A Brief Introduction to the Approximation Algorithm

He Huang
Soochow University
huangh@suda.edu.cn

1

Introduction

NP完全性理论讲解部分我们已经提及——现实中许多优化问题是NP-hard的，由复杂性理论知：若P≠NP(很可能为真)，就不可能找到多项式时间的算法来对问题的所有输入实例求出最优解。但若放松要求，就可能存在有效求解算法。

实用中可考虑3种放宽要求的可能性：

1.超多项式时间启发

不再要求多项式时间算法，有时求解问题存在超多项式时间算法，实用中相当快。例如，0/1背包问题是NPC问题，但存在1个伪多项式时间算法很容易解决它.

缺点：该技术只对少数问题有效(弱NPC问题)

2

2.启发式概率分析

不再要求问题的解满足所有的输入实例。在某些应用中，有可能输入实例的类被严格限制，对这些受限实例易于找到其有效算法。而这种结果往往归功于对输入实例约束的概率模型。

缺点：选取一个特殊的输入分布往往是不易的。

3.近似算法

不再要求总是找到最优解。在实际应用中有时很难确定一个最优解和近似最优解(次优解)之间的差别。

设计一个高效求解算法能够求出所有情况下的次优解来求解NP-hard问题往往是真正有效的手段。

3

Preliminaries and some basic concepts

定义1: 一个优化问题Π由三部分构成：

- 实例集D：输入实例的集合
- 解集S(I)：输入实例I∈D的所有可行解的集合
- 解的值函数f：给每个解赋一个值，f：S(I)→R

■一个最大值优化问题Π是：

对于给定的I∈D，找一个解 $\sigma_{opt}^{I} \in S(I)$ 使得：

$$\forall \sigma \in S(I),\ f(\sigma_{opt}^{I}) \geq f(\sigma)$$

此最优解的值称为OPT(I),即：$OPT(I) \triangleq f(\sigma_{opt}^{I})$

有时不太严格地可称其为最优解

4

■装箱问题(BP)

非形式地，给定一个size在0,1之间的项的集合，要求将其放入单位size的箱子中，使得所用的箱子数目最少。故有最小优化问题：

1)实例: $I=\{s_1, s_2, \ldots, s_n\}$, 满足$\forall i, s_i \in [0,1]$

2)解: $\sigma=\{B_1, B_2, \ldots, B_k\}$是I的一个不相交的划分，使得

$$\forall i, B_i \subset I \text{ 且 } \sum_{j \in B_i} S_j \leq 1$$

3)解的值：使用的箱子数,即 $f(\sigma) = |\sigma| = k$

最小优化问题是求最小的k

5

■约定

1)f的值域和I里的所有数是整数

易于扩展至有理数

2)对任何σ ∈S(I)，f(σ)受限于多项式(对I中数的size)

只讨论NP完全的优化问题，因为它易被转换为判定问题

定义2： 若一个NPH判定问题Π_1是多项式可归约为计算一个优化问题Π_2的解，则Π_2是NPH的 。

典型情况是问题Π_1是问题Π_2的判定版本。若Π_2是最大值问题，则Π_1的形式为：

“是否存在σ ∈S(I)，使得f(σ) ≥k？”

6

定义3：一个近似算法A，是一个多项式时间求解优化问题Π的算法，使得对一个给定的Π的输入实例I，它输出某个解σ∈S(I)。通常，我们用A(I)表示算法A所获得的解的值f(σ)。(有时不太严格地也可表示其解)

■近似算法的性能

算法质量是在最优解和近似解之间建立某种关系，这种度量也称为性能保证(Performance guarantee)

7

绝对性能保证

在可行的时间内，求装箱问题的最优解是不可能的，但可求次最优解是多少？显然，比最优解多使用1个箱子的解是次最优的。一般地，我们希望找到1个近似解，其值与最优解的值相差一个小的常数。

定义4：绝对性能度量 一个绝对近似算法是优化问题Π的多项式时间近似算法A，使得对某一常数k>0满足：

$$\forall I \in D,\ |A(I)-OPT(I)| \le k$$

显然,我们期望对任何NP-hard问题都有一个绝对近似算法，但是对于大多数NP-hard问题，仅当P=NP时，才能找到绝对近似算法(指多项式时间)。

8

绝对近似算法

1、图的顶点着色

使用最少的颜色数来为图G的顶点上色，使得所有相邻的顶点均有不同的颜色，即使G是平面图，该问题的判定版本也是NP-hard的，它有1个绝对近似算法。

定理1：判定一个平面图是否可3着色的问题是NPC的。

近似算法A(G) {//对任意平面图G染色

1)检验G是否可2染色(即G是二分图?),若是则G可2染色;

2)否则,计算5染色;//可在多项式时间内,∵任何平面图G是//可5染色的(实际上四色定理告诉我们G是可4染色的)

} 这就证明了算法A比最优解多用的颜色数不会超过2：

定理2：对给定的任意平面图G，近似算法A的性能满足:

$$|A(G)-OPT(G)| \le 2$$

9

绝对近似算法

2、图的边着色

使用最少的颜色为图的边上色，使得所有相邻的边有不同的颜色。如下定理(Vizing)说明最大度数Δ与边着色数之关系：

定理3：任一图至少需要Δ，至多需要Δ+1种颜色为边着色

Vizing定理的证明给出了一个多项式时间的算法A找到Δ+1边着色，但令人惊奇的是边着色问题即使是很特殊的情况也是NP-hard的，正如下述的Holyer定理：

定理4：确定一个3正则平面图所需的边着色数问题是NPH.

综上所述：存在绝对近似算法A，使得下述定理成立：

定理5：近似算法A有性能保证：|A(G)-OPT(G)| ≤ 1

10

绝对近似算法存在性

前面的例子似乎说明只有很特殊的一类优化问题可能有绝对近似算法：已知最优解的值或值所在的小范围。但最优解的值不易确定时是否有绝对近似算法仍然是open的；

对于大多数NP-hard问题，存在绝对近似算法(多项式时间)当且仅当存在多项式的精确算法(即：可在多项式时间内找到最优解)；

否定结果：证明问题的绝对近似算法的不存在性！

11

■0/1背包问题 问题实例：

- 项　　集：I={1,2,…,n}
- 大　　小：$s_1, s_2, \ldots, s_n$
- 利　　润：$p_1, p_2, \ldots, p_n$
- 背包容量：B

问题的一个可行解是子集I'⊆I，$\sum_{i\in I'} s_i \le B$

最优解是使得f(I') = $\sum_{i\in I'} p_i$ 最大的可行解

该问题(0/1背包)是NP-hard的，除非存在多项式时间算法能够找到最优解，否则不存在绝对近似算法。

12

■定理6 若P≠NP，则对任何确定的k，找不到近似算法A可解背包问题使得：|A(I)-OPT(I)| ≤ k

Pf：使用扩放法反证。

假定存在多项式时间算法A具有性能保证k(注意k是正整数)

设∀I∈D，可构造新实例I'使得

$s_i'=s_i$ (weight)，$p_i'=(k+1)p_i$ (profit) for $i\in[1,n]$

即：除了利润扩放k+1倍之外，其余参数不变。故I的可行解也是I'的可行解，反之亦然。只是解的值相差k+1倍。

在I'上运行算法A获得解A(I')，设A在实例I上的解是σ：

$|A(I')-OPT(I')| \le k \Rightarrow |(k+1)f(\sigma)-(k+1)OPT(I)| \le k$

$\Rightarrow |f(\sigma)-OPT(I)| \le k/(k+1) \Rightarrow |f(\sigma)-OPT(I)|=0$

即：我们已经找到了一个多项式时间的算法A，它对背包问题的任一输入实例I，均能找到最优解。

上述证明之关键是Scaling性质，输入实例中数字间线性相关很易使其成立。对非数值问题Scaling性质是否仍然成立？

■团(Clique)问题 找图G中最大团(最大完全子图)，该问题是NP-hard的。

定理7 若P≠NP，则对于团问题不存在绝对近似算法

Pf：定义图的m次幂G^m：取G的m个拷贝，连接位于不同副本里的任意两顶点。

Claim：G中最大团的size为α当且仅当G^m里最大团的size是m^α(Ex.)

14

反证：设G是任意的无向图，近似算法A给出的绝对误差是k。在G^{k+1}上运行A，若G中最大团size为α,则我们有

$|A(G^{k+1})-OPT(G^{k+1})| \le k \Rightarrow |A(G^{k+1})-(k+1)OPT(G)| \le k$

对于任给的G^m中体积为β的团，易于用多项式时间在G中找到一个体积为β/m的团。因此我们能够在G中找到一个团C，使得：

$$||C|-OPT(G)| \le k/(k+1)$$

因为|C|和OPT(G)均是整数，故A(G)=|C|是最优团。

即：多项式时间内找到了G的最优解A(G)

15

近似算法和近似比

近似算法

■当NP类问题的输入极大时，要求该问题的最优解，算法的执行时间将会是指数级别的。这时，我们很难找到问题的最优解(即使能够找到，花费的代价也是极大的)。因此我们通常会选择在多项式时间内再到问题的一个近似解。

近似比 (Approximation Ratio)

设一个最优化问题的最优值为C^*，该问题的近似算法求得了近似最优值C，有近似比

$$\eta = \max\left\{\frac{C}{C^*}, \frac{C^*}{C}\right\}$$ （最大化、最小化问题均适用）

一个能求得最优解的近似算法得到的近似比为1

最大化问题中，如果$C \le C^*$，则最优解C^*比近似解C大$\left(\frac{C^*}{C}-1\right)$倍

16

0-1背包问题的定义

0-1背包问题

■给定n种物品和一背包。物品i的重量是w_i，其价值为v_i，背包的容量为C。应如何选择装入背包的物品，使得装入背包中物品的总价值最大?

$$\max \sum_{i=0}^{n} v_i x_i$$

$$\begin{cases} \sum_{i=0}^{n} w_i x_i \le C \\ x_i \in \{0,1\}, 1 \le i \le n \end{cases}$$

17

0-1背包问题是一个NP完全问题

0-1背包问题的DP求解

■ 递归式

$$m(i,j) = \begin{cases} \max\{m(i+1,j), m(i+1,j-w_i)+v_i\} & j \ge w_i \\ m(i+1,j) & 0 \le j \le w_i \end{cases}$$

■ 算法复杂度分析：

从$m(i,j)$的递归式容易看出，算法需要$O(n \cdot C)$计算时间，C表示背包容量。当$C > 2^n$时，算法需要$\Omega(n \cdot 2^n)$计算时间。

对于0-1背包问题，当问题的输入变得无穷大时，我们往往需要付出很大的代价才能得到问题的最优解。对此，我们希望设计一个近似算法能够保证该问题在多项式时间内被解决(即一个求解时间短很多的近似算法)。

18

Definition of the approximation scheme

近似方案 (Approximation Scheme)

■ 设π是目标函数为f_π的*NP-hard*最优化问题，如果关于输入(I,ε)，I为π的实例，ε为误差参数，算法$\mathcal{A}$输出解S使得

1) 若π是最小化问题，$f_\pi(I,S) \le (1+\varepsilon)\cdot OPT$

目标函数在输入为I的实例下输出解S小于$(1+\varepsilon)\cdot$ OPT

2) 若π是最大化问题，$f_\pi(I,S) \ge (1-\varepsilon)\cdot OPT$

我们称算法$\mathcal{A}$是π的近似方案。

19

PTAS和FPTAS的定义

PTAS (Polynomial-Time Approximation Scheme)

■ 若对于每一个固定的$\varepsilon > 0$，算法$\mathcal{A}$的运行时间以实例I的规模的多项式为上界，则称$\mathcal{A}$是一个多项式时间近似方案（简称PTAS）。

FPTAS (Fully Polynomial-Time Approximation Scheme)

■ 在PTAS的基础上，我们进一步要求算法$\mathcal{A}$，即算法$\mathcal{A}$的运行时间以实例I的规模和$1/\varepsilon$的多项式为上界，则称$\mathcal{A}$是一个完全多项式时间近似方案（简称FPTAS）。

FPTAS被认为是最值得研究的近似算法，

仅有极少的*NP-hard*问题存在FPTAS。

20

伪多项式时间算法

■从给出近似方案的定义到确定FPTAS的过程中，我们的定义始终基于了假设，即出现在实例 I 中的数都以二进制的形式存放的，同时我们用$|I|$表示在该假定下实例I的规模。我们用I_u表示把出现的所有数以一进制形式存放的实例I，定义实例I的一进制规模为记录I_u所需的一进制位数，记作$|I_u|$。

■如果π的算法在实例I上的运行时间以$|I|$的多项式为上界，则称算法是有效的。如果π的算法在实例I上的运行时间以$|I_u|$的多项式为上界，则称算法是伪多项式时间算法。

21

一个0-1背包的伪多项式时间算法

■问题

给定物体集合$S=\{a_1,\dots,a_n\}$，每个物体有指定的大小$size(a_i)\in\mathbb{Z}^+$和指定的收益$profit(a_i)\in\mathbb{Z}^+$，以及背包容量$B\in\mathbb{Z}^+$，找一个子集，该子集的总大小以$B$为上界并且它的总收益最大。

■算法思路

定义P是收益最大的物体的收益，$P=\max_{a_i\in S} profit(a_i)$。

则nP是任何解能够获得的收益的一个平凡上界。

对于每个$i\in\{1,\dots,n\}$和$p\in\{1,\dots,nP\}$，设$S_{i,p}$表示$\{a_1,\dots,a_i\}$的一个子集，该子集的总收益正好是p，并且它的总大小Size达到最小。设$A(i,p)$表示集合$S_{i,p}$的大小(如果这样的集合不存在，则$A(i,p)=\infty$)。显然，对于每一个$p\in\{1,\dots,nP\}$，$A(1,p)$是已知的(递归的终止条件)。

22

一个0-1背包的伪多项式时间算法

我们可给出如下的递归式

$$A(i+1,p)=\begin{cases}\min\{A(i,p),size(a_{i+1})+A(i,p-profit(a_{i+1}))\} & if\ profit(a_{i+1})\le p\\ A(i,p) & otherwize\end{cases}$$

总大小以B为上界的各物体能够达到的最大收益为$\max\{p|A(n,p)\le B\}$。（p可能是指数级的）

上述递归可在$O(n^2P)$时间内计算出结果

↓

n个物品，每个物品对应1～nP，取值不同的子问题

23

0-1背包问题的FPTAS

■设计思路

若各物品收益均是比较小的数，即它们以n的多项式为上界，那么基于上述递推式所设计的DP算法是一个常规的多项式时间算法，因为它的运算时间以$|I|$(实例I的规模)的多项式为上界，但是，当p是指数级，就需要对物品价值进行放缩，即

忽略各物品价值的最后若干有效位(依赖于误差参数ε)

以便将修改后的收益看作是以n和$1/\varepsilon$的多项式为上界的数。

所以，我们可以在以n和$1/\varepsilon$的多项式为上界的时间内找到收益至少为$(1-\varepsilon)\cdot OPT$的解。

精度换取时间

24

算法 0-1背包问题的FPTAS

(实现方法 -- 缩放和取整)

STEP 1：给定$\varepsilon > 0$，设K = $\frac{\varepsilon P}{n}$.

STEP 2：对每个物品a_i，定义$profit'(a_i) = \left\lfloor \frac{profit(a_i)}{K} \right\rfloor$.

STEP 3：用这些值作为物品新的收益(价值)，利用前述动态规划算法找到收益最大的集合，记为S'.

STEP 4：输出S'.

25

定理1：算法的时间复杂度为$O\left(n^2 \cdot \left\lfloor \frac{n}{\varepsilon} \right\rfloor\right)$

（如$\varepsilon = 0.1$，能够保证DP算法的结果$\geq 0.9 \cdot OPT$，而算法的复杂度为$O(10n^3)$）

证明：代入新的物品价值后，DP所需要的时间

$$O(n^2 \cdot P) = O\left(n^2 \cdot \left\lfloor \frac{P}{K} \right\rfloor\right) = O\left(n^2 \cdot \left\lfloor \frac{P}{\frac{\varepsilon P}{n}} \right\rfloor\right) = O\left(n^2 \left\lfloor \frac{n}{\varepsilon} \right\rfloor\right)$$

26

定理2：算法输出的集合S'满足

$$profit(S') \geq (1-\varepsilon) \cdot OPT$$

证明：用O来表示未缩放之前的最优物品集合(对应最优解OPT)。对于任意物品a，我们在STEP 2中作了向下舍入操作。因此

$$\begin{cases} K \cdot profit'(a) \leq profit(a) \\ K \cdot profit'(a) > profit(a) - K \end{cases} \left(\Leftarrow \quad profit'(a) > \frac{profit(a)}{K} - 1\right)$$

有

$$profit(a) - K < K \cdot profit'(a) \leq profit(a)$$

27

由 $profit(a) - K < K \cdot profit'(a)$

得 $\forall i,\ profit(a) - K \cdot profit'(a) < K$

因此 $profit(O) - K \cdot profit'(O) < nK$

又因为DP算法所得解为最优解(S'是在新价值体系下至少与O一样好的集合)，因此

$$profit(S') \geq K \cdot profit'(O) \geq profit(O) - nK$$
$$= OPT - \varepsilon p$$
$$\because \quad OPT \geqq P$$
$$\therefore \quad profit(S') \geqq (1-\varepsilon) \cdot OPT$$

28

求解0-1背包问题的贪心算法

输入：正整数$W, w_1, v_1, w_2, v_2, \ldots, w_n, v_n$。

1) 依据v_i/w_i的大小，依单调递减序排列所有物品。
   不妨设$v_1/w_1 \geq v_2/w_2 \geq \cdots \geq v_n/w_n$
2) 若$\sum_{i=0}^{n} w_i \leq W$，则输出：$C_G \leftarrow \sum_{i=0}^{n} v_i$
   否则求出最大的k，使其满足：

$$\sum_{i=1}^{k} w_i \leq W < \sum_{i=1}^{k+1} w_i$$

   输出：$C_G \leftarrow max\{v_{k+1}, \sum_{i=1}^{k} v_i\}$

29

定理：对于背包问题的一个实例，设opt表示最优解目标函数值，C_G是由上述算法生成的近似解目标函数值，则$opt \leq 2C_G$(近似比为2)

证明：

1) 若$\sum_{i=0}^{n} w_i \leq W$，则$C_G = opt$
2) 若$\sum_{i=0}^{n} w_i > W$，设k是算法得到的整数
   可证$\sum_{i=1}^{k} v_i \leq opt < \sum_{i=1}^{k+1} v_i$

30

最优值*opt*的上界怎么算？

先放前k个物品，假定第$(k+1)$个物品可分割，将背包装满，用其他物品代替只会降低背包物品的价值/重量比值

OPT少于$\sum_{i=1}^{k+1} v_i$，最优值opt上界求解：

$$opt \le \hat{c} = \sum_{i=1}^{k} v_i + \frac{v_{k+1}}{w_{k+1}}\left(W - \sum_{i=1}^{k} w_i\right)$$
$$< \sum_{i=1}^{k} v_i + \frac{v_{k+1}}{w_{k+1}} \cdot w_{k+1} = \sum_{i=1}^{k+1} v_i$$

$$C_G = \max\left\{v_{k+1}, \sum_{i=1}^{k} v_i\right\} \ge \frac{1}{2}\sum_{i=1}^{k+1} v_i > \frac{opt}{2}$$

$$\because \max\{a,b\} \ge \frac{1}{2}(a+b)$$

31

多机调度

List调度算法(Graham)：将n个作业依次以online的方式分配到m台机器中的某一台上，规则是将当前作业分配到当时负载最小的机器上，而机器负载是分配给它的所有作业的总的运行时间。

Th.1.8 设A表示List调度算法，则对于所有输入实例I

$$\frac{A(I)}{OPT(I)} \le 2 - \frac{1}{m}$$

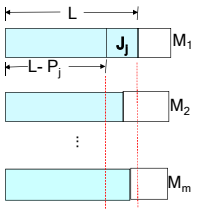
界是紧致的，因为存在一个实例I*，使得上式相等：

A(I*)/OPT(I*)=2-1/m

Pf：先证近似比上界。不失一般性，假定所有作业分配完毕后，机器M_1的负载最大。设L表示M_1上所有作业总运行时间，

32

Pf(续)：设J_j是M_1上最后一个分配到的作业，则其它机器上的负载均大于等于$L-P_j$。这是因为当J_j被分配到M_1时，M_1是负载最小的机器，其负载为$L-P_j$。于是可得：



$$\sum_{i=1}^{n} P_i \ge m(L - P_j) + P_j \quad (1)$$

但I的最优解的值显然满足： $$OPT(I) \ge \frac{\sum_{i=1}^{n} P_i}{m} \quad (2)$$

由于A(I)=L，故有：$OPT(I) \ge (L-P_j)+P_j/m = A(I)-(1-1/m)P_j$

∵必须有某台机器执行作业J_j，∴$OPT(I) \ge P_j$

于是：$A(I)/OPT(I) \le 2-1/m$

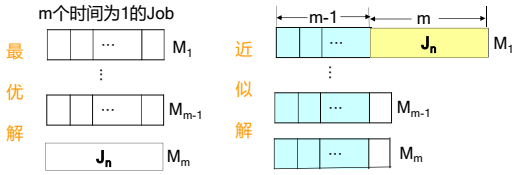
33

Pf(续)：现证近似比的紧确界。考虑输入实例I^*，设

$n = m(m-1)+1$

设前n-1个作业每个均有运行时间1，而最后1个作业J_n有$P_n=m$。易于证明：$OPT(I^*)=m$，而$A(I^*)=2m-1$

故有：$A(I^*) / OPT(I^*) = 2 - 1/m$



34

Def1.6 对于优化问题Π，近似算法A的绝对性能比R_A是

$R_A = \inf\{r \mid R_A(I) \le r, \forall I \in D\}$

即：R_A是性能比上界集合中的下确界(最大下界)

例如：对于List调度算法A,我们有：$R_A=2-1/m$

更好的调度是LPT(Longest Processing Time)：将作业按其运行时间递减序排序，然后用List策略调度,其结果：

Th.1.9：LPT算法的性能（近似）比： $$R_{LPT} = \frac{4}{3} - \frac{1}{3m}$$

Pf：当m=1时，

∵ A(I)=OPT(I) ∴A(I)/OPT(I)=4/3-1/3

设对某个m＞1，该定理不成立。

35

Pf(续)：则可设违反该定理具有最少作业数的实例I是$J_1,J_2,\ldots,J_n$，不妨设$P_1 \ge P_2 \ge \ldots \ge P_n$

LPT的调度次序是$J_1,J_2,\ldots,J_n$,其完成时间是A(I)。设其中最迟完成的作业是J_k，则k=n。否则，若k＜n，算法A运行作业$J_1,J_2,\ldots,J_k$(记为实例I'⊂I)的完成时间仍是A(I)，即A(I)=A(I')，而对最优解显然OPT(I') ≤OPT(I)，故有：

$$\frac{A(I')}{OPT(I')} \ge \frac{A(I)}{OPT(I)} > \frac{4}{3} - \frac{1}{3m}$$

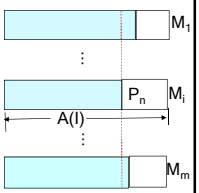
这与I是违反定理的最少作业数的实例矛盾,∵ | I' | < | I |

36

现证明：对于上述实例I的最优调度，在任何机器上分配的作业数不超过2，因此n≤2m

∵ J_n是LPT调度A中最迟完成的作业，

∴ 在A中它开始于时刻A(I)-P_n，且此刻其它机器均无空余时间，即：



$$A(I)-P_n \le \frac{1}{m}\sum_{i=1}^{n-1} P_i \quad \Rightarrow \quad A(I) \le \frac{1}{m}\sum_{i=1}^{n} P_i + \frac{m-1}{m}P_n$$

另一方面，因为 $OPT(I) \ge \frac{1}{m}\sum_{i=1}^{n} P_i$

$$\frac{4}{3}-\frac{1}{3m} < \frac{A(I)}{OPT(I)} \le \frac{OPT(I)+\frac{m-1}{m}P_n}{OPT(I)} = 1+\frac{m-1}{m}\frac{P_n}{OPT(I)}$$

37

$$\frac{4}{3}-\frac{1}{3m} < \frac{A(I)}{OPT(I)} \le \frac{OPT(I)+\frac{m-1}{m}P_n}{OPT(I)} = 1+\frac{m-1}{m}\frac{P_n}{OPT(I)}$$

$$4m-1 < 3m+3(m-1)\frac{P_n}{OPT(I)} \quad \Rightarrow \quad OPT(I) < 3P_n$$

∵ P_n是实例I中时间最短的作业

∴ 实例I的最优调度中任何机器上的作业数≤2

当最优调度在任何机器上至多包含2个作业时，LPT也是最优的。证明如下：

不妨设n=2m，若n<2m，则令$J_{n+1},\ldots,J_{2m}$的时间均为0，将其加入I，不妨设$P_1 \ge P_2 \ge \ldots \ge P_{2m}$

38

设最优调度使得每台机器恰有2个作业：J_i和J_j，则必有i≤m,j＞m。否则若某最优调度O有i,j≤m，则定有某台机器上有J_s和J_t，使得s, t＞m.

∵ $P_i,P_j \ge P_s,P_t$，交换P_j和P_t，则

$P_i+P_t \le P_i+P_j$　　$P_s+P_j \le P_i+P_j$

交换后的调度O'的最迟完成时间只可能减少，故O'也是最优调度。对于i,j>m可类似证明。

∴ 必有最优调度使$J_1,\ldots,J_m$分别分配到$M_1,\ldots,M_m$上，当将$J_{m+1},\ldots,J_{2m}$分配到M台机器上时，LPT是将长时间的作业分配到轻负载上，必与该最优调度结果相同。

$$\frac{A(I)}{OPT(I)} = 1 \le \frac{4}{3}-\frac{1}{3m} \quad (m \ge 2)$$

Ex. 完善此证明

39