第3章

单机调度问题及其智能优化算法

在生产调度研究领域，单机车间调度问题一直是研究的热点，理论上单机车间调度问题可以看作其他调度问题的特殊形式。因此，深入研究单机车间调度问题可以更好地理解复杂的其他调度问题的结构，求解单机车间调度问题的启发式算法亦可以作为求解复杂调度问题算法的基础。

3.1 单机调度问题描述

单机调度(single machine scheduling，SMS)问题[1]可以描述为，n 项相互独立的任务需要在系统中的一台机器上序贯处理，每项任务有加工时间、交货期等参数，此外还要满足一些调度环境和约束条件的要求，调度目标就是要找到一个最优的任务序列使得系统总成本最小。该问题可用三参数法进行表示，如 $1|r_j,\mathrm{prmp}|\sum\omega_jC_j$ 表示一个单机调度问题，其工件 j 在其提交日期 r_j 进入系统，允许中断。该问题的优化目标是最小化加权完成时间和。

在理论上，单机调度可看作其他调度问题的特殊形式，因此深入研究单机调度问题可以更好地理解复杂的多机调度问题的结构，求解单机调度问题的启发式算法也可以作为求解复杂调度问题算法的基础。在生产实践中，复杂调度问题往往可以分解为多个单机问题来解决，若一条生产线上的某台机器成为瓶颈，则整条生产线的调度可以围绕该机器进行，就可以转化为一个单机调度问题。单机调度问题也是一类经典的 NP-Hard 问题，对单机调度问题求解算法的研究可以提供求解复杂调度问题的算法基础。

3.2 单机调度问题的数学模型

3.2.1 单机总加权完成时间调度问题的数学模型

单机总加权完成时间[3]的三参数表示法是 $1\|\sum\omega_jC_j$，其中，工件 j 的权重 ω_j

是一个重要的因素，可以表示为每单位时间的持有成本，或者是已经附加到工件 j 上的价值。这个问题引出了调度理论中非常有名的规则——加权最短加工时间优先(weighted shortest processing time first，WSPT)规则[1]。根据这个规则，工件按 ω_j/p_j 的降序来排列。

现设定如下一些假设：

(1) 工件在该台机器上加工时不能被其他工件抢占。

(2) 该机器同一时刻至多加工一个工件。

(3) 该台机器一直可用，并且在时刻 0 时机器处于空闲状态。

(4) 该台机器前有无限缓存区。

(5) 工件之间没有加工次序约束。

(6) 工件完工后立即被运走。

(7) 工件准备时间已作为其加工时间的一部分。

下面针对第一种情况给出相应的数学模型：

目标：

$$z=\min\sum_{j=1}^{n}\omega_j C_j \tag{3-1}$$

约束：

$$\frac{p_1}{\omega_1}\leqslant\frac{p_2}{\omega_2}\leqslant\cdots\leqslant\frac{p_n}{\omega_n} \tag{3-2}$$

$$p_j\geqslant 0,\quad j=1,2,\cdots,n \tag{3-3}$$

目标函数(3-1)表示加权完工时间和最小，约束条件(3-2)保证工序按照 WSPT 规则排列，约束条件(3-3)表示所有工件的加工时间应该大于或者等于 0。

3.2.2 单机提前/拖期调度问题的数学模型[4]

提前/拖期调度问题在准时制(just in time，JIT)生产中有重要意义。该问题是近 20 年来重要的研究课题，取得了许多研究成果，它的重要性体现在准时交货，可以避免不必要的在制品与成品的库存，加速资金周转，提高生产率，增强企业在客户中的信誉，提高企业产品的竞争能力。在市场经济发展的今天，这一问题的研究具有十分重要的实际意义。从理论上讲，这一领域中还有许多困难问题尚未解决，所以对它的研究具有重要的理论意义。

单机提前/拖期调度问题可以描述如下：设有 n 个工件 $j_1,j_2,\cdots,j_n$，等待在同一台机器上加工，工件之间的加工顺序无约束。工件 j_i 的加工时间为 p_i，工件 j_i 的交货期为 d_i，设工件 j_i 的实际完工时间为 c_i，如果 $c_i<d_i$，则工件 j_i 受到提前惩罚，如果 $c_i>d_i$，则工件 j_i 受到拖期惩罚。工件 j_i 的单位提前、拖期惩罚系数分别为 h_i 和 $\omega_i(i=1,2,\cdots,n)$，则工件 j_i 所受到的提前、拖期惩罚总数为：

$$g_i(c_i)=h_i\cdot\max\{0,d_i-c_i\}+\omega_j\cdot\max\{0,c_i-d_i\} \tag{3-4}$$

调度目标是确定工件的加工顺序和每个工件的实际开工时间，使得工件的提前/拖期惩罚最小，即

$$\min \sum_{i=1}^{n} g_i(c_j)$$
$$\text{s.t } c_i \leqslant c_j - p_j \quad \text{或} \quad c_j \leqslant c_i - p_i \ \forall i,j, \quad i \neq j, \quad c_i \geqslant p_i \ \forall i \tag{3-5}$$

3.3 单机调度问题的典型调度规则

单机调度相关问题的最优启发式规则包括经典的 WSPT 规则、EDD 规则和 LST 规则。不同的调度规则在不同场景中表现的性能亦存在差异。如运用工件集合中工件所需的加工时间最小的优先规则 SPT 可使平均流水时间最短[6]；运用交货期在前的工件优先的规则 EDD，可使工件集合的最大拖期时间最短。

3.3.1 WSPT 规则

WSPT 规则对 $1 \| \sum \omega_j C_j$ 是最优的。

证明 采用反证法。假设一个非 WSPT 的调度 S 是最优的。在这个调度里，必须至少有两个相邻的工件，比方说工件 j，后面接着工件 k，使得

$$\frac{\omega_j}{p_j} < \frac{\omega_k}{p_k} \tag{3-6}$$

假设工件在时间 t 开始加工，对工件 j 和工件 k 执行所谓的邻对交换。在原来的调度 S 中，工件 j 在时间 t 开始加工，紧跟着是工件 k；而在新的调度规则下，工件 k 在时间 t 开始加工，紧跟着是工件 j，所有其他工件还是保持在原来的位置上。这里把新的调度叫作 S'。在工件 j 和 k 之前加工的工件总加权完成时间不受交换的影响。在工件 j 和 k 之后加工的工件总加权完成时间也不受交换的影响。因此，在调度规则 S 和 S' 下，目标值的不同仅仅取决于工件 j 和 k（见图 3-1）。在调度规则 S 下，工件 j 和 k 的总加权完成时间是：

$$(t + p_j)\omega_j + (t + p_j + p_k)\omega_k \tag{3-7}$$

而在调度规则 S' 下其总加权完成时间是：

$$(t + p_k)\omega_k + (t + p_k + p_j)\omega_j \tag{3-8}$$

很容易验证，如果 $\omega_j / p_j < \omega_k / p_k$，则在调度规则 S' 下的两个加权完成时间之和严格小于在调度规则 S 的。这和调度规则 S 的最优性相矛盾。

根据 WSPT 规则，对工件进行排序所需的计算时间是根据两个参数的比率对工件进行排序所需的时间，其时间复杂度 $O(n\log(n))$。

优先约束如何影响总加权完成时间的最小值呢？考虑优先约束最简单的形式

(即并行链形式的优先约束,见图 3-2),这个问题仍然可以通过相对简单而有效(多项式时间)的算法解决。这个算法是基于带有优先约束调度的基本性质提出的。

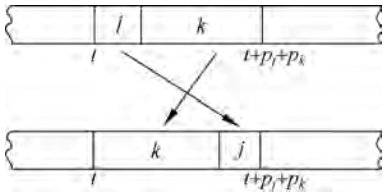

图 3-1　工件 j 和 k 的邻对交换[6]

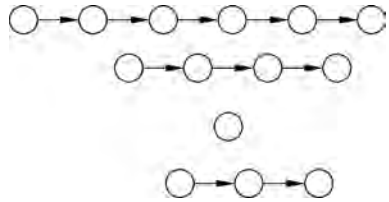

图 3-2　链式优先约束

考虑 2 条工件链条:定义第一链条Ⅰ,由工件 1,2,…,k 组成;而另一条链,定义第二链条Ⅱ,由工件 $k+1,k+2,\cdots,n$ 组成。其优先约束如下:

$$1\rightarrow 2\rightarrow\cdots\rightarrow k$$

以及

$$k+1\rightarrow k+2\rightarrow\cdots\rightarrow n$$

定理 3.1 基于如下假设:如果调度者决定开始加工一条链上的工件,在他可以继续加工另一条链上的工件之前,他必须完成整个链上的工件。问题是:如果调度者想要最小化 n 项工件的总加权完成时间,那么两条链中的哪一条应该先加工呢?

定理 3.1　如果

$$\frac{\sum_{j=1}^{k}\omega_j}{\sum_{j=1}^{k}p_j} > (<)\ \frac{\sum_{j=k+1}^{n}\omega_j}{\sum_{j=k+1}^{n}p_j} \tag{3-9}$$

那么,在工件链 $k+1,k+2,\cdots,n$ 之前(后)加工工件链 1,2,…,k 是最优的。

证明　采用反证法。在顺序 $1,2,\cdots,k,k+1,k+2,\cdots,n$ 下,总加权完成时间是:

$$\omega_1 p_1+\cdots+\omega_k\sum_{j=1}^{k}p_j+\omega_{k+1}\sum_{j=1}^{k+1}p_j+\cdots+\omega_n\sum_{j=1}^{n}p_j \tag{3-10}$$

而在顺序 $k,k+1,k+2,\cdots,n,1,2,\cdots,k$ 下,总加权完成时间是

$$\omega_{k+1}p_{k+1}+\cdots+\omega_n\sum_{j=k+1}^{n}p_j+\omega_1\Big(\sum_{j=k+1}^{n}p_j+p_1\Big)+\cdots+\omega_k\sum_{j=1}^{n}p_j \tag{3-11}$$

如果

$$\frac{\sum_{j=1}^{k}\omega_j}{\sum_{j=1}^{k}p_j} > \frac{\sum_{j=k+1}^{n}\omega_j}{\sum_{j=k+1}^{n}p_j} \tag{3-12}$$

那么，第 1 种顺序的总加权完成时间比第 2 种顺序的总加权完成时间小。结论成立。

2 条邻近工件链间的互换通常被称为邻近顺序互换（adjacent sequence interchange）。这样的互换是邻对互换的一般化。

链 $1\to2\to\cdots\to k$ 的一个重要特征定义如下：

存在 l^* 满足

$$\frac{\sum_{j=1}^{l^*}\omega_j}{\sum_{j=1}^{l^*}p_j}=\max_{1\leqslant l\leqslant k}\left(\frac{\sum_{j=1}^{l}\omega_j}{\sum_{j=1}^{l}p_j}\right) \tag{3-13}$$

左边的比率被称为链 $1,2,\cdots,k$ 的 ρ 因子，并记为 $\rho(1,2,\cdots,k)$。工件 l^* 被称为确定该链 ρ 因子的工件。

假设现在调度者允许在处理另一条链之前不必把第一条链的所有工件完成。他可以在处理某一条链的一些工件（只要坚持优先约束）时转到另一条链，然后在晚一点的时间再返回第 1 条链。在多条链的情况下，如果目标函数是总加权完成时间，那么下面的结果成立。

定理 3.2 如果工件 l^* 确定 $\rho(1,2,\cdots,k)$，那么存在一个最优顺序：连续加工工件 $1,2,\cdots,l^*$，而没有来自其他链的工件打断。

证明 采用反证法。假设在最优顺序下，子顺序 $1,2,\cdots,l^*$ 的加工被来自其他链的工件 v 中断。最优顺序包含子顺序 $1,2,\cdots,u,v,u+1,u+2,\cdots,l^*$，称为子顺序 S。足以说明，子顺序 $v,1,2,\cdots,l^*$（称为 S'），或者子顺序 $1,2,\cdots,l^*,v$（称为 S''）的总加权完成时间小于子顺序 S。如果第一个 S' 顺序不小于子顺序 S，那么第二个 S'' 顺序必然小于 S，反之亦然。由定理 3.1 可以得出，如果 S 小于 S' 的总加权完成时间，那么

$$\frac{\omega_v}{p_v}<\frac{\omega_1+\omega_2+\cdots+\omega_u}{p_1+p_2+\cdots+p_u} \tag{3-14}$$

由定理 3.1 也能得出，如果 S 小于 S'' 的总加权完成时间，那么

$$\frac{\omega_v}{p_v}>\frac{\omega_{u+1}+\omega_{u+2}+\cdots+\omega_{l^*}}{p_{u+1}+p_{u+2}+\cdots+p_{l^*}} \tag{3-15}$$

如果 l^* 是确定链 $1,2,\cdots,k$ 的 ρ 因子的工件，那么

$$\frac{\omega_{u+1}+\omega_{u+2}+\cdots+\omega_{l^*}}{p_{u+1}+p_{u+2}+\cdots+p_{l^*}}>\frac{\omega_1+\omega_2+\cdots+\omega_u}{p_1+p_2+\cdots+p_u} \tag{3-16}$$

如果 S 比 S'' 好，那么

$$\frac{\omega_v}{p_v}>\frac{\omega_{u+1}+\omega_{u+2}+\cdots+\omega_{l^*}}{p_{u+1}+p_{u+2}+\cdots+p_{l^*}}>\frac{\omega_1+\omega_2+\cdots+\omega_u}{p_1+p_2+\cdots+p_u} \tag{3-17}$$

因此，S' 比 S 好。如果链由多项工件打断，则可以进行同样的讨论。

定理 3.2 的结果是直观的。定理的条件意味着权重除以顺序 1,2,…,l^* 中工件加工时间的比率在某种意义上是递增的。如果已经决定开始加工一串工件,那么一直加工至工件 l^* 完成而没有任何其他工件在中间加工是最好的调度方案。

前面的两个定理包含了一个简单算法的基础,当优先约束表现为链式时,该算法可以最小化总加权完成时间。

算法 3.1　(带链式约束的总加权完成时间):只要机器空闲了,选择剩下链中具有最高 ρ 因子的链,无中断地加工该链直到并包括确定它的 ρ 因子的工件为止。例 3.1 说明了如何使用该算法。

例 3.1　(带链式约束的总加权完成时间)考虑下面两条链:

$$1\rightarrow 2\rightarrow 3\rightarrow 4 \quad 和 \quad 5\rightarrow 6\rightarrow 7$$

表 3-1 给出了工件的权重和加工时间。

表 3-1　例 3.1 工件的权重和加工时间

工件	1	2	3	4	5	6	7
ω_j	6	18	12	8	8	17	18
p_j	3	6	6	5	4	8	10

第 1 条链的 ρ 因子是(6+18)/(3+6)并且是由工件 2 确定。第 2 条链的 ρ 因子是(8+17)/(4+8)并且是由工件 6 确定。由于 24/9 大于 25/12,所以工件 1 和工件 2 先加工。第 1 条链剩下部分的 ρ 因子是 12/6 并且是由工件 3 确定的。由于 25/12 大于 12/6,所以工件 5 和 6 接着工件 1 和 2 进行加工。第 2 条链剩下部分的 ρ 因子是 18/10 并且是由工件 7 确定的,所以工件 3 在工件 6 之后加工。由于工件 7 的 ω_j/p_j 比率高于工件 4 的比率,所以工件 7 在工件 3 之后加工,工件 4 最后加工。

对于 $1|\text{prec}|\sum\omega_jC_j$,已经得到比刚才考虑的并行链更一般的优先约束的多项式时间算法。然而,对于任意的优先约束,该问题是强 NP-Hard 问题。

到目前为止,假设所有工件在零时刻均是可加工的。考虑工件到达时间不一致且允许中断的情况,即 $1|r_j,\text{prmp}|\sum\omega_jC_j$。首先要考虑的问题是,WSPT 规则的中断形式是否是最优的。一个 WSPT 规则的中断形式可简述如下:在任何时间点,选择权重和剩余加工时间比率最大的可加工工件进行加工。因此当工件加工时,它的优先级会增加;正因为如此,一个工件不会被另一个同样可加工的工件所中断。然而,一个工件可能被新提交的具有更高优先因子的工件所中断。尽管这个规则看起来像无中断的 WSPT 规则的逻辑性扩展,但它并不一定能够得到最优调度,因为这个问题是强 NP-Hard 问题。

如果所有的权重是相等的,那么 $1|r_j,\text{prmp}|\sum C_j$ 问题就很容易。但是这个问题的无中断形式(即 $1\mid r_j\mid\sum C_j$)仍然是强 NP-Hard 问题。

总加权折扣完成时间 $\sum\omega_j(1-e^{-rc_j})$(其中 r 是折扣因子),在某种程度上被描述成是总加权(无折扣)完成时间的一般化目标。$1\|\sum\omega_j(1-e^{-rc_j})$ 问题提出了一个不同的优先级规则,即按公式(3-18)

$$\frac{\omega_j e^{-rp_j}}{1-e^{-rp_j}} \tag{3-18}$$

的降序调度工件的规则。在后续章节中,这条规则被称为权重折扣最短加工时间优先(weight discounted shortest processing time first,WDSPT)规则。

3.3.2 EDD 规则[7]

EDD 规则考虑的目标和交货期相关。考虑与交货期相关的一般性问题,也就是问题 $1|\text{prec}|h_{\max}$,这里

$$h_{\max}=\max(h_1(c_1),h_2(c_2),\cdots,h_n(c_n)) \tag{3-19}$$

其中,$h_j(j=1,2,\cdots,n)$是非减成本函数,其目标与工期相关。即使当工件受限于任意优先约束,这个问题也可以用后向动态规划算法进行有效求解。

显然,最后一项工件的完工时间 $C_{\max}=\sum p_j$,这与调度方案是无关的。J 表示已经调度的工件集合,它们在时间区间内进行加工。

$$\left(C_{\max}-\sum_{j\in J}p_j,C_{\max}\right) \tag{3-20}$$

集合 J 的补集为集合 J^c,表示仍然等待调度的工件集合,而 J^c 的子集 J'表示在 J 前可以立即调度的工件集合(即所有直接后续已经在 J 中的工件集合)。集合 J'称作可调度工件集。下面是后向算法得到最优调度。

算法 3.2 (最小化最大成本)

步骤 1 设 $J=\varnothing$,$J^c=\{1,2,\cdots,n\}$,而 J'是没有后续的所有工件集合。

步骤 2 使 j^* 满足

$$h_{j^*}\left(\sum_{j\in J^c}p_j\right)=\min_{j\in J'}\left(h_j\left(\sum_{k\in J^c}p_k\right)\right) \tag{3-21}$$

将 $\boldsymbol{J}^*$ 加到 J,从 J^c 中删除 $\boldsymbol{J}^*$,修改 J'使它表示新的可调度工件集合。

步骤 3 如果 $J^c=\varnothing$,则停止,否则跳转到步骤 2。

定理 3.3 对 $1|\text{prec}|h_{\max}$ 应用算法 3.2 得到一个最优调度方案。

证明 采用反证法。假设在一个给定的迭代中从 J'中选出工件 $\boldsymbol{J}^{**}$,在 J'所有的工件中不具有最小完成成本

$$h_{j^*}\left(\sum_{j\in J^c}p_j\right) \tag{3-22}$$

则 $\boldsymbol{J}^{**}$ 是选择的工件。最小成本工件 $\boldsymbol{J}^*$ 必然在晚一些的迭代中进行调度,这意味着 $\boldsymbol{J}^*$ 必须在 $\boldsymbol{J}^{**}$ 之前出现在顺序中。许多工件可能会出现在 $\boldsymbol{J}^*$ 和 $\boldsymbol{J}^{**}$ 之

间，如图 3-3 所示。

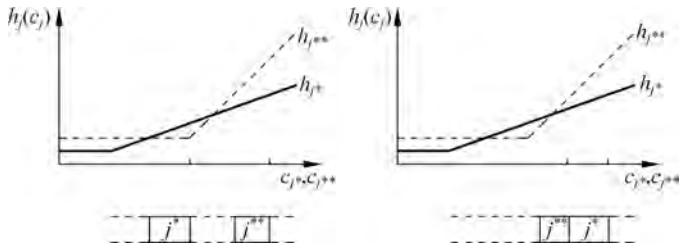

图 3-3　定理 3.3 的最优性证明

为说明这个顺序不可能是最优的，选取工件 $\boldsymbol{J}^{*}$，将其插入工件 $\boldsymbol{J}^{**}$ 后面且紧随其后，那么最初的调度中在 $\boldsymbol{J}^{*}$ 和 $\boldsymbol{J}^{**}$ 之间的所有工件，包括 $\boldsymbol{J}^{**}$，均得到提前。唯一的完成成本增加的工件是 $\boldsymbol{J}^{*}$。然而，由定义知道，现在它的完成成本比最初调度下工件 $\boldsymbol{J}^{**}$ 的完成成本小，因此在插入工件 $\boldsymbol{J}^{*}$ 后最大完成成本降低了。

例 3.2　（最小化最大成本）

考虑表 3-2 中的 3 个工件：

表 3-2　例 3.2 的 3 个工件

工　　件	1	2	3
p_j	2	3	5
$h_j(C_j)$	$1+C_1$	$1.2C_2$	10

制造期 $C_{\max}=10$ 并且 $h_3(10)<h_1(10)<h_2(10)$（因为 $10<11<12$）。因此，工件 3 最后进行调度并且必须在时间 5 的时候开始加工。为确定哪个工件在工件 3 之前进行加工，必须比较 $h_2(5)$ 和 $h_1(5)$。在最优调度中，工件 1 和 2 都可以在工件 3 前进行加工，因为 $h_2(5)=h_1(5)=6$，所以有两个调度是最优的：1，2，3 和 2，1，3。

问题 $1\|L_{\max}$ 是 $1|\text{prec}|h_{\max}$ 的最有名的特例。函数 h_j 定义为 C_j-d_j，按算法 3.2 获得的工期升序排列工件的调度方案，即最早交货期（EDD）规则。

动态规则的一个例子是最小松弛（minimum slack，MS）[8,9] 优先。

另一个常用到的规则就是先来先服务规则，等同于提交日期最早优先（earliest release date，ERD）规则。这个规则试图让各工件的等待时间相等（也就是说最小化等待时间的方差）。

3.3.3　总加权滞后时间[10]

问题 $1\left\|\sum\omega_j T_j\right.$ 是 $1\left\|\sum T_j\right.$ 问题的重要的一般化。许多学者研究了这个问题并且已经试验出了很多不同的方法。这些方法涵盖了从非常复杂的密集型计算机技术到相当粗糙的起初为应用目的而设计的启发式算法。

该问题描述的对 $1\|\sum T_j$ 的动态规划算法也可以处理一致的权重，$p_j \geqslant p_k \Rightarrow \omega_j \leqslant \omega_k$。

定理 3.4 如果有两个工件 j 和 k，$d_j \leqslant d_k$，$p_j \leqslant p_k$ 及 $\omega_j \geqslant \omega_k$，那么，存在工件 j 在工件 k 之前的最优调度。

证明基于(不必是相邻的)成对交换的讨论。

遗憾的是，对任意权重的 $1\|\sum \omega_j T_j$ 不能得到有效的算法。

定理 3.5 问题 $1\|\sum \omega_j T_j$ 是强 NP-Hard 问题。

证明 证明再次基于对 $1\|\sum \omega_j T_j$ 的三划分归结而完成。归结基于如下变形：再一次选择工件的数量 $n=4t-1$，以及

$d_j=0$，　$p_j=a_j$，　$\omega_j=a_j$，　$j=1,2,\cdots,3t$

$d_j=(j-3t)(b+1)$，　$p_j=1$，　$\omega_j=2$，　$j=3t+1$，　$3t+2,\cdots,4t-1$

使

$$z=\sum_{1\leqslant j\leqslant k\leqslant 3t} a_j a_k+\frac{1}{2}(t-1)tb \tag{3-23}$$

可以证明，存在目标值为 z 的调度，当且仅当三划分问题存在解。前 $3t$ 个工件的 ω_j/p_j 比率等于 1，并且在时间 0 到期。有 $t-1$ 个 ω_j/p_j 比率等于 2 的工件，它们的工期是在 $b+1$，$2b+2$，等等。可以得到值为 z 的解，如果这 $t-1$ 个工件可以精确地在区间

$[b,b+1]$，　$[2b+1,2b+2]$，　…，　$[(t-2)b+t-2,(t-1)b+t-1]$

进行加工(见图 3-4)。为把 $t-1$ 个工件填到 $t-1$ 个区间中，前 $3t$ 个工件必须拆分成 t 个子集，每个子集含 3 个工件，每个子集中 3 个加工时间的和等于 b。可以验证，这种情况下加权滞后之和等于 z。

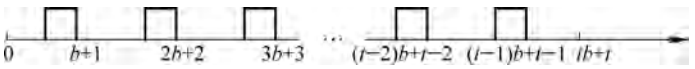

图 3-4　$1\|\sum \omega_j T_j$ 的三划分归结

当不存在这样一个划分时，那么至少存在一个子集，其中 3 个工件的加工时间之和大于 b，并且另一个子集中 3 个工件的加工时间之和小于 b。可以验证，在这种情况下，加权滞后之和大于 z。

通常，分枝定界法[12]用于 $1\|\sum \omega_j T_j$。一般来说，调度从末端开始构造(即后向时间)。在搜索树的 j 层上，工件被放在第 $(n-j+1)$ 位置。所以从 $j-1$ 层的每个顶点，有 $n-j+1$ 个分枝到达 j 层。没有必要评估所有可能的顶点，如定理 3.4 中描述的支配结论可以删除许多顶点。$\boldsymbol{J}$ 层顶点数量的上界是 $n!/(n-j)!$。后向构造顺序的依据是目标函数中较大的项可能和放在调度后面的工件相关联，所以从末端开始构造顺序是有利的。

有很多不同的定界技巧，较简单的定界技巧之一是将问题松弛为运输问题。在这个程序中，每个(整数)加工时间 p_j 的工件 j 分成 p_j 个工件，每项是单位加工时间。如果工件 j 的一个单位工件在时间区间 $(k-1,k)$ 内加工，那么决策变量 x_{jk} 就是 1，否则是 0。这些决策变量 x_{jk} 必须满足两组约束：

$$\sum_{k=1}^{c_{\max}} x_{jk} = p_j, \quad j = 1,2,\cdots,n \tag{3-24}$$

$$\sum_{j=1}^{n} x_{jk} = 1, \quad k = 1,2,\cdots,C_{\max} \tag{3-25}$$

显然，满足这些约束的解并不保证是没有中断的可行调度。定义成本系数 c_{jk}，它满足

$$\sum_{k=l-p_j+1}^{l} c_{jk} \leqslant \omega_j \max(l-d_j,0), \quad j = 1,2,\cdots,n; \quad l = 1,2,\cdots,C_{\max} \tag{3-26}$$

那么最小成本解提供了一个下界，因为对于 $x_{jk}=1(k=C_j-p_j+1,\cdots,C_j)$ 的运输问题的任何解，等式(3-27)成立：

$$\sum_{k=1}^{c_{\max}} c_{jk}x_{jk} = \sum_{k=c_j-p_j+1}^{c_j} c_{jk} \leqslant \omega_j \max(C_j-d_j,0) \tag{3-27}$$

非常容易找到满足这个关系的成本函数。例如，设

$$c_{jk} = \begin{cases} 0, & k \leqslant d_j \\ \omega_j, & k > d_j \end{cases} \tag{3-28}$$

运输问题的解为 $1\|\sum \omega_j T_j$ 提供了一个下界。这个定界技巧适用于树的每个点所对应的未调度工件。如果这个下界比任何已知的调度解大，那么就可以删除该点。

例 3.3　(最小化总加权滞后时间)

考虑表 3-3 中的 4 个工件：

表 3-3　例 3.3 的 4 个工件

工件	1	2	3	4
ω_j	4	5	3	5
p_j	12	8	15	9
d_j	16	26	25	27

由定理 3.4 立即得到在最优顺序中，工件 4 跟在工件 2 后面，工件 3 跟在工件 1 后面。根据时间后向构造分枝定界树。只有两个工件需考虑作为最后一个位置的候选，即工件 3 和工件 4。图 3-5 中描述了需要考查的分枝定界树的顶点。为了先选一分枝进行搜索，对 1 层的两个点确定边界。