算法设计
——近似算法

陈卫东

chenwd@scnu.edu.cn

华南师范大学计算机学院

2021-11

NP-hard问题的求解技术

如何对付NP-hard问题？

- 许多问题一般无需求最优解，求近似最优解即可
- NP-hard问题性质：集中研究某些典型NP-hard问题好算法
- 人类社会的丰富经验可以借鉴
- 大自然中的一些现象可以借鉴

NP-hard问题的求解技术

- 典型求解技术
  - 精确求解技术（Exact Algorithms）
  - 近似求解技术（Approximate Algorithms）
  - 启发式求解技术（Heuristic Algorithms）

NP-hard问题的求解技术

- **近似求解技术（Approximate Algorithms）**

  贪心法、定价法、线性规划松弛法 、动态规划法…

  - 多项式时间运行、近似求解，能快速得到质量有保障的解.
  - 如何评价这类算法？

  性能的理论分析通常需要一些技巧！

提 纲

- 近似算法的相关概念
  - 近似度的度量
  - 近似问题相关的问题复杂类
- 典型问题的近似算法举例
  - 点覆盖问题（Vertex Cover）
  - 集合覆盖问题（Set Cover）
  - 中心选址问题（Center Selection）
  - 负载均衡问题（多机调度问题）
  - 0-1背包问题（0-1 Knapsack）

近似算法的相关概念

- 近似度的度量

1. 相关概念

(1) 最优化问题（目标函数、约束条件）

(2) 解、可行解、解值、解集、最优解、最优值（最优解的目标值）

近似算法的相关概念

近似度的度量

2．近似算法和近似度的度量

(1) **近似算法**——多项式时间算法（往往是基于贪心策略的启发式算法），所得问题的解值在最优值附近。

(2) **近似度的度量**——近似比（Approximation Rate）

- 算法A求解问题实例I的近似比定义为

  $$r_A(I)=\max\{A(I)/O(I), O(I)/A(I)\},$$

  其中$A(I)$是算法A求解实例I所得解值，而$O(I)$是最优值。
- 设 $r_A(I)=1+\varepsilon$，则称算法A所得解是ε-最优的。
- 最坏近似比：$r_A(n)=\sup\{r_A(I)\mid |I|\le n\}$

  ——即大小不超过n的实例的近似比的上确界。
- 渐近近似比

  例如，$r_A(I)\le 11/9+4/O(I)$，因此其渐近近似比为11/9。

近似算法的相关概念

上述近似比称作相对性能界，为何不用绝对性能界(差界)？

——差界即$|A(I)-O(I)|\leq k$，k是某个常量。只有很少的 **NP**-hard问题有这样的近似算法。

- [例1]　平面图的3-着色问题
  ——容易得到一个差界为1的近似算法

- [例2]　0-1背包问题
  ——不存在带有差界的近似算法(可用反证法证明)

近似算法的相关概念

- 近似度的度量

3．近似问题——能在多项式时间内按给定的近似比（或渐近近似比）求解的最优化问题。

4．伪多项式算法（pseudo-polynomial time algorithm）——当问题实例中所有数据大小受限于多项式的情况下多项式时间运行的算法。

近似算法的相关概念

- 近似问题相关的复杂类

1. APX类

- **APX**($r(n)$)——有近似比不超过$r(n)$的多项式时间算法的近似问题集。
- **APX**——$\cup_{c>1}$**APX**(c)，即所有具有常数近似比的多项式时间算法的近似问题集。
- **APX***——$\cap_{c>1}$**APX**(c)，即所有具有任意小常数近似比多项式时间算法的近似问题集。

近似算法的相关概念

- 近似问题相关的复杂类

2. PTAS类

——包含所有具有多项式时间近似方案的问题。

- 多项式时间近似方案**PTAS**（polynomial time approximation scheme）：一个近似问题的算法，它的输入是该问题的一个实例I和任意常数$\varepsilon>0$，使得对任意固定的ε算法在多项式时间(关于I的大小)内产生实例I的近似解，且最坏近似度不超过$1+\varepsilon$。例如，$\Theta(n^{1/\varepsilon})$、$\Theta(n2^{1/\varepsilon})$、$\Theta(n/\varepsilon)$等。

近似算法的相关概念

- 近似问题相关的复杂类

3. FPTAS类

——包含所有具有完全多项式时间近似方案的问题。

- 完全多项式时间近似方案**FPTAS**（fully polynomial approximation scheme）：一个近似问题的算法，它的输入是该问题的一个实例I和任意常数$\varepsilon>0$，使得对任意固定的ε算法在多项式时间（关于I的大小和$1/\varepsilon$）内产生实例I的近似解，且最坏近似度不超过$1+\varepsilon$。

近似算法的相关概念

- 近似问题相关的复杂类

4. 几个复杂类间的关系：

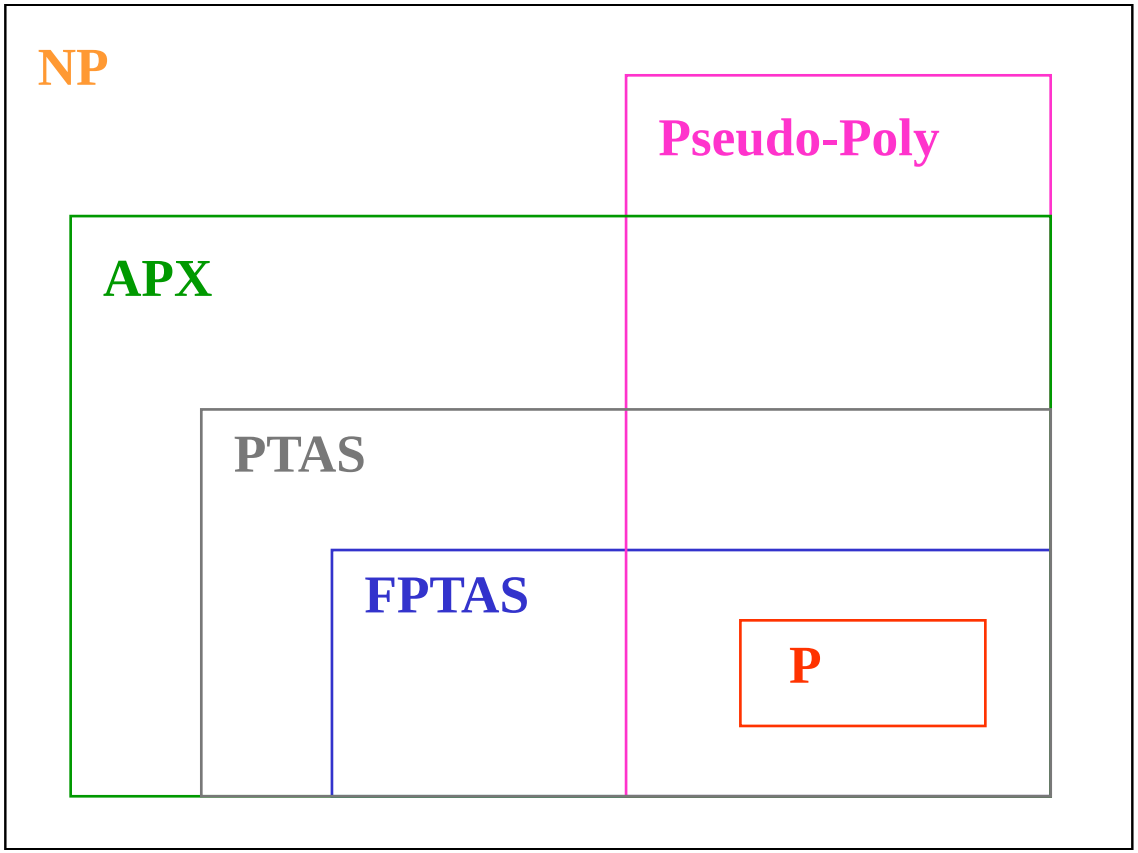
$$\mathbf{P} \subseteq \mathbf{FPTAS} \subseteq \mathbf{PTAS} \subseteq \mathbf{APX}$$

- 这里**P**限制为多项式时间复杂度的优化问题类。对于不属于**P**的优化问题类，希望能确定它是否属于**FPTAS**、**PTAS**或**APX**。若属于**APX**，则希望找到尽可能小的c使得问题属于**APX**(c)。如果不行的话(即对于一些更困难的问题)，则希望找到尽可能增加缓慢的函数$r(n)$使得问题属于**APX**($r(n)$)。

- 对于任何一个优化问题来说，一般应该存在一个函数$r(n)$使得问题属于**APX**($r(n)$)，这只需要考虑问题的平凡解（trivial solutions）即可，不过这个函数$r(n)$可能很差。我们感兴趣的是非平凡的求解算法。

- 求解**NP**-hard最优化问题的在多项式时间运行的近似算法具有重要的实用价值，有关的复杂性类可用于区别问题在什么近似程度能在多项式时间内求解。

近似算法的相关概念



典型问题的近似算法举例

- 点覆盖问题（Vertex Cover）
  - 问题描述
  - 2-近似算法

重复下列过程直到图中没有边可选择为止：
从图中选取一条边$e=<u,v>$，将u和v两点加入到点覆盖集合T中（T初始化为空集），并从图中删去这两点及与这两点相关联的所有边。

- 性能分析：设上述算法中有m条边被选取,由此得到一个$2m$大小的覆盖,即$A(I)=2m$. 注意到这m条边是两两不共点的,因此为了覆盖这些边,每条边上至少有一个点在最小覆盖中,因此最小覆盖的大小$O(I)\geq m$.故上述算法的近似比为: $A(I)/O(I)\leq 2$.

典型问题的近似算法举例

- **Vertex Cover**
  - **Greedy 算法**

重复下列过程直到图中所有边都被覆盖为止：
从图中选取一最大度点*v*加入到点覆盖集合***T***中（***T***初始化为空集），并从图中删去点*v*及其相关联的所有边。

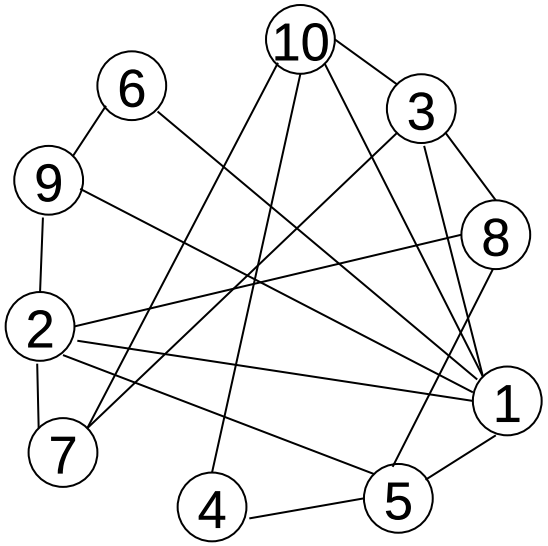
  - **性能分析**

（1）这里的Greedy算法通常要比第一个算法的效果要好。
（可以举出很多实例验证这一点，例如星型图、环型图等）

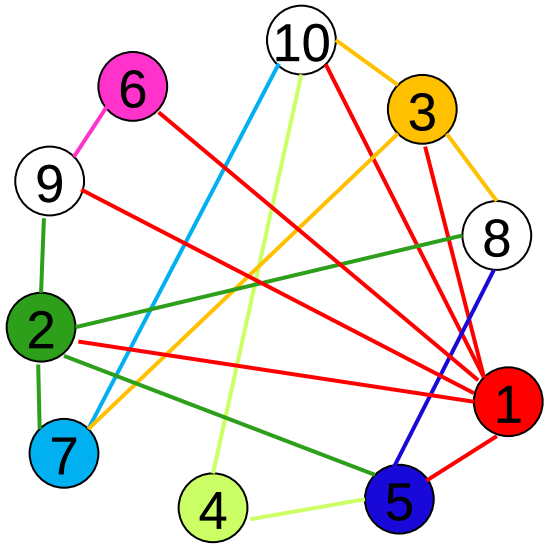
典型问题的近似算法举例

- **Vertex Cover**
  - **Greedy 算法**
  - **性能分析**

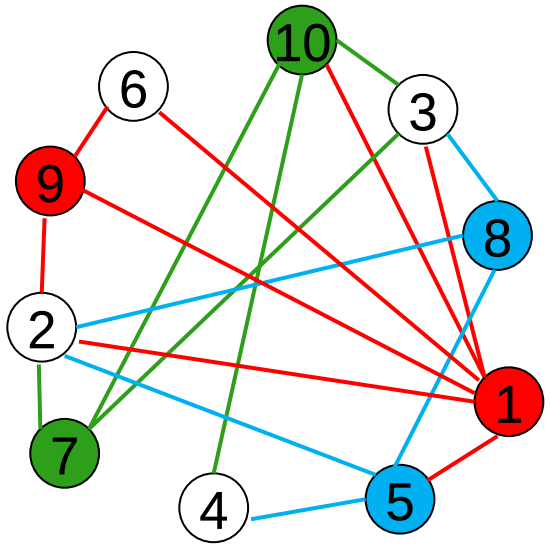
（2）然而，在某些实例上Greedy算法的效果要更差一些。



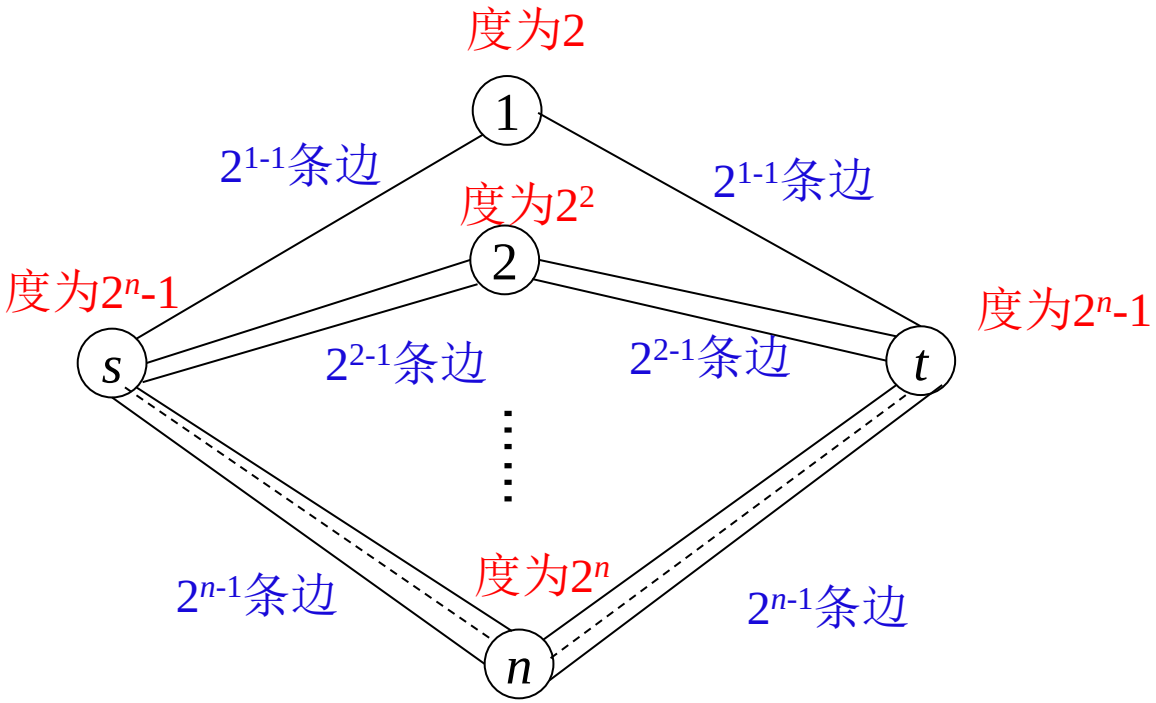
实例图1



Greedy算法执行结果



2-近似算法执行结果



实例图2

Greedy算法求解的结果为 {*n*,*n*-1,…,1}

2-近似算法求解的一种结果为 {1,*s*,2,*t*}

最优解为 {*s*,*t*}

典型问题的近似算法举例

- **Vertex Cover**
  - **Greedy 算法**
  - **性能分析**

（3）实际上，Greedy算法是一个$O(\ln\Delta)$-近似算法，其中Δ是图的最大度。证明思路如下：

- Greedy算法实际上是每次选择出一个平摊代价最小的点（一个点的平摊代价为1除以该点当时能覆盖住的尚未被覆盖的边数）。我们可以看作将该点的代价均分到此次被覆盖的那些边上。这一过程重复进行直到所有边都被覆盖为止。近似度分析可分三个步骤：

（1）所有边的代价和恰是选出的那些点的个数。即$\sum_{e\in E}\mathbf{Price}(e)=\boldsymbol{A}(\mathbf{I})$。

（2）与任意点v相关联所有边的代价和$\le H(d(v))$。 ——证明见后

（3）设最小点覆盖为$\{v_1, v_2,..,v_t\}$。由于图中每条边必与在这其中某点相关联，因此 $\boldsymbol{A}(\mathbf{I})=\sum_{e\in \boldsymbol{E}}\mathbf{Price}(e)\le\sum_{1\le i\le t}H(d(v_i))\le H(\Delta)*t=H(\Delta)*\boldsymbol{O}(\mathbf{I})$。

典型问题的近似算法举例

➢（2）与任意点v相关联所有边的代价和≤$H(d(v))$的证明。

证明：

设点v相关联的边有k条,即$d(v)=k$，且按它们在Greedy算法中被覆盖的次序依次设为$e_1, e_2,..,e_k$（若多条边同时被覆盖则其次序任意给定）。当e_i将被覆盖时（与它一起可能有多条边同时被覆盖），点v相关联的边中至少有$k-i+1$条没被覆盖（即至多$i-1$条边已被覆盖），点v的平摊代价至多为$1/(k-i+1)$。根据贪心选择规则，此时所作选择的平摊代价不大于$1/(k-i+1)$——此时可能选择点v导致e_i被覆盖，也可能选择别的某个点（其平摊代价不超过点v的平摊代价）导致e_i被覆盖。于是，

$$\mathbf{Price}(e_i)\le 1/(k-i+1)，\ 1\le i\le k。$$

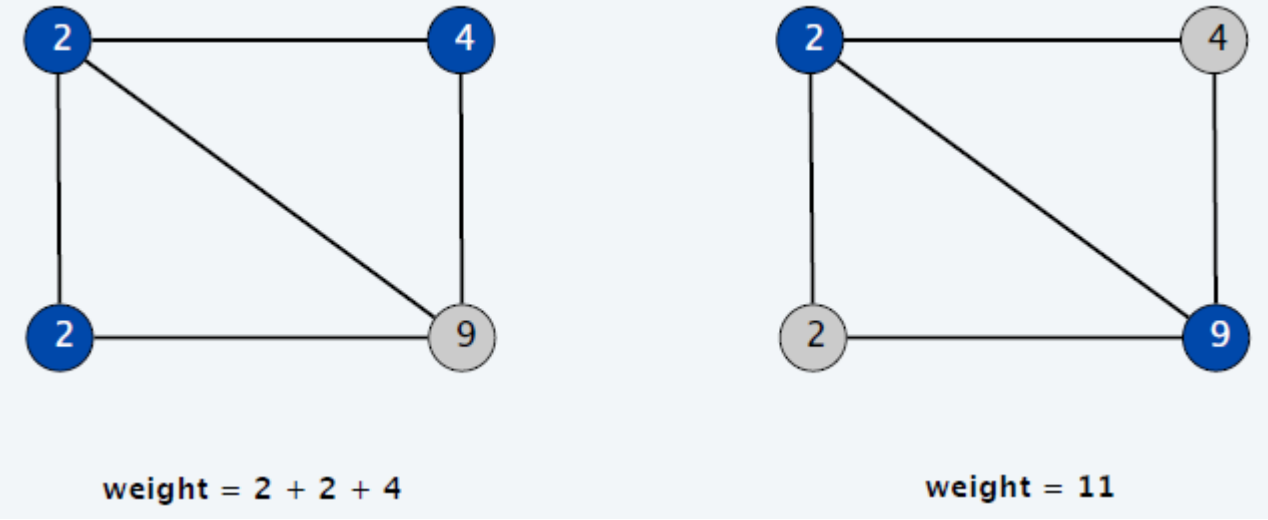
即，$\mathbf{Price}(e_1)\le 1/k$，　$\mathbf{Price}(e_2)\le 1/(k-1)$，……，$\mathbf{Price}(e_k)\le 1/1$。

因此，**与任意点v相关联所有边的代价和≤$1/k+1/(k-1)+\ldots+1/2+1=H(k)$**。

典型问题的近似算法举例

- 带权点覆盖问题（Weighted Vertex Cover）
  - 问题描述

给定无向图$G=<V,E>$，每个点v都有一个正权值$w(v)$，求图G的一个点覆盖使得其中的点权之和最小。



典型问题的近似算法举例

- 带权点覆盖问题（Weighted Vertex Cover）
  - 一个基于定价法的2-近似算法

Vertex-Cover-Approx(G, w)：

对所有的 $e \in E$，令 $p_e = 0$

While 存在边 $e=(i,j)$ 使得 i 和 j 都不是紧的

选择一条这样的边 e

在不违反公平性的条件下加大 p_e

EndWhile

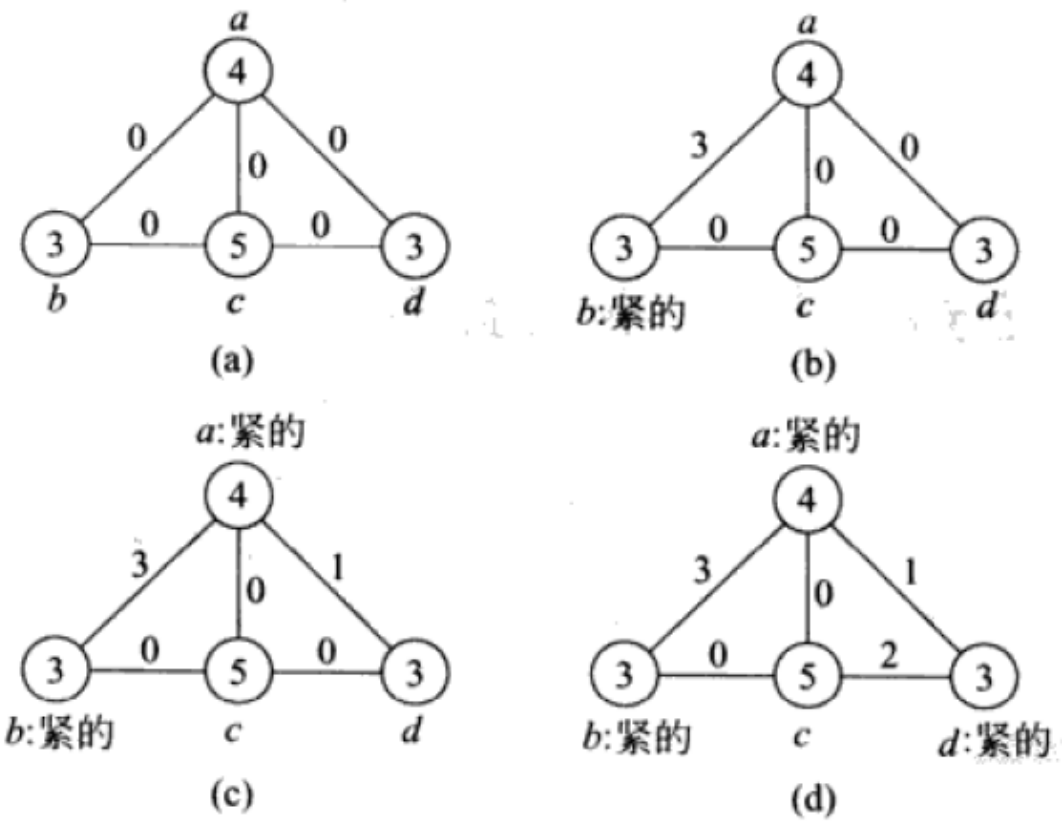
令 S 是所有紧顶点的集合

Return S

如果 $\sum_{e=(i,j)} p_e = w_i$，则称顶点 i 是紧的

典型问题的近似算法举例

- 带权点覆盖问题（ **Weighted Vertex Cover** ）
  - 一个基于定价法的2-近似算法



典型问题的近似算法举例

- 带权点覆盖问题（Weighted Vertex Cover）
  - 一个基于定价法的2-近似算法

算法显然在多项式时间内运行，且最终输出一个点覆盖S。下面分析近似比。设S^*是最优点覆盖，我们证明$w(S) \le 2w(S^*)$。

$$w(S) = \sum_{i \in S} w_i = \sum_{i \in S} \sum_{e=(i,j)} p_e \le \sum_{i \in V} \sum_{e=(i,j)} p_e = 2 \sum_{e \in E} p_e \le 2w(S^*)$$

↑ all nodes in S are tight

↑ $S \subseteq V$, prices ≥ 0

↑ each edge counted twice

↑ fairness lemma

典型问题的近似算法举例

- 带权点覆盖问题（ Weighted Vertex Cover ）
  - 一个基于LP的2-近似算法

思路：首先将最小带权点覆盖问题表述为一个0-1整数规划问题，然后通过松弛技术得到其相应的线性规划（**LP**）问题。求解该线性规划问题，然后对其解分量进行四舍五入可得到原问题的一个近似最优的可行解。

典型问题的近似算法举例

Step1. 将原问题表述为如下0-1整数规划（**IP**）问题：

$$\textbf{min} \sum_{v\in V} w(v)x(v)$$

$$\textbf{s.t.}\ \ x(u)+x(v)\geq 1,\ (u,v)\in E;$$

$$x(v)\in\{0,1\},\ \ v\in V$$

Step2. 上述0-1整数规划问题通过将条件“$x(v)\in\{0,1\}$”松弛为“$x(v)\in[0,1]$”就得到如下线性规划（**LP**）问题：

$$\textbf{min} \sum_{v\in V} w(v)x(v)$$

$$\textbf{s.t.}\ \ x(u)+x(v)\geq 1,\ (u,v)\in E;$$

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

Step3. 解上述线性规划问题得最优解$\boldsymbol{X}^*$。将$\boldsymbol{X}^*$中的各个分量通过四舍五入后得到一个解$\boldsymbol{S}$，输出$\boldsymbol{S}$及对应的权值即可。

典型问题的近似算法举例

- 性能分析

显然算法是在多项式时间内运行, 且最终输出一个点覆盖(可行解). 下面只需要分析近似比即可. 设$\boldsymbol{S}^*$是最优点覆盖, 则$\boldsymbol{S}^*$必是**Step** 2 中线性规划问题的可行解. 于是,

（1） $w(\boldsymbol{X}^*)=\sum_{v\in \boldsymbol{V}} w(v)\, x^*(v) \le w(\boldsymbol{S}^*) =\sum_{v\in \boldsymbol{S}^*} w(v)$

（2） $$\begin{aligned} w(\boldsymbol{X}^*)=\sum_{v\in \boldsymbol{V}} w(v)\, x^*(v) &\ge \sum_{v\in \boldsymbol{V} \text{ 且} x^*(v)\ge 0.5} w(v)x^*(v) \\ &\ge \sum_{v\in \boldsymbol{V} \text{ 且} x^*(v)\ge 0.5} w(v)/2 \\ &= \sum_{v\in \boldsymbol{S}} w(v)/2=w(\boldsymbol{S})/2 \end{aligned}$$

即, $w(\boldsymbol{S})/2 \le w(\boldsymbol{X}^*) \le w(\boldsymbol{S}^*)$.

因此, 上述算法的近似比 $w(\boldsymbol{S})/w(\boldsymbol{S}^*) \le 2$.

典型问题的近似算法举例

- 加权点覆盖问题的不可近似性

Theorem. [Dinur–Safra 2004] If **P** ≠ **NP**, then no ρ-approximation for WEIGHTED-VERTEX-COVER for any ρ < 1.3606 (even if all weights are 1).

On the Hardness of Approximating Minimum Vertex Cover

Irit Dinur* Samuel Safra†

May 26, 2004

Abstract

We prove the Minimum Vertex Cover problem to be NP-hard to approximate to within a factor of 1.3606, extending on previous PCP and hardness of approximation technique. To that end, one needs to develop a new proof framework, and borrow and extend ideas from several fields.

Open research problem. Close the gap.

典型问题的近似算法举例

- 集合覆盖问题（Set Cover）
  - 问题描述

回想一下在讨论 NP 完全性时集合覆盖问题基于 n 个元素的集合 U 和一组 U 的子集 $S_1,\cdots,S_m$，如果这些子集中的若干个的并集等于整个 U，则称这若干个子集是一个集合覆盖.

现在考虑的这个问题中，每一个子集 S_i 关联一个权 $w_i \geqslant 0$. 我们的目标是找一个集合覆盖 $\mathscr{C}$ 使得总的权

$$\sum_{S_i \in \mathscr{C}} w_i$$

最小. 注意这个问题至少和我们早先见到的集合覆盖的判定形式一样的难. 如果令所有的 $w_i=1$，那么集合覆盖的最小权小于等于 k 当且仅当存在不超过 k 个子集的集合覆盖 U.

典型问题的近似算法举例

- 集合覆盖问题（Set Cover）
  - 贪心算法

    每次选择出一个平摊代价最小的集合（一个集合S_i的平摊代价为该集合的权值w_i除以该集合当时能覆盖住的尚未被覆盖的元素个数$|S_i \cap R|$）且将该集合的代价均分到此次覆盖的那些元素上。这一过程重复进行直到所有元素都被覆盖为止。

Greedy-Set-Cover：

开始时 $R=U$ 且没有被选择的集合

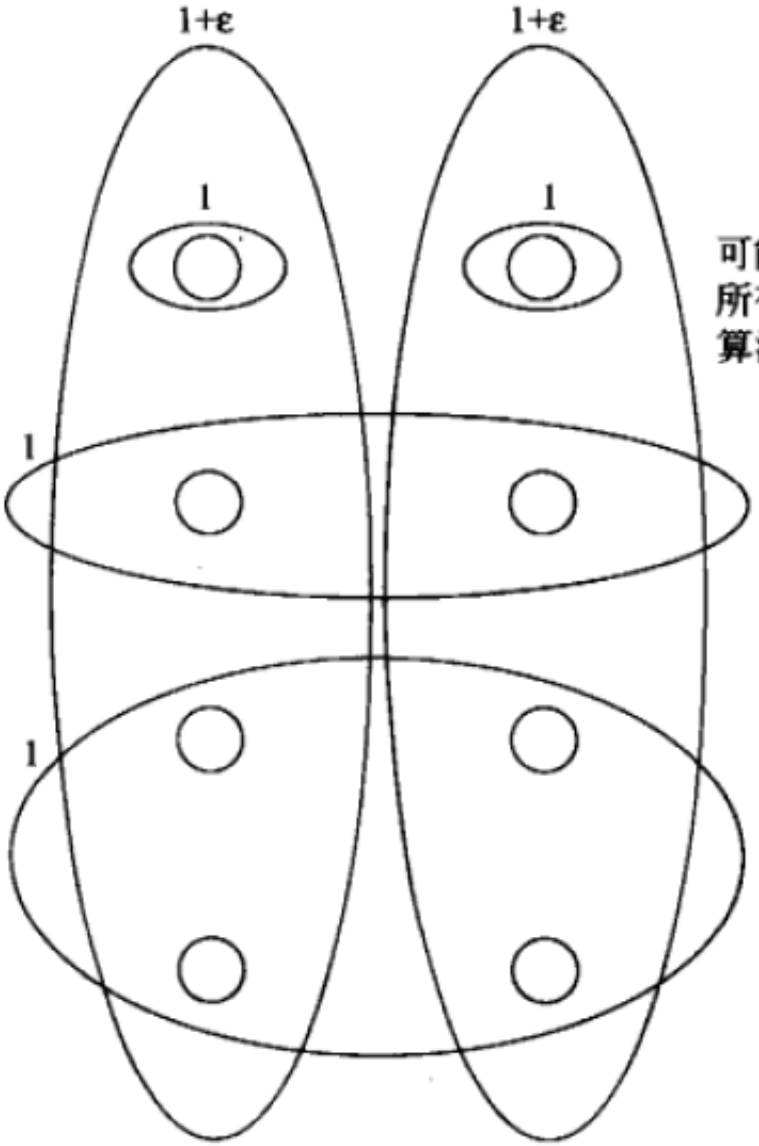
While $R \neq \varnothing$

选择 $w_i/|S_i \cap R|$ 最小的集合 S_i

从 R 中删去集合 S_i

EndWhile

Return 所有被选择的集合



可能用 2 个集合覆盖所有的结点，但贪心算法没有发现它们

集合覆盖问题的一个实例，集合的权为 1 或 1+ε，其中 $\varepsilon>0$ 是一个很小的数．贪心算法选择总权等于 4 的集合，而没有选择权为 $2+2\varepsilon$ 的最优解

典型问题的近似算法举例

- 集合覆盖问题（Set Cover）
  - 性能分析

设 $d^* = \max_i |S_i|$ 表示最大的集合的大小，有下述近似结果.

Greedy-Set-Cover 选择的集合覆盖 C 的权不超过最优权 w^* 的 $H(d^*)$ 倍.

证明思路：

（1）显然，所有元素代价和恰是选出的那些集合代价之和: $\sum_{e\in U}\mathbf{Price}(e)=\boldsymbol{A}(\mathbf{I})$。

（2）后面将证明：任意集合S_i中元素上代价之和$\le H(|S_i|)*\text{cost}(S_i)$，即

$$\sum_{e\in S_i}\mathbf{Price}(e)\le H(|S_i|)*\text{cost}(S_i)。$$

（3）设最优覆盖为$\{S_{j1}, S_{j2},..,S_{jt}\}$，则$U$中每个元素必包含在这其中的至少一个集合中。因此，

$$\boldsymbol{A}(\mathbf{I})=\sum_{e\in U}\mathbf{Price}(e)\le \sum_{1\le k\le t}\sum_{e\in S_{jk}}\mathbf{Price}(e)\le\sum_{1\le k\le t}H(|S_{jk}|)\text{cost}(S_{jk})$$
$$\le H(d^*)\sum_{1\le k\le t}\text{cost}(S_{jk})=H(d^*)\boldsymbol{O}(\mathbf{I})。$$

其中，$d^*=\boldsymbol{max}\{|S_i|\ |1\le i\le m\}$，$H(d^*)=1+1/2+1/3+...+1/d^*$。显然$d^*\le n$。因此，该算法是$H(d^*)$-近似算法，当然也是$H(n)$-近似算法。由于$H(n)=\Theta(\ln n)$，因此该算法是$\Theta(\ln n)$-近似算法。

典型问题的近似算法举例

- 集合覆盖问题（Set Cover）
  - 性能分析

设 $d^* = \max_i |S_i|$ 表示最大的集合的大小，有下述近似结果.

Greedy-Set-Cover 选择的集合覆盖 C 的权不超过最优权 w^* 的 $H(d^*)$ 倍.

（2）任意集合S_i中元素代价之和$\leq H(|S_i|)*\text{cost}(S_i)$，即$\sum_{e\in S_i}$ **Price**$(e)\leq H(|S_i|)*\text{cost}(S_i)$。

证明：设集合S中有k个元素，且按它们在贪心算法中被覆盖的次序依次设为$e_1, e_2,..,e_k$（若多个元素同时被覆盖则其次序任意给定）。当e_i将被覆盖时（与它一起可能有多个元素同时被覆盖），集合S中至少有k-i+1个元素没被覆盖（即至多i-1个元素已被覆盖），S的平摊代价至多为cost(S)/(k-i+1)，根据贪心选择规则，此时所做选择的平摊代价不大于cost(S)/(k-i+1)——此时可能选择S导致e_i被覆盖，也可能选择别的某个集合（其平摊代价比不超过S）导致e_i被覆盖。于是，

Price$(e_i)\leq$cost(S)/(k-i+1)，$1\leq i\leq k$。即，

Price$(e_1)\leq$cost(S)/k，**Price**$(e_2)\leq$cost(S)/(k-1)，……，**Price**$(e_k)\leq$cost(S)/1。

因此，有 $\sum_{e\in S}$ **Price**$(e)\leq$cost(S) $*(1/k+1/(k-1)+\ldots+1/2+1)=H(|S|)*\text{cost}(S)$。□

典型问题的近似算法举例

- 集合覆盖问题（Set Cover）
  - 性能分析

有趣的是注意到这个界限实质上是最好的，因为有实例使得贪心算法计算得这么坏. 为了看到这样的实例是如何产生的，再一次考虑图 11.6 中的例子. 推广这个例子的元素基础集 U 由两竖列组成，每列各有 $n/2$ 个元素. 对某个小的 $\epsilon>0$，仍有两个权为 $1+\epsilon$ 的集合，它们分别包含一列. 再构造 $\Theta(\log n)$ 个集合作为图中其他集合的推广：一个包含最底部的 $n/2$ 个结点，另一个包含上面一点的 $n/4$ 个结点，还有一个包含再上面一点的 $n/8$ 个结点，等等. 这些集合的权都为 1.

贪心算法依次选择大小为 $n/2, n/4, n/8, \cdots$ 的集合，产生一个权为 $\Omega(\log n)$ 的解. 另一方面，选择两个各自包含一列的集合产生权为 $2+2\epsilon$ 的最优解. 通过更加复杂的构造可以加强这个结果，产生使得贪心算法给出的权非常接近最优权的 $H(n)$ 倍的实例. 事实上，已经用复杂得多的方法证明没有多项式时间的近似算法能够达到比最优值 $H(n)$ 倍好得多的近似界限，除非 $\mathcal{P}=\mathcal{NP}$.

能够利用归约作近似吗？ 在开发算法之前，我们暂时停下来考虑一个有趣的问题：**顶点覆盖**容易归约到**集合覆盖**，而我们刚才已经看到**集合覆盖**的一个近似算法. 由此能够得到关于**顶点覆盖**可近似性的什么信息？这个问题的讨论说明近似结果与多项式时间归约相互作用的某些微妙方式.

首先考虑所有的权都等于 1 的特殊情况，即找一个最小的顶点覆盖. 把这称做*无权的情况*. 回想一下，我们用从判定形式的无权的**顶点覆盖**的归约，即

$$\textbf{顶点覆盖} \leqslant_p \textbf{集合覆盖}$$

证明**集合覆盖**是 NP 完全的. 这个归约说："如果有解**集合覆盖**问题的多项式时间算法，就可以用这个算法在多项式时间内解**顶点覆盖**问题." 现在有给出**集合覆盖**问题近似解的多项式时间算法. 这意味能够用这个算法设计一个**顶点覆盖**的近似算法吗？

命题 11.12 能够用**集合覆盖**的近似算法给出带权的**顶点覆盖**问题的 $H(d)$—近似算法，其中 d 是图的最大度数.

证 证明基于证明**顶点覆盖**$\leqslant_p$ **集合覆盖**的归约，这个归约也可以推广到带权的情况. 考虑一个带权的**顶点覆盖**实例，它由图 $G=(V,E)$ 和每一个顶点 i 的权 w_i 给出. 定义**集合覆盖**的实例如下. 基础集 U 等于 E. 对每一个顶点 i，定义集合 S_i 由所有与顶点 i 关联的边组成并且它的权等于 w_i. 覆盖 U 的一组集合恰好对应一个顶点覆盖. 注意 S_i 的最大大小恰好等于最大度数 d.

因此，能够用**集合覆盖**的近似算法找到一个顶点覆盖，其权不超过最小权的 $H(d)$ 倍.

典型问题的近似算法举例

中心选址问题（Center Selection）

问题描述

考虑下述情景. 我们有 n 个地点的集合 S，譬如说纽约州北部地区的 n 个小镇，要选择 k 个中心建设大型购物中心. 希望每一个小镇的人到这些中心中的一个购物，要选择这 k 个购物中心的地址成为中心的.

从更加形式地定义问题的输入开始. 给定整数 k，n 个地点（对应小镇）的集合 S 以及距离函数. 当把实例中的地点看做平面上的点时，距离函数是点之间标准的欧几里得距离，并且平面上的任何一点都可以作为中心的地址. 但是，我们开发的算法可以用于更一般的距离. 在应用中，距离有时表示直线距离，也可以表示从点 s 到点 z 的旅行时间或旅行距离（即，沿公路的距离），甚至是旅行费用. 允许任何满足下述自然性质的距离函数：

- 对所有的 $s \in S, dist(s,s)=0$
- 距离是对称的：对所有的地点 $s,z \in S, dist(s,z)= dist(z,s)$
- 三角不等式：$dist(s,z)+dist(z,h) \geqslant dist(s,h)$

典型问题的近似算法举例

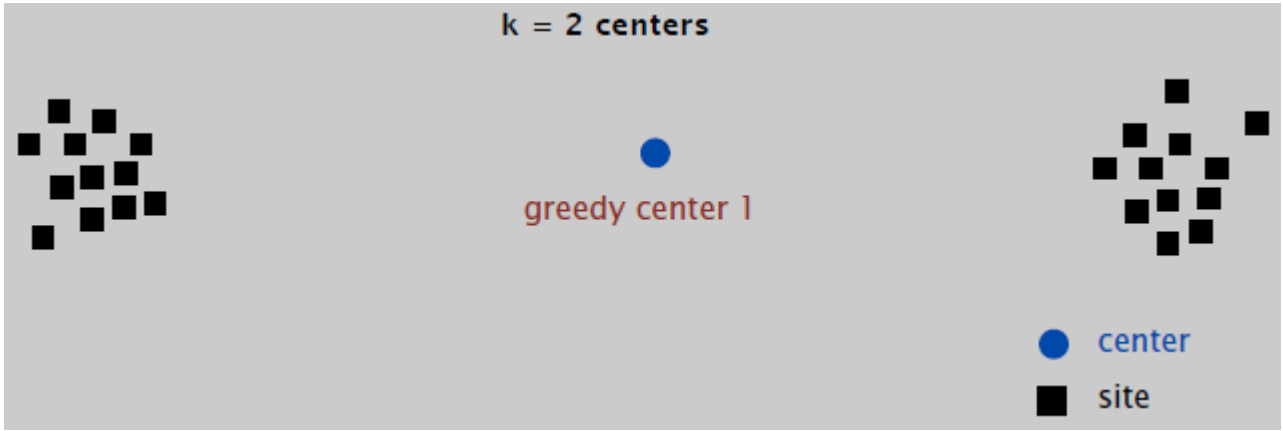
- 中心选址问题（Center Selection）
  - 问题描述

其次必须说明目标是希望中心成为“中心的”含义是什么. 设 C 是中心集. 假设小镇上的人都到最近的购物中心购物. 于是，定义地点 s 到中心集的距离 $dist(s, C) = \min_{c \in C} dist(s, c)$. 如果每一个地点离某一个中心的距离不超过 r，即对所有的 $s \in S$，$dist(s, C) \leqslant r$，则称 C 构成一个 r-覆盖. 使得 C 是 r-覆盖的最小 r 称为 C 的覆盖半径，记作 $r(C)$. 换句话说，中心集 C 的覆盖半径等于任何人到离他或她最近的中心的最远距离. 我们的目标是选择 k 个中心的集合 C 使得 $r(C)$ 尽可能的小.

典型问题的近似算法举例

- 中心选址问题（Center Selection）
  - 贪心算法

最简单的贪心算法大概会如下进行. 把第一个中心选在只有一个中心时最佳可能位置，然后添加中心，每一次使覆盖半径尽可能地缩小. 结果是这个方法有点太简单化了，以至于失去使其实用的价值：在某些情况下它可以导致非常坏的解.



典型问题的近似算法举例

- 中心选址问题（Center Selection）
  - 贪心算法

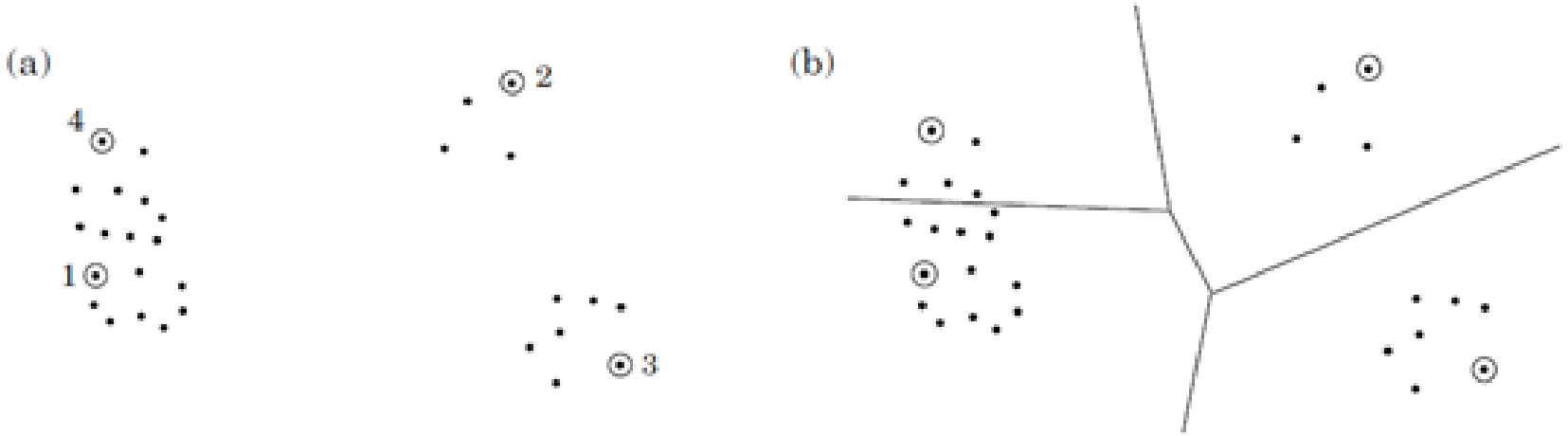
注意一个惊人的事实：我们最终的贪心 2-近似算法是第一个贪心算法非常简单的修改，而后者是没有使用价值的. 最重要的变化大概就是我们的算法总是选择原有地点作为中心(即，每一个购物中心建在一个小镇上，而不建在两个小镇的之间).

算法思想：首先在给定的n个点中随机地选定一点记作s_1。一般地，如果i-1个点$\{s_1, s_2,\ldots, s_{i-1}\}$已经选好（$2\le i\le k$），则在剩余点中选出到点集$\{s_1, s_2,\ldots, s_{i-1}\}$的距离最大的一点记作$s_i$。

按照上述方法选出k个点$s_1, s_2,\ldots, s_k$作为k个Cluster的中心，所有n个点按照距离Cluster中心最近的原则归属于某一个Cluster中。

典型问题的近似算法举例

- 中心选址问题（Center Selection）
  - 贪心算法



典型问题的近似算法举例

- 中心选址问题（Center Selection）
  - 贪心算法

设 $k \leqslant |S|$（否则规定 $C=S$）
任意选择一个地点 s 并且令 $C=\{s\}$
While $|C| < k$
　　选择一个地点 $s \in S$ 使得 $dist(s,C)$ 最大
　　把 s 加入 C
EndWhile
Return C 作为选中的中心集

近似比分析：考虑剩余点中选出到点集$\{s_1, s_2, \ldots, s_k\}$的距离最大的一点，记作$s_{k+1}$，该距离记作$r$。显然，上述算法所得的$k$个Cluster中每个Cluster的直径≤$2r$。即对于给定的问题实例**I**，算法所得解的目标值（所得k个Cluster对应圆饼的最大直径）**A(I)**≤$2r$。

另一方面，由于k+1个点$s_1, s_2, \ldots, s_k, s_{k+1}$中必有2个点属于最优解的某个Cluster中（鸽笼原理），由此得**O(I)**≥r（这k+1点中最近两点的距离是r，从而任意两点间的距离至少是r）。因此，算法近似比**A(I)/O(I)**≤2。

典型问题的近似算法举例

- 中心选址问题（Center Selection）
  - 不可近似性

Theorem. Unless **P = NP**, there no ρ-approximation for CENTER-SELECTION for any $\rho < 2$.

Pf. We show how we could use a $(2-\varepsilon)$ approximation algorithm for CENTER-SELECTION to solve DOMINATING-SET in poly-time.

- Let $G = (V, E)$, k be an instance of DOMINATING-SET.
- Construct instance G' of CENTER-SELECTION with sites V and distances
  - $dist(u, v) = 1$ if $(u, v) \in E$
  - $dist(u, v) = 2$ if $(u, v) \notin E$
- Note that G' satisfies the triangle inequality.
- G has dominating set of size k iff there exists k centers C^* with $r(C^*) = 1$.
- Thus, if G has a dominating set of size k, a $(2-\varepsilon)$-approximation algorithm for CENTER-SELECTION would find a solution C^* with $r(C^*) = 1$ since it cannot use any edge of distance 2. ▪

支配集问题的判定形式为: 给定无向图G, 问是否存在k个点能支配图中所有点？注意：一个点能支配其相邻的所有点.

该问题是NP-hard的.

证明到此处得到该多项式时间算法可解决支配集问题. 当**P≠NP**时,这是不可能的,由此产生矛盾.

典型问题的近似算法举例

- 负载均衡问题（多机调度问题）
  - 问题描述

负载均衡问题的形式描述如下. 给定 m 台机器 $M_1, \cdots, M_m$ 和 n 项作业，每一项作业 j 有处理时间 t_j. 要把每一项作业分配给一台机器使得所有机器的负载尽可能的“均衡”.

更具体地说，设有一个作业对机器的分配，令 $A(i)$ 表示分配给机器 M_i 的作业集，机器 M_i 需要工作的总时间为

$$T_i = \sum_{j \in A(i)} t_j$$

称这是机器 M_i 的负载. 我们要求工期——即，所有机器的最大负载 $T = \max_i T_i$——最小. 求最小工期分配的调度问题是 NP 难的，但这里不证明了.

  - 2-近似算法
  - 3/2-近似算法
  - ε-近似算法（多项式近似方案）

典型问题的近似算法举例

- 负载均衡问题（多机调度问题）
  - 2-近似算法

先考虑这个问题的一个很简单的贪心算法. 算法以任意的顺序枚举所有的作业，当轮到作业 j 时，把它分配给至今负载最小的机器.

Greedy-Balance：

　　开始时没有已分配的作业

　　对所有的机器 M_i，令 $T_i=0$ 和 $A(i)=\varnothing$

For $j=1,\cdots,n$

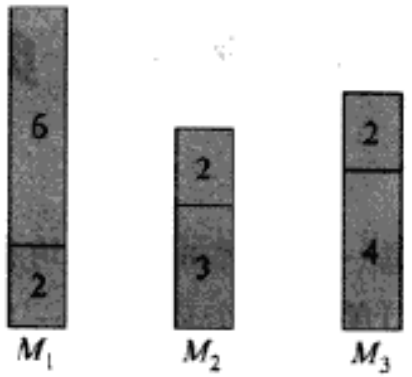
　　令 M_i 是达到最小值 $\min_k T_k$ 的机器

　　把作业 j 分配给机器 M_i

　　令 $A(i)\leftarrow A(i)\cup\{j\}$

　　令 $T_i\leftarrow T_i+t_j$

EndFor



对 3 台机器和大小为 2,3,4,6,2,2 的 6 项作业运行贪心的负载均衡算法的结果

典型问题的近似算法举例

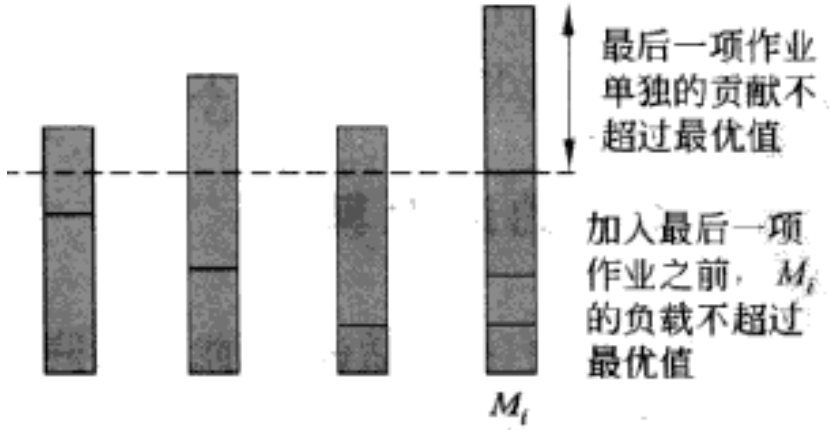
- 负载均衡问题（多机调度问题）
  - 2-近似算法

引理 11.1　$T^* \geqslant \frac{1}{m}\sum_j t_j$.

引理 11.2　$T^* \geqslant \max_j t_j$.

定理 11.3　算法 Greedy-Balance 产生一个作业对机器的分配且工期 $T \leqslant 2T^*$.

证明过程参见教材P421-422。



典型问题的近似算法举例

- 负载均衡问题（多机调度问题）
  - 3/2-近似算法

现在分析上述贪心算法的变种：首先按处理时间递降顺序排列作业，然后和前面一样地进行. 我们将证明所得到的分配的工期不超过最优值的 1.5 倍.

Sorted-Balance：

开始时没有已分配的作业
对所有的机器 M_i，令 $T_i=0$ 和 $A(i)=\varnothing$
按处理时间 t_j 的递降顺序排列作业
设 $t_1 \geqslant t_2 \geqslant \cdots \geqslant t_n$
For $j=1,\cdots,n$
    设机器 M_i 达到最小值 $\min_k T_k$
    把作业 j 分配给机器 M_i
    令 $A(i) \leftarrow A(i) \cup \{j\}$
    令 $T_i \leftarrow T_i + t_j$
EndFor

典型问题的近似算法举例

- 负载均衡问题（多机调度问题）
  - 3/2-近似算法

引理 11.4　如果有多于 m 项作业，则 $T^* \geqslant 2t_{m+1}$.

定理 11.5　算法 Sorted-Balance 产生一个作业对机器的分配且工期 $T \leqslant \frac{3}{2}T^*$.

证明过程参见教材**P421-422**。

典型问题的近似算法举例

- 负载均衡问题（多机调度问题）
  - ε-近似算法（多项式近似方案）

算法思路如下：

令$L=\Sigma_{1\le i\le n}\, t_i$. 对于任意小正数$\varepsilon'>0$，令$\varepsilon=\varepsilon'/(m-1)$。称处理时间$t_j\ge\varepsilon L$的作业$t_j$为大作业. 显然，大作业个数至多为$\lfloor 1/\varepsilon \rfloor$.

对所有大作业在所有m台机器上的具体安排进行枚举。对大作业的每一种具体安排，使用上述(1)中的贪心算法来安排剩下的所有(小)作业，得到对应的负载。算法最后输出负载最小的安排。

典型问题的近似算法举例

- 负载均衡问题（多机调度问题）
  - ε-近似算法（多项式近似方案，即PTAS）

算法分析：大作业至多有$c=m^{\lfloor 1/\varepsilon \rfloor}$种具体安排。对大作业的每一种具体安排，执行一次贪心算法，时间为O(nm)，故总时间为O(nmc)=O($nm^{1+\lfloor 1/\varepsilon \rfloor}$)，即O($nm^{1+\lfloor m-1/\varepsilon' \rfloor}$)。

（对于常数ε'这是关于n的多项式时间算法，但不是关于n和$1/\varepsilon'$的多项式时间算法）。

下面分析算法的近似比。设最优安排为S，考虑算法中当大作业的安排与最优安排S中大作业安排一致的那个时刻后使用贪心算法产生的结果。注意贪心算法将余下所有(小)作业依次安排在当前负载较轻的机器上。贪心算法结束后负载最大的那台机器上的负载为A(I)，下分两种情况讨论。

(i) 该机器上所分配作业均为大作业，此时显然A(I)=O(I)，即A(I)/O(I)=1≤1+ε'。

(ii) 该机器上所分配的作业中包含有贪心算法所分配的(小)作业，设最后一个(小)作业为j（其处理时间$t_j<\varepsilon L$）。负载A(I)分为两部分：最后那项作业之前的负载A(I)$-t_j \le (L-t_j)/m$，最后那项作业的处理时间$t_j<\varepsilon L$。因此，A(I)$\le L/m+(1-1/m)t_j \le (1+(m-1)\varepsilon)L/m$。考虑到O(I)$\ge L/m$，因此A(I)/O(I)$\le (1+\varepsilon(m-1)) \le 1+\varepsilon'$。

因此，上述算法是负载平衡问题的一个多项式近似方案。

典型问题的近似算法举例

- **0-1背包问题 (0-1 Knapsack)**
  - 问题描述
  - 2-近似算法

考虑下求解0/1背包问题的贪心算法：先按物品的单位价值由大到小排序，然后依次确定每件物品是否放入背包（如果放得下则放入背包，否则不放入背包，继续考虑下一物品），直到所有物品都处理完为止。

设自然数k满足在上述贪心法中前$k-1$个物品可全装入了背包，而再放第k件物品时则背包装不下。由此可得如下算法：将上述贪心算法的结果与v_k（第k物品的价值）比较，最终谁大就输出谁。

下面证明所得算法是2-近似算法。

考虑到贪心法是按单位价值由大到小来考虑放物品，故

$$\mathbf{O(I)} \le (v_1+v_2+\ldots+v_{k-1})+v_k。$$

由于$(v_1+v_2+\ldots+v_{k-1}) \le \mathbf{A(I)}$且$v_k \le \mathbf{A(I)}$，因此有$\mathbf{O(I)} \le 2\mathbf{A(I)}$,即近似比$\mathbf{O(I)/A(I)} \le 2$。

典型问题的近似算法举例

- **0-1背包问题 (0-1 Knapsack)**
  - 一个ε-近似算法（完全多项式近似方案，即FPTAS）

(1) 动态规划法

（所有物品价值为正整数，物品重量及背包容量可为任意正实数）

设$V=v_1+v_2+\ldots+v_n$，且设$L(i,p)$表示从物品1,2,…,i中选择物品装包时所形成总价值为p的装包的总重量最小的重量，其中$1\le i\le n$，$1\le p\le V$。

初始条件： 当$p=v_1$时, $L(1,p)=w_1$; 否则$L(1,p)=\infty$， 其中$1\le p\le V$。

递推公式：

case ($v_{i+1}>p$)：$L(i+1,p)=L(i,p)$，其中$1\le i\le n-1, 1\le p\le V$。

case ($v_{i+1}\le p$)：$L(i+1,p)=\mathbf{min}\{L(i,p),L(i,p-v_{i+1})+w_{i+1}\}$，其中$1\le i\le n-1, 1\le p\le V$。

最终求出 $\mathbf{max}\{p\mid L(n,p)\le M\}$即可。

算法时间复杂度为$O(nV)=O(n^2v^*)$，其中v^*是最大价值物品的价值。

- [例] 设0-1背包问题中n=4,C=9, n个物品的重量分别为{2,3,4,5}，其价值分别{3,4,5,7}。使用动态规划法求解最优装包方法。

设 $L[i,p]$表示从 $\{u_1, u_2, \ldots, u_i\}$挑选物件装包容时所形成总价值为$p$的装包的总重量最小的重量，其中$1 \le i \le 5$，$1 \le p \le 19$。则

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2																			
3																			
4																			

- [例] 设0-1背包问题中n=4,C=9, n个物品的重量分别为{2,3,4,5}，其价值分别{3,4,5,7}。使用动态规划法求解最优装包方法。

  - **另一个种动态规划法**

（所有物品容量、背包容量为正整数，物品价值可为任意正实数）

设 $V[i,j]$表示从 $\{u_1, u_2, \dots, u_i\}$挑选物件装入容量为j的背包中的最优装包的价值，其中$0 \le i \le 5$，$0 \le j \le 9$ 。则

初始条件: $V[i,0]=0$, $V[0,j]=0$

递推公式: 若 $i>0$且$j<w_i$， $V[i,j]=V[i-1,j]$

若 $i>0$且$j \ge w_i$, $V[i,j]=\max\{V[i-1,j], V[i-1,j-w_i]+v_i\}$

	0	1	2	3	4	5	6	7	8	9
0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
1	**0**	0	3	3	3	3	3	3	3	3
2	**0**	0	3	4	4	7	7	7	7	7
3	**0**	0	3	4	4	7	8	9	9	12
4	**0**	0	3	4	5	7	8	10	11	12

典型问题的近似算法举例

0-1背包问题 (0-1 Knapsack)

（2）基于动态规划法的完全多项式近似方案

基本思路：如果物品的价值都比较小，则上述动态规划算法可在多项式时间内得到最优解。如果物品价值较大时，则可将所有物品的价值按一给定比例缩小使得缩小后的价值是受限于问题大小n和$1/\varepsilon$的多项式。然后采用动态规划法求新实例的最优解，由此再得原实例的近似解。

对于任意正数$\varepsilon>0$，令$K=\varepsilon v^*/n$，A_ε算法描述如下：

Step1. 将实例I转换为实例I'：$v_i' = \lfloor v_i/K \rfloor$, $w_i' = w_i$, $M'=M$, $i=1,2,\ldots,n$

Step2. 使用动态规划算法求解实例I'，得最优解的物品集S'。

Step3. 令$A(I)=max\{\sum_{i\in S'} v_i, v^*\}$，其中$v^*$是实例$I$的最大价值物品的价值，并输出$A(I)$。

典型问题的近似算法举例

- **性能分析：**

易知算法时间复杂度为$O(n^2v^*/K)=O(n^3/\varepsilon)$。这是关于问题大小$n$和$1/\varepsilon$的多项式时间算法。下面分析算法的近似比。

易知实例I和实例I' 可行解集完全一样。由$v_i' =\lfloor v_i/K\rfloor$得

$$v_i-K\le Kv_i'=K\lfloor v_i/K\rfloor\le v_i。$$

设实例I的最优解为物品集$\boldsymbol{S}$，实例I'的最优解为物品集$\boldsymbol{S}'$。考虑到实例I和实例I' 可行解集完全一样，因此，

$$O(I)\ge\sum_{i\in S'}v_i\ge\sum_{i\in S'}K\lfloor v_i/K\rfloor=K\sum_{i\in S'}v_i'=KO(I')$$

$KO(I')\ge\sum_{i\in S}Kv_i'\ge\sum_{i\in S}(v_i-K)\ge O(I)-nK$，即$O(I)\le KO(I')+nK$

由于 $KO(I')\le\sum_{i\in S'}v_i\le\boldsymbol{A}(I)$且$v^*\le\boldsymbol{A}(I)$，因此，

$$O(I)\le KO(I')+nK\le\boldsymbol{A}(I)+\varepsilon v^*\le(1+\varepsilon)\boldsymbol{A}(I)，$$

从而近似比为：$O(I)/\boldsymbol{A}(I)\le1+\varepsilon$。

因此，上述算法是求解0-1背包问题的一个完全多项式近似方案。

设计近似算法的方法

- 近似算法设计与分析的基本技术
  - 贪心法
  - 定价法
  - 线性规划松弛法
  - 动态规划