

# 近似算法

# 近似算法的有关概念

## 近似算法

适用于组合优化问题  
一般是多项式时间的算法

## 近似算法A的可行解及值

A是一个多项式时间算法且对组合优化问题 $\Pi$ 的每一个实例 $I$ 输出一个可行解 $\sigma$  (满足约束条件的解), 记作

$$A(I) = c(\sigma),$$

称 $c(\sigma)$ 是 $\sigma$ 的值

# 算法A的近似比 $r$

设 $\text{OPT}(I)$ 表示实例  $I$  的最优解的值,

(1)  $\Pi$ 是最小化问题,  $r_A(I) = A(I)/\text{OPT}(I)$

(2)  $\Pi$ 是最大化问题,  $r_A(I) = \text{OPT}(I)/A(I)$

**最优化算法A:** 对所有的实例  $I$  恒有

$A(I) = \text{OPT}(I)$ , 即  $r_A(I) = 1$ .

A的近似比为  $r$  (A是 $r$ -近似算法):

对每一个实例  $I$ ,  $r_A(I) \leq r$ .

**A具有常数近似比:**  $r$ 是一个常数.

# 可近似性分类

假设 $P \neq NP$ ,  $NP$ 难的组合优化问题按可近似性可分成 3 类:

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

**可近似的:** 存在具有常数比的近似算法, 例如最小顶点覆盖问题、多机调度问题

**不可近似的:** 不存在具有常数比的近似算法, 例如货郎问题

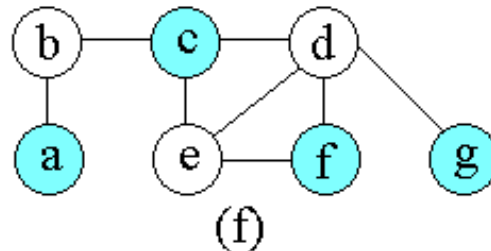
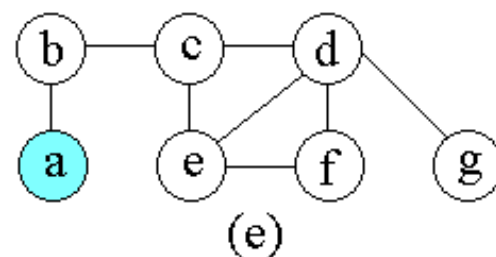
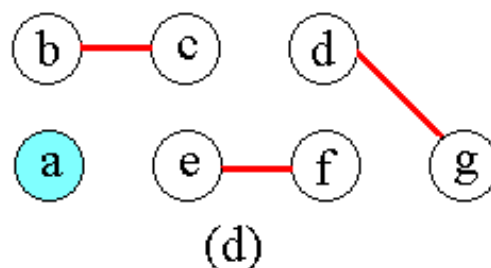
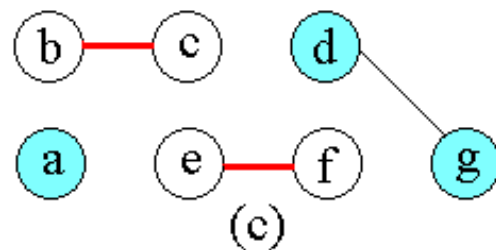
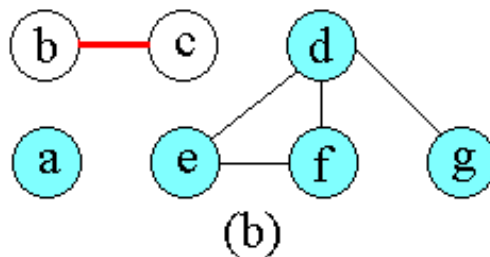
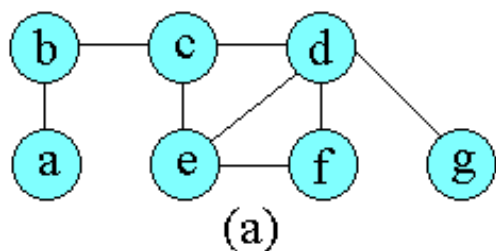
# 顶点覆盖问题的近似算法

问题描述：无向图 $G=(V,E)$ 的顶点覆盖是它的顶点集 $V$ 的一个子集 $V' \subseteq V$ ，使得若 $(u,v)$ 是 $G$ 的一条边，则 $v \in V'$ 或 $u \in V'$ 。顶点覆盖 $V'$ 的大小是它所包含的顶点个数 $|V'|$ 。

VertexSet **approxVertexCover** ( Graph g )

```
{  cset =  $\emptyset$  ;  
    e1 = g.e ;  
    while (e1 !=  $\emptyset$ ) {  
        从e1中任取一条边(u,v) ;  
        cset = cset  $\cup$  {u,v} ;  
        从e1中删去与u和v相关联的所有边 ;  
    }  
    return c  
}
```

Cset用来存储顶点覆盖中的各顶点。初始为空，不断从边集e1中选取一边 $(u, v)$ ，将边的端点加入cset中，并将e1中已被u和v覆盖的边删去，直至cset已覆盖所有边。即e1为空。



图(a)~(e)说明了算法的运行过程及结果。(e)表示算法产生的近似最优顶点覆盖 $cset$ ,它由顶点 $b, c, d, e, f, g$ 所组成。(f)是图G的一个最小顶点覆盖,它只含有3个顶点: $b, d$ 和 $e$ 。

算法 $approxVertexCover$ 的近似比为2。

# MVC算法分析

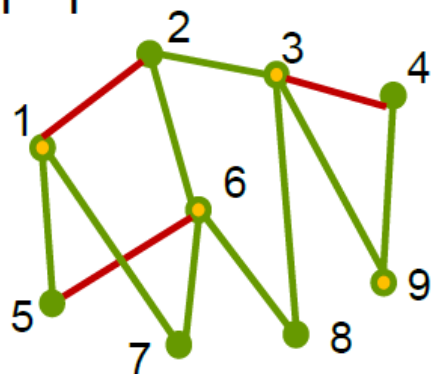
时间复杂度  $O(m)$ ,  $m=|E|$ .

近似比

上界：若算法选取  
 $k$  条边，那么  $|V'| = 2k$ ，  
 $V'$  由  $k$  条互不关联的边的端点组成。

为了覆盖这  $k$  条互不关联的边需要  $k$   
个顶点，从而  $\text{OPT}(I) \geq k$ . 于是

$$\frac{\text{MVC}(I)}{\text{OPT}(I)} \leq \frac{2k}{k} = 2$$



# MVC的近似比

下界

设图 $G$ 由  $k$  条互不关联的边构成.

显然

$$\text{MVC}(I) = 2k, \quad \text{OPT}(I) = k,$$

表明 MVC 的近似比不会小于2, 即上述MVC近似比已不可能再改进.

结论: MVC是2-近似算法.



$k=4$



# 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  $P \neq NP$  (或其他更强) 的假设下, 该问题近似算法的近似比的下界.

# 近似算法的分析

**运行时间：**一般应该是多项式时间

**近似比**

- (1) 估计上界：建立最优值与近似解的值之间的关系
- (2) 估计下界：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(最小化)达到或可以任意接近近似比的上界 (称作**紧实例**), 那么这个近似比已经是最好的.

# 小结

## 近似算法

适用于组合优化问题

对每个实例都能找到一个可行解

## 近似算法的评价指标及要求

时间复杂度——算法效率

要求：多项式时间

近似比——性能

要求：常数近似比

# 小结

## 近似比的分析方法

### (1) 上界

分析近似解与最优解的关系：找到对所有的实例都成立的量化不等式

### (2) 下界

找到一个规模可以任意大的紧实例，对这个实例算法的解与最优解的误差与上界尽可能接近

## 10.2 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

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

最小？

## 10.2.1 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10.1** 对多机调度问题的每一个有  $m$  台机器的实例  $I$ ,

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

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

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

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

# 证明

于是

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



# 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  $m$ .

算法方案: 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.

$$\mathbf{G}\text{-MPS}(I) = 2m-1.$$

最优方案: 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,  $\mathbf{OPT}(I)=m$ .

由于  $m$  可以任意大,  $\mathbf{G}\text{-MPS}$  是 2-近似算法.

## 10.2.2 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10.2** 对多机调度问题的每一个有  $m$  台机器的实例  $I$ ,

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

# 证明

证 设作业按处理时间从大到小排列为  $a_1, a_2, \dots, a_n$ , 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  $i \geq m+1$ ,  $\text{OPT}(I) \geq 2t(a_i)$ .

$$\begin{aligned}\text{DG} - \text{MPS}(I) &= t(M_j) \leq \frac{1}{m} \left( \sum_{k=1}^n t(a_k) - t(a_i) \right) + t(a_i) \\ &= \frac{1}{m} \sum_{k=1}^n t(a_k) + \left( 1 - \frac{1}{m} \right) t(a_i) \leq \text{OPT}(I) + \left( 1 - \frac{1}{m} \right) \cdot \frac{1}{2} \text{OPT}(I) \\ &= \left( \frac{3}{2} - \frac{1}{2m} \right) \text{OPT}(I)\end{aligned}$$

# 货郎问题及其子问题

**TSO:** 给定 $n$ 个城市, 任两个城市  $i$  和  $j$  之间距离为  $d(i,j)$ . 求一条遍历每个城市恰好一次的最短的旅行路线.

**子问题:** 满足三角不等式的货郎问题  
对任意三个城市顶点  $i, j, k$  有

$$d(i,j) + d(j,k) \geq d(i,k)$$

三角形两边之和大于等于第三边

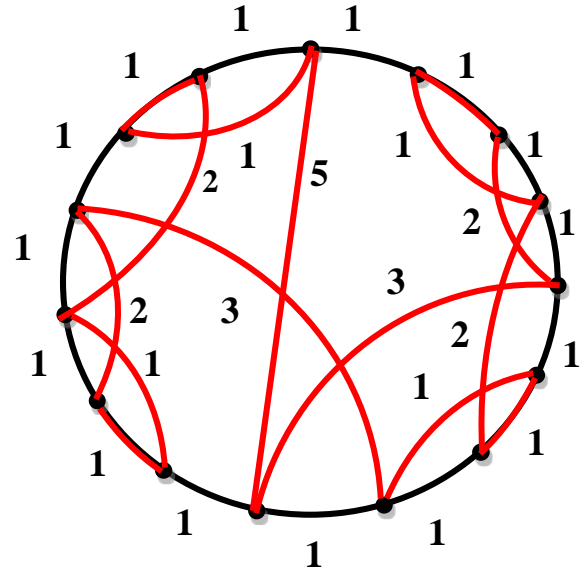
## 10.3 货郎问题

本节考虑满足三角不等式的货郎问题

### 10.3.1 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.

右图: 一个NN性能很坏的例子.



$$NN(I)=27, \text{ OPT}(I)=15$$

# 最邻近法的性能

**定理** 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

$$\text{NN}(I) \leq \frac{1}{2} (\lceil \log_2 n \rceil + 1) \text{OPT}(I).$$

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

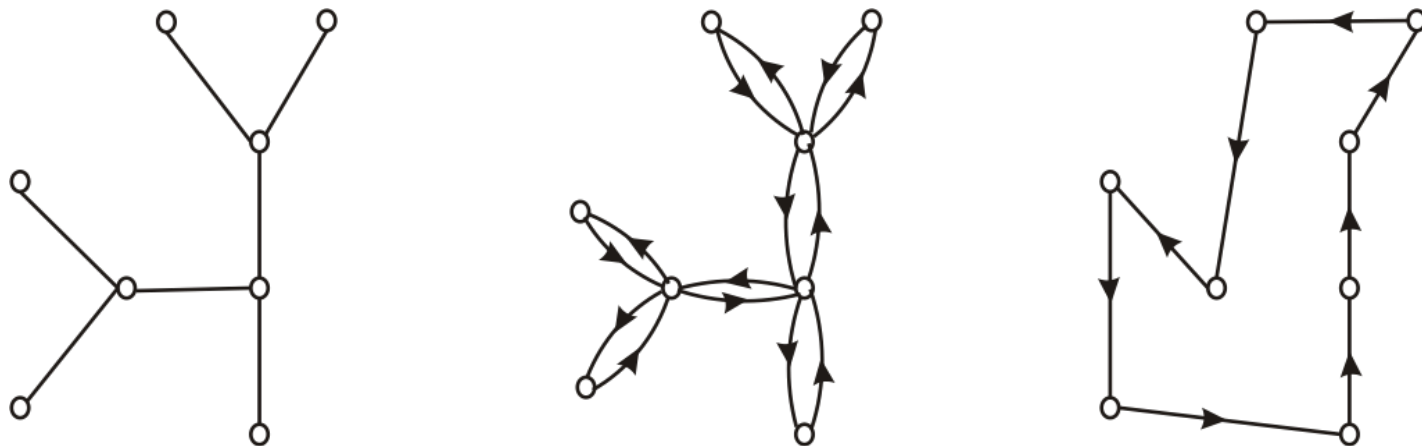
$$\text{NN}(I) > \frac{1}{3} \left( \log_2(n+1) + \frac{4}{3} \right) \text{OPT}(I).$$

**结论** 近似比不是常数, 不能实际应用.

## 10.3.2 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

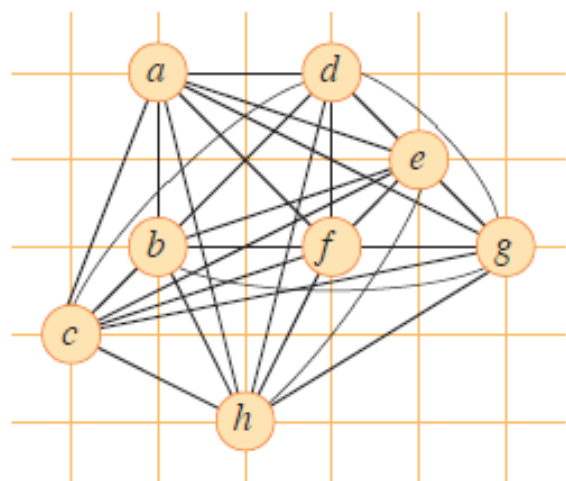


求最小生成树和欧拉回路都可以在多项式时间内完成, 故算法是多项式时间的.

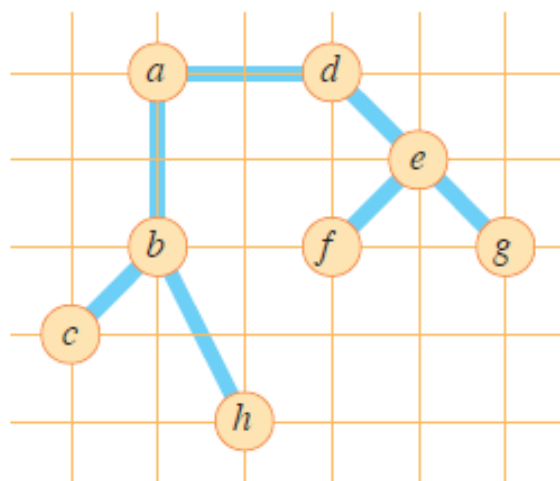
### APPROX-TSP-TOUR( $G, c$ )

- 1 select a vertex  $r \in G.V$  to be a “root” vertex
- 2 compute a minimum spanning tree  $T$  for  $G$  from root  $r$   
using MST-PRIM( $G, c, r$ )
- 3 let  $H$  be a list of vertices, ordered according to when they are first visited  
in a preorder tree walk of  $T$
- 4 **return** the hamiltonian cycle  $H$

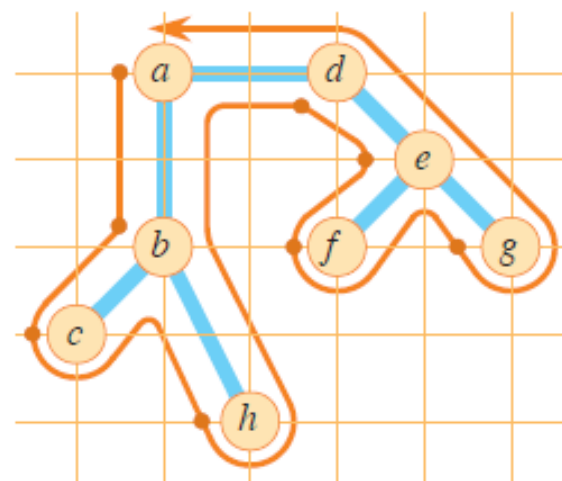




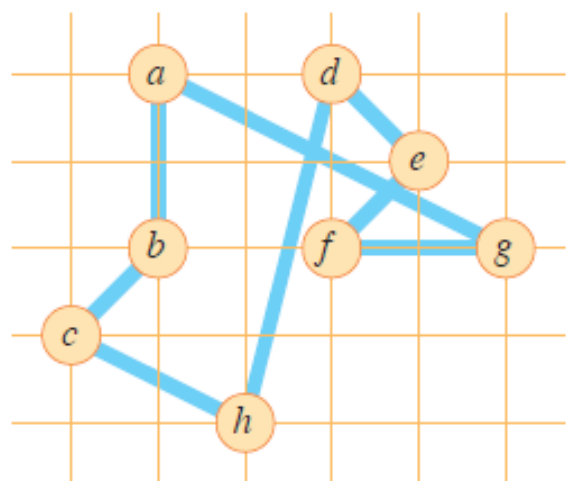
(a)



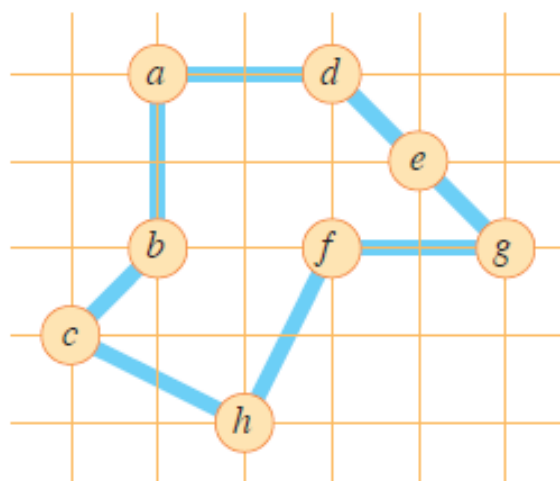
(b)



(c)



(d)



(e)

# 最小生成树法的性能

**定理10.4** 对货郎问题的所有满足三角不等式的实例 $I$ ,

$$\text{MST}(I) < 2\text{OPT}(I).$$

证 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  $\text{OPT}(I)$ . 沿  $T$  走两遍的长小于  $2\text{OPT}(I)$ . 因为满足三角不等式, 抄近路不会增加长度, 故

$$\text{MST}(I) < 2\text{OPT}(I).$$

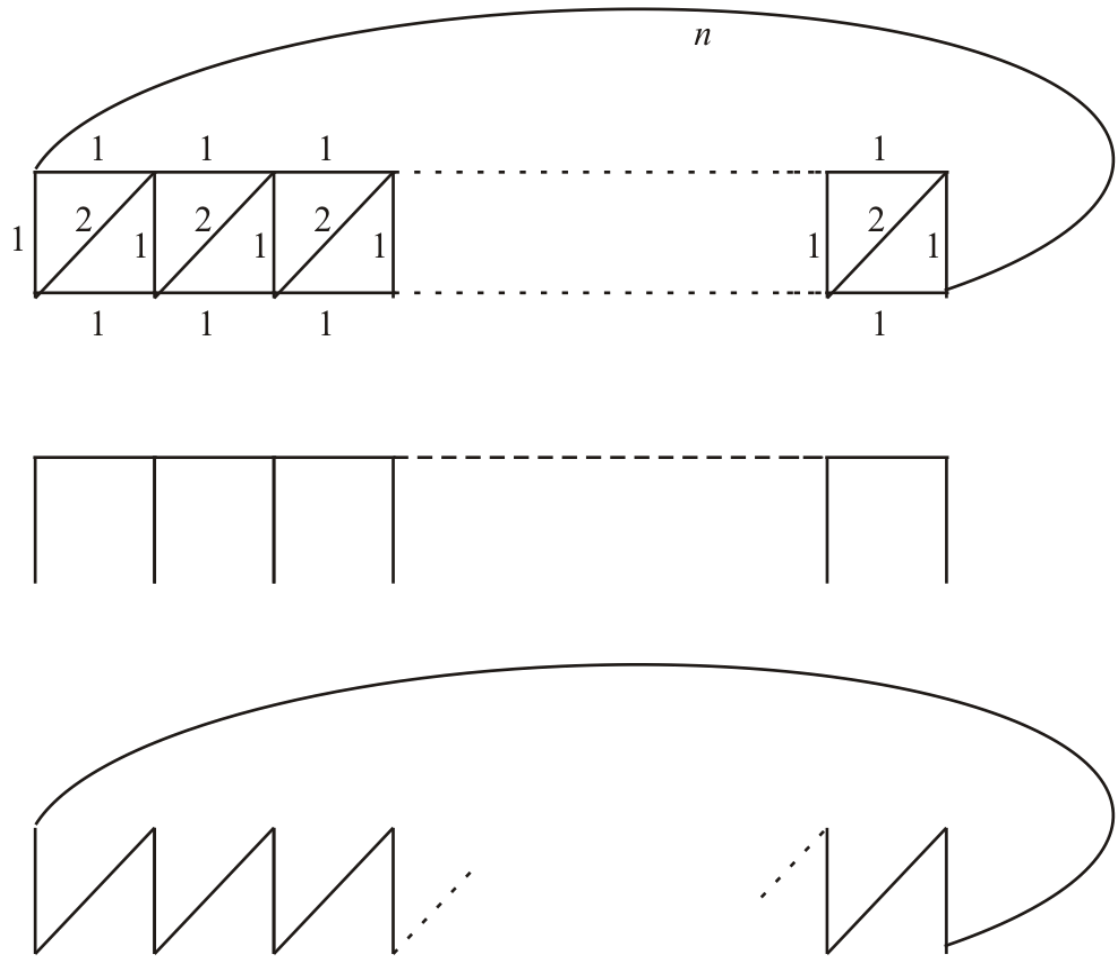
**MST 是 2-近似算法.**

# 紧实例

$$\text{OPT}(I) = 2n$$

$$\text{MST}(I) = 4n - 2$$

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



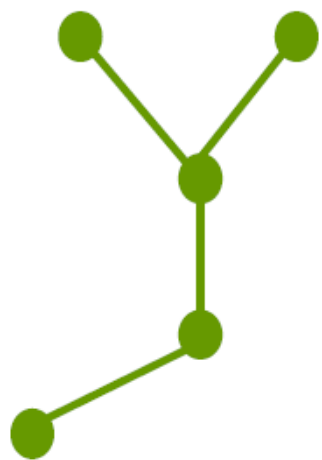
## 10.3.3 最小权匹配法

### 最小权匹配法MM:

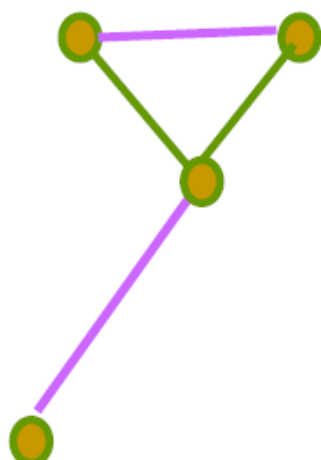
首先求图的一棵最小生成树  $T$ . 记  $T$  的所有奇度顶点在原图中的导出子图为  $H$ ,  $H$  有偶数个顶点, 求  $H$  的最小匹配  $M$ . 把  $M$  加入  $T$  得到一个欧拉图, 求这个欧拉图的欧拉回路; 最后, 沿着这条欧拉回路, 跳过已走过的顶点, 抄近路得到一条哈密顿回路.

求任意图最小权匹配的算法是多项式时间的, 因此 MM 是多项式时间的.

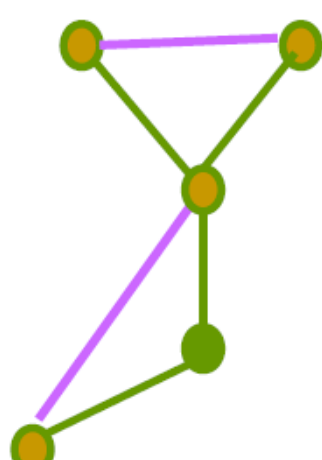
# 一个例子



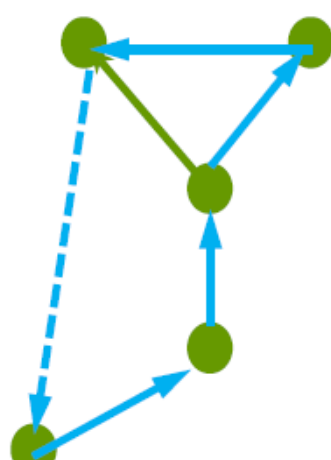
生成树



$H$  最小匹配



欧拉回路



哈密顿回路

# MM算法的性能

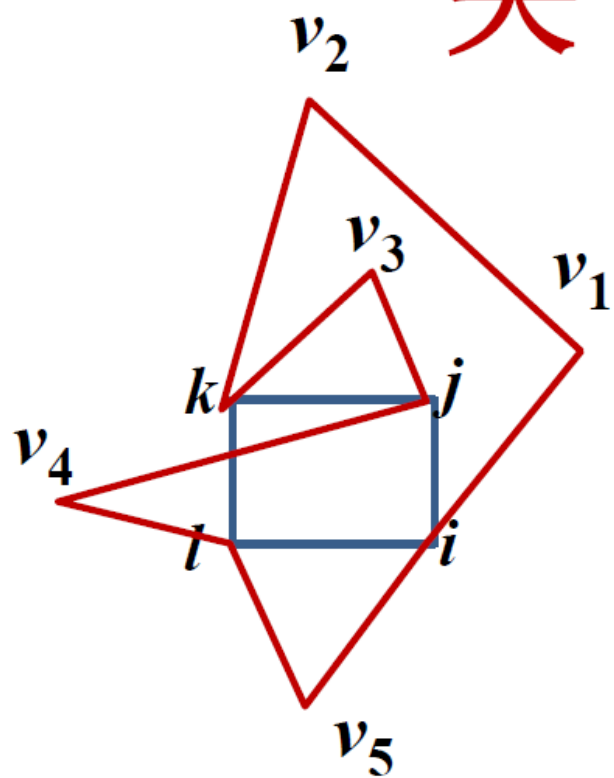
**定理** 对货郎问题的所有满足三角不等式的实例  $I$ ,  $MM(I) < (3/2)OPT(I)$ .

证 (1)  $C$ 是导出子图  $H$  中最短哈密顿回路, 长度为  $length(C)$ .

根据三角不等式性质有

$$Length(C) \leq OPT(I).$$

# 关于(1)的说明



$H$ 中最短哈密顿回路 $C$ :  
 $i-j-k-l-i$

$G$ 中最短哈密顿回路:  
 $i-v_1-v_2-k-v_3-j-v_4-l-v_5-i$

$i-j-k-l-i$  的长度  
 $\leq i-k-j-l-i$  的长度 ( $C$ 的最短性)  
 $\leq i-v_1-v_2-k-v_3-j-v_4-l-v_5-i$  长度(三角不等式)

# 定理证明

证 (1)  $C$  是导出子图  $H$  中最短哈密顿回路, 长度为  $\text{length}(C)$ .

根据三角不等式性质有

$$\text{Length}(C) \leq \text{OPT}(I).$$

(2) 沿  $C$  间隔取边, 得到  $H$  的匹配,

$$\text{length}(M) \leq \text{Length}(C)/2 \leq \text{OPT}(I)/2$$

(3) 抄近路不增加长度

$$\begin{aligned} \text{MM}(I) &\leq W(T) + \text{length}(M) \\ &< \text{OPT}(I) + \text{OPT}(I)/2 \\ &= (3/2)\text{OPT}(I) \end{aligned}$$



# 货郎问题的难度

**定理** 货郎问题(不要求满足三角不等式)是不可近似的, 除非  $P = NP$ .

证明思路:

假设  $A$  是解货郎问题的多项式时间的近似算法, 其近似比  $r \leq K$ ,  $K$  是常数.

设计一个调用算法  $A$  求解 哈密顿回路 (HC) 的算法, 该算法是多项式时间的, 从而 HC 属于  $P$ , 于是证明了  $P=NP$ .

# 算法设计

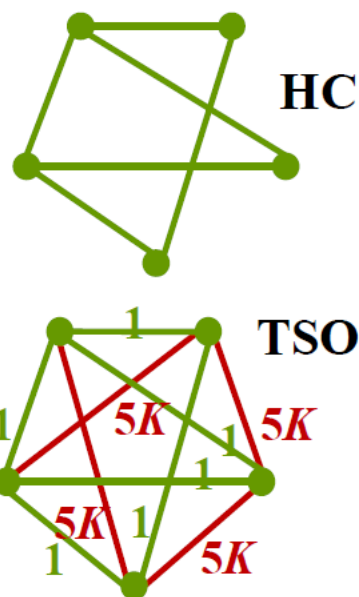
1. 任给HC的实例  $G = \langle V, E \rangle$ ,  
如下构造货郎问题的实例  $I_G$ :  
城市集  $V$ ,  $\forall u, v \in V$ ,  
若  $(u, v) \in E$ , 则  $d(u, v) = 1$ ;  
否则  $d(u, v) = Kn$

2. 对  $I_G$  调用算法 A 计算  
最短路线长  $A(I_G)$

3. if  $A(I_G) \leq Kn$  then return “Yes”

4. else return “No”

**时间:** 构造  $I_G$  可在  $O(n^2)$  时间内完成.  
 $|I_G| = O(n^2)$ , A 是多项式时间的



# 定理证明

若 $G$  有哈密顿回路, 则

$$\text{OPT}(I_G) = n$$

$$\text{A}(I_G) \leq r \text{ OPT}(I_G) \leq Kn$$

若 $G$  没有哈密顿回路, 则

$$\text{OPT}(I_G) > Kn,$$

$$\text{A}(I_G) \geq \text{OPT}(I_G) > Kn$$

**结论:**  $G$  有哈密顿回路  $\Leftrightarrow \text{A}(I_G) \leq Kn$

# 小结

- 满足三角不等式的货郎问题
  - (1) 最邻近法NN: 近似比不是常数
  - (2) 最小生成树法MST: 2-近似算法
  - (3) 最小权匹配法MM: 1.5-近似算法
- 一般货郎问题: 若 $P \neq NP$ , 是不可近似计算的, 即不存在常数近似比的算法.

# 10.4 背包问题

## 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  $v_i$ ,  $1 \leq i \leq n$ , 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  $S^* \subseteq \{1, 2, \dots, n\}$  使得

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

## 10.4.1 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

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

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  $v_k = \max\{v_i \mid i=1, 2, \dots, n\}$ . 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10.7** 对 0-1背包问题的任何实例  $I$ , 有  
$$\text{OPT}(I) < 2\text{G-KK}(I).$$

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

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

**G-KK** 是 2-近似算法.

## 10.4.2 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
$$v_1/w_1 \geq v_2/w_2 \geq \dots \geq v_n/w_n.$$
3. 对每一个  $t = 1, 2, \dots, m$  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  $\text{PTAS}_\varepsilon$ .



# 实例: $n=10, \varepsilon=0.2, m=5$

$t=1$ : 尝试 10次, 装入背包物品集分别为  $\{1\}, \{2\}, \dots, \{10\}$ , 再使用 G-KK算法

$t=2$ : 尝试 45次, 装入物品集是  $\{1,2\}, \{1,3\}, \dots, \{9,10\}$ , 再使用G-KK.  
...

$t=5$ : 尝试  $C_{10}^5$  次, 装入物品集为  $\{1,2,3,4,5\}, \{1,2,3,4,6\}, \{1,2,3,4,7\}, \dots, \{6,7,8,9,10\}$ , 再用G-KK算法.

总计运行G-KK算法次数至多为

$$C_{10}^1 + C_{10}^2 + \dots + C_{10}^5$$

# PTAS的性能

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

$$\text{OPT}(I) < (1+\varepsilon) \text{PTAS}_\varepsilon(I),$$

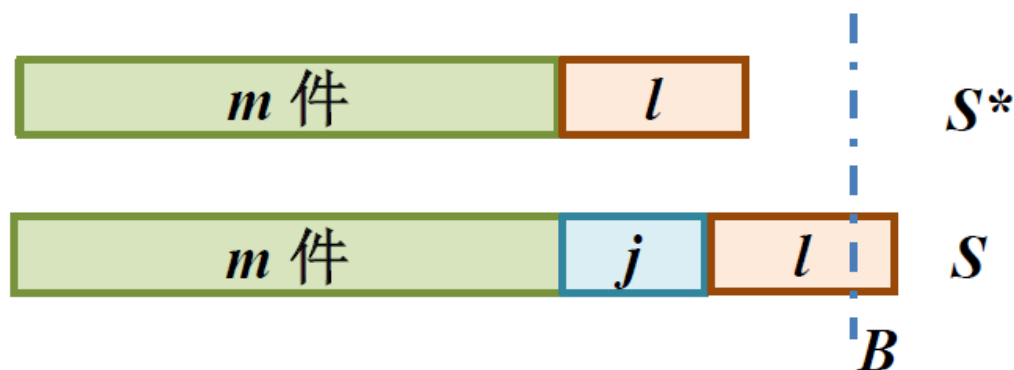
且  $\text{PTAS}_\varepsilon$  的时间复杂度为  $O(n^{1/\varepsilon+2})$ .

**证** 设最优解为  $S^*$ . 若  $|S^*| \leq m$ , 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  $S$ .

# 一个事实

设  $l$  是  $S^*$  中第一件不在  $S$  中的物品  
在此之前 **G-KK** 装入不属于  $S^*$  的物品  $j$  ( $j < l$ , 没有  $j$  占用, 贪心法将装入  $l$ )



$$\sum_{i \in S} w_i + w_l > B$$

$$\text{OPT}(I) < \sum_{i \in S} v_i + v_l$$

# 证明

$$\text{OPT}(I) < \sum_{i \in S} v_i + v_l$$

$S$  包括  $S^*$  中  $m$  件价值最大的物品,  
它们的价值都不小于  $v_l$ ,

蓝框  
公式

$$v_l \leq \sum_{i \in S} v_i / m \quad (*)$$

$$\begin{aligned} \text{OPT}(I) &< \sum_{i \in S} v_i + \underline{v_l} \\ &\leq \sum_{i \in S} v_i + \underline{\sum_{i \in S} v_i / m} \\ &\leq (1 + 1/m) \text{PTAS}_\varepsilon(I) \\ &\leq (1 + \varepsilon) \text{PTAS}_\varepsilon(I) \end{aligned}$$

公式  
\*

# 多项式时间近似方案

时间复杂度. 从  $n$  件物品中取  $t$  件 ( $t = 1, 2, \dots, m$ ), 所有可能取法的个数为

$$c_n^1 + c_n^2 + \dots + c_n^m \leq m \cdot \frac{n^m}{m!} \leq n^m.$$

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  $O(n^{m+1}) = O(n^{\frac{1}{\varepsilon}+2})$ .

多项式时间近似方案 以  $\varepsilon > 0$  和问题的实例  $I$  作为输入, 对每一个固定的  $\varepsilon > 0$ , 算法是  $1+\varepsilon$ -近似的.

# 小结

- 0-1背包问题定义
- 贪心算法 **G-KK**  
设计思想  
性能: 2-近似算法
- 多项式时间近似方案 **PTAS**  
带误差参数 $\varepsilon$ 的一组算法  
调用 **G-KK** 算法作为子过程  
时间:  $O(n^{1/\varepsilon+2})$   
性能:  $1+\varepsilon$ -近似算法

# 背包问题的对偶问题

$n$ 个物品，物品  $i$  的重量  $w_i$ ，价值  $v_i$ ， $i=1,2,\dots,n$ ，背包重量限制  $B$ ，如何选择物品，使得背包重量不超过  $B$  且价值最大？

$n$ 个物品，物品  $i$  的重量  $w_i$ ，价值  $v_i$ ， $i=1,2,\dots,n$ ，背包预期价值为  $V$ ，如何选择物品，使得背包达到价值  $V$  且重量最轻？

最大化  最小化

优化目标  约束条件

## 10.4.3 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，至少要装入物品重量。

$$G_k(d) = \min\{\sum_{i=1}^k w_i x_i \mid \sum_{i=1}^k v_i x_i \geq d, x_i = 0 \text{ 或 } 1, 1 \leq i \leq k\}$$

$$0 \leq k \leq n, 0 \leq d \leq D, D = \sum_{i=1}^n v_i, \text{ 约定: } \min \emptyset = +\infty.$$

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



# 算法设计与分析



算法的递推公式

$$G_0(d) = \begin{cases} 0, & \text{若 } d = 0, \\ +\infty, & \text{若 } d > 0, \end{cases}$$
$$G_{k+1}(d) = \begin{cases} \min\{ G_k(d), w_{k+1} \}, & \text{若 } d \leq v_{k+1}, \\ \min\{ G_k(d), G_k(d - v_{k+1}) + w_{k+1} \}, & \text{若 } d > v_{k+1}, \end{cases}$$
$$0 \leq k \leq n-1, \quad 0 \leq d \leq D.$$

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

# 算法 FPTAS

**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  $v_i' = \lceil v_i/b \rceil$ ,  $1 \leq i \leq n$ . 把所有  $v_i$  换成  $v_i'$ , 记新的实例为  $I'$ . 
3. 对  $I'$  应用算法  $A$  得到解  $S$ , 把  $S$  取作实例  $I$  的解.

## 实例

$$b = \max \left\{ \left\lfloor \frac{v_{\max}}{(1 + 1/\varepsilon)n} \right\rfloor, 1 \right\}, v_i' = \lceil v_i/b \rceil$$

$$v_1=1, v_2=5, v_3=6, v_4=9, v_5=10,$$

$$v_6=12, v_7=13, v_8=19, v_9=1000$$

- $\varepsilon = \underline{0.1}$ ,  $v_{\max} = 1000$ ,  $n = 9$ ,  $b = 10$ ,  $v_i' = \lceil v_i/10 \rceil$

$$v_1' = v_2' = v_3' = v_4' = v_5' = 1, v_6' = v_7' = v_8' = 2, v_9' = 100$$

- $\varepsilon = \underline{0.01}$ ,  $v_{\max} = 1000$ ,  $n = 9$ ,  $b = 1$ ,  $v_i' = \lceil v_i/1 \rceil$

$$v_i' = v_i, i = 1, 2, \dots, 9$$

更多物品价值被近似成同一个值，误差可能更大.

# 设计思想

- 将0-1背包问题转换成对偶问题，以物品项数、价值作为参数，用动态规划自底上向上迭代计算。
- 将物品价值等比例缩小，相当于每组取一个近似价值作为代表，以加快对偶问题动态规划算法的运行。

# FPTAS算法性能

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

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

并且 FPTAS 的时间复杂度为

$$O(n^3(1+1/\varepsilon)).$$

**对比**

	时间	近似比
PTAS	$O(n^{1/\varepsilon+2})$	$1+\varepsilon$
FPTAS	$O(n^3(1+1/\varepsilon))$	$1+\varepsilon$

**定理10.9** 对每一个  $\varepsilon > 0$  和 0-1背包问题的实例  $I$ ,

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

并且 FPTAS 的时间复杂度为  $O(n^3(1+1/\varepsilon))$ .

证 由于  $(v_i'-1)b < v_i \leq v_i'b$ , 对任意的  $T \subseteq \{1, 2, \dots, n\}$ ,

$$0 \leq b \sum_{i \in T} v_i' - \sum_{i \in T} v_i < b |T| \leq bn.$$

设  $I$  的最优解为  $S^*$ , 注意到  $S$  是  $I'$  的最优解, 故有

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

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  $v_{\max} \leq \text{OPT}(I)$ , 得

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

得  $\text{OPT}(I) < (1+\varepsilon) \text{FPTAS}(I)$ .

时间主要花是在  $A$  对  $I'$  的运算, 其时间复杂度为  $O(n^2 v_{\max} / b) = O(n^3(1+1/\varepsilon))$ .

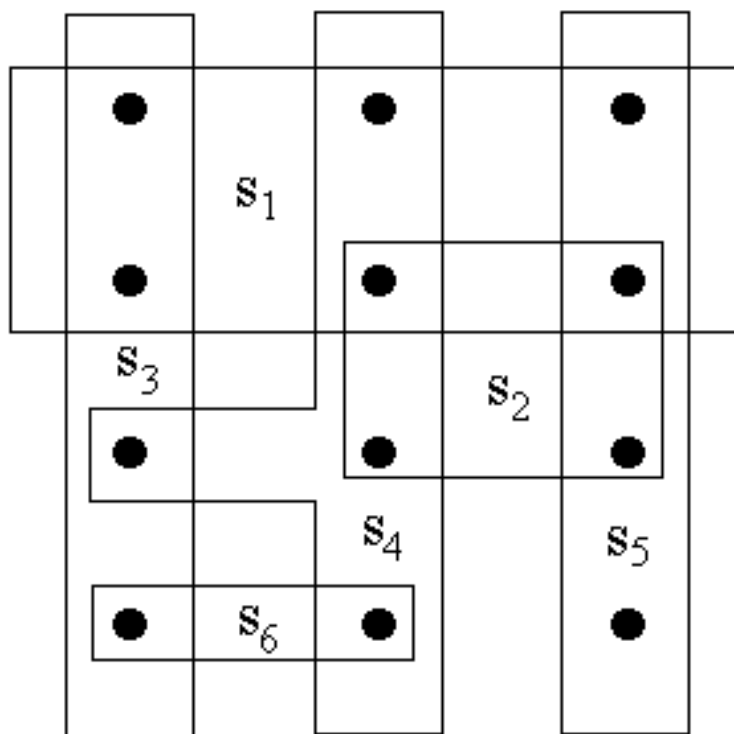
# 集合覆盖问题的近似算法

问题描述：给定一个完全无向图 $G=(V,E)$ ，其每一边 $(u,v) \in E$ 有一非负整数费用 $c(u,v)$ 。要找出 $G$ 的最小费用哈密顿回路。

集合覆盖问题的一个实例 $\langle X, F \rangle$ 由一个有限集 $X$ 及 $X$ 的一个子集族 $F$ 组成。子集族 $F$ 覆盖了有限集 $X$ 。也就是说 $X$ 中每一元素至少属于 $F$ 中的一个子集，即 $X = \bigcup_{S \in F} S$ 。对于 $F$ 中的一个子集 $C \subseteq F$ ，若 $C$ 中的 $X$ 的子集覆盖了 $X$ ，即 $X = \bigcup_{S \in C} S$ ，则称 $C$ 覆盖了 $X$ 。集合覆盖问题就是要找出 $F$ 中覆盖 $X$ 的最小子集 $C^*$ ，使得

$$|C^*| = \min\{|C| \mid C \subseteq F \text{ 且 } C \text{ 覆盖 } X\}$$

## 集合覆盖问题举例：



用12个黑点表示集合 $X$ 。

$F = \{S_1, S_2, S_3, S_4, S_5, S_6, \}$ ，如图所示。

容易看出，对于这个例子，最小集合覆盖为：

$C = \{S_3, S_4, S_5, \}$ 。



# 集合覆盖问题的近似算法

## 集合覆盖问题近似算法——贪心算法

Set **greedySetCover** (X,F)

```
{  
    U=X ;  
    C=∅ ;  
    while (U !=∅) {  
        选择F中使 $|S \cap U|$ 最大的子集S ;  
        U=U-S ;  
        C=C ∪ {S} ;  
    }  
    return C ;  
}
```

算法的循环体最多执行 $\min\{|X|, |F|\}$ 次。而循环体内的计算显然可在 $O(|X| |F|)$ 时间内完成。因此，算法的计算时间为 $O(|X| |F| \min\{|X|, |F|\})$ 。由此即知，该算法是一个多项式时间算法。

# 子集和问题的近似算法

问题描述：设子集和问题的一个实例为  $\langle S, t \rangle$ 。其中， $S = \{x_1, x_2, \dots, x_n\}$  是一个正整数的集合， $t$  是一个正整数。子集和问题判定是否存在  $S$  的一个子集  $S_1$ ，使得  $\sum_{x \in S_1} x = t$ 。

# 子集和问题的指数时间算法

```
int exactSubsetSum (S,t)
{
    int n=|S|;
    L[0]={0};
    for (int i=1; i<=n; i++) {
        L[i]=mergeLists(L[i-1],L[i-1]+S[i]);
        删去L[i]中超过t的元素;
    }
    return max(L[n]);
}
```

算法以集合 $S=\{x_1, x_2, \dots, x_n\}$ 和目标值 $t$ 作为输入。算法中用到将2个有序表 $L1$ 和 $L2$ 合并成为一个新的有序表的算法  
( $L1, L2$ )。

# 子集和问题的完全多项式 时间近似格式

基于算法exactSubsetSum，通过对表 $L[i]$ 作适当的修整建立一个子集和问题的**完全多项式时间近似格式**。

在对表 $L[i]$ 进行修整时，用到一个修整参数 $\delta$ ， $0 < \delta < 1$ 。用参数 $\delta$ 修整一个表 $L$ 是指从 $L$ 中删去尽可能多的元素，使得每一个从 $L$ 中删去的元素 $y$ ，都有一个修整后的表 $L_1$ 中的元素 $z$ 满足 $(1-\delta)y \leq z \leq y$ 。可以将 $z$ 看作是被删去元素 $y$ 在修整后的新表 $L_1$ 中的代表。

**举例：**若 $\delta=0.1$ ，且 $L = \langle 10, 11, 12, 15, 20, 21, 22, 23, 24, 29 \rangle$ ，则用 $\delta$ 对 $L$ 进行修整后得到 $L_1 = \langle 10, 12, 15, 20, 23, 29 \rangle$ 。其中被删去的数11由10来代表，21和22由20来代表，24由23来代表。

## 对有序表L修整算法

```
List trim(L,  $\delta$ )
{
  int m = |L|;
  L1 =  $\langle L[1] \rangle$  ;
  int last = L[1];
  for (int i = 2; i <= m; i++) {
    if (last < (1 -  $\delta$ ) * L[i]) {
      将L[i]加入表L1的尾部;
      last = L[i];
    }
  }
  return L1;
}
```

## 子集和问题近似格式

```
int approxSubsetSum(S, t,  $\epsilon$ )
{
  n = |S|;
  L[0] =  $\langle 0 \rangle$  ;
  for (int i = 1; i <= n; i++) {
    L[i] = Merge-Lists(L[i-1],
                      L[i-1] + S[i]);
    L[i] = Trim(L[i],  $\epsilon/n$ );
    删去L[i]中超过t的元素;
  }
  return max(L[n]);
}
```

# 小结

## 近似算法

适用于组合优化问题

对每个实例能够找到一个可行解

通常运行在多项式时间

## 近似算法的设计技术

贪心法

调用动态规划作为某些子过程

.....

# 小结（续1）

## 近似算法分析方法

- 近似比  $r$   
分析方法  
上界：近似解与最优解的不等式  
对所有的实例都成立  
下界：寻找一个紧实例  
要求：常数近似比
- 时间复杂度  
要求：多项式时间

# 小结（续2）

## 典型的近似算法

- 最小顶点覆盖（2-近似算法）
- 多机调度
  - 贪心法G-MPS（2-近似算法）
  - 改进贪心法DG-MPS（1.5近似算法）
- 货郎问题(满足三角不等式):
  - 最邻近法NN：不是常数近似比
  - 最小生成树法MST：2-近似算法
  - 最小匹配法MM：1.5-近似算法



# 小结（续3）

- 0-1 背包问题

贪心法G-KK: 2-近似算法

PTAS:  $1+\varepsilon$ -近似算法,  $O(n^{1/\varepsilon+2})$

FPTAS:  $1+\varepsilon$ -近似算法  $O(n^3(1+1/\varepsilon))$

## 不可近似性研究

对于货郎问题, 若 $N \neq NP$ , 则货郎问题不存在常数近似比的近似算法