HIT
CS&E

第八章

近似算法的设计分析原理

程思瑶
计算机科学与技术学院

HIT
CS&E

受过教育的人的一个标志就是，

他满足于接受事物自身所具有的精确程度，

而当仅能得到近似值时，他不去追求无法获得的精确值。

————亚里士多德

参考书

Vijay V. Vazirani, *Approximation Algorithms*, Springer- Verlag, 2001.

David P. Williamson, David B. Shmoys, *The Design of Approximation Algorithms*, Cambridge University Press, 2011.

Dorit S. Hochbaum, *Approximation Algorithms for NP-Hard Problems*, PWS Publishing Company, 2011.

提要

8.1 近似算法概述

8.2 基于组合优化的近似算法

8.3 基于贪心策略的近似算法

8.4 基于局部优化的近似算法

8.5 基于动态规划的近似算法

8.6 基于线性规划的近似算法

8.1 近似算法概述

- 近似算法的基本概念
- 近似算法的性能分析

近似算法的基本概念

- 近似算法的基本思想
  - 很多实际应用中问题都是NP-完全问题
  - NP-完全问题的多项式算法是难以得到的
  - 求解NP-完全问题的方法:
    - 如果问题的输入很小，可以使用指数级算法圆满地解决该问题
    - 否则使用多项式算法求解问题的近似优化解
  - 什么是近似算法
    - 能够给出一个问题的近似解的算法
    - 常用来解决优化问题

近似算法的性能分析

- 近似算法的时间复杂性
  - 分析目标和方法与传统算法相同
- 近似算法解的近似度
  - 本节讨论的问题是优化问题
    - 问题的每一个可能的解都具有一个正的代价
    - 问题的优化解可能具有最大或最小代价
    - 我们希望寻找问题的一个近似优化解
  - 我们需要分析近似解代价与优化解代价的差距
    - Ratio Bound
    - 相对误差
    - $(1\pm\varepsilon)$-近似

- **Ratio Bound**

定义1(Ratio Bound) 设A是一个优化问题的近似算法, A具有ratio bound $p(n)$, 如果

$$\max\left\{\frac{C}{C^*},\frac{C^*}{C}\right\}\leq p(n)$$

其中n是输入大小, C是A产生的近似解的代价, C^*是优化解的代价.

- 如果问题是最大化问题, max{C/C^*, C^*/C}=C^*/C
- 如果问题是最小化问题, max{C/C^*, C^*/C}=C/C^*
- 由于$C/C^*<1$当且仅当$C^*/C>1$, Ratio Bound不会小于1
- Ratio Bound越大, 近似解越坏

- 相对误差

  定义2(相对误差) 对于任意输入, 近似算法的相对误差定义为$|C-C^*|/C^*$, 其中C是近似解的代价, C^*是优化解的代价.

  定义3(相对误差界) 一个近似算法的相对误差界为$\varepsilon(n)$, 如果$|C-C^*|/C^* \leq \varepsilon(n)$.

结论1. $\varepsilon(n) \le p(n)-1$.

证. 对于最小化问题

$$\varepsilon(n)=|C-C^*|/C^*=(C-C^*)/C^*=C/C^* -1=p(n)-1.$$

对于最大化问题

$$\varepsilon(n)=|C-C^*|/C^*=(C^*-C)/C^*= (C^*/C -1)/(C^*/C)$$
$$= (p(n)-1)/p(n) \le p(n)-1.$$

- 对于某些问题，$\varepsilon(n)$和$p(n)$独立于n，用p和ε表示之.
- 某些NP-完全问题的近似算法满足：当运行时间增加时，Ratio Bound和相对误差将减少.
- 结论1表示，只要求出了Ratio Bound就求出了$\varepsilon(n)$

- 近似模式

定义4 (近似模式) 一个优化问题的近似模式是一个以问题实例I和$\varepsilon>0$为输入的算法族. 对于任意固定ε, 近似模式是一个$(1\pm\varepsilon)$-近似算法.

定义5 一个近似模式$A(I, \varepsilon)$称为一个多项式时间近似模式(*PTAS*), 如果对于任意$\varepsilon>0$, $A(I, \varepsilon)$的运行时间是关于输入实例大小$|I|$的多项式.

定义 6 一个近似模式称为完全多项式时间近似模式(*FPAS*,*FPTAS*), 如果它的运行时间是关于$1/\varepsilon$和输入实例大小n的多项式.

8.2 基于组合优化的近似算法

- 顶点覆盖问题
- 装箱问题
- TSP问题

8.2.1 顶点覆盖问题

- 问题的定义
- 近似算法的设计
- 算法的性能分析

问题的定义

输入：无向图$G=(V, E)$

输出：$C\subseteq V$, 满足

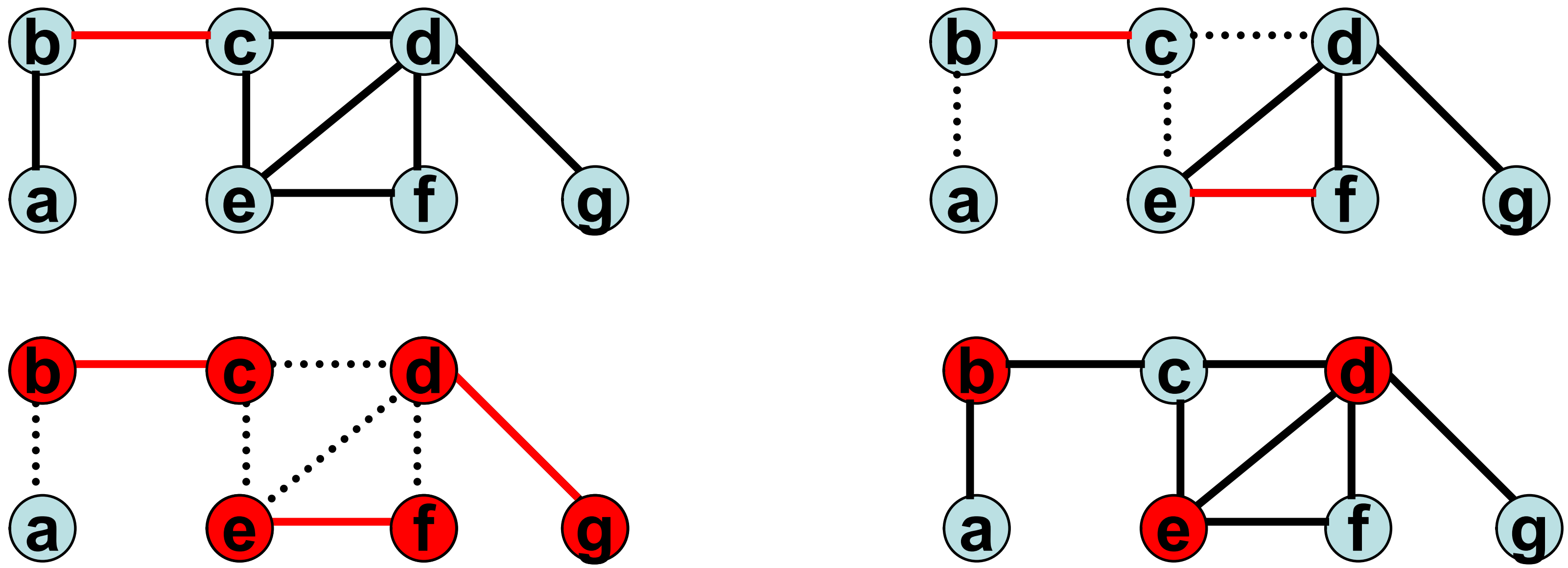
(1). $\forall(u, v)\in E$, $u\in C$、$v\in C$或$\{u,v\}\subseteq C$

(2). C是满足条件(1)的最小集合。

理论上已经证明优化结点覆盖问题是NP-完全问题.

- 算法的基本思想

每次任取一条边，结点加入C，删除相关边，重复该过程



算法解：{b, c, e, f, d, g}

优化解：{b, e, d}

- 算法

APPROX-Vertex-Cover (G)

1. $C=0$
2. $E'=E[G]$;
3. **While** $E'\neq 0$ **DO**
4. 任取$(u, v)\in E'$;
5. $C=C\cup\{u, v\}$;
6. 从E'中删除所有与u或v相连的边;
7. **Return** C

算法的性能分析

- 时间复杂性

  $T(G)=O(|E|)$

APPROX-Vertex-Cover (G)

1. $C=0$
2. $E'=E[G]$;
3. **While** $E'\neq 0$ **DO**
4. 任取$(u, v)\in E'$;
5. $C=C\cup\{u, v\}$;
6. 从E'中删除所有与u或v相连的边;
7. **Return** C

- Ratio Bound

定理. Approx-Vertex-Cover的Ratio Bound为2.

证. 令$A=\{(u, v) \mid (u, v)$是算法第4步选中的边$\}$.

若$(u,v)\in A$, 则与(u, v)邻接的边皆从E'中删除.

于是, A中无相邻接边.

第5步的每次运行增加两个结点到C, $|C|=2|A|$.

设C^*是优化解, C^*必须覆盖A.

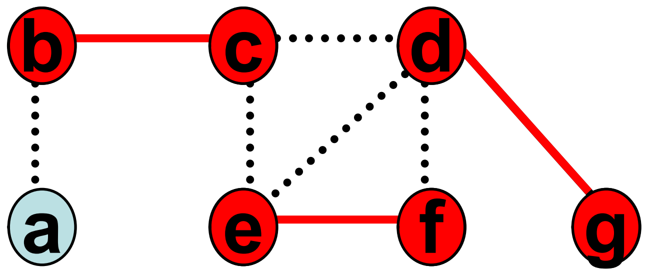
由于A中无邻接边,

C^*至少包含A中每条边的一个结点. 于是,

$|A|\leq|C^*|$,

$|C|=2|A|\leq 2|C^*|$,

即 $|C|/|C^*|\leq 2$.



8.2.2 装箱问题

- 问题的定义
- 近似算法的设计
- 算法的性能分析

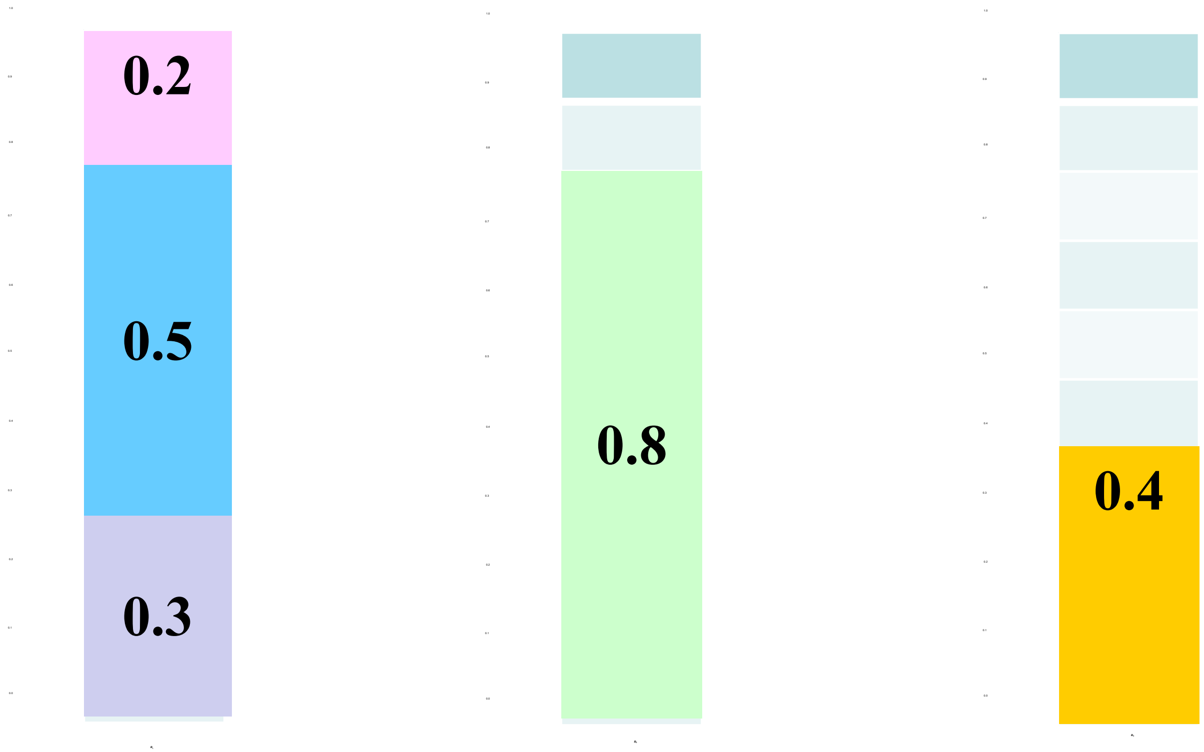
问题的定义

- 输入
  1. 体积依次为$a_1,\dots,a_n \in (0,1]$的n个物品
  2. 无穷个体积为1的箱子
- 输出
  物品的一个装箱方案，使得使用的箱子数量最少

- Bin-Packing是一个著名的NP完全问题.
- 实例：将n种奖券印刷在一些张具有标准尺寸的纸张上，每张奖券是一个物品，纸张是箱子，归一化处理之后变为Bin-Packing问题。

- **算法的基本思想(First-Fit)**

0.3, 0.5, 0.8. 0.2, 0.4



最优解也需要3个箱子

HIT

- 算法

First-Fit (G)

1. $k \leftarrow 0, B_1 \leftarrow \varnothing$
2. For i=1 to n Do
3. 从$B_1,\ldots,B_k$中选择能容纳a_i的第一个箱子B_j
4. 如果B_j存在，则$B_j \leftarrow B_j \cup \{a_i\}$
5. 否则， $k \leftarrow k+1$，$B_k \leftarrow \{a_i\}$
6. 输出$B_1,\ldots,B_k$

时间复杂性$O(n^2)$

定理. First-Fit的Ratio Bound为2.

证. 令k^*表示最优解代价(所用箱子的个数).

k 表示近似解代价.

$k^* \geq \sum_{1\leq i \leq n} a_i$

如果算法使用了k个箱子，则至少有k-1个箱子超过半满，即有：$\sum_{1\leq i \leq n} a_i > (k-1)\times 1/2$

$(k-1)/2 < k^*$

于是，$(k-1) < 2k^*$， $k \leq 2k^*$.

8.2.3 旅行商问题

- 问题的定义
- 近似算法设计
- 算法的性能分析

问题的定义

- 输入

  完全无向图$G=(V,E)$;

  代价函数$C: E \rightarrow$非负整数集合

  C满足三角不等式:

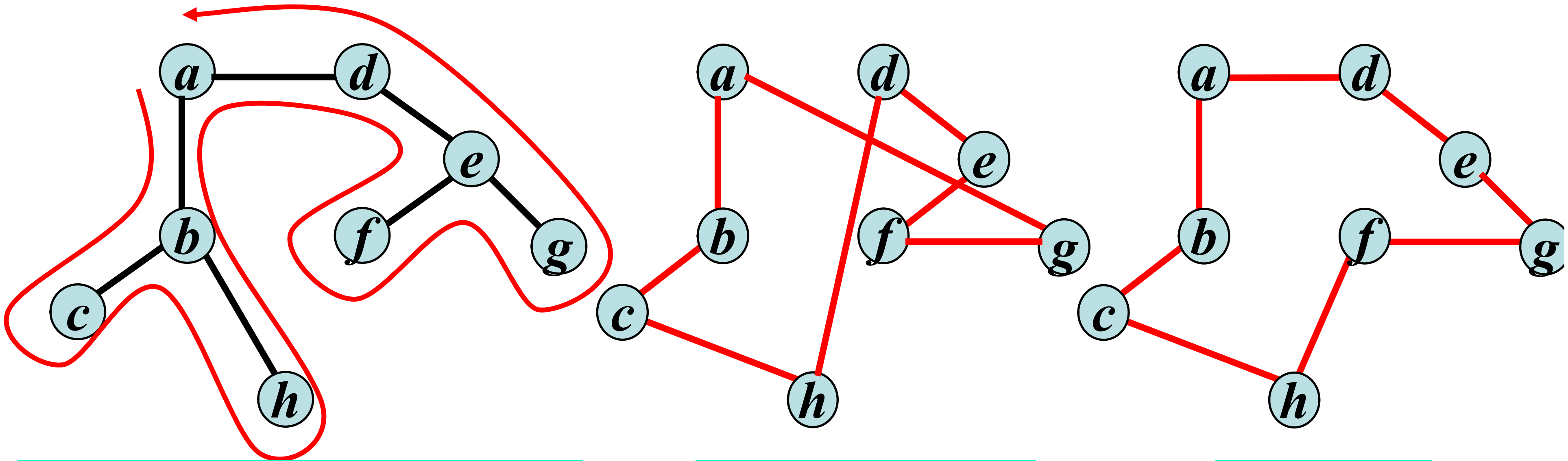
  $$C(u,w) \le C(u,v)+C(v,w).$$

- 输出

  具有最小代价的Hamilton环

- Hamilton环是一个包含V中每个结点一次的简单环.
- 代价函数的扩展: 设$A \subseteq E$, $C(A)=\sum_{(u,v) \in A} C(u, v)$.
- 不满足三角不等式的TSP问题不存在具有常数Ratio Bound的近似算法, 除非$NP=P$.

- 基本思想
  - 首先构造最小生成树(可以使用第五章的算法)
  - 先序遍历最小生成树，构造TSP的解



先序遍历: abchdefga

先序遍历解

优化解

- 近似算法

  APPROX-TSP-TOUR(G,C)

  1. 选择一个$r \in V[G]$作为生成树的根;
  2. 调用MST-Prim(G, C, r)生成一个最小生成树T;
  3. 先序遍历T, 形成有序结点表L;
  4. 按照L中的顺序访问各结点, 形成哈密顿环.

- 时间复杂性

  第2步：$O(|V|^2)$

  第3步：$O(|E|)=O(|V|^2)$，因为G是完全图，

  第4步：$O(|V|)$

  $T(G)=O(|V|^2)$

APPROX-TSP-TOUR(G,C)

1. 选择一个$r\in V[G]$作为生成树的根;

2. 调用MST-Prim(G, C, r)生成一个最小生成树T;

3. 先序遍历T，形成有序结点表L;

4. 按照L中的顺序访问各结点，形成哈密顿环.

- 解的精确度

定理1. APPROX-TSP-TOUR具有Ratio Bound 2.

证.

设H^*是TSP问题的优化解, H是算法产生的近似解.

我们需要证明$C(H) \leq 2C(H^*)$.

从H^*中删除任意一条边, 可以得到G的一个生成树T'.

设T是算法第2步产生的导致H的最小生成树, 则

$$C(T) \leq C(T') \leq C(H^*).$$

$C(T) \le C(T') \le C(H^*)$.

T的一个full walk W列出了所有结点(第一次访问的和以后从一个子树返回时再访问的). 前面例子的full walk给出顺序：a,b,c,b,h,b,a,d,e,f,e,g,e,d,a

由于W通过每条边两次, $C(W)=2C(T)$, 进而$C(W) \le 2C(H^*)$.

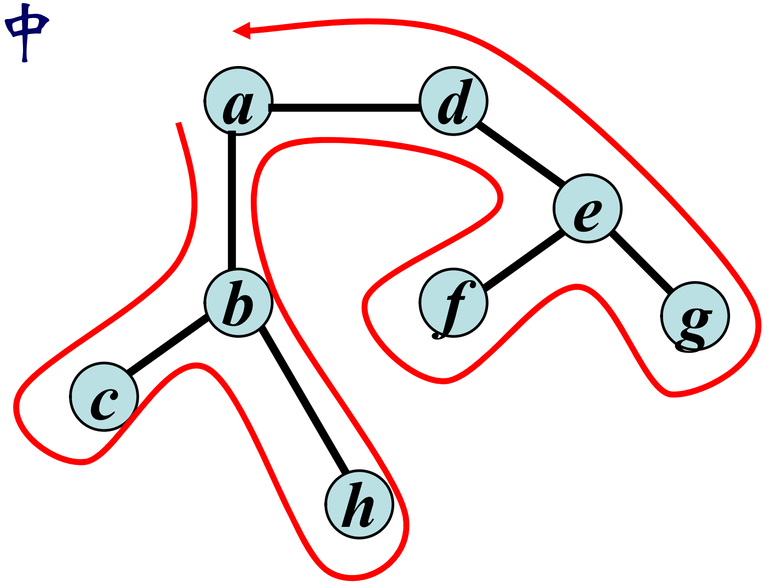
W不是哈密顿环, 因为它通过某些结点多于一次.

根据三角不等式, 我们可以从W中删除对一个结点的任何访问, 而不增加代价. (例如：从$u \rightarrow v \rightarrow w$ 删除v得$u \rightarrow w$)

反复地应用上述操作, 我们可以从W中删除所有对任何结点的非第一次访问, 得到一个算法中的*preorder walk*.

在我们的例子中, 操作结果是:
a, b, c, h, d, e, f, g, a.

由于T的*preorder walk*导致H, 我们有$C(H) \le C(W)$, 即
$C(H) \le 2C(H^*)$, 明所欲证.



8.3 基于贪心策略的近似算法

- 最小完工时间调度问题
- 集合覆盖问题
- 不相交路径问题
- 次模函数与贪心近似

8.3.1 最小完工时间调度问题

- 问题的定义
- 近似算法的设计
- 算法的性能分析

问题的定义

- **输入**

  计算时间分别为$t_1,\ldots,t_n$的n个任务；

  m台完全一样的机器

- **输出**

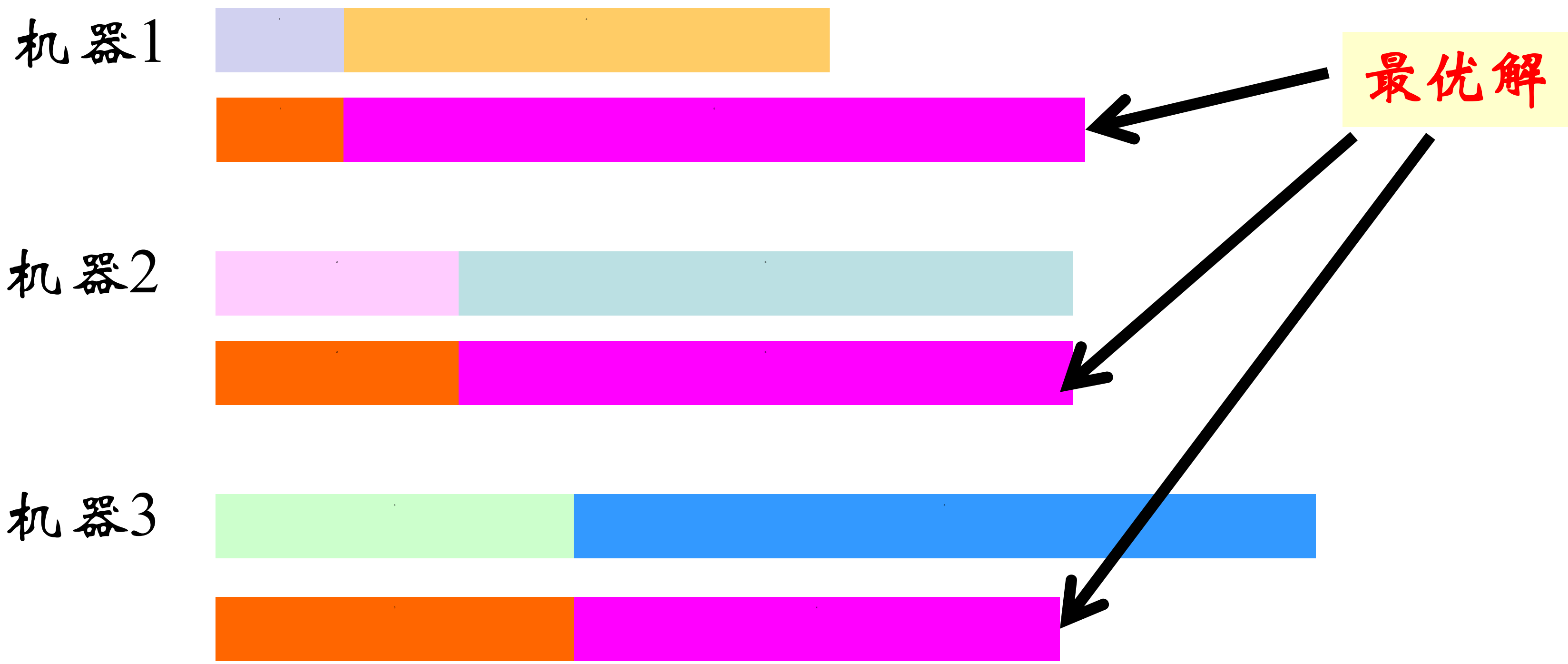
  计算任务在m台机器上的一个调度策略

  使并行执行时间最短

Minimum makespan scheduling on identical machines 是一个NP完全问题.

- 基本思想
  - 贪心选择:选择具有最短任务队列的机器

例如：m=3, t_1~t_6=1,2,3,4,5,6 (不限定任务序)



HIT

- 算法

MakeSpanScheduling ()

1. 任意排定所有任务的一个顺序$t_1,\ldots,t_n$
2. For $k \leftarrow 1$ To m Do
3. $T_k \leftarrow 0$, $M_k \leftarrow \varnothing$
4. For i=1 to n Do
5. 找出j使得$T_j = min_{1\le k \le m}\{T_k\}$
6. $T_j \leftarrow T_j + t_i$; $M_j \leftarrow M_j \cup \{i\}$
7. 输出$M_1,\ldots,M_m$

时间复杂性$O(nm)$

定理:MakeSpanScheduling算法的近似比为2

证明： 令 T^* 为最优解代价(并行时间)

有： $T^* \geq (\sum_{1\leq i\leq n} t_i)/m$ 且 $T^* \geq t_i$ $(1\leq i\leq n)$

令T为近似解的代价,

且近似解中最后处理任务为t_h, 其在第j台机器上执行,

则有$T=T_{start}+t_h$ (T_{start}为第h个任务开始执行的时间)

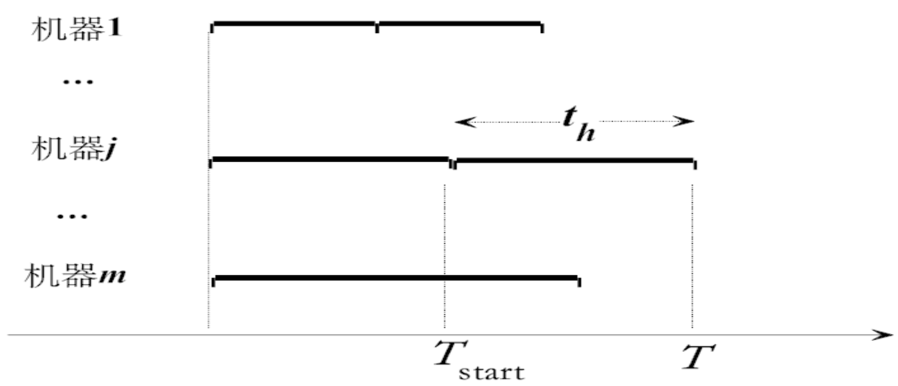
因$T_{start} \leq ((\sum_{1\leq i\leq n} t_i)-t_h)/m$

则$T \leq ((\sum_{1\leq i\leq n} t_i)-t_h)/m + t_h$

$= (\sum_{1\leq i\leq n} t_i)/m+(1-1/m)t_h$

$\leq T^* +(1-1/m)\, T^*$

$=(2-1/m)\, T^* \leq 2T^*$



8.3.2 集合覆盖问题

- 问题的定义
- 近似算法的设计
- 算法的性能分析

问题的定义

- 输入:

  有限集X, X的子集合族F, $X=\cup_{S\in F} S$

- 输出:

  $C\subseteq F$, 满足

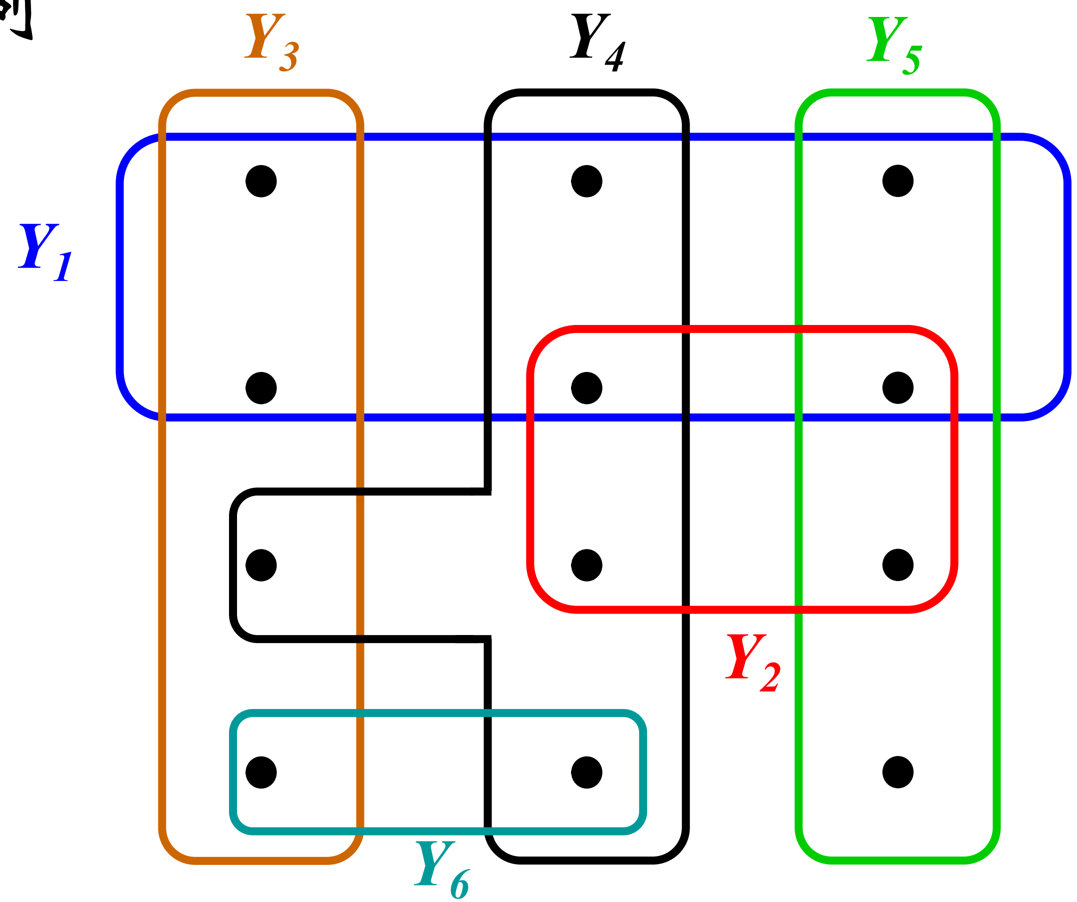
  (1). $X=\cup_{S\in C} S$,

  (2). C是满足条件(1)的最小集族, 即$|C|$最小.

*最小集合覆盖问题是很多实际问题的抽象.

*最小集合覆盖问题是NP-完全问题.

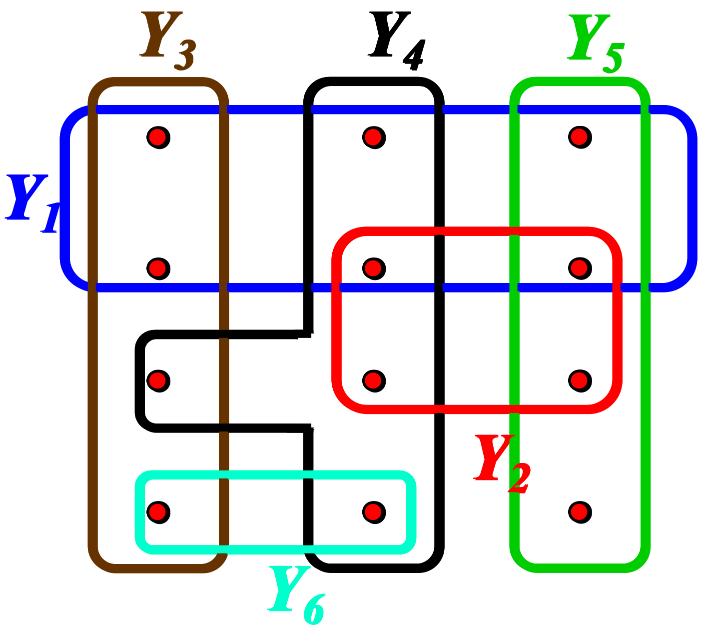
- 问题的实例



X=12个黑点，$F=\{Y_1, Y_2, Y_3, Y_4, Y_5, Y_6\}$

优化解$C=\{Y_3, Y_4, Y_5\}$

- 基本思想
  - 贪心选择:选择能覆盖最多未被覆盖元素的子集



$C=\{Y_1, Y_4, Y_5, Y_3\}$

- **算法**

Greedy-Set-Cover(X, F)

1. $U \leftarrow X$; /* U是X中尚未被覆盖的元素集 */
2. $C \leftarrow \theta$;
3. **While** $U \neq \theta$ **Do**
4. Select $S \in F$ 使得$|S \cap U|$最大;
   /* Greedy选择—选择能覆盖最多U元素的子集S */
5. $U \leftarrow U - S$;
6. $C \leftarrow C \cup \{S\}$; /* 构造$X$的覆盖 */
7. *Return C.*

- 时间复杂性
  - 3-6的循环次数至多为$|X|$
  - 计算$|S\cap U|$需要时间$O(|X|)$
  - 第4步需要时间$O(|F||X|)$
  - $T(X,F)=O(|F||X|min(|X|,|F|))$

Greedy-Set-Cover(X, F)
1. $U \leftarrow X$;
2. $C \leftarrow \theta$;
3. **While** $U \neq \theta$ **Do**
4. Select $S \in F$ 使得$|S\cap U|$最大;
5. $U \leftarrow U-S$; $F=F-\{S\}$
6. $C \leftarrow C \cup \{S\}$;
7. Return C.

HIT
CS&E

- Ration Bound

定理1. *Greedy-Set-Covers*的Ratio Bound是 $p(n)$ 多项式近似算法, $p(n)=H(max\{|S| \mid S\in F\})$, 其中 $H(d)=\sum_{1\leq i\leq d} 1/i$.

证. 设 C^* 是优化集合覆盖, C^* 的代价是 $|C^*|$.

令 S_i 是由Greedy-Set-Cover选中的第 i 个子集.

当把 S_i 加入 C 时, C 的代价加 1. 我们把选择 S_i 增加的代价均匀分配到由 S_i 首次覆盖的所有结点

.

$\forall x \in X$, 令c_x是分配到x的代价. 若x被S_i首次覆盖, 则

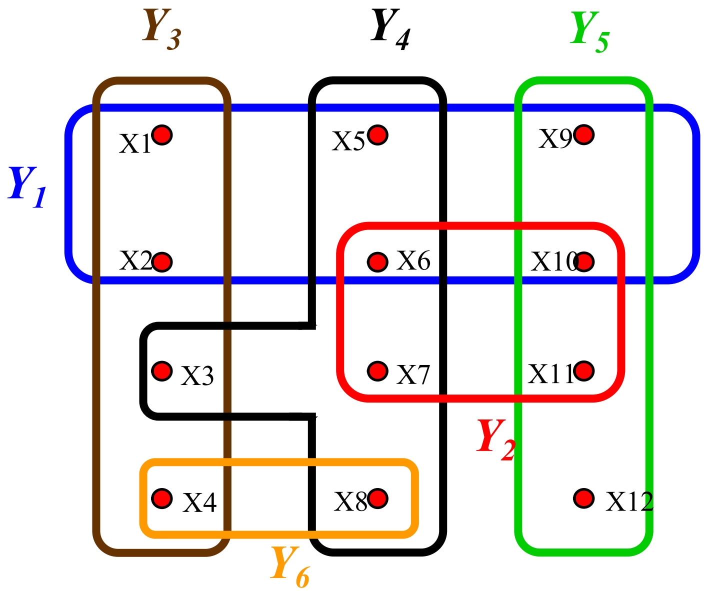
$$c_x = \frac{1}{|S_i - (S_1 \cup S_2 \cup \ldots \cup S_{i-1})|}$$

如右图中：

优化解$C^*=\{Y_3, Y_4, Y_5\}$, 代价=3

近似解$C=\{Y_1, Y_4, Y_5, Y_3\}$,代价=4

$$\sum_{x \in X} c_x = ? \qquad \sum_{S \in C^*} \sum_{x \in S} c_x = ?$$



	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12
Y1	1/6	1/6			1/6	1/6			1/6	1/6		
Y4			1/3		*	*	1/3	1/3				
Y5									*	*	1/2	1/2
Y3	*	*	*	1								

$$|C| = \sum_{x \in X} c_x \leq \sum_{S \in C^*} \sum_{x \in S} c_x$$

注意：上式的小于成立是因为C^*中各子集可能相交，某些c_x被加了多次，而左式每个c_x只加一次.

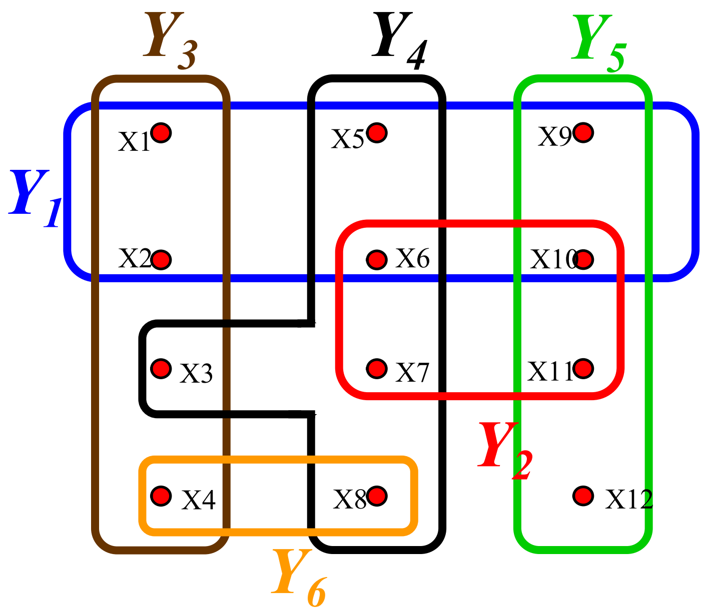
如果$\forall S \in F$, $\sum_{x \in S} c_x \leq H(|S|)$成立，则

$$|C| \leq \sum_{S \in C^*} H(|S|) \leq |C^*| \cdot H(max\{|S| \mid S \in F\}),$$

即$|C|/|C^*| \leq H(max\{|S| \mid S \in F\})$，定理成立.

下边我们来证明：对于$\forall S \in F$, $\sum_{x \in S} c_x \leq H(|S|)$.

HIT
CS&E



$C=\{Y_1, Y_4, Y_5, Y_3\}$

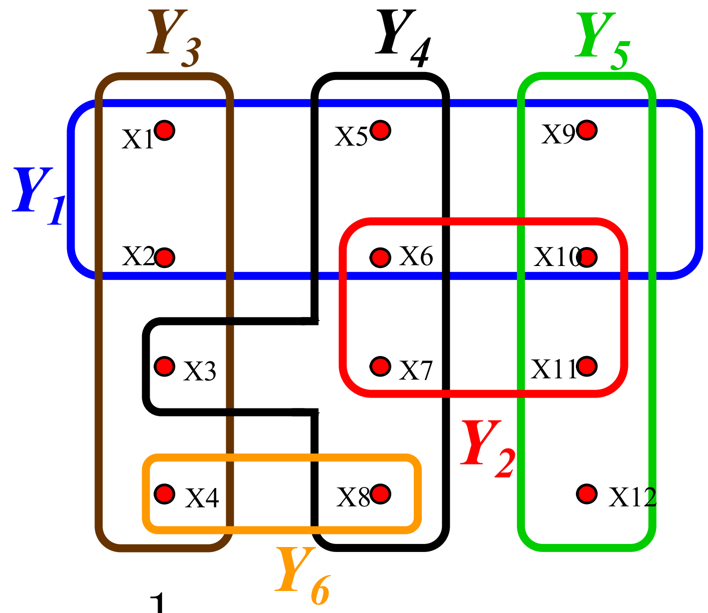
对于$\forall S\in F$和$i=1,2,\ldots|C|$, 令$u_i=|S-(S_1\cup S_2\cup \ldots\cup S_i)|$是$S_1$、$S_2$、…、$S_i$被选中后, S中未被覆盖的点数. S_i在S之前被选中

令$u_0=|S|$, $k=min\{i|u_i=0\}$, $u_i=0$意味S中每个元素被S_1、S_2、…、S_i中至少一个覆盖.

显然, $u_{i-1}\geq u_i$, $u_{i-1}-u_i$是S中由S_i第一次覆盖的元素数.

u_{i-1}-u_i：是S中由S_i第一次覆盖的元素数.
$|S-(S_1\cup S_2\cup \ldots \cup S_i)|$：是$S_1$、$S_2$、…、$S_i$被选中后，$S$中未被覆盖的点数.

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}
Y_1	1/6	1/6			1/6	1/6			1/6	1/6		
Y_4			1/3		*	*	1/3	1/3				
Y_5									*	*	1/2	1/2
Y_3	*	*	*	1								



于是，$$\sum_{x\in S} c_x = \sum_{i=1}^{k}(u_{i-1}-u_i)\times\frac{1}{|S_i-(S_1\cup S_2\cup\ldots\cup S_{i-1})|}$$

$$|S_i-(S_1\cup S_2\cup \ldots\cup S_{i-1})|\geq|S-(S_1\cup S_2\cup \ldots\cup S_{i-1}|=u_{i-1},$$

因为Greedy算法保证：S不能覆盖多于S_i覆盖的新结点数，否则S将在S_i之前被选中. 于是，

$$\sum_{x\in S} c_x \leq \sum_{i=1}^{k}(u_{i-1}-u_i)\bullet\frac{1}{u_{i-1}}$$

$$\sum_{x \in S} c_x \le \sum_{i=1}^{k} (u_{i-1} - u_i) \bullet \frac{1}{u_{i-1}}$$

$$= \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{u_{i-1}}$$

$$\le \sum_{i=1}^{k} \sum_{j=u_i+1}^{u_{i-1}} \frac{1}{j} \qquad (\because j \le u_{i\text{-}1})$$

$$= \sum_{i=1}^{k} \left(\sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^{u_i} \frac{1}{j} \right)$$

$$= \sum_{i=1}^{k} H(u_{i-1}) - H(u_i)$$

$$= H(u_0) - H(u_k)$$

$$= H(u_0) - H(0) \qquad (\because u_k = 0)$$

$$= H(u_0) = H(|S|) \qquad (\because H(0) = 0,\ u_0 = |S|)$$

推论1. Greedy-Set-Cover是一个Ratio Bound 为 $\ln|X|+1$ 的多项式时间近似算法.

证. 由不等式$H(n)\le\ln(n)+1$可知

$$H(max\{|S| \mid S\in F\})\le H(|X|)\le\ln|X|+1.$$

8.3.3 不相交路径问题

- 问题的定义
- 近似算法的设计
- 算法的性能分析

问题的定义

- **输入:**

  图 $G=(V,E)$,源点集$S\subseteq V$,汇点(目标点)集$T\subseteq V$

- **输出:**

  $A\subseteq S\times T$

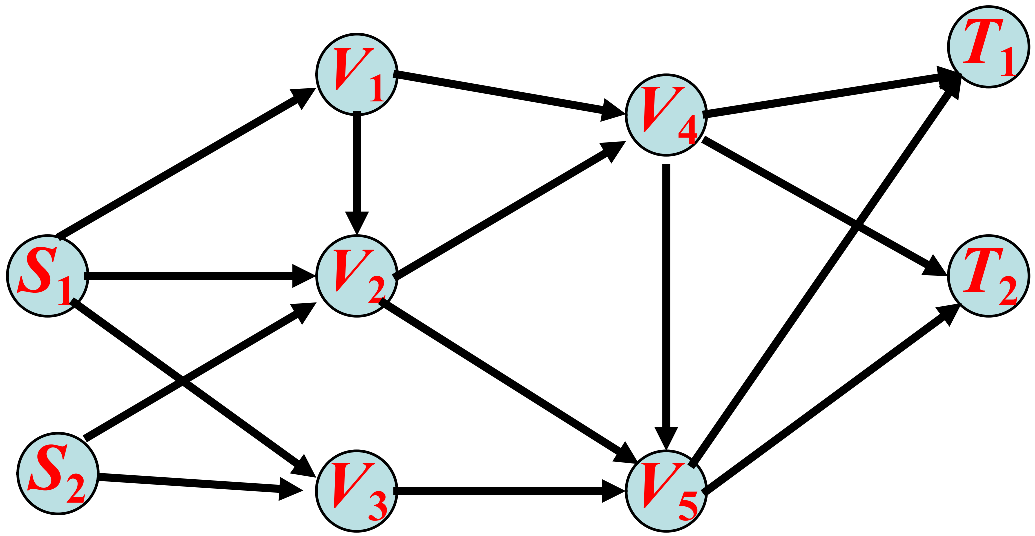
  (1). A中的所有顶点对在G中存在无公共边的路径

  (2). $|A|$最大.

*边不相交路径问题是很多实际问题的抽象.

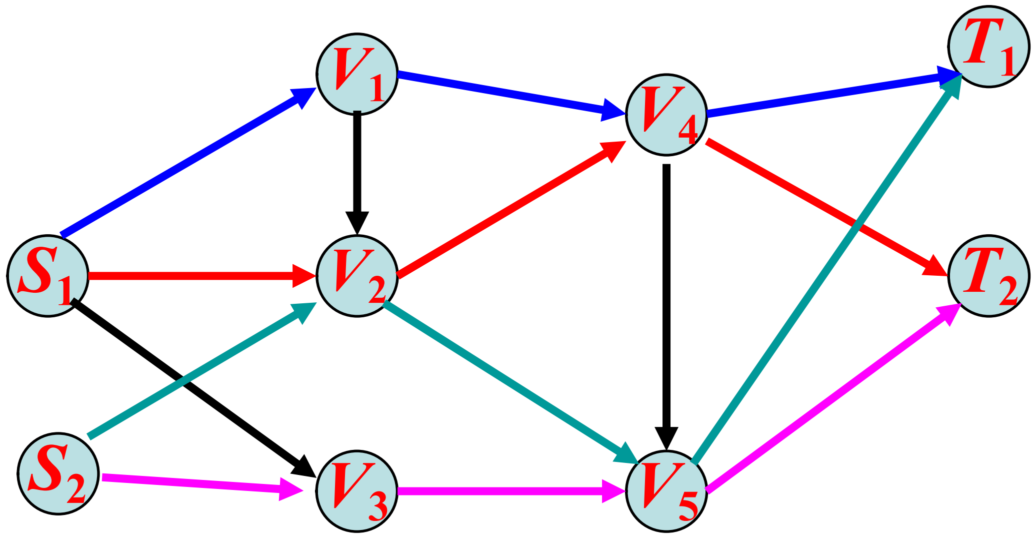
*是NP-完全问题.

- 问题的实例：图G



$S=\{S_1, S_2\}$ $T=\{T_1, T_2\}$

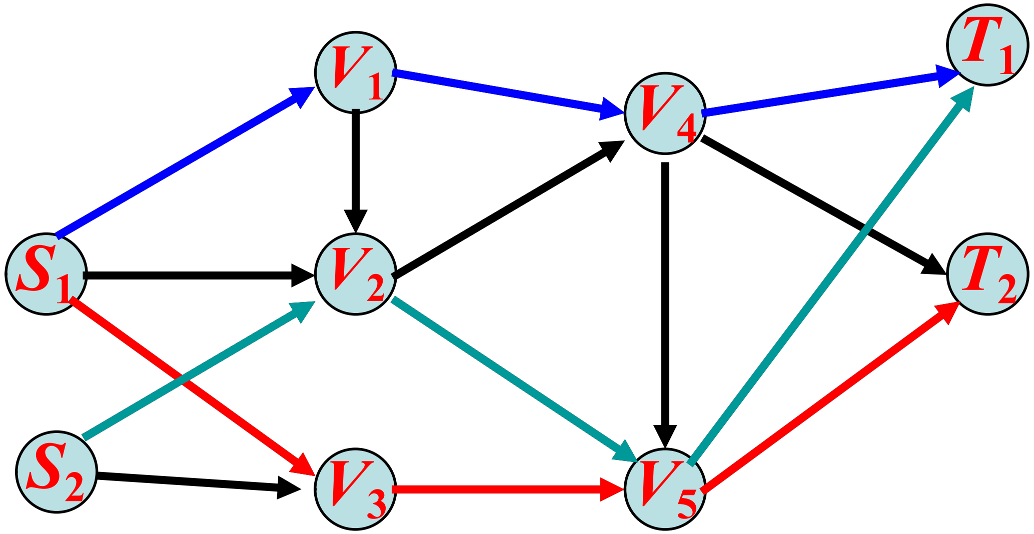
• 问题的实例：图G



$S=\{S_1, S_2\}$ $T=\{T_1, T_2\}$

优化解$A=\{(S_1,T_1),(S_1,T_2),(S_2,T_1),(S_2,T_2)\}$

- 基本思想
  - 贪心选择:选择$(u,v)\in S\times T$使得该顶点对间路径最短



$A=\{(S_1,T_1), (S_1,T_2), (S_2,T_1)\}$

精确解$A^*=\{(S_1,T_1),(S_1,T_2),(S_2,T_1),(S_2,T_2)\}$

- 算法

EdgeDisjointPath(G,S,T)

1. $A \leftarrow \varnothing$; $B \leftarrow S \times T$
2. While true Do
3. 计算B中所有顶点对在G中的最短路径构成P;
4. IF $P=\varnothing$ Then break;
5. 选择P中长度最短的路径$P_{u,v}$ /*贪心选择*/
6. $A \leftarrow A \cup \{(u,v)\}$; $G \leftarrow G-P_{u,v}$; $B \leftarrow B-\{(u,v)\}$;
   /*根据贪心选择更新A、图G和B*/
7. 输出A.

时间复杂度 $O(|S||T||V|^3)$

第3步的开销$O(|V|^3)$,第5-6步开销为 $O(|E|)$

- 解的精确度

定理. 算法EdgeDisjointPath的近似比为$O(m^{1/2})$,其中$m=|E|$

证.

A^*—问题最优解，　　A—近似算法输出的近似解

k—参数，任意固定的值

证明思路：

用参数k将A^*划分为两个部分S^*,L^*使得$A^*=S^*\cup L^*$

S^*—A^*中长度小于等于k的路径(短路径)，$|S^*|\le k|A|$(引理2)

L^*—A^*中长度大于k的路径(长路径),$|L^*|\le(m/k)|A|$(引理1)

$|A^*|=|S^*|+|L^*|\le(k+m/k)|A|$　　(对任意k成立)

取$k=m^{1/2}$时得到　　$|A^*|\le 2m^{1/2}|A|$

引理1. $|L^*|\le(m/k)|A|$对任意k成立。

证. A^*—最优解

L^*—A^*中长度大于k的路径

A—近似解，$1\le|A|$

L^*中任意两条路径没有公共边

L^*中的所有路径至少用到$k|L^*|$条边

$k|L^*| \le|E|=m$

$|L^*| \le(m/k)|A|$

引理2. $|S^*|\leq k|A|$对任意k成立。

证. A^*—最优解 S^*—A^*中长度$\leq k$的路径

A—近似解

任意$p^*\in A^*$必然与A中某条路径相交(有公共边)

否则，近似算法终止时，p^*仍然存在于图G中,与终止条件矛盾

任意$p^*\in S^*\subseteq A^*$， p^*中至多有k条边

p^*至少与A中某一条路径相交(A是由算法逐渐添加路径得到的)

假定计算A过程中，第一条与p^*相交的路径为p

算法选择p加入A而未选择p^*,说明p比p^*更短，$|p|\leq k$

$Q=\{p\in A$: p与S^*中某条短路径相交$\}$

Q中每条边至多与A^*(即S^*)中一条路径相交

$|S^*|\leq Q$中边的总数$=k\times|Q|\leq k|A|$

8.3.4 次模函数与贪心近似

- 次模函数及基本性质
- 次模函数示例
- 次模函数最大化贪心近似算法

次模函数=Submodular Function=亚模函数=子模函数

给定有限集U, 如果集合函数f:$2^U \rightarrow R$满足(1)或(2)，则称f是次模函数

(1): $f(A)+f(B) \geq f(A \cup B)+f(A \cap B)$ 对任意$A,B \subseteq U$成立

(2): $f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$ 对$\forall A \subseteq B \subseteq U$和$\forall x \in U-B$成立

证明：(1)⇒(2)：由$A \subseteq B$和$x \in U-B$可得

$$(A \cup \{x\}) \cup B = B \cup \{x\},\ (A \cup \{x\}) \cap B = A$$

因此如果(1)成立，则有

$$f(A \cup \{x\}) + f(B) \geq f(B \cup \{x\}) + f(A)$$

因而，有 $f(A \cup \{x\}) - f(A) \geq f(B \cup \{x\}) - f(B)$

次模函数=Submodular Function=亚模函数=子模函数

给定有限集U, 如果集合函数f:$2^U \rightarrow R$满足(1)或(2)，则称f是次模函数

(1): $f(A)+f(B) \geq f(A\cup B)+f(A\cap B)$ 对任意$A,B \subseteq U$成立

(2): $f(A\cup\{x\})-f(A) \geq f(B\cup\{x\})-f(B)$ 对$\forall A\subseteq B\subseteq U$和$\forall x\in U-B$成立

证明(2)⇒(1):对于任意$C\subseteq U$，$x\in U$，设$\Delta_C f(A)=f(A\cup C)-f(A)$，$\Delta_x f(A)=f(A\cup\{x\})-f(A)$

则公式(2)可简化为 $\Delta_x f(A)\geq \Delta_x f(B)$ 对$\forall A\subseteq B\subseteq U$和$\forall x\in U-B$成立

令$D=A-B=\{x_1,x_2,\ldots,x_k\}$，则

$$
\begin{aligned}
f(A)-f(A\cap B)=\Delta_D f(A\cap B) &=\sum_{i=1}^{k}\Delta_{x_i} f((A\cap B)\cup\{x_1,x_2,\ldots,x_{i-1}\}) \\
&\geq \sum_{i=1}^{k}\Delta_{x_i} f(B\cup\{x_1,x_2,\ldots,x_{i-1}\}) \\
&=\Delta_D f(B)=f(B\cup D)-f(B) \\
&=f(A\cup B)-f(B)
\end{aligned}
$$

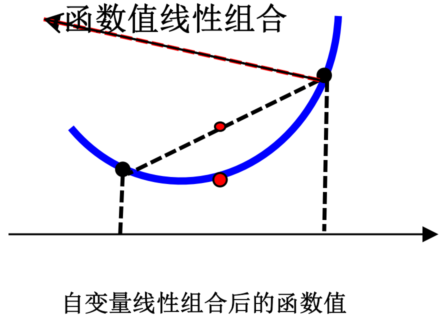
因而，可得 $f(A\cup B)+f(A\cap B) \leq f(A)+f(B)$

次模函数=Submodular Function=亚模函数=子模函数

给定有限集U, 如果集合函数f:$2^U \to R$满足(1)或(2)，则称f是次模函数

(1): $f(A)+f(B) \geq f(A\cup B)+f(A\cap B)$ 对任意$A,B \subseteq U$成立

(2): $f(A\cup\{x\})-f(A) \geq f(B\cup\{x\})-f(B)$ 对$\forall A\subseteq B\subseteq U$和$\forall x\in U-B$成立

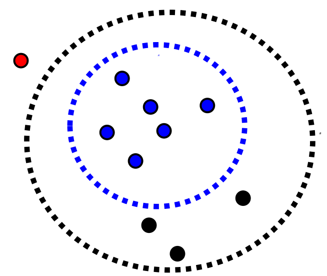


条件(1)的直观含义

集合变量函数的凸性

集合运算后的函数值

小于等于函数值的组合



条件(2)的直观含义

边缘增益递减

X在小集合上的增益

大于等于X在大集合上的收益

次模函数基本性质

(B): 集合函数$f:2^U\rightarrow R$是次模函数,则

(1) $f(B\cup A)-f(A)\le\sum_{x\in B}[f(A\cup\{x\})-f(A)]$

(2) $f(B\cup A)-f(A)\le\sum_{x\in B\setminus A}[f(A\cup\{x\})-f(A)]$

(3) 若$A\subseteq B$,则$f(B)-f(A)\le\sum_{x\in B\setminus A}[f(A\cup\{x\})-f(A)]$

证明: (1)令$B=\{b_1,..,b_k\}$,并记$B_i=\{b_1,b_2,\ldots,b_i\}$

$$f(A\cup B)=f(A\cup B_k)-f(A\cup B_{k-1})+f(A\cup B_{k-1})-\ldots-f(A\cup B_0)+f(A\cup B_0)$$

$$f(A\cup B)-f(A)=\sum_{i=1}^{k}[f(A\cup B_{i-1}\cup\{b_i\})-f(A\cup B_{i-1})]$$

$$\le\sum_{i=1}^{k}[f(A\cup\{b_i\})-f(A)]$$

f是次模函数且$A\cup B_{i-1}\supseteq A$

(2)$A\cup B=A\cup(B-A)$, 应用(1)即可得(2)

(3)$A\cup B=B$ 应用(2)即可得(3)

HIT
CS&E

次模函数运算性质

(O): 集合函数$f,g:2^U\to R$是次模函数且$T\subseteq U$，则

(1) $h(X)=c\cdot f(X)$对任意$c\geq 0$是次模函数

(2) $h(X)=f(X)+g(X)$是次模函数

(3) $h(X)=f(X\cap T)$ 是次模函数

(4) $h(X)=f(U-X)$ 是次模函数

(5)如果f是单调的且$c\geq 0$,则$h(X)=\min\{c,f(X)\}$是单调次模函数

f是单调的指的是：$A\subseteq B\Rightarrow f(A)\leq f(B)$

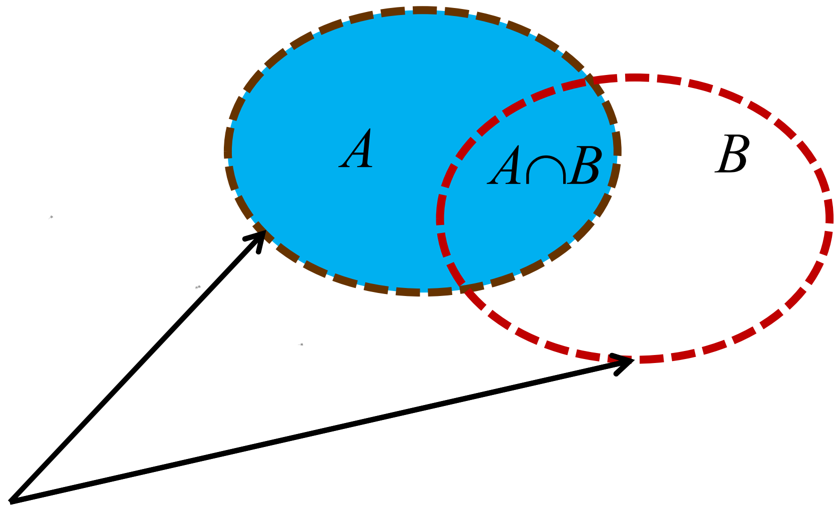
习题：证明上述性质

跳过其他性质

次模函数示例(1)

例1: 对任意$x \in U$赋予非负权值$w(x)$，定义$f(A)=\sum_{x\in A} w(x)$

则$f(A)$是次模函数



$$f(A\cup B) + f(A\cap B) = f(A) + f(B)$$

次模函数示例(2)

例2:预算函数[$f(A)$表示购买A中所有商品时愿意支付的预算]

对任意$x\in U$赋予非负权值$w(x)$并取$b \geq 0$, 定义$f(A)=\min\{\sum_{x\in A}w(x),b\}$

则$f(A)$是次模函数

由例1和亚模函数运算性质知f是次模函数

例3:秩函数

令$U=\{v_1,v_2,\ldots,v_m\}$是R^n中元素构成的集合。对$\forall A\subseteq U$定义$f(A)$是A中向量张成的线性子空间的维数，则$f(A)$是次模函数

线性代数中子空间维数定理

次模函数示例(3)

例4:覆盖函数

$E=\{e_1,\ldots,e_n\}$是有限集合。$U=\{s_1,\ldots,s_m\}$是E的一个子集族（即$s_i\subseteq E$）

对$\forall A\subseteq U$定义$f(A)=|\cup_{s\in A}s|$，则$f(A)$是标准化的单调次模函数

(1) $f(\varnothing)=0$　故f是标准化的

(2) 如果$A\subseteq B$，显然$f(A)\leq f(B)$，故$f(A)$是单调的

(3) 考虑任意$A\subseteq B\subseteq U$ 和$x\in U-B$

$$f(A\cup\{x\})-f(A)=\ |e\in E|\ e\in x-\cup_{s\in A}s\ |$$

$$\geq\ |e\in E|\ e\in x-\cup_{s\in B}s\ |=f(B\cup\{x\})-f(B)$$

故f是次模的

次模函数示例(4)

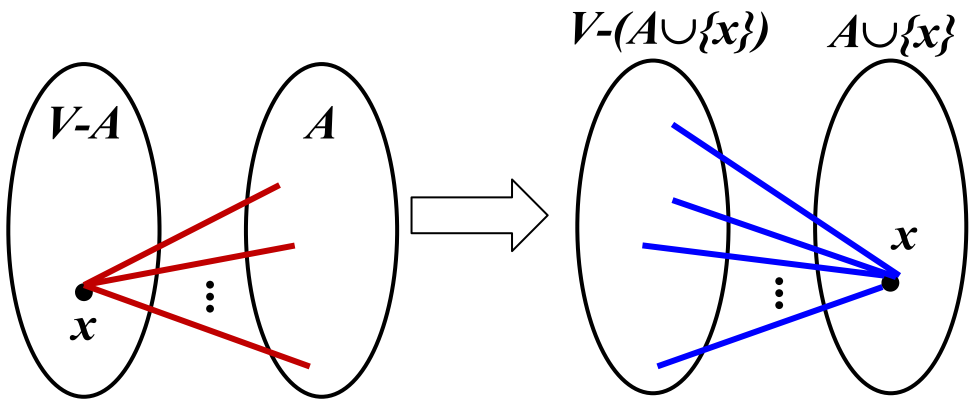
例5:割函数: 令G=(V,E)是一个非负加权图，任意$uv\in E$的权值为$w(uv)$。

任意$A\subseteq V$定义的割是$\delta(A)=\{uv\in E|u\in A,v\in V-A\}$。定义$f(A)=\sum_{uv\in\delta(A)}w(uv)$

则$f(\cdot)$是标准化对称次模函数

(1) $f(\varnothing)=0$ 故f是标准化的
(2) $f(A)=f(V-A)$，故$f(A)$是对称的

(3)次模性：



$f(A\cup\{x\})-f(A)=\sum_{u\in V-A}w(ux)-\sum_{v\in A}w(xv)$

For any $A\subseteq B\subseteq V$

$\sum_{v\in B}w(xv)-\sum_{v\in A}w(xv)\geq 0$

$\sum_{v\in V-B}w(xv)-\sum_{v\in V-A}w(xv)\leq 0$

$\Downarrow$

$f(A\cup\{x\}) - f(A) \geq f(B\cup\{x\}) - f(B)$

次模函数最大化贪心算法

问题

输入：空间U及其上的非负单调次模函数f:$2^U \to R^+$

判定子集A是否为解的约束条件$I=\{A \| |A| \le k\}$

输出：$A \subseteq U$使得

$$\max\ f(A)$$
$$\text{s.t.}\quad |A| \le k$$

算法：

1. $A \leftarrow \varnothing$
2. while $|A|<k$ Do
3. $\quad x^* \leftarrow \max_{x\in U-A} \{f(A\cup\{x\})-f(A)\}$ 贪心选择
4. $\quad A \leftarrow A \cup \{x^*\}$
5. Return A

结论： $f(A) \ge (1-1/e)\cdot f(A_{opt})$，亦即贪心算法的近似比为$(1-1/e)^{-1} \approx 1.59$

证明：令$A_i=\{x_1,x_2,\ldots,x_i\}$是循环进行$i$轮后得到的结果，显然$A=A_k$

$$f(A_i)-f(A_{i-1})=f(A_{i-1}\cup\{x_i\})-f(A_{i-1})$$

$$\geq \max_{x\in A_{opt}}\ [f(A_{i-1}\cup\{x\})-f(A_{i-1})]$$ 贪心选择

$$\geq \frac{\sum_{x\in A_{opt}}\ [f(A_{i-1}\cup\{x\})-f(A_{i-1})]}{k}$$ 最大值≥平均值

$$\geq \frac{[f(A_{OPT}\cup A_{i-1})-f(A_{i-1})]}{k}$$ $f(B\cup A)-f(A)\leq \sum_{x\in B}[f(A\cup\{x\})-f(A)]$

$$\geq \frac{[f(A_{OPT})-f(A_{i-1})]}{k}$$ 单调性

于是：$f(A_{OPT})-f(A_i)\leq (1-1/k)[f(A_{OPT})-f(A_{i-1})]$

$$f(A_{OPT})-f(A)=f(A_{OPT})-f(A_k)$$ $A=A_k$

$$\leq (1-1/k)[f(A_{OPT})-f(A_{k-1})]$$

$$\leq \ldots$$

$$\leq (1-1/k)^k[f(A_{OPT})-f(A_0)]$$ 注意$(1-1/k)^k\uparrow 1/e$

$$f(A)\geq f(A_{OPT})-(1-1/k)^k\ [f(A_{OPT})-f(\varnothing)]\geq[1-1/e]\cdot f(A_{OPT})-(1/e)\cdot f(\varnothing)$$

HIT
CS&E

注意： 当代价函数不满足单调性时，近似比没有保障

证明过程用到了单调性

反例： $U=\{x_1,\ldots,x_n\}$, 定义$f(A)=\begin{cases}|A| & \text{If } x_n\notin A\\ 2 & \text{If } x_n\in A\end{cases}$

作业： (1) 证明f是非单调次模函数

(2) 证明:$k\geq 1$时，贪心算法近似解必然包含x_n

(3) 证明:$k\geq 2$时，贪心算法近似比无常数上界

(4) 能否引入随机因素改善算法性能？

背包约束最大化贪心算法(2)

问题： 输入：空间U及其加权函数w, $w(A)=\sum_{x\in A}w(A)$

U上的非负次模函数$f:2^U\rightarrow R^+$,正数b

输出：$A\subseteq U$在$w(A)\leq b$ 的条件下使$f(A)$取得最大值

算法：

1. $A\leftarrow\varnothing$
2. while $U\neq\varnothing$ Do
3. $\quad x^*\leftarrow \max_{x\in U}\{f(A\cup\{x\})-f(A)/w(x)\}$ 贪心选择
4. $\quad$ If $w(A\cup\{x^*\})\leq b$ Then $A\leftarrow A\cup\{x^*\}$
5. $\quad U\leftarrow U-\{x^*\}$
6. $x_0\leftarrow \max_{x\in U} f(\{x\})$
7. $S\leftarrow A$和$\{x_0\}$中次模函数值较大者
8. Return S

结论： $f(A)\geq 0.5\cdot(1-e^{-1})\cdot f(A_{opt})$

证明(作业)

HIT
CS&E

引理： 算法第4步条件不成立时必有$f(A\cup\{x^*\})\geq(1-1/e)f(A_{\mathrm{OPT}})$

不妨设此时$A=A_i=\{x_1,\ldots,x_i\}$，并令$A_{i-1}=\{x_1,\ldots,x_{i-1}\}$

$$f(A_{OPT})\leq f(A_{i-1})+\sum_{x\in A_{OPT}-A_{i-1}}[f(A_{i-1}\cup\{x\})-f(A_{i-1})]$$ 次模函数基本性质

$$=f(A_{i-1})+\sum_{x\in A_{OPT}-A_{i-1}}w(x)\frac{[f(A_{i-1}\cup\{x\})-f(A_{i-1})]}{w(x)}$$ 恒等变形

$$\leq f(A_{i-1})+\frac{[f(A_{i-1}\cup\{x_i\})-f(A_{i-1})]}{w(x_i)}\sum_{x\in A_{OPT}-A_{i-1}}w(x)$$ 贪心选择

$$=f(A_{i-1})+\frac{[f(A_i)-f(A_{i-1})]}{w(x_i)}\sum_{x\in A_{OPT}-A_{i-1}}w(x)$$

$$\leq f(A_{i-1})+\frac{b}{w(x_i)}[f(A_i)-f(A_{i-1})]$$ 优化解满足背包约束

于是：$f(A_i)-f(A_{\mathrm{OPT}})\geq(1-w(x_i)/b)[f(A_{i-1})-f(A_{\mathrm{OPT}})]\geq[\prod_{k=1}^{i}(1-\frac{w(x_k)}{b})](f(\varnothing)-f(A_{\mathrm{OPT}}))$

$\geq-[\exp(-w(A)/b]\,f(A_{\mathrm{OPT}})$ $f(\varnothing)=0$且$1-x\leq e^{-x}$

$$f(A\cup x^*)\geq[1-\exp(-\frac{w(A)+w(x^*)}{b})]\cdot f(A_{\mathrm{OPT}})$$

$\geq(1-1/e)\cdot f(A_{\mathrm{OPT}})$ $w(A)+w(x^*)>b$

证明[结论]：设算法获得最终A之后第4步首次为假时选中x^*,则

$$2f(S) \ge f(A)+f(x_0)$$

$$\ge f(A)+f(x^*) \qquad x_0\text{的取法}$$

$$\ge f(A \cup x^*) \qquad \text{亚可加性}$$

$$\ge (1-1/e)f(S_{OPT}) \qquad \text{引理}$$

拟阵约束最大化贪心算法(3)

问题： 输入：空间U及其上的非负次模函数f:$2^U \rightarrow R^+$
约束拟阵(U,I)
输出：$A \in I \subseteq 2^U$的条件下使$f(A)$取得最大值

算法：

1. $A \leftarrow \varnothing$
2. while $U \neq \varnothing$ Do
3. $x^* \leftarrow \max_{x \in U} \{f(A \cup \{x\})-f(A)\}$ 贪心选择
4. If $A \cup \{x^*\} \in I$ Then $A \leftarrow A \cup \{x^*\}$
5. $U \leftarrow U-\{x^*\}$
5. Return A

结论： $f(A) \geq 0.5 \cdot f(A_{opt})$，即近似比$\leq 2$

证明(作业)

证明：令$A_{OPT}=\{x_1,x_2,\ldots,x_K\}$是问题的优化解

由于近似解A和A_{OPT}均是极大独立集，故$|A|=|A_{\mathrm{OPT}}|$

设$A=\{x_1,x_2,\ldots,x_K\}$且$A_i=\{x_1,x_2,\ldots,x_i\}$。显然$A=A_K$

$f(A_{\mathrm{OPT}})-f(A) \le \sum_{x\in A_{\mathrm{OPT}}}[f(A\cup\{x\})-f(A)]$ 次模函数基本性质

$\le \sum_{x_i^*\in A_{\mathrm{OPT}}}[f(A_{i-1}\cup\{x_i\})-f(A_{i-1})]$ 次模函数定义

$\le \sum_{x\in A}[f(A_{i-1}\cup\{x\})-f(A_{i-1})]$ 贪心选择,基交换

$= \sum_{i=1}^{K}[f(A_i)-f(A_{i-1})]$

$= f(A)-f(\varnothing)$

$f(A) \ge 0.5\cdot f(A_{\mathrm{OPT}})+f(\varnothing)$

8.4 基于局部搜索的近似算法

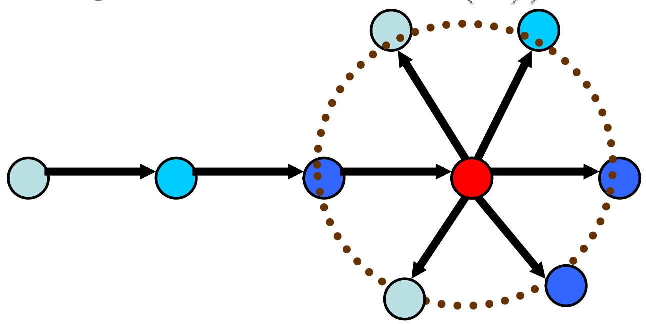
- 局部搜索原理
- 最大割问题
- 设施定位问题

8.4.1 局部搜索原理

局部搜索原理

- 局部搜索是解决最优化问题的一种启发式算法
- 工作过程：

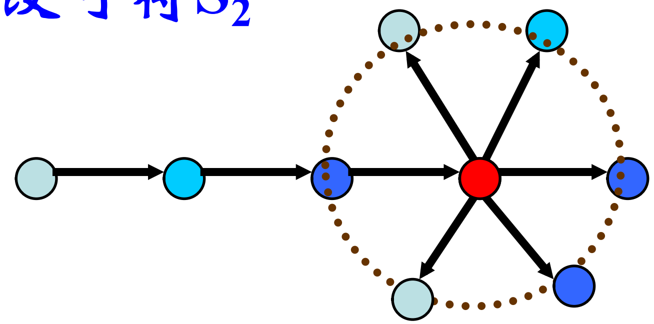
LocalSearch:
Find a "good" initial solution $S_0 \in \mathcal{S}(I)$
$S \leftarrow S_0$
repeat
  If ($\exists S' \in N(S)$ such that val(S') is strictly better than val(S))
    $S \leftarrow S'$
  Else
    S is a *local optimum*
    return S
  EndIf
Until (True)



从任意可行解出发,搜索其邻域(通过对其进行局部修改)，检查是否有更优的解存在(具有更优的解代价)，如存在则移动至该解并继续执行局部搜索，反之则意味着达到了一个局部优化解，输出局部优化解作为近似解

局部搜索原理

- 邻居关系的表示：可行解有向图
  - 每个顶点表示一个可行解
  - S_1到S_2之间存在边⇔S_1由局部修改可得S_2



- 局部搜索算法关键
  - 在搜索空间中如何定义邻域，即邻居关系的选择和定义
    - 保证每次的局部搜索在多项式时间完成
  - 在每一步选择相邻解的规则
    - 决定了算法的迭代次数，可否在有限步内找到局部最优解
  - 初始可行解的选择

局部搜索原理

- 局部搜索与贪心算法均属于启发式算法
- 与贪心算法的共性之处：
  - 做一系列局部优化的选择
  - 但这些局部选择未必可得到全局最优解
- 与贪心算法不同之处：
  - 贪心算法的解是一步步构造的，在每一步通过局部最优选择得到部分解
  - 局部搜索是从任意可行解出发，通过对其局部、小的修改，产生新的可行解，使得代价更优
    - 局部搜索无法保证一定能够在多项式时间内找到局部最优解
      - 如果可行解有向图中结点度较大的话

局部搜索原理

- 优点是
  - 简单、灵活及易于实现
  - 对于大多数难计算问题不难设计一个局部搜索算法
- 缺点是
  - 容易陷入局部最优
  - 解的质量难以保证，其与初始解和邻域的结构密切相关

8.4.2 最大割问题

- 问题定义
- 局部搜索算法
- 算法性能分析

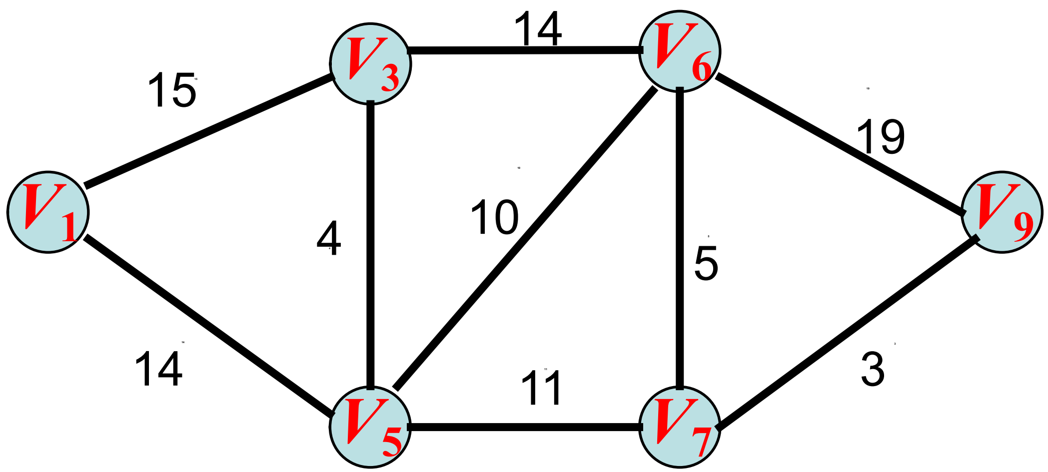
问题的定义

- **输入:**

  加权无向图$G=(V,E)$，权值函数$W:E\rightarrow N$

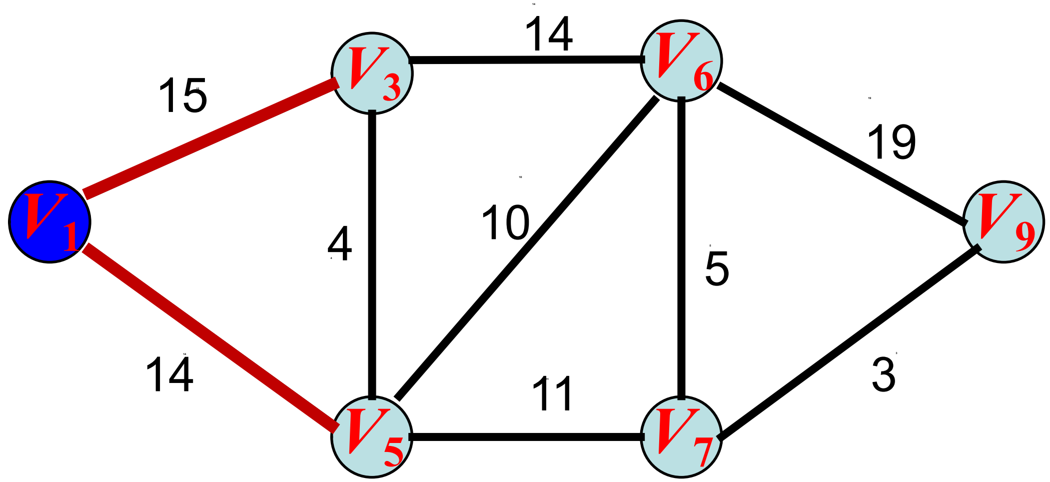
- **输出:**

  $S\subseteq V$ 使得$\sum_{u\in S,v\in V\text{-}S}W(uv)$最大



最大割问题是NP-完全问题.

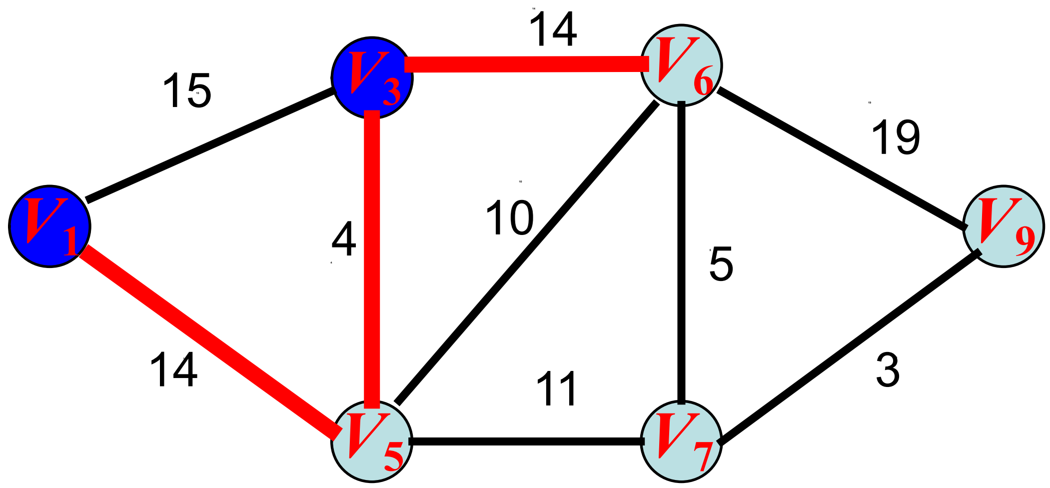
- 问题的实例



$S=\{V_1\}$ 代价29 交换V_3在S和$V\text{-}S$中的位置

HIT

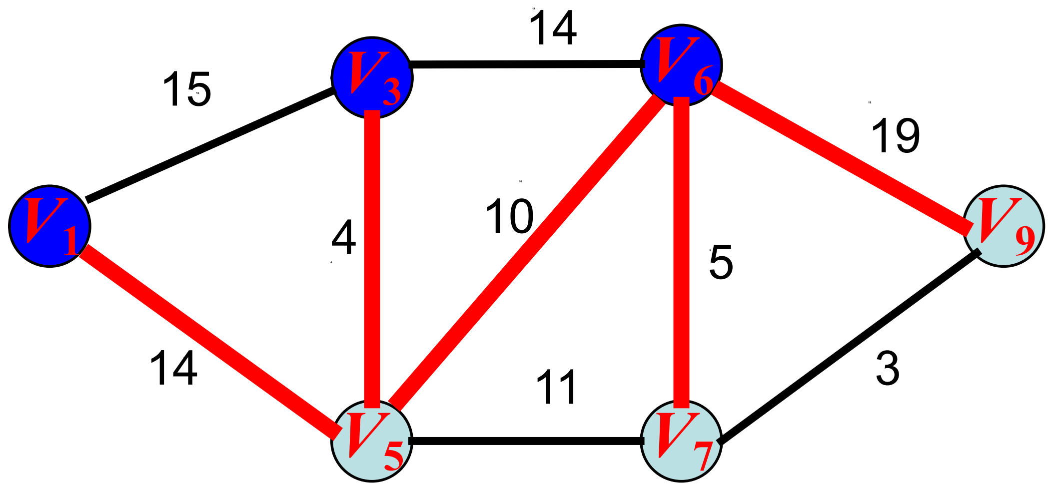
• 问题的实例



$S=\{V_1\}$　　代价29　　交换V_3在S和$V-S$中的位置

$S=\{V_1,V_3\}$　　代价32　　交换V_6在S和$V-S$中的位置

- 问题的实例

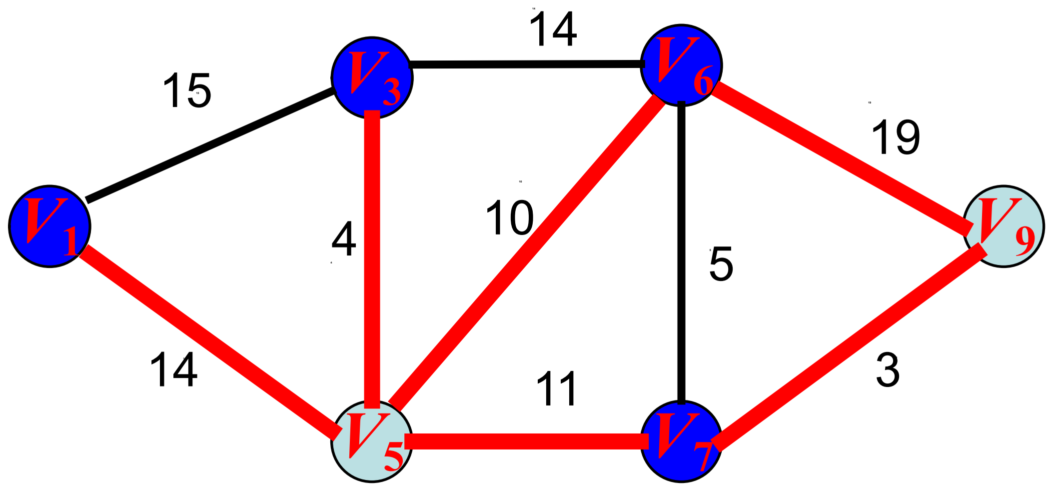


$S=\{V_1\}$ 代价29 交换V_3在S和V-S中的位置

$S=\{V_1,V_3\}$ 代价32 交换V_6在S和V-S中的位置

$S=\{V_1,V_3,V_6\}$ 代价52 交换V_7在S和V-S中的位置

- 问题的实例



$S=\{V_1\}$ 代价29 交换V_3在S和$V\text{-}S$中的位置

$S=\{V_1,V_3\}$ 代价32 交换V_6在S和$V\text{-}S$中的位置

$S=\{V_1,V_3,V_6\}$ 代价52 交换V_7在S和$V\text{-}S$中的位置

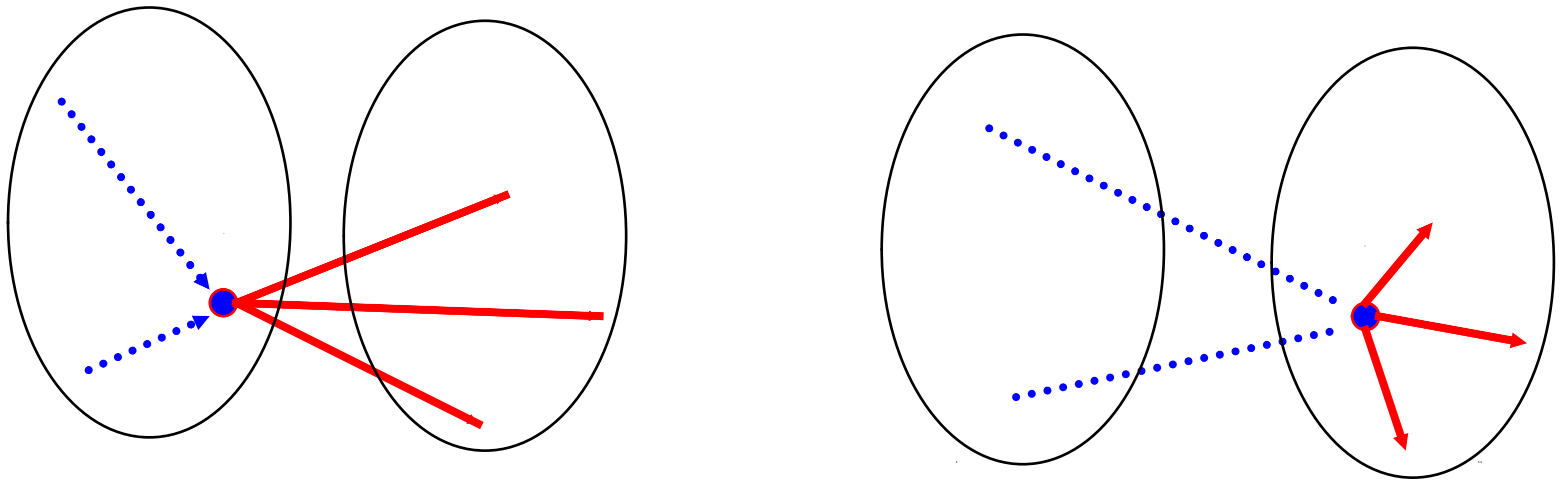
$S=\{V_1,V_3,V_6,V_7\}$代价61 局部优化解

全局优化解

HIT

算法思想：

- 局部修改操作——交换v在S和V-S中的位置
- 只有与v关联的部分边会影响割的代价



- $cost(v,S)=\sum_{u\in S:uv\in E}W(uv)-\sum_{u\in V\text{-}S:vu\in E}W(vu)$ $\qquad v\in S$
- $cost(v,S)=\sum_{u\in V\text{-}S:vu\in E}W(vu)-\sum_{u\in S:uv\in E}W(uv)$ $\qquad v\in V\text{-}S$

ApproxMaxCut算法

ApproxMaxCut(G(V,E))

输入: 加权图G=(V,E)，权值函数W:$E \rightarrow N$

输出: $S \subseteq V$ 使得$\sum_{u \in S, v \in V-S} W(uv)$最大

1. $S \leftarrow \{u\}$; /*u是V中任意顶点*/
2. repeat
3. 任取使得cost(v,S)>0的顶点v, 交换v在S,V-S中的位置;
4. until 不存在$v \in V$使得cost(v,S)>0
5. 输出S.

定理1：ApproxMaxCut算法会终止并输出S集合

证明：算法每次循环，至少使$\sum_{u\in S, v\in V\text{-}S} W(uv)$增加1

又$\sum_{u\in S, v\in V\text{-}S} W(uv) \leq \sum_{u,v\in V} W(uv)$

因而，算法一定在$\sum_{u,v\in V} W(uv)$步内终止

ApproxMaxCut(G(V,E))

输入：加权图G=(V,E)，权值函数W:$E\rightarrow N$

输出：$S\subseteq V$ 使得$\sum_{u\in S, v\in V\text{-}S} W(uv)$最大

1. $S\leftarrow\{u\}$; /*u是V中任意顶点*/
2. repeat
3. 任取使得cost(v,S)>0的顶点v，交换v在S，V-S中的位置;
4. until 不存在$v\in V$使得cost(v,S)>0
5. 输出S.

算法复杂度

- 第3步每次运行的时间开销为$O(|V|^2)$
  - ✓ 计算$cost(v,S)$的开销为$|V|$,至多为$|V|$个顶点计算
- 第2-4步至多运行$\sum_{uv\in E}W(uv)\leq|V|^2\times max_{uv\in E}W(uv)=O(|V|^2W)$
  - ✓ 循环每做一次，割的代价至少增大1
- 总的时间开销为$O(|V|^4W)$

ApproxMaxCut($G(V,E)$)
输入：加权图$G=(V,E)$，权值函数$W:E\rightarrow N$
输出：$S\subseteq V$ 使得$\sum_{u\in S,v\in V-S}W(uv)$最大
1. $S\leftarrow\{u\}$;　　　　/*u是V中任意顶点*/
2. repeat
3. 　　任取使得$cost(v,S)>0$的顶点v,交换v在$S,V-S$中的位置;
4. until 不存在$v\in V$使得$cost(v,S)>0$
5. 输出S.

HIT
CS&E

定理2：ApproxMaxCut的近似比为2。

证明：假定问题最优解为S^*，算法输出的近似解为S

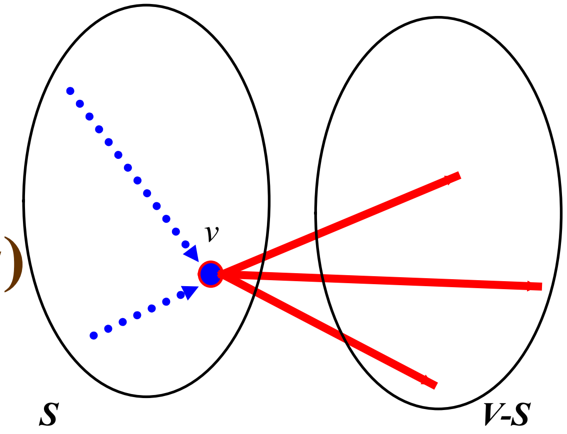
由于算法输出局部最优解，

$\forall v \in S$，若$cost(v,S) \le 0$

则有 $\sum_{u\in S} W(uv) \le \sum_{u\in V\text{-}S} W(vu)$

$\sum_{u\in S} W(uv) + \sum_{u\in V\text{-}S} W(uv) \le 2\times \sum_{u\in V\text{-}S} W(vu)$

即：$(1/2)\times \sum_{u\in V} W(uv) \le \sum_{u\in V\text{-}S} W(vu)$



同理，$\forall v \in V\text{-}S$，若$cost(v,S) \le 0$

必有：$(1/2)\times \sum_{u\in V} W(uv) \le \sum_{u\in S} W(uv)$

故：$\sum_{uv\in E} W(uv) \le 2W(S)$

从而：$W(S^*) \le \sum_{uv\in E} W(uv) \le 2W(S)$

8.4.3 设施定位问题（简介）

问题的定义

- **输入:**

  设施集合F,用户集合U, 距离函数$d:U\times \mathrm{F}\rightarrow R^{+}$

  – 距离函数满足三角不等式;
  – $\forall i\in F$,开启设施i的代价为f_i, $\forall S\subseteq F$的开启代价$C_f(S)=\sum_{i\in S} f_i$
  – $\forall j\in U$, $\forall S\subseteq F$,

    S向j提供服务的代价为$r(j,S)=\min_{i\in S} d(j,i)$

    S向U提供服务的总代价为$C_r(S)=\sum_{j\in U} r(j,S)$
  – $\forall S\subseteq F$, S的代价定义为$\mathrm{C(S)}=C_f(S)+C_r(S)$

- **输出:**

  $S\subseteq F$ 使得$C(S)$最小

*设施定位问题是很多实际问题的抽像.

*设施定位问题是NP-完全问题.

算法思想:局部修改操作

- 对任意可行解S可以施行如下三类局部操作
  - 添加——向S添加一个设施
  - 删除——从S中删除一个设施
  - 替换——将S中的一个设施i替换为另一个设施j'

算法

LocalSearchFacility(F,U,ε)

1. $S \leftarrow F$的任意子集;
2. IF 存在添加、删除或替换操作使S的代价下降因子$(1-\varepsilon/n^2)$ Then
   执行该操作
3. 重复第2步，直到不存在满足条件的操作
4. 输出S.

•该算法在多项式时间内终止,且近似比为$3+o(\varepsilon)$(参见教材)

8.5 基于动态规划的近似算法

- Rounding Data与动态规划
- 0-1背包问题的完全多项式近似模式
- 子集求和问题的完全多项式近似模式
- Bin-Packing 问题的近似模式

8.5.1 Rounding Data与动态规划

- 动态规划算法
  - 问题具有优化子结构
  - 重叠子问题
  - 用系统化的方法搜索优化子结构涉及的所有子问题
- 问题?
  - 例如：伪多项式时间复杂性算法
- 解决办法
  - 以某种方式对输入数据或解空间进行Rounding(舍入)处理，使得舍入后的数据具有多项式大小
  - 之后采用动态规划设计算法

Rounding data

- rounding策略1：
  - 原问题的实例*I*转换为一个特殊实例*I'*
  - 用动态规划方法求解实例*I'*
  - 将*I'*的解转化为*I*的近似解
  - 近似比取决于变形过程的性质
- rounding策略2
  - 将动态规划方法视为解空间的枚举过程
  - 仅枚举整个解空间的一个子空间，则得到一个近似解
  - 近似比取决于所枚举的子空间与整个空间之间的"间隙"大小

8.5.2 0-1背包问题的完全多项式近似模式

- 问题定义及其动态规划算法
- 问题的变形
- 完全多项式近似模式
- 算法性能分析

问题定义

给定n种物品和一个背包，物品i的重量是w_i，价值v_i，背包承重为C，问如何选择装入背包的物品，使装入背包中的物品的总价值最大?

对于每种物品只能选择完全装入或不装入，一个物品至多装入一次。

- 输入：$C>0$, $w_i>0$, $v_i>0$, $1\leq i\leq n$
- 输出：$(x_1, x_2, \ldots, x_n)$, $x_i\in\{0, 1\}$, 满足

$$\sum_{1\leq i\leq n} w_i x_i \leq C,\quad \sum_{1\leq i\leq n} v_i x_i \text{ 最大}$$

0-1背包问题是NP完全问题

解法一

- 递归方程：

$[\{i, i+1, \ldots, n\}, j]$

$0 \leq j < w_i,\ m(i, j) = m(i+1, j)$

$j \geq w_i, \quad m(i, j) = max\{m(i+1, j), m(i+1, j-w_i)+v_i\}$

$[\{n\}, j]$

总结：

$$m(n, j) = 0, \quad 0 \leq j < w_n$$

$$m(n, j) = v_n, \quad j \geq w_n$$

$$m(i, j) = m(i+1, j), \ 0 \leq j < w_i$$

$$m(i, j) = max\{m(i+1, j), m(i+1, j-w_i)+v_i\}, \ j \geq w_i$$

解法二

- $S_i= (1,\ldots,i)$表示前i个物品构成的集合
- $B[i,j]$表示从S_i中取物品总价值量正好为j所需的最小背包容量，其中$1\le i\le n$, $0\le j\le \sum_{1\le i\le n} v_i$
- 问题的优化子结构可以重述为：

$$
\begin{array}{ll}
B[i,j]=B[i-1,j] & j<v_i \\
B[i,j]=\min(B[i-1,j],\ B[i-1,j-v_i]+w_i) & j\ge v_i \\
B[1,j]=w_i & j=v_i \\
B[1,j]=0 & j=0 \\
B[1,j]=\infty & otherwise
\end{array}
$$

- 可得到$O(n\sum_{1\le i\le n} v_i)$时间的DP算法

Rounding data

- $V_{max}=max_{1\le i\le n}v_i$,
- 给定参数ε,令$\mu=\varepsilon V_{max}/n$ 利用μ对v_i进行rounding处理
- $\forall v_i, v'_i=\lfloor v_i/\mu \rfloor$
  - 将v_i rounding down为与其最接近的一个整数v'_i乘以μ
  - 即有： $\mu v'_i \le v_i \le \mu(v'_i+1)$
- 实例$I=\{(w_1, \ldots,w_n), (v_1\ldots,v_n)\}$变形为

  $I'=\{(w_1, \ldots,w_n), (v'_1\ldots, v'_n)\}$
- 在I'上运行DP算法(时间复杂性为$O(n^3/\varepsilon)$)
- 用I'的精确解作为I的近似解

算法ApproxPacking(W[1:n],V[1:n],C,ε)

输入：容量C,重量数组W[1:n],价值数组V[1:n],误差参数ε

输出：0-1背包问题的(1-ε)-近似解$\langle x_1,\ldots,x_n\rangle$

1. $V_{max}=max_{1\leq i\leq n}v_i$;
2. $\mu = \varepsilon V_{max}/n$;
3. For i=1 to n Do

   $V[i]\leftarrow\lfloor V[i]/\mu)\rfloor$;

4. 以(W[1:n],V[1:n],C)为输入，调用任意解0-1背包问题DP算法，得到计算结果($x_1,\ldots,x_n$)
5. 输出($x_1,\ldots,x_n$)作为原始问题的近似解

Rounding data

定理：算法ApproxPacking是一个*FPTAS*近似算法

(1) ApproxPacking是关于n和$1/\varepsilon$的多项式时间的算法

证明：

$$V' = \sum_{1\le i\le n} v'_i = \sum_{1\le i\le n} \lfloor v_i/(\varepsilon V_{max}/n) \rfloor$$
$$= (n/\varepsilon) \sum_{1\le i\le n} \lfloor v_i/V_{max}) \rfloor \le (n/\varepsilon) \sum_{1\le i\le n} v_i/V_{max}$$
$$\le n^2/\varepsilon$$

已知I'上运行DP算法的时间复杂性为$O(n\min(B,V'))$，

即为$O(n^3/\varepsilon))$，证毕

$V_{max} = max_{1\le i\le n} v_i$；

$\mu = \varepsilon V_{max}/n$；

定理：算法ApproxPacking是一个*FPTAS*近似算法

(2) (1-ε)-近似性

证：假设 $(z_1,\ldots,z_n)$是原始问题I的最优解,

I的最优解代价 $Z=\sum_{1\le i\le n}v_i z_i \ge V_{max}$;

$(x_1,\ldots,x_n)$是ApproxPacking算法返回的I'最优解,

I'的最优解代价$X'=\sum_{1\le i\le n}v'_i x_i$,

I'的最优解是I的近似解，其代价$X=\sum_{1\le i\le n}v_i x_i \le Z$

而I的最优解又是I'的近似解，其代价$Z'=\sum_{1\le i\le n}v'_i z_i \le X'$

由$\forall v_i$, $v'_i=\lfloor v_i/\mu \rfloor$可得： $\mu v'_i \le v_i \le \mu(v'_i+1)$，进而$\mu v'_i \ge v_i-\mu$

$X=\sum_{1\le i\le n}v_i x_i \ge \sum_{1\le i\le n}\mu v'_i x_i = \mu\sum_{1\le i\le n}v'_i x_i \ge \mu\sum_{1\le i\le n}v'_i z_i$

$= \sum_{1\le i\le n}\mu v'_i z_i \ge \sum_{1\le i\le n}(v_i-\mu)z_i = \sum_{1\le i\le n}v_i z_i - \sum_{1\le i\le n}\mu z_i$

$\ge \sum_{1\le i\le n}v_i z_i - n\mu$

由$\mu=\varepsilon V_{max}/n$, $X \ge \sum_{1\le i\le n}v_i z_i - \varepsilon V_{max} \ge Z-\varepsilon Z = (1-\varepsilon)Z$. 证毕

8.5.3 最大子集和问题

- 问题的定义
- 指数时间算法
- Rounding与完全多项式时间近似模式

问题定义

- 输入:

  (S, t), 满足:

  (1). $S=\{x_1, x_2, \ldots, x_n\}$, x_i是正整数;

  (2). t=正整数

- 输出:

  $A \subseteq S$, $\sum_{x \in A} x$, 满足:

  (1). $\sum_{x \in A} x \le t$

  (2). $\sum_{x \in A} x = max\{ \sum_{x \in B} x \le t \mid B \subseteq S \}$

HIT
CS&E

指数时间算法

- 算法的基本思想
  - 设S是集合, x是正整数, 定义$S+x=\{s+x \mid s \in S\}$

对n作数学归纳法:

$L_n = L_{n-1} \cup L_{n-1}+x_n$ (不大于t)

  - $L_2=<0, x_1, 0+x_2, x_1+x_2> = L_1 \cup L_1+x_2$

递归计算:

$L_0=<0>$

$L_n = L_{n-1} \cup L_{n-1}+x_n$ (不大于t)

  - L_i=前i个元素所有子集的和(不大于t) $= L_{i-1} \cup L_{i-1}+x_i$

- 指数时间算法

Exact-Subset-Sum($S=\{x_1, x_2, \ldots, x_n\}, t$)

1. $n \leftarrow |S|$;
2. $L_0 \leftarrow <0>$;
3. For $i \leftarrow 1$ To n Do
4. $\quad L_i \leftarrow$ Merge-List($L_{i-1}, L_{i-1}+x_i$);
5. $\quad$ 删除L_i中所有大于t的元素;
6. Return L_n中最大元素.

- 时间复杂性

第4步： $|L_i|=2|L_{i-1}|=2^2|L_{i-2}|=\ldots=2^i|L_0|=2^i$

$T(n)=O(2^1+2^2+\ldots+2^n)=O(2^n)$, 如果$t$比较大

1. $n \leftarrow |S|$;
2. $L_0 \leftarrow <0>$;
3. For $i \leftarrow 1$ To n Do
4. $L_i \leftarrow$ Merge-List(L_{i-1}, $L_{i-1}+x_i$);
5. 删除L_i中所有大于t的元素;
6. Return L_n中最大元素.

完全多项式时间近似模式

- 基本思想: Rounding L

修剪L, 对于多个相近元素, 只留一个代表, 尽量缩小每个L的长度

– 设$\delta(0<\delta<1)$是修剪参数, 根据δ修剪L:

(1). 从L中删除尽可能多的元素,

(2). 如果L'是L修剪后的结果, 则对每个从L中删除的元素y, L'中存在一个元素z, 使得

$$y(1-\delta) \le z \le y$$

– 如果y被修剪掉, 则存在一个代表y的z在L中, 而且z相对于y的相对误差小于δ: $(y-z)/y<\delta$

- 修剪算法

Trim($L=\{y_1, y_2, \ldots, y_m\},\delta$) /* $y_i \le y_{i+1}$, $0<\delta<1$, 输出缩小的表L'. */

$m \leftarrow |L|$;

$L' \leftarrow \langle y_1 \rangle$;

$last \leftarrow y_1$;

For $i \leftarrow 2$ To m Do /* 考察y_i是否剪裁掉 */

  If $y_i \times (1-\delta) > last$ /* 即 $y_i \times (1-\delta) \le last \le y_i$ 不成立*/

  /*即$last < y_i \times (1-\delta)$, 由$L$和$L'$有序, 对$\forall y \in L'$, 不满足$y_i \times (1-\delta) \le y \le y_i$ */

  Then y_i加入到L'末尾; /* 因L'中目前没有能够表示y_i的元素 */

    $last \leftarrow y_i$;

Return L'.

- 复杂性: $O(|L|)=O(m)$

- 完全多项式近似模式

输入：$S=\{x_1, x_2, ..., x_n\}$, $t \geq 0$, $0<\varepsilon<1$

输出：近似解z

Approx-Subset-Sum(S, t, ε)

1. $n \leftarrow |S|$;
2. $L_0 \leftarrow <0>$
3. For $i \leftarrow 1$ To n Do
4. $L_i \leftarrow$ Merge-List(L_{i-1}, $L_{i-1}+x_i$);
5. $L_i \leftarrow$ Trim(L_i, ε/n) /* 修剪参数$\delta=\varepsilon/n$ */
6. 从L_i中删除大于t的元素;
7. 令z是L_n中最大值;
8. Return z .

- 性能分析

定理1. Approx-Subset-Sum是子集求和问题的一个相对误差≤ε的完全多项式时间近似模式.

证. 令$P_0=\{0\}$, $P_i=\{x \mid x=\sum_{y\in A} y,\ A\subseteq\{x_1, x_2, \ldots, x_i\}\}$.

使用数学归纳法可以证明: $P_i=P_{i-1}\cup (P_{i-1}+x_i)$.

使用数学归纳法可以证明L_i是P_i中所有不大于t的元素的有序表.

L_i经第5步修剪以及第6步的大于t元素的删除, 仍然有$L_i\subseteq P_i$.

于是, 第8步返回的z是S的某个子集的和.

我们需证明

(1). $(C^*-z)/C^*\le\varepsilon$, 即$C^*(1-\varepsilon)\le z$, C^*是优化解, z是近似解.

注意, 由于子集合求和问题是最大化问题, $(C^*-z)/C^*$是算法的相对误差.

(2). 算法是关于$|S|$和$1/\varepsilon$的多项式时间算法.

(1). 往证$C^*(1-\varepsilon)\le z$, 即$(C^*-z)/C^*\le\varepsilon$

对i作归纳法证明: $\forall y\in P_i$, $y\le t$, 存在一个$z'\in L_i$使$(1-\varepsilon/n)^i y\le z'\le y$.

当$i=0$时$P_i=\{0\}$, $L_i=\{0\}$, 命题成立.

设当$i\le k$时命题成立. $P_{k+1}=P_k\cup\{P_k+x_{k+1}\}$.

由归纳假设, $\forall y\in P_{k+1}\cap P_k$, $y\le t$, 存在$z'\in L_k\subseteq L_{k+1}$使

$$(1-\varepsilon/n)^k y\le z'\le y.$$

于是, $(1-\varepsilon/n)^{k+1}y\le z'\le y$.

对于$\forall y'\in P_{k+1}-P_k$, $y'=y+x_{k+1}\le t$, $y\in P_k$. 由归纳假设, 存在$z'\in L_k\subseteq L_{k+1}$使$(1-\varepsilon/n)^k y\le z'\le y$. 于是,

$$(1-\varepsilon/n)^k y+x_{k+1}\le z'+x_{k+1}\le y+x_{k+1}.$$

由于$z'\in L_k$, 若 $z'+x_{k+1}\in L_{k+1}$, 即$z'+x_{k+1}$没有被修剪掉, 由

$$((1-\varepsilon/n)^k y+x_{k+1})-((1-\varepsilon/n)^{k+1}(y+x_{k+1}))$$
$$=(1-\varepsilon/n)^k(y-(1-\varepsilon/n)y)+(x_{k+1}-(1-\varepsilon/n)^{k+1}x_{k+1})>0,$$

即$(1-\varepsilon/n)^{k+1}(y+x_{k+1})\le z'+x_{k+1}\le y+x_{k+1}$.

HIT
CS&E

$$(1-\varepsilon/n)^k y+x_{k+1} \le z'+x_{k+1} \le y+x_{k+1}.$$

如果 $z'+x_{k+1} \notin L_{k+1}$，即 $z'+x_{k+1}$ 被剪掉，则必存在 $z'' \in L_{k+1}$，使得

$$(z'+x_{k+1}) \times (1-\varepsilon/n) \le z'' \le z'+x_{k+1}.$$

于是，$((1-\varepsilon/n)^k y+x_{k+1}) \times (1-\varepsilon/n) \le z'' \le y+x_{k+1}.$

$$((1-\varepsilon/n)^k y+x_{k+1}) \times (1-\varepsilon/n)-((1-\varepsilon/n)^{k+1}(y+x_{k+1}))$$

$$=(1-\varepsilon/n)^{k+1} y+x_{k+1} \times (1-\varepsilon/n) - ((1-\varepsilon/n)^{k+1}(y+x_{k+1}))$$

$$= x_{k+1}\,(1-\varepsilon/n) \times (1-(1-\varepsilon/n)^k)$$

>0，因为 $(1-\varepsilon/n)>0$，$1-(1-\varepsilon/n)^k>0$.

即 $(1-\varepsilon/n)^{k+1}(y+x_{k+1}) \le z'' \le y+x_{k+1}$.

$C^* \in P_n$是子集合求和问题的优化解, 则存在一个$z' \in L_n$, 使

$$(1-\varepsilon/n)^n C^* \le z' \le C^*.$$

因算法解$z=\max(L_n)$, $(1-\varepsilon/n)^n C^* \le z' \le z \le C^*$.

由于$(1-\varepsilon/n)^n$的一阶导数大于0, $(1-\varepsilon/n)^n$是关于n递增的函数.

因为$n \ge 1$, $(1-\varepsilon) \le (1-\varepsilon/n)^n$.

于是, $(1-\varepsilon)C^* \le z$, 即近似解z与优化解的相对误差不大于ε.

HIT
CS&E

(2). 往证算法的时间复杂性是n与$1/\varepsilon$的多项式

先计算$|L_i|$的上界. 修剪后, L_i中的相邻元素y_{j-1}和y_j满足:

$$y_{j-1}<(1-\varepsilon/n)y_j,\ \text{即}\ y_j/y_{j-1}>1/(1-\varepsilon/n),\ y_j>y_{j-1}\,1/(1-\varepsilon/n).$$

如果$L_i=\{y_0, \ldots, y_{k+1}\}$, 则必有

$$y_0=0,\ y_1=z_0,\ y_2>z_0\cdot 1/(1-\varepsilon/n),\ y_3>z_0\cdot 1/(1-\varepsilon/n)^2, \ldots,\ y_{k+1}>z_0\cdot 1/(1-\varepsilon/n)^k$$

而且$z_0\cdot 1/(1-\varepsilon/n)^k\leq t$.

由$z_0\cdot 1/(1-\varepsilon/n)^k\leq t$, $k\leq \log_{1/(1-\varepsilon/n)}(t/z_0)\leq \log_{1/(1-\varepsilon/n)}t$,

$$|L_i|=k+2\leq 2+\log_{1/(1-\varepsilon/n)}t$$

对$\log_{1/(1-\varepsilon/n)}t$换底, 并展开$\ln(1-\varepsilon/n)$, $x/(x-1)\leq\ln(1-x)\leq -x$

$$|L_i|\leq 2+\log_{1/(1-\varepsilon/n)}t=2+(\ln t\ /-\ln(1-\varepsilon/n))\leq 2+n\ln t/\varepsilon.$$

算法的运行时间是$|L_i|$的多项式, 即n和$1/\varepsilon$的多项式.

8.5.4 装箱问题的近似模式

- 问题定义
- 对问题进行Rounding变形
- 多项式近似模式

问题的定义

- 输入

  体积依次为$a_1,...,a_m \in (0,1]$的m个物品

  无穷个体积为1的箱子

- 输出

  物品的一个装箱方案，使得使用的箱子数量最少

Rounding策略：

- 小体积物品对优化解的影响较小
  - 多个小体积的物品可以容纳在少数箱子中
- 可以先忽略小体积物品得到实例I^{up}
  - 缩小问题的解空间
  - 实例I^{up}的优化解具有某种优良的性质
- 在I^{up}上用动态规划算法得到精确解s'
- 将s'调整为I的近似解s

算法ApproxBinPacking(I,ε)

输入:装箱问题的实例I和相对误差参数ε<1

输出:I的一个近似最优的装箱方案;

- I', I^{down}, $I^{up}$$\leftarrow$Transfrom($I$, ε); /*变换*/
- $S'$$\leftarrow$DynamicSearch($I^{up}$, $1/\varepsilon^2$); /*DP*/
- $S$$\leftarrow$SolutionTrans($S'$,$I$,$\varepsilon$); /*得到近似解*/
- 输出S;

下面依次介绍第1,2,3步,实现复杂度为$O(m^{1/\varepsilon^2})$的算法

HIT

实例变化算法Transform(I,ε)

输入：装箱问题的实例I和变换参数ε

输出：变形后的三个实例I', I^{down}和I^{up}

Steps:

1. 删除I中所有体积小于ε的物品，得到I'，记$n=|I'|$;
2. 将I'中所有物品按体积大小递增排序，划分为 $K=1/\varepsilon^2$组，每组$n/K=n\varepsilon^2$个物品；
3. 将各组内物品的体积修改为组内最大体积，得I^{up};
4. 将各组内物品的体积修改为组内最小体积，得I^{down};
5. 输出I', I^{down}和I^{up};

Transform(I,ε)的时间复杂度为$O(n\log n) \le O(m\log m)$

HIT
CS&E

举例：

I={0.1，0.05，0.3，0.55，0.45，0.7，0.15，0.8，0.64，0.2，0.4，0.6，0.85，0.5，0.46，0.67，0.75}，

$\varepsilon=0.4$

从I中删除体积小于ε的物品，并按体积递增顺序排序

I'={0.4, 0.45, 0.5, 0.55, 0.6, 0.64, 0.67, 0.7, 0.75, 0.8, 0.85}

将I'划分为$K=1/\varepsilon^2=6$组

{<u>0.4, 0.45</u>, <u>0.5, 0.55</u>, <u>0.6, 0.64</u>, <u>0.67, 0.7</u>, <u>0.75, 0.8</u>, <u>0.85}</u>

I^{up}={0.<u>45, 0.45</u>, 0.<u>55, 0.55</u>, 0.<u>64, 0.64</u>, <u>0.7, 0.7</u>, <u>0.8, 0.8</u>, <u>0.85</u>}

I^{down}={<u>0.4, 0.4</u>, <u>0.5, 0.5</u>, <u>0.6, 0.6</u>, <u>0.67, 0.67</u>, <u>0.75, 0.75</u>, <u>0.85}</u>

引理1：I'、I^{down}和I^{up}满足下列性质

(1)每个箱子至多容纳I',I^{down}和I^{up}的$L=\lceil 1/\varepsilon \rceil$个物品；

(2)I^{down}和I^{up}中物品体积至多有$K=1/\varepsilon^2$个不同的取值

(3)$Opt(I^{down})\leq Opt(I')$,且$n\varepsilon\leq Opt(I')$,其中$n$是$I'$中物品个数；

证明：

I'的最优解是I^{down}的可行解

I'的每个可行解也是I^{down}的可行解

故有：$Opt(I^{down})\leq Opt(I')$

又因所有物品的总体积至少为$n\varepsilon$

因此，$n\varepsilon\leq Opt(I')$

I'={0.4，0.45， 0.5，0.55，0.6， 0.64，0.67，0.7，0.75，0.8，0.85}

最优解9个箱子

I^{down}={0.4, 0.4, 0.5, 0.5, 0.6, 0.6, 0.67, 0.67, 0.75, 0.75, 0.85}

最优解8个箱子

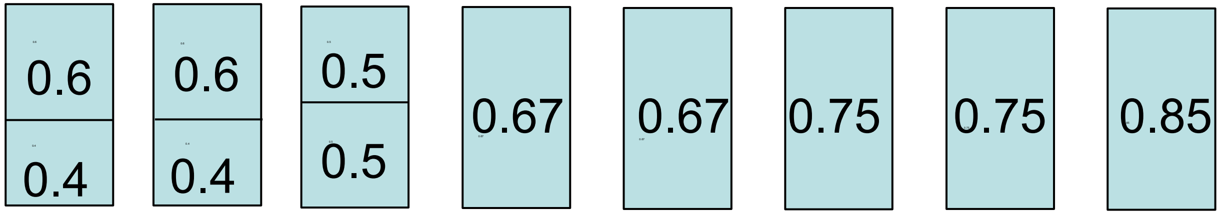
I^{up}={0.45, 0.45, 0.55, 0.55, 0.64, 0.64, 0.7, 0.7, 0.8, 0.8, 0.85}

得到的I^{up}的可行解S最坏情况下需要的箱子数：

1. 第一步中删除所有属于I^{down}第一组的物品时，箱子的个数没有发生变化（即没有减少）

I^{up}最后剩余每个物品用一个新箱子，最后剩余最多有$n\varepsilon^2$个物品，最坏情况下需要的新箱子个数为$n\varepsilon^2$。因此，可行解S最坏情况下需要的箱子个数$\leq \mathrm{OPT}(I^{down})+n\varepsilon^2$.

I^{down}的最优解S'：



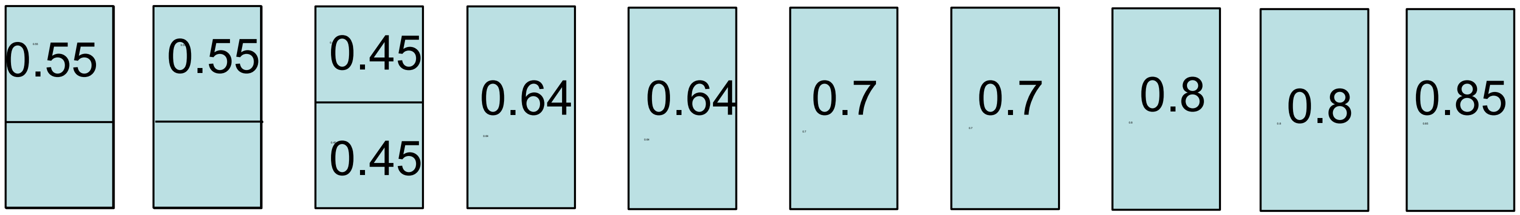
由I^{down}的最优解S'构造I^{up}的一个可行解S：

S'代价与S代价关系？

step1: 删除S'中所有属于I^{down}第1组的物品

step2: 对于$i\geq 2$,将属于I^{down}第i组的物品替换为I^{up}第i-1组的物品

step3: I^{up}最后每个物品用一个新箱子



HIT

引理1(续):I',I^{down}和I^{up}满足下列性质

(3)$Opt(I^{down})\le Opt(I')$,且$n\varepsilon\le Opt(I')$,其中n是I'中物品个数;

(4)$Opt(I^{up})\le(1+\varepsilon)Opt(I')$;

证明:

令S'是I^{down}的优化解,对S'进行如下变换,从而得到I^{up}的一个可行解S:

step1: 删除S'中属于I^{down}第1组的物品

step2: 对于$i\ge 2$,将属于I^{down}第i组的物品替换为I^{up}第i-1组的物品

step3: I^{up}最后剩余每个物品用一个新箱子,至多新增$n\varepsilon^2$箱子

$$\mathrm{Opt}(I^{up}) \le \mathrm{Opt}(I^{down}) + n\varepsilon^2 \le \mathrm{Opt}(I') + n\varepsilon^2 \quad \text{(引理1性质3可知)}$$

$$\le \mathrm{Opt}(I') + \varepsilon\mathrm{Opt}(I') = (1+\varepsilon)Opt(I')$$

- 用动态规划算法求解问题实例I^{up}
  - n个物品，
  - 体积至多有$K=1/\varepsilon^2$个不同取值$s_1,\ldots,s_K$
  - 请同学们自己描述问题的优化子结构并实现算法DynamicSearch(I^{up},$1/\varepsilon^2$), 要求时间复杂度为$O(Kn^K)=O(n^{1/\varepsilon^2})\leq O(m^{1/\varepsilon^2})$

解转换算法SolutionTrans(S,I,ε)

输入:实例I，I中体积大于ε的物品的近似装箱方案S

输出:I的一个近似解

1. For I中体积小于ε的每个物品i Do
2. If S中存在箱子能容纳物品i Then 将i装入该箱子
3. Else 开启新箱子将i装入，将更新后的方案仍记为S;
4. 输出更新后的装箱方案S;

SolutionTrans(I,ε)的时间复杂度为$O(m^2)$

定理：ApproxBinPacking是一个$1+2\varepsilon$-近似算法

证明：设SolutionTrans新开的箱子个数记为*new*，则

$$\text{Approx}(I) = Opt(I^{up})+new$$

若$new=0$，则

$\text{Approx}(I) = Opt(I^{up})+new \le (1+\varepsilon)\, Opt(I') \le (1+2\varepsilon)Opt(I)$

若$new\neq 0$，则

近似解$\text{Approx}(I)$个箱子中，除最后一个箱子之外，每个箱子的空闲空间都小于ε，即每个箱子所装物品体积$>1-\varepsilon$。

$(1-\varepsilon)[\text{Approx}(I) -1]<$ 前$\text{Approx}(I) -1$个箱子内物品总体积

<所有物品总体积

又由于所有物品总体积$\le Opt(I)$

故有：$(1-\varepsilon)[\text{Approx}(I) -1] \le Opt(I)$

$\text{Approx}(I) \le [1/(1-\varepsilon)]Opt(I) +1$

$\le (1+2\varepsilon)Opt(I) +1$ /* $1/(1-\varepsilon)\le 1+2\varepsilon$, 当$\varepsilon\le 0.5$时 */

得证。

8.6 基于线性规划的近似算法

- 线性规划
- 舍入法
- Primal-dual Schema

8.6.1 线性规划概念

- 线性规划问题
  - 是指在线性不等式约束下使一个线性目标函数达到最优(最小或最大)的问题
- 线性规划的标准形式

最小化问题

$$\text{Min} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{St} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad i=1,\ldots,m$$

$$x_j \geq 0 \quad j=1,\ldots,n$$

最大化问题

$$\text{Max} \quad \sum_{i=1}^{m} b_i y_i$$

$$\text{St} \quad \sum_{i=1}^{m} a_{ij} y_i \leq c_j \quad j=1,\ldots,n$$

$$y_i \geq 0 \quad i=1,\ldots,m$$

例

$$\text{Min} \quad 7x_1+x_2+5x_3$$

$$\text{St.} \quad x_1-x_2+3x_3 \geq 10$$

$$5x_1+2x_2-x_3 \geq 6$$

$$x_1, x_2, x_3 \geq 0$$

$$\text{Max} \quad 10y_1+6y_2$$

$$\text{St.} \quad y_1+5y_2 \leq 7$$

$$-y_1+2y_2 \leq 1$$

$$3y_1-y_2 \leq 5$$

$$y_1, y_2 \geq 0$$

线性规划

满足所有约束条件的一组变量称为线性规划问题的**可行解**

使得目标函数达到最优取值的可行解称为线性规划问题的**最优解**

例 minimize $7x_1+x_2+5x_3$

subject to $x_1-x_2+3x_3\geq 10$

$5x_1+2x_2-x_3\geq 6$

$x_1,x_2,x_3\geq 0$

$x=(2,1,3)$是上述线性规划问题的一个可行解;$x=(7/4,0,11/4)$是上述线性规划问题的最优解，目标函数的最优值为26.

线性规划问题可以在多项式时间内求解：Karmarkar算法

HIT

很多组合优化问题可以表达成整数线性规划问题

例如：最小节点覆盖问题

输入：无向图$G=(V, E)$, 每个节点具有权$w(v)$.

输出：$C\subseteq V$, 满足

(1). $\forall(u, v)\in E$, $u\in C$或者$v\in C$

(2). $w(C)$最小, $w(C)=\sum_{c\in C}w(c)$.

对于$\forall v\in V$, 定义$x(v)\in\{0, 1\}$如下：

若v在节点覆盖中, 则$x(v)=1$, 否则$x(v)=0$.

$\forall(u, v)\in E$, 若u、v或两者在覆盖中, 则$x(u)+x(v)\geq 1$.

对应的0-1整数规划问题$ILP_{0\text{-}1}$

优化目标：最小化$\sum_{v\in V}w(v)x(v)$

约束条件：$x(u)+x(v)\geq 1 \quad \text{for } \forall(u, v)\in E$

$x(v)\in\{0, 1\} \quad \text{for } \forall v\in V$

0-1整数规划问题ILP_{0-1}

优化目标：最小化 $\sum_{v\in V} w(v)x(v)$

约束条件：$x(u)+x(v)\geq 1$ for $\forall(u, v)\in E$

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

LP-松弛！

整数线性规划问题是NP-C问题，如何近似求解？

将整数约束条件放宽，即得到一个线性规划问题

ILP_{0-1}对应的线性规划问题LP

- 优化目标：最小化 $\sum_{v\in V} w(v)x(v)$
- 约束条件：$x(u)+x(v)\geq 1$ for $\forall(u, v)\in E$

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

HIT
CS&E

如何将线性规划问题的解，变成整数得到原问题的一个近似解？

方法1：舍入法　　保证舍入得到的近似解代价不会大幅度增加

方法2：primal-dual schema

构造LP-松弛问题的一个整数可行解x作为输出

构造LP-松弛问题的对偶问题的可行解z

比较上述两个解的代价可以得到近似比的界限

两种方法的主要区别在于运行时间，第一种方法需要精确求解线性规划，而第二种不需要。此外，由第二种方法得到的算法可能能够转换成组合优化算法。

8.6.2 舍入法

- 最小节点覆盖问题的线性规划算法
- 加权集合覆盖问题的线性规划算法

- 最小节点覆盖问题的线性规划算法

最小节点覆盖问题

问题定义

- 输入：无向图$G=(V, E)$，每个节点具有权$w(v)$.
- 输出：$C \subseteq V$，满足

  (1). $\forall (u, v) \in E$, $u \in C$ 或者 $v \in C$

  (2). $w(C)$最小，$w(C)=\sum_{c \in C} w(c)$.

以前的节点覆盖算法不再适用！

- 问题转化为*0-1*整数规划问题$ILP_{0\text{-}1}$
  - 对于$\forall v \in V$, 定义$x(v) \in \{0, 1\}$如下:
    - 若v在节点覆盖中, 则$x(v)=1$, 否则$x(v)=0$.
    - $\forall (u, v) \in E$, 若u、v或两者在覆盖中, 则$x(u)+x(v) \geq 1$.
  - 对应的*0-1*整数规划问题$ILP_{0\text{-}1}$

$$
\begin{aligned}
&\text{Min} && \textstyle\sum_{v\in V} w(v)x(v) \\
&\text{St.}: && x(u)+x(v) \geq 1 \quad \text{for } \forall (u, v) \in E \\
& && x(v) \in \{0, 1\} \quad \text{for } \forall v \in V
\end{aligned}
$$

  - *0-1*整数规划问题是NP-完全问题
  - 我们需要设计近似算法

- 用线性规划问题的解近似0-1整数规划问题的解
  - 对于 $\forall v \in V$, 定义 $x(v) \in [0, 1]$
  - $ILP_{0\text{-}1}$对应的线性规划问题LP
    - 优化目标: 最小化 $\sum_{v \in V} w(v)x(v)$
    - 约束条件: $x(u)+x(v) \geq 1 \quad \text{for } \forall (u, v) \in E$

      $x(v) \in [0, 1] \quad \text{for } \forall v \in V$
  - 线性规划问题具有多项式时间算法
  - $ILP_{0\text{-}1}$的可能解是LP问题的可能解
  - $ILP_{0\text{-}1}$最优解的代价$\geq LP$的最优解的代价

Approx-Min-VC(G, w)

1. $C=0$;
2. 计算LP问题的优化解x;
3. For each $v \in V$ Do
4. If $x(v) \geq 1/2$ Then $C=C \cup \{v\}$;

   /* 用四舍五入法把LP的解近似为$P_{0\text{-}1}$的解 */
5. Return C.

定理. Approx-Min-VC是一个多项式时间2-近似算法

证.

由于求解LP需多项式时间, Approx-Min-VC的For循环需要多项式时间, 所以算法需要多项式时间.

下边证明Approx-Min-VC的近似比是2.

往证算法产生的C是一个节点覆盖.

$\forall (u, v) \in E$, 由约束条件可知 $x(u)+x(v) \geq 1$. 于是, $x(u)$和$x(v)$至少一个大于等于$1/2$, 即u、v或两者在C中. C是一个覆盖.

往证 $w(C)/w(C^*) \le 2$.

令 C^* 是 $P_{0\text{-}1}$ 的优化解, z^* 是 LP 优化解的代价. 因为 C^* 是 LP 的可能解, $w(C^*) \ge z^*$.

$$z^* = \sum_{v\in V} w(v)x(v) \ge \sum_{v\in V:\, x(v)\ge 1/2} w(v)x(v)$$

$$\ge \sum_{v\in V:\, x(v)\ge 1/2} w(v)1/2$$

$$= (1/2) \sum_{v\in C} w(v)$$

$$= (1/2)w(C).$$

由 $w(C^*) \ge z^*$, $w(C^*) \ge (1/2)w(C)$, 即 $w(C)/w(c^*) \le 2$.

- 加权集合覆盖问题的线性规划算法

问题的定义

- **输入：**

  有限集X, X的子集合族F, $X=\cup_{S\in F} S$,

  $\forall S\in F$具有非负权值$w(S)$。

- **输出：**

  $C\subseteq F$, 满足

  (1). $X=\cup_{S\in C} S$,

  (2). $\sum_{S\in C} w(S)$最小.

以前的集合覆盖算法不再适用!

- 问题转化为0-1整数规划问题$ILP_{0\text{-}1}$
  - 对于$\forall S \in F$, 定义$x_S \in \{0, 1\}$如下:
    - 若S在集合覆盖C中, 则$x_S = 1$, 否则$x_S = 0$.
    - $X = \cup_{S \in C} S$意味着：对于$\forall e \in X$，集族F中至少有一个包含元素e的子集在结果集中，即：

$$\sum_{S \in F : e \in S} x_S \geq 1$$

  - 对应的0-1整数规划问题$ILP_{0\text{-}1}$

$$\begin{aligned} \textbf{Min} \quad & \textstyle\sum_{S \in F} w(S)\, x_S \\ \textbf{St.} \quad & \sum_{S \in F : e \in S} x_S \geq 1 \quad \forall e \in X \\ & x_S \in \{0, 1\} \quad \textbf{for } \forall S \in F \end{aligned}$$

- 用线性规划问题的解近似*0-1*整数规划问题的解
  - 对于 $\forall S \in F$, 定义 $x_S \in [0, 1]$
  - ILP_{0-1}对应的线性规划问题LP

$$\text{Min} \quad \sum_{S \in F} w(S)\, x_S$$

$$\text{St.}: \sum_{S \in F: e \in S} x_S \geq 1 \qquad \forall e \in X$$

$$x_S \in [0, 1] \quad \text{for } \forall S \in F$$

线性规划问题具有多项式时间算法

  - ILP_{0-1}的可能解是LP问题的可能解
  - ILP_{0-1}最优解的代价$\geq LP$的最优解的代价

Approx-Set-Cover(X,F, w)

1. $C=0$;
2. 计算LP问题的优化解x;
3. **For each** $S\in F$ **Do**
4. **If** $x_S\geq 1/f$ **Then** $C=C\cup\{S\}$;
   /* f是X的元素在集族F中的最大频率*/
5. **Return** C.

定理. Approx-Set-Cover是f-近似算法

证.

首先证算法产生的C是一个集合覆盖.

$\forall e \in X$，由于e的频率不超过f，故F中至多有f个集合S满足$e \in S$；

由于x是LP的优化解，则x必满足：$\sum_{S \in F: e \in S} x_S \geq 1$

因此，必存在$S_i \in F$使得$e \in S_i$且$x_{S_i} \geq 1/f$，即：$S_i \in C$.

进而，由e的任意性可知：$X = \cup_{S \in C} S$

往证 $w(C)/w(C^*)\leq f$.

令 C 是 $ILP_{0\text{-}1}$ 的近似解，C^* 是 $ILP_{0\text{-}1}$ 的优化解，z^* 是 LP 优化解的代价. 因为 C^* 是 LP 的可能解，$w(C^*)\geq z^*$.

$$
\begin{aligned}
w(C) &= \textstyle\sum_{S\in C} w(S) \\
&= f\times \textstyle\sum_{S\in C} w(S)/f \\
&\leq f\times (\textstyle\sum_{S\in C} w(S)\times x_S+\sum_{S\in F\text{-}C} w(S)\times x_S) \\
&= f\times z^* \\
&\leq f\times w(C^*)
\end{aligned}
$$

即 $w(C)/w(C^*)\leq f$.

8.6.3 Primal-dual Schema

- 线性规划对偶定理
- Min-Max关系和线性规划对偶
- 基于Primal-dual Schema的近似算法

线性规划对偶定理

给定如下最小化问题，如何获得问题最优解的下界?

例如：

$$\text{minimize } 7x_1+x_2+5x_3$$
$$\text{subject to } x_1-x_2+3x_3\geq 10$$
$$5x_1+2x_2-x_3 \geq 6$$
$$x_1,x_2,x_3 \geq 0$$

由第1条约束可得到一个下界：

$$7x_1+x_2+5x_3 \geq x_1-x_2+3x_3 \geq 10$$

结合第1、2条约束可得到一个更好的下界：

$$7x_1+x_2+5x_3 \geq (x_1-x_2+3x_3)+ (5x_1+2x_2-x_3)\geq 10+6=16$$

再进一步，则得到一个更紧的下界：

$$7x_1+x_2+5x_3 \geq 2(x_1-x_2+3x_3)+ (5x_1+2x_2-x_3)\geq 20+6=26$$

原始、对偶问题

给定如下最小化问题，如何获得问题最优解的下界？（无需解题）

$$\text{minimize } \sum_j c_j x_j$$

$$\text{subject to } \sum_j a_{ij} x_j \geq b_i, \quad 1 \leq i \leq m$$

$$x_j \geq 0, \quad 1 \leq j \leq n$$

用一个非负因子乘上述约束条件左右表达式，得到一个新的约束，其满足原始问题的所有可行解：

$$\sum_i y_i \left(\sum_j a_{ij} x_j\right) \geq \sum_i y_i b_i, \quad 1 \leq i \leq m$$

若$\sum_j c_j x_j \geq \sum_i y_i \left(\sum_j a_{ij} x_j\right)$成立，则得到原始问题优化解的下界：$\sum_i y_i b_i$

原始、对偶问题

举例：

对每个约束找到一个合适的非负乘积因子，使得当对这些约束求和时，在和中每个x_i的系数均被目标函数中对应的系数所控制

假定例子中两个约束的非负乘积因子分别为y_1和y_2，则有：

$$7x_1+x_2+5x_3 \ge (x_1-x_2+3x_3)y_1 + (5x_1+2x_2-x_3)y_2 \ge 10y_1+6y_2$$

从而得到一个新的线性规划：

对偶问题

Max $\quad 10y_1+6y_2$

St.

$$y_1+5y_2 \le 7$$
$$-y_1+2y_2 \le 1$$
$$3y_1-y_2 \le 5$$
$$y_1, y_2 \ge 0$$

原始问题

Min $7x_1+x_2+5x_3$

St.

$$x_1-x_2+3x_3 \ge 10$$
$$5x_1+2x_2-x_3 \ge 6$$
$$x_1, x_2, x_3 \ge 0$$

对偶最优＝原始最优

0 　对偶问题解　 26 　原始问题解 　∞

对于一般的线性规划问题

最小化问题

$$\text{Min} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{St} \quad \sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad i=1,\ldots,m$$

$$x_j \geq 0 \quad j=1,\ldots,n$$

最大化问题

$$\text{Max} \quad \sum_{i=1}^{m} b_i y_i$$

$$\text{St} \quad \sum_{i=1}^{m} a_{ij} y_i \leq c_j \quad j=1,\ldots,n$$

$$y_i \geq 0 \quad i=1,\ldots,m$$

原始问题

一组

机械

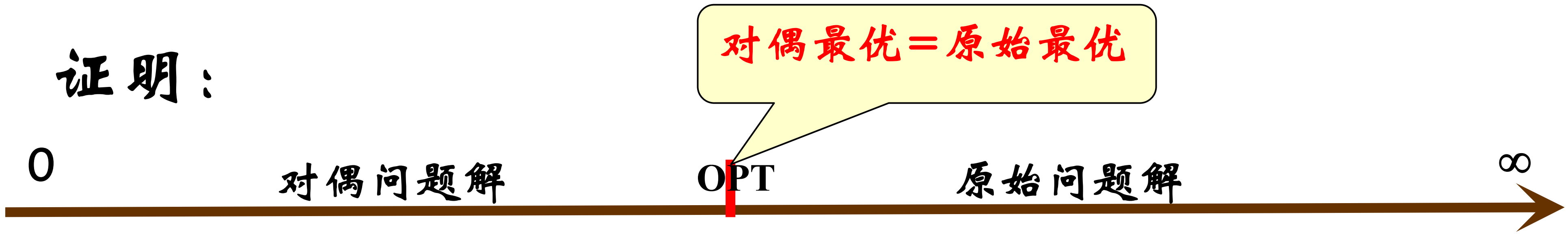
步骤

对偶问题

对偶定理. 在线性规划问题中，原问题的最优值有限当且仅当对偶问题的最优值有限。并且，如果 $x^*=(x_1^*,\ldots,x_n^*)$ 和 $y^*=(y_1^*,\ldots,y_m^*)$ 分别是原问题和对偶问题的最优解，则 $cx^*=b^Ty^*$。

弱对偶定理. 在线性规划问题中，如果 $x=(x_1,\ldots,x_n)$ 和 $y=(y_1,\ldots,y_m)$ 分别是原问题和对偶问题的可行解，则 $cx\geq by$。

证明：



原始问题

Min $\sum_{j=1}^{n} c_j x_j$

St $\sum_{j=1}^{n} a_{ij} x_j \geq b_i \quad i=1,\ldots,m$

$x_j \geq 0 \quad j=1,\ldots,n$

对偶问题

Max $\sum_{i=1}^{m} b_i y_i$

St $\sum_{i=1}^{m} a_{ij} y_i \leq c_j \quad j=1,\ldots,n$

$y_i \geq 0 \quad i=1,\ldots,m$

由于y是对偶问题的可行解，且x_j非负

$$\sum_{i=1}^{m} a_{ij} y_i \leq c_j \Rightarrow \sum_{j=1}^{n} c_j x_j \geq \sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i\right) x_j$$

由于x是原始问题的可行解，且y_i非负

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i \Rightarrow \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j\right) y_i \geq \sum_{i=1}^{m} b_i y_i$$

又有：
$$\sum_{j=1}^{n} \left(\sum_{i=1}^{m} a_{ij} y_i\right) x_j = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij} x_j\right) y_i$$

证毕

HIT

互补松弛条件定理. 如果$x=(x_1,\ldots,x_n)$和$y=(y_1,\ldots,y_m)$分别是原始问题和对偶问题的可行解，则x和y分别是原始问题和对偶问题的最优解当且仅当下面的条件同时成立：

原始问题的互补松弛条件：对于$1\le j\le n$: $x_j=0$或者$\sum_{i=1}^{m} a_{ij}y_i=c_j$

对偶问题的互补松弛条件：对于$1\le i\le m$: $y_i=0$或者$\sum_{j=1}^{n} a_{ij}x_j=b_i$

证明⇒： x和y分别是原始问题和对偶问题的可行解，

$$\sum_{j=1}^{n} c_j x_j \ge \sum_{j=1}^{n}\left(\sum_{i=1}^{m} a_{ij}y_i\right)x_j = \sum_{i=1}^{m}\left(\sum_{j=1}^{n} a_{ij}x_j\right)y_i \ge \sum_{i=1}^{m} b_i y_i$$

若x和y分别是最优解，则：$\sum_{j=1}^{n} c_j x_j = \sum_{i=1}^{m} b_i y_i$

即：$$\sum_{j=1}^{n} c_j x_j = \sum_{j=1}^{n}\left(\sum_{i=1}^{m} a_{ij}y_i\right)x_j \qquad \sum_{i=1}^{m}\left(\sum_{j=1}^{n} a_{ij}x_j\right)y_i = \sum_{i=1}^{m} b_i y_i$$

HIT

互补松弛条件定理. 如果$x=(x_1,\ldots,x_n)$和$y=(y_1,\ldots,y_m)$分别是原始问题和对偶问题的可行解，则x和y分别是原始问题和对偶问题的最优解当且仅当下面的条件同时成立：

原始问题的互补松弛条件：对于$1\le j\le n$: $x_j=0$或者$\sum_{i=1}^{m}a_{ij}y_i=c_j$

对偶问题的互补松弛条件：对于$1\le i\le m$: $y_i=0$或者$\sum_{j=1}^{n}a_{ij}x_j=b_i$

证明$\Leftarrow$： x和y分别是原始问题和对偶问题的可行解，

$$\sum_{j=1}^{n}c_jx_j \ge \sum_{j=1}^{n}\left(\sum_{i=1}^{m}a_{ij}y_i\right)x_j = \sum_{i=1}^{m}\left(\sum_{j=1}^{n}a_{ij}x_j\right)y_i \ge \sum_{i=1}^{m}b_iy_i$$

若互补松弛条件成立，则：

$$\sum_{j=1}^{n}c_jx_j = \sum_{j=1}^{n}\left(\sum_{i=1}^{m}a_{ij}y_i\right)x_j \qquad \sum_{i=1}^{m}\left(\sum_{j=1}^{n}a_{ij}x_j\right)y_i = \sum_{i=1}^{m}b_iy_i$$

即： $\sum_{j=1}^{n}c_jx_j = \sum_{i=1}^{m}b_iy_i$ x和y分别是最优解

HIT

命题1. 如果$x=(x_1,\ldots,x_n)$和$y=(y_1,\ldots,y_m)$分别是原问题和对偶问题的可行解，且满足

原问题的互补松弛条件：$\alpha\geq 1$

对于 $1\le j\le n$：$x_j=0$或者 $\frac{c_j}{\alpha}\le\sum_{i=1}^{m}a_{ij}y_i\le c_j$

对偶问题的互补松弛条件：$\beta\geq 1$

对于 $1\le i\le m$：$y_i=0$或者 $b_i\le\sum_{j=1}^{n}a_{ij}x_j\le\beta\bullet b_i$

则 $\sum_{j=1}^{n}c_jx_j\le\alpha\bullet\beta\bullet\sum_{i=1}^{m}b_iy_i$

证明：$\sum_{j=1}^{n}c_jx_j\le\alpha\sum_{j=1}^{n}(\sum_{i=1}^{m}a_{ij}y_i)x_j=\alpha\sum_{i=1}^{m}(\sum_{j=1}^{n}a_{ij}x_j)y_i\le\alpha\bullet\beta\bullet\sum_{i=1}^{m}b_iy_i$

基于Primal-dual schema的近似算法以该命题为理论基础

- **Min-Max关系和线性规划对偶**

HIT
CS&E

网络最大流问题

输入：有向图$G=(V,E)$，源点$s\in V$, 汇点$t\in V$,

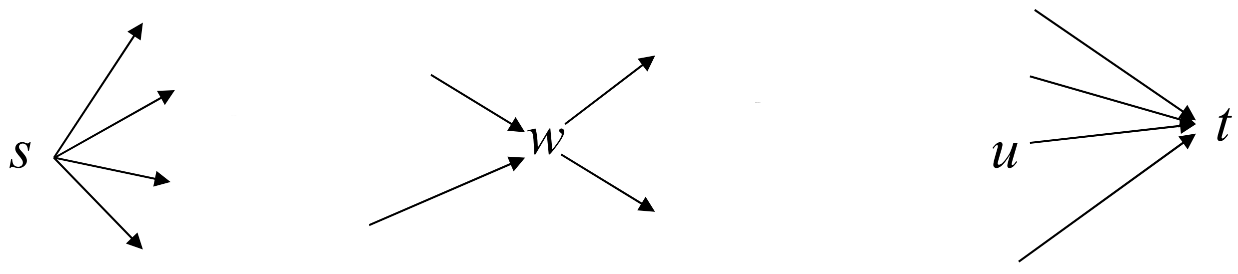
每条边e的容量限制$c(e)>0$.

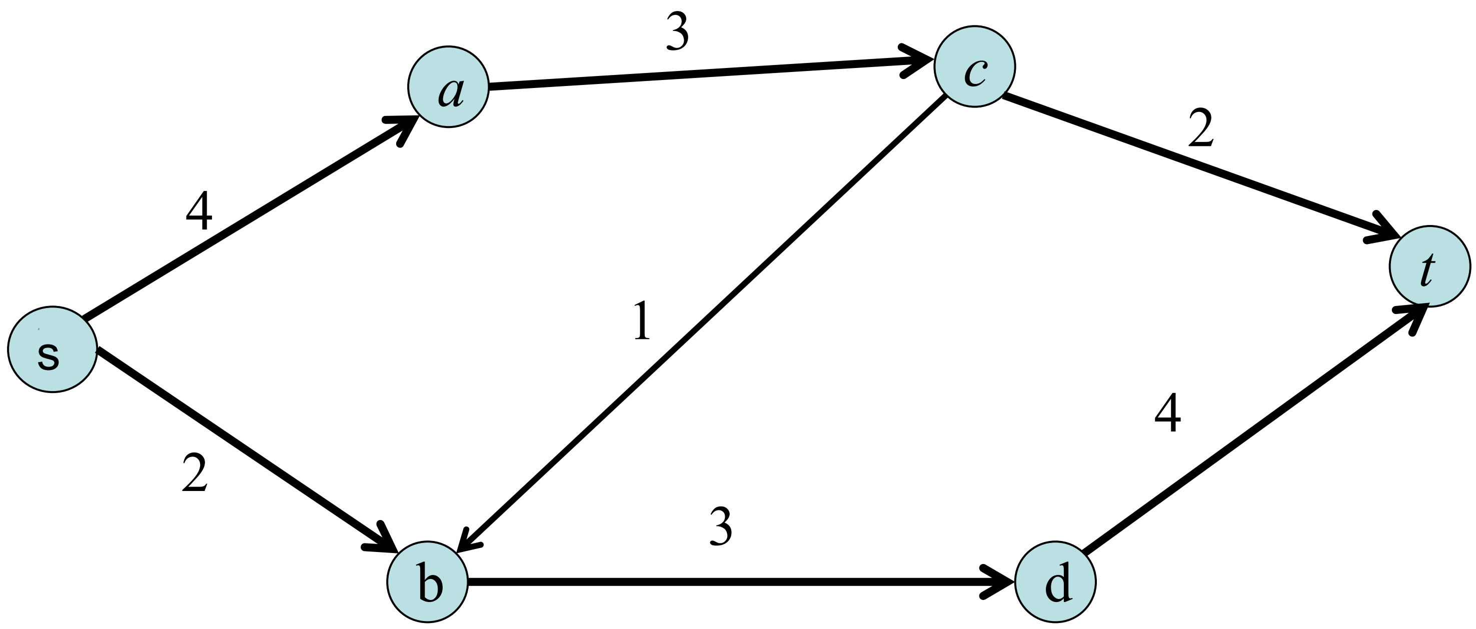
输出：从s到t的最大流。

即，对每条边e赋值$f(e)$使得$\sum_{ut\in E} f(ut)$最大且满足

流量约束: $f(e)<c(e)$

守恒约束：$\sum_{vw\in E} f(vw) = \sum_{wv'\in E} f(wv')$　　$w\in V$　$w\neq s$, $w\neq t$





最大流为5

s-t最小割问题

输入：有向图$G=(V,E)$，源点$s\in V$,汇点$t\in V$, 每条边e的流量限制$c(e)>0$.

输出：s和t之间的最小割。

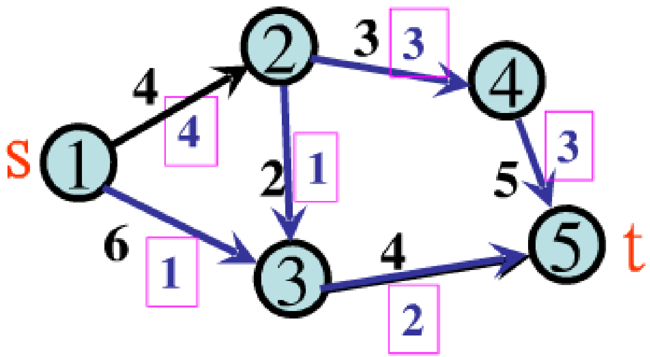
即，$s\in S\subseteq V$，$t\in T=V-S$使得$\sum_{u\in S,v\in T=V-X}c(uv)$最小

一个s-t割是这样一个边的集合：
把这些边从网络中删除之后，s到t就不可达了。

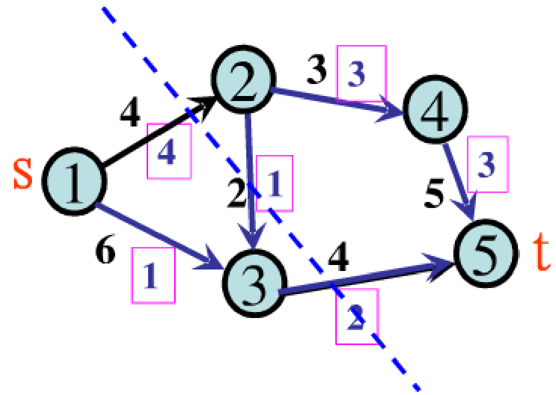
割（CUT）是网络中顶点的一个划分，它把网络中的所有顶点划分成两个顶点集合S和T，其中源点s∈S，汇点t∈T。记为CUT（S,T）。

如右图：源点：s=1；汇点：t=5。

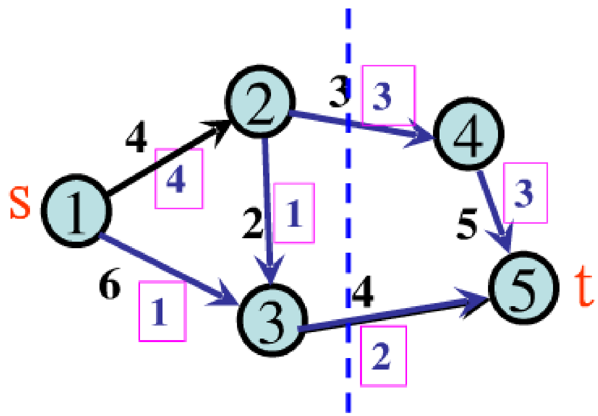
框外是容量，框内是流量



顶点集合S={1， 3}， T={2， 4， 5}
构成一个割。

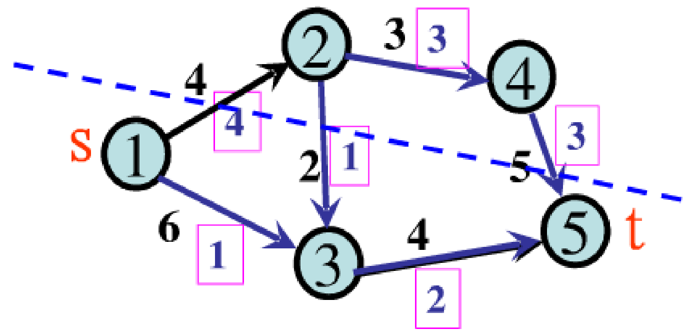


1）、
顶点集合S={1， 2， 3}和T={4， 5}
构成一个割。



顶点集合S={1， 3， 5}， T={2， 4} 不能
构成一个割。

?



最小割问题与最大流问题间的min-max关系

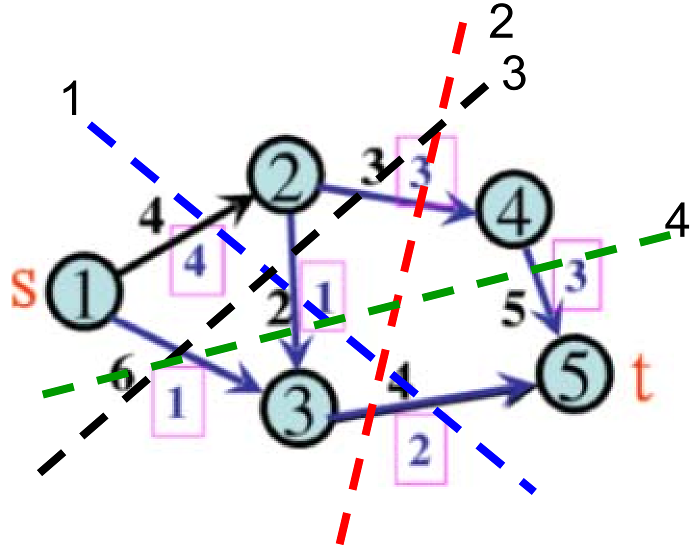
任意s,t-割的容量给出了任意s,t-可行流的流量的上界

因此：如果一个s,t-割的容量等于一个s,t-可行流的流量，则该割是一个最小割且该流是一个最大流

网络流量：5

割的流量

割	正	逆
1	6	1
2	5	0
3	5	0
4	5	0



最大流问题的线性规划

$$Max \sum_{p} x_p$$

$$\sum_{\{p|e\in p\}} x_p \le c(e) \qquad \forall e \in E$$

$$x_p \ge 0 \qquad \forall p$$

其中，p是图G中由s到t的一条路径
x_p是路径p上的实际流量

对偶问题的线性规划

$$Min \sum_{e\in E} y_e c(e)$$

$$\sum_{e\in p} y_e \ge 1 \qquad \forall p$$

$$y_e \ge 0 \qquad \forall p$$

y_e： “选择”边e作为割

- 基于Primal-dual Schema的近似算法

求解集合覆盖问题的线性规划算法

问题的定义

- **输入：**

  有限集X, X的一个子集族F, $X=\cup_{S\in F}S$, 每个集合S的代价$c(S)$

- **输出：**

  $C\subseteq F$，满足

  (1). $X=\cup_{S\in C}S$，

  (2). C是满足条件(1)的代价最小的集族，即$\sum_{S\in C}c(S)$最小.

线性规划表示

对F中的每个集合S，引入一个变量x_S

$x_S=0$ 表示$S \notin C$　　$x_S=1$ 表示$S \in C$

集合覆盖问题的整数规划表示

$$Minimize \quad \sum_{S \in F} c(S) x_S$$

$$Subject\ to \quad \sum_{S: e \in S} x_S \geq 1, \quad e \in X,$$

$$x_S \in \{0,1\}, \quad S \in F$$

输入：

有限集X, X的一个子集族F, $X=\cup_{S\in F}S$，每个集合S的代价$c(S)$

输出：

$C\subseteq F$，满足

(1). $X=\cup_{S\in C}S$,

(2). C是满足条件(1)的代价最小的集族，即$\sum_{S\in C}c(S)$最小.

设$X=\{e, f, g\}$,

$F=\{S_1, S_2, S_3\}$，其中$S_1=\{e, f\}$，$S_2=\{f, g\}$，$S_3=\{e, g\}$

$$c(S_1)= c(S_2)=c(S_3)=1$$

一个整数覆盖必定要选取两个集合，因而费用是2。

而一个分数覆盖可以每个集合取1/2，则费用是3/2。

HIT

设$X=\{e, f, g\}$，$F=\{S_1, S_2, S_3\}$，其中$S_1=\{e, g\}$，$S_2=\{f, g\}$，$S_3=\{e, f\}$

$$c(S_1)= c(S_2)=c(S_3)=1$$

$$\text{Minimize} \quad \sum_{S\in F} c(S)x_S$$

$$\text{Subject to} \quad \sum_{S:e\in S} x_S \ge 1, \quad e\in X,$$

$$x_S\in[0,1], \quad \forall S\in F$$

LP-松弛问题–原始问题：

$$\text{Minimize} \quad c(S_1)x_{s_1} + c(S_2)x_{s_2} + c(S_3)x_{s_3}$$

$$\text{Subject to} \quad e: x_{s_1} + x_{s_3} \ge 1$$

$$f: x_{s_2} + x_{s_3} \ge 1$$

$$g: x_{s_1} + x_{s_2} \ge 1$$

$$x_{s_1}, x_{s_2}, x_{s_3} \ge 0$$

$$e: y_e x_{s_1} + y_e x_{s_3} \ge y_e$$

$$f: y_f x_{s_2} + y_f x_{s_3} \ge y_f$$

$$g: y_g x_{s_1} + y_g x_{s_2} \ge y_g$$

对偶问题：

$$\text{Maximize} \quad y_e + y_f + y_g$$

$$\text{Subject to} \quad S_1: y_e + y_g \le c(S_1)$$

$$S_2: y_f + y_g \le c(S_2)$$

$$S_3: y_e + y_f \le c(S_3)$$

$$y_e, y_f, y_g \ge 0$$

HIT
CS&E

LP-松弛问题—原始问题

$$\text{Minimize} \quad \sum_{S \in F} c(S) x_S$$

$$\text{Subject to} \quad \sum_{S: e \in S} x_S \geq 1, \quad e \in X,$$

$$0 \leq x_S \leq 1, \quad S \in F$$

对每个元素$e \in X$引入变量y_e，
得到**对偶问题**

$$\text{Maximize} \quad \sum_{e \in X} y_e$$

$$\text{Subject to} \quad \sum_{e: e \in S} y_e \leq c(S), \quad S \in F$$

$$y_e \geq 0, \quad e \in X$$

HIT
CS&E

最优(整数)集合覆盖的代价为OPT，
最优分数集合覆盖的代价记为OPT_f，显然有：
$OPT_f \le OPT$

对偶问题的任意可行解的代价都是OPT_f的下界，
因此也是OPT的下界。

Primal-dual schema

基于Primal-dual Schema的集合覆盖近似算法

1. $x \leftarrow 0$; /*原问题初始解，F中的每个集合S对应一个分量x_s*/
2. $y \leftarrow 0$; /*对偶问题初始可行解，
   X中每个元素e对应一个分量y_e*/
3. $U \leftarrow \phi$; /*记录由被覆盖的元素构成的集合*/
4. **while** $U \neq X$ **Do**
5. 取$e_i \in X-U$;
6. 增加y_{e_i}直到 $\sum_{e:\, e\in S} y_e = c(S)$ 对某个$S\in F$成立;
7. 对第6步中满足$\sum_{e:\, e\in S} y_e = c(S)$ 的所有$S\in F$，令$x_s=1$, $U=U\cup S$;
8. 输出x中$x_s=1$的所有集合构成的子集族;

HIT

举例： $X=\{p_1, p_2, p_3, \ldots\ldots, p_{n+1}\}$

$S_1=\{p_1, p_n\}, S_2=\{p_2, p_n\}, \ldots, S_{n-1}=\{p_{n-1}, p_n\},$

$S_n=\{p_1, \ldots, p_{n+1}\},$

$c(S_1)=c(S_2)=\ldots\ldots c(S_{n-1})=1, c(S_n)=1+\varepsilon$

初始化原始问题解： $x_{S1}=x_{S2}=x_{S3}=x_{S4}=x_{S5}=x_{S6}=0$

初始化对偶问题的可行解： $y_{p1}=\ldots\ldots=y_{p12}=0$

$U=$空

$p_n \in X-U$，提高 y_{pn} 至1， 集合 $S_1, S_2, \ldots\ldots, S_{n-1}$ 变成紧的

$x_{S1}=$

$U=$

p_{n+1} ... n 变成紧的，即： $\sum_{e:\, e\in S_n} y_e = c(S_n) = 1+\varepsilon$

$x_{Sn}=$ ... $\ldots, p_{n+1}\} = X$

原始问题近似解 $C=\{S_1, S_2, \ldots\ldots, S_n\}$，

代价 $\text{cost}(C) = n+\varepsilon$

最优解 $C^* = \{S_n\}$

最优解代价 $= 1+\varepsilon$

$$\text{Minimize} \sum_{S\in F} c(S)x_S$$

$$\text{Subject to} \sum_{S:e\in S} x_S \geq 1, \quad e\in X,$$

$$0 \leq x_S \leq 1, \quad S\in F$$

$$\text{Maximize} \sum_{e\in X} y_e$$

$$\text{St.} \sum_{e:e\in S} y_e \leq c(S) \quad S\in F$$

$$0 \leq y_e \leq 1 \quad e\in X$$

x_S	x_{S1}	x_{S1}	……	……	x_{Sn}
	0	0	0	0	0

y_e	y_{p1}	y_{p2}	……	……	y_{pn}	y_{pn+1}
	0	0	0	0	0	0

引理1. 在上述算法中，while循环结束后，x和y分别是原问题和对偶问题的可行解。

证明

1. While循环结束后，X中的所有元素均被覆盖。

2. 算法初始时， $0=\sum_{e:\, e\in S} y_e \le c(S)$对任意$S\in F$成立。

   算法运行过程中，当$\sum_{e:\, e\in S} y_e = c(S)$对某个$S\in F$成立后，$S$中的所有元素均被加入到$U$中，因此在算法以后运行的各个阶段内第5步不会再选中S中的任何元素，即$\sum_{e:\, e\in S} y_e$不会再增加。

   基于以上两条原因，算法结束后，$\sum_{e:\, e\in S} y_e \le c(S)$对任意$S\in F$成立。

引理2. 在上述算法中，while循环结束时x和y满足以下两个性质：

(1) 对于$\forall S\in F$，$x_s\neq 0 \Rightarrow \sum_{e:\, e\in S} y_e = c(S)$；

(2) 对于$\forall e\in X$, $y_e\neq 0 \Rightarrow \sum_{S:\, e\in S} x_s \leq f$（$X$中元素的最大频率）

证明

1. 根据算法的第7步即可证得1。
2. 根据f的定义，对于$\forall e\in X$, e至多属于f个集合; 且对于$\forall S\in F$，$x_s = 1$或0; 从而结论(2)成立.

HIT

定理. 基于primal-dual schema的集合覆盖近似算法的近似比为f

证明 由引理1和引理2，我们知道，算法结束时x和y分别是原问题和对偶问题的可行解，且

(1) 对于$\forall S\in F$， $x_s=0$ 或 $c(S)/1 \le \sum_{e:\, e\in S} y_e \le c(S)$；

(2) 对于$\forall e\in X$， $y_e=0$ 或 $1 \le \sum_{S:\, e\in S} x_s \le f\cdot 1$；

这恰好是命题1中的条件（$\alpha=1$，$\beta=f$）。

由命题1，$cost(C) = c(S) = \sum_{S:\, S\in F} x_s c(S) \le 1\cdot f\cdot \sum_{e\in X} y_e$

由于y是对偶问题的可行解，故$\sum_{e\in X} y_e \le cost(C^*)$.

命题1. 如果$x=(x_1,\ldots,x_n)$和$y=(y_1,\ldots,y_m)$分别是原问题和对偶问题的可行解，且满足：

(1) 原问题的补松弛条件：$\alpha\ge 1$ 对于 $1\le j\le n$： $x_j=0$或者 $\frac{c_j}{\alpha}\le \sum_{i=1}^{m} a_{ij}y_i \le c_j$

(2) 对偶问题的补松弛条件：$\beta\ge 1$ 对于 $1\le i\le m$： $y_i=0$或者 $b_i \le \sum_{j=1}^{n} a_{ij}x_j \le \beta\bullet b_i$

则 $\sum_{j=1}^{n} c_j x_j \le \alpha\bullet\beta\bullet\sum_{i=1}^{m} b_i y_i$