近似算法

算法设计与分析小班课

2022 年 5 月

注意: 以下内容主要是给课本作一些注解并补充习题, 可能不能准确地代表课程内容和考核的方向, 也不能替代阅读课本、听课、完成作业题、做往年题等活动; 其中的内容没有经过课程主管老师的审核, 也可能存在错误. 不过出现的所有错误都由作者本人负责.

近似算法是处理难解的问题策略之一, 它是一个人们现在还了解地十分不清楚的领域 ($\approx$ 是一个重要的、活跃的研究话题). 在算法设计与分析课程中, 我们主要是理解一些简单的例子和想法.

1 设计与分析的策略

设计和分析近似算法的主要想法并**不是**直接求解问题, 而是给问题的解找一个上界或者下界, 去近似这个界; 上界和下界可能是从问题本身出发发现的, 也可能和问题有一定的联系, 但已经不是问题本身了. 一些常见的方法是:

(1) 贪心法. 根据人们的日常想法, 构造解的过程视为一个决策序列, 借助排序或者其他启发式方法进行近似. (课本上介绍的主要方法)
(2) 局部搜索. 和 “著名” 课程 AI 引论里的局部搜索讲的是一个意思——对于一个可行解, 看看其周遭的解是否更好, 如果更好就移动过去. 有时候这个方法可以获得近似保证.
(3) 基于线性规划的方法. 这主要是因为组合优化问题很多都可写成整数线性规划.
    (a) 松弛法. 因为线性规划问题可以多项式时间求解, 所以可以得到一个分数解, 然后改造成整数解 (LP Rounding).
    (b) 对偶法. 利用对偶问题是原问题的上界或者下界设计算法并分析. (以下不提供例子, 参阅注 3)
(4) 随机化. (待随机算法部分补充)
(5) 归约. (但是归约不一定保证近似比, 参看注 14)

另外, 并不是所有的问题都能得到常数近似的算法. 非常数近似的算法并非不好——特别是如果能够证明不能更好的话.

1.1 贪心法 这部分是两个覆盖问题, 所用到的贪心法会比课本稍微复杂一些. 以下例子中我们会顺便练习一下 NP-完全性/难的证明.

问题 1 回忆子集覆盖问题: 给定 $U = \{u_1, \dots, u_n\}$ 的子集 $S_1, \dots, S_m \subseteq U$, 集合 S_i 有最小权 w_i, 求一个子集族 $\mathcal{S} \subseteq \{S_1, \dots, S_m\}$, 使得 $\bigcup \mathcal{S} = U$, 同时最小化 $\mathcal{S}$ 中子集的权值和. 其判定版本是问是否存在方案使得权值和 $\le W$.

(1) 证明这个问题的判定版本是 NP-完全的.

(2) 设计一个近似算法并分析近似比.

解 (1) 简单题, 从顶点覆盖归约, 对 $G(V, E)$ 中每个边制造一个权值为 1 的集合 $\{v_i, v_j\}$ 即可.

(2) 想法: 能够尽可能多覆盖尚未覆盖元素的集合最有用.

算法:

(i) 初始化已经覆盖的元素集合 $R \leftarrow \varnothing$.

(ii) 选出一个集合 S_i 最小化 $\frac{w_i}{|S_i \setminus R|}$, 每个 $S_i \setminus R$ 中的元素消耗的 (均摊) 权值代价视为 $\frac{w_i}{|S_i \setminus R|}$.

(iii) $R \leftarrow R \cup S_i$, 若 $R = U$ 就结束, 否则转 (ii).

分析: 每个元素消耗的权值代价和最优解的代价相差不大.

引理 2 *设贪心算法在第 k 次迭代中选出了集合 S_ℓ, 那么每个新覆盖的元素消耗的代价不超过 $\frac{\mathsf{OPT}}{n-k+1}$. 这里 OPT 是最优解的权值和.*

证明 对于此时没有覆盖的元素 $U \setminus R$, 最优算法肯定能用不超过 OPT 的权值和来覆盖它们 (因为整个过程都只消耗了 OPT). 根据平均值原理, 剩下每个元素消耗的权值代价肯定有不超过 $\frac{\mathsf{OPT}}{|U \setminus S|} \le \frac{\mathsf{OPT}}{n-k+1}$ 的. 又因为算法选出的是平均代价最小的新元素, 所以命题成立. □

注意算法的性能就是每次选入的新元素平均代价和, 所以

$$\mathsf{Greedy} \le \mathsf{OPT}\left(\frac{1}{n} + \cdots + \frac{1}{2} + 1\right) = O(\log n) \cdot \mathsf{OPT}.$$

近似比是 $\log n$. □

注 3 (1) 可以证明 [Sla96], 除非 P = NP, 以上近似算法是几乎最优的.

(2) 这个算法的设计、近似比和分析方法还可以从线性规划的对偶性中想出来, 这是一个系统的方法 (c.f. 前述方法 (3b)), 可参考 [Vaz01, 第 13 章].

(3) 用 LP Rounding 的方法也可以得到另一个近似算法, 方法类似于例题 13 的 (2). ♤

问题 4 带度量的 k-聚类问题: 度量 $d(\cdot, \cdot)$ (满足非负性、对称性和三角不等式) 下有 n 个点 $p_1, \dots, p_n$, 希望找到 k 个中心点 $c_1, \dots, c_k$ 来尽可能最小化:

$$r = \max_{p_i} \min_{c_j} d(p_i, c_j).$$

其判定版本是问是否存在方案使得 $r \le r_0$.
(1) 证明这个问题的判定版本是 NP-完全的.
(2) 设计一个近似算法并分析近似比.

解 (1) 选点 + 覆盖点 $\Longrightarrow$ 支配集. 给定 $G(V, E)$ 和支配集参数 K, 我们构造 V 上的距离函数如下:

$$d(v_i, v_j) = \begin{cases} 0 & (i = j), \\ 1 & (\{v_i, v_j\} \in E), \\ 114514 & (\{v_i, v_j\} \notin E). \end{cases}$$

显然这个距离函数是满足三角不等式的. 这时我们询问是否存在 $r \le 1$ 的 K-聚类即可, 不难看出答案就对应支配集, 这就完成了归约.

(2) 想法: 对于距离目前的所有中心的最近距离最远的点 (最影响算法性能的点), 赶紧让它成为中心. 算法:

(i) 任取一个顶点 $c_1 \in \{p_1, \dots, p_n\}$, 计算结果 $\mathscr{C} \leftarrow \{c_1\}$.
(ii) 找到一个点 $p \in \operatorname{argmax}_{p_i} \min_{c \in \mathscr{C}} d(p_i, c)$, 加入中心 $\mathscr{C} \leftarrow \mathscr{C} \cup \{p\}$.
(iii) 重复 (ii) 直到 $|\mathscr{C}| = k$ 时算法停止.

分析: 这是一个 2-近似算法. 设最优解的距离为 OPT, 贪心算法得到的解为 r. 假设 $r > 2\text{OPT}$, 那么有一个点 x 到全部中心点的距离都超过 2OPT.

我们断言这表明所有中心两两的距离都超过 2OPT. 首先, 被选择的中心之间的距离随着算法的进展是单调下降的. 其次, x 没有被选为中心点 (否则距离为 0), 这说明在选择最后一个中心点 c_k 的时候, c_k 到剩余中心点的距离比 r 的还大. 这就证明了断言. 于是全部的中心点和 x 合起来得到 $k + 1$ 个两两距离超过 2OPT 的点. 根据抽屉原理, 最优解中这些点至少有一对被分配到同一个中心点, 而且距离不超过 OPT, 根据三角不等式便导出了矛盾. □

1.2 局部搜索

局部搜索的例子是最大割.

问题 5 在 NP-完全性的讨论中我们已经知道最大割是一个困难的问题. 这里我们考虑带权的最大割 (每条边 e 有权值 $w(e)$), 请设计一个近似算法并分析近似比.

解 想法: 对于任何一个割 $(X, \overline{X})$, 如果一个顶点从 X (resp. $\overline{X}$) 转到 $\overline{X}$ (resp. X) 会导致割的权值增加, 我们就这样调整, 直到陷入局部最优.

分析: 首先算法必然能结束, 因为每次调整必然导致割的权值增加, 但增加是有上界的. 设最优解的值是 OPT, 图中顶点的全体是 V, 则我们有

$$\text{LocalSearch} = \frac{1}{2} \sum_{v \in V} \sum_{e\ \text{是割中边而且与}\ v\ \text{邻接}} w(e)$$

$$
\begin{aligned}
&\geq \frac{1}{2} \sum_{\substack{v \in V \\ e\text{ 是和 }v\text{ 邻接的边}}} \frac{1}{2} w(e) && (\text{不然的话 } v \text{ 可以换边}) \\
&= 2\frac{\sum_{e\in E} w(e)}{4} && (\text{握手定理}) \\
&\geq \frac{1}{2}\mathsf{OPT}. && (\text{因为 OPT 至多割掉所有的边})
\end{aligned}
$$

因此近似比是 2. □

注 6 最大割问题有一个达到 0.878OPT 的近似算法, 用的是类似于 LP + 松弛的方法 (但是 LP $\longrightarrow$ 半定规划, 总之都是凸优化的一种, 相对易解). 但如果不承认额外的复杂性假设的话, 目前只知道最好的算法性能大概在 [0.878OPT, 0.941OPT) 之间——但如果承认, 则答案就是 0.878OPT. 对此有一个非常有意思的科普视频 https://www.youtube.com/watch?v=3sZ0a17Uqms. ♤

1.3 LP 方法

线性规划方法的好处是比较通用.

在线性规划部分中, 我们知道许多组合优化问题都可以写成整数线性规划 (ILP) 问题. 另一方面, 如果去掉整数的要求 (松弛), 线性规划问题可以多项式时间求解. 通过以合适的方法把分数解变成整数 (LP Rounding), 可以得到原问题的近似解.

例 7 (顶点覆盖的近似算法) 对图 $G(V,E)$ 的 (带权) 顶点覆盖问题, 顶点 v_i 的权为 $w_i \geq 0$. 我们用变量 $\boldsymbol{x} = (x_i)_{1\leq i\leq |V|}$ 表示顶点的选择, 那么问题可以写成:

$$
\begin{aligned}
\min_{\boldsymbol{x}} \quad & \sum_i w_i x_i, \\
\text{s.t.} \quad & x_i + x_j \geq 1 \quad \forall \{v_i, v_j\} \in E, \\
& x_i \in \{0,1\} \quad \forall v_i \in V.
\end{aligned}
$$

去掉最后一行 x_i 为整数的要求, 先求解得到分数解 $\boldsymbol{x}^* \in \mathbb{R}^{|V|}$. 对于每个分量 x_i^*, 若 $x_i^* \geq \frac{1}{2}$, 则我们将其改成 1, 否则改成 0. 这样得到近似解 $\tilde{\boldsymbol{x}}$. 换言之, 我们选择顶点集合 $S = \{v_i \in V : x_i^* \geq 1/2\}$.

首先这一定是一个顶点覆盖, 因为 $x_i^* + x_j^* \geq 1$, 所以必有一个 $\geq \frac{1}{2}$, 这个顶点会被算法选入. 注意我们得到的顶点覆盖的代价最多是 LP 松弛解的两倍, 而松弛解是实际最优解的下界, 所以近似比是 2. 具体地说,

$$
\begin{aligned}
\mathsf{OPT} &\geq \sum_i w_i x_i^* && (\text{LP} \leq \text{ILP}) \\
&\geq \sum_{v_i \in S} w_i (\frac{1}{2} x_i) && \left(\text{因为 } x_i = 1 \text{ 当且仅当 } x_i^* \geq \frac{1}{2}\right) \\
&= \frac{1}{2}\mathsf{LPRounding}.
\end{aligned}
$$
◇

以上是最简单的例子, 在不同问题中, LP Rounding 的方法不同.

2 近似算法的补充例题

在本课程中, 主要还是用贪心法.

问题 8 装箱: 有 n 个重量分别为 $w_1, \dots, w_n \in \mathbb{Z}_{\geq 1}$ 的物品要装入一系列载重量为 W 的卡车中, 想要使用尽量少的卡车. 根据课本中的证明我们知道这个问题是困难的, 请设计一个常数近似算法并分析近似比.

解 算法: 直接按 $w_1, w_2, \dots$ 的顺序装车, 一旦装不下了, 就用新的一辆卡车, 装完为止.

分析: 关键想法是, 对于两辆前后被装满的车, 里面的货物重量一定超过了 W. 设想我们在有必要时多用一辆空车, 使得最终用掉的卡车数目是偶数 $2m$. 那么 $mW < \sum_i w_i$. 另一方面, 任何的解都至少需要 $\frac{\sum_i w_i}{W}$ 辆车, 因此

$$\text{OPT} \geq \frac{\sum_i w_i}{W} > m = \frac{1}{2}\text{Greedy},$$

即近似比是 2. 注意这个问题排序是没有用的, 我们只需要把每个物品的重量设为 $\frac{W}{2} + \varepsilon$ 即可, 同时这还是 2-近似比的紧实例. □

问题 9 无度量的聚类问题变种: 给一个集合 $P = \{p_1, \dots, p_n\}$, 两两之间有距离 $d(\cdot, \cdot)$, 此"距离" 只满足非负性和对称性. 再给定实数 $\Delta \geq 0$, 希望找出一个尽可能小的**代表集** $P' \subseteq P$, 使得对任何 $p \in P$ 都存在 $p' \in P'$ 使得 $d(p, p') \leq \Delta$. 问题的判定版本是问是否存在大小不超过 k 的代表集.

(1) 证明这个问题的判定版本是 NP-完全的.

(2) 设计一个 $\log n$-近似算法并分析近似比.

解 这问题其实就是子集覆盖——对每个 p_i, 定义 $S_i = \{p_j : \Delta(p_i, p_j) \leq \Delta\}$. 因此剩下的工作重复子集覆盖的设计和分析就可以. □

问题 10 给定正整数的集合 $S = \{a_1, \dots, a_n\}$ 和正整数 B, 考虑如下优化问题:

$$\max_{S' \subseteq S} \sum_{a \in S'} a \quad \text{s.t.} \quad \sum_{a \in S'} a \leq B.$$

问题的判定版本是问是否存在 $\sum_{a \in S'} \geq B_0$ 的解.

(1) 证明这个问题的判定版本是 NP-完全的.

(2) 设计一个常数近似算法并分析近似比.

解 这是背包问题的一个简化版, 可以模仿处理.

(1) 从子集和归约, 取 $B_0 = B$ 即可.

(2) 我们依次考察 $a_1, \ldots, a_n$, 令 $j^* = \max_{1\le j\le n}\left\{\sum_{i=1}^{j} a_j \le B\right\} + 1$, 并返回集合 $\{a_1, \ldots, a_{j^*-1}\}$ 和 $\{a_{j^*}\}$ 中求和大的那一个. 根据 j^* 的构造我们知道 $\max\{a_1 + \cdots + a_{j^*-1}, a_{j^*}\} \ge \frac{B}{2}$, 而 B 是最优解的上界, 所以近似比是 2. (也可使用背包问题的 FPTAS.)□

问题 11 设 V 为图 G 的顶点集, 可以证明 [Hås96], 以不超过 $|V|^{1-\varepsilon}$ 的近似比求解最大独立集问题是不太可能的. 考虑如下简单贪心算法: 每次都选择图中度最小的顶点加到解中, 然后删除这个点及其邻居, 重复直到图中没有顶点为止. 证明这是一个 $(\Delta+1)$-近似算法, 这里 Δ 是图中顶点的最大度.

证明 一方面, 最优解 OPT $\le |V|$. 另一方面, 若把近似解记为 S, 那么最多删掉了 $\Delta|S|$ 个点, 于是

$$\text{OPT} \le |V| \le (1+\Delta)|S|,$$

即得到 $(\Delta+1)$-近似. □

注 12 类似问题: 考虑在一个 $n \times n$ 上的网格图上求最大独立集, 假设每个顶点都有一个正整数作为点权而且互不相同, 请设计一个 4-近似算法计算出一个最大权独立集. ♤

问题 13 考虑 HittingSet 问题的优化版本: 给定 $A = \{a_1, \ldots, a_n\}$, a_i 有权值 w_i. 再设子集族 $B_1, \ldots, B_m \subseteq A$, 求权值和最小的集合 $H \subseteq A$, 使得对每个 $1 \le i \le m$ 都有 $H \cap B_i \ne \varnothing$.
(1) 设计一个 $\log n$-近似算法并分析近似比.
(2) 令 $b = \max_i |B_i|$, 设计一个 b-近似算法并分析近似比.

解 (1) HittingSet 和子集覆盖其实是互为对偶的问题, 这里我们建立二者可行解的一一对应. 对于 A 和子集族 $\mathscr{B} = \{B_1, \ldots, B_m\}$, 对每个 $x \in A$ 构造集合 $S_x = \{B_i \in \mathscr{B} : x \in B_i\}$, 其权值和 x 一样. 然后考虑子集覆盖问题 $(\mathscr{B}, \{S_x\}_{x\in A})$. 此时, 我们把 hitting set H 转换为 $C = \{S_x : x \in H\}$, 则 C 是 $\mathscr{B}$ 的相同大小的子集覆盖, 反之亦然. 所以可以用求解子集覆盖的 $\log n$-近似比算法来求解 HittingSet, 而 (2) 中的方法也可以得到子集覆盖的新近似算法.

(2) 使用 LP Rounding 的方法, 用 x_i 表示是否选择 a_i, 则线性规划为

$$\begin{aligned}
\min_{\boldsymbol{x}} \quad & \sum_i w_i x_i, \\
\text{s.t.} \quad & \sum_{i : a_i \in B_j} x_i \ge 1 \quad \forall B_j \in \mathscr{B}, \\
& x_i \in \{0,1\} \quad \forall a_i \in A.
\end{aligned}$$

先求解得到分数解 $\boldsymbol{x}^* \in \mathbb{R}^{|A|}$. 如何进行 rounding? 如果将 x_i^* 理解为选中频率的话, 那么 $1/b$ 是一个形象的阈值, 当然从 rounding 之后还要满足约束也可以想到这一点. 故可以取 $H = \{a_i : x_i^* \ge 1/b\}$. 这样, 因为 x_i^* 满足第一条约束, 而每个集合至多有 b 个元素, 所以对每个 B_j, 一定存在某个 $x_i^* \ge 1/b$, 得到的近似解一定是 hitting set. 现在仿照顶点覆盖的

方法, 我们得到的顶点覆盖的代价最多是 LP 松弛解的 b 倍, 而松弛解是实际最优解的下界, 所以近似比是 b. 具体地说,

$$\begin{aligned}
\mathsf{OPT} &\geq \sum_i w_i x_i^* && (\text{LP} \leq \text{ILP}) \\
&\geq \sum_{a_i \in H} w_i \left(\frac{1}{b} x_i\right) && \left(\text{因为 } x_i = 1 \text{ 当且仅当 } x_i^* \geq \frac{1}{b}\right) \\
&= \frac{1}{b} \cdot \mathsf{LPRounding}.
\end{aligned}$$

□

注 14 上题的 (1) 是基于归约构造近似算法. 但并不是说能够归约就能保证近似比. 比如说我们可以把独立集归约到顶点覆盖, 因为独立集的补是顶点覆盖. 但是顶点覆盖的 2-近似算法不能拿来求解独立集问题——假设某个最小顶点覆盖是 $|V|/2$ 大小, 而近似算法得到了 $|V|$, 那么取补将得到 0 个顶点的独立集, 这是平凡的. ♤

问题 15 分快慢的多机调度问题: 假设你有 m 台一模一样的服务器和 k 台一模一样的树莓派. 有 n 个任务, 每个任务 i 在服务器上需要处理 $\frac{1}{2}t_i$ 分钟, 而在树莓派上需要处理 t_i 分钟 (每个任务只能选一台机器处理, 不能中途换上换下). 请设计一个常数近似算法来调度这些任务, 使得最后全部完成的时间最短.

解 让我们来假想所有的机器都是树莓派, 然后运行课本上的带排序的多机调度算法, 我们得到一个 (相对于树莓派的) $\frac{3}{2}$-近似. 但是最优解可能比较好地运用了服务器, 但最优解最多缩小两倍, 所以算法是一个 3-近似. (可以试试用不排序的那个算法来做这个问题, 最后仍然会得到 3-近似的结果.) □

问题 16 子集划分: 设有正整数集合 $W = \{w_1, \dots, w_n\}$, 希望将它们划分为 m 个集合 $W_1, \dots, W_m$, 使得

$$S = \min_{1 \leq j \leq m} \sum_{w_i \in W_j} w_i$$

尽可能大. 假设 W 中没有特别大的元素, $\max_i w_i \leq \frac{\sum_i w_i}{2m}$, 设计一个常数近似算法并分析近似比.

解 按顺序看 W 中元素, 把每个元素添加到当前求和最小的集合中 (如果有多个最小, 就随便选择一个).

分析: 一方面, 最优解满足 $\mathsf{OPT} \leq \frac{\sum_i w_i}{m}$. 另一方面, 注意我们总是尽量均匀地分布元素, 可以断言最后 $S \geq \frac{\sum_i w_i}{2m}$. 如果不然, 不妨设 W_1 是贪心解中元素求和最小的集合, 其和严格地小于 $\frac{\sum_i w_i}{2m}$, 由于 W 中的元素都不太大, 我们根据算法的选择证明, 如果是这样所有的元素和起来会太小. 用 V_i 表示算法结束之后 W_i 中所有元素的求和.

事实上, 任意考虑一个集合 W_i $(j \neq i)$, 假设最后一次往其中添加元素是添加了元素

w_ℓ, 那么此时 j 的求和 V_j' 是不小于 i 的求和 V_i' 的, 这说明

$$V_i = V_i' + w_\ell \le V_j' + w_\ell \le V_j + w_\ell \le V_j + \frac{\sum_i w_i}{2m}.$$

于是,

$$\sum_k V_k \le V_j + (m-1)\left(V_j + \frac{\sum_i w_i}{2m}\right) < m \times \frac{\sum_i w_i}{2m} + \frac{m-1}{2m}\frac{\sum_i w_i}{2m} < \sum_i w_i,$$

矛盾. 因此 $S \ge \frac{\sum_i w_i}{2m}$, 这说明上面的算法是一个 2-近似. □

问题 17 在上面的子集划分问题中取 $m = 2$, 划分为两个集合 A, B, 要求最小化

$$S = \max\left\{\sum_{w_i \in A} w_i, \sum_{w_i \in B} w_i\right\}.$$

但不对元素的权值作任何假设, 设计尽可能好的近似算法.

解 无妨设 $w_1 \ge \cdots \ge w_n$, 按照这个顺序处理, 每次都把当前考虑的数 w_i 分配给此时集合中总和小的那个.

设 A 是总和较大的那个集合, 最后加入 A 的数是 w_i. 如果 $|A| = 1$, 则必有 $i = 1$, 否则 A 中至少有 2 个数. 因为 $\mathsf{OPT} \ge \max_i w_i$, 所以此时算法一定得到最优解. 如果不是, 则 $i \ge 3$, 由输入顺序知道 $w_i \le w_3$, 所以 $\mathsf{OPT} \ge 2w_i$. 另一方面, 因为因为 w_i 被算法分配给 A 时, A 中数之和更小, 而分配结束之后 A 中数之和就更大, 所以分配到 w_i 结束后的瞬间两个集合求和都不超过 $\frac{1}{2}(w_1 + \cdots + w_{i-1}) + w_i$, 从而

$$\mathsf{Greedy} \le \frac{1}{2}\left(\sum_k w_k - w_i\right) + w_i \le \mathsf{OPT} + \frac{1}{2} \times \frac{1}{2}\mathsf{OPT}.$$

第二个不等式是因为 $\mathsf{OPT} \ge \max\left\{2w_i, \frac{1}{2}\sum_i w_i\right\}$. 因此近似比是 1.25.

实际上, 可以用背包问题的 FPTAS 算法证明该问题是完全可近似的. 假设每件物品的价值为 $v_i = w_i$, 重量是 w_i, 背包总重不能超过 $\left\lfloor \frac{\sum_i w_i}{2} \right\rfloor$, 则背包问题要求出 $\max_{A \subseteq W, \sum_{w_i \in A} w_i \le \left\lfloor \frac{\sum_i w_i}{2} \right\rfloor} \sum_{w_i \in A} w_i$, 这刚好使得 $\sum_{w_i \in B} w_i = S$, 且让 S 尽可能小, 和题给问题完全等价. 所以可以用 $O(n^3/\varepsilon_0)$ 时间求出原问题的 $(1+\varepsilon)$-近似, 其中

$$\varepsilon = \frac{\sum_i w_i - \sum_{w_i \in A} w_i}{\sum_i w_i - \sum_{w_i \in A^*} w_i} - 1 = \frac{1 + \varepsilon_0 - 1}{\frac{\sum_i w_i}{\sum_{w_i \in A} w_i} - 1 - \varepsilon_0} \le \frac{\varepsilon_0}{1 - \varepsilon_0} \le \varepsilon_0,$$

式中 A^* 为背包问题的最优解. 第一个等号是依据背包问题的近似比, 第二个不等号来源

于 $\sum_{w_i \in A} w_i \le \left\lfloor \frac{\sum_i w_i}{2} \right\rfloor$. (思考前一个问题是否可以用 $m = 2$ 时的办法来改进近似比或者去掉对权值的假设.) □

问题 18 三维匹配问题: 给定元素个数相同的有限集 X, Y, Z 和 $T = X \times Y \times Z$. 称集合 $M \subseteq T$ 是一个三维的匹配, 如果每个 $X \cup Y \cup Z$ 中的元素至多在 T 的元组中出现一次. 设计一个常数近似的算法来获得尽可能大的 M. (之前我们已经看到过, 这个问题的判定版本是 NP 完全的 [KT06, §8.6 节].)

解 按任意的顺序看 T 中元素, 如果该元组和当前得到的匹配无冲突, 就放到匹配中, 否则丢弃, 看完为止.

分析: 设最优解是 M^*, 我们通过以下对应算法来分析 M^* 和算法得到的 M 的大小关系: 计数器初始化为 0, 然后逐个考察 M^* 中选入的元组 (x, y, z), 如果它和 M 中某个元组 (x', y', z') 有分量的交集, 计数器就 $+1$, 同时把 M^* 中和 (x', y', z') 有分量交叉的元组去掉. 一方面, 每次考察的时候都会 $+1$, 否则的话可以把 (x, y, z) 加到 M 中, 这和贪心算法的过程矛盾. 另一方面, 每次扔掉的元素个数不会超过 3 个——因为 M^* 是匹配. 综上所述, 最后计数器的值至少是 $|M^*|/3$, 这说明贪心算法是一个 3-近似算法. 这可以推广到 k-维匹配得到 k-近似. □

问题 19 考虑背包问题的另一个 "近似" 版本. 假设物品的重量和价值分别为 $w_1, \ldots, w_n$ 和 $v_1, \ldots, v_n$, 背包的重量限制是 W. 给定 V, 条件保证存在一个子集 $O \subseteq [n]$ 使得 $\sum_{i \in O} w_i \le W, \sum_{i \in O} v_i = V$ (但算法不知道 O 具体为何). 请设计一个多项式时间的算法算出一个物品集合 $A \subseteq [n]$ 使得 $\sum_{i \in A} w_i \le (1 + \varepsilon) W, \sum_{i \in A} v_i \ge V$.

解 为了利用动态规划算法, 我们以 $\frac{\varepsilon W}{n}$ 为精度离散化物品的重量——把所有的 w_i 向下舍入到距离其最近的 $\frac{\varepsilon W}{n}$ 的整数倍, 然后把它们和 W 都乘以 $\frac{n}{\varepsilon W}$. 这时可以用动态规划算法在多项式时间 ($O(n^3/\varepsilon)$) 求出背包问题的近似解——根据条件它的价值一定不低于 V. 另一方面, 这个近似解的每个物品可能比原来设想的重 $\frac{\varepsilon W}{n}$, 所以最多超出 εW 的限制, 满足题目要求. □

问题 20 (**) 你的网站有 n 位客户 (集合 U) 需要经常访问. 服务器可以架设在 m 个可能的地点 (集合 V), 放在地点 s 需要费用 $f_s \ge 0$. 用户 u 访问地点 s 的服务器的成本是 $d_{us} \ge 0$, 每位用户会选择和它最近 (费用最小) 的服务器. 我们希望选择一些地点 $S \subseteq V$, 尽可能最大化社会福利, 就是使

$$\sum_{s \in S} f(s) + \sum_{u} \min_{s \in S} d_{us}$$

尽可能小. 问题的判定版本是问是否存在总费用不超过 k 的布置服务器的方案.

(1) 证明这个问题的判定版本是 NP-完全的.

(2) 利用子集覆盖的近似算法的想法, 设计一个 $\log n$-近似算法并分析近似比.

解 (1) 这个问题是子集覆盖的推广. 给定子集覆盖问题 $U=\{u_1,\dots,u_n\}, S_1,\dots,S_m\subseteq U$ 和参数 k', 我们建立 m 个服务器, 每个服务器的布置费用都是 1; 对每个 u_i, 它到服务器 S_j 的距离为 $\infty\cdot\mathbf{1}_{\{u_i\in S_j\}}$, 新问题的判定参数为 $k=k'+n$, 这就完成了归约.

(2) 同样, 我们希望用架设好服务器之后, 相对于它服务的每个用户的平均费用作为贪心法的启发式函数[1]. 即若设 U_s 是 s 服务的用户, 那么希望 $\frac{f_s+\sum_{u\in U_s}d_{us}}{|U_s|}$ 尽可能小. 首先, 我们逐个观察还没有被分配服务器的用户集合 $U\setminus R$ 中的用户, 看看增加一个新服务器能否减少他/她的平均费用, 如果不能的话就让他/她用旧的费用最低的服务器, 否则就计算出一个最佳的 s 和 $U_s\subseteq U\setminus R$, 使得其最小化 $\frac{f_s+\sum_{u\in U_s}d_{us}}{|U_s|}$ 并让这些人用新的服务器. 具体算法如下:

(i) 初始化已经考虑的用户集合 $R\leftarrow\varnothing$, 布置服务器的位置集合 $S\leftarrow\varnothing$.

(ii) 计算 $c=\min_{u\in U\setminus R, s\in S} d_{us}$, 并获得 u, S (无解则视为 ∞).

(iii) 求解优化问题

$$\min_{s\in V\setminus S, U_s\subseteq U\setminus R}\frac{f_s+\sum_{u\in U_s}d_{us}}{|U_s|}\tag{2.1}$$

获得 s, U_s 和最优解的值 c'. (贪心法: 可以在 $O(|V||U|\log|U|)$ 时间内完成, 对每个 $s\in V\setminus S$, 把 $U\setminus R$ 中元素按 d_{us} 从小到大排序然后逐个加入, 选 $O(|U|)$ 个可行解中最大的. 正确性证明可以用交换法.)

(iv) 若 $c\le c'$, 令 $R\leftarrow R\cup U_s, S\leftarrow S\cup\{s\}$, 每个用户 $u\in U_s$ 的均摊代价都记为 $c_u=c'$.

(v) 否则令 $R\leftarrow R\cup\{u\}$, 用户 u 的均摊代价记为 $c_u=c$.

(vi) 若 $R=U$ 就结束, 否则转 (ii).

分析: 首先我们注意, 算法确保 $u\in U_s$ 时, u 就会选择服务器 s. 因此贪心算法的解的费用确实就是 $\sum_{u\in U}c_u$. 设最优解选出的布置点集合是 T^*, 且对 $s\in T^*$, 它服务的用户集合是 U_s^*, 那么类似引理 2 我们有

$$\sum_{u\in U_s^*}c_u\le\left(1+\frac{1}{2}+\cdots+\frac{1}{n}\right)\underbrace{\left(f_s+\sum_{u\in U_s^*}d_{us}\right)}_{\stackrel{\text{def}}{=}C}.\tag{2.2}$$

上式的证明: 固定 s, 按照贪心算法覆盖 U_s^* 中用户的顺序将它们排成一列 $u_1,\dots,u_d$, 考虑 u_i 被覆盖时:

▷ **若 s 不在 T 中** 此时贪心算法会在式 (2.1) 中考虑 s, 因为它总是选用均摊代价最小的服务器, 所以 c_{u_i} 作为式 (2.1) 的最优解, 其值一定小于取 $s, U_s^*\cap(U\setminus R)$ 的均摊代价, 也就是

$$c_{u_i}\le\frac{C}{|U_s^*|-i+1}\quad(\text{分子放大到 }C).$$

[1]如果用对偶法的话可以更清楚地看到这个算法及其分析是怎么来的.

▷ **若 s 已经在 T 中** 此时贪心算法会在 (ii) 中考虑 $d_{u_i,s}$ (但不考虑式 (2.1)), 因此使用上界 $c_{u_i} \le d_{u_i,s}$.

因为 s 加入 T 后就一直不会被删除, 所以可假设在第 u_ℓ (不含) 之后 $s \in T$, 则有

$$\begin{aligned}
\sum_{u \in U_s^*} c_u &\le C\left(\frac{1}{|U_s^*|} + \cdots + \frac{1}{|U_s^*| - \ell + 1}\right) + \sum_{\ell < i \le d} d_{u_i,s} \\
&\le C\left(\frac{1}{|U_s^*|} + \cdots + \frac{1}{|U_s^*| - \ell + 1}\right) + \mathbf{1}_{\{\ell \ne d\}} C & & \left(\ell \ne d \text{ 时用 } \sum_{\ell < i \le d} d_{u_i,s} \le C\right) \\
&\le C\left(1 + \frac{1}{2} + \cdots + \frac{1}{n}\right). & & (|U_s^*| \text{ 放大到 } n)
\end{aligned}$$

这就证明了式 (2.2). 利用这一式子, 我们立刻就有

$$\begin{aligned}
\mathsf{OPT} &= \sum_{s \in T^*}\left(f_s + \sum_{u \in U_s^*} d_{us}\right) \\
&\ge \sum_{s \in T^*}\left(1 + \frac{1}{2} + \cdots + \frac{1}{n}\right)^{-1} \sum_{u \in U_s^*} c_u & & (\text{式 } (2.2)) \\
&\ge \frac{1}{\log n + 1} \sum_{s \in T^*} \sum_{u \in U_s^*} c_u = \frac{1}{\log n + 1} \mathsf{Greedy}. & & (\text{因为最优解服务了所有用户})
\end{aligned}$$

即贪心算法的近似比是 $\log n$. □

参考文献

第一个近似算法是由 Graham 给出的 [Gra69], 其中就是研究了多机调度问题, 但是他还同时给出了一个 $\frac{4}{3}$-近似算法, 现在也已知更加复杂的方法得到更好的近似比. 经典的近似算法教材是 [Vaz01], 内含许多有意思的近似算法设计方法.

对于近似算法, 也有相应的下界问题——高效的近似算法最多能做多好? 这已经来到了当代 (1990–) 计算复杂性理论的领域之中. [AB09, 第 11 和 22 章] 对此有介绍.

[AB09] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. ISBN: 978-0-52-142426-4.

[Gra69] R. L. Graham. "Bounds on Multiprocessing Timing Anomalies". 刊于: *SIAM Journal on Applied Mathematics* 17.2 (1969), pp. 416–429.

[Hås96] J. Håstad. "Clique is Hard to Approximate within $n^{1-\varepsilon}$". 刊于: *Proceedings of the 37th Annual Symposium on Foundations of Computer Science*. FOCS '96. USA: IEEE Computer Society, 1996, p. 627.

[KT06] J. Kleinberg and É. Tardos. *Algorithm Design*. Pearson Education, 2006. ISBN: 978-0-32-129535-4.

[Sla96] P. Slavík. "A Tight Analysis of the Greedy Algorithm for Set Cover". 刊于: *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96. Philadelphia, Pennsylvania, USA: Association for Computing Machinery, 1996, pp. 435–441.

[Vaz01] V. V. Vazirani. *Approximation Algorithms*. Vol. 1. Springer, 2001. ISBN: 978-3-64-208469-0.

编写: WC
E-mail: wchang@pku.edu.cn