算法设计与分析

第13讲 近似算法

汪小林

北京大学 信息科学技术学院

主要内容

- **8.1** 近似算法及其近似比
- **8.2** 多机调度问题的近似算法
- **8.3** 货郎问题的近似算法
- **8.4** 背包问题与多项式时间近似方案

8.1 近似算法及其近似比

- 近似算法：

  A是一个多项式时间算法且对组合优化问题Π的每一个实例I 输出一个可行解σ，记$A(I) = c(\sigma)$，$c(\sigma)$是σ的值。

- 最优化算法：

  恒有$A(I) = OPT(I)$，即A总是输出I的最优解。

- 当Π是最小化问题时，记$r_A(I)=A(I)/OPT(I)$

- 当Π是最大化问题时，记$r_A(I)=OPT(I)/A(I)$

- A的近似比为r （A是r–近似算法）：

  对每一个实例I，$r_A(I) \le r$

- A具有常数比：如果r是一个常数.

可近似性分类

- 假设P≠NP , NP难的组合优化问题按可近似性可分成3类:

(1) 完全可近似的:对任意小的$\varepsilon>0$, 存在$(1+\varepsilon)$-近似算法.

(2) 可近似的: 存在具有常数比的近似算法.

(3) 不可近似的: 不存在具有常数比的近似算法.

最小顶点覆盖问题

- 问题: 任给图$G=<V,E>$, 求G的顶点数最少的顶点覆盖.
- 算法MVC:
  1. 开始时令$V'=\varnothing$.
  2. 任取一条边(u,v), 把u和v加入V'并删去u和v及其关联的边.
  3. 重复上述过程, 直至删去所有的边为止.
  4. V'为所求的顶点覆盖.
- 分析: 算法时间复杂度为$O(m)$, $m=|E|$.

- 问题：近似比是多少？是否是紧的？是否能改进？

最小顶点覆盖问题

- 记$|V'| = 2k$, V'由k条互不关联的边的端点组成.
- 为了覆盖这k条边需要k个顶点, 从而$\mathrm{OPT}(I) \geq k$. 于是,

$$\mathrm{MVC}(I)/\mathrm{OPT}(I) \leq 2k/k = 2$$

- 又, 设图G由k条互不关联的边构成, 显然

$$\mathrm{MVC}(I)=2k,\ \mathrm{OPT}(I)=k.$$

- 这表明MVC的近似比不会小于2, 上面估计的MVC的近似比已不可能再进一步改进.

近似算法的分析

- 研究近似算法的两个基本方面

  设计算法和分析算法的运行时间与近似比.

- 分析近似比

  1. 关键是估计最优解的值.
  2. 构造使算法产生最坏的解的实例.
     如果这个解的值与最优值的比达到或者可以任意的接近得到的近似比(这样的实例称作紧实例), 那么说明这个近似比已经是最好的、不可改进的了;
     否则说明还有进一步的研究余地.
  3. 研究问题本身的可近似性, 即在P≠NP(或其他更强)的假设下, 该问题近似算法的近似比的下界.

8.2 多机调度问题

- 多机调度问题: 任给有穷的作业集A和m台相同的机器, 作业a的处理时间为正整数$t(a)$, 每一项作业可以在任一台机器上处理. 如何把作业分配给机器才能使完成所有作业的时间最短? 即, 如何把A划分成m个不相交的子集A_i使得

$$\max\left\{\sum_{a\in A_i} t(a) \mid i=1,2,\cdots,m\right\}$$

- 最小?

8.2.1 贪心的近似算法

- 负载: 分配给一台机器的作业的处理时间之和.
- 贪心法G-MPS: 按输入的顺序分配作业, 把每一项作业分配给当前负载最小的机器. 如果当前负载最小的机器有2台或2台以上, 则分配给其中的任意一台.

- 例如 3台机器, 8项作业, 处理时间为3,4,3,6,5,3,8,4. 算法给出的分配方案是 {1,4}, {2,6,7}, {3,5,8},
- 负载分别为3+6=9, 4+3+8=15, 3+5+4=12, 完成时间为15.
- 最优的分配方案是 {1,3,4},{2,5,6}, {7,8},
- 负载分别为3+3+6=12, 4+5+3=12, 8+4=12, 完成时间为12.

贪心法的性能

- **定理8.1** 对多机调度问题的每一个有m台机器的实例I,

$$\mathrm{G-MPS}(I) \le \left(2-\frac{1}{m}\right)\mathrm{OPT}(I).$$

- 证明： 显然,

$$(1)\ \mathrm{OPT}(I) \ge \frac{1}{m}\sum_{a\in A} t(a),\ (2)\mathrm{OPT}(I) \ge \max_{a\in A} t(a).$$

- 设机器M_j的负载最大, 记作$t(M_j)$. 又设b是最后被分配给机器M_j的作业. 根据算法, 在考虑分配b时M_j的负载最小, 故

$$t(M_j)-t(b) \le \frac{1}{m}\left(\sum_{a\in A} t(a)-t(b)\right).$$

证明

- 于是

$$\begin{aligned}
\mathrm{G-MPS}(I) = t(M_j) &\le \frac{1}{m}\left(\sum_{a\in A} t(a) - t(b)\right) + t(b) \\
&= \frac{1}{m}\sum_{a\in A} t(a) + \left(1 - \frac{1}{m}\right) t(b) \\
&\le \mathrm{OPT}(I) + \left(1 - \frac{1}{m}\right)\mathrm{OPT}(I) \\
&= \left(2 - \frac{1}{m}\right)\mathrm{OPT}(I).
\end{aligned}$$

紧实例

- m台机器, $m(m-1)+1$项作业, 前$m(m-1)$项作业的处理时间都为1, 最后一项作业的处理时间为m.
- 算法把前$m(m-1)$项作业平均地分配给m台机器, 每台$m-1$项, 最后一项任意地分配给一台机器. G-MPS(I)=2m-1.
- 最优分配方案是把前$m(m-1)$项作业平均地分配给$m-1$台机器, 每台m项, 最后一项分配给留下的机器, OPT(I)=m.

- G-MPS是2-近似算法

8.2.2 改进的贪心近似算法

- 递降贪心法DG-MPS: 首先按处理时间从大到小重新排列作业, 然后运用G-MPS.
- 例如 对上一小节的紧实例得到最优解.
- 对前面的实例, 计算结果如下: 先重新排序 8,6,5,4,4,3,3,3; 3台机器的负载分别为 8+3=11, 6+4+3=13, 5+4+3=12. 比G-MPS的结果好.
- 与G-MPS相比, DG-MPS仅增加对作业的排序, 需要增加时间$O(n\log n)$, 仍然是多项式时间的.

- 定理8.2 对多机调度问题的每一个有m台机器的实例I,

$$\mathrm{DG-MPS}(I) \le \left(\frac{3}{2} - \frac{1}{2m}\right)\mathrm{OPT}(I)$$

证明

- 证明： 设作业按处理时间从大到小排列为$a_1, a_2, \ldots, a_n$, 仍考虑负载最大的机器M_j和最后分配给M_j的作业a_i.
- (1) M_j只有一个作业, 则 $i=1$, 必为最优解.
- (2) M_j有2个或2个以上作业, 则$i \geq m+1$, $\mathrm{OPT}(I) \geq 2t(a_i)$. 于是,

$$\mathrm{DG-MPS}(I) = t(M_j) \leq \frac{1}{m}\left(\sum_{k=1}^{n} t(a_k) - t(a_i)\right) + t(a_i)$$

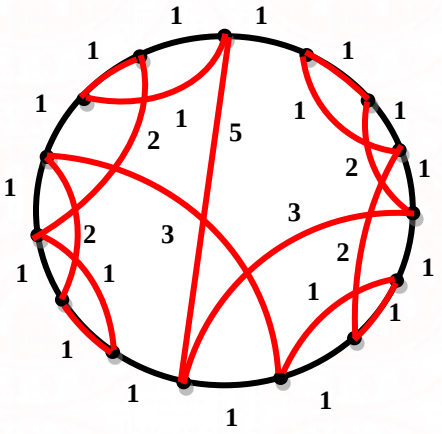
$$= \frac{1}{m}\sum_{k=1}^{n} t(a_k) + \left(1-\frac{1}{m}\right)t(a_i) \leq \mathrm{OPT}(I) + \left(1-\frac{1}{m}\right)\cdot\frac{1}{2}\mathrm{OPT}(I)$$

$$= \left(\frac{3}{2} - \frac{1}{2m}\right)\mathrm{OPT}(I)$$

8.3 货郎问题

- 本节考虑满足三角不等式的货郎问题
- **8.3.1 最邻近法**
- **最邻近法NN**: 从任意一个城市开始, 在每一步取离当前所在城市最近的尚未到过的城市作为下一个城市. 若这样的城市不止一个, 则任取其中的一个. 直至走遍所有的城市, 最后回到开始出发的城市.

- 一个NN性能很坏的例子



最邻近法的性能

- **定理8.3** 对于货郎问题所有满足三角不等式的n个城市的实例I, 总有

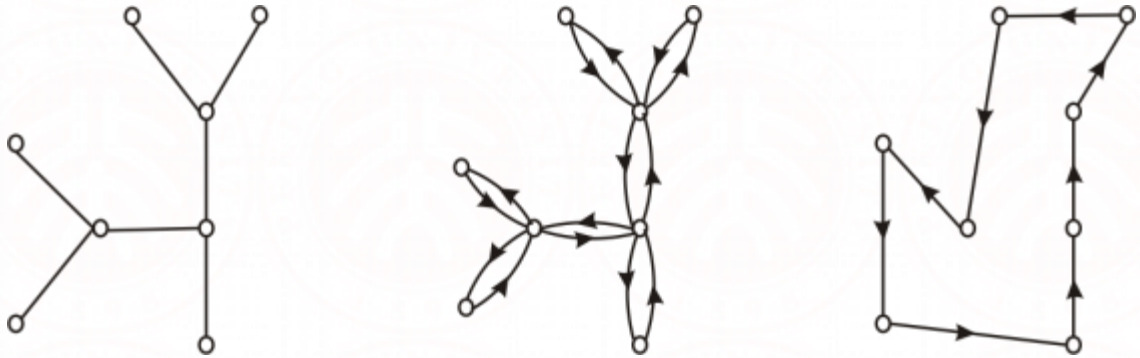
$$\mathrm{NN}(I) \le \frac{1}{2}\left(\lceil \log_2 n \rceil + 1\right)\mathrm{OPT}(I).$$

- 而且, 对于每一个充分大的n, 存在满足三角不等式的n个城市的实例I使得

$$\mathrm{NN}(I) > \frac{1}{3}\left(\log_2 (n+1) + \frac{4}{3}\right)\mathrm{OPT}(I).$$

8.3.2 最小生成树法

- 最小生成树法MST: 首先, 求图的一棵最小生成树T. 然后, 沿着T走两遍得到图的一条欧拉回路. 最后, 顺着这条欧拉回路, 跳过已走过的顶点, 抄近路得到一条哈密顿回路.
- 例



- 求最小生成树和欧拉回路都可以在多项式时间内完成, 故算法是多项式时间的.

最小生成树法的性能

- **定理8.4** 对货郎问题的所有满足三角不等式的实例I,

$$MST(I)<2OPT(I).$$

- 证明： 因为从哈密顿回路中删去一条边就得到一棵生成树, 故T的权小于$OPT(I)$. 沿T走两遍的长小于$2OPT(I)$.
- 因为满足三角不等式, 抄近路不会增加长度, 故

$$MST(I)<2OPT(I).$$

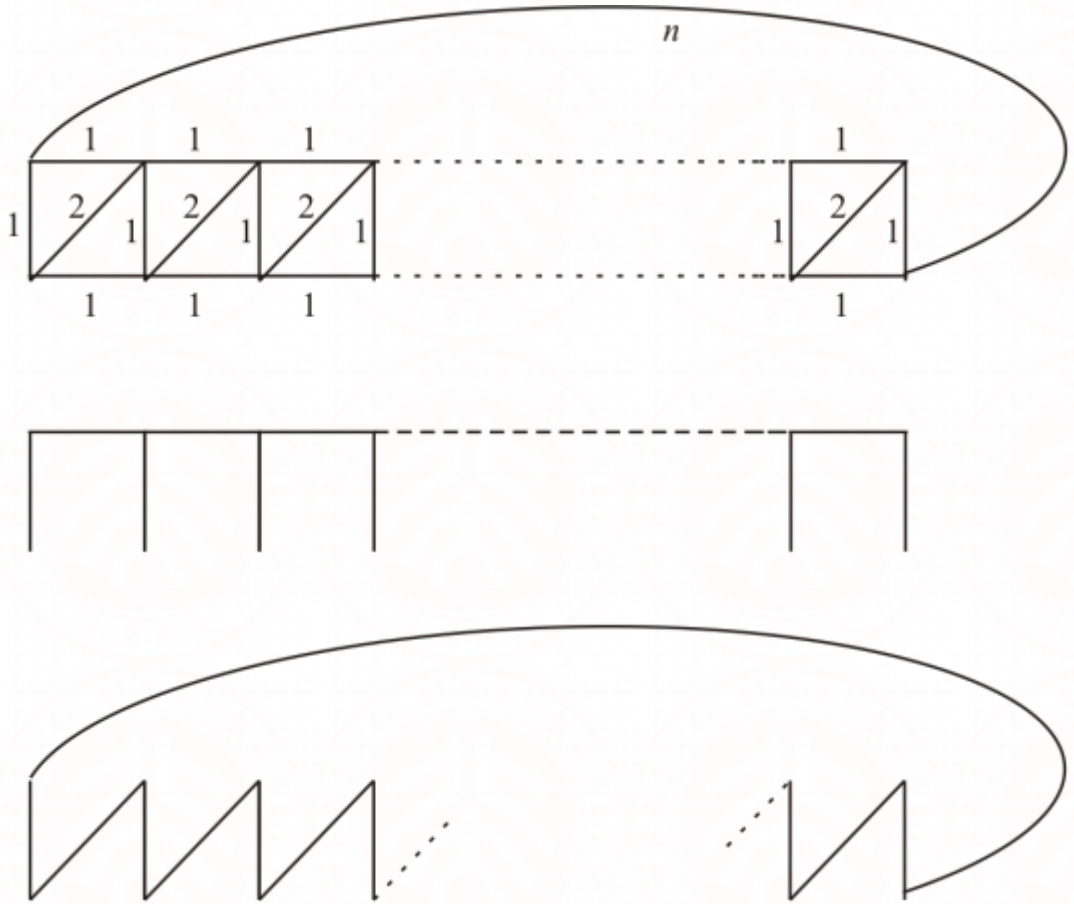
- MST是2-近似算法.

紧实例

OPT(*I*)=2*n*

MST(*I*)=4*n*-2

$$= \left(2 - \frac{1}{n}\right) \mathrm{OPT}(I)$$



8.3.3 最小权匹配法

- 最小权匹配法**MM**: 首先求图的一棵最小生成树T. 记T的所有奇度顶点在原图中的导出子图为H, H有偶数个顶点, 求H的最小匹配M. 把M加入T得到一个欧拉图, 求这个欧拉图的欧拉回路; 最后, 沿着这条欧拉回路, 跳过已走过的顶点, 抄近路得到一条哈密顿回路.

- 求任意图最小权匹配的花算法是多项式时间的, 因此**MM**是多项式时间的.

最小权匹配法的性能

- **定理8.5** 对货郎问题的所有满足三角不等式的实例I,

$$\mathrm{MM}(I) < \frac{3}{2}\mathrm{OPT}(I).$$

- 证 由于满足三角不等式, 导出子图H中的最短哈密顿回路C的长度不超过原图中最短哈密顿回路的长度$\mathrm{OPT}(I)$.
- 沿着C隔一条边取一条边, 得到H的一个匹配. 总可以使这个匹配的权不超过C长的一半.
- 因此, H的最小匹配M的权不超过 $\mathrm{OPT}(I)/2$, 求得的欧拉回路的长小于 $(3/2)\mathrm{OPT}(I)$.
- 抄近路不会增加长度, 得证$\mathrm{MM}(I) < (3/2)\mathrm{OPT}(I)$.
- MM是3/2 -近似算法.

货郎问题的难度

- **定理8.6** 货郎问题(不要求满足三角不等式)是不可近似的, 除非P=NP.
- 证 假设不然, 设A是货郎问题的r-近似算法, $r \le K$, K是常数.
- 任给图$G=<V,E>$, 如下构造货郎问题的实例I_G:
  - 城市集V, $\forall u,v \in V$, 若$(u,v) \in E$, 则令$d(u,v)=1$; 否则令$d(u,v)=Kn$,其中$|V|=n$.
  - 若G有哈密顿回路, 则$OPT(I_G)=n$, $A(I_G) \le rOPT(I_G) \le Kn$;
  - 否则$OPT(I_G)>Kn$, $A(I_G) \ge OPT(I_G) > Kn$.
  - 所以, G有哈密顿回路当且仅当$A(I_G) \le Kn$.

证明

- 于是, 下述算法可以判断图G是否有哈密顿回路: 首先构造货郎问题的实例I_G, 然后对I_G运用算法A. 若$A(I_G)\leq Kn$, 则输出“Yes”; 否则输出“No”.
- 由于K是固定的常数, 构造I_G可在$O(n^2)$时间内完成且$|I_G|=O(n^2)$. A是多项式时间的, A对I_G可在n的多项式时间内完成计算, 所以上述算法是HC的多项式时间算法. 而HC是NP完全的, 推得P=NP.

8.4 背包问题与多项式时间近似方案

- **0-1背包问题的优化形式**: 任给n件物品和一个背包, 物品i的重量为w_i, 价值为v_i, $1 \le i \le n$, 背包的重量限制为B, 其中w_i, v_i以及B都是正整数. 把哪些物品装入背包才能在不超过重量限制的条件下使得价值最大?即, 求子集$S^* \subseteq \{1,2,\ldots,n\}$使得

$$\sum_{i \in S^*} v_i = \max\left\{ \sum_{i \in S} v_i \mid \sum_{i \in S} w_i \le B, S \subseteq \{1,2.\cdots,n\} \right\}.$$

8.4.1 一个简单的贪心算法

贪心算法G-KK

1. 按单位重量的价值从大到小排列物品. 设

$$v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n.$$

2. 顺序检查每一件物品, 只要能装得下就将它装入背包, 设装入背包的总价值为V.

3. 求$v_k=\max\{v_i|i=1,2,\ldots,n\}$. 若$v_k>V$, 则将背包内的物品换成物品$k$.

- 例如 (w_i,v_i): (3,7), (4,9), (2,2), (5,9); $B=6$.
- G-KK给出的解是装入(3,7)和(2,2), 总价值为9. 若把第4件物品改为(5,10), 则装入第4件, 总价值为10.
- 这两个实例的最优解都是装入(4,9)和(2,2), 总价值为11.

G-KK的性能

- 定理8.7 对0-1背包问题的任何实例I, 有

$$OPT(I)<2G\text{-}KK(I).$$

- 证 设物品l是第一件未装入背包的物品, 由于物品按单位重量的价值从大到小排列, 故有

$$OPT(I)<G\text{-}KK(I)+v_l\leq G\text{-}KK(I)+ v_{max} \leq 2G\text{-}KK(I).$$

- G-KK是2-近似算法.

8.4.2 多项式时间近似方案

算法PTAS 输入$\varepsilon>0$和实例I.

- 1. 令$m=\lceil 1/\varepsilon \rceil$.
- 2. 按单位重量的价值从大到小排列物品. 设

$$v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n.$$

- 3. 对每一个$t=1,2,\ldots,m$和t件物品, 检查这t件物品的重量之和. 若它们的重量之和不超过B, 则接着用G-KK把剩余的物品装入背包.
- 4. 比较得到的所有装法, 取其中价值最大的作为近似解.

- PTAS是一簇算法. 对每一个固定的$\varepsilon>0$,PTAS是一个算法, 记作PTAS_ε .

PTSA的性能

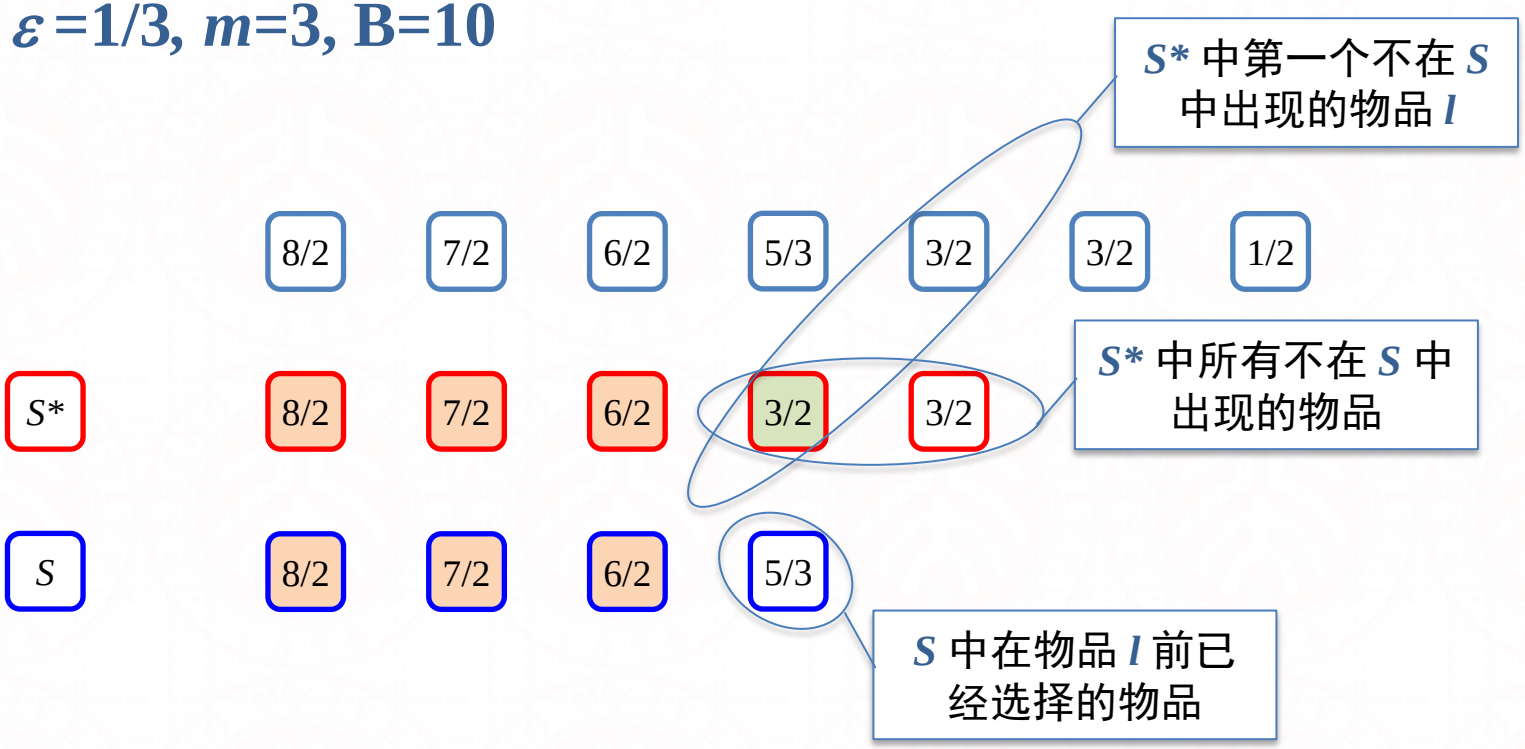
- **定理8.8** 对每一个$\varepsilon>0$和0-1背包问题的实例I,

$$OPT(I)<(1+\varepsilon)PTAS_{\varepsilon}(I),$$

- 且$PTSA_{\varepsilon}$的时间复杂度为$O(n^{1/\varepsilon+2})$.
- 证 设最优解为S^*. 若$|S^*|\le m$, 则算法必得到S^*. 设$|S^*|>m$. 考虑计算中以S^*中m件价值最大的物品为基础, 用G-KK得到的结果S. 设物品l是S^*中第一件不在S中的物品, 在此之前G-KK装入的不属于S^*的物品(肯定有这样的物品, 否则应该装入物品l)的单位重量的价值都不小于v_l/w_l, 当然也不小于S^*中所有没有装入的物品的单位重量的价值, 故有$OPT(I)<\sum_{i\in S}v_i+v_l$. 又, S包括S^*中m件价值最大的物品, 它们的价值都不小于v_l, 故又有$v_l\le\sum_{i\in S}v_i/m$.

PTSA图示

- ε=1/3, m=3, B=10



$$OPT(I) < \sum_{i \in S} v_i + v_l \qquad v_l \leq \sum_{i \in S} v_i / m$$

多项式时间近似方案

- 于是
- $OPT(I)<\sum_{i\in S} v_i+v_l\le\sum_{i\in S} v_i+\sum_{i\in S} v_i/m$
- $\le(1+1/m)PTAS_\varepsilon(I)\le(1+\varepsilon)PTAS_\varepsilon(I)$.
- 时间复杂度. 从n件物品中取t件(t=1,2,…,m), 所有可能取法的个数为

$$c_n^1+c_n^2+\cdots+c_n^m\le m\cdot\frac{n^m}{m!}\le n^m.$$

- 对每一种取法, G-KK的运行时间为$O(n)$, 故算法的时间复杂度为$O(n^{m+1})=O(n^{1/\varepsilon+2})$.
- 多项式时间近似方案: 以$\varepsilon>0$和问题的实例作为输入I, 对每一个固定的$\varepsilon>0$, 算法是$1+\varepsilon$-近似的.

8.4.3 伪多项式时间算法与完全多项式时间近似方案

- 完全多项式时间近似方案: 以$\varepsilon>0$和问题的实例I作为输入, 时间复杂度为二元多项式$p(|I|,1/\varepsilon)$,且对每一个固定的$\varepsilon>0$, 算法的近似比为$1+\varepsilon$.

- 动态规划算法A 记$G_k(d)$: 当只考虑前k件物品时, 为了得到不小于d的价值, 至少要装入的物品重量.

- $$G_k(d)=\min\left\{\sum_{i=1}^{k}w_i x \;\middle|\; \sum_{i=1}^{k}v_i x_i \geq d,\ x_i=0\text{或}1,\ 1\leq i\leq k\right\},$$

$0\leq k\leq n$, $0\leq d\leq D$, $D=v_1+v_2+\ldots+v_n$, 约定: $\min\varnothing=+\infty$.

$\mathrm{OPT}(I)=\max\{d \mid G_n(d)\leq B\}$.

动态规划算法

- 递推公式

$$G_0(d)=\begin{cases}0, & 若d=0,\\ +\infty, & 若d>0,\end{cases}$$

$$G_{k+1}(d)=\begin{cases}\min\{G_k(d),w_{k+1}\}, & 若d\le v_{k+1},\\ \min\{G_k(d),G_k(d-v_{k+1})+w_{k+1}\}, & 若d>v_{k+1},\end{cases}$$

- $0\le k\le n-1,\ 0\le d\le D$.

- A的时间复杂度为$O(nD)=O(n^2v_{max})$, 是伪多项式时间算法.

完全多项式时间近似方案

- 算法FPTAS
- 输入$\varepsilon>0$和实例I.
- 1. 令 $b=\max\left\{\left\lfloor \frac{v_{\max}}{(1+1/\varepsilon)n} \right\rfloor, 1\right\}$.
- 2. 令 $v_i'=\lceil v_i/b \rceil$, $1\le i\le n$. 把所有v_i换成 v_i', 记新得的实例为I'.
- 3. 对I'应用算法A得到解S, 把S取作实例I的解.

- 定理8.9 对每一个$\varepsilon>0$和0-1背包问题的实例I,
- $OPT(I)<(1+\varepsilon)FPTSA(I)$,
- 并且FPTAS的时间复杂度为$O(n^3(1+1/\varepsilon))$.

证明

- 证 由于$(v_i'-1)b< v_i \le v_i'b$, 对任意的$T\subseteq\{1,2,\ldots,n\}$,
- $0\le b\sum_{i\in T}v_i' - \sum_{i\in T}v_i < b|T|\le bn$.
- 设I的最优解为S^*, 注意到S是I'的最优解, 故有

$$\text{OPT}(I)\text{-FPTAS}(I)= \sum_{i\in S^*}v_i - \sum_{i\in S}v_i$$
$$=(\sum_{i\in S^*}v_i-b\sum_{i\in S^*}v_i')+(b\sum_{i\in S^*}v_i'-b\sum_{i\in S}v_i')+(b\sum_{i\in S}v_i'-\sum_{i\in S}v_i)$$
$$\le(b\sum_{i\in S}v_i' - \sum_{i\in S}v_i)<bn.$$

- 对每一个$\varepsilon>0$, 若$b=1$, 则I'就是I, S是I的最优解.
- 设$b>1$, 注意到$v_{\max}\le\text{OPT}(I)$, 得

$$\text{OPT}(I)\text{-FPTAS}(I)<v_{\max}/(1+1/\varepsilon)\le\text{OPT}(I)/(1+1/\varepsilon),$$

- 得 $\text{OPT}(I)<(1+\varepsilon)\text{FPTSA}(I)$.
- 时间主要花在是A对I'的运算, 其时间复杂度为

$$O(n^2v_{\max}/b)= O(n^3(1+1/\varepsilon)) .$$

随机近似算法

- 用随机算法计算近似解
- 随机算法的期望代价$E[C]$
- 如果任意给定输入大小n，都有

$$\max(E[C]/C^*, C^*/E[C]) \le \rho(n)$$

那么随机算法的近似比例为 $\rho(n)$

该算法为随机$\rho(n)$-近似算法

- 随机近似算法和确定近似算法唯一的区别：

其代价是期望代价

MAX-3-CNF可满足问题

- **3-CNF**可满足性问题
- 如果不能使所有从句都满足，也要最大化可满足的从句个数，该问题称为**MAX-3-CNF**可满足问题。
- 如果对每个变量随机分配**0-1**值（概率是**0.5:0.5**），那么可以证明这种随机算法实际上是随机**8/7-**近似算法
- 假定在**3-CNF**中，每个从句都有三个不同的文字，而且不允许 x 和 $\neg x$ 同时出现在同一个从句中。

MAX-3-CNF可满足问题的随机近似算法

定理35.6：假设有一个**MAX-3-CNF**可满足性问题的实例，有 n 个变量 $x_1, x_2, \ldots, x_n$ 和 m 个从句，如果对每个随机变量随机赋值，为 **1** 的概率为 **0.5**，为 **0** 的概率也是 **0.5**，那么这个随机算法是随机 **8/7-** 近似算法。

证明：定义指示器随机变量 Y_i 表示从句 i 是否被满足。

显然只要从句 i 中某一个文字被赋值为 **1**，则 $Y_i = 1$。

因为每个从句都有三个不同的文字，如果 x 和 $\neg x$ 不同时出现在同一个从句中。则从句中三个文字的赋值是彼此独立的。只有当其中的三个文字都被赋值为 **0** 时从句才为 **0**，这个概率为

$$\Pr\{\text{从句 } i \text{ 不能被满足}\} = (1/2)^3 = 1/8。$$

MAX-3-CNF可满足问题的随机近似算法

如果 x 和 $\neg x$ 同时出现在同某个从句中，则该从句一定满足。

因此$\Pr\{$从句 i 能被满足$\} \geq 1-1/8 = 7/8$。于是

$$E[Y_i] \geq 7/8。$$

设随机变量 Y 是可以满足的从句的个数，那么

$$Y = Y_1 + Y_2 + \ldots + Y_m$$

于是

$$E[Y] = E[\sum_{i=1}^{m} Y_i] = \sum_{i=1}^{m} E[Y_i] \geq \sum_{i=1}^{m} 7/8 = 7m/8$$

m 是可满足从句个数的上界，所以该算法的近似比例最多是

$$m/(7m/8) = 8/7$$

集合覆盖问题的贪心算法

- **Set Cover**：给定集合 A 包含 n 个元素，和 A 的一些子集的集合 $W = \{S_1, S_2, \ldots, S_k\}$，以及一个代价函数 $c(S_i)$，要找到一个代价最小的集合 $U = \{S_i\}$ 使 A 的每一个元素都被覆盖。
- 每次都选取最“廉价”的子集 S_i，去掉所覆盖的元素，直到所有元素被覆盖为止。
- 怎样衡量子集的“廉价”
  - 子集 S 的成本效率： $\alpha = c(S) / |S-V|$
  - 其中 V 表示已经被覆盖的元素
  - 即 S 覆盖新元素的平均代价
- 定义 $price(x) : x$ 被某个子集覆盖时的平均代价

贪心集合覆盖算法

1. 初始化，$V \leftarrow \phi$
2. while $V \neq A$ do
   ① 找到成本效率最佳（最小）的子集 S
   ② 赋值 $\alpha(S) = c(S) / |S - V|$
   ③ 选取 S 并对每个 $x \in S - V$，赋值 $price(x) = \alpha(S)$
   ④ $V \leftarrow V \cup S$
3. 输出所有选取的子集

贪心集合覆盖算法的近似比分析

假设上述算法覆盖元素的顺序为 $x_1, x_2, \ldots, x_n$（如果多个元素被同时覆盖，则他们之间的顺序任意选定）。

引理：对于任意的 k $(1 \le k \le n)$，$price(x_k) \le c^* / (n - k + 1)$，其中 c^* 是集合覆盖问题的最优解的代价。

证明：假设最优解选中的集合为 $S^* = \{S^*_i\}$。

在第 j 次循环中，最优解中还没有被选中的子集记为

$$S^*_j = \{S^*_{j,i}\}$$

还没有被覆盖的元素集合记为

$$V' = A - V$$

那么 S^*_j 要覆盖 V' 的代价至多为 c^*。

贪心集合覆盖算法的近似比分析

在元素 x_k 被覆盖的这轮循环中，V' 至少还有 $n-k+1$ 个元素。那么在 S^*_j 中至少有一个子集 $S^*_{j,i}$ 满足

$$(S^*_{j,i}) <= c^* / |V'|$$

否则，如果每一个 $S^*_{j,i}$ 都是 $(S^*_{j,i}) > c^* / |V'|$，则

$$c(S^*_{j,i}) / |S^*_{j,i} - V| = \alpha(S^*_{j,i}) > c^* / |V'|$$

$$c(S^*_{j,i}) > (c^* / |V'|) |S^*_{j,i} - V|$$

$$c(S^*_j) = \Sigma_i(c(S^*_{j,i})) > \Sigma_i((c^* / |V'|) |S^*_{j,i} - V|) \geq c^*$$

（因为 $\Sigma_i(|S^*_{j,i} - V|) \geq |V'|$）。从而有 $c(S^*_j) > c^*$ ，矛盾！

由于 x_k 被最廉价子集覆盖，所以

$$price(x_k) \quad c^* / |V'| \quad c^* / (n - k + 1)$$

贪心集合覆盖算法的近似比分析

定理：该贪心算法是H_n-近似算法（H_n =1+1/2+…+1/n）。

证明：

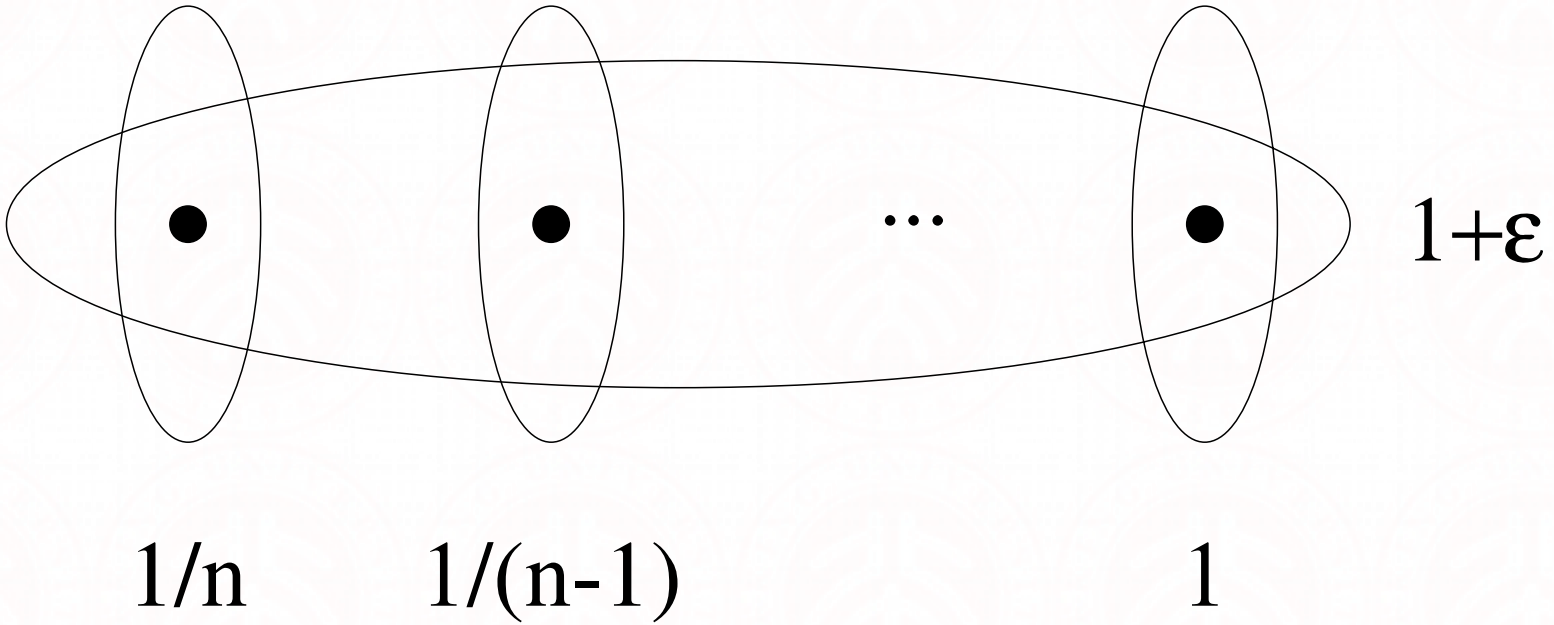
由于贪心算法中每次选取的子集的代价是平均分摊在每个元素上，所以算法中所选取的所有子集的总代价为

$$\sum_{k=1}^{n} price(x_k)$$

根据刚才证明的引理，有

$$\sum_{k=1}^{n} price(x_k) \le (1+\frac{1}{2}+\cdots+\frac{1}{n})c^* = H_n c^*$$

紧的实例



Bin Packing

- **Given *n* items with sizes $a_1, \ldots, a_n \in (0, 1]$, find a packing in unit-sized bins that minimizes the number of bins used.**

- **Approximation algorithm: First-Fit**
  First-Fit $\leq$ 2OPT ： First-Fit $\leq$ (17/10)OPT + 3 ？

- **Theorem BP-1: For any $\varepsilon > 0$, there is no approximation algorithm having a guarantee of $3/2 - \varepsilon$ for the bin packing problem, assuming P $\neq$ NP.**

An asymptotic PTAS

- **Asymptotic PTAS - 渐进多项式时间近似方案**
- **Theorem BP-2: For any ε, $0 < \varepsilon \leq 1/2$, there is an algorithm A_ε that runs in time polynomial in n and finds a packing using at most $(1 + 2\varepsilon)OPT + 1$ bins.**
- **The sequence of algorithms, A_ε, form an asymptotic polynomial time approximation scheme for bin packing, since for each $\varepsilon > 0$ $\exists N > 0$, and a polynomial time algorithm in this sequence, say B, such that B has an approximation guarantee of $1 + \varepsilon$ for all instances having $OPT \geq N$.**

Restricted problem with polynomial time algorithm

- **Lemma BP-3: Let $\varepsilon > 0$ be fixed, and let K be a fixed nonnegative integer. Consider the restriction of the bin packing problem to instances in which each item is of size at least ε and the number of distinct item sizes is K. There is a polynomial time algorithm that optimally solves this restricted problem.**

Restricted problem with polynomial time algorithm

- **Proof: The number of items in a bin is bounded by $\lfloor 1/\varepsilon \rfloor$. Denote this by M. Therefore, the number of different bin types is bounded by R which is a (large!) constant**

$$R = \begin{pmatrix} M+K \\ M \end{pmatrix}$$

- **Clearly, the total number of bins used is at most n. Therefore, the number of possible feasible packings is bounded by P, which is polynomial in n.**

$$P = \begin{pmatrix} n+R \\ R \end{pmatrix}$$

- **Enumerating them and picking the best packing gives the optimal answer.**

Restricted problem within a factor of $(1 + \varepsilon)$

- **Lemma BP-4: Let $\varepsilon > 0$ be fixed. Consider the restriction of the bin packing problem to instances in which each item is of size at least ε. There is a polynomial time approximation algorithm that solves this restricted problem within a factor of $(1 + \varepsilon)$.**

Restricted problem within a factor of (1 + ε)

- **Proof: Let I denote the given instance. Sort the n items by increasing size, and partition them into $K = \lceil 1/\varepsilon^2 \rceil$ groups each having at most $Q = \lceil n\varepsilon^2 \rceil$ items. Notice that two groups may contain items of the same size.**
- **Construct instance J by rounding up the size of each item to the size of the largest item in its group. Instance J has at most K different item sizes.**

- **Therefore, Lemma BP-3, we can find an optimal packing for J. This will also be a valid packing for the original item sizes.**

Restricted problem within a factor of $(1 + \varepsilon)$

- **Let us construct another instance, say J', by rounding down the size of each item to that of the smallest item in its group. Clearly $\mathrm{OPT}(J') \leq \mathrm{OPT}(I)$. The crucial observation is that a packing for instance J' yields a packing for all but the largest Q items of instance J. Therefore,**

$$\mathrm{OPT}(J) \leq \mathrm{OPT}(J') + Q \leq \mathrm{OPT}(I) + Q$$

- **Since each item in I has size at least ε, $\mathrm{OPT}(I) \geq n\varepsilon$. Therefore, $Q = \lfloor n\varepsilon^2 \rfloor \leq \varepsilon\mathrm{OPT}$. Hence,**

$$\mathrm{OPT}(J) \leq (1 + \varepsilon)\mathrm{OPT}(I).$$

Proof of Theorem BP-2

- **Let I denote the given instance, and I' denote the instance obtained by discarding items of size $< \varepsilon$ from I. By Lemma BP-4, we can find a packing for I' using at most $(1 + \varepsilon)\mathrm{OPT}(I')$ bins. Next, we start packing the small items (of size $< \varepsilon$) in a First-Fit manner in the bins opened for packing I'. Additional bins are opened if an item does not fit into any of the already open bins.**
- **If no additional bins are needed, then we have a packing in $(1+\varepsilon)\mathrm{OPT}(I') \leq (1 + \varepsilon)\mathrm{OPT}(I)$ bins.**

Proof of Theorem BP-2

- **Clearly, all but the last bin must be full to the extent of at least $1 - \varepsilon$. Therefore, the sum of the item sizes in I is at least $(M - 1)(1 - \varepsilon)$. Since this is a lower bound on OPT, we get**

$$M \leq \text{OPT}/(1-\varepsilon) + 1 \leq (1+2\varepsilon)\text{OPT} + 1$$

- **where we have used the assumption that $\varepsilon \leq 1/2$. Hence, for each value of ε, $0 < \varepsilon \leq 1/2$, we have a polynomial time algorithm achieving a guarantee of $(1+2\varepsilon)\text{OPT}+1$.**

Algorithm A_ε for bin packing

1. **Remove items of size $< \varepsilon$.**
2. **Round to obtain constant number of item sizes (Lemma BP-4).**
3. **Find optimal packing (Lemma BP-3).**
4. **Use this packing for original item sizes.**
5. **Pack items of size $< \varepsilon$ using First-Fit.**

课堂练习

9.1 Give an example on which First-Fit does at least as bad as $5/3 \cdot \mathrm{OPT}$.

9.2 (Johnson [149]) Consider a more restricted algorithm than First-Fit, called Next-Fit, which tries to pack the next item only in the most recently started bin. If it does not fit, it is packed in a new bin. Show that this algorithm also achieves factor 2. Give a factor 2 tight example.

9.3 (C. Kenyon) Say that a bin packing algorithm is *monotonic* if the number of bins it uses for packing a subset of the items is at most the number of bins it uses for packing all n items. Show that whereas Next-Fit is monotonic, First-Fit is not.