第二十四届挑战杯参赛作品

高端制造业批量处理问题的优化方案研究

史忠顺 黄泽文 梅英男 吴易繁

工业工程与管理系

北京大学工学院

2016 年 3 月 25 日

目录

1. 问题背景

高端制造业是一个国家或地区工业化过程中的必然产物。迄今为止，学术界对高端制造业还缺乏统一的界定，更缺乏一个明确的统计分类标准。很多学者认为，高端制造业的概念应该从行业和产业链环节两个角度来进行界定。从行业的角度讲，高端制造业是指制造业中新出现的具有高技术含量、高附加值强竞争力的行业；从所处产业链的环节上讲，高端制造业处于某个产业链的高端环节，例如半导体制造、火箭卫星厂等。

在这类企业中，生产调度是最重要也是经常遇到的问题。一个好的调度不仅可以缩短生产周期，而且可以减少在制品库存以及提高机器利用率。在以往的研究中，生产调度主要是集中在单机上，单机一次只能加工一个订单，例如经典的 Flow Shop 和 Job Shop 排程。但在实际生产中，我们经常会遇到批处理机器，批处理机器一次可以同时加工多个订单。例如图 1.1 中，在半导体生产线上不同的加工设备具有不同的加工方式，主要包括以下三种：单片加工、串行批量加工、单卡并行批量加工和多卡并行批量加工。单片加工的设备每次只能加工一片硅片，如快速热处理设备、光刻机、刻蚀机；串行批量加工的设备是按片加工的，但一次可装入多片硅片，如离子注入机；并行批量加工的设备则可一次加工一卡或多卡硅片，如氧化炉、扩散炉等。上面提到的这三种加工方式中，单片加工设备属于单机加工设备，串行批量加工设备和并行批量加工设备都属于批加工设备；在钢铁行业，轧钢、烧结等工序属于批处理；在半导体封装测试环节，同样也有熔炉热处理工序，这些工序也属于批处理。

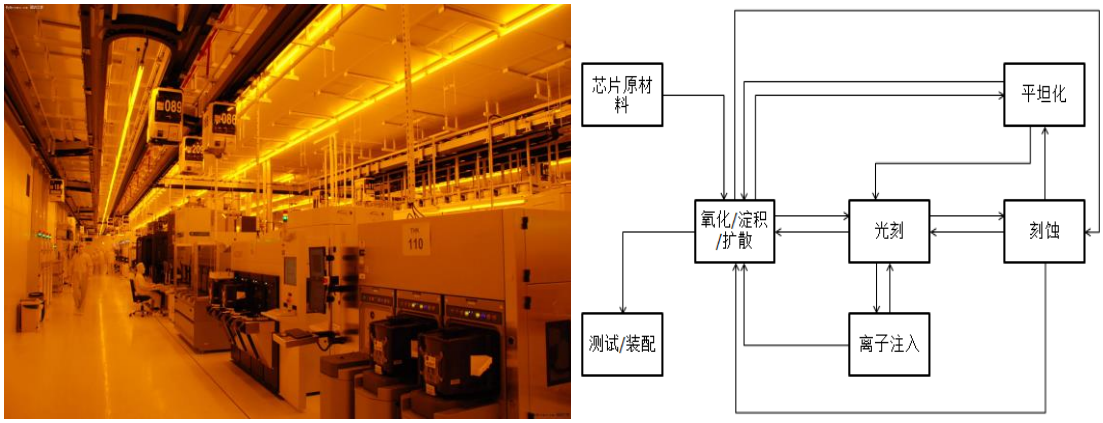

图 1.1 半导体加工流程示意图

批处理机在许多工程领域中都有着广泛的应用，但对于这些领域来说批处理机往往意味着瓶颈，因为批处理机的工作即耗时又昂贵，这对于工作流水线的优化有着不利的影响。同时混合流水生产线在半导体生产企业中也是十分常见的，在这样的生产线中批处理机的上游或下游工作往往并非由批处理机而是由单机处理的，这使得该类工厂中的生产调度问题显得极为棘手。

本文主要研究高端制造业中的批处理问题，我们给出相应问题的混合整数规划模型，在这些模型的基础上提出该问题的下界，设计逼近式算法，最后通过数值试验证明算法的有效性。

2. 研究现状及必要性

本章对相关研究现状进行总结，同时提出本文研究的必要性。本文所研究的批量处理机处于上游而单机处于下游的二阶段流水作业调度问题是二阶段流水作业研究的一个重要分支。为了问题描述的简化，下文我们以$\beta \rightarrow \delta|Z$代表此问题，其中$\beta$代表批处理机而$\delta$代表单机。Z 则代表了问题的目标。

Ahmadi et al. [1] 研究了$\beta \rightarrow \delta$问题并就工作周期（C_{max}）最小化这一目标给出了 Full Batch-dealing LPT 算法。他还就总完工时间（$\sum C_j$）最小化的目标给出了 Full Batch-dealing SPT 算法。同时他还拓展了$\beta \rightarrow \delta$问题，为其加入了不相容工作族的条件并以工作周期（C_{max}）最小化为目标提出了 LPT-Johnson 规则。在 Ahmadi et al.工作的基础上，Hoogeveen and Velde [2] 利用了位置完成时间的概念并以$\beta \rightarrow \delta|\sum C_j$为目标提出了一种时间复杂度为 $O(n \log n)$的算法并给出了其一个拉格朗日下界。Kim and Kim [3] 利用信息不变性原则为解决$\beta \rightarrow \delta|\sum C_j$提出了一种以 GA 为基础的新的方案。Su et al. [4] 则考虑了第二阶段有限等待时间这一限制就$\beta \rightarrow \delta|C_{max}$问题提出了一个启发式算法及混合线性整数规划模型。Su and Chen [5] 则就不等同工作尺寸限制条件下的$\beta \rightarrow \delta|C_{max}$提出了一种启发式算法和一种分支定界算法。Yao et al. [6] 考虑了动态工作到达及不相容工作族、有限等待时间限制条件下的$\beta \rightarrow \delta|C_{max}$问题。他们都为我们解决这些问题提供了诸多

启发。

所有上述研究都将缓冲区视为无限大，然而实际中由于空间及储存设施的限制缓冲区的大小并不如同设想的那般理想化。目前考虑这一限制的研究还比较少。Gong and Tang [7] 以及 Gong et al. [8] 在考虑C_{max}和总堵塞时间的目标下对$\beta \rightarrow \delta$问题进行了研究。若单机仍处于繁忙状态，批处理机中已完工的工作将继续停留在批处理机中。他们提出了几种近似算法。Su et al. [9] 研究了有限缓冲区限制条件下的$\beta \rightarrow \delta|C_{max}$问题并给出了一种思路以及一个分支定界程序以进行基准化分析。Fu et al. [10] 就不相容工作族及有限缓冲区条件下的$\beta \rightarrow \delta|\bar{C}$ 问题进行了研究，其中$\bar{C}$代表平均完工时间。他们提出了一个下界、两种启发式算法及一种 DE 算法。他们假设单机上的工作处理顺序与批处理器上的完全相同。同时他们还就缓冲区设定了第一个进入第一个退出（FIFO）的规则。当没有不相容的工作族存在时，流水型排序比其余安排更有效率[1]。但这一假设在不相容工作族存在时并不适用。

从该问题的研究现状，我们得出$\beta \rightarrow \delta|Z$问题的研究很多方面存在着空白，例如带有不相容工作族的$\beta \rightarrow \delta|\sum w_j C_j$问题和带有不相容工作族及有限缓冲区限制条件的$\beta \rightarrow \delta|\sum C_j$问题，而且这些空白在现实生产中更普遍，更接近实际情况，在研究中具有很大的实际意义和必要性。本文研究的出发点就是填补这些空白。正如 Ahmadi et al. [1] 所述的，$\beta \rightarrow \delta|\sum C_j$问题是 NP-Complete 问题，本文所研究的带有不相容工作族的$\beta \rightarrow \delta|\sum w_j C_j$问题和带有不相容工作族及有限缓冲区限制条件的$\beta \rightarrow \delta|\sum C_j$问题很显然也是 NP-Complete 问题。接下来我们在第三章讨论带有不相容工作族的$\beta \rightarrow \delta|\sum w_j C_j$问题，在第四章讨论带有不相容工作族及有限缓冲区限制条件的$\beta \rightarrow \delta|\sum C_j$问题

3. 带有不相容工作族的$\beta \rightarrow \delta|\sum w_j C_j$问题

本节研究两阶段流水车间调度问题的总加权完成时间最小化问题。该流水线

由两部分构成：上游的批量处理机器与下游的单机。批量处理机器有完全相同的批量处理时间与固定的容量。批量处理机器可同时容纳多至容量上限的工件。当处理开始后，处理器不能中途停止。来自不同工件族的工件不能被分在同一个批次内。工件在被批量处理器释放之后，可以以任意顺序在单机上进行处理。

3.1 问题模型

在这部分当中，我们首先给出 $\beta\rightarrow\delta$，$incompat|\sum w_jC_j$ 问题的满批次性，其中 incompat 指不相容工作族，随后在满批次性的基础上将问题用公式表示为一个混合整数线性规划模型。为了更好地表述，假设、参数与决策变量说明如下：

假设：

1) 在调度开始时所有工件都准备就绪，即所有工件都可以在调度开始时在批量处理器上处理。
2) 每一个工件族中的工件数量都是批量生产机器容量的整数倍
3) 所有的工件尺寸相同
4) 所有的批次在批处理机器上拥有相同的处理时间
5) 优先权在批量处理器与单机上都不被允许
6) 两个处理器间的缓冲区是无限的
7) 工件可以以任意顺序在单机上进行处理

参数：

N 工件数量 N=｛1,2, … ,n｝;

F 工件族数量 F=｛1,2, … ,m｝;

n_k 工件族 k 中的工件数量

B 批次的数量 B=｛1,2, … ,|B|｝;

F_k 工件族 k 中的工件数量

p_i 工件 i 在单机上的处理时间

t 批量处理时间

b 批次容量

M 一个足够大的正数

w_i 工件 i 完成时间的权重

变量：

x_{il} $x_{il}=1$ 当工件 i 被分配到批次 l，否则$x_{il}=0$；

y_{kl} $y_{kl}=1$ 当批次 l 包含工件族 k 的工件，否则$y_{kl}=0$；

u_{ij} $u_{ij}=1$ 当工件 i 在工件 j 之前在单机上进行处理，否则$u_{ij}=0$；

C_i 工件 i 的完成时间

Uzsoy [11]证明了单批量处理器的满批次性在正则测度下单个工件的完成时间是单调非减的。因此，根据 [11]，满批次性可以在如下引理中直接延伸至这个两阶段流水车间问题。

引理 3.1：存在一个 $\beta\rightarrow\delta$，$incompat|\sum w_jC_j$调度问题的最有解，除了每个工件族的最后一个批次之外，所有批次都是满批次。

我们假设在任何工件族 k 当中，其工件数量n_k是批次容量 b 的整数倍。这意味着工件总数$n=\sum_k n_k$=αb，其中 α 代表满批次性下的批次数量。因此，基于以上批注与决策变量，该问题可以用公式表示为如下形式：

BDMI： $\min \sum_{i\in N} w_jC_j$ （1）

$$\text{s.t. } \sum_{l\in B} x_{il}=1, \quad \forall i\in N, \quad （2）$$

$$\sum_{i\in N} x_{il}=b, \quad \forall l\in B, \quad （3）$$

$$\sum_{k\in F} y_{kl}=b, \quad \forall l\in B, \quad （4）$$

$$\sum_{l\in B} y_{kl}=\frac{n_k}{b}, \quad \forall k\in F, \quad （5）$$

$$x_{il}\le y_{kl}, \quad \forall i\in F, l\in B, \quad （6）$$

$$u_{ij}+u_{ji}=1, \quad \forall i,j\in N, i<j, \quad （7）$$

$$C_i-p_i\ge C_j-Mu_{ij}, \quad \forall i,j\in N, i\ne j, \quad （8）$$

$$C_i-p_i\ge t\sum_{l\in B} lx_{il}, \quad \forall i\in N, \quad （9）$$

$$u_{ij}\in\{0,1\}, \quad \forall i,j\in N, i\ne j, \quad （10）$$

$$y_{kl}\in\{0,1\}, \quad \forall k\in F, l\in B, \quad （11）$$

$$x_{il}\in\{0,1\}, \quad \forall i\in N, l\in B, \quad （12）$$

目标函数（1）是最小化总加权完成时间。约束（2）确保一个工件只会被分

配到一个批次。约束（3）保证所有批次都是满的。约束（4）确保每个批次内只有同一个工件族的工件。约束（5）确保 k 工件族的工件应该被分配到n_k/b批次。约束（6）证明只有当批次 l 当中包含工件族 k 的工件时，工件族 k 的工件才可以被分配到该批次当中。约束（7）确保工件在单机上的处理具有优先级。约束（8）保证了只有当一个工件在单机上处理完毕时，其后的工件才可以在单机上开始处理。约束（9）确保了只有当工件 i 所在的批次在批量处理器上完成了处理之后，工件 i 才可以在单机上进行处理。约束（10），（11）与（12）为变量的范围。在此，我们可以设置 B=｛1,2, … ,α|｝以及$M = \alpha t + \sum_{i\in N} p_i$

值得注意的是，通过在任何工件族当中添加额外的零权重与零处理时间的虚设工件，可以很容易地将公式延伸到工件族工件数量不为批次容量整数倍的情况。我们使用 BDMI 等式定制满批次数据来进行计算实验。对应的计算结果将在 3.4 部分中展示。

3.2 问题下界

在本节中，我们发展了针对 $\beta\rightarrow\delta$，$incompat|\sum w_jC_j$的两种下界确定方法，这两种下界确定的方法可以被用于分析我们所提出算法所得解的质量。接下来，我们介绍目前广泛被运用的再分配不等式，而相应的证明我们则省略。

引理 3.2：当$a_1 \le \cdots \le a_n$与$b_1 \le \cdots \le b_n$成立时，我们可以得到

$$a_n b_1 + \cdots + a_1 b_n \le a_{\sigma(1)} b_1 + \cdots + a_{\sigma(n)} b_n \le a_1 b_1 + \cdots + a_n b_n$$

其中$a_{\sigma(1)}, \ldots, a_{\sigma(n)}$为$a_1, \ldots, a_n$的任意顺序。

考虑以下情形：所有工件在批量处理后都可以立即在单机上进行后续处理，即，单机的等待时间可以忽略。据此，我们在提法 1 中推导第一个下界 LB_1。

推论 3.1:将权重 w 按降序排列，将处理时间按升序排列，则 $LB_1=\sum_{j=1}^{n} \overline{w}_j\left(\left\lceil \frac{j}{b} \right\rceil t + \overline{p}_j\right)$是问题 $\beta\rightarrow\delta$，$incompat|\sum w_jC_j$的一个下界。

证明：设（x^*, y^*, u^*, C^*）是问题的一个最优解。对于每个工件 j,记其所需完成时间为C_j^*，令w_j^*和p_j^*分别为其权重和处理时间。显然，每个C_j^*包含三个部分：批量处理器上的完成时间 $t\sum_{l\in B} x_{jl}^*$ 在缓冲区的等待时间$\Delta_j^* \ge 0$以及单机处理时

间p_j^*。因此，为了使目标函数$\sum_{j=1}^{n} w_j^* C_j^*$最小化，我们需要以下不等式：

$$\sum_{j=1}^{n} w_j^* C_j^* = \sum_{j=1}^{n} w_j^* \left(t\sum_{l\in B} x_{jl}^* + \Delta_j^* + p_j^*\right)$$
$$\geq \sum_{j=1}^{n} w_j^* \left(t\sum_{l\in B} x_{jl}^* + p_j^*\right)$$
$$\geq \sum_{j=1}^{n} \bar{w}_j\left(\left\lceil\frac{j}{b}\right\rceil t + \bar{p}_j\right) = LB_1$$

由引理 3.2，最后一个不等式成立，证明完成。

接着，我们从一种不同的角度来推导另一个下界，记为 LB_2.考虑以下情况：在话费时间 t 处理完第一批工件后，所有的工件都从批量处理器中取出，从而这个问题就变为了释放时间全部为 t 的单机时间规划问题。这类问题可以通过 WSPT 法则，即 Smith's rule,来取得最优解。

推论 3.2：将所有工件按p_j/w_j进行增序排列，$\forall j \in N$, 可得到一个序列：$\hat{p}_1\hat{w}_1 \leq \hat{p}_2\hat{w}_2 \leq \cdots \leq \hat{p}_n\hat{w}_n$，则 $L\ B_2 = \hat{w}_j(t + \sum_{i=1}^{j}\hat{p}_i)$ 为 $\beta\rightarrow\delta$，*incompat*|$\sum w_jC_j$的一个下界。

证明：记（x^*, y^*, u^*, C^*）是问题的一个最优解，对于每个工件 j, 记其所需完成时间为C_j^*，令w_j^*和p_j^*分别为其权重和处理时间。为了实现$\sum_{j=1}^{n} w_j^* C_j^*$的最优化，我们有以下不等式：

$$\sum_{j=1}^{n} w_j^* C_j^* \geq \sum_{j=1}^{n} w_j^* \left(t + \sum_{i=1}^{j} p_i^*\right)$$

$$\geq \sum_{j=1}^{n} \hat{w}_j\, t + \sum_{j=1}^{n} w_j^* \sum_{i=1}^{j} p_i^*$$

$$\geq \sum_{j=1}^{n} \hat{w}_j\, t + \sum_{j=1}^{n} \hat{w}_j \sum_{i=1}^{j} \hat{p}_i = LB_2$$

最后一个不等式的成立由 smith's rule 保证

LB_1和 LB_2在封闭形式下的时间复杂度均为 O(nlogn)，均可以被较快地算出。由于 推导过程的不同，它们的适用性高度依赖于批量处理时间 t。当批量处理时间 t 足够小的时候，LB_2相比 LB_1更接近实际。当 t 足够小，批量处理的时间对于目标的影响就很小，从而工件也就更可能被较早地释放入缓冲器。另一方面，如果 t 足够大，那么单机的等待时间就会显得很少，从而 LB_1就会更精确。除此之

外，BDMI 的线性规划松弛从直觉上来说也是问题的一个下界。因此，这两种下界，以及线性规划松弛，被用于评估我们算法的表现。

3.3 算法设计

在本节中，我们提出两种近似算法，并提供了它们的最坏情况分析。两种算法都包含了两个阶段，其一是确定如何组织批次以及在处理器上的批次顺序。第二阶段则是确定工件在单机上的处理顺序。

3.3.1 GRWC-WSPT 算法

Uzsoy [11] 提出了一种用于求解一个批次处理机器上不兼容工件组的最小加权时间工序（GRWC）的贪心算法。我们将批次处理器上的 GRWC 过程、单机上的 WSPT 规则以及一些其他的有用信息组合起来并设计了的第一个启发式算法，我们将其记作 GRWC-WSPT。下表描述了其流程：

GRWC-WSPT：

第一步：对第 k 组工件麻将工件按权重降序排列

第二步：用贪心法将每个工件组中的第一个工件选出并组成一个批次

第三步：计算每个批次的总权重W_l。按W_l的降序来排列批次。

第四步：当有可操作的单机时，选取当前缓冲期中p_i/w_i值最小的工件进行加工。重复该过程直到所有工件都加工完成。反回该规划π_1极其函数值 Obj(π_1)。

第五步：让单机处于待机状态，直到所有的批次都已释放，之后再按 p/w 的增序在单机上进行处理。返回该规划π_2极其函数值 Obj(π_2)。

第六部：在两者见取最小值

该算法的时间复杂度是 O(nlogn)。接下来，我们给出该问题封闭形式的一个上界，记为 UB。UB 可被用于分析所提算法的最坏情况。

引理 3.3：令 UB=$\alpha t\sum_{j=1}^{n} w_j + \Gamma_{WSPT}$，则 UB 为 $\beta\rightarrow\delta$，$incompat|\sum w_j C_j$的一个上界，其中 α 为批次总数量，$\Gamma_{WSPT} = \sum_{j=1}^{n} \hat{w}_j \sum_{i=1}^{j} \hat{p}_i$由 WSPT 对所有工件给出。

证明：我们直接考虑最坏情况，即，不论我们如何在第一阶段中组批次、安

排顺序，单机都要在所有工件都被释放入缓冲期之后才能开始处理工件。在这种情况下，第 i 批工件中的工件的释放时间可以被视为$it + (\alpha - i) = \alpha\, t$，该公式意味着对单机而言所有的工件都有着相同的释放时间αt，我们将上述的工序记为H_0。对H_0我们可以找到一个可行解，我们很容易确认其目标值为$Obj(H_0)= \alpha t \sum_{j=1}^{n} w_j + \Gamma_{WSPT} = UB$. 证明完成。

基于引理 3，在以下的推论 3 中我们给出了 GRWC-WSPT 算法的最坏情况估计。

推论 3.3：GRWC-WSPT 是 $\beta\rightarrow\delta$，$incompat|\sum w_j C_j$问题的 α 近似算法，其中 α 为批次总数。

证明：考虑引理三证明中提出的工序H_0出现在算法的第五步，我们有 Obj(GRWC-WSPT)≤ UB，基于第四节提出的LB_1 LB_2，以及引理 3 给出的 UB，不难验证$LB_1 \geq t\sum_{j=1}^{n} w_j$以及$UB - LB_2 = \alpha t \sum_{j=1}^{n} w_j - \sum_{j=1}^{n} \widehat{w}_j t$。因此，以下不等式成立，其中 Obj(OPT)代指最小目标值。

$$\begin{aligned}\frac{Obj(GRWC-WSPT)}{Obj(OPT)} &\leq 1 + \frac{Obj(GRWC-WSPT) - LB_2}{LB_1} \\ &\leq 1 + \frac{UB - LB_2}{LB_1} \\ &= 1 + \frac{\alpha t \sum_{j=1}^{n} w_j - \sum_{j=1}^{n} \widehat{w}_j t}{LB_1} \\ &\leq 1 + \frac{\alpha t \sum_{j=1}^{n} w_j - t\sum_{j=1}^{n} w_j}{t\sum_{j=1}^{n} w_j} = \alpha\end{aligned}$$

这意味着 GRWC-WSPT 是 α 近似算法，证明完成。

3.3.2 LP-WSPT 算法

在本节中，我们希望能运用 LP-relaxation 给我们提供的信息能被用于获取一个高质量的解，因此，我们发展了一个基于 LP 的算法，我们称其为 LP-WPST。LP-WPST 的流程如下：

第一步：解 BDMI 的线性规划弛豫问题，并获取每个工件的弛豫完成时间 c。

第二步：在第 k 个工件组中，按 c 的增序排列工件

第三步：用贪心法从每个工件组中选取第一个工件组成批次

第四步：计算每个批次的c_l。这里的c_l是第 l 个批次中所有工件的 c 的综合。将批次按c_l增序排列。

第五步：当单机可供操作时，选取当前缓冲器中p_i/w_i值最小的工件进行加工。重复此过程直至所有工件都被加工完毕。返回该计划表π_1及其目标值 Obj(π_1)

第六步：重定义$c_j=c_j/w_j$，重复步骤 2-5，返回计划表π_2及相依目标值 Obj(π_2)

第七步：让单机处于待机状态，直到所有的批次都已释放，之后再按p_i/w_i的增序在单机上进行处理。返回该规划π_3极其函数值 Obj(π_3)。

第八步：从三个规划中选取目标值最小者

在以下的提法 4 中，我们给出 LP-WSPT 的最坏情况分析。

推论 3.4：LP-WSPT 是 $\beta\rightarrow\delta$，$incompat|\sum w_j C_j$问题的 α 近似算法，其中 α 为批次总数。

证明：考虑到引理 3 中提出的 H0 被整合入算法的第七步，同推论 3 相似，我们不难确认：

$$\frac{\mathrm{Obj(LP-WSPT)}}{\mathrm{Obj(OPT)}} \le 1+\frac{\mathrm{Obj(LP-WSPT)}-\mathrm{LB_1}}{\mathrm{LB_2}} \le 1+\frac{\mathrm{UB}-\mathrm{LB_1}}{\mathrm{LB_2}} \le \alpha$$

因此，LP-WSPT 算法是一个α近似算法，证明完成。

注意到线性规划所用的椭圆法和内点法的时间代价都是多项式级别，LP-WSPT 算法的时间复杂度也可能是多项式级别的。然而，考虑到对大多数情况单纯形法均有 的表现，在本文中我们选用了 Cplex 中的“LP solver”来解决 BDMI 的 LP-relaxation 问题。

3.4 数值实验

本节进行了一些计算试验来检验所提出的算法的表现。所有的试验均在 64 位的 Windows7 系统上运行，硬件配置为 Intel Core3.3GHZ 的 CPU, RAM 4.0GB。算法用 C++编写，用 VS2010 运行。BDMI 的数学公式求解由 Cplex12.5 进行（默认的求解时间上限为 3600s）

3.4.1 案例生成

我们用 n-m-b 组合来表述案例，其中 n 表示工件数，m 表示工件组数，不表示批次容量。如同参考文献[1]和[12]一样，我们用以下方法产生案例：

对于每个组合“n-m-b”，设置$n_j = b\left\lfloor \frac{n}{mb} \right\rfloor, j = 1, \ldots, m-1$,和$n_m = n - \sum_{j=1}^{m-1} n_j$。对于每个工件 i，其权重由 1-2 之间的均匀分布给出，处理时间p_j和单机则由 1-10 的均匀分布给出。批次处理时间 t 被设置为$\frac{n}{b}\sum_{i=1}^{n} p_i$。这样的设计可以有效避免目标值由批次处理时间或是单机处理时间主导的情况。令$G_1G_2G_3$表示小规模组，中规模组，大规模组。每组的构成如下：

$$G_1：n = \{8,12,16,20\}, m = \{2,4\}, b = \{2,4\}$$

$$G_2：n = \{100,200,300,400\}, m = \{2,4,10\}, b = \{10,20\}$$

$$G_3：n = \{800,1000\}, m = \{4,10\}, b = \{20,50\}$$

我们假如额外要求$\frac{n}{mb} \geq 1$来去掉一些简单组合，然后对于每个组合都生成了 10 个案例。因此我们一共使用了 450 个例子来测试所提公式与算法的下过。

3.4.2 数值结果与分析

在这小节中，我们展示了$G_1G_2G_3$组的数值结果。我们主要在两个方面比较算法：解的准确性与运行时间。为了简便起见，以下符号被用于相关表格中：

Obj：相应算法的平均目标值

Time：相应算法的平均求解时间

Ratio：H 的目标值雨 LB 的比值

Gap_1：GRWC-WSPT 与 BDMI 结果的平均差值

Gap_2：LP-WSPT 与 BDMI 结果之间的平均差值

Gap_1与Gap_2的计算方法如下：

$$Gap_{1(2)} = \frac{Obj(H) - Obj(BDMI)}{Obj(H)} \times 100.$$

显然，Gap<0 意味着响应的启发式算法比 BDMI 的表现更好，$Gap_{1(2)}$>0 则

意味着 BDMI 表现更好，$Gap_{1(2)}$=0 则意味着两者的表现相同。

表 3.1 给出了G_1组 BDMI 的数值结果。Lp-r 栏表示 BDMI 的线性规划 ，Obj 栏表示一小时内 Cplex 解得的 BDMI 的平均最佳目标函数值。Time 栏表示求解所需平均时间。Gap_0表示 Cplex 在一小时求解中最后一步的 BDMI 平均差值。如表中所述的一样的，Lp-relaxation 更为精密，由此法可在个人电脑上在一小时内解 12 个工件的问题。当工件数目增加至 20 时，没有任何案例可以在一小时内求解，对于所有组合，Gap_0的最大值是 6.64%。

表 3.1 算例 G1 的 BDMI 模型求解结果

n-m-b	*LP-r*	*Obj*	*Time*(sec)	Gap_0(%)	*Num*
8-2-2	347	373	0.23	0.00	10
8-2-4	477	551	0.16	0.00	10
8-4-2	402	435	0.13	0.00	10
12-2-2	739	789	506.43	0.00	10
12-2-4	827	943	23.03	0.00	10
12-4-2	778	830	6.17	0.00	10
16-2-2	1307	1380	3600	2.80	0
16-2-4	1271	1410	1478.29	3.56	4
16-4-2	1173	1243	2127.25	2.01	4
16-4-4	1434	1599	15.59	0.00	10
20-2-2	1729	1800	3600	2.96	0
20-2-4	1979	2185	3600	6.64	0
20-4-2	1870	1969	3600	3.42	0
20-4-4	2077	2290	3600	2.91	0

表 3.2 展示了两种算法的结果的对比以及G_1组中由 Cplex 解得的 BDMI 结果。每个组合的结果都是 10 个案例的平均。值得注意的是对于每个例子，我们都计算了 LB_1, LB_2 以及 Lp-r 中的最大值，而 LB 栏中的值则是这 10 个值的平均值。显然，$Gap_1 Gap_2$的平均值都是正值，即 BDMI 的解的质量高于我们所提的两种算法。然而，我们所提算法的求解时间要显著低于 BDMI，而Gap_1和Gap_2的最大均值仅为 3.49%。另一方面，GRWC-WSPT 的最大平均比值是 1.1256，而 LP-WSPT 则是比其稍好（除了组合 16-2-4）。

表 3.2 算例 G1 的均值对比结果

		GRWC-WSPT				*LP-WSPT*				*BDMI*	
n-m-b	*LB*	*Obj*	*Time*(sec)	Gap_1(%)	*Ratio*	*Obj*	*Time*(sec)	Gap_2(%)	*Ratio*	*Obj*	*Time*(sec)
8-2-2	347	383	0.00	2.37	1.1017	383	0.03	2.37	1.1017	373	0.23
8-2-4	513	557	0.00	0.97	1.0876	557	0.01	0.97	1.0876	551	0.16
8-4-2	406	438	0.00	0.77	1.0788	438	0.02	0.77	1.0788	435	0.13
12-2-2	743	810	0.00	2.60	1.0912	805	0.01	1.94	1.0838	789	506.43
12-2-4	862	955	0.00	1.26	1.1111	955	0.02	1.26	1.1111	943	23.03
12-4-2	778	852	0.00	2.77	1.0980	852	0.02	2.77	1.0980	830	6.17
16-2-2	1307	1430	0.00	3.49	1.0948	1430	0.03	3.44	1.0939	1380	3600
16-2-4	1278	1438	0.00	1.93	1.1253	1439	0.03	1.96	1.1256	1410	2751.34
16-4-2	1173	1280	0.00	2.85	1.0913	1270	0.03	2.13	1.0831	1243	3600
16-4-4	1438	1608	0.00	0.50	1.1178	1608	0.02	0.50	1.1178	1599	15.59
20-2-2	1729	1842	0.00	2.23	1.0649	1835	0.03	1.92	1.0615	1800	3600
20-2-4	1999	2216	0.00	1.44	1.1099	2207	0.03	1.06	1.1055	2185	3600
20-4-2	1870	2019	0.00	2.55	1.0809	2016	0.04	2.37	1.0790	1969	3600
20-4-4	2079	2322	0.00	1.41	1.1167	2322	0.03	1.41	1.1167	2290	3600