算法设计与分析

贪心算法（一）

1. 贪心算法简介
2. 区间调度
3. 最优装载
4. 最小化延迟调度
5. 分数背包问题

大纲

动机

像国际象棋这样的博弈只能通过预判来获胜

- 一个只关注眼前利益的玩家很容易被击败。

但在许多其他博弈中，例如拼字游戏（Scrabble）

- 在当前时刻做出看起来最好的选择是可行的，而不必过多担心未来的后果。

动机

像国际象棋这样的博弈只能通过预判来获胜

- 一个只关注眼前利益的玩家很容易被击败。

但在许多其他博弈中，例如拼字游戏（Scrabble）

- 在当前时刻做出看起来最好的选择是可行的，而不必过多担心未来的后果。

这种短视行为简单且方便，使其成为一种有吸引力的算法策略。

贪心算法

贪心算法有效：正确性证明

- 区间调度：对步数进行归纳
- 最优装载：对输入规模进行归纳
- 最小化延迟调度：交换论证

贪心算法无效：寻找反例

- 硬币找零问题

大纲

区间调度

输入：$S = \{1, 2, \ldots, n\}$ 是 n 个任务的集合，任务 i 从 s_i 开始，在 f_i 结束。

- 两个任务 i 和 j 是相容的，如果它们不重叠：$s_i \geq f_j$ 或 $s_j \geq f_i$

目标：找到最大的互相相容任务子集。

区间调度

输入：$S = \{1, 2, \ldots, n\}$ 是 n 个任务的集合，任务 i 从 s_i 开始，在 f_i 结束。

- 两个任务 i 和 j 是相容的，如果它们不重叠：$s_i \geq f_j$ 或 $s_j \geq f_i$

目标：找到最大的互相相容任务子集。

实例

i	1	2	3	4	5	6	7	8
s_i	0	1	3	3	4	5	6	8
f_i	6	4	5	8	7	9	10	11

解：$\{2, 5, 8\}$

示例

*[*区间调度示意图：*8* 个任务在时间轴 *0-11* 上的分布*]*

任务 4 和 7 不相容

区间调度：贪心算法

贪心模板

- 按某种自然顺序考虑任务，然后选择每个与已选任务相容的任务。
- 选择策略是短视的 $\rightsquigarrow$ 顺序可能不是最优的

区间调度：贪心算法

贪心模板

- 按某种自然顺序考虑任务，然后选择每个与已选任务相容的任务。
- 选择策略是短视的 $\rightsquigarrow$ 顺序可能不是最优的

候选选择策略

始时间 按 s_i 的升序考虑任务

束时间 按 f_i 的升序考虑任务

短区间 按 $f_i - s_i$ 的升序考虑任务

少冲突 对每个任务 j，计算冲突任务数 c_j。按 c_j 的升序调度。

最早开始时间的反例

[反例图：一个长任务覆盖了多个短任务]

最短区间的反例

[反例图：中间的短任务阻止了两侧长任务的选择]

最少冲突的反例

[反例图：中间冲突最少的任务不是最优选择]

贪心算法：最早结束时间优先

算法 1: GreedySelect$(S, s_i, f_i, i \in [n])$

输出： 最大相容子集 $A \subseteq S$

按结束时间排序任务，使得 $f_1 \leq \cdots \leq f_n$;
$n \leftarrow |S|$;
$A \leftarrow \emptyset$;
for $i = 1$ to n **do**
  if 任务 i 与 A 相容 **then**
    $A \leftarrow A \cup \{i\}$;
  end if
end for
return A;

问：如何判断任务 i 与 A 相容?
答：记录最后加入 A 的任务 j^*。任务 i 与 A 相容当且仅当 $s_i \geq f_{j^*}$。

最早结束时间优先演示

输入：$S = \{1, 2, \ldots, 8\}$

i	1	2	3	4	5	6	7	8
s_i	0	1	3	3	4	5	6	8
f_i	6	4	5	8	7	9	10	11

解：$A = \{2, 4, 8\}$

复杂度：总体 $\Theta(n \log n)$

- 按结束时间排序：$\Theta(n \log n)$
- 比较检查相容性：$O(n)$

引理

最早结束时间优先算法总是给出正确的解。

如何证明?

贪心算法的数学归纳法

贪心算法的证明模板

1. 将正确性描述为关于自然数 n 的命题，该命题声称贪心算法产生正确的解。
   - 这里，n 可以是算法步数或输入规模。
2. 证明该命题对所有自然数成立。
   - 归纳基础：从最小的实例开始
   - 归纳步骤：第一类或第二类归纳

最早结束时间优先的命题

设 S 为规模为 n 的任务集，s_i 和 f_i 是开始时间和结束时间，A 是 S 的一个最大相容子集。

命题：当算法 GreedySelect 执行到第 k 步时，它选择了 k 个任务 $(i_1 = 1, i_2, \ldots, i_k)$，这恰好是 A 的前 k 个任务。

根据上述命题，$\forall k$，前 k 步的选择恰好是某个最大相容子集 A 的前 k 个任务，并且最多在 n 步内产生 A。

数学归纳法：归纳基础

设 $S = \{1, 2, \ldots, n\}$ 为已排序的任务集：$f_1 \leq \cdots \leq f_n$

数学归纳法：归纳基础

设 $S = \{1, 2, \ldots, n\}$ 为已排序的任务集：$f_1 \leq \cdots \leq f_n$

归纳基础：$k = 1$，证明 A 包含任务 1

数学归纳法：归纳基础

设 $S = \{1, 2, \ldots, n\}$ 为已排序的任务集：$f_1 \leq \cdots \leq f_n$

归纳基础：$k = 1$，证明 A 包含任务 1
对于任意最大相容子集 A，按结束时间升序对 A 中的任务排序。
如果 A 中的第一个任务是 j 且 $j \neq 1$，则用任务 1 替换任务 j，得到 A'：

$$A' = (A - \{j\}) \cup \{1\}$$

- 1 不会出现在 $(A - \{j\})$ 中 $\Rightarrow |A| = |A'|$
- $f_1 \leq f_j \Rightarrow$ 替换不影响相容性 $\Rightarrow A'$ 也是 A 的最大相容子集且包含任务 1。

数学归纳法：归纳步骤 (1/2)

假设命题对 k 成立，证明它对 $k+1$ 也成立

- 第 $(k+1)$ 步选择的任务 i_{k+1} 与 $(i_1, \ldots, i_k)$ 构成 S 的某个 A 的前 $k+1$ 个任务。

证明：在 k 步之后，算法选择了 $i_1 = 1, i_2, \ldots, i_k$。
前提 $\Rightarrow \exists$ 一个最大相容集 A 包含 $i_1, i_2, \ldots, i_k$。

- 设 B 为 A 中其他元素的集合（已排序且非空），S' 为关于 $\{i_1, i_2, \ldots, i_k\}$ 的相容元素集合。

$$A = \{i_1, i_2, \ldots, i_k\} \cup B$$

$$S' = \{i \mid i \in S, s_i \geq f_k\}$$

数学归纳法：归纳步骤 (2/2)

根据任务 i_{k+1} 是否是 B 中的第一个任务来考虑两种情况。

数学归纳法：归纳步骤 (2/2)

根据任务 i_{k+1} 是否是 B 中的第一个任务来考虑两种情况。

- 如果 i_{k+1} 恰好是 B 中的第一个任务，则结论直接成立，第 $(k+1)$ 步选择仍然产生 A 的部分解。

数学归纳法：归纳步骤 (2/2)

根据任务 i_{k+1} 是否是 B 中的第一个任务来考虑两种情况。

- 如果 i_{k+1} 恰好是 B 中的第一个任务，则结论直接成立，第 $(k+1)$ 步选择仍然产生 A 的部分解。
- 如果 i_{k+1} 不是 B 中的第一个任务，则必有 $i_{k+1} \notin B$
  - 贪心算法的选择策略 $\Rightarrow i_{k+1}$ 的结束时间必须早于 B 中第一个任务
  - 此时，我们可以用任务 i_{k+1} 替换 B 中的第一个任务，得到 B'。显然，$|B'| = |B|$。

$$\{i_1, i_2, \ldots, i_k\} \cup B' = A'$$

注意 $|A| = |A'| \Rightarrow A'$ 仍然是 S 的最大相容集。这证明了归纳步骤。

大纲

最优装载问题

问题：给定 n 个集装箱，重量为 w_i，以及一艘最大载重量为 W 的船（无体积限制）。
目标：找到一个装载方案，使船上的集装箱数量最大化。
分析：这个问题是 0-1 背包问题的特例。

- 物品：集装箱
- 船：背包
- 所有 $v_i = 1$

建模

设 $(x_1, x_2, \ldots, x_n)$ 为解向量，$x_i \in \{0, 1\}$。

- $x_i = 1$ 当且仅当第 i 个集装箱在船上

目标函数：

$$\max \sum_{i=1}^{n} x_i$$

约束：

$$\sum_{i=1}^{n} w_i x_i \leq W, \quad x_i \in \{0, 1\}, \quad i \in [n]$$

算法设计

贪心策略：最轻优先

算法步骤

- 按重量升序对集装箱排序，确保 $w_1 \leq w_2 \leq \cdots \leq w_n$
- 从最小标签开始装载集装箱，直到装载下一个集装箱会超过限制时停止

正确性证明（对输入规模归纳）

引理

$\forall$ 输入规模 n，算法产生正确的解。

设 $S = \{1, 2, \ldots, n\}$ 为已按升序排序的集装箱集合，且 $w_1 \leq w_2 \leq \cdots \leq w_n$。

- 归纳基础：证明当输入规模 $n = 1$（只有一个集装箱）时，贪心算法产生正确的解。显然成立。
- 归纳步骤：证明如果贪心算法对输入规模 n 产生最优解，它对输入规模 $n + 1$ 也产生最优解。

贪心算法分析：解释

$S = \{1, 2, \ldots, n+1\}, w_1 \leq \cdots \leq w_{n+1}$

↓

若 $W < w_1$，返回 $S = \{\perp\}$

↓

否则移除集装箱 1，令 $W' = W - w_1$

输入规模为 n：$S' = \{2, 3, \ldots, n+1\}$

↓

(S', W') 的最优解 I'

↓

$I \leftarrow \{I'\} \cup \{1\}$

↓

证明 I 是 (S, W) 的最优解

正确性证明 (1/2)

归纳前提：贪心策略对输入规模 n 产生最优解，考虑输入规模 $n+1$

$$S = \{1, 2, \ldots, n+1\}, w_1 \leq w_2 \leq \cdots \leq w_{n+1}$$

归纳前提 $\Rightarrow$ 对于输入规模 n

$$S' = \{2, \ldots, n+1\}, W' = W - w_1$$

贪心策略对 (S', W') 产生最优解 I'。
令 $I = I' \cup \{1\}$。

正确性证明 (2/2)

声明：I 是 (S, W) 的最优解。
反证法：若不是，假设存在 (S, W) 的最优解 I^* 且 $|I^*| > |I|$。

- 不失一般性地假设 $1 \in I^*$，否则我们可以用 1 替换 I^* 中的第一个集装箱，也得到最优解。
- $I^* - \{1\}$ 构成 (S', W') 的一个解，且

$$|I^* - \{1\}| > |I - \{1\}| = |I'|$$

I^* 的存在与 I' 是 (S', W') 的最优解这一前提矛盾。

I	1	$I - \{1\}$
I^*	1	$I^* - \{1\}$

$\Rightarrow$

$I - \{1\}$	不是最优
$I^* - \{1\}$	矛盾！

总结

0-1 背包是一个 $\mathcal{NP}$-困难问题

- 最优装载是 0-1 背包问题的变体，可以用贪心算法高效求解

正确性证明：对输入规模归纳

大纲

最小化延迟调度

最小延迟问题（最小延迟调度）

- 任务集 A，单个资源每次处理一个任务，所有任务在时刻 0 到达
- 任务 j 需要 t_j 单位的处理时间，截止时间为 d_j。显然，$t_j \leq d_j$。
- 如果任务 j 在时刻 s_j 开始，则在时刻 $f_j = s_j + t_j$ 结束。
- 调度：$S : A \to \mathbb{N}$，$S(j) = s_j$ 是任务 j 的开始时间。
- 延迟：延迟函数计算任务的延迟：

$$L(j) = \ell_j = \max\{0, f_j - d_j\} = \max\{0, s_j + t_j - d_j\}$$

目标与约束

目标：调度所有任务以最小化最大延迟

$$\min\{\max_{j\in A} \ell_j\} = \min\{\max_{j\in A}\{\max\{0, s_j + t_j - d_j\}\}\}$$

约束：无重叠

$$\forall i, j \in A, i \neq j$$

$$s_i + t_i \leq s_j \vee s_j + t_j \leq s_i$$

示例 1

A	1	2	3	4	5
S	0	5	13	17	27
T	5	8	4	10	3
D	10	12	15	11	20
L	0	1	2	16	10

表：顺序调度

示例 2

A	1	4	2	3	5
S	0	5	15	23	27
T	5	10	8	4	3
D	10	11	12	15	20
L	0	4	11	12	10

表：最早截止时间优先

最小化延迟：贪心算法

贪心模板：按某种自然顺序调度任务。

间优先 按处理时间 t_j 的升序调度任务。

A	1	2
T	1	10
D	100	10

- $\ell_1 = 0, \ell_2 = 11 - 10 = 1$
- $\ell_2 = 0, \ell_1 = 0$（更好）

他时间 按松弛时间 $d_j - t_j$ 的升序调度任务。

A	1	2
T	1	10
D	2	10

- $\ell_2 = 10 - 10 = 0, \ell_1 = 11 - 2 = 9$
- $\ell_1 = 0, \ell_2 = 10 + 1 - 10 = 1$（更好）

最小化延迟：最早截止时间优先

算法 2: Schedule(A, T, D)

对 A 中的 n 个任务排序使得 $d_1 \le d_2 \le \cdots \le d_n$;
$t \leftarrow 0$ //从时刻 0 开始;
for $j = 1$ to n **do**
    将任务 j 分配到区间 $[t, t + t_j]$;
    $s_j \leftarrow t$;
    $f_j \leftarrow t + t_j$;
    $t \leftarrow t + t_j$
end for
return 区间 $[s_1, f_1], \ldots, [s_n, f_n]$

主要思想

- 最早截止时间优先
- 连续分配任务，无空闲时间

正确性证明：交换论证

证明框架

- 分析最优解与算法解之间的差异（例如不同的顺序）
- 设计一个变换操作（例如交换），从而在有限步内逐步将最优解转换为算法解
- 变换不影响解的最优性，因为每一步都保持最优性

在这种情况下，贪心算法解 的两个性质：

- 无空闲时间：每个时刻都有任务在处理
- 无逆序：若 $d_i > d_j$ 但 $s_i < s_j$，则称 (i, j) 构成逆序

关于算法解的关键引理

引理

所有无逆序且无空闲时间的调度具有相同的最小最大延迟时间。

证明：无逆序 $\Rightarrow$ 任务按 d_i 升序排列。
可能有多个任务具有相同的截止时间。具有相同截止时间 d 的任务 $i_1, i_2, \ldots, i_k$ 可以任意排列。（绿色部分相同）

- 开始时间是 t_0，这些任务之一的结束时间是 t，在这些任务中，最大延迟是 $\max\{0, t-d\}$ $\Leftarrow$ 与 $i_1, i_2, \ldots, i_k$ 的顺序无关。

$$t = t_0 + (t_{i_1} + t_{i_2} + \cdots + t_{i_k})$$

推论

所有可能的算法解具有相同的最小最大延迟时间。

考察最优解

观察：总存在一个无空闲时间的最优调度。

算法解：最早截止时间优先调度无空闲时间。

- 我们已经消除了最优解与算法解之间的一个差异。
- 还有另一个差异：逆序

最小化延迟：逆序

逆序：给定调度 S，逆序是一对任务 i 和 j，满足 $d_i < d_j$ 但 j 排在 i 前面，即 $s_j < s_i$。

图：如前所述，任务按 $d_1 \leq d_2 \leq \cdots \leq d_n$ 编号

事实：如果一个（无空闲时间的）调度有逆序，则它至少有一对连续调度的逆序任务。（根据定义）

最小化最大延迟：逆序

声明

交换两个相邻的逆序任务会使逆序数减少 1，且不会增加最大延迟。

证明：设 ℓ 为交换前的延迟，ℓ' 为交换后的延迟。

- $i \leftrightarrow j$ 不影响其他任务的延迟时间：$\ell'_k = \ell_k$ 对所有 $k \neq i, j$
- $\ell'_i \leq \ell_i$（因为任务 i 被前移了）
- $\ell'_j = \max\{0, t^* - d_j\}$（定义），$i$ 和 j 是逆序的 $\Rightarrow d_i < d_j$，因此 $\ell'_j \leq \max\{0, t^* - d_i\} = \ell_i$

$$\Rightarrow \max\{\ell_i, \ell_j\} \geq \max\{\ell'_i, \ell'_j\}$$

综合以上内容

定理

最早截止时间优先调度 S 是最优的。

证明：定义 S^* 为最优调度。让我们看看会发生什么。

- 总可以假设 S^* 没有空闲时间。
- 如果 S^* 没有逆序，则由关键引理 $S \sim S^*$，停止。
- 如果 S^* 有逆序，设 $i \leftrightarrow j$ 是一个相邻逆序。交换 i 和 j:
  - 不增加最大延迟
  - 严格减少逆序数
- 继续上述过程直到没有逆序，我们也可以得出 $S \sim S^*$。

最大逆序数是 $n(n-1)/2$（完全逆序），因此变换将在有限步内停止。

贪心分析技巧总结

分析：找出最优解 与算法解 之间的差异。

交换论证：逐步将最优解变换为贪心算法找到的解。

- 最多需要有限步（似乎不必要）
- 变换的每一步不损害解的质量

大纲

分数背包问题

输入：给定 n 个物品，重量向量 $(w_1, \ldots, w_n)$ 和价值向量 $(v_1, \ldots, v_n)$，以及重量限制 $W > 0$。

目标：找到 $x = (p_1, \ldots, p_n) \in [0, 1]^n$（选择 n 个物品的某些比例）以满足：

- 优化目标：最大化 $\sum_{i=1}^{n} p_i v_i$
- 约束：$\sum_{i=1}^{n} p_i w_i \leq W$

区别在于现在物品是可以无限分割的。

贪心算法

贪心策略：最大单位价值优先

算法

- 按单位价值 $\alpha_i = v_i/w_i$ 的降序对 n 个物品排序。
- 迭代选择单位价值最大的物品
- 如果在某一步，背包无法装下当前单位价值最大的整个物品，我们取其中一部分来填满背包。

正确性证明 (1/3)

引理

$\forall$ 输入规模 n，算法产生最优解。

证明思路：对输入规模进行数学归纳。
归纳基础：当 $n = 1$ 时，贪心算法显然是最优解。
归纳步骤：假设算法对 $n = k$ 是最优的，则它对 $n = k + 1$ 也是最优的。

- 设 p_1 是算法对第一个物品的输出，$I' = (p_2, \ldots, p_{k+1})$ 是对实例 $(w_2, \ldots, w_{k+1})$，$(v_2, \ldots, v_{k+1})$ 和 $W - p_1 w_1$ 的输出。
- 根据归纳前提，I' 是上述规模为 $n = k$ 的子实例的最优解。令 $I = p_1 \cup I'$。

声明：I 是 $n = k + 1$ 的最优解。

正确性证明 (2/3)

反证法：若不是，则必存在一个更优解 I^*，其最大价值为 V^*。
证明 I^* 的第一个元素 p_1^* 必须等于 I 的 p_1。

1. $p_1^* = p_1$：无需证明。
2. $p_1^* > p_1$ 是不可能的，因为贪心策略保证 I 的 p_1 已经尽可能大。
3. 如果 $p_1^* < p_1$，我们总可以通过减少剩余 k 个物品的总重量 $\Delta = (p_1 - p_1^*)w_1$ 来将其增加到 p_1。注意这种调整是合理的，因为剩余 k 个物品的总重量必须大于 Δ。否则，必有 $V^* < V$，这根据前提是不可能的。然后我们考虑调整后的两种子情况：
   - 总价值不变。这只有当至少存在另一个物品 j 使得 $\alpha_j = \alpha_1$ 时才可能。
   - 总价值更高。然而，这种情况永远不会发生，因为它违背了 I^* 的假定最优性。

正确性证明 (3/3)

我们得出结论：要么 $p_1^* = p_1$，要么我们可以在不损害最优性的情况下将其调整为这种情况。

$I^* - \{p_1\}$ 构成 $W - p_1^* w_1 = W - p_1 w_1$ 的一个解，物品为 $(2, \ldots, n+1)$，总价值 $V^* - p_1 v_1 > V - p_1 v_1 \rightsquigarrow$ 矛盾于 I' 的最优性

这证明了 I 是输入规模 $n = k + 1$ 的最优解。