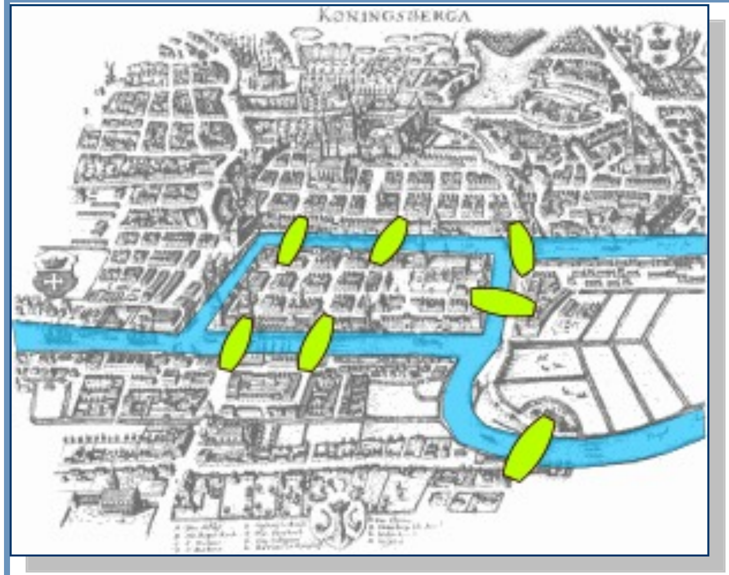


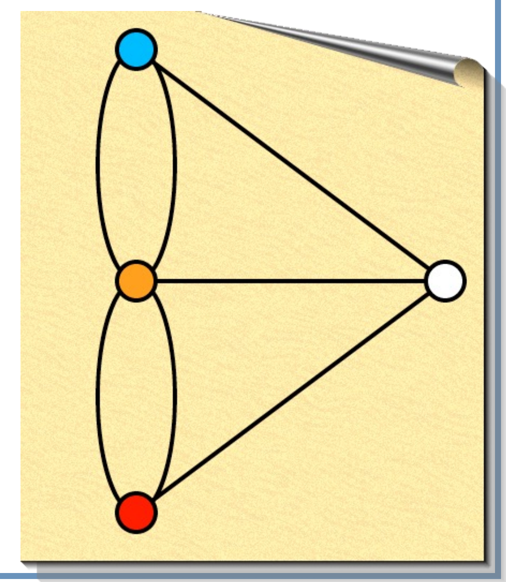

# 通信网络算法思维

## Part 2: 在线问题的求解

——经典案例

**王晟**

博士　教授　博导



# 在线算法示例

1. Scheduling
2. Bin Packing
3. Steiner Tree
4. Bipartite Matching

# 问题定义

## Online Scheduling:

- 给定: $m$个相同的机器;
- 但是任务是一个一个到达的; 同时输入该任务$j$的处理时间: $p_j$
  - 在线问题的特点: 某些输入是"逐渐呈现"的.
- 要求: 给每个任务指定一个执行它的机器.
  - 在线问题的特点: 这种决策不能延迟, 也不可撤销.
- 目标: **Makespan**(最大负载)最小化.

## Notes:

- 在线问题中, 一般假定过去到达的输入/任务不会离去. 因此它们的影响会持续到最后.如果可以离去, 称为"动态"问题.
- 在线调度在云计算环境中具有重要的应用.

# 一切都很直观

- **Graham算法(1966) [也称为List Scheduling, LS算法]**
  - 当新任务到达时, 将它指派到具有最小负载的机器上去.
- **在线算法的评价**
  - 跟近似算法一样, 需要一个对比的依据(**benchmark**).
  - 上帝视角: 如果知道所有未来的输入, 能得到多好的目标值.
    - ➡ 相当于把“离线”优化的最佳解/值作为对比依据.
    - ➡ 又跟近似算法一样.
  - 但是, 在线问题的离线版本有时是**NP-H**的.其最佳解难以获得.
    - ➡ 还是跟近似算法一样, 给OPT定界.
    - ➡ 利用LP对偶理论经常是最好的选择.
      但对在线调度问题, 根据物理意义来定界同样可行.

# 竞争比

- 两个显而易见的下界:
  - $\boxed{\textbf{Lemma-1:}\, \boldsymbol{OPT \geq \max_j p_j}}$

    任何机器的负载都不可能低于RHS.

  - $\boxed{\textbf{Lemma-2:}\, \boldsymbol{OPT \geq \frac{1}{m} \sum_{j=1}^{n} p_j}}$

    允许任意拆分才能达到RHS.

- **Claim-3:Graham算法是个$(2-\frac{1}{m})$-竞争算法.**

  竞争比(Competitive Ratio): 算法目标值与最佳(上帝视角)目标值之间的比例.

  与近似算法中的“近似比”并无本质不同.

# 证明

**Claim-3: Graham算法是个$(2-\frac{1}{m})$-竞争算法.**

- 令算法结束时机器$i$具有最大负载.
  - 故, makespan为该机器上所有任务的处理时间之和.
- 令任务$j$为指派给机器$i$的最后一个任务.
- 由算法的贪心规则可知, 在任务$j$之前, 机器$i$具有最小负载. 令该负载为$S_i^j$.
  - 当时, 机器$i$的负载小于平均负载: $S_i^j \le \frac{1}{m}\sum_{k=1}^{j-1} p_k$
  - 故, makespan$= S_i^j + p_j \le \frac{1}{m}\sum_{k=1}^{j-1} p_k + p_j = \frac{1}{m}\sum_{k=1}^{j} p_k + \left(1-\frac{1}{m}\right)p_j$

$$\le \frac{1}{m}\sum_{k=1}^{j} p_k + \left(1-\frac{1}{m}\right)p_{max}$$

$$\le OPT + \left(1-\frac{1}{m}\right)OPT$$

$$= \left(2-\frac{1}{m}\right)OPT$$

- 得证.

# 紧例及内涵

- 例#1:
  - 3个任务, 2个机器. $m=2, p_1=p_2=1, p_3=2$.
  - Makespan=3, OPT=2.  2-1/m=3/2
- 例#2:
  - $m(m-1)+1$个任务, $m$个机器. 最后一个任务处理时间为m, 其他为1.
  - Makespan=2m-1, OPT=m.
- 或许你已经看出来了:
  - 在线调度最大的劣势在于“无法预测未来”.
  - 事实上, 可以利用这种特性建立一个“敌对者”论证.

# 竞争比下限

- **Claim-4: 任何在线调度算法的竞争比都不可能低于3/2.**
  - 考虑任一个在线算法A.
  - 先输入m个任务, 处理时间都是1.
    - Case#1: 如果算法A的结果中有空闲的机器,
      ➔该算法的竞争比至少为2.[因为最佳解为1]
    - Case#2: 如果算法A的结果是“每个任务占一个机器”,
      - 再输入一个任务, 处理时间为2.
      - 最佳解为2, 而算法给出的解为3.
  - 无论哪种case, 我们结论都成立. 得证.
- **Notes:**
  - 敌对者充分利用了在线算法无法预测未来的劣势.
  - 离线调度怎样利用这种优势?

# 离线调度: LPT

- 离线调度也是NP-H问题. 只能近似.
  - LPT: Longest Processing Time First.
  - 按照处理时间降序排列所有任务. 然后调用Graham算法.
- Claim-5: LPT算法是个$\left(\frac{4}{3}-\frac{1}{3m}\right)$-近似算法.
  - 讨论LPT算法的目的是进一步揭示在线问题的难度所在.
  - 仅仅排个序就可以改善性能.
  - 现实中碰到在线问题的思路: 先想想能不能克服“在线”约束.
  - 为了证明Claim-5, 需要先证明两个关键的事实.

# 关键事实#1

**Lemma-6: 可以假定最后完成的任务就是最小任务.**

- 假定任务按照处理时间降序编号: $p_1 \geq p_2 \geq \cdots \geq p_n$
- 令任务k是LPT解中最后完成的任务.

➡ **即: 其完成时间就是LPT的makespan.**

- 如果$k < n$, 则所有编号更大的任务$j > k$都开始得更晚, 完成更早.

➡ **这些任务不会对LPT的makespan造成影响.**

➡ **它们的存在只会对最佳解的makespan不利.**

- 因此, 在对比LPT与最佳解时, 去掉它们只会让近似比变差.

➡ **在Claim-5的证明中, 可以假定k=n.**

# 关键事实#2

**Lemma-7: 如果约束每个机器最多安排2个任务，那么LPT得到的就是最佳解.**

- 若$n \le m$, LPT显然是最佳解.
- 若$m < n \le 2m$, 可以假定$n = 2m$. [添加$2m - n$个大小为0的任务]

LPT会怎样安排这2m个任务? → "折半拼接"

- 现在考虑最佳解中的两个机器$i$和$i'$.
- 令这两个机器上的四个任务为$a, b, c, d$, 且:$p_a \ge p_b \ge p_c \ge p_d$
- 四个任务分到两个机器上的方式一共只有三种:
  分组#1: a,b一组, c,d另一组.
  分组#2: a,c一组, b,d另一组.
  分组#3: a,d一组, b,c另一组.

→ 注意: #3就是"折半拼接"

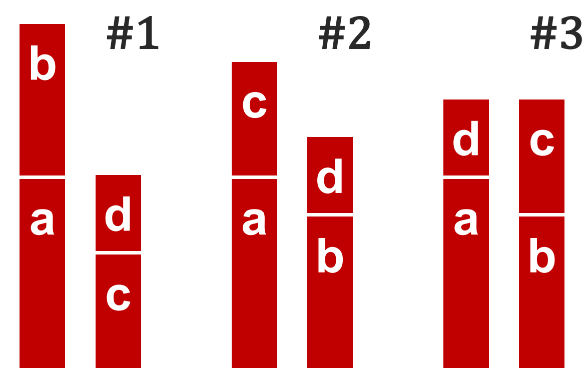
- 令每种分组方式下对应的最大负载为$L_1, L_2, L_3$
- 待证: $L_1 \ge L_2 \ge L_3$

# 分组#3负载最小

**Lemma-7: 如果约束每个机器最多安排2个任务，那么LPT得到的就是最佳解.**

- 待证: $L_1 \geq L_2 \geq L_3$
- 分组方式#1中, a,b那一组肯定负载更大,
  即: $L_1 = p_a + p_b$
- 分组方式#2中, a,c那一组肯定负载更大,
  即: $L_2 = p_a + p_c$

➡ 由于$p_b \geq p_c$, 故$L_1 \geq L_2$




- 考虑分组方式#3中的两种情况:
  Case#1: $L_3 = p_a + p_d$, 此时有: $L_2 \geq L_3$
  Case#2: $L_3 = p_b + p_c$, 此时同样有: $L_2 \geq L_3$

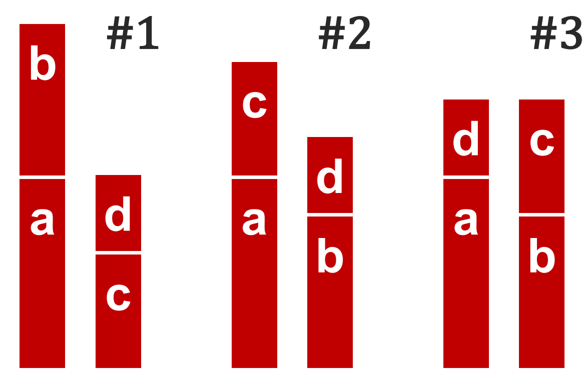
? 证得$L_1 \geq L_2 \geq L_3$,意味着什么?

➡ 我们可以应用Exchange Argument了.

# Exchange Argument

**Lemma-7: 如果约束每个机器最多安排2个任务，那么LPT得到的就是最佳解.**

- 已知: $L_1 \geq L_2 \geq L_3$
- 考虑以下对最佳解的处理过程:检查任意两个机器, 若出现分组**#1**或**#2**,则替换为**#3**.
  - ➡ 替换后的解makespan不可能更大.
  - ➡ 若变小了, 那它不是最佳解.
  - ➡ 若不变, 则说明“折半拼接”就是最佳.
- 得证.



# 证明Claim-5

**Claim-5: LPT算法是个$(\frac{4}{3}-\frac{1}{3m})$-近似算法.**

- 考虑第$n$个任务的大小.
  - 由Lemma-6, 它既是最后完成的任务, 也是最小的任务.
- Case#1: $p_n > \frac{1}{3}OPT$
  - 此时所有任务都超过$\frac{1}{3}OPT$, 故最佳解里每个机器最多2个任务.
  - 由Lemma-7, LPT达到最佳.
- Case#2: $p_n \leq \frac{1}{3}OPT$
  - 此时可以应用Claim-3中的证明策略.
  - 令LPT安排下, 最大负载出现在机器$i$上.
  - 令$S_i^n$表示安排任务$n$之前机器$i$上的负载.

# 证明Claim-5(续)

**Claim-5: LPT算法是个$\left(\frac{4}{3}-\frac{1}{3m}\right)$-近似算法.**

- **Case#2:** $p_n \le \frac{1}{3} OPT$

➡ 令LPT安排下, 最大负载出现在机器$i$上.

➡ 令$S_i^n$表示安排任务$n$之前机器$i$上的负载.

➡ makespan$= S_i^n + p_n$

$$\le \frac{1}{m}\sum_{k=1}^{n-1} p_k + p_n$$

$$= \frac{1}{m}\sum_{k=1}^{n} p_k + \left(1-\frac{1}{m}\right) p_n$$

$$\le \frac{1}{m}\sum_{k=1}^{n} p_k + \left(1-\frac{1}{m}\right)\cdot\frac{1}{3} OPT$$

$$\le OPT + \left(\frac{1}{3}-\frac{1}{3m}\right) OPT$$

$$= \left(\frac{4}{3}-\frac{1}{3m}\right) OPT$$

➡ 最大负载的假设

➡ 贪心准则的效果

➡ 代数

➡ Case#2

➡ Lemma-2

- 得证.

# 在线算法示例

1 Scheduling

2 Bin Packing

3 Steiner Tree

4 Bipartite Matching

# 问题描述

## Online Bin Packing:

- 给定: 逐个到达的物品序列, $L = \{s_1, s_2, \dots\}$;
  其中$s_i \in (0, 1]$为第$i$个物品的大小.

  ➡ 在线问题: 输入"逐渐呈现".

- 要求: 把物品装入容积为1的箱子.

  ➡ 在线问题: 每个物品来了,都需要立刻装箱, 不可延迟, 也不可撤销.

- 目标: 最小化箱子的数目.

## Notes:

- 任何最小化资源消耗的资源配置问题都可以看作是在装箱.
- 即使没有在线约束, 该问题也是NP-H的.

# 近似/竞争比的下限

**Claim-8: $\forall \epsilon > 0$, 没有算法能保证近似/竞争比优于$\frac{3}{2} - \epsilon$, 除非P=NP.**

- Partition判定问题(NP-C): 给定n个整数$a_1, a_2, \dots, a_n$, 问能否分为2组, 每组的和都是$\frac{1}{2}\sum_{i=1}^{n} a_i$.
  - 简单的变换, 即可将Partition问题实例变换为装箱实例.
  - 如果装箱最佳解为2, 则Partition的答案为YES; 否则答案为NO.
- 现在调用$\frac{3}{2} - \epsilon$近似的算法来求解该装箱实例.
  - 若OPT=4, 该算法最多用几个箱子? → 5个.
  - 若OPT=2, 该算法最多用几个箱子? → 2个.
  - 若近似算法的解为2, 则Partition问题的答案为YES.

# 渐近近似/竞争比

## Claim-8的证明中, 用到的是非常特殊的装箱实例.

- 只有很少的箱子(2~3个), 但物品数目可以任意大.
- 更“典型”的输入中: 箱子数目会随着物品数目的增大而增大.

更典型的实例中, 近似/竞争比会是什么样子?

## Asymptotic Performance Ratio

- 令$ALG(I)$表示算法$ALG$针对实例$I$得到的目标值; $OPT(I)$表示最佳值.
  - 特定实例$I$的目标值比例为: $\frac{ALG(I)}{OPT(I)}$
  - 算法的竞争/近似比为: $R_{ALG} = sup_I \left\{\frac{ALG(I)}{OPT(I)}\right\}$
  - 渐近竞争/近似比为: $R_{ALG}^{\infty} = \lim_{n\to\infty} sup_I \left\{\frac{ALG(I)}{OPT(I)} \middle| OPT(I) = n\right\}$

# 解读

$$R_{ALG}^{\infty} = \lim_{n\to\infty} sup_I \left\{ \frac{ALG(I)}{OPT(I)} \middle| OPT(I) = n \right\}$$

## 解读之一:

- 如果$\exists N$, 对$OPT(I) \geq N$的实例, 算法ALG能够保证$\frac{ALG(I)}{OPT(I)} \leq \alpha$, 则称该算法为$\alpha$-渐近竞争/近似.

➡ 换句话说, 只看OPT足够大以后的竞争/近似比.

## 解读之二:

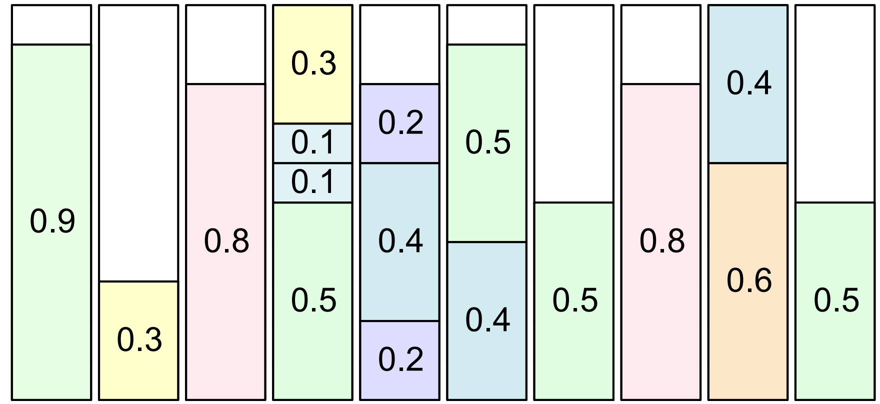
- 也可以理解为, 使得下式成立的最小的$\alpha$:

$$\forall I, ALG(I) \leq \alpha \cdot OPT(I) + c$$

其中$c$为与输入无关的非负常数.

➡ 换句话说, 依旧是在评估算法目标值与最佳值之间的比例, 但是却给了算法一些优惠(比如可以多用几个箱子).

# Next-Fit




## Next-Fit算法:

- 始终只维护一个打开的箱子.
- 到达的物品可以装入, 则装入; 否则, 关闭该箱, 开一个新的.
- 关闭了的箱子永远不再开启.

$$L = \{0.9, 0.3, 0.8, 0.5, 0.1, 0.1, 0.3, 0.2, 0.4, 0.2, 0.4, 0.5, 0.5, 0.8, 0.6, 0.4, 0.5\}$$

## Notes:

- 你或许会觉得这个算法很傻, 但它有一个明显的优点: 有限开箱.
- 在某些应用中, 有限开箱是必须的约束.
  - ➡ 这种算法同时保证了有“持续的”输出.
  - ➡ 某种意义上说, 这更像“在线”的语义.
- 况且, 下面的分析将表明: 它或许没有你想象的那么傻.

# 分析: Next-Fit

- **Lemma-9: $\forall I, OPT(I) \geq \sum_{i=1}^{n} s_i$**

- 最少的箱子数目也会超过物品大小之和.

- **Claim-10: Next-Fit的竞争比最多为2.**

  - 相继的两个箱子所装的物品体积总和一定超过1.

    ➡ 假定NF一共用了$m$个箱子, 则有$m \leq 2 \cdot \sum_{i=1}^{n} s_i$

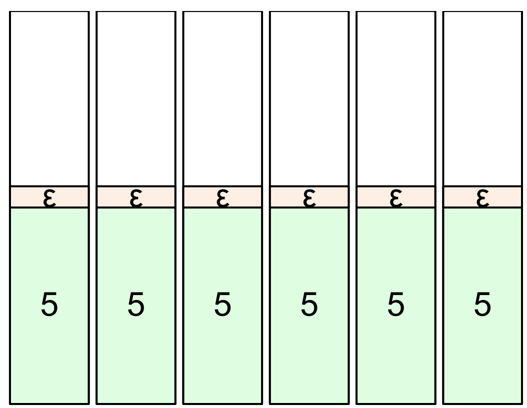
    ➡ 再由Lemma-9可知, $m \leq 2 \cdot OPT$

- 考虑如下紧例:

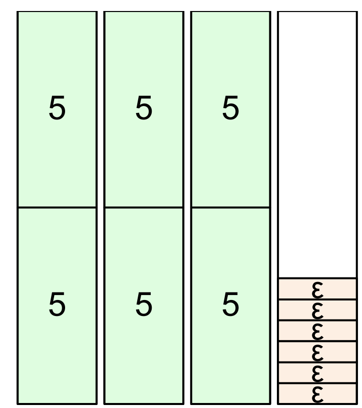
  - $L = \{0.5, \epsilon, 0.5, \epsilon, \dots\}$

    ➡ $OPT \approx n/2$

    ➡ $ALG = n$




NextFit

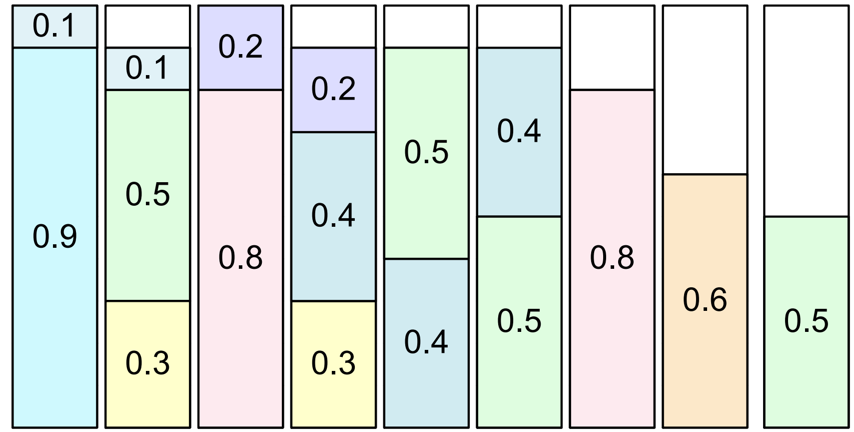



OPT

# FF & BF

## First-Fit算法:

- 除非满了，否则箱子一直开着;
- 将物品装入“第一个”有足够空间的箱子.

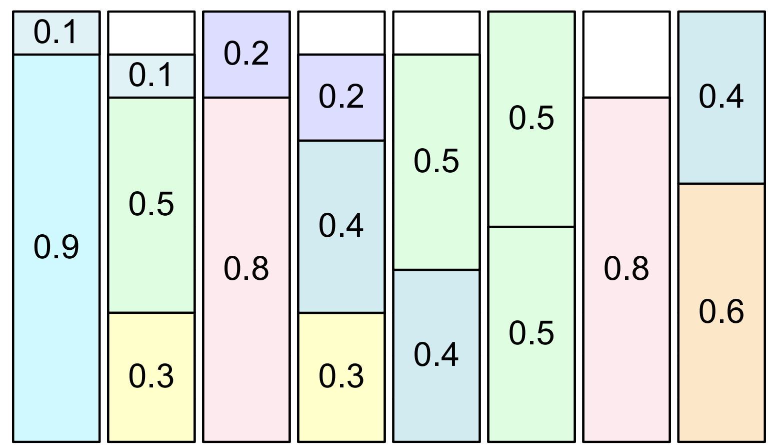



$$L = \{0.9, 0.3, 0.8, 0.5, 0.1, 0.1, 0.3, 0.2, 0.4, 0.2, 0.4, 0.5, 0.5, 0.8, 0.6, 0.4, 0.5\}$$

## Best-Fit算法:

- 除非满了，否则箱子一直开着;
- 将物品装入“最好的”有足够空间的箱子.

➡ 最好是指剩余空间最小.




## 这两个都是无限开箱算法.

# 简单分析: FF

**Claim-11: First-Fit的渐近竞争比最多为2.**

- 如果FF一共用了$m$个箱子, 那么其中$m-1$个都是超过半满的.
  - 如果有2个以上没到半满, 意味着什么?
- 因此有: $\frac{m-1}{2} < \sum_{i=1}^{n} s_i \le OPT$
  - 故: $m \le 2 \cdot OPT + 1$
  - 去掉常数1, 可得: 渐近近似比最多为2.

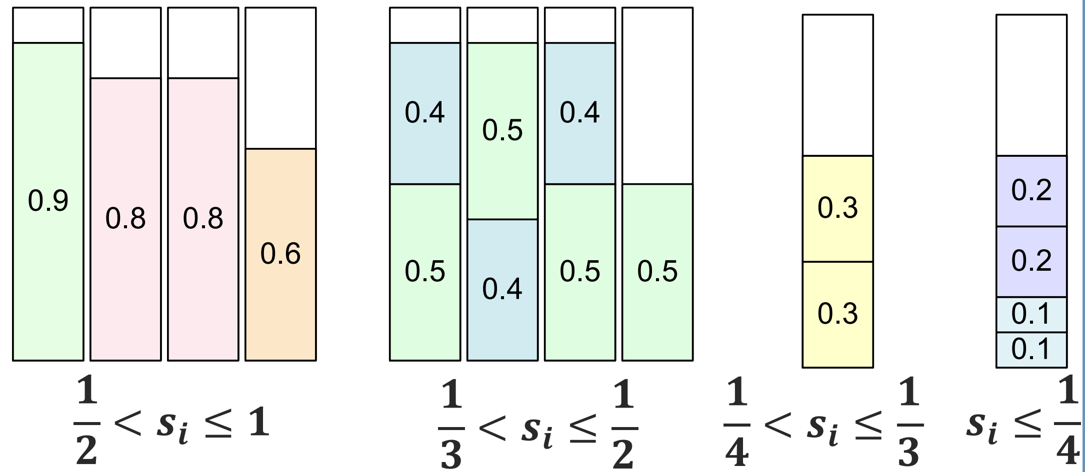
**Notes:**

- 很明显, 上述论证同样适用于BF. 因此结论也适用.
- 但没有紧例.
  - 为改进分析, 需要引入权重方法, 而这需要理解“调和”算法.

# 调和算法

## Harmonic算法

- 物品按照大小分类.
- 总共K类:
  第1类: $(\frac{1}{2}, 1]$;第2类: $(\frac{1}{3}, \frac{1}{2}]$
  第$i$类: $\left(\frac{1}{i+1}, \frac{1}{i}\right], \forall i < K$
  第$K$类: $(0, \frac{1}{K}]$




“调和”的由来

$$L = \{0.9, 0.3, 0.8, 0.5, 0.1, 0.1, 0.3, 0.2, 0.4, 0.2, 0.4, 0.5, 0.5, 0.8, 0.6, 0.4, 0.5\}$$

- 每类单独装箱: Next-Fit
- 开启的箱子始终不超过K.

有限开箱

- 注意: $\forall i < K$, 刚好$i$个第$i$类物品“占用”一个箱子.

# 权重方法

- **Weighting Method是装箱问题中广泛使用的证明技术.**
  - 为了证明渐近竞争比最多为$\alpha$, 可以执行以下步骤:
  - **Step#1:** 定义权重函数$w(s)$, 即为每种物品大小$s$定义一个权重.
    通常要求 $w(s) \geq s$.
  - **Step#2:** 证明在算法的最终解中, 除了(最多)常数c个箱子之外,
    其他每个箱子中装的物品的权重和都至少为1.
  - **Step#3:** 证明在最佳解中, 每个箱子中装的物品的权重和最多为$\alpha$.
- **解读:**
  - 令$W$为所有物品的权重总和. 令$m$表示算法解中用到的箱子总数.

➡ Step#2证得: $m \leq W + c$

➡ Step#3证得: $OPT \geq W/\alpha$

➡ 故: $m \leq \alpha \cdot OPT + c$ ➡ 即渐近竞争比为$\alpha$.

# 权重分析: 调和算法

**Claim-12: 调和算法的渐近竞争比最多为1.691.**

- **Step#1:** 定义权重函数
  - $\forall i < K$, 区间$(\frac{1}{i+1}, \frac{1}{i}]$内的物品(第$i$类物品)的权重为$\frac{1}{i}$.
  - 区间$(0, \frac{1}{K}]$内的物品(第$K$类物品)的权重为$\frac{K}{K-1}s$.

➡ 至少对调和算法, “权重”的语义是清楚的:
物品$i$的权重是它“实际占用”的箱子数目.

➡ 因此权重总和$W$就是调和算法所使用的箱子总数.

➡ 另一方面, 最佳解也必须“装下”这么多权重.

➡ 所以, 权重既反映了算法解的性质, 又与最佳解联系了起来.

# Step#2

**Claim-12: 调和算法的渐近竞争比最多为1.691.**

- Step#2待证: 除了最多c个箱子之外, 其他每个箱子都至少装入权重1.
  - $\forall i < K$, 除了这一类的最后一个之外, 每个箱子都刚好装入权重1.
  - 现在考虑第$K$类.
    - 考虑某个第$K$类箱子B, 假定它不是这类的最后一个.
    - 箱子B中剩余的空间不可能超过1/$K$.

➡ 因此B里的物品总量超过$(K-1)/K$

➡ 再由第K类权重的定义可知:

$$\sum_{s_i \in B} w(s_i) = \sum_{s_i \in B} \frac{K}{K-1} s_i = \frac{K}{K-1} \sum_{s_i \in B} s_i \geq \frac{K}{K-1} \cdot \frac{K-1}{K} = 1$$

➡ 除了每类的最后一个之外(总和为$K$, 即$c$=$K$), 其他箱子都能保证最终装入的物品的权重和至少为1.

# Step#3

**Claim-12: 调和算法的渐近竞争比最多为1.691.**

- **Step#3待证: 最佳解中每个箱子最多装下权重和为$\alpha$的物品.**
- 针对所有可能的实例, 最多能在一个箱子里装入多大的权重, 就是$\alpha$.
  - 这是个Knapsack问题, 而且是连续背包.
  - 只需按照“权重密度”贪心求解最大权重即可.
- $\forall i < K$, 第$i$类物品权重为$\frac{1}{i}$, 大小至少为$\frac{1}{i+1}$, 故权重密度最大为$\frac{i+1}{i}$.
- 第$K$类物品权重为$\frac{K}{K-1}s$, 大小为$s$, 故权重密度为$\frac{K}{K-1}$.
  - 按照类别编号的升序, 就是权重密度的降序.

# Step#3(续)

**Claim-12: 调和算法的渐近竞争比最多为1.691.**

- 贪心选择的第一个物品, 应该是第1类物品, (权重,大小)=$\left(1, \frac{1}{2}+\epsilon\right)$.
- 第二次选择时, 第1类物品已经装不下了. 只能选第2类物品, $\left(\frac{1}{2}, \frac{1}{3}+\epsilon\right)$.
- 第三次选择时, 第3、4、5类物品都已经装不下了.
  只能选第6类物品, $\left(\frac{1}{6}, \frac{1}{7}+\epsilon\right)$.
- 以此类推.

只要$K$足够大, 装入的权重为:

$$\mathbf{1+\frac{1}{2}+\frac{1}{2\times 3}+\frac{1}{6\times 7}+\frac{1}{42\times 43}+\cdots \approx 1.691}$$

- Claim-12得证.

# 紧例

## 证明中出现的那种装包序列就是构造紧例的依据.

- 考虑这样的序列: 先来$m$个$\frac{1}{43}+\epsilon$的物品, 然后是$m$个$\frac{1}{7}+\epsilon$,
  然后是$m$个$\frac{1}{3}+\epsilon$, 最后是$m$个$\frac{1}{2}+\epsilon$.

- 最佳解: 刚好每类挑一个装在同一个箱子里, 共需$m$个箱子.

- 调和算法: 每类单独开箱, 故箱子总数为:

$$m\cdot\left(\frac{1}{42}+\frac{1}{6}+\frac{1}{2}+1\right)$$

- 我们可以在这个序列前面继续增加“前缀”.

因此, 渐近意义上(只要序列够长), 调和算法会使用$\alpha\cdot m$个箱子.

# FF: 权重分析

**Claim-13: FF算法的渐近竞争比最多为1.7**

- 基本思路与调和算法类似.
- **Step#1:** 想明白在FF中各种大小的物品“浪费”了多少箱子空间. 从而构造对应的权重函数.
- **Step#2:** 权重函数的构造很容易导致这一步的结论.
- **Step#3:** 非常麻烦, 需要枚举各种case.

目前最好的界(17/10)来自于2013年. 他们构造的权重函数为:

$$w(s)=\begin{cases}\frac{6}{5}s, & 0\le s\le\frac{1}{6}\\ \frac{9}{5}s-\frac{1}{10}, & \frac{1}{6}<s\le\frac{1}{3}\\ \frac{6}{5}s+\frac{1}{10}, & \frac{1}{3}<s\le\frac{1}{2}\\ \frac{6}{5}s+\frac{4}{10}, & \frac{1}{2}<s\le 1\end{cases}$$

# Notes

- FF也有紧例, 但很不直观. 这里略去.
- 构造非常类似的权重函数, 可以证明Best-Fit的渐近竞争比也是1.7
- 离线算法FFD和BFD(先降序排列物品)的渐近近似比为11/9
- 存在“渐近PTAS”, 但只用于理论研究.
- 现实中最常用的还是BF/FF这类简单策略, 原因是这类方法的平均性能非常好(=1).

# 应用：调度问题

## 考虑Scheduling on Identical Parallel Machines

- $m$个机器, $n$个任务, 执行时间分别为$p_1, p_2, \dots, p_n$
- 存在makespan=t的调度, 等价于n个物品能用m个容积为t的箱子装下.

➡ 二分搜索t, 每给定一个t值, 构造相应的BinPacking实例.

➡ 调用BinPacking的渐近PTAS来求解该实例.

➡ 最终得到的是调度问题的PTAS.

## BinPacking的PTAS

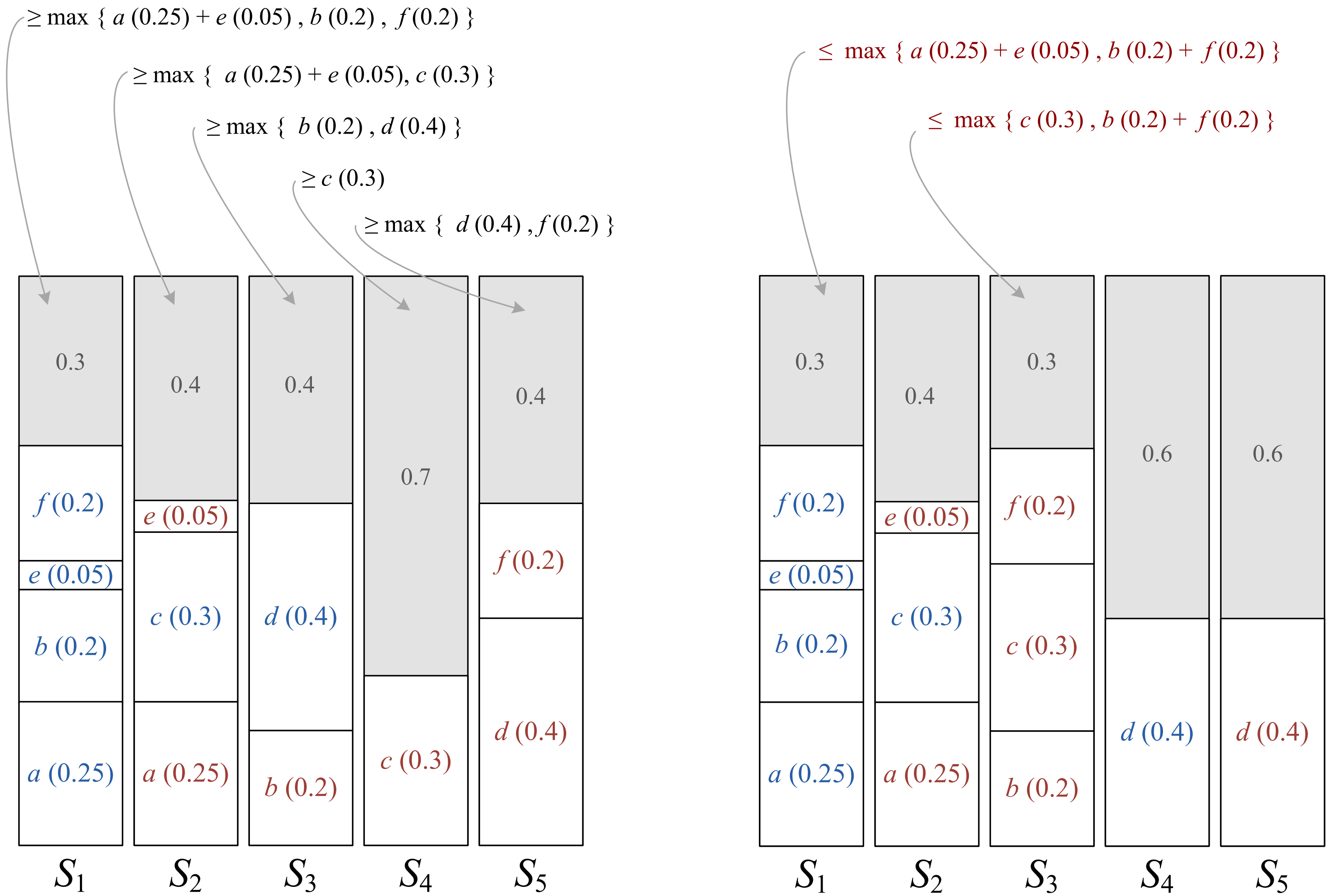
- 物品按大小分类, 分类数目依赖于参数$\epsilon$. 同一区间内的物品"rounding"为统一大小.➔误差受控于$\epsilon$.
- 然后用动态规划来枚举各种可能的装箱组合.

# 应用：服务器合并

## Fault-Tolerant Bin Packing [Server Consolidation in Cloud]

- 箱子代表服务器，物品代表租户(Tenant)
  - 租户可以是完整应用(例如Web服务/数据库)，也可以是单个文件(例如Netflix的一部电影).
- 物品大小代表该租户需要占用多少服务能力
  - 通常不是指存储需求，而是提供多少速率的下载，访问等.
- 服务器的失效不能影响租户继续提供服务
  - 每个物品都需复制一份(红、蓝)，放置在不同的箱子内.
- 正常情况下，红、蓝合作响应用户需求.
  - 物品大小为s的话，红蓝各占1/2.
- 异常情况下，失效服务器的负载要转移到其他有拷贝的服务器上.
  - 各个箱子都不能占满，需要预留容量.

# 实例：有效与无效的方案




$$L = \{a = 0.5, b = 0.4, c = 0.6, d = 0.8, e = 0.1, f = 0.4\}$$

# 求解思路

## 镜像算法:

- 红蓝两类物品各自装箱, 分别用BF.
- 每个箱子的大小都设置为0.5. [预留0.5]

## 交织算法:

- 本质上仍是BF, 但允许红蓝物品交织放入相同箱子.
- 但对箱子的预留要分类考虑.

## 水平调和算法:

- 同样需要对箱子的预留方案逐个进行考虑.
- 但本质上是调和算法.

前两个算法的渐近近似比不可能好于2, 最后一个接近1.59.

# 一般下界(1/3)

**Claim-14: 任何在线装箱算法的竞争比最少为4/3**

- 考虑以下序列: $L_1 = m\times\left\{\frac{1}{2}-\epsilon\right\}$
  - 显然, $OPT(L_1) = \frac{m}{2}$
  - 假定ALG算出的箱子数目为$\alpha \cdot m, \alpha \in [\frac{1}{2}, 1]$
  - 因此, 针对这个特定实例, $\frac{ALG(L_1)}{OPT(L_1)} = 2 \cdot \alpha$
  - 结论#1: 竞争比至少为$2 \cdot \alpha$.
- 考虑另一个序列: $L_2 = m\times\left\{\frac{1}{2}-\epsilon\right\} + m\times\left\{\frac{1}{2}+\epsilon\right\}$
  - 显然, $OPT(L_2) = m$

# 一般下界(2/3)

**Claim-14: 任何在线装箱算法的竞争比最少为4/3**

- 考虑另一个序列: $L_2 = m\times\left\{\frac{1}{2}-\epsilon\right\}+m\times\left\{\frac{1}{2}+\epsilon\right\}$
- 根据前面的假设, ALG对前面m个物品开了$\alpha\cdot m$个箱子.
  - 其中有多少个箱子只装了1个物品?
    - 全装1个物品的话, 需要$m$个箱子, 实际用了$\alpha\cdot m$个.
    - $\alpha m-(m-\alpha m)=2\alpha m-m$个.
- 第二批物品来的时候, ALG最多只能利用个$2\alpha m-m$旧箱子.
  - 由于第二批物品的大小, 故新开箱$2m-2\alpha m$个.
- 故: $ALG(L_2)=2m-\alpha m$
- 结论#2: 竞争比至少为$2-\alpha$

# 一般下界(3/3)

- **Claim-14: 任何在线装箱算法的竞争比最少为4/3**
  - 结论#1: 竞争比至少为$2\cdot\alpha$ → 序列$L_1$
  - 结论#2: 竞争比至少为$2-\alpha$ → 序列$L_2$
  - 竞争比至少为: $max\{2\alpha, 2-\alpha\}$
  - 因此, 至少为4/3.
- **Note:**
  - 这个证明过程不过是讨论一个具体的例子.
  - 沿着这种构造实例的思路, 可以证明更紧的界.[State of art, 1992]
- **Claim-15: 任何在线装箱算法的竞争比最少为1.54037**

# 敌对者

**Note#1: 上述证明没有使用“敌对者”论证.**

- 敌对者根据算法的决策结果来决定下一步输入, 从而导出性能界.
  - 关于Scheduling的一般下界证明中, 用的就是敌对者论证.
- 一般的在线算法中, 若困难来自于敌对者, 引入随机算法往往奏效.
  - 但装箱问题中, 随机性没有帮助.

- 这也可以从另一个方面说明为什么BF/FF这类简单算法在实际应用中效果良好: 敌对者无法利用其结果.

- 其他在线问题就没这么幸运了.

# 两种约束

- **Note#2: 上述证明主要用的是“递进”约束.**
  - 在线算法有两个约束, 一个是在线约束:输入逐渐呈现; 另一个是递进约束: 不能修改以前的决定.
  - 一般这两个约束是同时起作用的.
  - 但在一般下界的证明中, 可以只利用其中一个.
    - ➡ 敌对者论证实际上主要利用的是在线约束.
    - ➡ 上面的证明中主要利用的是递进约束.
- ➡ 这意味着, 我们在实际中可以分开考虑这两个约束. 只要你的问题中允许放松任何一个约束, 都有希望明显提高算法性能.

# 在线算法示例

1. Scheduling
2. Bin Packing
3. Steiner Tree
4. Bipartite Matching

# Online Steiner Tree

## 问题描述:

- 已知: 无向图$G(V,E)$, 边权: $\forall e \in E, c_e \geq 0$
- 在线到达: 终端(terminals) $t_1, t_2, \ldots, t_k \in V$
- 在线决定: 每个终端到达时, 都要决定如何“连起来”.
  [保证目前为止到达的所有终端都可相互连通]
  - 选择图中的边, 保证子图连通.
  - 过去的选择不可更改.
- 目标: 覆盖所有的终端所需的代价最小.
  - 因此, 该子图必是树.

## Notes:

- 可以假定图是完全图: 增加权重足够大的边.
- 典型的动态网络构造问题. 应用广泛.

# Metric Case

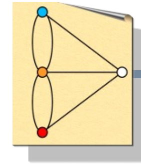
- 我们研究的是Metric Steiner Tree问题.
  - 即假定所有边权满足三角不等式.
    - 基本的内涵: 单跳路径最短.
  - 一般也同时假定: 给定的是完全图.
- 神奇的特例
  - 对**Steiner**树问题来说, 任何一般**case**都可归约为**metric case.**
    - 这与我们前面研究过的Metric TSP, Metric Steiner Forest不一样.
- Claim-16: 任何Steiner树问题都可以归约为Metric Case.

# 归约(1/2)

**Claim-16: 任何Steiner树问题都可以归约为Metric Case.**

- 任给无向连通图$G$, 都可以构造一个无向完全图$G'$, 满足三角不等式.
  - Step#1: 求所有节点对之间的最短路.
  - Step#2: 构造完全图$G'$, 边权为最短路权重.
  - 该图一般称为原图的Metric Closure.
  - 该图边权显然满足三角不等式.
- $G'$中求得的最小Steiner树必然小于等于$G$中求得的最小Steiner树.
  - 因为$G'$中每条边都小于等于$G$中的对应边.
- 待证: $G'$中求得的最小Steiner树$T'$可以转换成$G$中的Steiner树, 且权重不会变大.

# 归约(2/2)



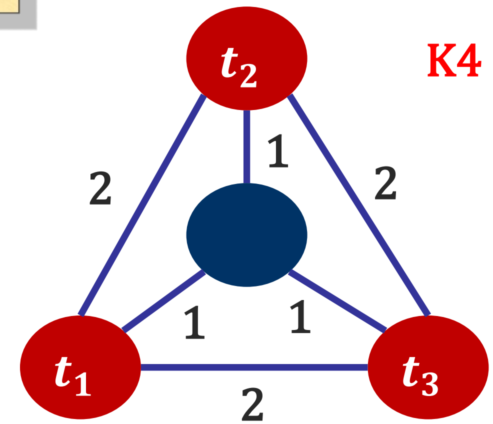
**Claim-16: 任何Steiner树问题都可以归约为Metric Case.**

- 待证: $G'$中求得的最小Steiner树$T'$可以转换成$G$中的Steiner树$T$, 且权重不会变大.
  - $G'$中的每条边都是$G$中的一条路, 所有这些路径包含的边构成的$G$子图$G''$.
  - $G''$中显然包含了所有的终端.
  - $G''$中可能含有圈. 去掉圈上最大权重的边.
  - 这样得到的树$T$一定权重更小.
- 因此, 任给实例$G$, 得到Metric实例$G'$, 求得最小Steiner树$T'$后, 再转换成$G$中的Steiner树$T$, 得到的一定是原图的最佳Steiner树.
- 得证.

# 在线贪心算法

## 算法: [Online Metric Steiner Tree]

- $T = \Phi$
- 当终端$t_i, i > 1$到达时:
  比较$t_i$与$t_j$ $(j < i)$之间的边权
  将最小边加入$T$.
- 返回$T$.

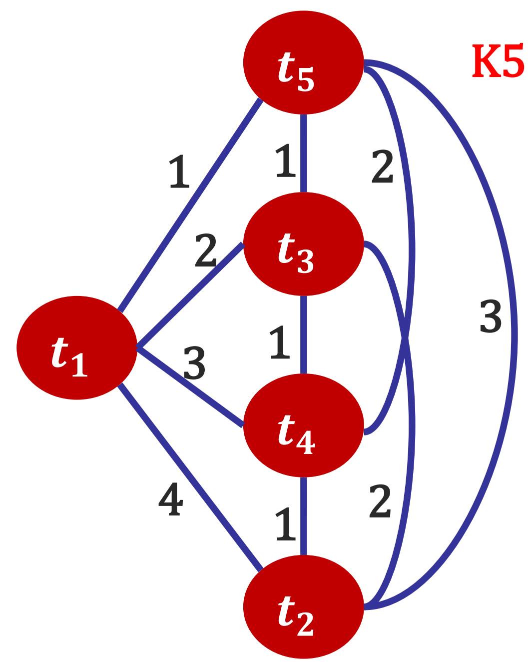



## 考虑坏例:

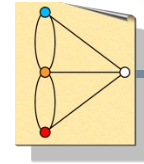
- 容易验证, 这两个图都是metric case.
- K4图中, OPT=3, 贪心解=4
- K5图中, OPT=4, 贪心解=8

➡ 遵照该模式可以构造一系列例子, 足以说明: 竞争比不可能优于log$k$

- 有人证明: 任何在线算法的竞争比都差于log$k$



# 在线贪心是最佳的



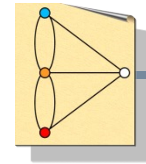
- **Claim-17: 在线贪心算法的竞争比为O(log$k$)**
- **Lemma-18: 贪心解得到的树上, 第$i$大的边权$\leq 2 \cdot \frac{OPT}{i}$**
  - 总共$k$个终端, 故$i = 1, 2, \ldots, k-1$
  - 你可以想象, 在贪心算法运行结束后, 对树上边权降序排列.
  - ➡ Lemma-18对所有这些边权各自设定了一个上限.
  - Lemma-18 ➔ Claim-17
    - ➡ 贪心解的权重和$\leq 2 \cdot OPT \cdot \sum_{i=1}^{k-1} 1/i$
    - ➡ 积分近似调和级数的和, 有: $\sum_{i=1}^{k-1} 1/i \leq \ln k$
    - ➡ 故有: 贪心解上界为$2\ln k \cdot OPT$.
    - ➡ Claim-17得证.

# 关键引理

**Lemma-18: 贪心解得到的树上, 第$i$大的边权$\leq 2 \cdot \frac{OPT}{i}$**

- 终端的连接代价: 贪心算法中, 每个终端到达后, 都使用了一条边连接到当前子图上. 该边权重就是该终端的连接代价.
- 令$s_1, s_2, \ldots, s_i$表示连接代价最大的$i$个终端. $[1 \leq i \leq k]$
- 我们将证明, 这$i$个终端中, 至少有一个的连接代价上界为$2 \cdot OPT/i$
  - 注意: 这$i$个连接代价就是最大的$i$个边权.
  - 第$i$大的边权是这些连接代价中最小的一个.
  - 所以, 只要有一个连接代价小于等于那个上界, 即证得Lemma-18.
- 待证: 终端$s_1, s_2, \ldots, s_i$中, 至少有一个的连接代价上界为$2 \cdot OPT/i$

# Doubling&Shortcut



**Lemma-18: 贪心解得到的树上, 第$i$大的边权$\leq 2 \cdot \frac{OPT}{i}$**

- 待证: 终端$s_1, s_2, \dots, s_i$中, 至少有一个的连接代价上界为$2 \cdot OPT/i$
- 考虑最佳解$T^*$.
- 对$T^*$进行Tree-Doubling操作, 得到欧拉图H.
  - ➡ 显然: $\sum_{e \in H} c_e = 2 \cdot OPT$
- 求H中的欧拉圈, 得到C.
  - ➡ 同样很显然: $\sum_{e \in C} c_e = 2 \cdot OPT$
- 注意:终端$s_1, s_2, \dots, s_i$在C中至少出现一次.
- 对C进行Shortcut操作, 得到只包含$s_1, s_2, \dots, s_i$的简单圈, $C_i$.
  - ➡ 由于三角不等式成立, 故: $\sum_{e \in C_i} c_e \leq 2 \cdot OPT$

# 得证

**Lemma-18: 贪心解得到的树上, 第$i$大的边权$\leq 2 \cdot \frac{OPT}{i}$**

- 待证: 终端$s_1, s_2, \ldots, s_i$中, 至少有一个的连接代价上界为$2 \cdot OPT/i$
- 已知: 存在一个包含$s_1, s_2, \ldots, s_i$的简单圈$C_i$, 有: $\sum_{e \in C_i} c_e \leq 2 \cdot OPT$
  - ➡ 由于该圈中共$i$条边, 故平均权重最多为$2 \cdot OPT/i$
  - ➡ 由此可断言: 至少存在一条边$(s_h, s_j)$, 其权重最多为$2 \cdot OPT/i$
- 假定终端$s_h$先到达.[反之同理]
- 终端$s_j$到达时, 贪心算法要比较它与所有已到达终端的边权.
  - ➡ 边$(s_h, s_j)$当然也要参与比较.
- 算法最终挑选的是这些边权的最小者, 故$s_j$的连接代价不超过$(s_h, s_j)$
  - ➡ 故, $s_j$的连接代价上界为$2 \cdot OPT/i$
- 得证.

# 离线Metric Steiner树

## 问题描述:[NP-H]

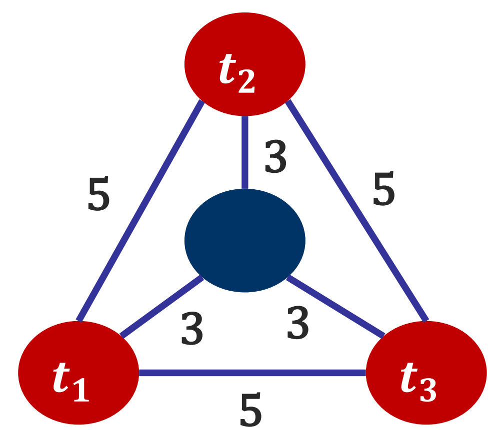
- 已知: 无向完全图$G(V,E)$, 边权: $\forall e \in E, c_e \geq 0$, 满足三角不等式.
- 给定终端集合:$R = \{t_1, t_2, \ldots, t_k\} \subseteq V$
- 要求: 最小代价子图覆盖R. 即在该子图中R中顶点相互连通.

## 基于最小生成树的算法:

- 从图$G$中删除其他顶点和边, 只留下R中顶点及其边. 得到图$G'$.
- 计算$G'$的最小生成树$T$, 返回$T$.

## 坏例

- 由于metric约束, 你或许会以为去掉的边没用.
- 但其实不然. 别忘了, 这是个NP-H问题.
- 考虑右图实例.



# 近似比更好

**Claim-19: 基于MST的离线算法2-近似.**

- 考虑最佳解$T^*$.
- 对$T^*$进行Tree-Doubling, 得到欧拉图H.
- 求H中的欧拉圈, 令为C.

  显然: $\sum_{e\in C} c_e = 2\cdot OPT$

- 对C进行Shortcut, 只保留R中顶点, 得到圈$C'$.

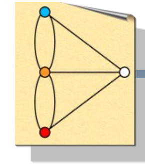
  由于三角不等式成立, 故: $\sum_{e\in C'} c_e \le 2\cdot OPT$

- 从圈$C'$中去掉一条边, 得到的是覆盖R的一棵生成树.
- 算法求得的是最小生成树.
- 得证.

# 在线算法示例

1. Scheduling
2. Bin Packing
3. Steiner Tree
4. **Bipartite Matching**

# 偶图最大匹配



## 离线问题:[属于P]

- 已知: 偶图$G(L \cup R, E)$.
- 要求: 匹配$M \subseteq E$, 最大化$|M|$.
  - 该问题可以归约为求s-t最大流. 有多项式算法.

## 在线问题:

- 已知: 左顶点集合$L$.
- 在线到达: 右顶点($\in R$), 以及与之关联的边($\in E$)
- 在线决定: 对到达的顶点, 决定是否匹配, 以及用哪条边完成匹配.
- 决策目标: 最大化匹配的数量. [与离线问题目标一致]

## Notes:

- 在线约束会导致:无论怎样决策, 都必有实例让你后悔. [敌对者论证]
- 哪怕离线问题是简单的.

# Killer APP

## 在线广告投放:

- 商业公司向互联网公司买广告, 一般都会指定该广告的受众. [例如20~30岁的男性; 15~18岁的在校学生, 等等]
- 用户点击门户网站, 或者在搜索引擎上键入关键字后, 后台的"广告服务器"将根据用户信息(注册信息,以前的访问记录等)作出决定: 在即将呈现给你的网页上应该投放怎样的广告.

## 与在线偶图匹配的关系:

- 左顶点集L: 对应各种广告.
- 右顶点集R: 对应在线到达的用户.
- 边集E: 用户满足哪些广告的预设受众约束.
- 优化目标: 最大化广告的投放量.

# 贪心在线匹配算法

## 算法描述:

- 只要有机会就给出匹配; 如果有多条边可选, 任选.

➡ 你能想象比这更糙的算法吗?

## Claim-20: 该算法1/2-竞争.

➡ 有更简单的证明方法. 但这里我们用一用**Dual Fitting.**

**匹配-LPR**

$$\textbf{Maximize} \sum_{e\in E} x_e$$

**subject to**

$$\sum_{e\in\delta(v)} x_e \le 1, \forall v \in L \cup R$$

$$x_e \ge 0, \forall e \in E$$

**匹配-LPR-D**

$$\textbf{Minimize} \sum_{v\in L\cup R} p_v$$

**subject to**

$$p_v + p_w \ge 1, \forall e(v,w) \in E$$

$$p_v \ge 0, \forall v \in L \cup R$$

# Dual-Fitting

**Claim-20: 该算法1/2-竞争.**

- $\forall v \in L \cup R$, 定义变量:

$$q_v = \begin{cases} \frac{1}{2}, & \text{如果该点被匹配} \\ 0, & \text{如果该点未被匹配} \end{cases}$$

- 显然, $\sum_{v \in L \cup R} q_v = |M|$

- 定义另一组变量: $p = 2q$

- 由于$\forall e(v,w) \in E, q_v + q_w \geq 1/2$ [否则意味着算法没有把明显可以加入M的边加入]

- 因此, $p$是对偶可行解.

**匹配-LPR**

Maximize $\sum_{e \in E} x_e$

subject to

$$\sum_{e \in \delta(v)} x_e \leq 1, \forall v \in L \cup R$$

$$x_e \geq 0, \forall e \in E$$

**匹配-LPR-D**

Minimize $\sum_{v \in L \cup R} p_v$

subject to

$$p_v + p_w \geq 1, \forall e(v,w) \in E$$

$$p_v \geq 0, \forall v \in L \cup R$$

- 故有: $|M| = \frac{1}{2}\sum_v p_v \geq \frac{1}{2} \cdot OPT$

- 得证.

# Journey Ahead

- 至此, 我们讨论了4个在线算法的单独案例.
  - 调度、装箱、施泰纳树的离线问题是NP-H的.
  - 偶图匹配的离线问题属于P.
    - 它们的在线问题的解都离最优解更远.
    - 这足以说明在线约束是个特殊的困难.
  - 有没有更系统化的设计在线算法的途径?
    - 再一次, LP及其对偶提供了新的视角.
    - 但这一次, 与LP近似算法又有很大不同.
- 接下来的论题: Online Primal-Dual.