



第九章 Approximation Algorithm

骆吉洲
计算机科学与技术学院



提纲

9.1 近似算法简介

- 9.2 基于组合优化的近似算法
- 9.3 基于贪心策略的近似算法
- 9.4 基于局部优化的近似算法
- 9.5 基于动态规划的近似算法
- 9.6 基于线性规划的近似算法
- 9.7 近似难度



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
 - 能够给出一个优化问题的近似优化解的算法
 - 近似算法主要解决优化问题



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确的代价
 - 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
 - $(1+\epsilon)$ -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 $C/C^*<1$ 且极当 $C^*/C>1$, 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 $\delta(n)$, 如果 $|C-C^*|/C^* \leq \delta(n)$.

结论1. $\delta(n) \leq p(n)-1$.

证. 对于最小化问题

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

对于最大化问题

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

对于某些问题, $\delta(n)$ 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 $\delta(n)$

• 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 $\epsilon > 0$ 为输入的算法. 对于任意固定 ϵ , 近似模式是一个 $(1+\epsilon)$ -近似算法.

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



HIT
CS&E

9.2 基于组合优化的近似算法

- 9.2.1 顶点覆盖问题
- 9.2.2 装箱问题
- 9.2.3 最短异行调度问题
- 9.2.4 TSP问题
- 9.2.5 子集和问题



问题的定义

9.2.1 The Vertex-cover Problem

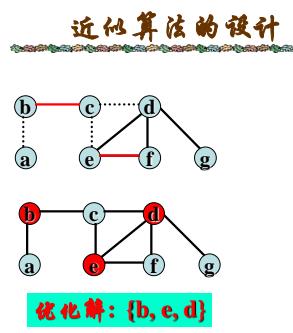
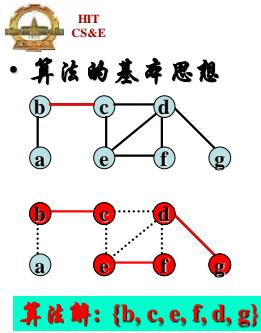
- 问题的定义
- 近似算法的设计
- 算法的性能分析

输入: 无向图 $G=(V, E)$

输出: $C \subseteq V$, 满足

- (1). $\forall (u, v) \in E, u \in C$ 或者 $v \in C$
- (2). C 是满足条件(1)的最小集合。

理论上已经证明优化顶点覆盖问题是NP-完全问题.



-
- 算法
- APPROX-Vertex-Cover (G)**
1. $C=0$
 2. $E'=E[G]$;
 3. While $E' \neq \emptyset$ DO
 4. 选取 $(u, v) \in E'$;
 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

- 时间复杂性
 $T(G)=O(|E|)$
- 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 中每条边的一个端点。于是,
 $|A| \leq |C^*|$, $|C|=2|A| \leq 2|C^*|$, 即 $|C|/|C^*| \leq 2$.

算法的性能分析

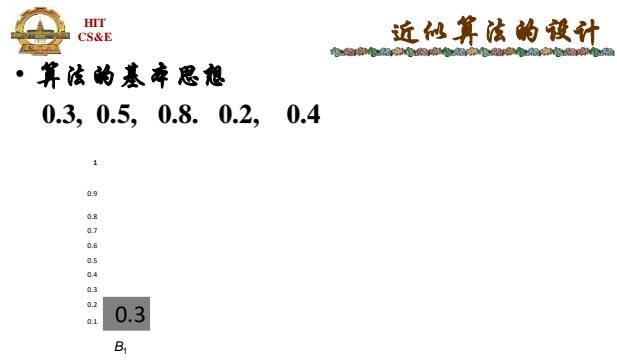


9.2.2 Bin-Packing Problem

- 问题的定义
- 近似算法的设计
- 算法的性能分析

- 输入
体积依次为 $a_1, \dots, a_n \in (0,1]$ 的 n 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

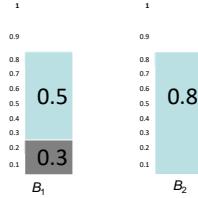
• Bin-Pack 是一个著名的 NP 完全问题。
• 实例: 将 n 种奖券印刷在一些能具有标准尺寸的纸张上, 每张奖券是一个物品, 纸张是箱子, 统一化处理之后变易 Bin-Pack 问题。





• 算法的基本思想

0.3, 0.5, 0.8, 0.2, 0.4

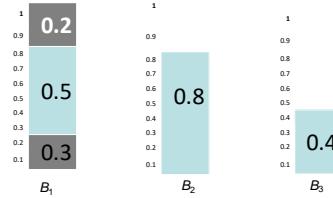


近似算法的设计



• 算法的基本思想

0.3, 0.5, 0.8, 0.2, 0.4



近似算法的设计

优化解也需要3个箱子

• 算法

First-Fit(G)

1. $k \leftarrow 0, B_1 \leftarrow \emptyset$
2. For $i=1$ to n Do
3. 从 B_1, \dots, 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, \dots, B_k

时间复杂性 $O(n^2)$

• 记号

近似比的分析

- k^* —最优解所用箱子的个数
- k —近似解所用箱子的个数
- $|B_i|$ —箱子 B_i 中物品总体积
- $|B_i| + |B_j| > 1$ 对任意 $i \neq j$ 成立
- $k^* \geq \sum_{1 \leq i \leq n} a_i$
- $|B_1| + \dots + |B_k| = \sum_{1 \leq i \leq n} a_i$
- $k/2 < (|B_1| + |B_2|)/2 + \dots + (|B_{k-1}| + |B_k|)/2 + (|B_k| + |B_1|)/2$
 $= |B_1| + |B_2| + \dots + |B_k|$
 $\leq k^*$

定理: First-Fit 算法的近似比为 2



9.2.3 最短并行调度问题

- 问题的定义
- 近似算法的设计
- 算法的性能分析



• 输入

计算时间分别为 t_1, \dots, 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
使并行时间最短

问题的定义

• 最短并行调度是一个著名的 NP 完全问题。



• 算法的基本思想

$$m=3, t_1 \sim t_6 = 1, 2, 3, 4, 5, 6$$

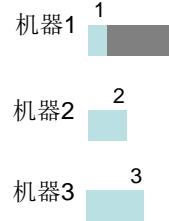


近似算法的设计



• 算法的基本思想

$$m=3, t_1 \sim t_6 = 1, 2, 3, 4, 5, 6$$

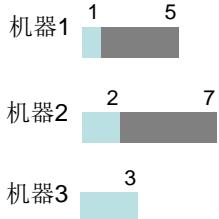


近似算法的设计



• 算法的基本思想

$$m=3, t_1 \sim t_6 = 1, 2, 3, 4, 5, 6$$

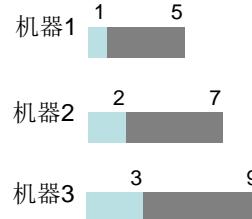


近似算法的设计



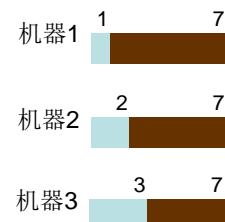
• 算法的基本思想

$$m=3, t_1 \sim t_6 = 1, 2, 3, 4, 5, 6$$



近似算法的设计

优化解



• 算法

MakeSpanScheduling ()

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

时间复杂性 $O(nm)$

• 记号

- T^* —最优解的并行时间
- T —近似解的并行时间

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

- 近似解中第 j 台机器的处理时间最长, 最后处理任务 h

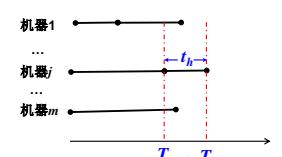
➢ 记第 h 个任务开始执行的时间为 T_{start}

$$\text{则 } T = T_{start} + t_h$$

$$T_{start} \leq (\sum_{1 \leq i \leq h} t_i) / m$$

$$T_{start} \leq T^*$$

$$T = T_{start} + t_h \leq 2T^*$$



定理: MakeSpanScheduling 算法的近似比为 2



问题的定义

9.2.4 The Traveling-salesman Problem

- 问题的定义
- 近似算法设计
- 算法的性能分析

• 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:

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

• 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 问题无具有常数 Ratio Bound 的近似算法, 除非 $NP=P$.

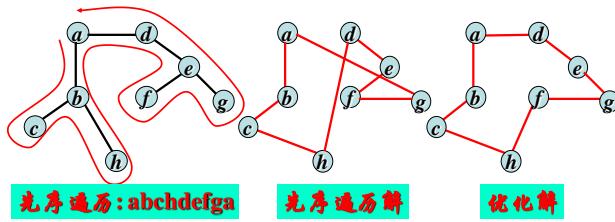


近似算法的设计



• 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



• 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 , 形成有序结点集 L ;
4. 按照 L 中的顺序访问各结点, 形成哈密顿环.



算法的性能分析



• 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 $C(T) \leq C(T') \leq C(H^*)$.

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

• 时间复杂性

- 第2步: $O(|E| + |V| \log |V|) = O(|V|^2 + |V| \log |V|) = O(|V|^2)$
- 第3步: $O(|E|) = O(|V|^2)$, 因为 G 是完全图,
- 第4步: $O(|V|)$
- $T(G) = O(|V|^2)$



由于 W 通过每条边两次, $C(W)=2C(T)$, 进而 $C(W) \leq 2C(H^*)$.
 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.
由于 T 的 *preorder walk* 至少 H , 我们有 $C(H) \leq C(W)$, 即
 $C(H) \leq 2C(H^*)$,
明晰收敛.

9.2.5 The Subset-sum Problem

- 问题的定义
- 指数时间算法
- 完全多项式时间近似模式



问题定义

- 输入:
 (S, t) , $S = \{x_1, x_2, \dots, x_n\}$,
 x_i 和 t 均是正整数
- 输出:
 $\sum_{x \in A} x$, 满足:
 $A \subseteq S$, $\sum_{x \in A} x \leq t$
 $\sum_{x \in A} x = \max \{ \sum_{x \in B} x \mid B \subseteq S \}$



指数时间算法

- 算法
(设 S 是集合, x 是正整数, 定义 $S+x=\{s+x \mid s \in S\}$)
Exact-Subset-Sum($S=\{x_1, x_2, \dots, x_n\}$, t)
 - $n \leftarrow |S|$;
 - $P_0 \leftarrow <0>$;
 - For $i \leftarrow 1$ To n Do
 - $P_i \leftarrow \text{Merge-List}(P_{i-1}, P_{i-1}+x_i)$;
 - 删除 P_i 中所有大于 t 的元素;
 - Return P_n 中最大元素.



计算过程:

- $P_0 = <0>$
- $P_1 = <0, x_1>$ /* 前一个元素所有子集的和(不大于 t) */
- $P_2 = <0, x_1, x_2, x_1+x_2>$
/* 前二个元素所有子集的和(不大于 t) */
- $P_3 = <0, x_1, x_2, x_1+x_2, x_3, x_1+x_3, x_2+x_3, x_1+x_2+x_3>$
/* 前三个元素所有子集的和(不大于 t) */
- $P_i =$ 前 i 个元素所有子集的和(不大于 t)

对 n 作数学归纳法可以证明:

$$P_n = \text{前 } n \text{ 个元素所有子集的和(不大于 } t \text{)}$$



时间复杂性

第4步: $|L_i| = 2|L_{i-1}| = 2^2|L_{i-2}| = \dots = 2^i|L_0| = 2^i$
 $T(n) = O(2^n)$ 随累 t 比较大

- $n \leftarrow |S|$;
- $P_0 \leftarrow <0>$;
- For $i \leftarrow 1$ To n Do
 - $P_i \leftarrow \text{Merge-List}(P_{i-1}, P_{i-1}+x_i)$;
 - 删除 P_i 中所有大于 t 的元素;
- Return P_n 中最大元素.



完全多项式时间近似模式

- 基本思想:

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 $z \leq y$, 使得

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

- 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 .

- 修剪算法

```

Trim( $L = \{y_1, y_2, \dots, y_m\}, \delta$ ) /*  $y_i \leq y_{i+1}$ ,  $0 < \delta < 1$ , 输出缩小的  $L'$  */
 $m \leftarrow |L|$ ;
 $L' \leftarrow \langle y_1 \rangle$ ;
 $last \leftarrow y_1$ ;
For  $i \leftarrow 2$  To  $m$  Do
    If  $last < (1-\delta)y_i$ 
        /* 即  $y_{i-1} < (1-\delta)y_i$ , 由  $L$  和  $L'$  有序, 对  $\forall y \in L'$ , 不满足  $(1-\delta)y_i \leq y \leq y_{i-1}$  */
        Then  $y_i$  加入到  $L'$  尾部; /* 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: $S = \{x_1, x_2, \dots, x_n\}, t \geq 0, 0 < \varepsilon < 1$

输出: 近似解 z

Approx-Subset-Sum(S, t, ε)

1. $n \leftarrow |S|$;
2. $L_0 \leftarrow \langle 0 \rangle$
3. For $i \leftarrow 1$ To n Do
4. $L_i \leftarrow \text{Merge-List}(L_{i-1}, L_{i-1} + x_i)$;
5. $L_i \leftarrow \text{Trim}(L_i, \varepsilon/n)$ /* 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完全多项式时间近似模式.

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

令 $S = \{1, 4, 5\}$, 则

$P_1 = \{0, 1\}$,

$P_2 = \{0, 1, 4, 5\}$,

$P_3 = \{0, 1, 4, 5, 6, 9, 10\}$.

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 的元素的有序系.

(1). 证明 $C^*(1-\varepsilon) \leq z$

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

当 $i=0$ 时 $P_0 = \{0\}, L_0 = \{0\}$, 命题成立.

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

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

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

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

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

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

由于 $z' \in L_k, z' + x_{k+1} \in L_{k+1}$, 而且

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

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

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

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

注意, 由于子集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.

最后,若 $C^* \in P_n$ 是子集合族和问题的优化解,则存在一个 $z' \in L_n$,使 $(1-\varepsilon/n)^n C^* \leq z' \leq C^*$.因算法解 $z = \max(L_n)$, $(1-\varepsilon/n)^n C^* \leq z \leq z' \leq C^*$.

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 $n > 1$, $(1-\varepsilon) < (1-\varepsilon/n)^n$.

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

(2). 检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. 修省后, L_i 中的相邻元素 z 和 z' 满足:

$$z' < (1-\varepsilon/n)z, \text{ 即 } z/z' > 1/(1-\varepsilon/n).$$

如果 L_i 中具有 $k+2$ 个元素,则必有 $y_0=0, y_1=z_0, y_2>z_0 \cdot 1/(1-\varepsilon/n), y_3>z_0 \cdot 1/(1-\varepsilon/n)^2, \dots, y_{k+1}>z_0 \cdot 1/(1-\varepsilon/n)^k$, 而且 $z_0 \cdot 1/(1-\varepsilon/n)^k \leq t$.

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

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

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



9.3 基于贪心策略的近似算法

- 9.3.1 集合覆盖问题
- 9.3.2 不相交路径问题
- 9.3.3 亚模函数与贪心近似



问题的定义

9.3.1 The Set-covering Problem

- 问题的定义
- 近似算法的设计
- 算法的性能分析

• 输入:

有限集 X , X 的子集集族 F , $X = \bigcup_{S \in F} S$

• 输出:

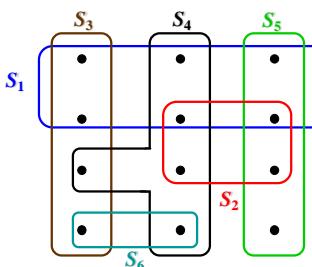
$C \subseteq F$, 满足

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



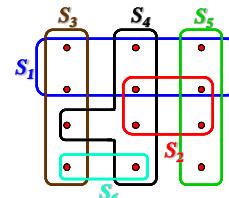
$X=12$ 个黑点, $F=\{S_1, S_2, S_3, S_4, S_5, S_6\}$
优化解 $C=\{S_3, S_4, S_5\}$



近似算法的设计

• 基本思想

- 贪心选择: 选择能覆盖最多未被覆盖元素的子集



$C=\{S_3, S_4, S_5\}$



算法性能的分析

• 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. $C \leftarrow \emptyset$;
3. While $U \neq \emptyset$ 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 $\min(|X|, |F|)$
- 计算 $|S \cap U|$ 需要时间 $O(|X|)$
- 第4步需要时间 $O(|F||X|)$
- $T(X, F) = O(|F||X|\min(|X|, |F|))$

• Ration Bound

定理1. 令 $H(d) = \sum_{I \leq d} I/I$. *Greedy-Set-Covers* 是多项式 $p(n)$ -近似算法, $p(n) = H(\max\{|S| / S \in F\})$.

证. 我们已经算法是多项式算法, 只需计算 Ratio Bound.
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

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



显然, 算法给出的解 C 的代价是 $|C|$, $|C|$ 平均地分摊到 X 的所有点. 由于 C^* 也覆盖 X , 我们有

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

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, 而应为每个 c_x 只加一次.

如果 $\forall S \in F, \sum_{x \in S} c_x \leq H(|S|)$ 成立, 则
 $|C| \leq \sum_{S \in C^*} H(|S|) \leq |C^*| \cdot H(\max\{|S| / S \in F\})$,
 $|C| / |C^*| \leq H(\max\{|S| / S \in F\})$, 定理成立.

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



对于 $\forall S \in F$ 和 $i=1, 2, \dots, |C|$, 令 $u_i = |S - (S_1 \cup S_2 \cup \dots \cup S_{i-1})|$ 是 S_1, S_2, \dots, S_i 被选中后, S 中未被覆盖的点数. S_i 先于 S 被选中.

令 $u_0 = |S|$, k 是满足下列条件的最小数: $u_k = 0$, 即 S 中每个元素被 S_1, S_2, \dots, S_k 中至少一个覆盖.

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

$$\sum_{x \in S} c_x = \sum_{i=1}^k (u_{i-1} - u_i) \cdot \frac{1}{|S_i \cap (S_1 \cup S_2 \cup \dots \cup S_{i-1})|}$$

注意: $|S_i \cap (S_1 \cup S_2 \cup \dots \cup S_{i-1})| \geq |S - (S_1 \cup S_2 \cup \dots \cup S_{i-1})| = u_{i-1}$, 因为 Greed 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,

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



$$\begin{aligned} \sum_{x \in S} c_x &\leq \sum_{i=1}^k (u_{i-1} - u_i) \cdot \frac{1}{u_{i-1}} \\ &= \sum_{i=1}^k \sum_{j=u_{i-1}}^{u_{i-1}} \frac{1}{u_{i-1}} \\ &\leq \sum_{i=1}^k \sum_{j=u_{i-1}}^{u_{i-1}} \frac{1}{j} \\ &= \sum_{i=1}^k \left(\sum_{j=1}^{u_{i-1}} \frac{1}{j} - \sum_{j=1}^u \frac{1}{j} \right) \\ &= \sum_{i=1}^k [H(u_{i-1}) - H(u_i)] \\ &= H(u_0) - H(u_k) \\ &= H(u_0) \\ &= H(|S|) \end{aligned}$$

$$\begin{aligned} u_k &= 0 \\ u_0 &= |S| \end{aligned}$$



推论1. Greedy-Set-Cover是一个多项式 $\ln(|X|+1)$ -近似算法。

证. 由不等式 $H(n) \leq \ln(n+1)$ 可知

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

9.3.2 Disjoint-Path Problem

- 问题的定义
- 近似算法的设计
- 算法的性能分析



• 输入:

图 $G=(V,E)$, 源顶点集 S , 汇顶点集 T

• 输出:

$$A \subseteq S \times T$$

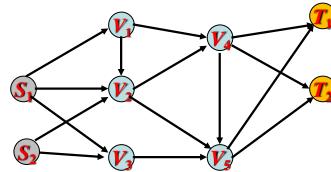
(1). A 中的所有顶点对在 G 中存在无公共边的路径

(2). $|A|$ 最大。

* 不相交路径问题是很多实际问题的抽象。

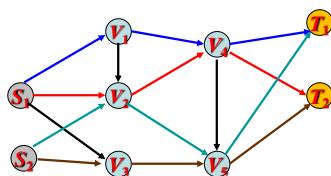
* 不相交路径问题是NP-完全问题。

• 问题的实例



图G
 $S=\{S_1, S_2\}$ $T=\{T_1, T_2\}$

• 问题的实例

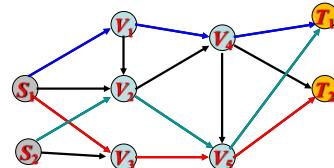


图G
 $S=\{S_1, S_2\}$ $T=\{T_1, T_2\}$
优化解 $A^*=\{(S_1, T_1), (S_1, T_2), (S_2, T_1), (S_2, T_2)\}$



• 基本思想

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



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

近似算法的设计

• 算法

EdgeDisjointPath(G, S, T)

1. $A \leftarrow \emptyset; B \leftarrow S \times T$
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 $P = \emptyset$ 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 $O(|S||T||V|^4)$

第3步的开销 $O(|V|^4)$, 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^* —问题精确

A —近似算法输出的近似解

k —参数, 任意固定的值

证明思想:

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), $|S^*| \leq k|A|$ (引理 2)

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

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

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



HIT
CS&E

引理 1. $|L^*| \leq (m/k)|A|$ 对任意 k 成立。

证. A^* —精确解

$L^*—A^*$ 中长度大于 k 的路径

A —近似解, $1 \leq |A|$

- L^* 中任意两条路径的无公共边
- L^* 中的所有路径至少用到 $k|L^*|$ 条边
- $k|L^*| \leq |E|=m$
- $|L^*| \leq (m/k)|A|$

引理 2. $|S^*| \leq k|A|$ 对任意 k 成立。

证. A^* —精确解

$S^*—A^*$ 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 是算法操作逐断添加路径得到的)
 - > 由近似算法得到 A 时, 第一条与 p^* 相交的路径是 p
 - > 算法选择 p 加入 A 而未选择 p^* , 说明 p 比 p^* 更短, $|p| < k$
 - > A^* (除 S^*) 中与 p 有公共边的路径至多有 k 条
- $|S^*| \leq k \times |\{p \in A: p \text{ 与某条 } S^* \text{ 中某条短路径相交}\}| \leq k|A|$



HIT
CS&E

9.3.3 亚模函数与贪心近似

- 亚模函数及基本性质
- 亚模函数示例
- 亚模函数最大化贪心近似算法

推荐阅读:

- Coresets for data efficient training of machine learning models, ICML 2020
 - > 运用亚模函数贪心算法构造 Coresets 加速机器学习模型训练速度



HIT
CS&E

亚模函数

亚模函数=Submodular Function=次模函数=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 是亚模函数

(1): $f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$ 对 $\forall X \subseteq Y \subseteq U$ 和 $\forall x \in U - Y$ 成立

(2): $f(X \cup Y) + f(X \cap Y) \leq f(X) + f(Y)$ 对任意 $X, Y \subseteq U$ 成立

(1) \Rightarrow (2): 假设(1)成立, 任取 $A, B \subseteq U$, 往证 $f(A \cup B) + f(A \cap B) \leq f(A) + f(B)$

对 $|A \cup B|/|A \cap B|$ 做数学归纳:

$|A \cup B|/|A \cap B|=0$ 时, $A=B$, 结论成立

假设 $|A \cup B|/|A \cap B| < k$ 时, 结论成立

当 $|A \cup B|/|A \cap B|=k$ 时, 不妨设 $\exists u \in A - B$

$$f(A) + f(B) = [f(A) - f(A - \{u\})] + [f(A - \{u\}) + f(B)]$$

$$\geq [f(A) - f(A - \{u\})] + [f((A \cup B) - \{u\}) + f(A \cap B)]$$

归纳假设

$$\geq [f(A \cup B) - f(A \cup B - \{u\})] + [f((A \cup B) - \{u\}) + f(A \cap B)]$$

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

(1)



亚模函数

亚模函数=Submodular Function=次模函数=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 是亚模函数

(1): $f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$ 对 $\forall X \subseteq Y \subseteq U$ 和 $\forall x \in U - Y$ 成立

(2): $f(X \cup Y) + f(X \cap Y) \leq f(X) + f(Y)$ 对任意 $X, Y \subseteq U$ 成立

(2) \Rightarrow (1):假设(2)成立,任取 $A \subseteq B \subseteq U$ 且 $u \in U - B$,往证 $f(A \cup \{u\}) - f(A) \geq f(B \cup \{u\}) - f(B)$

$$\begin{aligned} \text{由于 } A \subseteq B \text{ 且 } u \in U - B, & (A \cup \{u\}) \cup B = B \cup \{u\} \\ & X \quad Y \\ & (A \cup \{u\}) \cap B = A \end{aligned}$$

由(2)得到

$$f(B \cup \{u\}) + f(A) \leq f(A \cup \{u\}) + f(B)$$

整理即得(1)



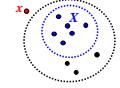
亚模函数

亚模函数=Submodular Function=次模函数=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 是亚模函数

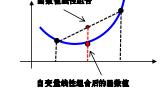
(1): $f(X \cup \{x\}) - f(X) \geq f(Y \cup \{x\}) - f(Y)$ 对 $\forall X \subseteq Y \subseteq U$ 和 $\forall x \in U - Y$ 成立

(2): $f(X \cup Y) + f(X \cap Y) \leq f(X) + f(Y)$ 对任意 $X, Y \subseteq U$ 成立



条件(1)的直观含义

边缘增益递减



条件(2)的直观含义

高阶集合变量函数的凸性

x 在小集合上的增益大于等于在大集合上的收益 集合运算后的函数值小于等于函数值的组合



(B):集合函数 $f:2^U \rightarrow R$ 是亚模函数,则

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

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

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

$$(4) \text{若 } f \text{ 是单调函数, 则 } f(B) - f(A) \leq \sum_{x \in B \setminus A} [f(A \cup \{x\}) - f(A)]$$

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

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

$$\begin{aligned} f(A \cup B) - f(A) &= \sum_{i=1}^k [f(A \cup B_{i-1} \cup \{b_i\}) - f(A \cup B_{i-1})] \\ &\leq \sum_{i=1}^k [f(A \cup \{b_i\}) - f(A)] \quad f \text{是亚模函数且 } A \cup B_{i-1} \supseteq A \end{aligned}$$

(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)

$$(4) f(B) - f(A) \leq f(A \cup B) - f(A) \leq \sum_{x \in B \setminus A} [f(A \cup \{x\}) - f(A)]$$



亚模函数运算性质

(O):集合函数 $f, g:2^U \rightarrow R$ 是亚模函数且 $T \subseteq U$, 则

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

(2) $h(X)=f(X)+g(X)$ 是亚模函数

(3) $h(X)=f(X \cap T)$ 是亚模函数

(4) $h(X)=f(U-X)$ 是亚模函数

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

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



亚模函数其他性质(1)

(I): $f(X)$ 是亚模函数 $\Leftrightarrow g(X)=f(X)-f(\emptyset)$ 是亚模函数

含义: 可以假设亚模函数在空集上取0值——标准化的亚模函数

证明: $X \subseteq Y \subseteq U$ 且 $x \in U - Y$

$$\begin{aligned} \Rightarrow: & \qquad \Leftarrow: \\ g(X \cup \{x\}) - g(X) &= f(X \cup \{x\}) - f(X) \\ =[f(X \cup \{x\}) - f(\emptyset)] - [f(X) - f(\emptyset)] &= [f(X \cup \{x\}) - f(\emptyset)] - [f(X) - f(\emptyset)] \\ =f(X \cup \{x\}) - f(X) &= g(X \cup \{x\}) - g(X) \\ \geq f(Y \cup \{x\}) - f(Y) \quad (\text{已知条件}) & \geq g(Y \cup \{x\}) - g(Y) \quad (\text{已知条件}) \\ =[f(Y \cup \{x\}) - f(\emptyset)] - [f(Y) - f(\emptyset)] &=[f(Y \cup \{x\}) - f(\emptyset)] - [f(Y) - f(\emptyset)] \\ =g(Y \cup \{x\}) - g(Y) &= f(Y \cup \{x\}) - f(Y) \end{aligned}$$



亚模函数其他性质(2)

(II): $f(X \cup A) - f(X) \geq f(Y \cup A) - f(Y)$ 对 $\forall X \subseteq Y \subseteq U$ 和 $\forall A \subseteq U - Y$ 成立

含义: 函数取值对集合增量的增益也是单调递减的

证明: 令 $A=\{a_1, a_2, \dots, a_k\}$

$$\begin{aligned} f(X \cup \{a_1\}) - f(X) &\geq f(Y \cup \{a_1\}) - f(Y) \quad \text{因 } X \subseteq Y \subseteq S \text{ 且 } a_1 \in U - Y \\ f(X \cup \{a_1, a_2\}) - f(X \cup \{a_1\}) &\geq f(Y \cup \{a_1, a_2\}) - f(Y \cup \{a_1\}) \\ &\dots \\ f(X \cup \{a_1, \dots, a_k\}) - f(X \cup \{a_1, \dots, a_{k-1}\}) &\geq f(Y \cup \{a_1, \dots, a_k\}) - f(Y \cup \{a_1, \dots, a_{k-1}\}) \end{aligned}$$

所有不等式两端相加得

$$f(X \cup A) - f(X) \geq f(Y \cup A) - f(Y)$$

推论: 亚模函数在 $\forall X \subseteq Y \subseteq U$ 和 $\forall A \subseteq U - Y$ 上满足 $f(Y \cup A) - f(X \cup A) \leq f(Y) - f(X)$



亚模函数其他性质(3)

(III) 标准集合函数 $f: 2^U \rightarrow R$ 是亚模函数，则 $f_T(X) = f(T \cup X) - f(T)$ 是亚模函数
证明： $\Leftrightarrow X \subseteq Y \subseteq U$ 且 $x \in U - Y$

情形1: $x \in T$ 则 $f_T(X \cup \{x\}) - f_T(X) = f_T(Y \cup \{x\}) - f_T(Y) = 0$ 亚模性成立

情形2: $x \notin T$ 则 $f_T(X \cup \{x\}) - f_T(X)$
 $= f(T \cup X \cup \{x\}) - f(T \cup X)$
 $\geq f(T \cup Y \cup \{x\}) - f(T \cup Y)$ f 的亚模性
 $= f_T(Y \cup \{x\}) - f_T(Y)$



亚模函数其他性质(4)

(IV) 标准集合函数 $f: 2^U \rightarrow R$ 是亚模函数 $\Rightarrow f(X)$ 是亚可加性的

证明： $\Leftrightarrow \forall A, B \subseteq U$ 且 $B = \{b_1, b_2, \dots, b_k\}$, 并记 $B_i = \{b_1, b_2, \dots, b_i\}$

$$\begin{aligned} f(A \cup B) &= f(A \cup B_k) - f(A \cup B_{k-1}) + f(A \cup B_{k-1}) - \dots - f(A \cup B_0) + f(A \cup B_0) \\ &= f(A) + f(A \cup B_k) - f(A \cup B_{k-1}) + \dots + f(A \cup B_1) - f(A \cup B_0) \\ &\leq f(A) + f(B_k) f(B_{k-1}) + \dots f(B_1) f(B_0) \quad \text{性质 II 的推论} \\ &= f(A) + f(B) \end{aligned}$$



亚模函数其他性质(5)

(V) 标准集合函数 $f: 2^U \rightarrow R$ 是亚模函数 $\Leftrightarrow f_T(X)$ 对任意 $T \subseteq U$ 是亚可加的
证明： $\Rightarrow f$ 是亚模的，因此 $f_T(X)$ 是亚模的，进而由性质 IV 知 $f_T(X)$ 亚可加
 $\Leftrightarrow \forall X \subseteq Y \subseteq U$ 且 $x \in U - Y$

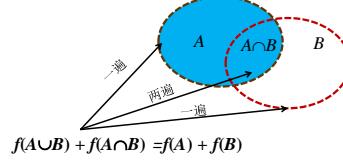
$$\begin{aligned} f(Y \cup \{x\}) - f(Y) &= [f(X \cup (Y-X) \cup \{x\}) - f(X)] - [f(X \cup (Y-X)) - f(X)] \\ &= f_X((Y-X) \cup \{x\}) - f_X(Y-X) \\ &\leq f_X((Y-X) + f_X(\{x\}) - f_X(Y-X)) \quad f_X \text{ 亚可加} \\ &= f_X(\{x\}) \\ &= f(X \cup \{x\}) - f(X) \end{aligned}$$



亚模函数示例(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 $f(A)$ 是亚模函数



亚模函数示例(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 $f(A)$ 是亚模函数

由例1和亚模函数运算法则知 f 是亚模函数

例3: 秩函数

令 $U = \{v_1, v_2, \dots, v_m\}$ 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 $f(A)$ 是亚模函数

线性代数中子空间维数定理



亚模函数示例(3)

例4: 覆盖函数

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

对 $\forall A \subseteq U$ 定义 $f(A) = |\cup_{e \in A} s_e|$, 则 $f(A)$ 是标准化的单调亚模函数

(1) $f(\emptyset) = 0$ 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 是亚模的



亚模函数示例(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 $f(\cdot)$ 是标准化对称亚模函数

- (1) $f(\emptyset) = 0$, 故 $f(\cdot)$ 是标准化的
- (2) $f(A) \leq f(V-A)$, 故 $f(\cdot)$ 是对称的
- (3) 亚模性留作习题。

提示: 参照例4中的思维方法



亚模函数最大化贪心算法(1)

计算问题

输入: 空间 U 及其上的非负亚模函数 $f: 2^U \rightarrow R^+$
判定子集 A 是否为解的约束条件 $I \subseteq 2^U$
输出: $A \subseteq U$ 使得

$$\begin{aligned} &\max f(A) \\ &\text{s.t. } A \in I \subseteq 2^U \end{aligned}$$



基数约束最大化贪心算法(2)

计算问题

输入: 空间 U 及其上的非负单调亚模函数 $f: 2^U \rightarrow R^+$
判定子集 A 是否为解的约束条件 $I = \{A | |A| \leq k\}$
输出: $A \subseteq U$ 使得

$$\begin{aligned} &\max f(A) \\ &\text{s.t. } |A| \leq k \end{aligned}$$

算法:

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

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



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

$$\begin{aligned} f(A_i) - f(A_{i-1}) &= f(A_{i-1} \cup \{x_i\}) - f(A_{i-1}) \\ &\geq \max_{x \in A_{OPT}} [f(A_{i-1} \cup \{x\}) - f(A_{i-1})] && \text{贪心选择} \\ &\geq \frac{\sum_{x \in A_{OPT}} [f(A_{i-1} \cup \{x\}) - f(A_{i-1})]}{k} && \text{最大值} \geq \text{平均值} \\ &\geq \frac{k}{k} [f(A_{OPT}) - f(A_{i-1})] && f(B \cup A) \cdot f(A) \leq \sum_{x \in B} [f(A \cup \{x\}) - f(A)] \\ &\geq \frac{k}{k} [f(A_{OPT}) - f(A_{i-1})] && \text{单调性} \end{aligned}$$

于是: $f(A_{OPT}) - f(A_i) \leq (1-1/k)[f(A_{OPT}) - f(A_{i-1})]$
 $f(A_{OPT}) - f(A) = f(A_{OPT}) - f(A_k)$ $A = A_k$
 $\leq (1-1/k)[f(A_{OPT}) - f(A_{k-1})]$
 $\leq \dots$
 $\leq (1-1/k)^k [f(A_{OPT}) - f(A_0)]$ 注意 $(1-1/k)^k \uparrow 1/e$
 $f(A) \geq f(A_{OPT}) - (1-1/k)^k [f(A_{OPT}) - f(\emptyset)] \geq [1-1/e] \cdot f(A_{OPT}) - (1/e) \cdot f(\emptyset)$



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: $U = \{x_1, \dots, x_n\}$, 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 是非单调亚模函数
 (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)

计算问题: 输入: 空间 U 及其加权函数 w , $w(A) = \sum_{x \in A} w(x)$
 U 上的非负单调亚模函数 $f: 2^U \rightarrow R^+$, 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. $A \leftarrow \emptyset$
2. while $U \neq \emptyset$ Do
 3. $x^* \leftarrow \max_{x \in U} \{f(A \cup \{x\}) - f(A)/w(x)\}$ **贪心选择**
 4. If $w(A \cup \{x^*\}) \leq b$ Then $A \leftarrow A \cup \{x^*\}$
 5. $U \leftarrow U - \{x^*\}$
 6. $x_0 \leftarrow \max_{x \in U} f(x)$
 7. $S \leftarrow A$ 和 $\{x_0\}$ 中亚模函数值较大者
 8. Return S

结论: $f(S) \geq 0.5 \cdot (1 - \frac{1}{e}) \cdot f(S_{OPT})$



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

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

$$\begin{aligned}
 f(A_{OPT}) &\leq f(A_{i-1}) + \sum_{x \in A_{OPT} \setminus A_{i-1}} [f(A_{i-1} \cup \{x\}) - f(A_{i-1})] && \text{亚模函数基本性质} \\
 &= f(A_{i-1}) + \sum_{x \in A_{OPT} \setminus A_{i-1}} w(x) \frac{[f(A_{i-1} \cup \{x\}) - f(A_{i-1})]}{w(x)} && \text{恒等变形} \\
 &\leq f(A_{i-1}) + \frac{[f(A_{i-1} \cup \{x_i\}) - f(A_{i-1})]}{w(x_i)} \sum_{x \in A_{OPT} \setminus A_{i-1}} w(x) && \text{贪心选择} \\
 &= f(A_{i-1}) + \frac{[f(A_i) - f(A_{i-1})]}{w(x_i)} \sum_{x \in A_{OPT} \setminus A_{i-1}} w(x) \\
 &= f(A_{i-1}) + \frac{b}{w(x_i)} [f(A_i) - f(A_{i-1})] && \text{优化解满足背包约束} \\
 \text{于是: } f(A_i) - f(A_{OPT}) &\geq (1-w(x_i)/b)[f(A_{i-1}) - f(A_{OPT})] \geq \prod_{i=1}^i (1-\frac{w(x_i)}{b})[f(\emptyset) - f(A_{OPT})] \\
 &\geq -[\exp(-w(A)/b)]f(A_{OPT}) && f(\emptyset)=0 \text{ 且 } 1-x \leq e^{-x} \\
 f(A \cup x^*) &\geq [1 - \exp(-\frac{w(A)+w(x^*)}{b})]f(A_{OPT}) \\
 &\geq (1-1/e)f(A_{OPT}) && w(A)+w(x^*)>b
 \end{aligned}$$



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

$$\begin{aligned}
 2f(S) &\geq f(A) + f(x_0) \\
 &\geq f(A) + f(x^*) && x_0 \text{ 的取法} \\
 &\geq f(A \cup x^*) && \text{亚可加性} \\
 &\geq (1-1/e)f(S_{OPT}) && \text{引理}
 \end{aligned}$$



拟阵约束最大化贪心算法(3)

计算问题: 输入: 空间 U 及其上的非负单调亚模函数 $f: 2^U \rightarrow R^+$
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

1. $A \leftarrow \emptyset$
2. while $U \neq \emptyset$ 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. $U \leftarrow U - \{x^*\}$
6. Return A

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



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

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

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

$$\begin{aligned}
 f(A_{OPT}) - f(A) &\leq \sum_{x_i \in A_{OPT}} [f(A \cup \{x_i\}) - f(A)] && \text{亚模函数基本性质} \\
 &\leq \sum_{x_i^* \in A_{OPT}} [f(A_{i-1} \cup \{x_i^*\}) - f(A_{i-1})] && \text{亚模函数定义} \\
 &\leq \sum_{x \in A} [f(A_{i-1} \cup \{x\}) - f(A_{i-1})] && \text{贪心选择, 基交换} \\
 &= \sum_{i=1}^K [f(A_i) - f(A_{i-1})] \\
 &= f(A) - f(\emptyset)
 \end{aligned}$$

$$f(A) \geq 0.5 \cdot f(A_{OPT}) + f(\emptyset)$$



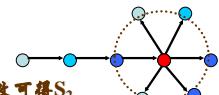
9.4 基于局部搜索的近似算法

- 9.4.1 局部搜索原理
- 9.4.2 最大割问题
- 9.4.3 设施定位问题



9.4.1 局部搜索原理

- 计算问题有很多可行解
- 从任意可行解出发, 进行局部修改, 产生其他可行解
- 达到一个局部优化解(比相邻可行解代价更优)
- 输出局部优化解作为近似解
- 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 $\Leftrightarrow S_1$ 由局部修改可得 S_2
- 局部搜索适用条件
 - 顶点的度数小(多项式), 确保多项式时间内达到“相邻”解
 - 从任意可行解开始, 多项式时间内必然能找到局部优化解





问题的定义

• 输入:

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

• 输出:

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

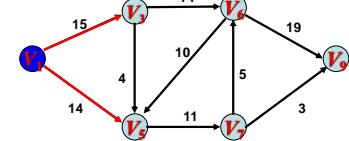
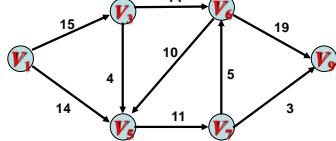
*最小割问题存在多项式时间算法.

*最大割问题是NP-完全问题.

9.4.2 最大割问题

- 问题定义
- 局部搜索算法
- 时间复杂度分析
- 近似比分析

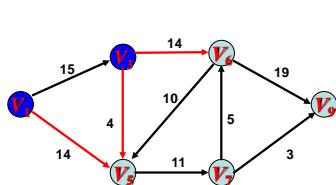
问题的实例



$S_0 = \{V_1\}$

代价 29

交换 V_3 在 S 和 $V-S$ 中的位置

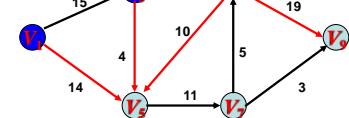


$S_0 = \{V_1\}$

代价 29 交换 V_3 在 S 和 $V-S$ 中的位置

$S_1 = \{V_1, V_3\}$

代价 32 交换 V_6 在 S 和 $V-S$ 中的位置



$S_0 = \{V_1\}$

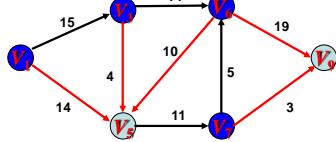
代价 29 交换 V_3 在 S 和 $V-S$ 中的位置

$S_1 = \{V_1, V_3, V_6\}$

代价 32 交换 V_6 在 S 和 $V-S$ 中的位置

$S_1 = \{V_1, V_3, V_6\}$

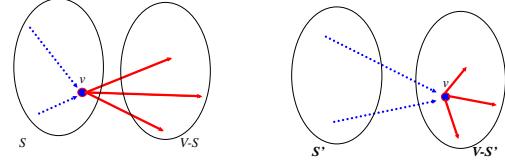
代价 57 交换 V_7 在 S 和 $V-S$ 中的位置



$S_0 = \{V_1\}$ 代价29 变换 V_3 在 S 和 $V-S$ 中的位置
 $S_1 = \{V_1, V_3\}$ 代价32 变换 V_6 在 S 和 $V-S$ 中的位置
 $S_2 = \{V_1, V_3, V_6, V_7\}$ 代价50 局部优化解
全局优化解

算法思想:

- 局部修改操作—变换 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



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



近似比分析

• 定理: ApproxMaxCut的近似比为2。

证明: 由于算法输出局部最优解,

- $\forall v \in S, cost(v, S) \leq 0$
 - ✓ $\sum_{u \in S} W(uv) \leq \sum_{u \in V-S} W(uv)$
 - ✓ $\sum_{u \in S} W(uv) + \sum_{u \in V-S} W(uv) \leq 2 \times \sum_{u \in V-S} W(uv)$
 - ✓ $(1/2) \times \sum_{u \in V} W(uv) \leq \sum_{u \in V-S} W(uv)$
- $\forall v \in V-S, cost(v, S) \leq 0$
 - ✓ $(1/2) \times \sum_{u \in V} W(uv) \leq \sum_{u \in S} W(uv)$
- $\sum_{uv \in E} W(uv) \leq 2W(S)$
- $W(S^*) \leq \sum_{uv \in E} W(uv) \leq 2W(S)$

• 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. $S \leftarrow \{u\}$; $/ * u \text{是 } V \text{中任意顶点} */$
2. repeat
 3. 选取使得 $cost(v, S) > 0$ 的顶点 v , 变换 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 $\sum_{uv \in E} W(uv) \leq |V|^2 \times max_{uv \in E} W(uv) = O(|V|^2 W)$ 递增
 - ✓ 循环每做一次, 割的代价至少增大1
- 总的时间开销为 $O(|V|^4 W)$



9.4.3 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 $d: U \times F \rightarrow R^+$

— 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 $C(S) = C_f(S) + C_r(S)$

• 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象.

*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 i'
- 算法

LocalSearchFacility(F, U, ϵ)

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

• 该算法在多项式时间尚终止，且近似比为 $3 + o(\epsilon)$ (参见讲义)

9.5 基于动态规划的近似算法

- 9.5.1 用动态规划设计近似算法
- 9.5.2 0-1背包问题的完全多项式近似模式
- 9.5.3 Bin-Packing 问题的近似模式

4. 动态规划算法

9.4.1 动态规划与近似算法

- 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
- 近似策略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
- 近似策略2
 - 将动态规划方法视作解空间的枚举过程
 - 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



9.5.2 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
- 近似比分析



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 ，背包承重量 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： $(x_1, x_2, \dots, x_n), x_i \in \{0, 1\}$, 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



第4章曾介绍了0-1背包问题的动态规划算法

- 将第 $i, i+1, \dots, n$ 个物品装入背包中
- $b_{i,j}$ 表示欲获得总价值量 j 所需的最小背包容量
- 其中 $1 \leq j \leq \sum_{1 \leq i \leq n} v_i$
- 问题的优化子结构可以重述为：

$$\begin{aligned} b_{i,j} &= b_{i+1,j} & j < v_i \\ b_{i,j} &= \min(b_{i+1,j}, b_{i+1,j} + w_i) & j \geq v_i \\ b_{n,j} &= w_n & j \geq v_n \\ b_{n,j} &= 0 & j < v_n \end{aligned}$$

- 用相同计算过程可得到 $O(n \sum_{1 \leq i \leq n} v_i)$ 时间的DP算法
- 将该算法称为BoolPacking算法



问题定义的实例的变形

• 给定参数 ϵ ,令 $K=n/\epsilon$, $v_{max}=\max_{1 \leq i \leq n} v_i$

• 实例 $I=\langle w_1, \dots, w_n, v_1, \dots, v_n \rangle$ 变形为 $I'=\langle w_1, \dots, w_n, v'_1, \dots, v'_n \rangle$

$$\cdot v'_i = \lfloor v_i \times (K/v_{max}) \rfloor$$

$$\cdot \sum_{1 \leq i \leq n} v'_i = \sum_{1 \leq i \leq n} \lfloor v_i \times (K/v_{max}) \rfloor \leq \sum_{1 \leq i \leq n} K \times (v_i/v_{max}) = n^2/\epsilon$$

• BoolPacking算法在 I' 上仅需运行多项式时间



完全多项式近似模式

算法 ApproxPacking($W[1:n], V[1:n], C, \epsilon$)

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 $1+\epsilon$ -近似解 $\langle x_1, \dots, x_n \rangle$

1. $K=n/\epsilon$;

2. $v_{max} \leftarrow \max_{1 \leq i \leq n} v_i$;

3. $V[i] \leftarrow \lfloor K \times (V[i]/v_{max}) \rfloor$; $/* i=1,2,\dots,n */$

4. $\langle x_1, \dots, x_n \rangle \leftarrow \text{BoolPacking}(W[1:n], V[1:n], C)$; $/* DP */$

5. 输出 $\langle x_1, \dots, x_n \rangle$

时间复杂度 $O(n^3/\epsilon)$

主要取决于第4步的开销

定理：算法 ApproxPacking 是一个 $1+\epsilon$ -近似算法

证： $\langle z_1, \dots, z_n \rangle$ —优化解, $Z = \sum_{1 \leq i \leq n} v_i z_i$ —优化解代价, $Z' = \sum_{1 \leq i \leq n} v'_i z_i$
 $\langle x_1, \dots, x_n \rangle$ —近似解, $X = \sum_{1 \leq i \leq n} v_i x_i$ —近似解代价

$\langle x_1, \dots, x_n \rangle$ 是 I' 的优化解, $X' = \sum_{1 \leq i \leq n} v'_i x_i$ 是 I' 最优代价

• $\langle z_1, \dots, z_n \rangle$ 是 I' 的近似解 $\Rightarrow Z' \leq X'$

• $v'_i = \lfloor v_i \times (K/v_{max}) \rfloor \leq (K/v_{max}) v_i \Rightarrow X' \leq (K/v_{max}) X$

$$\begin{aligned} X &\geq (v_{max}/K) X' \geq (v_{max}/K) Z' && \text{(联立两个不等式)} \\ &= (v_{max}/K) \sum_{1 \leq i \leq n} v'_i z_i && \text{(Z'的定义)} \\ &= (v_{max}/K) \sum_{1 \leq i \leq n} \lfloor v_i \times (K/v_{max}) \rfloor z_i \\ &\geq (v_{max}/K) \sum_{1 \leq i \leq n} [v_i \times (K/v_{max}) - 1] z_i \\ &= \sum_{1 \leq i \leq n} v_i z_i - (v_{max}/K) \sum_{1 \leq i \leq n} z_i \\ &= Z - \epsilon v_{max} \\ &\geq Z(1-\epsilon) \end{aligned}$$



9.5.3 Bin-Packing 问题的近似模式

- 问题定义
- 问题的变形
- 多项式近似模式
- 近似比分析



问题的定义

• 输入

体积依赖向量 $a_1, \dots, a_n \in (0,1]$ 的 n 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



基本想法—给定实例 I, ϵ

- 小体积物品对优化解的影响较小
 - > 多个小体积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



算法框架

算法 ApproxBinPacking(I, ε)

输入: 装箱问题的实例 I 和相对误差参数 $\varepsilon < 1$
输出: I 的一个近似最优的装箱方案;

- $I', I^{down}, I^{up} \leftarrow \text{Transform}(I, \varepsilon)$; /*变换*/
- $S' \leftarrow \text{DynamicSearch}(I^{up}, 1/\varepsilon^2)$; /*DP*/
- $S \leftarrow \text{SolutionTrans}(S', I, \varepsilon)$; /*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 $O(n^{2/\varepsilon^2})$ 的算法



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}

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 $n\varepsilon^2$ 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 $O(n \log n)$

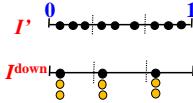
Transform(I, ε) 算法的性质

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

证明:

- $\text{Opt}(I^{down}) \leq \text{Opt}(I')$
 - I' 的每个可行解也是 I^{down} 的可行解
- $n\varepsilon \leq \text{Opt}(I')$
 - 所有物品的总体积至少为 $n\varepsilon$



Transform(I, ε) 算法的性质

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) $\text{Opt}(I^{down}) \leq \text{Opt}(I')$, 且 $n\varepsilon \leq \text{Opt}(I')$, 其中 n 是 I' 中物品个数;
- (4) $\text{Opt}(I^{up}) \leq (1+\varepsilon)\text{Opt}(I')$;

证明: I^{down} 的优化解 S' 可修改得到 I^{up} 的可行解 S

- 所有箱子去除 I^{down} 第 1 组的物品
- 所有箱子属于 I^{down} 第 i 组的物品
替换为 I^{up} 第 $i-1$ 组的物品
- I^{up} 最后一组每个物品用一个新箱子, 至多新增 $n\varepsilon^2$ 箱子

$$\text{Opt}(I^{up}) \leq \text{Opt}(I') + n\varepsilon^2 \leq \text{Opt}(I') + \varepsilon\text{Opt}(I') = (1+\varepsilon)\text{Opt}(I')$$



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 $O(n^2)$

近似比分析

$$I \xrightarrow{\epsilon} I' \xrightarrow{\text{随机}} I^{up}$$

- I^{up} 的优化解作为 I' 的近似解 S $Opt(I^{up}) \leq (1+\epsilon) Opt(I)$
 - S 中使用的箱子个数即为 $Opt(I^{up})$
- SolutionTrans 断开的箱子个数记为 new ;
- $Approx(I) = Opt(I^{up}) + new$
- 若 $new=0$, 则
- $Approx(I) = Opt(I^{up}) + new \leq (1+\epsilon) Opt(I) \leq (1+2\epsilon) Opt(I)$
- 若 $new \neq 0$, 则
 - 近似解中, 除最后一个箱子之外, 每个箱子的空间都小于 ϵ
 - 这些箱子内物品总体积 $> (1-\epsilon)[Approx(I) - 1]$, 但 $< \text{所有物品总体积}$
 - $Opt(I) \geq \text{所有物品总体积}$
 - $(1-\epsilon)[Approx(I) - 1] \leq Opt(I)$
 - $Approx(I) \leq [1/(1-\epsilon)]Opt(I) + 1 \leq (1+2\epsilon)Opt(I)$

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



HIT
CS&E

求解 Max-3-CNF 问题随机近似算法

• 基本概念

定义1. 设 C 是随机近似算法 RAS 产生的问题 P 的近似解的代价, C^* 是问题 P 的准确解的代价, n 是 P 的大小. 若 $\max(C/C^*, C^*/C) \leq p(n)$, 则称 RSA 具有近似比 $p(n)$. 我们也称 RAS 是一个随机 $p(n)$ -近似算法.



HIT
CS&E

• Max-3-CNF 问题的定义

输入: 合取范式 CNF,

每个析取式具有三个变量,

没有任何变量和它的非在同一个析取式中

输出: 一个变量赋值, 最大化值为 1 的析取式个数

• 随机算法

Random-Max-3-CNF(CNF)

1. For 对于 CNF 中的每个变量 x Do

2. 随机地给 x 赋值: $x=0$ 的概率为 $1/2$, $x=1$ 的概率为 $1/2$;

3. Return.

• 性能分析

定理. Random-Max-3-CNF 是一个随机 $8/7$ -近似算法.

证.

假定输入 CNF 中具有 n 个变量, m 个析取式, 第 i 个析取式的形

式为 $x_{i1} \vee x_{i2} \vee x_{i3}$.

对 $i=1, 2, \dots, m$, 定义随机变量:

$Y_i=1$ 如果第 i 个析取式为 1, 否则 $Y_i=0$.

$Pr(\text{第 } i \text{ 个析取式为 } 0) = Pr(x_{i1}=0)Pr(x_{i2}=0)Pr(x_{i3}=0) = (1/2)^3 = 1/8$.

$Pr(\text{第 } i \text{ 个析取式为 } 1) = 1 - 1/8 = 7/8$.

$E[Y_i] = 7/8$.

令 $Y = Y_1 + Y_2 + \dots + Y_m$. Y 是 CNF 中值为 1 的析取式的个数.

$E[Y] = \sum_{1 \leq i \leq m} E[Y_i] = \sum_{1 \leq i \leq m} 7/8 = m \cdot 7/8$.

显然, 优化解的代价为 m . 于是近似比 $= m / (m \cdot 7/8) = 8/7$.



HIT
CS&E

9.6 基于线性规划的近似算法

9.6.1 线性规划与对偶原理

9.6.2 max-min 关系

9.6.3 用线性规划设计近似算法的两类方法

9.6.4 应用实例一: 顶点覆盖问题的贪心算法

9.6.5 应用实例二: 集合覆盖问题的近似算法



HIT
CS&E

9.6.1 线性规划

线性规划问题: 在约束条件下通过线性表达式的形式进行优化

$$\text{例 1} \quad \begin{aligned} & \text{minimize } 7x_1 + x_2 + 5x_3 \\ & \text{subject to } x_1 - x_2 + 3x_3 \geq 10 \end{aligned}$$

$$\begin{aligned} & \text{maximize } 10y_1 + 6y_2 \\ & \text{subject to } y_1 + 5y_2 \leq 7 \end{aligned}$$

$$5x_1 + 2x_2 - x_3 \geq 6 \quad -y_1 + 2y_2 \leq 1$$

$$x_1, x_2, x_3 \geq 0 \quad 3y_1 - y_2 \leq 5$$

线性规划的一般形式

$$-y_1 - y_2 \leq 0$$

最小化问题

最大化问题

$$\min cx$$

$$\max by$$

$$\text{st. } Ax \geq b$$

$$\text{st. } By \leq c$$



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

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

$x=(2,1,3)$ 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



线性规划问题的对偶问题

$$\begin{aligned} & \text{minimize } 7x_1 + x_2 + 5x_3 \\ & \text{subject to } x_1 - x_2 + 3x_3 \geq 10 \\ & \quad 5x_1 + 2x_2 - x_3 \geq 6 \\ & \quad x_1, x_2, x_3 \geq 0 \\ & y_1(x_1 - x_2 + 3x_3) \geq 10y_1 \\ & y_2(5x_1 + 2x_2 - x_3) \geq 6y_2 \end{aligned}$$

上述线性规划问题中, 目标函数的最优值 z^* 至多是 a 吗?

$$\begin{aligned} & \text{由可行解: } x=(2,1,3) \text{ 和 } z^* \leq 30 \\ & Z^* \text{ 的上界最小应该是多少?} \\ & 7x_1 + x_2 + 5x_3 \geq y_1(x_1 - x_2 + 3x_3) + y_2(5x_1 + 2x_2 - x_3) \geq 10y_1 + 6y_2 \\ & (y_1 + 5y_2)x_1 + (-y_1 + 2y_2)x_2 + (3y_1 - y_2)x_3 \end{aligned}$$

对偶问题



线性规划问题的对偶问题

$$\begin{array}{ll} \text{minimize } & 7x_1 + x_2 + 5x_3 \\ \text{subject to } & x_1 - x_2 + 3x_3 \geq 10 \\ & 5x_1 + 2x_2 - x_3 \geq 6 \\ & x_1, x_2, x_3 \geq 0 \\ y_1(x_1 - x_2 + 3x_3) \geq 10y_1 & \\ y_2(5x_1 + 2x_2 - x_3) \geq 6y_2 & \text{对偶问题} \\ 7x_1 + x_2 + 5x_3 \geq y_1(x_1 - x_2 + 3x_3) + y_2(5x_1 + 2x_2 - x_3) \geq 10y_1 + 6y_2 & \\ (y_1 + 5y_2)x_1 + (-y_1 + 2y_2)x_2 + (3y_1 - y_2)x_3 & \end{array}$$

对偶问题

26

原问题

对于一般的线性规划问题

$$\text{Min } cx$$

$$\text{St } Ax \geq b$$

$$x \geq 0$$

原问题

真对偶问题

$$\text{Max } b^T y$$

$$\text{St } A^T y \leq c^T$$

对偶问题

$$y \geq 0$$



对偶定理

定理1. 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 $x^* = (x_1^*, \dots, x_n^*)$ 和 $y^* = (y_1^*, \dots, y_m^*)$ 分别是原问题和对偶问题的最优解, 则 $cx^* = b^T y^*$ 。

定理2. 在线性规划问题中, 如果 $x = (x_1, \dots, x_n)$ 和 $y = (y_1, \dots, y_m)$ 分别是原问题和对偶问题的可行解, 则 $cx \geq by$ 。

证明:



HIT
CS&E

由于 y 是对偶问题的可行解, 且 x_j 非负

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

由于 x 是对偶问题的可行解, 且 y_i 非负

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

注意到

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

证毕



定理3. 如果 $x=(x_1, \dots, x_n)$ 和 $y=(y_1, \dots, y_m)$ 分别是原问题和对偶问题的可行解，则 x 和 y 分别是原问题和对偶问题的最优解当且仅当下面的条件同时成立：

原问题的互补松弛条件：

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

对偶问题的互补松弛条件：

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

定理4. 如果 $x=(x_1, \dots, x_n)$ 和 $y=(y_1, \dots, y_m)$ 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 $1 \leq j \leq n$: $x_j = 0$ 或者 $\frac{c_j}{\alpha} \leq \sum_{i=1}^m a_{ij} y_i \leq c_j$

对偶问题的互补松弛条件： $\beta \geq 1$

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

则

$$\sum_{j=1}^n c_j x_j \leq \alpha \cdot \beta \cdot \sum_{i=1}^m b_i y_i$$

证明： $\sum_{j=1}^n c_j x_j \leq \alpha \sum_{j=1}^n (\sum_{i=1}^m a_{ij} y_i) x_j = \alpha \sum_{i=1}^m (\sum_{j=1}^n a_{ij} x_j) y_i \leq \alpha \cdot \beta \cdot \sum_{i=1}^m b_i y_i$

基于 Primal-dual schema 的近似算法以定理4为理论基础



9.6.2 max-min 关系

最大流问题

输入：有向图 $G=(V,E)$, 源顶点 $s \in V$, 接收顶点 $t \in V$, 每条

边 e 的流量限制 $c(e) > 0$.

输出：从 s 到 t 的最大流。

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

流量约束： $f(e) \leq c(e)$

守恒约束： $\sum_{uv \in E} f(uv) = \sum_{vu \in E} f(v^* u) \quad u \in V \quad u \neq s, u \neq t$



最小割问题

输入：有向图 $G=(V,E)$, 源顶点 $s \in V$, 接收顶点 $t \in V$, 每条

边 e 的流量限制 $c(e) > 0$.

输出： s 和 t 之间的最小割

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

最小割问题与最大流问题间的 max-min 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



最大流问题的线性规划表示

$\max f_s$ $s.t. \quad f_{\bar{j}} \leq c_{ij} \quad \bar{j} \in E$ $\sum_{j:j \in E} f_{\bar{j}} - \sum_{j:j \in E} f_{\bar{j}} \leq 0 \quad i \in V$ $f_{\bar{j}} \geq 0 \quad \bar{j} \in E$	$\min \sum_{\bar{j} \in E} c_{ij} d_{\bar{j}}$ $s.t. \quad d_{\bar{j}} - p_i + p_{\bar{j}} \geq 0 \quad \bar{j} \in E$ $p_s - p_t \geq 1$ $d_{\bar{j}} \geq 0 \quad \bar{j} \in E$ $p_i \geq 0 \quad i \in V$
---	---

考虑整数规划的解
 $p_s^*=1, p_t^*=0$
 $X=\{i | p_i^*=1\} \quad V-X=\{i | p_i^*=0\}$
 $X, V-X$ 是一个最小 s, t -割



9.6.3 两类基于 LP 方法的近似算法

很多组合优化问题可以表达成整数线性规划问题

将整数约束条件放宽，即得到一个线性规划问题 LP-松弛问题

线性规划问题可以用 Karmarkar 算法在多项式时间内求解

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。



9.6.4 应用实例之一

- 求解最小节点覆盖问题的线性规划算法

- 问题的定义

- 输入: 无向图 $G=(V, E)$, 每个节点具有权 $w(v)$.
- 输出: $C \subseteq V$, 满足
 - $\forall (u, v) \in E, u \in C$ 或者 $v \in C$
 - $w(C)$ 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线性规划问题 P_{0-1}

- 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 P_{0-1}

- 优化目标: 最小化 $\sum_{v \in V} w(v)x(v)$
- 约束条件: $x(u)+x(v) \geq 1 \quad \text{for } \forall uv \in E$
 $x(v) \in \{0, 1\} \quad \text{for } \forall v \in V$

- 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规划问题的解

- 对于 $\forall v \in V$, 定义 $x(v) \in [0, 1]$

- P_{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: $x(u)+x(v) \geq 1 \quad \text{for } \forall v \in V$
 $x(v) \in [0, 1] \quad \text{for } \forall v \in V$

- 线性规划问题具有多项式时间算法

- P_{0-1} 的可能解是 LP 问题的可能解

- P_{0-1} 解的代价 $\geq LP$ 的解的代价



- 近似算法

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
 - 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_{0-1} 的解 */
5. Return C .

- 算法的性能

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.



推证 $w(C)/w(C^*) \leq 2$.

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

$$\begin{aligned} z^* &= \sum_{v \in V} w(v)x(v) \geq \sum_{v \in V: x(v) \geq 1/2} w(v)x(v) \\ &\geq \sum_{v \in V: x(v) \geq 1/2} w(v)/2 \\ &= \sum_{v \in C} w(v)/2 \\ &= (1/2) \sum_{v \in C} w(v) \\ &= (1/2)w(C). \end{aligned}$$

由 $w(C^*) \geq z^*$, $w(C^*) \geq (1/2)w(C)$, 即 $w(C)/w(C^*) \leq 2$.



9.6.5 应用实例之二

集合覆盖问题

集合覆盖问题的线性规划表示

对偶松合方法

舍入法

随机会入方法

Primal-dual schema



问题的定义

• 输入:

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

• 输出:

$C \subseteq F$, 满足

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

(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.



集合覆盖问题的线性规划表示

对 F 中的每个集合 S , 引入一个变量 x_S

$x_S = 0$ 表示 $S \notin C$

原问题的线性规划表示

$$\begin{aligned} &\text{minimize}_{x_S} \sum_{S \in F} C(S) \cdot x_S \\ &\text{subject to} \sum_{S \in e} x_S \geq 1 \quad e \in X \\ &\quad x_S \in \{0,1\} \quad S \in F \end{aligned}$$

LP 松弛问题

$$\begin{aligned} &\text{minimize}_{x_S} \sum_{S \in F} C(S) \cdot x_S \\ &\text{subject to} \sum_{S \in e} x_S \geq 1 \quad e \in X \\ &\quad x_S \geq 0 \quad S \in F \end{aligned}$$

OPT | 原问题
LP 松弛问题



LP 松弛问题 $\text{minimize}_{x_S} \sum_{S \in F} C(S) \cdot x_S$

$$\begin{aligned} &\text{subject to} \sum_{S \in e} x_S \geq 1 \quad e \in X \\ &\quad x_S \geq 0 \quad S \in F \end{aligned}$$

对偶问题

$$\begin{aligned} &\text{maximize}_{y_e} \sum_{e \in X} y_e \\ &\text{subject to} \sum_{e \in S} y_e \leq C(S) \quad S \in F \\ &\quad y_e \geq 0 \quad e \in X \end{aligned}$$

OPT_f

OPT

原问题

对偶问题

LP 松弛问题



贪心算法 (对偶松合方法)

贪心算法:

1. $U = \emptyset$; $C = \emptyset$;
2. while $U \neq X$ do
 3. 从 F 中挑选一个集合 S 使得 $c(S)/|S-U|$ 最小;
 4. $\alpha = c(S)/|S-U|$;
 5. 对任意 $e \in S-U$, 令 $price(e) = \alpha$;
 6. $C = C \cup \{S\}$, $U = U \cup S$, $F = F - \{S\}$;
 7. 输出 C

C 的代价是 $\sum_{e \in X} price(e)$

引理1. 对任意 $e \in X$ 令 $y_e = \text{price}(e)/H_n$, 则由所有 y_e 构成的向量 y 是对偶问题的一个可行解。

证明: 既然 $y \geq 0$ 对任意 $e \in X$ 成立, 只需对任意 $S \in F$ 验证 $\sum_{e \in S} y_e \leq c(S)$ 成立。

假设将 S 中所有元素按它们在算法中被覆盖的先后次序列出为 e_1, \dots, e_k

考察 e_i 第一次被覆盖的时刻, 由于 S 中此时还有 $k-i+1$ 个元素未被覆盖, 将 $c(S)$ 平摊到这些元素上, 每个元素分得的代价是 $c(S)/(k-i+1)$ 。

由于在覆盖 e_i 时, S 本身也是候选集合, 直接将集合 S' 时要求 $c(S')/|S'-F|$ 达到最小, 故 $\text{price}(e_i) \leq c(S)/(k-i+1)$ 。进而

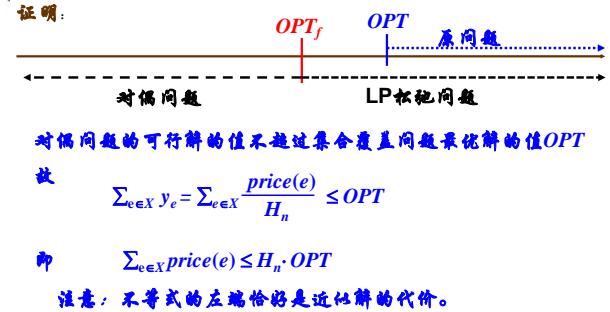
$$y_{e_i} \leq \frac{1}{H_n} \cdot \frac{c(S)}{k-i+1}$$

故

$$\sum_{e \in S} y_e = \sum_{i=1}^k y_{e_i} \leq \frac{c(S)}{H_n} \left(\frac{1}{k} + \frac{1}{k-1} + \dots + 1 \right) = \frac{H_k}{H_n} C(S) \leq C(S)$$

定理5. 贪心算法的近似比是 H_n 。

证明:



对偶问题的可行解的值不超过集合覆盖问题最优解的值 OPT

故

$$\sum_{e \in X} y_e = \sum_{e \in X} \frac{\text{price}(e)}{H_n} \leq OPT$$

即

$$\sum_{e \in X} \text{price}(e) \leq H_n \cdot OPT$$

注意: 不等式的左端恰好是近似解的代价。



舍入法

频率: 对于 $e \in X$, e 的频率指的是 F 中包含 e 的集合的个数

f : X 中元素的最大频率

集合覆盖问题的 LP-舍入算法

1. 用 Karmarkar 算法求得 LP-松弛问题的最优解 x

2. For $S \in F$ Do

 IF $x_S \geq 1/f$ THEN $C = C \cup \{S\}$

3. 输出 C

定理6. 对于集合覆盖问题, LP-舍入算法的近似比是 f 。

证明:

对于任意 $e \in X$, 由于 e 至多属于 f 个集合中, 为了确保

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

必有某个 x_S 使得 $x_S \geq 1/f$ 。因此, 算法输出的集族中必有一个集合包含了 e , 进而, 算法的输出覆盖了 X 。

在舍入过程中, 对任意 $S \in C$, x_S 被舍入为 1, 至多被放大 f 倍。因此

$$OPT_f = \sum_{S \in S} c(S)x_S = \sum_{S \in C} c(S)x_S + \sum_{S \in C} c(S)x_S \geq \sum_{S \in C} \frac{1}{f} + \sum_{S \in C} c(S)x_S \geq \frac{1}{f} \sum_{S \in C} c(S)$$

从而, $\sum_{S \in C} c(S) \leq f \cdot OPT_f \leq f \cdot OPT$.

证毕



集合覆盖问题的 LP-随机舍入算法

LP-随机舍入算法

1. 用 Karmarkar 算法求解 LP-松弛问题得到最优解 $x = \langle x_S : S \in F \rangle$

2. $C = \emptyset$

3. For $\forall S \in F$ Do

4. 独立地产生一个随机数 $rand$

5. IF $rand > 1 - x_S$ THEN $C = C \cup \{S\}$;

 /* S 被选入 C 的概率是 x_S */

6. 输出 C

定理7. 对于集合覆盖问题的 LP-随机舍入算法, C 的代价的数学期望是 OPT_f , 其中 OPT_f 是 LP-松弛问题的最优解的值。

证明: $E(\text{cost}(C)) = \sum_{S \in F} p_r[S \text{ 被选入 } C] \cdot c(S)$

$$= \sum_{S \in F} x_S \cdot c(S)$$

$$= OPT_f$$

定理8. 对于集合覆盖问题的 LP-随机舍入算法, $\forall a \in X$ 被 C 覆盖的概率大于 $1 - 1/e$ 。

证明:

设 a 属于 F 的 k 个集合中, 将 LP-松弛问题中这些集合对应的变量记为 x_1, \dots, x_k .

在 LP-松弛问题的优化解中, $x_1 + \dots + x_k \geq 1$.

$$Pr[a \text{ 被 } C \text{ 覆盖}] = (1-x_1)(1-x_2)\dots(1-x_k)$$

$$\leq (1-(x_1+\dots+x_k)/k)^k$$

$$\leq (1-1/k)^k$$

$$Pr[a \text{ 被 } C \text{ 覆盖}] = 1 - Pr[a \text{ 不被 } C \text{ 覆盖}]$$

$$\geq 1 - (1-1/k)^k$$

$$\geq 1-1/e$$

改造策略：为了得到完整的集合覆盖，独立运行LP-随机贪心入算法 $c \log n$ 次，其中 c 满足 $\frac{1}{e} \cdot c \log n \leq \frac{1}{4n}$ ，将所有输出集合求并得到 C' ，然后输出 C' 。

$$Pr[C' \text{ 能覆盖 } X] \leq \sum_{a \in X} Pr[C' \text{ 能覆盖 } a] \leq n \cdot [(1/e)^{c \log n}] = 1/4$$

$$E(\text{cost}(C')) = OPT_f \cdot c \cdot \log n$$

$$Pr[\text{cost}(C') \geq OPT_f / 4c \log n] \leq 1/4$$

$$Pr[C' \text{ 能覆盖 } X \text{ 且 cost}(C') \leq OPT_f / 4c \log n] = 1 - Pr[C' \text{ 能覆盖 } X \text{ 且 cost}(C') \geq OPT_f / 4c \log n] \\ \geq 1 - (1/4 + 1/4) = 1/2$$



Primal-dual schema

基于 Primal-dual Schema 的集合覆盖近似算法

1. $x \leftarrow 0$; /* 向量 F 中的每个集合 S 对应一个分量 x_s */
2. $y \leftarrow 0$; /* 向量 X 中的每个元素 e 对应一个分量 y_e */
3. $U \leftarrow \emptyset$; /* 记录已被覆盖的元素 */
4. while $U \neq X$ Do
5. 取 $e_0 \in X - U$;
6. 增大 y_{e_0} 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 $\sum_{e: e \in S} y_e = c(S)$ 的任意 $S \in F$, 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 C ;

引理2. 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. 算法执行时, $0 = \sum_{e: e \in S} y_e \leq c(S)$ 对任意 $S \in F$ 成立。

算法执行过程中, 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, 因此在算法执行后运行的各个阶段向第 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, $\sum_{e: e \in S} y_e \leq c(S)$ 对任意 $S \in F$ 成立。

引理3. 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 ; 从而结合(2)成立。

定理9. 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 2 和引理 3, 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 $c(S)/f \leq \sum_{e: e \in S} y_e \leq c(S)$;

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

这恰恰是定理 4 中的条件 ($\alpha=1$, $\beta=f$)。

由定理 4, $\text{cost}(C) = \sum_{S: S \in F} c(S) = \sum_{S: S \in F} x_s c(S) \leq 1/f \cdot \sum_{e \in X} y_e$

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

在算法设计过程中, 我们实际上要先寻找恰当的 α 和 β 使得定理 4 中的条件得到满足, 然后用算法确保该条件成立。通常, 我们往往固定 α 或 β 为 1, 仅让另一个参数变化。算法近似比的好坏, 往往取决于所确定的参数的优劣。