM COVERAGE问题转化为子模函数的形式，我们寻求最大化 $f(S)$ 的值，使得 $|S| \leq k$。 我们可以通过将第 2行的条件改为： **while** $|S| \leq k$，来应用贪婪子模算法解决这个问题。

注意。对于SUBMODULAR MAXIMUM COVERAGE问题，函数 f必须既是子模的，又是单调的。

练习：证明贪婪算法对于SUBMODULAR MAXIMUM COVERAGE问题的近似解是(1− 1/e)。

参考文献

[1] U. Feige. A Threshold of ln n for Approximating Set Cover. *J. of the ACM* 45(5): 634–652, 1998.

CS 598CSC：近似算法	讲座日期：1/28/11
讲师：Chandra Chekuri	记录员：Lewis Tseng（2009年）

讲师于2011年编辑的笔记。

我们介绍了线性规划（LP）在近似算法的设计和分析中的应用。主题包括顶点覆盖，集合覆盖，随机舍入，对偶适应。 假设学生们对线性规划的基础知识有一定的了解。

1 顶点覆盖通过LP

设 $G=（V，E）$是一个具有弧权重 w： $V \rightarrow R^{+}$的无向图。 回顾上一讲的顶点覆盖问题。 我们可以将其表述为整数线性规划问题，如下所示。 对于每个顶点 v，我们有一个变量 x_v。 我们解释变量的含义如下：如果 $x_v=1$，则 v被选择包含在顶点覆盖中，否则 $x_v=0$。 通过这种解释，我们可以很容易地将最小权重顶点覆盖问题表述为以下整数线性规划问题。

$$\begin{aligned} \min \quad & \sum_{v \in V} w_v x_v \\ \text{满足条件} & \\ x_u + x_v \quad & \geq \quad 1 \qquad \text{对于所有的} e=(u, v) \in E \\ x_v \quad & \in \quad \{0,1\} \qquad \forall v \in V \end{aligned}$$

然而，解决整数线性规划问题是NP-难的。因此，我们使用线性规划(LP) 来近似整数规划问题的最优解，即 $\mathrm{OPT}(I)$。 首先，我们可以放宽约束条件 $x_v \in \{0,1\}$为 $x_v \in [0,1]$。 进一步简化为 $x_v \geq 0$，对于所有的$v \in V$。因此，顶点覆盖的线性规划形式为：

$$\begin{aligned} \min \quad & \sum_{v \in V} w_v x_v \\ \text{满足条件} & \\ x_u + x_v \quad & \geq \quad 1 \qquad \text{对于所有的} e=(u, v) \in E \\ x_v \quad & \geq \quad 0 \end{aligned}$$

现在我们使用以下算法：

通过线性规划求解顶点覆盖：
通过求解线性规划问题获得最优的分数解 x^*
令 $S=\{v \mid x_v^* \geq \frac{1}{2}\}$
输出 S

那么以下命题是正确的：

命题 1 S是一个顶点覆盖。

证明：考虑任意一条边， $e=(u,v)$。 根据 x^*的可行性， $x_u^* + x_v^* \geq 1$，因此要么 $x_u^* \geq {}^1{}_2$，要么 $x_v^* \geq \frac{1}{2}$。 因此，至少 u和 v中的一个将在 S中。 □

命题 **2** $w(S) \leq 2\mathrm{OPT}_{LP}(I)$。

证明： $\mathrm{OPT}_{LP}(I) = \sum_v w_v x_v^* \geq \frac{1}{2}\sum_{v\in S} w_v = \frac{1}{2}w(S)$ □

因此， $\mathrm{OPT}_{LP}(I) \geq \frac{\mathrm{OPT}(I)}{2}$ 对于所有实例 I。

注意：对于最小化问题： $\mathrm{OPT}_{LP}(I) \leq \mathrm{OPT}(I)$，其中 $\mathrm{OPT}_{LP}(I)$是通过LP找到的最优解；对于最大化问题， $\mathrm{OPT}_{LP}(I) \geq \mathrm{OPT}(I)$。

整数间隙

我们引入了整数间隙的概念，以展示我们可以通过使用LP最优解作为下界来获得的最佳近似保证。

定义：对于一个最小化问题Π，线性规划松弛/形式 LP的整数间隙是$\sup_{I\in\pi} \frac{\mathrm{OPT}(I)}{\mathrm{OPT}_{LP}(I)}$。

也就是说，整数间隙是Π的所有实例 I的最优整数值与最优分数值之间的最坏情况比率。 请注意，对于同一问题的不同线性规划形式可能具有不同的整数间隙。

声明1和2表明上述顶点覆盖线性规划的整数间隙最多为2。

问题：对于顶点覆盖线性规划，这个界限是否紧密?

考虑以下例子：取一个完全图，K_n，有n个顶点，每个顶点的权值为1。 很明显，我们必须选择n-1个顶点来覆盖所有的边。 因此，$\mathrm{OPT}(\ K\ n) =_n -1$。 然而，对于LP，对于每个顶点$_v$，$x_v = {}^1{}_2$是一个可行解，总权值为$\frac{n}{2}$。 因此间隙为$2 - \frac{1}{n}$，当$^n \to \infty$时趋近于2。

顶点覆盖的其他结果

1。 顶点覆盖的当前最佳近似比为$2 - \Theta(\frac{1}{\sqrt{\log n}})$ [1].

2。 开放问题：获得$2-\varepsilon$近似或证明对于任意固定的ε >0，获得$2-\varepsilon$是NP难的。 目前最佳的近似难度：除非P=NP，否则对于顶点覆盖没有1.36近似[2]。

3. 对于二分图，顶点覆盖问题可以在多项式时间内被最优解决。
这是根据所谓的K´´onig定理得出的。

4. 顶点覆盖问题允许使用多项式时间近似方案（PTAS），即对于任意固定的$\epsilon > 0$，对于平面图，它是一个（$1+\epsilon$）-近似。 这是根据Baker的一般方法得出的 [?]。

2 通过线性规划解决集合覆盖问题

集合覆盖问题的输入由一个有限集合 $U = \{1, 2, ..., n\}$和 m个 U的子集$S_1, S_2, ..., S_n$组成。 每个集合 S_j有一个非负权重 w_j，目标是找到覆盖 U所有元素的最小权重集合（即它们的并集是 U）。集合覆盖的线性规划松弛是：min

$$\sum_j w_j x_j$$

$$\begin{array}{rcll} \overset{\text{约束条件}}{\sum}_{j:i\in S_j} x_j & \geq & 1 & \forall i \in \{1, 2, ..., n\} \\ x_j & \geq & 0 & 1 \leq j \leq m \end{array}$$

它的对偶是：

$$\max \quad \sum_{i=1}^{n} y_i$$

$$\begin{array}{rcll} \overset{\text{约束条件}}{\sum}_{i\in S_j} y_i & \leq & w_j & \forall i \in \{1, 2, ..., n\} \\ y_i & \geq & 0 & \forall i \in 1, 2, ..., n \end{array}$$

我们基于这个原始/对偶线性规划提供了几种集合覆盖的算法。

2.1 确定性舍入

通过线性规划求解集合覆盖:
求解线性规划以获得最优解 x^*，其中包含分数。
令 $P = \{i \mid x_i^* \geq \frac{1}{f}\}$，其中 f是包含任何元素的集合的最大数量
输出 $\{S_j \mid j \in P\}$

请注意，上述算法，即使专门用于顶点覆盖，也与我们之前看到的算法不同。 它包括在LP的最优解中具有严格正值的所有集合。

令 x^* 为原始线性规划的最优解，y^* 为对偶问题的最优解，令 $P = \{j \mid x^*j > 0\}$。 首先，注意到通过强对偶性，$\sum_j w_j x_j^* = \sum_{i\in S_j} y_i^* = w_j$。其次，根据互补松弛条件，如果 $x^*_j > 0$，则对应的对偶约束条件是紧束的，即

定理3：算法的输出是给定实例的可行集覆盖。

证明：练习。 □

定理4：$\sum_{j\in P} w_j \leq f \sum_j w_j x^*_j = \mathrm{OPT}_{LP}$。

证明：

$$\sum_{j\in P} w_j = \sum_{j:x_j^*>0} (\omega_j) = \sum_{j:x_j^*>0} \left(\sum_{i\in S_j} y_i^*\right) = \sum_i y_i^* \left(\sum_{j:i\in S_j, x_j^*>0} 1\right) \le f\sum_i y_i^* \le f\mathrm{OPT}_{LP}(I).$$

. □

注意到第二个等式是由互补松弛条件引起的（如果 $x_j > 0$，则对应的对偶约束是紧的），倒数第二个不等式使用了 f的定义，最后一个不等式来自于弱对偶性（对偶问题的可行解是最优原始解的下界）。

因此，我们可以得出算法输出的覆盖重量最多为 $f\mathrm{OPT}_{LP}$。 我们注意到 f可以达到 n的大小，此时算法给出的界限相当弱。 事实上，我们无法构造出能够证明分析的紧密性的例子。

备注：分析关键地使用了x^*是一个最优解的事实。 另一方面，顶点覆盖算法更加健壮。

2.2 随机舍入

现在我们描述一个不依赖于f的舍入方法，得到一个近似界。

> 通过随机舍入解决集合覆盖问题：$A=\emptyset$，令x^*为线性规划的最优解。对于k=1到$2\ln n$，执行以下操作：
>
> 以概率x^*_j独立地选择每个集合S_j，如果S_j被选择，$A=A\cup\{j\}$。
>
> 输出在A中的集合的索引。

Claim 5 $Pr[i\text{在迭代中未被覆盖}] = \prod_{j:i\in S_j} (1 - x_j^*) \le \frac{1}{e}$.

直觉：我们知道 $\sum_{j:i\in S_j} x_j^* \ge 1$. 在满足这个约束条件的情况下，如果我们想要最小化概率，我们可以让 x_j^*彼此相等，那么概率$= (1 - \frac{1}{k})^k$，其中 $x_j^* = 1/k$。

证明：$Pr[i\text{在迭代中未被覆盖}] = \prod_{j:i\in S_j} (1 - x_j^*) \le \prod_{j:i\in S_j} e^{-x_j^*} \le e^{-\sum_{j:i\in S_j} x_j^*} \le \frac{1}{e}$. □

然后我们得到以下推论：

推论：$Pr[i\text{在算法结束时未被覆盖}] \le e^{-2\log n} \le \frac{1}{n^2}$.

推论：$Pr[\text{算法停止后，所有元素都被覆盖}] \ge 1 - \frac{1}{n}$. 上述结果来自于并集边界。 i未被覆盖的概率最多为 $1/n^2$，因此存在某个 i未被覆盖的概率最多为 $1/n^2 \le 1/n$。

令 C_t= 迭代 t中选择的集合的成本，则 $E[C_t] = \sum^{mj}_{=1} w_j x^*_j$，其中 $E[X]$ 表示随机变量 X的期望。 然后，令 $C = \sum^{2\ln}_{t=1} {}^{n} C_t$；我们有 $E[C] = \sum^{2\ln}_{t=1} {}^{n} E[C_t] \leq 2 \ln n \mathrm{OPT}_{LP}$。 根据马尔可夫不等式，我们知道$Pr[C > 2E[C]] \leq {}^{1}{}_{2}$，因此我们有$Pr[C \leq 4 \ln n \mathrm{OPT}_{LP}] \geq$- ${}^{1}{}_{2}$。因此，$P\,r[C \leq 4 \ln n \mathrm{OPT}_{LP}$并且所有项目都被覆盖$] \geq {}_{2}{}^{1} - {}^{1}_{n}$。因此，随机舍入算法，以接近$1/2$的概率成功给出一个可行的
成本为 $O(\log n)\mathrm{OPT}_{LP}$的解决方案。请注意，我们可以检查解决方案是否满足所需的
属性（可行性和成本），如果不满足，则重复该算法。

1. 我们可以检查舍入后的解决方案是否满足所需的属性，例如所有元素都被覆盖，或成本最多为$2c \log n \mathrm{OPT}_{LP}$。 如果不满足，重复舍入。 成功的预期迭代次数是一个常数。

2. 我们还可以使用Chernoff边界（大偏差边界）来证明单次舍入
以高概率成功（概率至少为$1 -$ $\frac{1}{\text{多项式(在)}}$）.

3. 该算法可以被去随机化. 去随机化是一种去除随机性的技术，或者尽可能少地使用随机性。有许多去随机化的技术，例如条件期望法、差异理论和扩展图。Bro*d*er等人[5]使用最小哈希独立排列来去随机化S. Rajagopalan和V. Va*z*irani[6]提出的近似集覆盖的RNC算法。

4. 经过几轮后，选择覆盖每个未覆盖元素的最便宜的集合。 这样的成本较低。 该算法确保可行性，但只能在期望意义上保证成本。

与集覆盖相关的其他结果

1. 除非$\mathrm{P} = \mathrm{NP}$，否则对于某个固定的c，不存在 $c \log n$的近似算法[4]。

2. 除非NP $\subseteq DTIME(n^{O(\log \log n)})$，否则不存在$(1 - o(1)) \ln n$-近似算法[3]。

3. 除非$\mathrm{P} = \mathrm{NP}$，否则对于任何固定的$\varepsilon > 0$，max-coverage问题没有$(1 - \frac{1}{e} + \varepsilon)$的近似解。

2.3 双拟合

在本节中，我们介绍了双拟合技术用于近似算法的分析。 在高层次上，方法如下：

1. 构造对偶线性规划的可行解。

2. 证明算法返回的解的成本可以用对偶解的值来界定。

请注意，算法本身不一定是基于线性规划的。 这里，我们以集合覆盖问题为例。
有关集合覆盖问题的原始线性规划和对偶线性规划的形式，请参考前一节。

我们可以这样解释对偶问题：将 y_i视为元素 i愿意支付的金额，对偶问题最大化总支付金额，同时满足每个集合中元素的总支付金额不超过集合的成本。加权集合覆盖的贪婪算法如下：

贪心集合覆盖:
覆盖 $= \emptyset$;
$A = \emptyset$;
当覆盖 $= U$时
  $j \leftarrow \arg\min_k(\frac{w_k}{|S_k \cap \text{未覆盖} U|})$;
  覆盖 $=$ 覆盖 $\cup S_j$;
  $A = A \cup \{j\}$.
结束循环;
将集合 A作为覆盖输出

定理6贪心集合覆盖选择的解的成本 $\leq H_d \cdot \mathrm{OPT}_{LP}$，其中 d是最大的集合大小，即 $d = \max_j |S_j|$.

为了证明这一点，我们可以稍微改进算法：

加权集合覆盖的增强贪心算法:
覆盖 $= \emptyset$;
当覆盖 $= U$时
  $j \leftarrow \arg\min_k(\frac{w_k}{|S_k \cap \text{未覆盖} U|})$;
  如果 i未覆盖且 $i \in S_j$，则设置 $p_i = \frac{w_j}{|S_j \cap \text{未覆盖}|}$;
  覆盖 $=$ 覆盖 $\cup S_j$;
  $A = A \cup \{j\}$.
结束循环;
将集合 A作为覆盖输出

很容易看出算法输出了一个集合覆盖。

命题7 $\sum_{j \in A} w_j = \sum_i p_i$.
证明：考虑当 j被添加到 A时。 令 $S'_j \subseteq S_j$为在 j被添加之前未覆盖的元素。 对于每个 $i \in S'_j$，算法设置 $p_i = w_j/|S'_j|$。因此， $\sum$ 此外，很容易看出集合 S'_j， $j \in A$是不相交的，并且共同分割了 U。因此，

$$\sum_{j \in A} w_j = \sum_{j \in A} \sum_{i \in S'_j} p_i = \sum_{i \in U} p_i.$$

□

对于每个 i，令 $y'_i = \frac{1}{H_d} p_i$.

声明8 $'$是对偶LP的可行解。

假设该声明为真，则贪婪集覆盖的解的成本 $= \sum_i p_i = H_d \sum_i y'_i \leq$ 最后一步是因为对偶问题的任何可行解都是原始LP值的下界（弱对偶性）。

现在，我们证明该声明。 设 S_j为任意集合，且 $|S_j| = t \leq d$。 设 $S_j = \{i_1, i_2, ..., i_t\}$，其中元素按照 i_1被贪婪算法覆盖的时间不晚于 i_2， i_2不晚于 i_3，依此类推。

声明**9** 对于 $1 \leq h \leq t$，有 $p_{i_h} \leq \frac{w_j}{t-h+1}$.

证明：设 $S_{j'}$ 为贪心算法中覆盖 i_h的集合。当贪心算法选择 $S_{j'}$ 时，元素 $i_h, i_{h+1}, \ldots\ldots\ldots\ldots\ldots\ldots$， i_t从 S_j中未被覆盖，因此贪心算法也可以选择 S_j。这意味着当选择 $S_{j'}$ 时，其密度不超过 $\frac{w_j}{t-h+1}$。因此 p_{i_h} 即密度不超过 $S_{j'}$的密度 $\frac{w_j}{t-h+1}$. □

根据上述论断，我们有

$$\sum_{1 \leq h \leq t} p_{i_h} \leq \sum_{1 \leq h \leq t} \frac{w_j}{t-h+1} = w_j H_t \leq w_j H_d 。$$

因此，将 y'_i设置为 p_i的密度缩小 H_d倍的值，得到一个可行解。

参考文献

[1] G. Karakostas. 顶点覆盖问题的更好近似比。 *E*CCC报告 TR04-084，2004年。

[2] I. Dinur 和 S. Safra. 偏见的重要性。第*34*届*ACM*理论计算研讨会论文集, 第*33-4*2页, 2002年.

[3] U. Feige. 近似集覆盖的ln n阈值。 *ACM*期刊, v.45 n.4, p.634 - 652, 1998年.

[4] R. Raz 和 M. Safra. 一个次常数错误概率的低次数测试，以及一个次常数错误概率的NP的PCP特征。*STOC 1997*会议论文集, 第475-484页, 1997年.

[5] A. Z. Broder, M. Charikar 和 M. Mitzenmacher. 使用最小哈希独立排列的去随机化离散算法期刊, 第1卷, 第1期, 第11-20页, 2003年.

[6] S. Rajagopalan 和 V. Vazirani. Primal-Dual RNC 近似算法用于集合覆盖和覆盖整数规划*SIAM Journal on Computing*, Volume 28, Issue 2, p.525 - 540,1999.

CS 598CSC：近似算法	讲座日期：2011年2月9日
讲师：Chandra Chekuri	记录员：Kyle Fox (2009)

在这个讲座中，我们探讨了背包问题。 这个问题为学习一些用于近似算法的重要过程提供了良好的基础，这些算法以更高的运行时间为代价提供更好的解决方案。

1 背包问题

1.1 问题描述

在背包问题中，我们给定一个背包容量 B，和一个包含 N个物品的集合。 每个物品i都有一个给定的大小$\mathsf{S}_i \geq 0$ 和一个利润$\mathsf{P}_i \geq 0$。 给定物品的一个子集 $A \subseteq N$，我们定义两个函数，$\mathsf{S}(A) = \sum_{i \in A} s_i$ 和 $p(A) = \sum_{i \in A} p_i$，表示物品组的总大小和利润。 目标是选择物品的一个子集 A，使得A的大小$(A) \leq B$且A的利润(A)最大化。 我们假设每个物品的大小最多为 B。 很容易看出，如果所有的利润都是1，自然的贪心算法将物品按大小排序，然后选择最小的物品，将得到一个最优解。 假设利润和大小都是整数，我们仍然可以相对快速地使用动态规划找到问题的最优解，时间复杂度为 $O(nB)$或$O(nP)$，其中 $P = \sum_{i=1}^{n} p_i$。 找到这些算法的细节是作业的一部分。 虽然这些算法似乎在多项式时间内运行，但应注意，假设输入中的数字不是以一元方式编写， B和 P可能是输入大小的指数级。 我们将这些称为伪多项式时间算法，因为它们的运行时间只有在输入中的数字以一元方式给出时才是多项式的。

1.2 贪婪算法

考虑下面的贪婪算法来解决背包问题，我们将其称为贪婪背包。 我们按照利润与尺寸的比率对所有物品进行排序，以便 $\frac{p_1}{s_1} \geq \frac{p_2}{s_2} \geq \cdots \geq \frac{p_n}{s_n}$。 然后，只要将物品添加到背包中不超过容量，我们就贪婪地按照这个顺序取物品。 事实证明，这个算法可能非常糟糕。假设我们只有两个物品 N。 令 $s_1 = 1$， $p_1 = 2$， $s_2 = B$， $p_2 = B$。贪婪背包只会选择物品1，但只选择物品2会得到更好的解决方案。 事实证明，我们可以通过简单地选择GreedyKnapsack的解决方案或最有利可图的物品来修改这个算法，从而提供一个2近似解。 我们将这个新算法称为修改贪婪算法。

定理1 ModifiedGreedy对于背包问题有一个近似比为 $1/2$。

证明：设 k为第一个不被GreedyKnapsack接受的物品的索引。 考虑以下声明：

声明2 $p_1 + p_2 + \ldots p_k \geq \mathrm{OPT}$。 事实上， $p_1 + p_2 + \cdots + \alpha p_k \geq \mathrm{OPT}$，其中 $\alpha = \frac{B - (s_1 + s_2 + \cdots + s_{k-1})}{s_k}$。是第 k个物品在装入第 $k-1$个物品后仍然可以放入背包的部分。

定理1的证明立即从该声明得出。 特别地， $p_1+p_2+\cdots+p_{k-1}$或 p_k必须至少为 $\mathrm{OPT}/2$。 我们现在只需要证明声明2。 我们给出一个背包问题的线性规划松弛如下：这里， $x_i \in [0,1]$表示物品 i 在背包中的比例。

$$
\begin{aligned}
\text{最大化} \quad & \sum_{i=1}^{n} p_i x_i \\
\text{受限于} \quad & \sum_{i=1}^{n} s_i x_i \le B \\
& x_i \le \text{对于所有 } i \in \{1 \ldots n \\
& \}, \; x_i \ge 0
\end{aligned}
$$

令 OPT'为线性规划实例中目标函数的最优值。
任何 KNAPSACK的解都是线性规划的可行解，并且两个问题共享相同的目标函数，因此 $\mathrm{OPT}' \ge \mathrm{OPT}$。 现在设置 $x_1 = x_2 = \cdots = x_{k-1} = 1$， $x_k = \alpha$，并且对于所有$i > k$， $x_i = 0$。 这是一个可行解，通过改变任何一个紧约束条件都无法改进，因为我们对物品进行了排序。 因此， $p_1 + p_2 + \cdots + \alpha p_k = \mathrm{OPT}' \ge \mathrm{OPT}$。 引理的第一个陈述由第二个陈述得出，因为 $\alpha \le 1$。

□

1.3 多项式时间近似方案

利用上一节的结果，我们得出了一些简单的观察。 其中一些观察导致了更好的近似。

观察**3** 如果对于所有的i，$si \le B$，贪婪背包算法给出了一个$(1-\varepsilon)$的近似。

这是根据以下推导得出的：

$$
\begin{aligned}
& \text{对于} 1 \le i \le k, \; pi/si \ge pk/sk \\
\Rightarrow\ & p_1+p_2+\ldots+p_k \ge (s_1+s_2+\ldots+s_k)p_k/s_k \Rightarrow p_k \\
& \le \; s_k(p_1+p_2+\ldots+p_k)/B \\
& \le \; \varepsilon(p_1+p_2+\ldots+p_k) \\
& \le \; \epsilon(p_1 + p_2 + \cdots + p_{k-1})/(1-\epsilon) \\
\Rightarrow\ & p_1 + p_2 + \cdots + p_{k-1} \; \ge \; (1-\epsilon)\mathrm{OPT}
\end{aligned}
$$

第三行的推导是因为 $s_1 + s_2 + \cdots + s_k > B$，并且最后一行是根据第二个命题得出的。

观察**4** 如果对于所有的 i， $p_i \le \epsilon \mathrm{OPT}$，贪心背包算法给出了 $(1-\epsilon)$的近似解。

这个结论立即可以从第二个命题得出。

观察**5** 在任何最优解中，至多有 $\lceil \frac{1}{\epsilon} \rceil$个项目的利润至少为 $\epsilon \mathbf{OP}\, T$。

我们现在可以描述以下算法。 令 $\epsilon \in (0,1)$ 为一个固定常数，令 $h = \lceil \frac{1}{\epsilon} \rceil$。 –
我们将尝试猜测最有利可图的项目，并贪婪地打包剩下的项目。

猜测 H + 贪婪算法(N, B)：
对于每个 $S \subseteq N$使得 $|S| \leq h$：
  将 S打包到大小不超过 B的背包中
  令i为S中最不赚钱的项目。 删除所有满足$pj > pi$的项目$j \in N-S$。在剩余容量$B-$
  $\sum$上运行贪心背包算法。
从上述中输出最佳解决方案。

定理**6**：猜测$\mathbf{h}$ + 贪心g给出了$(^1-\varepsilon)$的近似解，并且运行时间为$O(n^{\wedge\lceil 1/}\varepsilon+^1)$。

证明：对于运行时间，观察到有$O(n^h)$个N的子集。 对于每个子集，我们花费线性时间贪心地打包剩余的物品。 由于剩余的运行时间，最初花费在排序物品上的时间可以忽略不计。

对于近似比率，考虑循环的运行，其中S实际上是最优解中最有利的物品集合，算法的贪心阶段打包了集合$A' \subseteq (N\text{-}S)$的物品。
让OPT'是在N-S中打包较小物品的最优方式，使得$\mathrm{OPT}=p(S)+\mathrm{OPT}'$。
让物品k是被N-S的贪心打包拒绝的第一个物品。 我们知道$pk \leq \mathrm{f}^\star\mathrm{OPT}$，所以根据Claim 2，$p(A') \geq \mathrm{OPT}'\text{-}\mathrm{f}^\star\mathrm{OPT}$。 这意味着循环运行中找到的总利润为$p(S)+p(A') \geq (1\text{-}\mathrm{f})^\star\mathrm{OPT}$。

□

请注意，对于任何固定的epsilon选择，上述算法在多项式时间内运行。 这种类型的算法被称为多项式时间近似方案或PTAS。我们说一个最大化问题Π有一个PTAS，如果对于所有的$\varepsilon>0$，存在一个多项式时间算法，给出一个$(1\text{-}\varepsilon)$的近似解（对于最小化问题是$(1+\varepsilon)$）。 一般来说，通过在首先搜索一个好的基础上贪婪地填充解，可以经常找到一个PTAS来解决问题。

如下所述，背包问题实际上有一个更强的近似方案，称为完全多项式时间近似方案或FPTAS。一个最大化问题Π有一个FPTAS，如果对于所有的$\varepsilon>0$，存在一个算法，给出一个$(1\text{-}\varepsilon)$的近似解（对于最小化问题是$(1+\varepsilon)$），并且运行时间多项式地依赖于输入大小和$1/\varepsilon$。

1.4 四舍五入和缩放

之前我们提到了基于动态规划的精确算法，它们的运行时间为 $O(nB)$ 和 $O(nP)$，但我们注意到B和P可能非常大。 如果我们能够以多项式时间减少其中一个，而不会丢失太多信息，我们可能能够找到基于这些算法的近似解。 令$p\max = \max(_i p_i)$，并注意以下内容。

观察**7**：$p_{\mathrm{m}}\mathbf{a}_{\mathrm{x}} \leq \mathbf{O}\mathrm{PT} \leq \mathbf{n}p_{\mathrm{m}}\mathbf{a}_{\mathrm{x}}$

现在，固定某个 $\in (0, 1)$。 我们希望将利润进行缩放并四舍五入为整数，以便我们可以高效地使用$O(^n P)$算法，同时保留足够的数字信息以进行准确的近似。 对于每个i，令$p'i = \lfloor ^n \square \in -\frac{1}{p_{\max}} p_i \rfloor$. 观察到 $p'_i \leq {}^n{}_{\square}$，所以现在利润的总和 P'最多为 $^{n^2}$ $_{\epsilon.}$ 此外，注意到我们在舍入过程中最多损失了 n的利润，但是缩小比例的 OPT仍然至少为 $^n{}_{\square}$. 我们只损失了解决方案的一小部分ϵ. 在精确算法中，舍入和缩放值的过程在许多其他最大化问题中都有用途。 我们现在正式陈述算法ROUND&SCALE并证明其正确性和运行时间。

ROUND&SCALE(N, B):
对于每个 i集合 $p'_i = \lfloor \frac{n}{p_{\max}} p_i \rfloor$
使用运行时间为 $O(nP')$的精确算法运行 A
输出 A

定理8 ROUND&SCALE 给出了 $(1-\epsilon)$ 的近似解，并且运行时间为 $O(\frac{n^3}{\epsilon})$时间。

证明：舍入可以在线性时间内完成，且 $P' = O(\frac{n^2}{\epsilon})$，算法的动态规划部分运行时间为 $O(\frac{n^3}{\epsilon})$。为了证明近似比例，令 $\alpha = \frac{n}{p_{\max}} - \frac{1}{p_{\max}}$ 并且令 A为算法返回的解， A^*为最优解。 观察到对于所有 $X \subseteq N$， $\alpha p(X) - |X| \leq p'(X) \leq \alpha p(X)$，因为舍入最多降低每个缩放利润1。 算法在给定缩放和舍入值的情况下返回最佳选择，因此我们知道$p'(A) \geq p'(A^*)$。

$$p(A) \geq \frac{1}{\alpha} p'(A) \geq \frac{1}{\alpha} p'(A^*) \geq p(A^*) - \frac{n}{\alpha} = \mathrm{OPT} - \epsilon p_{\max} \geq (1-\epsilon)\mathrm{OPT}$$

□

需要注意的是，这不是 KNAPSACK已知的最佳FPTAS。 特别地，[1]显示了一个在 $O(n \log(1/\epsilon) + 1/\epsilon^4)$时间内运行的FPTAS。

其他问题

以下问题与Knapsack相关，它们也可以看作是先前讨论的集合覆盖问题的特例。你能看出吗?

2.1 装箱问题

装箱问题给出了 KNAPSACK中的n个物品。 每个物品 i被赋予一个大小 $s_i \in (0,1]$。目标是找到需要装下所有物品的最小容量为1的箱子数量。作为SET COVER的特例，有一个 $O(\log\ n)$的近似算法可以用于BINPACKING，但是可以看出我们可以做得更好。事实上，大多数贪心算法给出了2倍或更好的近似度。有一个算法可以保证装箱大小为$(1+\epsilon)\mathrm{OPT} + 1$。

2.2 多重背包

多重背包给我们 m个相同大小的背包 B和 n个具有大小和利润的物品就像背包问题一样。 我们再次希望尽可能地装入物品以获得最大的利润，只是现在我们有多个背包可以使用。基于最大覆盖的显然贪心算法可以得到$(1-1/e)$的近似解；你能做得更好吗?

参考文献

[1] E. L. Lawler. 快速近似算法解决背包问题。运筹学数学, 4(4): 339–356, 1979.

CS 598CSC：近似算法	讲座日期：2011年2月11日
讲师：Chandra Chekuri	记录员：CC

在上一节课中，我们讨论了背包问题。 在本节课中，我们讨论其他的装箱 和 独立集 问题。

1 最大独立集问题

一个具有许多应用的基本图优化问题是图中的最大（加权）独立集问题（MIS）。

定义1 给定一个无向图$G = (V, E)$，节点的子集$S \subseteq V$是一个独立集（稳定集）当且仅当S中的任意两个节点之间没有边。 节点的子集S是一个团，如果S中的任意两个节点在G中有一条边连接。

MIS问题是：给定一个图$G = (V, E)$，找到G中的一个最大基数的独立集。 在加权情况下，每个节点$v \in V$都有一个非负权重$w(v)$，目标是找到一个最大权重的独立集。这个问题是NP-Hard的，因此寻求近似算法是很自然的。 不幸的是，正如下面的著名定理所示，这个问题非常难以近似。

定理1（**H˚astad [1]**） 除非 $P = NP$ 否则没有 $\frac{1}{n^{1-\epsilon}}$-对于任何固定的MIS近似度$\epsilon > 0$其中 n是给定图中的节点数。

备注：最大团问题是在给定图中找到最大基数团的问题。
它与MIS问题近似等价；简单地对图进行补集操作。

该定理基本上表明：存在一类图，其中最大独立集大小要么小于 n^{δ}要么大于 $n^{1-\delta}$，并且决定给定图是否属于前一类别或后一类别是NP-Complete的。

下界结果表明应该关注特殊情况，并且已知有几个有趣的正面结果。 首先，我们考虑一个简单的贪婪算法来解决无权问题。

```
GREEDY(G):
S ← ∅
当 G 不为空时
  令 v为 G中度数最小的节点
  S ← S ∪ {v}
  从 G中移除 v及其邻居
结束循环
输出 S
```

定理2：贪心算法输出一个独立集合 S满足 $|S| \geq n/(\Delta + 1)$其中 Δ是图中任意节点的最大度数。

证明：我们对 $V \setminus S$中的节点数进行上界估计如下。 节点 u在 $V \setminus S$中是因为当贪心算法将节点 v 添加到 S时， u作为 v的邻居被移除。将u记为 v的负债。 节点 $v \in S$最多可以被记为 Δ次，因为它最多有Δ个邻居。 因此，我们有$|V \setminus S| \leq \Delta |S|$。 由于每个节点要么在$S$中，要么在 $V \setminus S$中，我们有 $|S| + |V \setminus S| = n$，因此$(\Delta + 1)|S| \geq n$，这意味着 $|S| \geq n/(\Delta + 1)$。

□

由于图中的最大独立集大小为n，我们得到以下结果。

推论**3**：贪心算法给出了一个 $\frac{1}{\Delta+1}$-近似*(unweighted)*MIS在最多Δ度的图中。

练习：证明贪心算法输出的独立集大小至少为 $\frac{n}{2(d+1)}$，其中d是G的平均度。

备注：众所周知的图兰定理通过巧妙的论证表明总是存在一个大小为n的独立集。 $\frac{n}{(d+1)}$，其中d是G的平均度。

备注：对于无权图的情况，可以获得$\Omega(\log d)$的近似比。 其中d是平均度。 令人惊讶的是，在一个被称为唯一游戏猜想的复杂性理论猜想下，已知在$O(\frac{\log d}{d \log \log d})$的因子内近似MIS是NP难的。 在最大度数为Δ的图中，当Δ足够大时。 $O(\frac{\log \Delta}{\Delta})$

练习：考虑最大度数为Δ的图上的加权MIS问题。 修改贪心算法，按照节点权重的非递增顺序排序，并证明它能够得到一个 $\frac{1}{\Delta+1}$-近似解。 在加权情况下，是否可以获得一个$\Omega(1/d)$-近似解，其中 d是平均度数?

LP松弛：可以为（加权）MIS问题制定一个简单的线性规划松弛，其中对于每个节点$v \in V$，我们有一个变量 $x(v)$表示是否选择节点 v到独立集中。我们有约束条件，对于每条边(u, v)，只能选择u或v中的一个。

$$\begin{aligned} \text{最大化} \quad & \sum_{v \in V} w(v)x(v) \\ \text{满足约束条件} \ & x(u) + x(v) \leq 1 \quad (u, v) \in E \\ & x(v) \in [0, 1] \qquad v \in V \end{aligned}$$

虽然上述是一个有效的整数规划松弛问题，当变量被限制在 $\{0, 1\}$时，对于 MIS来说，它并不是一个特别有用的表述，原因如下。

定理**4** 对于任何图，上述线性规划松弛问题的最优值至少为 $w(V)/2$。 特别地，在无权图的情况下，它至少为 $n/2$。

只需将每个 $x(v)$设置为$1/2$!

通过观察，可以得到以下更强的表述，如果 S是 G中的团，则任何独立集最多只能选择一个节点来自 S。

$$\text{最大化} \quad \sum_{v \in V} w(v)x(v)$$

$$\text{受限于} \sum_{v \in S} x(v) \leq 1 \qquad S\text{是 } G\text{中的团}$$

$$x(v) \in [0,1] \qquad v \in V$$

上述线性规划有指数多个变量，通常情况下无法在多项式时间内解决，但对于一些特殊情况，上述线性规划可以在多项式时间内解决（或近似解决），并导致精确算法或良好的近似界限。

顶点覆盖和最大独立集的近似性：以下是一个基本事实，很容易证明。

命题**5** 在任何图$G = (V, E)$中，如果S是G的一个顶点覆盖，则$V \setminus S$是G的一个独立集。 因此，$\alpha(G) + \beta(G) = |V|$，其中$\alpha(G)$是$G$的最大独立集的大小，$\beta(G)$是$G$的最小顶点覆盖的大小。

上述结果表明，如果顶点覆盖或最大独立集中的一个是NP-HaRd问题，则另一个问题也是NP-HaRd。我们已经看到，顶点覆盖问题可以得到一个2近似解，而最大独立集问题则无法得到任何常数近似解。 虽然顶点覆盖问题的2近似解对最大独立集问题没有任何有用的信息，但是它对于理解为什么$\alpha(G) + \beta(G) = |V|$很有用。 假设$S^*$是一个最优顶点覆盖集合，并且其大小$\geq |V|/2$。 那么一个2近似算法只能保证得到一个大小为$|V|$的顶点覆盖集合！因此，通过补充近似顶点覆盖集合，我们无法得到一个非平凡的独立集合。

最大独立集问题的一些特殊情况：我们在文献中提到了一些已经被考虑过的最大独立集问题的特殊情况，这并不是一个详尽无遗的列表。

- 区间图；这些是一条线上区间的相交图。 可以通过动态规划获得精确算法，并且可以通过线性规划方法解决更一般的版本。

- 注意，在图 G中，最大（权重）匹配可以看作是图 G的线图中的最大（权重）独立集，并且可以在多项式时间内精确解决。 这已经扩展到被称为无爪图的图形。

- 平面图和有界亲和图的推广，以及排除一个固定次要的图。 对于这样的图形，由于最初来自布伦达·贝克的思想，可以获得一个PTAS。

- 几何交集图。 例如，给定平面上的n个圆盘，找到不重叠的最大数量的圆盘。 可以考虑其他（凸）形状，如轴对齐矩形，线段，伪圆盘等。已知一些结果。 例如，已知平面上的圆盘有一个PTAS。当矩形加权时，平面上的轴对齐矩形有一个$\Omega(\log n)$近似解，而$\Omega(\frac{1}{\log n})$近似解。 对于无权重情况，$\Omega(\frac{1}{\log \log n})$-近似。

2个装箱整数规划（PIPs）

我们可以将背包问题表示为以下整数规划。 我们将背包容量缩放为1，不会损失一般性。

$$\begin{aligned} \text{最大化} \quad & \sum_{i=1}^{n} p_i x_i \\ \text{受限于} \quad & \sum_{i} s_i x_i \leq 1 \\ & x_i \in \{0,1\} \qquad 1 \leq i \leq n \end{aligned}$$

更一般地，如果对“物品”有多个线性约束，我们得到以下整数规划。

定义**2**装箱整数规划（PIP）是形式为 $\max\{wx \mid Ax \leq 1, x \in \{0,1\}^n\}$的整数规划，其中 w是一个 $1 \times n$非负向量， A是一个 $m \times n$矩阵，其条目在 $[0,1]$内。 如果所有条目都在 $\{0,1\}$中，则称其为 $\{0,1\}$-PIP。

在某些情况下，将问题定义为$\max\{wx \mid Ax \leq b, x \in \{0,1\}^n\}$是有用的/自然的，其中 A和 b中的条目需要是有理数/整数值。 我们可以通过将 A的每一行除以 b_i来将其转换为上述形式。

当 m A的行数（或约束条件）较小时，问题是可解的。
有时它被称为 m维背包问题（回想一下HW 1中的问题），并且可以获得任何固定常数 m的PTAS 。 然而，当 m很大时，我们观察到 MIS可以被看作是 $\{0,1\}$-PIP的一个特殊情况。 它与我们在前一节中看到的简单整数/线性规划完全对应。 因此，这个问题至少和 MIS一样难以近似。 在这里，我们通过巧妙的LP-rounding思想展示了可以将有界度数的概念推广到PIP中的列稀疏性，并获得相关的近似。 然后，我们将介绍约束条件的宽度概念，并展示它如何提供改进的界限。

定义**3**：如果矩阵A的每一列中非零元素的数量最多为k，则称其为k-列稀疏的PIP。 如果对于矩阵A中的每个元素A_{ij}，有$max_{(i,j)} A_{ij}/b_i \leq 1/W$，则称其为宽度为$W$的$PIP$。

2.1 随机舍入与修改的PIP

我们已经看到，随机舍入给出了Set Cover问题的$O(\log n)$近似算法，这是一个经典的覆盖问题。 在这里，我们将考虑随机舍入在装箱问题中的应用。 假设x是PIP的自然LP松弛问题的最优分数解，其中我们将约束条件$x \in \{0,1\}^n$替换为$x \in [0,1]^n$。 假设我们应用独立的随机舍入，其中我们以概率x_i将x_i'设置为1。 令x'为得到的整数解。 该解的期望权重正好是$\sum_i w_i x_i$是LP解的值。
然而， x'可能不满足由 $Ax \leq b$给出的约束条件。 满足约束条件的一个自然策略是以概率 cx_i将 x'_1设置为1，其中$c < 1$ 是某个缩放常数。 这可能有助于满足约束条件，因为缩放在约束条件中创建了一些空间；现在我们有了期望的解的值为 $c\sum_i w_i x_i$，缩放因子为 c的损失。仅仅通过自身的缩放并不能

允许我们声称所有约束条件在很大概率下得到满足。在这种情况下，一种非常有用的技术是修改技术；我们明智地修复/改变舍入解决方案 x'以强制它满足约束条件，通过将一些在 x'中为1的变量设置为0。关键是以一种方式来做到这一点，以便能够掌握变量被设置为1的最终概率。我们将以背包问题为例说明这一点，然后将这个思想推广到 k-稀疏PIP。我们所提出的算法来自[2]。

舍入背包问题：考虑背包问题。 方便起见，我们可以将其视为PIP的上下文。因此，我们有$ax \leq 1$，其中a_i现在表示物品$_i$的大小，背包容量为1；w_i是物品的重量。 假设x是一个分数解。 如果$a_i > 1/2$，则称物品$_i$为“大”物品；否则称为“小”物品。 令S为小物品的索引，B为大物品的索引。 考虑以下舍入算法。

舍入和修改背包算法：
令x为最优的分数解。
独立地以概率$xi/4$将每个i舍入为1。 令x'为舍入解。
x''=x'
如果（对于恰好一个大物品$_i$，x'_i=1）
  对于每个j=$_i$，设置x''j=0；否
则如果（$\sum_{i \in S} s_i x'_i > 1$或者选择了两个或更多大项目在 x'中）
  对于每个 j集合 $x''_j = 0$
结束如果
输出可行解 x''

换句话说，算法按照以下方式修改了四舍五入的解 x'。 如果在 x'中只选择了一个大项目，则算法保留该项目并拒绝所有其他小项目。 否则，如果在 x'中选择了两个或更多大项目，或者选择的所有小项目的总大小超过容量，则算法拒绝所有项目。

以下声明很容易验证。

声明**6** 整数解 x''是可行的。

现在让我们分析项目 i在最终解中存在的概率。 令 $\mathcal{E}_1$为事件 $\sum_{i \in S} a_i x'_i > 1$，即在 x'中选择的小项目的大小之和超过容量。 令 $\mathcal{E}_2$为至少选择了一个大项目在 x'中的事件。

声明 **7** $\Pr[\mathcal{E}_1] \leq 1/4.$

证明：令 $X_s = \sum_{i \in S}$为衡量所选小项大小之和的随机变量。 根据期望的线性性质，我们有

$$\mathbb{E}[X_s] = \sum_{i \in S} a_i \mathbb{E}[x'_i] = \sum_{i \in S} a_i x_i / 4 \leq 1/4.$$

根据马尔可夫不等式，$\Pr[X_s > 1] \leq \mathbb{E}[X_s]/1 \leq 1/4.$ □

声明 **8** $\Pr[\mathcal{E}_2] \leq 1/2.$

证明：由于 B中每个大项的大小至少为 $1/2$，我们有 $1 \geq \sum_{i \in B} a_i x_i \geq \sum_{i \in B} x_i/2.$
因此 $\sum_{i \in B}$事件 $\mathcal{E}_2$发生的条件是在随机选择中选择了某个项目$_{i \in B}$。
由于 i以概率 $x_i/4$被选择，根据并集界，$\Pr[\mathcal{E}_2] \leq \sum_{i \in B} x_i/4 \leq 1/2.$ □

引理**9**设 Z_i为指示随机变量，如果 $x_i''=1$，则为 1，否则为 0。 则 $\mathbb{E}[Z_i] = \Pr[Z_i = 1] \geq x_i/16.$

证明：我们考虑二进制随机变量 X_i，如果$x_i'=1$，则为。 我们有 $\mathbb{E}[X_i] = \Pr[X_i = 1] = x_i/4$。我们写

$$\Pr[Z_i = 1] = \Pr[X_i = 1] \cdot \Pr[Z_i = 1 \mid X_i = 1] = \frac{x_i}{4} \Pr[Z_i = 1 \mid X_i = 1].$$

为了下界$\Pr[Z_i=1 \mid X_i=1]$，我们上界概率$\Pr[Z_i=0|X_i=1]$，也就是拒绝 i的概率，条件是它在随机解 x'中被选择。

首先考虑一个在 x'中被选择的大物品 i。 然后，如果在 x'中选择了另一个大物品， i就会被拒绝；这个概率可以由$\Pr[\mathcal{E}_1]$上界。 如果物品 i是小的，那么只有在 $\mathcal{E}_2$发生或者选择了一个大物品（这个概率是$\Pr[\mathcal{E}_1]$）时，它才会被拒绝。无论哪种情况

$$\Pr[Z_i = 0|X_i = 1] \leq \Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2] \leq 1/4 + 1/2 = 3/4.$$

因此,

$$\Pr[Z_i = 1] = \Pr[X_i = 1] \cdot \Pr[Z_i = 1 \mid X_i = 1] = \frac{x_i}{4}(1 - \Pr[Z_i = 0 \mid X_i = 1]) \geq \frac{x_i}{16}.$$

□

可以改进上述分析以显示$\Pr[Z_i=1] \geq x_i/8.$

定理**10**随机算法输出的期望权重至少为 $\sum_{i=1}^{n} w_i x_i/16.$

证明：输出的期望权重为

$$\mathbb{E}[\sum_i w_i x_i''] = \sum_i w_i \mathbb{E}[Z_i] \geq \sum_i w_i x_i/16$$

其中我们使用了前面的引理来下界 $\mathbb{E}[Z_i].$ □

四舍五入用于 k-稀疏 **PIPs**：我们现在将上面的四舍五入算法和分析扩展到k-稀疏 **PIPs**。让 x是一个可行的分数解，最大化$\{wx \mid Ax \leq 1, x \in [0,1]^n\}$。 对于一个列索引 i，我们定义 $N(i) = \{j \mid A_{j,i} > 0\}$为 i具有非零元素的行的索引。 由于 A是 k-列稀疏的，我们有 $|N(i)| \leq k$对于 $1 \leq i \leq n$。 当我们有多个约束时，我们不能将项目/索引 i分类为大或小，因为它可能对某些约束来说很大，对其他约束来说很小。 我们说对于约束 $j \in N(\ i)$，如果 $A_{j,i} \leq 1/2$，则i对于约束 j是小的否则， i对于约束$_j$是大的。 让 $S\ j = \{i \mid j \in N(i),$且 i对于 j是小的$\}$为$_j$的所有小列的集合并且 $B\ j = \{i \mid j \in N(i),$且 i对于 j是小的$\}$为$_j$的所有大列的集合。 注意 $S_j \cap B$j是所有 i满足 $A_{j,i} > 0$的集合。

舍入与修改用于 k-稀疏 PIPs：
设 x为最优的分数解
独立地将每个 i以概率 $x_i/(4k)$ 舍入为 1。 设 x'为舍入解。
$x''=x'$
对于 $j=1$ 到 m执行以下操作
  如果 ($x'_i=1$，其中只有一个 $i \in B_j$)
    对于每个 $h \in S_j \cup B_j$且 $h=i$，设置 $x''_h=0$
  否则如果 ($\sum_{i \in S_j} A_{j,i}x'_i > 1$ 或 B_j中选择了两个或更多项在 x'中)
    对于每个 $h \in S_j \cup B_j$，设置 $x''_h=0$
  结束如果
结束循环
输出可行解 x''

在选择随机解 x'之后，算法按照以下方式修改它：对每个约束 j分别应用先前算法的策略。 因此，元素 i可以在不同的约束 $j \in N(i)$ 中被拒绝。 我们需要限制拒绝的总概率。 与之前一样，以下声明很容易验证。

声明**11**：整数解x''是可行的。

现在让我们分析物品i在最终解中出现的概率。 令$\mathcal{E}_1(j)$为事件，即j对应的物品大小之和大于容量。 $\sum_{i \in S_j} A_{j,i}x$对于$j$大于1的情况，即选择的物品大小之和超过容量。
令$\mathcal{E}_2(j)$为事件，即j对应的大物品至少选择了一个。 在x'中至少选择了一个大物品。 以下的声明与之前的推理相同，唯一的变化是缩放因子。

声明$_1$**2**：$\Pr[\mathcal{E}_1(j)] \leq {}_1/(4k)$。

声明1**3**：$\Pr[\mathcal{E}_2(j)] \leq 1/({}_2k)$。

引理1**4**：令Z_i为指示随机变量，如果$x''_i = 1$，则Z_i为1，否则为0。 则$\mathbb{E}[Z_i] = \Pr[Z_i = 1] \geq x_i/(16k)$。

证明：我们考虑二进制随机变量X_i，如果在随机舍入后，$X_i = 1$，则该变量为1。 我们有$\mathbb{E}[X_i] = \mathbf{Pr}[X_i = 1] = x_i \,/\, (4k)$。我们写作

$$\Pr[Z_i = 1] = \Pr[X_i = 1] \cdot \Pr[Z_i = 1 \mid X_i = 1] = \frac{x_i}{4k}\Pr[Z_i = 1 \mid X_i = 1]。$$

我们上界概率$\Pr[Z_i = 0 \mid X_i = 1]$，即在随机解$x'$中选择了$_i$的条件下，我们拒绝$_i$的概率。我们观察到

$$\Pr[Z_i = 0 | X_i = 1] \leq \sum_{j \in N(i)} (\Pr[\mathcal{E}_1(j)] + \Pr[\mathcal{E}_2(j)] \leq k(1/(4k) + 1/(2k)) \leq 3/4.$$

我们使用了以下事实：N（i）$\leq k$和上述声明。因此，

$$\Pr[Z_i = 1] = \Pr[X_i = 1] \cdot \Pr[Z_i = 1 \mid X_i = 1] = \frac{x_i}{4k}(1 - \Pr[Z_i = 0 \mid X_i = 1]) \geq \frac{x_i}{16k}。$$

□

通过使用上述引理和期望的线性性质来比较随机算法输出的期望权重与分数解的权重，可以得到下面的定理。

∑ 定理**15** 随机算法输出的可行解的期望权重至少为
对于k稀疏PIP，存在$1/(16k)$-近似算法。

较大的宽度有助于：我们在对背包问题的讨论中看到，如果所有物品相对于容量约束都很小，则可以获得更好的近似解。 对于PIP，我们将给定实例的宽度定义为W，如果$\max_{i,j} A_{ij}/b_i \leq 1/W$，则没有任何约束的容量超过$1/W$倍。 可以使用非常类似的算法和分析来证明近似界限改进为$\Omega(1/k^{\lceil W \rceil})$对于具有$W$的实例。 因此，如果$W = 2$，我们得到一个$\Omega(1/\sqrt{k})$近似而不是$\Omega(1/k)$-近似。更一般地，当 $W \geq c \log k/\epsilon$对于某个足够大的常数 c时，我们可以得到一个 $(1-\epsilon)$-近似。

参考文献

[1] J. Håstad. Clique 难以在$1-\epsilon$内近似。*Acta Mathematica*, 182:105–142, 1999.

[2] N. Bansal, N. Korula, V. Nagarajan, A. Srinivasan. 关于 k-Column 稀疏装箱程序。*IPCO*的处理，2010年。 可在`http://arxiv.org/abs/0908.2256`找到。

CS 598CSC：近似算法	讲座日期：2011年2月18日
讲师：Chandra Chekuri	记录员：CC

在上一节课中，我们讨论了形式为 $\max\{wx \mid Ax \leq 1, x \in \{0,1\}^n\}$ 的装箱问题其中 A是一个非负矩阵。 在本讲中，我们考虑在装箱约束条件下的“拥塞最小化”。 我们解决了一个激发这类问题的路由问题。

1 Chernoff-Hoeffding界限

在下一节的分析中，我们需要一个定理，它给出了关于偏离期望的概率的定量估计，对于一个由二进制随机变量的和组成的随机变量。

定理1（**Chernoff-Hoeffding**）设 X_1，X_2，... ，X_n是独立的二进制随机变量并且 a_1， a_2，... ， a_n是 $[0,1]$中的系数。设 $X = \sum_i a_i X_i$。然后

- 对于任意的 $\mu \geq \mathbb{E}[X]$和任意的$\delta > 0$， $\Pr[X > (1+\delta)\mu] \leq \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu$.
- 对于任意的 $\mu \leq \mathbb{E}[X]$和任意的$\delta > 0$， $\Pr[X < (1-\delta)\mu] \leq e^{-\mu\delta^2/2}$。

上述定理中的界限被称为无维度的，因为依赖关系仅仅在 $\mathbb{E}[X]$上，而不是变量的数量上。

下面的推论对我们很有用。在下面的陈述中，我们注意到 m与变量的数量 n无关。

推论2 在上述定理的条件下，存在一个通用常数 α，使得对于任意的 $\mu \geq \max\{1, \mathbb{E}[X]\}$，以及足够大的 m和 $c \geq 1$，有 $\Pr[X > \frac{\alpha c \ln m}{\ln \ln m} \cdot \mu] \leq 1/m^c$。

证明：选择 δ，使得 $1+\delta = \frac{\alpha c \ln m}{\ln \ln m}$。对于某个足够大的常数 α，我们稍后会指定。 让 m足够大，使得 $\ln \ln m - \ln \ln \ln m > (\ln \ln m)/2$。 现在应用上述定理的第一部分对于 μ和 δ的上尾部分，我们有

$$\begin{aligned}
\Pr[X > \frac{\alpha c \ln m}{\ln \ln m} \cdot \mu] &= \Pr[X > (1+\delta)\mu] \\
&\leq \left(\frac{e^\delta}{(1+\delta)^{(1+\delta)}}\right)^\mu \\
&\leq \frac{e^\delta}{(1+\delta)^{(1+\delta)}} \quad (\text{因为 } \mu \geq 1\text{，而括号内的项对于较大的 } \alpha \text{和 } m \text{小于} 1) \\
&\leq \frac{e^{(1+\delta)}}{(1+\delta)^{(1+\delta)}} \\
&= (\frac{\alpha c \ln m}{e \ln \ln m})^{-\alpha c \ln m / \ln \ln m} \\
&= \exp((\ln \alpha c/e + \ln \ln m - \ln \ln \ln m)(-\alpha c \ln m / \ln \ln m)) \\
&\leq \exp(0.5 \ln \ln m(-\alpha c \ln m / \ln \ln m)) \quad (m \text{和 } \alpha \text{足够大以确保这一点}) \\
&\leq 1/m^{c\alpha/2} \leq 1/m^c \quad (\text{假设 } \alpha \text{大于} 2)
\end{aligned}$$

□

2 拥塞最小化路由

设 $G = (V, E)$ 是一个表示可以进行路由的网络的有向图。 每条边 $e \in E$ 都有一个非负容量 $c(e)$。 有 k 对节点 $(s_1, t_1), ...$ ，(sk, tk) 并且每对 i 都与一个非负需求 di 相关联，需要在 si 和 ti 之间的单一路径上进行路由。 在第一个版本中，我们假设对于每对 i，我们明确给出了一组路径 $\mathcal{P}i$ 和需求必须在 $\mathcal{P}i$ 中的一条路径上进行路由。 给定路径的选择，例如，$p_1, p_2, ...$ ，$pi \in \mathcal{P}i$，我们在每条边 e 上有一个诱导流量。 边 e 上的流量是包含 e 的所有路径对的总需求，即 $x(e) = \sum$

我们定义拥塞为最大值$\{1, x(_e)/c(\ e)\}$. 在拥塞最小化问题中，目标是选择路径以使所有边的最大拥塞最小化。 我们将做出以下自然假设。 对于任意路径$_p\in \mathcal{P}_i$和边$e_\in\ p$, $c(e) \geq d_i$. 可以将此问题写成线性规划松弛形式如下。 我们有变量 $x_{i,p}$对于 $1 \leq_i \leq k$和 $p\in \mathcal{P}_i$，它们指示路径 p是否被选择为$_i$的路径。

$$
\begin{array}{lll}
\text{最小化 } \lambda & & \\
\text{满足以下条件} \displaystyle\sum_{p\in\mathcal{P}_i} x_{i,p} = 1 & & 1 \leq i \leq k \\
\displaystyle\sum_{i=1}^{k} d_i \sum_{p\in\mathcal{P}_i, e\in p} x_{i,p} \leq \lambda c(e) & & e \in E \\
x_{i,p} \geq 0 & & 1 \leq i \leq k, p \in \mathcal{P}_i
\end{array}
$$

从技术上讲，目标函数应该是$\max\{1, \lambda\}$，我们可以通过添加一个约束 $\lambda \geq 1$来实现。

让 λ^*是上述线性规划的最优解。 它给出了最优拥塞的下界。 我们如何将分数解转换为整数解？ Raghavan和Thompson在他们的有影响力的工作[1]中提出了一个简单的随机舍入算法。

随机舍入：
让 x是一个最优的分数解对于 $i= 1$到 k

独立于其他对，随机选择一个路径 $p\in \mathcal{P}_i$，使得$\Pr[p$被选择$] = x_{i,p}$

请注意，对于给定的一对 i，我们只选择一条路径。 可以按照以下方式实现这一步骤。 由于 $\sum_{p\in\mathcal{P}_i} x_{i,p}= 1$，我们可以以任意方式对 $\mathcal{P}_i$中的路径进行排序，并将区间[0,1]划分为长度为 $x_{i,p}$的区间，其中 $p\in \mathcal{P}_i$。 我们在[0,1]中随机选择一个数 θ，并确定所选路径所在的区间。

现在我们分析随机算法的性能。 设 $X_{e,i}$为一个二进制随机变量，如果选择的路径中包含边 e，则为1。设 $X_e = \sum$ $_i\, d_i X_{e,i}$为通过边 e传输的总需求。 我们将以下命题的证明留给读者作为练习。

命题**3** $\mathbb{E}[X_{e,i}] = \Pr[X_{e,i} = 1] = \sum_{p \in \mathcal{P}_i, e \in p} x_{i,p}$.

主要引理如下。

引理**4** 存在一个常数 β 使得 $\Pr[X_e > \beta \frac{\ln m}{\ln \ln m} \cdot c(e) \max\{1, \lambda^*\}] \leq 1/m^2$
其中 m是图中的边数。

证明：回忆一下 $X_e = \sum_i d_i X_{e,i}$.

$$\mathbb{E}[X_e] = \sum_i d_i \mathbb{E}[X_{e,i}] = \sum_i d_i \sum_{p \in \mathcal{P}_i, e \in p} x_{i,p} \leq \lambda^* c(e).$$

第二个等式是根据上面的声明得出的，不等式是由LP松弛中的约束得出的。

令 $Y_e = X_e / c(e) = \sum_i \frac{d_i}{c(e)} X_{e,i}$. 从上面 $\mathbb{E}[Y_e] \leq \lambda^*$. 变量 $X_{e,i}$是独立的，因为选择不同对的路径是独立的。Y_e是独立二进制随机变量的和，每个系数 $d_i/c_{(e)} \leq 1$（回想一下假设）。 因此，我们可以对 Y_e应用Chernoff-Hoeffding界，并特别是Corollary 2，其中$c = 2$。

$$\Pr[Y_e \geq \frac{2\alpha \ln m}{\ln \ln m} \max\{1, \lambda^*\}] \leq 1/m^2.$$

上述常数 α是Corollary 2中保证的。我们可以设置 $\beta = 2\alpha$。 通过注意到 $X_e = c(e) Y_e$，这证明了引理。 □

定理**5** 随机舍入，至少以概率 $(1 - 1/m)$（这里 m是边的数量）输出一个拥塞上界为 $O(\frac{\ln m}{\ln \ln m}) \max\{1, \lambda^*\}$.

证明：根据引理4，对于任意固定的边 e，拥塞在 e上超过的概率
$\frac{\beta \ln m}{\ln \ln m} \max\{1, \lambda^*\}$至多为$1/m^2$. 因此，对于任意边，它超过这个界限的概率至多为 $m \cdot 1/m^2 \leq 1/m$通过对 m个边进行联合边界。 因此，至少以概率$(1 - 1/m)$，所有边的拥塞上界都被限制在 $\beta \frac{\ln m}{\ln \ln m} \max\{1, \lambda^*\}$. □

上述算法可以被去随机化，但它需要悲观估计器技术
这是Raghavan [2]的另一个创新。

2.1 不可分割流问题：当路径没有明确给出时

我们现在考虑问题的变体，其中 $\mathcal{P}_i$是 s_i和t_i之间的所有路径的集合。 对于每对路径，我们没有明确给出路径，而是隐含给出。 这个问题被称为不可分割流问题。 在扩展之前的方法中，主要的技术问题是 $\mathcal{P}_i$可以是节点数 n的指数级。 我们甚至不能在多项式时间内写出之前开发的线性规划！然而，事实证明我们实际上可以隐式地解决线性规划并找到一个最优解，该解具有多项式大小的支持；换句话说，严格为正的变量 $x_{i,p}$的数量将是多项式。这应该不会让人感到惊讶，因为线性规划只有多项式数量的非平凡约束，因此它具有具有小支持的最优基本解。 一旦我们有了一个多项式大小的支持解，随机舍入算法可以通过简单地使用多项式时间来实现。

那些具有非零流量的路径。 我们如何解决线性规划？ 这需要使用椭球法在对偶问题上进行计算，然后通过互补松弛法解决原始问题。
我们将在以后的时间讨论这个问题。

另一种方法是解决一个基于流的线性规划，它具有与基于路径的规划相同的最优值。然而，为了实现随机舍入，我们需要将流量分解成路径，这在网络流中是相当标准的。 现在我们来描述基于流的松弛。 我们有变量 $f(e,i)$ 对于每条边 e和对(s_i,t_i) 来说，它是对于边 e上的对 i的总流量。我们将从 s_i发送一单位的流量到 t_i，这对应于寻找一条路径进行路由。 在计算拥塞时，我们将再次按总需求进行缩放。

$$
\begin{aligned}
& \min \lambda \\
\text{受限于} \quad & \sum_{i=1}^{k} d_i f(e,i) \le \lambda c(e) && e \in E \\
& \sum_{e \in \delta^+(s_i)} f(e,i) - \sum_{e \in \delta^-(s_i)} f(e,i) = 1 && 1 \le i \le k \\
& \sum_{e \in \delta^+(v)} f(e,i) - \sum_{e \in \delta^-(v)} f(e,i) = 0 && 1 \le i \le k, v \in \{s_i, t_i\} \\
& f(e,i) \ge 0 && 1 \le i \le k, e \in E
\end{aligned}
$$

上述线性规划可以在多项式时间内解决，因为它只有 mk个变量和 $O(m+kn)$个约束条件，其中 m是图中的边数， k是配对数。
给定一个可行解f，对于上述线性规划，我们可以将流向向量$f(.,i)$分解为最多m条路径上的流量。 然后我们在随机舍入中使用这些路径。 为此，我们需要以下关于s-t流的流分解定理。

引理**6** 给定一个有向图$G=(V,E)$和节点$s,t \in V$和一个s-t流f：$E \to R_+$，则在G中存在沿s-t路径和循环的f的分解。 更正式地说，令$\mathcal{P}st$为所有s-t路径的集合，$\mathcal{C}$为G中的有向循环的集合。 那么存在一个函数g：$\mathcal{P}st \cup \mathcal{C} \to R_+$，满足以下条件：

- 对于每个 e，$\sum$对于每个$_{q \in \mathcal{P}}$，使得$_{\cup \mathcal{C}}$，$_{e \in q}$，$g(_q) = f($e
- $\sum$对于每个 $_{p \in \mathcal{P}}$，使得 $g(p)$ 等于流 f 的值。
- 函数 g 的支持最多为 m，其中 m 是图 G 中的边数，即 $|\{q \mid g(q) > 0\}| \le m$。特别地，如果 f 是无环的，则对于所有 $q \in \mathcal{C}$，$g(q) = 0$。

此外，给定 f，满足上述属性的 g 可以在多项式时间内计算，其中输出仅包含具有非零 g 值的路径和循环。

通过应用上述要素，我们得到以下结果。

定理 **7**：在不可分割流问题中，存在一个 $O(\log n \,/\, \log\log n)$ 的随机近似算法来最小化拥塞。

参考文献

[1] P. Raghavan 和 C. D. Thompson. 随机舍入：一种可证明良好的算法和算法证明技术。Combinatorica 7(4)：365–374，1987年。

[2] P. Raghavan. 概率构造确定性算法：近似装箱整数规划。*JCSS*，37(2)：130–143，1988年。

CS 598CSC：近似算法	讲座日期：2011年2月23日和25日。讲师：Alina Ene
Alina Ene	记录员：Alina Ene

1. 无关并行机上的调度

我们有一个作业集合J和一个机器集合M。 作业i在机器j上的处理时间是pij。 让f：$J \leftarrow M$是一个将每个作业分配给恰好一个机器的函数。 f的完成时间是$\max_{1 \leq j \leq m}$。 $\sum_{i:f(i)=j} p_{ij}$，其中 $\sum_{i:f(i)=j} p_{ij}$是分配给机器 j的作业的总处理时间。 在无关并行机器调度问题中，目标是找到使得最小完成时间的作业分配方案。

我们可以为该问题编写一个与前面讲座中的路由线性规划非常相似的线性规划。 对于每个作业 i和每个机器 j，我们有一个变量 x_{ij}表示作业 i是否分配给机器 j。 我们还有一个变量 λ表示完成时间。 对于每个作业，我们有一个约束条件确保作业被分配给某个机器，对于每个机器，我们有一个约束条件确保分配给机器的作业的总处理时间不超过完成时间 λ。

$$
\begin{array}{lll}
\text{最小化} & \lambda & \\
\text{受限于} & \sum_{j \in M} x_{ij} = 1 & \forall i \in J \\
& \sum_{i \in J} x_{ij} p_{ij} \leq \lambda & \forall j \in M \\
& x_{ij} \geq 0 & \forall i \in J, j \in M
\end{array}
$$

上述LP非常自然，但不幸的是它具有无界整数间隙。假设我们有一个单一的作业，在每台机器上的处理时间为 T。 显然，最优调度的完成时间为 T。 然而，LP可以在每台机器上将作业调度到$1/m$的程度，即它可以设置 $x_{1j}= 1/m$对于所有 j，而结果分数调度的完成时间仅为T/m。

为了克服这个困难，我们稍微修改了LP。 假设我们知道最优解的完成时间等于 λ，其中 λ是一个固定的数字。 如果作业 i在机器 j上的处理时间 p_{ij}大于 λ，则作业 i不会被调度在机器 j上，并且我们可以通过将 x_{ij}设置为0或等效地删除变量来加强LP。 更准确地说，令 $\mathcal{S}_\lambda = \{(i,j) \mid i \in J, j \in M, p_{ij} \leq \lambda\}$。 给定一个值 λ，我们可以为该问题编写以下LP。

$$
\begin{array}{ll}
\underline{\mathbf{LP}(\lambda)} & \\
\sum_{j:\,(i,j) \in \mathcal{S}_\lambda} x_{ij} = 1 & \forall i \in J \\
\sum_{i:\,(i,j) \in \mathcal{S}_\lambda} x_{ij} p_{ij} \leq \lambda & \forall j \in M \\
x_{ij} \geq 0 & \text{对于所有的} (i,j) \in \mathcal{S}_\lambda
\end{array}
$$

请注意，上述LP没有目标函数。 在接下来的内容中，我们只关心LP是否可行，即是否存在满足所有约束条件的分配。 此外，我们可以将 λ视为一个参数，将 $\mathbf{LP}(\lambda)$视为一组LPs，每个参数值对应一个LP。 一个有用的观察是，如果 λ是最优调度的最小完成时间的下界，那么 $\mathbf{LP}(\lambda)$是可行的，并且它是Scheduling on UnrelatedParallel Machines问题的一个有效松弛。

引理**1**。 令 λ^*为参数 λ的最小值，使得 $\boldsymbol{LP}(\lambda)$是可行的。**.** 我们可以在多项式时间内找到 λ^*。

证明：对于任意固定的 λ值，我们可以使用多项式时间算法检查 $\mathbf{LP}(\lambda)$是否可行。因此，我们可以使用二分搜索从区间开始找到 λ^*。
$[0, \sum_{i,\ j} p_{ij}]$。 □

接下来，我们将展示如何将 $\mathbf{LP}(\lambda^*)$的解舍入，以便得到一个最大完成时间不超过$2\lambda^*$的调度。正如我们很快将看到的，将 $\mathbf{LP}(\lambda^*)$的解舍入为一个顶点解将会有所帮助。

设 x为 $\mathbf{LP}(\lambda^*)$的一个顶点解。 设 G为顶点集 $J \cup M$上的二分图，对于每个变量 $x\ ij{=}\ 0$，图 G都有一条边$_{ij}$。 如果存在某个j使得 $x_{ij} \in (0{,}1)$，则称作作业 i是分数设置的。 设 F为所有分数设置的作业集合，设 H为顶点集 $F \cup M$上的二分图，对于每个变量 $x\ ij \in (0{,}1)$，图 H都有一条边$_{ij}$；注意 H是G在顶点集 $F \cup M$上的诱导子图。 正如引理2所示，图H有一个匹配，将 F中的每个作业与一台机器匹配，我们将在舍入算法中使用这样的匹配。

引理**2.** 图 G有一个匹配，将每个作业与 F中的一个机器匹配。

我们现在准备提供舍入算法。

SUPM-舍入
找到 λ^*
找到一个顶点解 x以满足 $\mathbf{LP}(\lambda^*)$
对于每个 i和 j，使得 $x_{ij}{=}\ 1$，将作业 i分配给机器 j
构建图 H
在 H中找到一个最大匹配 $\mathcal{M}$
根据匹配 $\mathcal{M}$分配分数设置的作业

定理**3**. 考虑由 SUPM-舍入构建的分配。. 每个作业都被分配给一个机器，调度的完成时间不超过 $2\lambda^*$。

证明：根据引理2，匹配 $\mathcal{M}$将每个分数设置的作业与一个机器匹配，因此所有作业都被分配。 在分配所有整数设置的作业后，（部分调度的）完成时间不超过 λ^*。 由于 $\mathcal{M}$是一个匹配，每个机器最多接收一个额外的作业。设 i是一个分数设置的作业，并假设 i（在 $\mathcal{M}$中）与机器 j匹配。 由于对(i, j) 在 $\mathcal{S}_{\lambda^*}$中，处理时间 p_{ij}最多为 λ^*，因此分配分数设置的作业后，机器 j的总处理时间最多增加 λ。 因此，最终调度的完成时间不超过$2\lambda^*$。

□

练习：给出一个例子，证明定理3是紧的。 也就是说，给出一个实例和一个顶点解，使得调度的最大完成时间至少为$(2-(1))\lambda^*$。

由于λ^*是最优调度的最大完成时间的下界，我们得到以下推论。

推论**4**。 SUPM-*Rounding*可以达到2-近似。

现在我们将注意力转向引理2和顶点解的一些其他性质，以及$\mathrm{LP}(\lambda)$。

引理**5**。 如果$\mathbf{LP}(\lambda)$是可行的，任何顶点解最多有$m+n$个非零变量，并且至少设置了$n-m$个作业的整数值。

证明：设x是$\mathrm{LP}(\lambda)$的一个顶点解。 设r表示$\mathcal{S}\lambda$中的对数。 注意$\mathrm{LP}(\lambda)$有r个变量，每个对$(\mathrm{i}, j)\in\mathcal{S}\lambda$对应一个变量。 如果$x$是一个顶点解，它满足$\mathrm{LP}(\lambda)$的$r$个约束条件。 第一组约束条件包括$m$个约束条件，第二组约束条件包括$n$个约束条件。 因此，至少有$r-(m+n)$个紧约束条件来自第三组约束条件，即至少有$r-(m+n)$个变量被设置为零。

我们说作业i是分数设置的，如果存在某个j，使得$xij\in(0,1)$；作业i是整数设置的，如果对于所有j，$xij\in\{0,1\}$ 令I和F分别为整数设置和分数设置的作业集合
任何分数设置的作业i都被分配（分数）给至少两台机器，即存在$j\neq j'$使得$xij\in(0,1)$和$xi'j\in(0,1)$
因此，至少有$2|F|$个不同的非零变量对应于分数设置的作业。此外，对于每个整数设置的作业i，存在一个非零变量xij 因此，非零变量的数量至少为$|I|+2|F|$ 因此，$|I|+|F|=n$，并且$|I|+2|F|\leq m+n$，这给出了$|I|$至少为n - m

□

定义**1.** 一个连通图如果边的数量最多比顶点数量多一个，则称为伪树。如果一个图的每个连通分量都是伪树，则称为伪森林。

引理**6.** 图 G是一个伪森林。

证明：设 C是 G的一个连通分量。 我们将 $\mathbf{LP}(\lambda)$ 和 x限制在 C中的作业和机器上，得到 $\mathbf{LP}'(\lambda)$ 和 x'。 注意， x'是 $\mathbf{LP}'(\lambda)$ 的一个可行解。 此外， x'是 $\mathbf{LP}'(\lambda)$ 的一个顶点解。 如果不是，则 x'是 $\mathbf{LP}'(\lambda)$ 的两个可行解 x'_1和 x'_2的凸组合。 我们可以使用 x中不在 x'中的条目将 x'_1和 x'_2扩展为两个解 x_1和 x_2到 $\mathbf{LP}(\lambda)$。 根据构造， x_1和 x_2是 $\mathbf{LP}(\lambda)$ 的可行解。 此外， x是 x_1和 x_2的凸组合，这与 x是一个顶点解的事实相矛盾。 因此， x'是 $\mathbf{LP}'(\lambda)$ 的一个顶点解，并且根据引理5， x'最多有 $n'+m'$个非零变量，其中n'和 m'是 C中的作业和机器的数量。 因此， C有 $n'+m'$个顶点和最多$n'+m'$条边，因此它是一个伪树。

□

引理**2**的证明：注意每个整数集合的工作在G中的度为1。 我们从G中移除每个整数集合的工作；注意到得到的图是H。 由于我们从G中移除了相同数量的顶点和边，所以H也是一个伪森林。 现在我们按照以下方式构建一个匹配$\mathcal{M}$。

注意每个工作顶点的度至少为2，因为该工作被分配给至少两台机器。 因此，H的所有叶子节点（度为1的顶点）都是机器。 当H至少有一个叶子节点时，我们将与叶子节点相邻的边添加到匹配中，并从图中移除这两个端点。 如果H没有任何叶子节点，那么H是一个由顶点不相交的循环组成的集合，因为它是一个伪森林。 此外，每个循环的长度都是偶数，因为H是二分图。 我们为每个循环构建一个完美匹配（通过选择交替边），并将其添加到我们的匹配中。

□

练习：（在[3]中的练习17.1）使用Hall定理证明引理2。

2个广义分配问题

广义分配问题是调度无关并行机问题的一般化，其中每个作业-机器对都有相关成本，除了处理时间之外。 更准确地说，我们有一个作业集合J，一个机器集合M和一个目标λ。 作业i在机器j上的处理时间是pij，将作业i分配给机器j的成本是cij。 让f：$J \to M$是一个将每个作业分配给恰好一个机器的函数。 如果分配f的完成时间不超过λ（回忆一下λ是输入的一部分），并且它的成本是

$\sum$ 在广义分配问题中，目标是构建一个最小成本的可行分配f，前提是存在一个可行分配。

在接下来的内容中，我们将展示，如果存在成本为C且最大完成时间不超过λ的调度，则我们可以构建一个成本不超过C且最大完成时间不超过2λ的调度。

与之前一样，我们将$\mathcal{S}_\lambda$表示所有满足$pij \leq \lambda$的所有配对(ti, j)的集合。 我们可以将上一节中的松弛$\mathbf{LP}(_\lambda)$推广为以下$\mathbf{LP}$。

GAP-LP

$$
\begin{aligned}
&\min && \sum_{(\mathrm{t}i,\, j) \in \mathcal{S}_\lambda} x_{ij} c_{ij} && \\
&\text{受限于} && \sum_{j:\ (i,j) \in \mathcal{S}_\lambda} x_{ij} = 1 && \forall i \in J \\
& && \sum_{i:\ (i,j) \in \mathcal{S}_\lambda} x_{ij} p_{ij} \leq \lambda && \forall j \in M \\
& && x_{ij} \geq 0 && \text{对于所有的} (i, j) \in \mathcal{S}_\lambda
\end{aligned}
$$

由于我们还需要保留成本，我们不能再使用之前的舍入方法；事实上，很容易看出之前的舍入方法对于广义分配问题是任意糟糕的。 然而，我们仍然会寻找一个匹配，但在一个稍微不同的图中。

但在给出广义分配问题的舍入算法之前，我们先来看一下在二分图中寻找最小成本匹配的问题。 在最小成本二分图匹配问题中，我们给定一个带有边上成本c_e的二分图$B = (V_1 \cup V_2, E)$，我们希望构建一个最小成本匹配$\mathcal{M}$，如果存在这样的匹配的话，使得每个V_1中的顶点都被匹配。 对于每个顶点v，令$\delta(v)$表示与v关联的所有边的集合。 我们可以为该问题编写以下LP。

二分图匹配 (B)

$$\begin{aligned} \min \quad & \sum_{e \in E(B)} c_e y_e \\ \text{受限于} \quad & \sum_{e \in \delta(v)} y_e = 1 && \forall v \in V_1 \\ & \sum_{e \in \delta(v)} y_e \leq 1 && \forall v \in V_2 \\ & y_e \geq 0 && \forall e \in E(B) \end{aligned}$$

定理**7**（**Edmonds** [**2**]）。 对于任何二分图 B，任何顶点解$\boldsymbol{BipartiteMatching}(B)$都是整数解。**.** 此外，给定一个可行的分数解，我们可以在多项式时间内找到一个可行解 z，使得 z是整数且

$$\sum_{e \in E(B)} c_e z_e \leq \sum_{e \in E(B)} c_e y_e \circ$$

设 x是 **GAP-LP**的最优顶点解。 与之前一样，我们希望构造一个图 G，使得它有一个匹配 $\mathcal{M}$，匹配所有的作业。 图 G现在在其边上有成本，我们希望找到成本不超过 C的匹配。 回想一下，对于不相关的并行机调度我们在顶点集 $J \cup M$ 上定义了一个二分图，对于每个非零变量 x ij都有一条边ij。 我们可以为Generalized Assignment构造相同的图，并且可以为每条边$_{ij}$分配一个成本 c ij。 如果解 x实际上是一个分数匹配——也就是说，如果 x是**BipartiteMatching**(G)的可行解——定理7将给出我们所需的匹配。 解 x满足与顶点 $v \in J$相对应的约束条件，但不一定满足与顶点 $v \in M$相对应的约束条件，因为一个机器可以被分配多个作业。为了解决这个困难，我们将引入表示相同机器的多个节点，并使用 x构造结果图的分数匹配。

分数解决方案 x分配 $\sum_{i \in J} x_{ij}$ 作业到机器 j。 我们将构建二分图 G如下所示。 对于每个作业 $_i$，我们有一个节点 i。 对于每台机器 j，我们有 k_j个节点（j，1），$\cdots$，（j，k_j）。 我们可以将节点（j，1），$\cdots$，（j，k_j）看作是机器 j上的插槽。 由于现在每台机器上有多个插槽，我们需要一个分数分配 y来分配作业到机器上的插槽。 更准确地说， y对于每个作业 i和每个插槽（j，s）都有一个条目 y i，（j，s）表示分配给插槽的作业 i的分数。 我们给出了从 x构建 y的算法如下。 一旦我们有了解决方案 y，我们在任何作业 i和任何机器插槽（j，s）之间添加一条边，使得 y i，（j，s）非零。 此外，我们为 G的每条边（ i，（j，s））分配一个成本 c i，（j，s）等于 c $_{ij}$。

当我们构建 y时，我们依次考虑每台机器。 设 j为当前机器。 回想一下，我们希望确保 y将最多一个作业分配给每个槽位；因此，我们将机器 j上的每个槽位视为容量为1的箱子。 我们贪心地将作业“装入”箱子中。 我们只考虑满足 p_{ij}不超过 λ的作业 i；设 h表示这样的作业数量。 我们假设不失一般性地，这些作业标记为$1, 2, \cdots, h$，并且 $p_{1j} \geq p_{2j} \geq \cdots \geq p_{hj}$。 简单来说，当我们构建 y时，我们按照$1, 2, \cdots, h$的顺序考虑这些作业。 此外，我们跟踪未填充的箱子和该箱子上可用的空间 s。当我们考虑作业 i时，

贪心装箱 (x)

$y = 0$ ⟨⟨将y初始化为0⟩⟩
$s = 1$ ⟨⟨s是当前箱子⟩⟩
$R = 1$ ⟨⟨R是箱子s上的可用空间⟩⟩
对于 i从1到h
⟨⟨将x_{ij}装入箱子⟩⟩
如果 $x_{ij} \le R$
$y_{i,(j,s)} = x_{ij}$
$R = R - x_{ij}$
如果 $R = 0$
$s = s + 1$
$R = 1$
否则
$y_{i,(j,s)} = R$
$y_{i,(j,s+1)} = x_{ij} - R$ ⟨⟨将$x_{ij} - R$装入下一个箱子⟩⟩
$R = 1 - y_{i,(j,s+1)}$
$s = s + 1$
return y

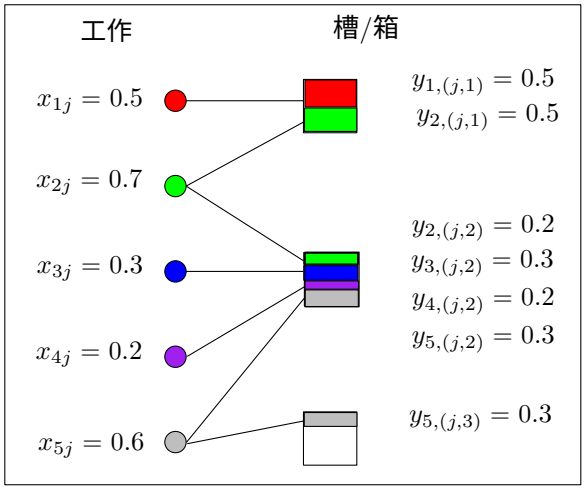

图1.从 x构建 y.

我们尝试将 x_{ij}装入当前箱子：如果至少有 x_{ij}的空间可用，即 $x_{ij} \le s$，我们将整个数量装入当前箱子；否则，我们尽可能多地将其装入当前箱子，并将剩余部分装入下一个箱子。（见图1的示例。）

引理**8**. *Greedy*Packing构造的解y是BipartiteMatching(G)的可行解。
此外，

$$\sum_{(i,(j,s))\in E(G)} y_{i,(j,s)} c_{i,(j,s)} = \sum_{(\mathrm{t}i,\, j)\in \mathcal{S}_\lambda} x_{ij} c_{ij}.$$

证明：注意，根据构造，有$\mathrm{t}x_{ij} = \sum_{s=1}^{k_{\ j}} y_{i,(j,s)}$。 因此，对于任何作业$i$，我们有

$$\sum_{(i,(j,s))\in\delta(i)} y_{i,(j,s)} = \sum_{j:\ (i,j)\in\mathcal{S}_\lambda} \sum_{s=1}^{k_j} y_{i,(j,s)} = \sum_{j:\ (i,j)\in\mathcal{S}_\lambda} x_{ij} = 1$$

此外，由于我们对每个时间段关联的箱子施加了容量为1的限制，因此对于任何时间段(j, s)，有

$$\sum_{(i,(j,s))\in\delta((j,s)))} y_{i,(j,s)} \le 1$$

因此 y是一个可行解BipartiteMatching(G)。最后，

$$\sum_{(i,(j,s))\in E(G)} = \sum_{i=1}^{n} \sum_{j:\ (i,j)\in\mathcal{S}_\lambda} \sum_{s=1}^{k_j} y_{i,(j,s)} c_{ij} = \sum_{(\mathrm{t}i,\, j)\in\mathcal{S}_\lambda} x_{ij} c_{ij}$$

□

定理7给出了以下推论。

推论**9**。　　图 G有一个匹配 $\mathcal{M}$，它匹配每个工作，并且其成本最多为 $\sum_{(i,j)\in\mathcal{S}_\lambda} x_{ij}c_{ij}$。 此外，我们可以在多项式时间内找到这样的匹配。

GAP-Rounding

令 x为 **GAP-LP**的最优解
$y = \textsc{GreedyPacking}(x)$
构建图 G
在 G中构建一个匹配 $\mathcal{M}$，使得 $\mathcal{M}$匹配每个工作
　　并且 $\mathcal{M}$的成本最多为 $\sum_{(i,j)\in\mathcal{S}_\lambda} x_{ij}c_{ij}$
对于每条边（i，（j，s））$\in$
　　$\mathcal{M}$，将作业i分配给机器j

定理**10.** 让$C = \sum$　　***GAP-Rou**nd**ing***返回的调度成本最多为C，并且完成时间最多为2_{λ}。

证明：根据推论9，调度的成本最多为C。 因此，我们只需要对调度的完成时间进行上界估计。

考虑一台机器j。 对于机器j上的任何时间段（j，s），令q

$$j\mathrm{s} = \max_{i:y_{i,(j,s)}>0} p_{ij}$$

也就是说，qj_s是任何分配给时间段（j，$_s$）的作业i的最大处理时间。 由此可知，$\mathcal{M}$分配给机器j的作业的总处理时间最多为$\sum_{s=1}^{k} j\, qj_s$。

由于**GAP-LP**只有对于满足p_i_j小于等于λ的$(i,\,_j)$对才有变量x_i_j，因此q_{j1}也小于等于λ。 因此我们只需要证明$\sum(^{k}{}_{=}2 \text{ to } _j)\; q_{js}$也小于等于$\lambda$。 考虑一个插槽$_s$，其中$_s$大于$1$。 回想一下，我们将与机器$_j$相关的作业标记为$1, 2, ..., \mathrm{h}$，其中$p_{1}j$大于等于$p_2_j$大于等于...大于等于$p_\mathrm{h}_j$。 考虑一个被分配到插槽$_s$的作业$\ell$。 由于$\mathbf{G}\mathrm{reedy}\mathbf{P}\mathrm{ac}^{k}i\mathrm{ng}$根据作业的处理时间非递增顺序考虑作业，作业$\ell$的处理时间$p\ell_j$小于等于分配到插槽$_s$-$1$的任何作业的处理时间。 因此，$p\ell_j$也受分配到插槽$_s$-$1$的作业处理时间的任何凸组合的上界限制。 由于插槽$_s$-$1$已满，

$\sum_{i}$为y_i，对于所有（$_j$，$_s$-1），$p_{\ell j}=_1$，因此$p\grave{}_j$最多为$\sum\ _i$为y_i，对于所有（$_j$，$_s$-1），p_{ij}。 由此可得

$$\sum_{s=2}^{k_j} q_{js} \le \sum_{s=2}^{k_j} \sum_{i\text{，对于所有（}j\text{，}s\text{-}1\text{），}p_{ij}\le} y_i \le \sum_{s=1}^{k_j} \sum_{i\text{，对于所有（}j\text{，}s\text{），}p_{ij}} y_i$$

根据构造，$\sum$　　$_s$为y_i，对于所有（$_j$，$_s$），$x_{ij}=$，因此

$$\sum_{s=1}^{k_j} \sum_{i\text{，对于所有（}j\text{，}s\text{），}p_{ij}=} y_i = \sum_i p_{ij} \sum_{s=1}^{s} y_{i,(j,s)} = \sum_i p_{ij}x_{ij}$$

由于x是**GAP-LP**的可行解，

$$\sum_{s=2}^{k_j} q_{js} \le \sum_i p_{ij}x_{ij} \le \lambda$$

证明完成。 □

参考文献

[1] Chandra Chekuri，近似算法讲义，第12讲，h*ttp*://www.*cs*.uiu*c*.edu/~chekuri/teaching/fall2006/lect12.pdf，2006年。

[2] Alexander Schrijver,组合优化, 第17章, Springer Verlag, 2003.

[3] Vijay Vazirani,近似算法, 第17章, Springer Verlag, 2001.

[4] David Williamson and David Shmoys,近似算法设计, 第11章, http://www.designofapproxalgs.com/, 2010.

CS 598CSC：近似算法	讲座日期：2011年3月2日
讲师：Chandra Chekuri	记录员：CC

局部搜索是一种强大且广泛使用的启发式方法（具有各种扩展）。 在这个讲座中，我们将在近似算法的背景下介绍这种技术。 局部搜索的基本概述如下。 对于给定问题的实例 I，令 $\mathcal{S}$（I）表示 I的可行解集合。 对于解 S，我们使用术语（局部）邻域 S的集合，该集合由所有解 S'组成，这些解 S'可以通过某些局部移动从 S获得。 我们将 N（S）表示为 S的邻域。

局部搜索：
找到一个“好”的初始解 $S_0 \in \mathcal{S}(I)$
$S \leftarrow S_0$
重复
  如果 $(\exists S' \in N(S)$ 使得 $\mathrm{val}(S')$ 严格优于 $\mathrm{val}(S))$
    $S \leftarrow S'$
  否则
    S是一个局部最优解
    返回 S
  结束如果
直到 (真)

对于最小化问题，如果 $\mathrm{val}(\ S') < \mathrm{val}(\ S)$ 则S'严格优于S，而对于最大化问题，如果 $\mathrm{val}(S') > \mathrm{val}(S)$ 则是这种情况。

通用局部搜索算法的运行时间取决于几个因素。 首先，我们需要一个算法，给定一个解 S，要么声明 S是一个局部最优解，要么找到一个解 $S' \in N(S)$ 使得 $\mathrm{val}(S')$ 严格优于 $\mathrm{val}(S)$。 一种标准且简单的方法是确保局部移动的定义方式使得 $|N(S)|$在输入规模 $|I|$中是多项式的，并且可以高效地枚举 $N(S)$；因此可以检查每个 $S' \in N(S)$ 看看是否有任何一个比 S更好。 然而，在一些更高级的设置中， $N(S)$ 可能是输入规模的指数级，但是可能能够在多项式时间内找到一个在 $S' \in N(S)$ 中改进 S的解。 其次，算法的运行时间还取决于从 S_0到局部最优解所需的迭代次数。 在最坏的情况下，迭代次数可能是 $|\mathrm{OPT} - \mathrm{val}(S_0)|$，这不一定是输入规模的强多项式。 我们将看到通常可以使用标准的缩放技巧来解决这个问题；基本上，我们停止算法，除非对当前的 S获得的改进是 $\mathrm{val}(S)$ 的一个显著比例。 最后，初始解 S_0的质量也影响运行时间。

1 最大割问题的局部搜索

我们演示了对于著名的最大割问题的局部搜索。 在最大割问题中，给定一个无向图 $G = (V, E)$，目标是将 V分成 $(S, V \setminus S)$ 以最大化跨越 S的边的数量，即 $|\delta_G(S)|$。 在加权版本中，每条边 e都有一个非负权重 $w(e)$，目标是最大化跨越 S的边的权重，即 $w(\delta_G(S))$，其中$w(A) = \sum_{e \in A} w(e)$。

我们考虑了一个简单的局部搜索算法来解决最大割问题，该算法从任意集合$S \subseteq V$开始，在每次迭代中，要么将一个顶点添加到 S中，要么从 S中移除一个顶点，只要这样做可以改善割的容量。

```
最大割的局部搜索：
S ← ∅
重复
  如果（∃v ∈ V \ S使得 w（δ（S + v））> w（δ（S））
      S ← S + v
  否则如果（∃v ∈ S使得 w（δ（S − v））> w（δ（S））
      S ← S − v
  否则
      S是一个局部最优解
      返回 S
  结束如果
直到 (真)
```

我们首先关注局部搜索算法输出的解的质量。

引理1 假设 S是局部搜索算法的局部最优解。 那么对于每个顶点 v，
$w（\delta（S）\cap\delta（v））\geq w（\delta（v））/2$。

证明：令 $\alpha_v = w（\delta（S）\cap\delta（v））$ 表示与 $v（\delta（v））$ 相交的边的权重，这些边跨越了割 S。令 $\beta_v = w（\delta（v））- \alpha_v$。

我们声称对于每个$_v$，$\alpha_v \geq \beta_v$。 如果$_{v \in V \setminus S}$且$\alpha_v < \beta_v$，则将$_v$移动到S将严格增加$w(\delta(S))$，并且S不能是局部最优解。 同样，如果$_{v \in S}$且$\alpha_v < \beta_v$，则$w(\delta(S-_v)) > w(\delta(S))$，并且$S$不是局部最优解。

□

推论2 如果S是局部最优解，则$w(\delta(S)) \geq w(E)/2 \geq \mathrm{OPT}/2$。

证明：由于每条边恰好与两个顶点相邻，我们有$w(\delta(S)) =_2 {}^1$。 $- \sum_{v \in V} w(\delta(S) \cap \delta(_v))$。
应用上述引理，

$$\begin{aligned} w(\delta(S)) &= \frac{1}{2}\sum_{v \in V} w(\delta(S) \cap \delta(v)) \\ &\geq \frac{1}{2}\sum_{v \in V} w(\delta(v))/2 \\ &\geq \frac{1}{2} w(E) \\ &\geq \frac{1}{2}\mathrm{OPT}, \end{aligned}$$

因为 $\mathrm{OPT} \leq w(E).$ □

局部搜索算法的运行时间取决于局部改进的次数；检查是否存在导致改进的局部移动可以通过尝试所有可能的顶点来完成。 如果图是无权图，则算法在最多 $|E|$次迭代中终止。 然而，在有权图的情况下，已知算法在权重较大时可能需要指数时间。

当权重较大时，许多局部搜索算法可以稍微修改以终止于近似局部最优解。 这样的修改算法的运行时间在输入规模上是强多项式的，并且解的质量与原始局部搜索非常相似。 我们以最大割为例说明这些思想。 考虑以下算法，其中$\epsilon > 0$ 是可以选择的参数。 设 G中的节点数为 n。

修改的局部搜索最大割(ϵ):
$S \leftarrow \{v^*\}$ 其中 $v^* = \arg\max_{v \in V} w(\delta(v))$
重复
  如果 $(\exists v \in V \setminus S$使得 $w(\delta(S+v)) > (1+\frac{\epsilon}{n}) w(\delta(S)))$
    $S \leftarrow S + v$
  否则如果 $(\exists v \in S$使得 $w(\delta(S-v)) > (1+\frac{\epsilon}{n}) w(\delta(S)))$
    $S \leftarrow S - v$
  否则
    返回 S
  结束如果
直到 (True)

上述算法终止，除非改进相对于当前解的值是$(1+\frac{\epsilon}{n})$的倍数。 因此，最终输出 S是一个近似的局部最优解。

引理**3** 设 S为修改后的局部搜索算法的输出结果MAX CUT. 那么 $w(\delta(S)) \geq \frac{1}{2(1+\epsilon/4)} w(E)$.

由于S是一个近似的局部最优解，我们声称对于每个v

$$\beta_v - \alpha_v \leq \frac{\epsilon}{n} w(\delta(S)).$$

否则，使用 v进行局部移动将会使 S的改进超过$(1+\epsilon/n)$倍。 (正式的证明留给读者作为练习)。我们有,

$$\begin{aligned} w(\delta(S)) &= \frac{1}{2}\sum_{v \in V} \alpha_v \\ &= \frac{1}{2}\sum_{v \in V} ((\alpha_v + \beta_v) - (\beta_v - \alpha_v))/2 \\ &\geq \frac{1}{4}\sum_{v \in V} (w(\delta(v)) - \frac{\epsilon}{n} w(S)) \\ &\geq \frac{1}{2} w(E) - \frac{1}{4}\sum_{v \in V} \frac{\epsilon}{n} w(S) \\ &\geq \frac{1}{2} w(E) - \frac{1}{4}\epsilon \cdot w(S). \end{aligned}$$

因此 $w(S)(1+\epsilon/4) \geq w(E)/2$，引理得证。 □

现在我们讨论算法的迭代次数。

引理4：修改后的局部搜索算法在改进步骤的 $O(\frac{1}{\epsilon}n\log n)$次迭代后终止。

证明：我们观察到$w(E)/2n$为什么？$w(\delta(v^*))/2$每次局部改进迭代都会将 $w(\delta(S))$以$(1+\epsilon/n)$的乘法因子改进。 因此，如果 k是算法运行的迭代次数，则$(1+\epsilon/n)^k w(S_0) \leq w(\delta(S_)$其中 S是最终输出。然而，$w(\delta(S)) \leq w(E)$。因此

$$(1+\epsilon/n)^k w(E)/2n \leq w(E)$$

这意味着 $k = O(\frac{1}{\epsilon}n\log n)$。 □

局部最优的一个紧密例子：局部搜索算法是否比$1/2$更好?
在这里，我们证明局部最优不会比$1/2$近似更好。 考虑一个完全的二分图 $K_{2n,2n}$每部分有$2n$个顶点。 如果 L和 R是部分集合 S其中$|S\cap L| = n = |S\cap R|$是一个局部最优解，且 $|\delta(S)| = |E|/2$。 这个实例的最优解是 $|E|$。

最大有向割：与最大割相关的问题是最大有向割，在这个问题中，给定一个有向边权重图 $G=(V, E)$，目标是找到一个集合 $S\subseteq V$，使得离开 S 的有向边的权重最大。 可以应用类似于最大割的局部搜索。 然而，下面的例子表明输出的 S 可以是任意糟糕的。设 $G=(V,E)$ 是一个以中心 v 为中心的有向星形图，连接每个 $v_1, ..., v_n$ 到 v。 那么 $S=\{v\}$ 是一个局部最优解，其中 $\delta^+(S)=\emptyset$，而 $\mathrm{OPT}=n$。 然而，对算法进行微小调整可以得到 $1/3$ 近似解！不是返回局部最优解 S，而是返回 S 和 $V\setminus S$ 中更好的一个。 这一步是必需的，因为有向割不是对称的。

2 本地搜索子模函数最大化

在本节中，我们考虑了本地搜索在最大化非负子模函数方面的效用。 设 $f: 2^V \to \mathbb{R}_+$是一个在基础集合V上的非负子模集函数。 回顾一下，如果对于所有的$A, B \subseteq V$，有 $f(A)+f(B) \geq f(A\cup B)+f(A\cap B)$，则 f是子模的。 等价地，如果对于所有的$A \subset B$和 $v \in B$，有 $f(A+v)-f(A) \geq f(B+v)-f(B)$，则 f是子模的。 如果对于所有的$A \subseteq B$，有 $f(A) \leq f(B)$，则 f是单调的。. 如果对于所有的$A \subseteq V$，有 $f(A)=f(V\setminus A)$，则 f是对称的。. 子模函数在组合优化中的许多场景中出现。 以下是两个重要的例子。

例子：集合系统中的覆盖。令S_1，S_2，...为集合$\mathcal{U}$的子集。令$V=\{1, 2, ..., n\}$，并定义$f: 2^V \to \mathbb{R}_+$，其中$f(A)=|\cup_{i\in A} S_i|$。 ，S_nt为集合$\mathcal{U}$的子集。令$V=\{1, 2, ..., n\}$，并定义$f: 2^V \to \mathbb{R}_+$，其中$f(A)=|\cup_{i\in A} S_i|$。 f是一个单调次模函数。. 也可以通过函数$w: \mathcal{U} \to \mathbb{R}_+$为$\mathcal{U}$的元素关联权重；定义的函数$f(A)=w(\cup_{i\in A} S_i)$也是单调次模的。

例子：在图中剪切函数。 设 $G=(V, E)$ 是一个非负边权重的无向图
边权重 $w: E \to \mathbb{R}_+$。 剪切函数 $f: 2^V \to \mathbb{R}_+$ 定义为 $f(S)=\sum_{e\in\delta_G(S)} w(e)$ 是一个对称的次模函数；除非图是平凡的，否则它不是单调的。如果 G 是有向的，并且我们定义 f 为 $f(S)=\sum_{e\in\delta^+_G(S)} w(e)$ 那么 f 是次模的，但不一定是对称的。

下面的问题推广了最大割和最大有向割，我们已经见过了。

问题：最大次模函数。 给定一个非负的次模集合函数 f 在一个基础集合 V 上通过一个值预言机 [1] 找到 $\mathbf{m}\mathrm{AX}_{S\subseteq V} f(S)$。

注意，如果 f 是单调的，那么问题是平凡的，因为 V 是最优解。 因此，只有当 f 不一定是单调的时候，问题才是有趣的（且 NP-Hard）。 我们考虑一个简单的局部搜索算法来解决最大次模函数，并且证明它在 f 对称时给出了 $1/3$-近似和 $1/2$-近似。这在 [2] 中已经被证明。

最大子模函数的局部搜索：
$S \leftarrow \emptyset$
重复
  如果 ($\exists v \in V \setminus S$使得 $f(S+v) > f(S)$)
    $S \leftarrow S + v$
  否则如果 ($\exists v \in S$使得 $f(S-v) > f(S)$)
    $S \leftarrow S - v$
  否则
    S是一个局部最优解返回 S
    和 $V \setminus S$中更好的一个直到 (True)

我们从一个关于子模性的基本引理开始分析算法。

引理 **5** 设 $f\colon 2^V \to \mathbb{R}_+$是 V上的子模集函数。设 $A \subset B \subseteq V$。那么

- 如果 $f(B) > f(A)$，那么存在一个元素 $v \in B \setminus A$使得 $f(A+v) - f(A) > 0$。 更一般地，存在一个元素 $v \in B \setminus A$使得 $f(A+v) - f(A) \geq \frac{1}{|B\setminus A|}(f(B) - f(A))$.
- 如果 $f(A) > f(B)$那么存在一个元素 $v \in B \setminus A$使得 $f(B-v) - f(B) > 0$. 更一般地，存在一个元素 $v \in B \setminus A$使得 $f(B-v) - f(B) \geq \frac{1}{|B\setminus A|}(f(A) - f(B))$.

我们得到以下推论。

推论**6** 设 S为局部搜索算法的局部最优解， S^*为最优解。 那么 $f(S) \geq f(S \cap S^*)$且 $f(S) \geq f(S \cup S^*)$.

定理**7**：局部搜索算法是$1/3$近似算法，并且如果f是对称的，则是$1/2$近似算法。

证明：设S为给定实例的局部最优解，S^*为全局最优解。 根据前面的推论，我们有$f(S) \geq f(S \cap S^*)$和$f(S) \geq f(S \cup S^*)$。 注意算法输出S和$V \setminus S$中更好的解。 根据次模性，我们有：

$$f(V \setminus S) + f(S \cup S^*) \geq f(S^* \setminus S) + f(V) \geq f(S^* \setminus S)$$

[1] 一个用于集合函数 f的值预言机: $2^V \to \mathbb{R}$通过给出 $f(A)$ 的值来提供对函数的访问权限，当给出集合 A时。

在第二个不等式中，我们使用了 f的非负性。 将这些不等式放在一起，

$$\begin{aligned} 2f(S) + f(V \setminus S) &= f(S) + f(S) + f(V \setminus S) \\ &\geq f(S \cap S^*) + f(S^* \setminus S) \\ &\geq f(S^*) + f(\emptyset) \\ &\geq f(S^*) = \mathrm{OPT}. \end{aligned}$$

因此，$2f(S) + f(V \setminus S) \geq \mathrm{OPT}$，因此$\max\{f(S), f(V \setminus S)\} \geq \mathrm{OPT}/3$.

如果 f是对称的，我们可以这样论证。 使用引理5，我们声称 $f(S) \geq f(S \cap S^*)$ 与之前一样但也有 $f(S) \geq f(S \cup \bar{S}^*)$ 其中$\bar{A}$ 是 $V \setminus A$的简写。 由于
f是对称的 $f(S \cup \bar{S}^*) = f(V \setminus (S \cup \bar{S}^*)) = f(\bar{S} \cap S^*) = f(S^* \setminus S)$。因此，

$$\begin{aligned} 2f(S) &\geq f(S \cap S^*) + f(S \cup \bar{S}^*) \\ &\geq f(S \cap S^*) + f(S^* \setminus S) \\ &\geq f(S^*) + f(\emptyset) \\ &\geq f(S^*) = \mathrm{OPT}。 \end{aligned}$$

因此 $f(S) \geq \mathrm{OPT}/2$. □

局部搜索算法的运行时间可能不是多项式的，但可以像我们对Max Cut做的那样修改算法，以获得一个强多项式时间算法，该算法给出一个$(1/3 - o(1))$的近似解（对称情况下为$(1/2 - o(1))$）。更多细节请参见[2]。对于带有额外约束的子模函数最大化问题，已经有很多研究工作。局部搜索在这些算法中是一个强大的工具。有关这些结果和进一步的指引，请参考参考文献。

参考文献

[1] C. Chekuri, J. Vondr´ak, and R. Zenklusen. 通过多线性松弛和争用解决方案进行子模函数最大化。*ACM STOC*会议论文集, 2001. 即将出版.

[2] U. Feige, V. Mirrokni和J. Vondr´ak. 最大化非单调子模函数。
*Proc. IEEE FOCS*的，461-471，2007年。

[3] J. Lee, V. Mirrokni, V. Nagarajan和M. Sviridenko。在拟阵和背包约束下最大化非单调子模函数。*SIAM J. on Disc. Math.*，23（4）：2053-2078，2010年。 *ACM STOC*的初步版本，323-332，2009年。

[4] S. Oveis Gharan和J. Vondr´ak。通过模拟退火进行子模最大化。*ACM-SIAM SODA*的，1098-1116，2011年。