第3章 单机模型(确定性模型)

对各种原因来说,单机模型都是很重要的。单机环境非常简单,并且是所有其他加工环境的一个特例。单机模型通常具有并行或串行机器都没有的性质。从单机模型中得到的结果不但可以对单机环境提供深刻的认识,也能为应用于更复杂机器环境的启发式算法提供一些基本性质。在实际中,复杂机器环境里的调度问题通常可以分解为若干单机问题。例如,一个具有单个瓶颈的复杂机器环境可以建立单机模型来研究。

本章将详细分析各种单机模型。首先考虑总加权完成时间的目标,接着是几个和工期相关的目标,包括最大延迟、滞后工作的数量、总滞后时间和加权总滞后时间。本章中考虑的所有目标函数都是规则的。

在本章考虑的绝大多数模型中,可中断的调度方案没有任何优势。对于这些模型,可以证明在可中断调度规则类别里面的最优调度是无中断的。然而,如果工作不在同一时间提交,那么中断也许会带来好处。如果在一个不可中断环境中工作不是在同一时间提交,那么具有非强制空闲也许是有利的(即也许没有非延迟的最优调度规则)。

3.1 总加权完成时间

首先要考虑的目标是总加权完成时间 (即 $1\|\sum w_jC_j$)。工作 j 的权重 w_j是一个重要的因素,它可以代表每单位时间的持有成本,或者是已经附加到工作 j 上的价值。这个问题引出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。根据这个规则,工作按 w_j/p_j的降序来排列。

定理 3.1.1 WSPT 规则对 $1\|\sum w_jC_j$ 是最优的。

证明:反证法。假设一个非 WSPT 的调度 S 是最优的。在这个调度里,必须至少有两项相邻的工作——比方说工作 j,后面接着工作 k,使得

$$\frac{w_j}{p_j}<\frac{w_k}{p_k}$$

假设工作 j 在时间 t 开始加工。对工作 j 和工作 k 执行所谓的邻对交换。在原来的调度 S 中，工作 j 在时间 t 开始加工，紧跟着是工作 k；而在新的调度规则下，工作 k 在时间 t 开始加工，紧跟着是工作 j。所有其他工作还是保持在它们原来的位置上。把新的调度叫做 S'。在工作 j 和 k 之前加工的工作总加权完成时间不受交换的影响。在工作 j 和 k 之后加工的工作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作 j 和 k(见图 3.1)。在 S 下工作 j 和 k 的总加权完成时间是

$$(t+p_j)w_j+(t+p_j+p_k)w_k$$

而在 S' 下的是

$$(t+p_k)w_k+(t+p_k+p_j)w_j$$

很容易验证，如果 $w_jp_j<w_k/p_k$，则在 S' 下的两个加权完成时间之和严格小于在 S 下的。这和 S 的最优性相矛盾。 【证毕】

根据 WSPT，对工作进行排序所需要的计算时间是根据两个参数的比率对工作进行排序所需要的时间。一个简单的排序可以在时间 $O(n\log(n))$ 内完成，见附录 C 中例 C.1.1。

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而提出的。

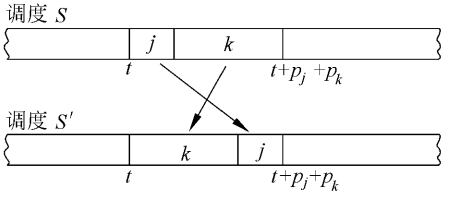

图 3.1 工作 j 和 k 的邻对交换

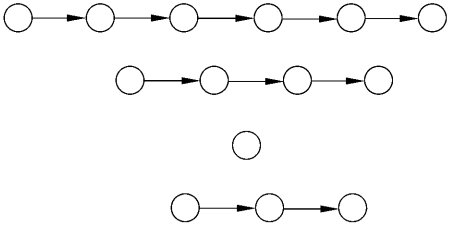

图 3.2 链式优先约束

考虑两条工作链条。一条链，比方说链Ⅰ，由工作 $1,2,\cdots,k$ 组成；而另一条链，比方说链Ⅱ，由工作 $k+1,k+2,\cdots,n$ 组成。优先约束如下：

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

以及

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

下一个引理基于如下假设：如果调度者决定开始加工一条链上的工作，在他可以继续加工另一条链上的工作之前，他必须完成整个链上的工作。问题是：如果调度者想要最小化 n 项工作的总加权完成时间，那么两条链中的哪一条应该先加工呢？

引理 3.1.2　如果

$$\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} > (<) \frac{\sum_{j=k+1}^{n} w_j}{\sum_{j=k+1}^{n} p_j}$$

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

证明: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

$$w_1 p_1 + \cdots + w_k \sum_{j=1}^{k} p_j + w_{k+1} \sum_{j=1}^{k+1} p_j + \cdots + w_n \sum_{j=1}^{n} p_j$$

而在顺序 $k+1,k+2,\cdots,n,1,2,\cdots,k$ 下,它是

$$w_{k+1} p_{k+1} + \cdots + w_n \sum_{j=k+1}^{n} p_j + w_1 \left(\sum_{j=k+1}^{n} p_j + p_1 \right) + \cdots + w_k \sum_{j=1}^{n} p_j$$

如果

$$\frac{\sum_{j=1}^{k} w_j}{\sum_{j=1}^{k} p_j} > \frac{\sum_{j=k+1}^{n} w_j}{\sum_{j=k+1}^{n} p_j}$$

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。　【证毕】

两条邻近工作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 $1 \to 2 \to \cdots \to k$ 的一个重要特征定义如下:使得 l^* 满足

$$\frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j} = \max_{1 \leqslant l \leqslant k} \left(\frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j} \right)$$

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

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作完成。他可以在处理某一条链的一些工作(只要坚持优先约束)时转到另一条链,然后在晚一点时间再返回到第 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。

引理 3.1.3　如果工作 l^* 确定 $\rho(1,2,\cdots,k)$,那么存在一个最优顺序:连续加工工作 $1,2,\cdots,l^*$,而没有来自其他链的工作打断。

证明: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作,比方说工作 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1个顺序不小于，那么第2个顺序必然小于。反之亦然。由引理3.1.2可以得出，如果 S 小于 S' 的总加权完成时间，那么

$$\frac{w_v}{p_v}<\frac{w_1+w_2+\cdots+w_u}{p_1+p_2+\cdots+p_u}$$

由引理3.1.2也能得出，如果 S 小于 S'' 的总加权完成时间，那么

$$\frac{w_v}{p_v}>\frac{w_{u+1}+w_{u+2}+\cdots+w_{l^*}}{p_{u+1}+p_{u+2}+\cdots+p_{l^*}}$$

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

$$\frac{w_{u+1}+w_{u+2}+\cdots+w_{l^*}}{p_{u+1}+p_{u+2}+\cdots+p_{l^*}}>\frac{w_1+w_2+\cdots+w_u}{p_1+p_2+\cdots+p_u}$$

如果 S 比 S'' 好，那么

$$\frac{w_v}{p_v}>\frac{w_{u+1}+w_{u+2}+\cdots+w_{l^*}}{p_{u+1}+p_{u+2}+\cdots+p_{l^*}}>\frac{w_1+w_2+\cdots+w_u}{p_1+p_2+\cdots+p_u}$$

因此，S' 比 S 好。如果链由多项工作打断，则可以进行同样的讨论。 【证毕】

引理3.1.3的结果是直观的。引理的条件意味着权重除以串 $1,2,\cdots,l^*$ 中工作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作，那么一直加工直到工作 l^* 完成而没有任何其他工作在之间加工是好的调度方案。

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

算法3.1.4（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 因子的工作为止。

下面的例子说明了如何使用该算法。

例3.1.5（带链式约束的总加权完成时间）

考虑下面两条链：

$$1\rightarrow2\rightarrow3\rightarrow4 \text{ 和 } 5\rightarrow6\rightarrow7$$

下面的表格给出了工作的权重和加工时间。

工作	1	2	3	4	5	6	7
w_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)并且是由工作2确定的。第2条链的 ρ 因子是(8+17)/(4+8)并且是由工作6确定的。由于24/9大于25/12，所以工作1和工作2先加工。第1条链剩下部分的 ρ 因子是12/6并且是由工作3确定的。由于25/12大于12/6，所以工作5和6接着工作1和2进行加工。第2条链剩下的部分的 ρ 因子是18/10并

且是由工作 7 确定的,所以工作 3 在工作 6 之后加工。由于工作 7 的w_j/p_j比率高于工作 4 的比率,所以工作 7 在工作 3 之后加工,工作 4 最后加工。

对于 $1 \mid \text{prec} \mid \sum w_j C_j$,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 难的。

到目前为止,所有工作都被假设成在时间零点(已经提交)是可加工的。考虑工作在不同的时间点提交并且调度者可以中断的问题(即 $1 \mid r_j, \text{prmp} \mid \sum w_j C_j$)。第一个要考虑的问题是,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作进行加工。因此当工作加工时,它的优先级会增加;也因为这样,一项工作不会被另一项在它开始加工的时间已经可加工的工作所中断。然而,一项工作可能被新提交的具有更高优先因子的工作所中断。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 难的(参见附录 D)。

如果所有的权重是相等的,那么 $1 \mid r_j, \text{prmp} \mid \sum C_j$ 问题是很容易的(见练习 3.15)。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 难的。

在第 2 章中,总加权折扣完成时间 $\sum w_j(1-\mathrm{e}^{-rC_j})$,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。$1 \| \sum w_j(1-\mathrm{e}^{-rC_j})$ 问题提出了一个不同的优先级规则,即按

$$\frac{w_j \mathrm{e}^{-rp_j}}{1-\mathrm{e}^{-rp_j}}$$

的降序调度工作的规则。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。

定理 3.1.6　对 $1 \| \sum w_j(1-\mathrm{e}^{-rC_j})$, WDSPT 规则是最优的。

证明:反证法。再次假设一个不同的调度规则,比如说调度 S,是最优的。在这个调度下,必然有两项工作 j 和 k,工作 k 在工作 j 之后,满足

$$\frac{w_j \mathrm{e}^{-rp_j}}{1-\mathrm{e}^{-rp_j}} < \frac{w_k \mathrm{e}^{-rp_k}}{1-\mathrm{e}^{-rp_k}}$$

假设工作 j 在时间 t 开始它的加工。两项工作的邻对互换得到调度规则 S'。很明显,目标中唯一的不同之处是由于工作 j 和 k 引起的。在 S 中,工作 j 和 k 对目标函数的贡献等于

$$w_j(1-\mathrm{e}^{-r(t+p_j)}) + w_k(1-\mathrm{e}^{-r(t+p_j+p_k)})$$

在 S'中,工作 j 和 k 对目标的贡献通过互换这个表达式中的 j 和 k 获得。容易证明,S'小于 S 中的目标函数值。这导致矛盾。　【证毕】

正如第 2 章中讨论过的那样，总无折扣加权完成时间是总折扣加权完成时间 $\sum w_j(1-e^{-rC_j})$ 的极限情况。如果 r 足够接近于 0(注意 WDSPT 对 $r=0$ 没有进行合适的定义)，WDSPT 能得到和 WSPT 相同的排序。

$\sum w_jC_j$ 和 $\sum w_j(1-e^{-rC_j})$ 都是更一般的目标函数 $\sum w_jh(C_j)$ 的特殊情况。已经证明，仅当函数 $h(C_j)=C_j$ 和 $h(C_j)=1-e^{-rC_j}$ 时，可以得到工作按某种函数 $g(w_j,p_j)$ 降序排列的简单优先规则。对其他任何成本函数 h 不存在一个最优的这种优先级函数 g。然而，目标 $\sum w_jh(C_j)$ 可通过动态规划来处理(参见附录 B)。

和引理 3.1.2 对 WSPT 一般化邻对互换的讨论相似，对 WDSPT 的最优性证明中存在一个邻对顺序互换结果，它是邻对互换讨论的一般化(见练习 3.21)。

3.2 最大延迟

下面 4 节(3.2～3.5 节)考虑的目标是和工期相关的。首先考虑的和工期相关的模型具有一般的性质，也就是问题 $1|\text{prec}|h_{\max}$，这里

$$h_{\max}=\max(h_1(C_1),h_2(C_2),\cdots,h_n(C_n))$$

其中，$h_j(j=1,\cdots,n)$是非减成本函数。因为函数 h_j 可以是图 2.1 中所描述的任何一种形式，所以目标显然是和工期相关的。即使当工作受限于任意优先约束时，这个问题也可以用后向动态规划算法进行有效的求解。

显然，最后一项工作完成发生在制造期 $C_{\max}=\sum p_j$，这和调度方案是无关的。J 表示已经调度的工作集合，它们在时间段

$$\left[C_{\max}-\sum_{j\in J}p_j,C_{\max}\right]$$

内进行加工。集合 J 的补集为集合 J^c，表示仍然等待调度的工作集合，而 J^c 的子集 J' 表示在 J 前可以立即调度的工作集合(即所有直接后继已经在 J 中的工作集合)。集合 J' 称作可调度工作集。下面的后向算法得到最优调度。

算法 3.2.1(最小化最大成本)

步骤 1 设 $J=\varnothing$，$J^c=\{1,2,\cdots,n\}$，而 J' 是没有后继的所有工作集合。

步骤 2 使 j^* 满足

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

将 j^* 加到 J，从 J^c 中删除 j^*，修改 J' 使它表示新的可调度工作集合。

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

定理 3.2.2 对 $1|\text{prec}|h_{\max}$，算法 3.2.1 得到一个最优调度方案。

证明：反证法。假设在一个给定的迭代中从 J' 中选出的工作 j^{**}，在 J' 所有的工作中不具有最小完成成本

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

则说 j^{**} 是选择的工作。最小成本工作 j^* 必然在晚一些的迭代中进行调度，这意味着 j^* 必须在 j^{**} 之前出现在顺序中。许多工作可能会出现在 j^* 和 j^{**} 之间(见图 3.3)。

为说明这个顺序不可能是最优的，选取工作 j^* 并且将它插入在工作 j^{**} 后面并紧随其后。最初的调度中在 j^* 和 j^{**} 之间的所有工作，包括 j^{**}，现在都提前了。唯一的完成成本增加的工作是 j^*。然而，现在它的完成成本，从定义知道，比最初调度下工作 j^{**} 的完成成本小，因此在插入 j^* 后最大完成成本降低了。　【证毕】

此算法需要的计算时间，最坏情况可确定如下。调度 n 项工作有 n 个步骤，每一步最多有 n 项工作需要考虑，因此该算法总的运行时间以 $O(n^2)$ 为界。

下面的例子说明了该算法的应用。

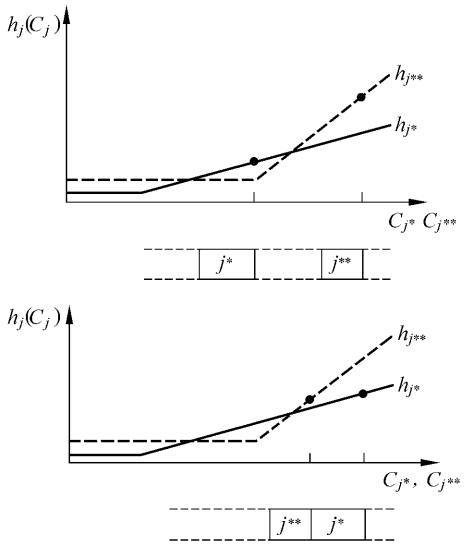

图 3.3　定理 3.2.2 的最优性证明

例 3.2.3(最小化最大成本)

考虑以下 3 项工作：

工作	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 $10<11<12$)。因此工作 3 最后进行调度并且必须在时间 5 的时候开始它的加工。为确定哪项工作在工作 3 之前进行加工，必须比较 $h_2(5)$ 和 $h_1(5)$。在最优调度中，工作 1 和 2 都可以在工作 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 $1\parallel L_{\max}$ 是 $1\mid \text{prec}\mid h_{\max}$ 的最有名的特例。函数 h_j 定义为 C_j-d_j，而算法得到按工期升序排列工作的调度，即最早工期(earliest due date，EDD)优先。

$1\parallel L_{\max}$ 的一般化是工作在不同的时间点提交的 $1\mid r_j\mid L_{\max}$ 问题。这个一般化形式不允许中断，比那些在同一时间可加工工作的调度问题难得多。最优调度不必是非延迟

的调度。在新的工作提交之前保持机器空闲可能是有好处的。

定理 3.2.4 问题 $1 \mid r_j \mid L_{\max}$ 是强 NP 难的。

证明：证明基于对 $1 \mid r_j \mid L_{\max}$ 三划分问题的归结。给定整数 $a_1, a_2, \cdots, a_{3t}, b$，使得 $b/4 < a_j < b/2$ 以及 $\sum_{j=1}^{3t} a_j = tb$ 可以构造出下面 $1 \mid r_j \mid L_{\max}$ 的实例。工作的数量 n 等于 $4t-1$，以及

$$r_j = jb + (j-1), \quad p_j = 1, \quad d_j = jb + j, \quad j = 1, 2, \cdots, t-1$$

$$r_j = 0, \quad p_j = a_{j-t} + 1, \quad d_j = tb + (t-1), \quad j = t, t+1, \cdots, 4t-1$$

令 $z=0$。$L_{\max} \leqslant 0$ 的调度存在，当且仅当每项工作 $j, j=1,2,\cdots,t-1$，可以在 r_j 和 $d_j = r_j + p_j$ 之间加工。这可以完成当且仅当剩余的工作可以拆分在 t 个长度为 b 的区间上，而这可以完成当且仅当三划分问题有解(见图 3.4)。 【证毕】

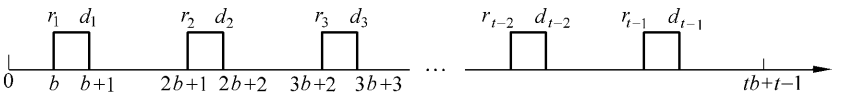

图 3.4 $1 \mid r_j \mid L_{\max}$ 是强 NP 难的

问题 $1 \mid r_j \mid L_{\max}$ 非常重要，因为在流水车间和加工车间问题中它作为启发式程序的子问题频繁出现。它已经得到相当多的研究，产生了很多合理有效的枚举分支定界算法。分支定界程序基本是枚举方法，通过证明从这个类中得到的调度目标函数值比可证明的下界大，它把某些或某类调度结果删掉，这个下界大于等于之前获得的调度的目标值。

对 $1 \mid r_j \mid L_{\max}$ 的分支定界程序可构造如下。分支过程可以基于事实，调度是从调度的早期开始逐步形成的。在第 0 层有一个单独的点，它是树的顶端。在这个点，没有任何一项工作放在顺序中。在第 1 层有 n 支分解到 n 个顶点。这一层的每个点有一项特定的工作放在调度的第 1 个位置。所以在每个这样的点上，仍然有 $n-1$ 项工作，它们在调度中的位置还必须进行确定。第 1 层的每个顶点有 $n-1$ 条弧指向第 2 层。因此第 2 层有 $(n-1)(n-2)$ 个顶点。在第 2 层的每个顶点上，前两个位置的两项工作已经确定；在第 k 层，前 k 个位置的工作已经确定。事实上，没有必要考虑每一项剩余工作都作为下一个位置的候选。如果在第 $k-1$ 层的一个顶点，工作 $j_1, j_2, \cdots, j_{k-1}$ 已经调度安排为前 $k-1$ 项工作，那么只有当

$$r_{j_k} < \min_{l \in J}(\max(t, r_l) + p_l)$$

时工作 j_k 才需要被考虑。这里，J 表示尚未调度的工作集合，t 表示假设工作 j_k 开始的时间。这个条件的原因很清楚，如果工作 j_k 不满足这个不等式，那么选择最小化右端的工作而不是 j_k，不增加 $L_{\max}$ 的值。因此分支规则相当简单。

有很多方法可以得到顶点的界。对第 $k-1$ 层的一个顶点的简单下界可以通过根据

中断 EDD 规则调度剩余的工作集合 J 来确定。众所周知,中断 EDD 规则对 $1|r_j,\text{prmp}|L_{\max}$ 是最优的(见练习 3.24),因此为该问题提供了一个下界。如果中断 EDD 规则得到了无中断的调度规则,那么可以忽略所有具有更大下界的顶点。

例 3.2.5(应用于最小化最大延迟的分支和定界)

考虑下面 4 项工作:

工作	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10

在搜索树的第 1 层,有 4 个点:(1,*,*,*),(2,*,*,*),(3,*,*,*),(4,*,*,*)。很容易看出,点(3,*,*,*)和(4,*,*,*)可以立即删掉。工作 3 在时间 3 提交,如果工作 2 在时间 1 开始加工,则工作 3 仍然可以在时间 3 开始加工。工作 4 在时间 5 提交,如果工作 1 在时间 0 开始加工,则工作 4 仍然可以在时间 5 开始加工(见图 3.5)。

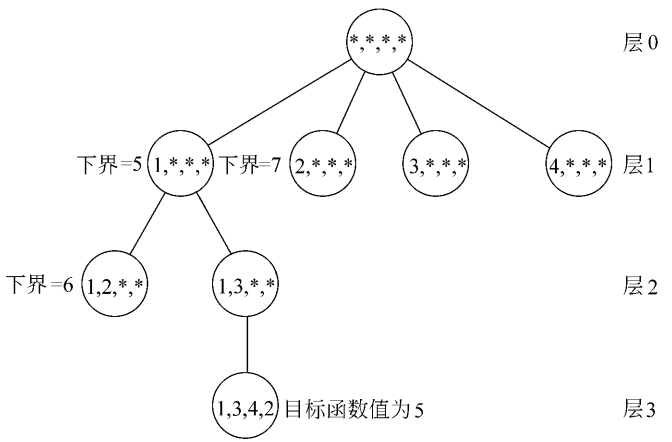

图 3.5　例 3.2.5 的分支定界程序

根据中断 EDD 规则计算(1,*,*,*)点的一个下界得到一个调度,工作 3 在时间区间[4,5]内加工,工作 4 在时间区间[5,10]内,工作 3(再一次)在时间区间[10,15]内,工作 2 在时间区间[15,17]内。这个调度的 $L_{\max}$ 为(1,*,*,*)点提供下界,是 5。用相似的方法可以得到点(2,*,*,*)的下界,其值是 7。

考虑第 2 层的点(1,2,*,*),其下界是 6,是由(无中断)调度 1,2,4,3 确定的。继续第 2 层的点(1,3,*,*),其下界是 5,是由(无中断)调度 1,3,4,2 确定的。从(1,*,

，)的下界是 5 而(2，*，*，*)的下界大于 5 的事实，得出调度 1,3,4,2 必是最优的。

$1 \| r_j$，prec $| L_{\max}$可以用相似的方法处理。这个问题，从枚举的观点看，比没有优先约束的问题简单，因为利用优先约束可以立即排除某些调度。

3.3 滞后工作的数量

另一个和工期相关的目标是 $\sum U_j$。这个目标也许首先是人为设计出来的，并没有实际的考虑。然而，在现实世界中，它是经常用来监测和评估管理者的绩效指标。它等价于按时发货的百分比。

$1 \| \sum U_j$ 的一个最优调度采取一个能够满足工期的工作集合的形式，它们首先进行调度，然后是那些剩下的不能满足工期的工作集合，它们最后进行调度。按照前面章节的结论，第 1 个工作集合必须按照 EDD 进行调度，以确保 $L_{\max}$是负值；第 2 个工作集合里的调度顺序是无关紧要的。问题 $1 \| \sum U_j$ 很容易用前向算法解决。集合 J 再次用来表示已经调度了的工作集合。集合 J^{d}表示已经考虑调度但被删掉的工作集合，因为它们在最优调度里不满足它们的工期。集合 J^{c}表示还没有考虑调度的工作集合。

算法 3.3.1(最小化滞后工作的数量)

步骤 1 设 $J=\varnothing$，$J^{\mathrm{d}}=\varnothing$，$J^{\mathrm{c}}=\{1,2,\cdots,n\}$。

步骤 2 使 j^* 表示满足 $d_{j^*}=\min\limits_{j\in J^{\mathrm{c}}}(d_j)$ 的工作。将 j^* 加到 J，从 J^{c}中删除 j^*，到步骤 3。

步骤 3 如果 $\sum\limits_{j\in J} p_j \leqslant d_{j^*}$，到步骤 4；否则，令 k^* 表示满足

$$p_{k^*}=\max_{j\in J}(p_j)$$

的工作。从 J 中删除 k^*，将 k^* 加到 J^{d}。

步骤 4 如果 $J^{\mathrm{c}}=\varnothing$，停止；否则跳转到步骤 2。

总之，该算法可以描述如下。工作按照工期的升序添加到按时完成工作的集合中。如果加入的工作 j^* 导致该项工作推迟完成，那么已经调度的最长加工时间的工作，比如说 k^*，标记为推迟并被删除。因为该算法是按照工作的工期对工作进行排序的，所以最坏的计算时间是简单排序的时间，即是 $O(n\log(n))$。

定理 3.3.2 对 $1 \| \sum U_j$，算法 3.3.1 得到一个最优调度。

证明：不失一般性，假设 $d_1 \leqslant d_2 \leqslant \cdots \leqslant d_n$。让 J_k 表示满足如下两个条件的工作子集：

(i) 在工作$\{1,2,\cdots,k\}$中，有最大数量的工作，设为 N_k，在它们的工期前完成。