文章编号: 1001-0920 (2015) 12-2145-08 DOI: 10.13195/j.kzyjc.2014.1704

不同容量平行机下差异工件尺寸的批调度算法

贾兆红, 李晓浩, 温婷婷, 李龙澍

(1. 安徽大学 计算智能与信号处理教育部重点实验室, 合肥 230039;
2. 安徽大学 计算机科学与技术学院, 合肥 230601)

摘 要: 在容量不同的平行批处理机环境下, 针对工件带有不同尺寸和机器适用限制的最小化制造跨度的批调度问题, 提出一种有效的蚁群优化算法. 该算法基于解的浪费空间定义启发式信息, 针对机器容量约束提出两种用于构建解的候选集, 从而有效缩小搜索空间, 并引入局部优化方法提高解的质量. 仿真实验结果表明, 所提出算法具有较好的性能, 并且优于已有的其他算法.

关键词: 平行批处理机; 差异尺寸工件; 不同机器容量; 机器适用限制; 蚁群优化算法

中图分类号: TP301 **文献标志码:** A

Algorithms for scheduling on parallel batch machines with non-identical capacities and non-identical job sizes

JIA Zhao-hong, *LI Xiao-hao*, *WEN Ting-ting*, *LI Long-shu*

(1. Key Lab of Intelligent Computing and Signal Processing of Ministry of Education, Anhui University, Hefei 230039, China; 2. School of Computer Science and Technology, Anhui University, Hefei 230601, China. Correspondent: JIA Zhao-hong, E-mail: jiazhaohong001@163.com)

Abstract: To address the problem of minimizing the makespan for parallel batch process machines with non-identical machine capacities, non-identical jobs sizes and machine eligibility restriction, an effective ant colony optimization(ACO) algorithm is proposed. Based on the wasted space of the solution, the heuristic information is defined. Meanwhile, two candidate sets for constructing the solution are used to narrow the search space. To further enhance the solution quality, a local optimization approach is incorporated. The simulation results show that the proposed algorithm outperforms the other available algorithms.

Keywords: parallel batch processing machines; non-identical job sizes; non-identical machine capacities; machine eligibility restriction; ant colony optimization algorithm

0 引 言

批处理机 (BPM) 调度 (简称批调度) 是广泛存在于工业制造、港口货物装卸、船闸调度等领域的一类新型调度问题. 针对批调度问题的研究最初源于半导体制造业, 在半导体生产过程的最后测试阶段, 所有芯片 (作业) 被放入加热板 (批), 再将加热板放入烤炉 (批机器) 中高温灼烧一段时间, 该工序耗时较其他工序长, 目前已成为半导体加工的瓶颈. 因此, 提高 BPM 的使用率, 对于提高芯片制造业的生产率具有重要的理论和实践意义. 由于采用批处理机同时对一组作业进行加工, 批调度中的作业分组后以批的形式在机器上加工. 批的加工时间可以被定义为批中最大工件加工时间或是批中所有工件加工时间之和, 根据这两种情况可以将批调度问题分为平行和串行两种, 本文研究的是平行批调度问题. 在实际应用中, 批调度的作业尺寸往往不同, 且批中所有工件的尺寸之和不能超过加工该批的机器容量.

目前, 关于批调度的研究主要集中在单机问题上. Uzsoy[1]最先对作业尺寸不同的单机批调度问题 (SSBN) 进行了研究, 证明了最小化制造跨度的 SSBN 是 NP 难题, 提出了一种启发式算法来对其进行求解; Dupont 等[2]提出了另外两种启发式算法; Melouk 等[3]

收稿日期: 2014-11-07; **修回日期:** 2015-04-22.

基金项目: 国家自然科学基金项目(71171184, 61202227); 教育部人文科学研究项目(15YJC630041); 安徽省教育厅自然科学研究项目(KJ2015A062); 安徽大学自然科学项目(33050044).

作者简介: 贾兆红(1976−), 女, 副教授, 博士, 从事商务智能、生产调度算法等研究; 李晓浩(1990−), 男, 硕士生, 从事智能调度算法的研究.

和 Damodaran 等[4]分别采用模拟退火 (SA) 和遗传算法 (GA) 求解了最小化制造跨度的 SSBN, 其中遗传算法的性能较优; 许瑞等[5-6]基于蚁群优化 (ACO) 分别求解了最小化总完工时间的 SSBN 和作业带有到达时间最小化制造跨度的 SSBN. 然而, 在现实应用中, 单台设备问题相对较少, 多数情况下是多台设备并行加工, 这时不仅要对作业进行合理分批, 还要考虑将批分配到不同机器. 针对差异工件的平行批处理机调度问题 (SPBN), Chang 等[7]提出了模拟退火算法对其进行求解, 实验结果表明, SA 的求解性能优于商业软件 CPLEX; Damodaran 等[8]提出了若干启发式算法, 将问题分解为工件分组和批分配至机器两个子问题, 分别采用了 FFLPT 和 BFLPT 分批, 再利用 LPT 和 MultiFit 启发式调度批, 实验结果表明, 在大规模问题上, 所提启发式算法的性能优于 CPLEX, 却与 SA 相当; Shao 等[9]将神经网络应用于该问题, 通过与文献 [8] 中启发式算法进行对比, 验证了该方法的优越性; Chen 等[10]针对工件到达的 SPBN, 分别设计 ACO 和 GA 算法进行求解; 此外, 李小林等[11]还针对同类机环境下的 SPBN 问题进行了研究.

上述研究均是针对容量相同的平行批处理机展开的, 而在现实生产环境中, 由于作业性质和设备性能不同, 不同类型 (如容量不同) 的机器常常同时出现在生产线上. 针对机器容量不同的 SPBN 问题, Xu 等[12]提出了一个随机键遗传算法 (RKGA), 结果表明, 在大规模问题上, RKGA 较 CPLEX 得到的结果更好; Wang 等[13]对于带工件到达的容量不同的 SPBN, 分别提出了 GA 和 SA 算法进行求解, 并通过实验验证了所提算法的有效性; Damodaran 等[14]提出了粒子群算法 (PSO) 对该问题进行求解. 实际上, 由于作业尺寸和机器尺寸均不同, 某些大尺寸作业将不能在小容量机器上加工, 而目前在针对机器容量不同的 SPBN 研究中, 绝大部分都是基于单个工件尺寸小于最小机器容量的假设, 仅有文献 [14] 涉及了作业带有机器约束的问题, 因此本文针对机器容量不同且作业带有机器约束的 SPBN 问题进行研究, 给出该问题的一个下界并对其正确性进行证明, 提出采用基于 MultiFit 的启发式算法和基于 ACO 的元启发式算法分别求解问题, 通过仿真实验比较并验证 ACO 算法的性能.

1 问题模型

本文研究机器容量不同且最小化制造跨度的 SPBN 问题. 为了描述该问题, 作如下假设.

1) 工件集合 $J=\{J_1,J_2,\cdots,J_n\}$. 其中: 工件 J_j 的加工时间为 p_j, 尺寸为 s_j.

2) m 台批处理机的集合 $M=\{M_1,M_2,\cdots,M_m\}$. 其中: 机器 M_i 的容量为 S_i, J 中作业分批加工. 满足机器容量约束的任意工件均可放入该机器上的批中, 即任一批 B_b 的尺寸等于该批中所有工件的尺寸之和, 且不超过加工该批的机器 M_i 的容量 S_i, 即 $\sum_{j=1}^{|B_b|}\{s_j|J_j\in B_b\}\leqslant S_i$. 由于部分工件的尺寸大于部分机器的容量, 部分工件只能在部分机器上加工, 即每个工件 J_j 的可加工机器集为 M_j ($M_j\subseteq M$). 机器的加工时间定义为分配在该机器上的所有批的加工时间之和, 机器的完工时间为该机器加工的最后一个批的完工时间.

3) 工件分组后得到的批集合为 B, 根据平行批调度中对批的定义, 任一批 B_b 的加工时间为 $P_b=\max\{p_j|J_j\in B_b\}$. 批一旦开始加工, 则不允许中断.

4) 优化目标为极小化最大完工时间 $C_{\max}$. 参照 Graham 等[15]的三参数表示法, 该问题可表示为 $P_m|p-\text{batch},p_j,s_j,S_m,M_j|C_{\max}$. 基于上述假设, 可建立如下数学模型:

$$\min\ C_{\max}. \tag{1}$$

$$\text{s.t.}\ \sum_{B_b\in B}\sum_{M_i\in M}X_{jbi}=1,\ \forall J_j\in J; \tag{2}$$

$$\sum_{J_j\in J}s_jX_{jbi}\leqslant S_i,\ \forall B_b\in B,\ M_i\in M; \tag{3}$$

$$P_{bi}\geqslant p_jX_{jbi},\ \forall J_j\in J,\ B_b\in B,\ M_i\in M; \tag{4}$$

$$C_{\max}\geqslant\sum_{B_b\in B}P_{bi},\ \forall M_i\in M; \tag{5}$$

$$X_{jbi}=\begin{cases}1,\ J_j\in B_b\ \text{that is processed on}\ M_i;\\ 0,\ \text{otherwise.}\end{cases} \tag{6}$$

其中: 式 (1) 表示问题目标是最小化最大完工时间; 约束条件 (2) 表示任一工件只能被分组在一个批中, 并且只能在一台机器上加工; 式 (3) 表示机器容量约束; 式 (4) 定义了批的加工时间; 式 (5) 表示最大完工时间大于等于每台机器上最后一个批的完工时间; 式 (6) 表示决策变量是二进制变量, 其值为 1 表示工件 J_j 被分组到批 B_b 且该批被分配在机器 M_i 上加工.

2 问题下界

2.1 下界计算方法

Uzsoy[1]已经证明了 SSBN 问题为 NP 难问题, 而本文所研究的问题比 SSBN 更加复杂, 因此该问题也是 NP 难的. 针对 NP 难问题, 通常需要找出问题下界来评估算法的有效性. 本文所研究问题的下界可以通过如下松弛问题得到: 将工件 J_j 松弛为 p_js_j 个单位尺寸及单位加工时间的单元, 然后将松弛后的单位工件按照一定的调度规则分组后再分配在平行批处理机上加工. 由此所得的最大完工时间即为原问题的一个下界. 以 3 种不同的机器容量 S_1, S_2, S_3 (不妨设 $S_1<S_2<S_3$) 且每种容量对应机器数分别为 m_1,

m_2, m_3 为例, 下界的求解过程 (算法 LB) 描述如下.

Step 1: 根据不同的机器容量将工件分成 3 个子集, 记为 J^1, J^2, J^3, 即

$$J = J^1 \bigcup J^2 \bigcup J^3. \tag{7}$$

其中: $J^1 = \{J_j | s_j \leqslant S_1\}$, $J^2 = \{J_j | S_1 < s_j \leqslant S_2\}$, $J^3 = \{J_j | S_2 < s_j \leqslant S_3\}$.

Step 2: 按下式计算下界:

$$\mathrm{LB} = \max\{L_1, L_2, L_3, L_4\}. \tag{8}$$

其中: $L_1 = \max\{p_j | J_j \in J\}$ 为所有工件的最长加工时间; $L_2 = \left\lceil \sum_{J_j \in J^3} p_j s_j \Big/ (m_3 S_3) \right\rceil$ 表示将 J^3 中的工件单位化后分配在容量为 S_3 的机器上加工所得到的最大完工时间; $L_3 = \left\lceil \sum_{J_j \in J^2 \bigcup J^3} p_j s_j \Big/ \sum_{i=2}^{3} (m_i S_i) \right\rceil$ 表示将 J^2 和 J^3 中的工件单位化后, 依次分配在容量为 S_2 和 S_3 的机器上加工所得到的最大完工时间; $L_4 = \left\lceil \sum_{J_j \in J^1 \bigcup J^2 \bigcup J^3} p_j s_j \Big/ \sum_{i=1}^{3} (m_i S_i) \right\rceil$ 表示将所有工件单位化后, 依次分配在所有机器上所得的最大完工时间.

2.2　下界正确性证明

定义 1　m 台容量分别为 $S_1, S_2, \cdots, S_m$ 的平行批处理机的浪费空间 (WS) 是由批内浪费空间与机器间完工时间差异所导致的浪费空间组成, 表示为

$$\mathrm{WS} = \Big(\sum_{i=1}^{m} S_i\Big) C_{\max} - \sum_{J_j \in J} s_j p_j. \tag{9}$$

两个容量不相同机器的浪费空间示意图如图 1 所示.

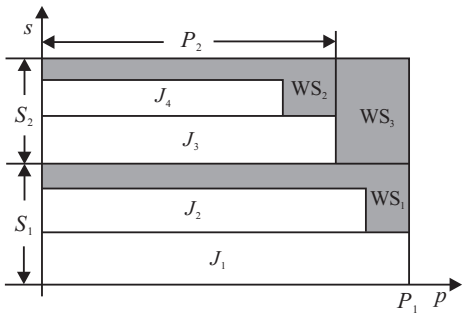

图 1　两个容量不同机器的浪费空间

在图 1 中: 横坐标为加工时间; 纵坐标为尺寸; 容量为 S_1 和 S_2 的两台机器上各分配了一个批, 每个批中包含两个工件, 图中白色部分表示工件, 阴影部分表示浪费空间. 平行批处理机浪费空间 WS 由三个部分组成, 即批内浪费空间 WS_1、WS_2 和机器完工时间差异所引起的空间浪费 WS_3. 其中, WS_1 和 WS_2 是由批内工件尺寸与加工时间的差异引起的, 其差异性越大, 批内浪费空间越大, WS_3 是由平行批处理机之间的完工时间的差异引起的, 其差异性越大, WS_3 越大.

定理 1　对于机器容量不同的 SPBN 问题, 工件单位化所得最大完工时间不超过原问题的最大完工时间.

证明　假设有 m 台容量分别为 $S_1, S_2, \cdots, S_m$ 的平行批处理机, 根据第 2.1 节的工件单位化方法, 令工件单位化后调度得到的目标值为 $C^U_{\max}$, 对应的浪费空间为 WS^U, 则 $\mathrm{WS}^U = \Big(\sum_{i=1}^{m} S_i\Big) C^U_{\max} - \sum_{J_j \in J} s_j p_j$. 根据式 (8), 对于单位工件的平行机批调度问题, 有 $C^U_{\max} = \left\lceil \sum_{J_j \in J} p_j s_j \Big/ \sum_{i=1}^{m} S_i \right\rceil$, 这时可将平行机看作一台容量为 $\sum_{i=1}^{m} S_i$ 的批机器, 则解中只有最后一个批中可能出现浪费空间, 其余的批均被单位工件填满而无浪费空间, 即 $\mathrm{WS}^U < \sum_{i=1}^{m} S_i$. 由式 (9) 可以得到

$$\mathrm{WS}^U - \mathrm{WS} = \Big(\sum_{i=1}^{m} S_i\Big) C^U_{\max} - \Big(\sum_{i=1}^{m} S_i\Big) C_{\max} < \sum_{i=1}^{m} S_i - \mathrm{WS} < \sum_{i=1}^{m} S_i \Rightarrow C^U_{\max} - C_{\max} < 1.$$

由于目标值为整数, $C^U_{\max} - C_{\max} \leqslant 0$, 即

$$C^U_{\max} \leqslant C_{\max}. \qquad \square$$

定理 2　算法 LB 可得到所求问题的一个下界.

证明　令 $C^*_{\max}$ 为最优制造跨度. 假设有容量 $S_1 < S_2 < S_3$, 对应的机器集合为 M^1, M^2, M^3, 且 $|M^i| = m_i$, 根据式 (7) 将 J 划分为 3 个子集. 对于任一批 B_b, 有 $P_b = \max\{p_j | J_j \in B_b\}$. 由式 (5) 可知, $C^*_{\max} \geqslant \sum_{B_b \in B} P_{bi} \geqslant \max\{P_b | B_b \in B\} \geqslant \max\{p_j | J_j \in J\}$, 即 $C^*_{\max} \geqslant \max\{p_j | J_j \in J\}$.

另一方面, 令 $J^i (i = 1, 2, 3)$ 和 J^i 单位化后在 M^i 上加工得到的目标值分别为 $C^i_{\max}$ 和 $C^{i\_U}_{\max}$, 则有以下几种情况:

1) $C^3_{\max} \geqslant \max\{C^1_{\max}, C^2_{\max}\}$. 由于 J^3 不能在容量小于 S_3 的机器上加工, $C^*_{\max} = C^3_{\max}$. 由定理 1 可知, $C^{3\_U}_{\max} \leqslant C^3_{\max}$, 故有 $C^*_{\max} \geqslant C^{3\_U}_{\max}$, 即 $C^*_{\max} \geqslant \sum_{J_j \in J^3} p_j s_j \Big/ (m_3 S_3)$.

2) $C^2_{\max} \geqslant \max\{C^1_{\max}, C^3_{\max}\}$. 由于 J^2 可在 M^2 和 M^3 上加工, 可将 J^2 中完工时间超过 $C^3_{\max}$ 的工件在 $M^2 \bigcup M^3$ 上重新分配来进一步减小目标值, 这相当于在满足机器容量的约束下, 将 $J^2 \bigcup J^3$ 分配在 $M^2 \bigcup M^3$ 上, 令由此得到的子问题目标值为 $C^{2+3}_{\max}$, 则有 $C^{2+3}_{\max} \leqslant C^2_{\max}$. 此时又分两种情况:

① 若 $C^{2+3}_{\max} \geqslant C^1_{\max}$, 则 $C^*_{\max} = C^{2+3}_{\max}$, 由定理 1 可知, 对该子问题的工件单位化后可以得到目标值 $C^{2+3\_U}_{\max} \leqslant C^{2+3}_{\max} \Rightarrow C^*_{\max} \geqslant C^{2+3\_U}_{\max}$, 从而有

$$C^*_{\max} \geqslant \sum_{J_j \in J^2 \cup J^3} p_j s_j \Big/ \sum_{i=2}^{3} m_i S_i.$$

② 若 $C^{2+3}_{\max} < C^{1}_{\max}$, 由于 J^1 还可在 $M^2 \bigcup M^3$ 上加工, 将 J^1 中完工时间超过 $C^{2+3}_{\max}$ 的工件在 M 上重新分配, 这相当于在满足机器容量的约束下, 将 $J^1 \bigcup J^2 \bigcup J^3$ 分配在 $M^1 \bigcup M^2 \bigcup M^3$ 上, 令该子问题的目标值为 $C^{1+2+3}_{\max}$, 则必有 $C^{2+3}_{\max} \leqslant C^{1+2+3}_{\max} \leqslant C^{1}_{\max} \Rightarrow C^{*}_{\max} = C^{1+2+3}_{\max}$. 由定理 1 可得, $C^{1+2+3\_U}_{\max} \leqslant C^{1+2+3}_{\max}$, 则有

$$C^{*}_{\max} \geqslant C^{1+2+3\_U}_{\max} \Rightarrow C^{*}_{\max} \geqslant \sum_{J_j \in J} p_j s_j \Big/ \sum_{i=1}^{3} m_i S_i.$$

综上所述, 式 (8) 所得 LB 为本文研究问题的一个下界. □

2.3 启发式求解算法

基于 MultiFit 策略[16]可设计一个启发式算法 H 对所求问题进行求解. 为了描述方便, 同样假设有 3 种不同机器容量 $S_1 < S_2 < S_3$, 对应机器数和机器集合分别为 m_i 和 M_i $(i = 1, 2, 3)$. 算法 H 的流程图如图 2 所示.

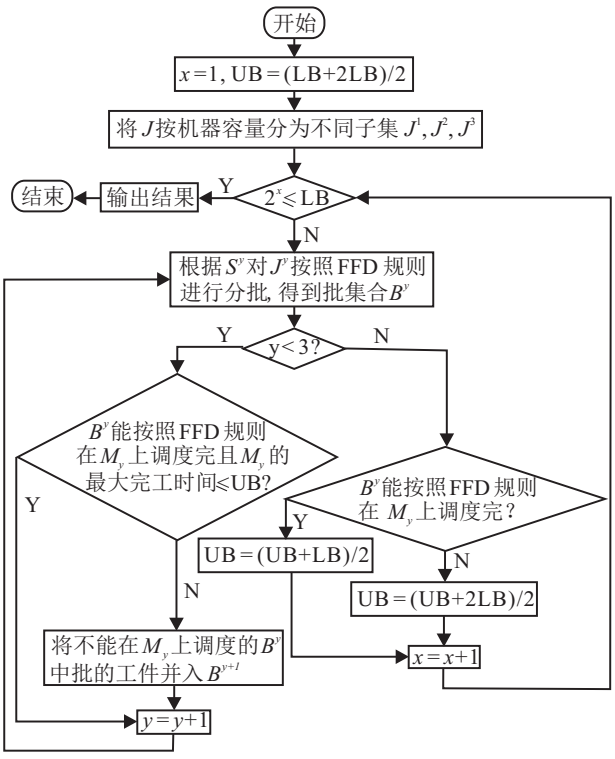

图 2 启发式算法 H 的流程

3 基于ACO的元启发式算法

3.1 蚁群优化算法

ACO 利用人工蚂蚁间存在的自组织交互的特性进行寻优[17], 具有良好的搜索性能, 已被成功应用于求解多种 NP 难的离散优化问题[18-20]. 近年来, ACO 算法也被成功应用于求解 SPBN 问题[13, 21-22].

3.2 编码机制

解的编码是应用 ACO 算法求解批调度的关键问题之一, 本文算法中对批序列直接编码. 针对差异工件的批调度问题, 蚂蚁对于每台机器, 根据机器容量约束来选择未调度工件构建批.

3.2.1 信息素定义

信息素是依赖于问题的参数, 对于解的质量起到关键作用. 这里信息素 τ_{bj} 表示将工件 J_j 分配在批 B_b 中的期望度, 定义为

$$\tau_{bj} = \sum_{J_x \in B_b} \varphi_{xj} / |B_b|. \tag{10}$$

其中: φ_{xj} 表示工件 J_x 和 J_j 分在同一批中的期望值, φ_{xj} 均初始化为 $((1-\rho)\mathrm{LB})^{-1}$, ρ 是介于 0 与 1 之间的随机数, LB 是由式 (8) 所得的下界.

3.2.2 启发式信息

启发式信息也是构建解的重要信息, 与信息素不同的是, 它通常与将工件加入当前批的成本或成本的估计有关. 由于批的浪费空间直接影响了解的质量, 本文基于浪费空间定义启发式信息.

定理 3 对于机器容量不同的 SPBN 问题, 解的浪费空间与制造跨度正相关.

证明 根据定义 1, 解的浪费空间的定义如式 (9) 所示. 工件集合 J、工件尺寸 s_j、加工时间 p_j 和 m 台平行机的容量 $S_1, S_2, \cdots, S_m$ 都是事先给定的, 则式 (9) 中除了制造跨度 $C_{\max}$, 其余都是常量, 因此 WS 与 $C_{\max}$ 正相关. □

显然, 尽量减少每台机器的浪费空间将有助于减少总的浪费空间, 因而将 J_j 分配到 M_i 当前批 B_b 的启发式信息定义为该批所在机器浪费空间的变化, 即

$$\eta_{ibj} = \begin{cases} S_i(P_b - \max\{p_j, P_b\}) + s_j p_j, \ \mathrm{WS}_i > \mathrm{WS}^{*}_i; \\ 1, \ \mathrm{WS}_i \leqslant \mathrm{WS}^{*}_i. \end{cases} \tag{11}$$

其中 WS_i 和 WS^{*}_i 分别表示 J_j 分配到 M_i 上在 B_b 之前和之后的 M_i 的浪费空间.

3.2.3 候选列表构建

由于所求问题工件的加工机器集不同, 即某些工件尺寸大于某些机器容量, 在算法中定义两种候选列表以减小搜索范围. 候选表 L^1_i 定义为满足 M_i 容量约束的工件集合

$$L^1_i = \{J_j | s_j \leqslant S_i\}; \tag{12}$$

候选表 L^2_{ib} 为满足 M_i 上批 B_b 剩余容量的工件集

$$L^2_{ib} = \left\{ J_j \middle| s_j \leqslant \left(S_i - \sum_{J_l \in B_b} s_l \right) \right\}. \tag{13}$$

3.2.4 信息素更新

本文的 ACO 算法利用全局最优解 (即自迭代开始至当前代的最好解) 来更新信息素, 这是 ACO 的一个重要的搜索特性.

根据第 3.2.1 节, $\varphi_{xj}(t)$ 表示第 t 代工件 J_x 与 J_j

分在同一批的期望值. 设 $m_{xj}(t)$ 表示到 t 代为止, J_x 与 J_j 分在同一批的频率. 引入变量 $\Delta\varphi_{xj}(t)$: 若在第 t 代, J_x 与 J_j 没有被分在同一批, 则 $\Delta\varphi_{xj}(t)=0$; 否则 $\Delta\varphi_{xj}(t)=Q/C^*_{\max}(t)$. 其中: Q 为输入参数, $C^*_{\max}(t)$ 为当前代的全局最优解. 基于上述定义, 信息素更新公式为

$$\varphi_{xj}(t+1)=(1-\rho)\varphi_{xj}(t)+m_{xj}(t)\Delta\varphi_{xj}(t). \quad (14)$$

其中: ρ 为信息素蒸发率, 用于控制信息素的挥发速度, 以避免信息素无限累积.

3.2.5　解的构建

为了更好地描述解的构建过程, 这里引入一个初值为 $[1,2,\cdots,n]$ 的 n 维禁忌表 TB, 表示初始时没有工件被调度. 一旦某个工件被调度, 则将 TB 中对应的工件编号更新为 0, TB $=[0,0,\cdots,0]$ 表示所有工件均被调度. 每一代中有 AntNum 只蚂蚁, 每只蚂蚁根据输入的工件集 J 和机器集 M 通过算法 CL 构建解, 算法 CL 如下.

Step 1: 初始化 TB.

Step 2: 当 TB $\neq[0,0,\cdots,0]$ 时, 进行如下步骤:

1) 将 M 中的机器按当前完工时间非减序排序.

2) 根据式 (12) 构建所有机器的候选表 L_i^1 $(i=1,2,\cdots,m)$.

3) 对机器序列中第 1 个 $L_i^1\neq\varnothing$ 的机器 M_i 构建一个新批, 从候选表 L_i^1 中随机选择一个工件 J_j 加入 M_i 的当前批 B_b, 更新 TB.

4) 根据式 (13) 构建 M_i 当前批 B_b 的候选表 L_{ib}^2.

5) 当 $L_{ib}^2\neq\varnothing$ 时, 根据概率 p_{ibj} 从 L_{ib}^2 中选择工件 J_j 加入到当前批

$$p_{ibj}=\begin{cases}\dfrac{(\tau_{bj})^{\alpha}(\eta_{ibj})^{\beta}}{\displaystyle\sum_{J_x\in L_{ib}^2}(\tau_{bj})^{\alpha}(\eta_{ibj})^{\beta}}, & J_j\in L_{ib}^2;\\ 0, \text{ otherwise.}\end{cases} \quad (15)$$

并更新 TB, 根据式 (13) 更新 L_{ib}^2.

Step 3: 输出调度解.

3.2.6　局部优化策略

针对每只蚂蚁构建的可行解, 采用如下局部优化算法 LO 对其进行改进.

Step 1: 找出解中完工时间最大和最小的机器, 记为 M_a 和 M_b, 令其完工时间分别为 C_a 和 C_b;

Step 2: 对于 M_a 上的每个批, 如果该批中只有 1 个加工时间最长的工件 (记为 J_k), 且满足 $p_k+C_b<C_a$ 且 $s_k\leqslant S_b$, 则将 J_k 从 M_a 移至 M_b, 更新 M_a 和 M_b 的完工时间, 转 Step 1;

Step 3: 输出解.

3.2.7　蚁群算法过程

根据上述讨论, 基于蚁群优化的平行机批调度算法 ACO 描述如下.

Step 1: 初始化参数: 最大迭代次数 $T_{\max}$、Q、信息素蒸发率 ρ、信息素和启发式信息权值 α 和 β、蚂蚁数 AntNum;

Step 2: 计算下界 LB;

Step 3: 当前代数 $t=0$, 计算初始信息素;

Step 4: 若 $t\geqslant T_{\max}$, 则输出全局最优解;

Step 5: 调用算法 CL 为每只蚂蚁构建解, 并根据式 (14) 更新信息素;

Step 6: 调用算法 LO 优化每只蚂蚁构建的解;

Step 7: 更新局部最优解和全局最优解;

Step 8: $t\leftarrow t+1$, 转 Step 4.

4　仿真实验

4.1　实验设计

为了评价本文所提算法的性能, 随机生成测试算例对算法进行测试. 根据不同的工件数生成 6 组测试实例, 工件数分别为 90, 108, 126, 144, 162, 180, 每组有 10 个测试实例. 工件加工时间均匀分布于 [8, 48]. 机器总数设为 10, 机器容量有 3 种, 分别为 10, 25, 65. 考虑到在实际应用中, 容量大的机器通常价格较高, 因而实验中大容量的机器数相对较少. 具体地, 对应每种容量的机器数为 5, 3, 2. 工件集 J 中的工件根据机器容量分为 3 组, 即 $J=J^1\bigcup J^2\bigcup J^3$. 其中: J^1 中的工件可以在所有的 10 台机器上加工, J^2 中的工件可以在容量不小于 S_2 的 5 台机上加工, J^3 只能在容量为 S_3 的 2 台机器上加工, $J^1\sim J^3$ 的工件数分别设置为 $2n/3$, $2n/9$ 和 $n/9$, 以使 $|J^1|>|J^2|>|J^3|$.

工件尺寸是测试实例的一个关键参数, 通常基于均匀分布或正态分布来生成, 这样获得小尺寸与大尺寸工件的概率相同. 由于大尺寸工件往往会单独成批而使问题变得相对简单, 采用文献 [23] 方法, 即基于泊松分布生成工件尺寸, 具体有 $\{s_j|J_j\in J^1\}\sim P(\lambda_1)$, $\{s_j|J_j\in J^2\}\sim P(\lambda_2)+10$, $\{s_j|J_j\in J^3\}\sim P(\lambda_3)+25$. 其中: $\lambda_1=5$, $\lambda_2=12.5$, $\lambda_3=32.5$, 使得同一组工件中的小尺寸工件的数量较大尺寸工件多, 这样算法 H 可以将更多的小尺寸工件分配到其他较大尺寸工件的批中, 从而使调度过程更加复杂. 考虑到工件尺寸边界需与机器容量相容, 则有

$$s_j=\begin{cases}S_{k-1}, & s_j<S_{k-1};\\ s_j, & S_{k-1}\leqslant s_j\leqslant S_k;\\ S_k, & s_j>S_k.\end{cases} \quad (16)$$

其中 $S_0=1$. 为了确保有足够的小尺寸工件来填满

大容量的机器, 再分别从 $(S_{k-1}, \lambda]$ 和 $(\lambda, S_k]$ 中随机选择 70% 和 30% 的数据组成最终测试工件集.

实验中使用的相关参数具体如表 1 所示.

表 1 实验设置

因素	分类和取值
机器容量	$S_1 = 10, S_2 = 25, S_3 = 65$
每种容量的机器数	$m_1 = 5, m_2 = 3, m_3 = 2$
工件个数	$n \in \{90, 108, 126, 144, 162, 180\}$
工件尺寸	$s_j \sim P(\lambda_i)$
工件加工时间	$p_j \sim U[8, 48]$

4.2 ACO 算法的参数设置

为了得到高质量的解, 首先通过准备实验来确定 ACO 算法的参数值. 影响 ACO 算法性能的参数包括蚂蚁数 AntNum、迭代次数 $T_{\max}$、信息素和启发式信息重要性参数 α 和 β, 以及信息素蒸发率 ρ 等. 根据文献 [24], 部分参数值确定为: $\text{AntNum} = 20$, $T_{\max} = 200$, $\rho = 0.5$, $Q = n$.

为了确定 α 和 β 的值, 分别从 6 组工件数不同的工件集中随机选择两个实例, 再从 [1, 10] 内分别选择几个不同的 α 和 β 值进行组合, 在这 12 个实例上进行测试, 同时保持其他参数值均不变, 所得结果如图 3 所示.

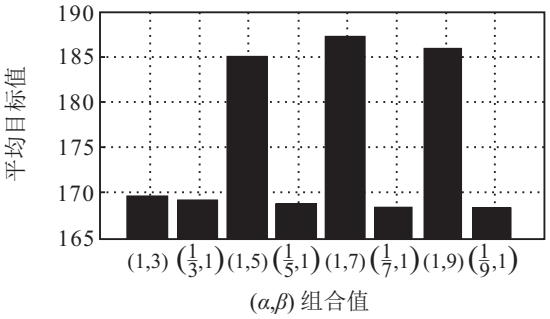

图 3 不同参数组合的平均实验结果

在图 3 中, 横坐标代表不同的 (α, β) 参数组合值, 纵坐标代表不同组合值所得的平均制造跨度. 由图 3 可以看出, 由于本文算法中信息素的值小于 1, 而基于浪费空间的启发式信息值大于 1, 当 $1/\alpha > \beta$ 时, 算法的平均性能优于 $1/\alpha < \beta$ 时的结果, 当 $\alpha = 1/9$, $\beta = 1$ 时, 解的质量相对最好, 这说明增大信息素的权重, 将有利于找到质量更好的解. 故在后面的实验中均采用所得解质量最好的这组参数值.

4.3 实验结果与分析

为了验证本文算法的性能, 将算法 ACO、H 与现有文献中的算法进行比较. 文献 [12-13] 虽然涉及了机器容量不同的平行机批调度问题, 但均假定工件可以在任何机器上加工, 即工件的尺寸不超过任何机器的容量. 目前, 只有文献 [14] 研究的问题与本文的问题相同, 因此这里将本文所提的算法与文献 [14] 提出的 PSO 算法进行比较, 其相关参数均按照原文进行设置. 为了更客观、准确地反映算法的性能, 将 ACO 和 PSO 两个算法分别在每个测试实例上运行 10 次, 统计其平均目标值并进行比较. 所有算法采用 VC++ 编程实现, 运行环境为 Pentium(R) Dual-Core 2.8 GHz CPU, 2 GB内存.

定义算法 A 所得解与下界的距离为

$$R_A = (C_{\max}^A / \text{LB} - 1) \times 100. \tag{17}$$

其中: $C_{\max}^A$ 为算法 A 得到的制造跨度值; R_A 值越小, 算法 A 所得解的目标值越接近下界, 表明所得解的质量越好.

针对不同工件数测试实例的分组实验结果如表 2 ～表 7 所示.

表 2 $n = 90$ 时各算法性能比较

实例编号	LB	PSO 算法		H 算法		ACO 算法	
		$\overline{R}$	$\overline{t}$	$\overline{R}$	$\overline{t}$	$\overline{R}$	$\overline{t}$
*J*1_1	103	39.32	0.80	30.10	0.09	**13.23**	4.70
*J*1_2	102	35.10	0.82	20.59	0.03	**16.30**	5.27
*J*1_3	90	41.44	0.80	24.44	0.03	**17.60**	4.61
*J*1_4	82	42.93	0.79	25.61	0.04	**18.42**	4.75
*J*1_5	98	32.96	0.81	21.43	0.02	**16.32**	5.07
*J*1_6	88	42.27	0.79	22.73	0.03	**14.11**	5.13
*J*1_7	98	38.67	0.81	30.61	0.03	**16.13**	5.05
*J*1_8	96	37.71	0.79	21.88	0.04	**13.74**	4.80
*J*1_9	94	35.00	0.91	31.91	0.03	**16.53**	4.92
*J*1_10	98	33.67	0.80	21.43	0.03	**18.02**	4.65
AVG	94.9	37.91	0.81	25.07	0.04	**16.04**	4.89

表 3 $n = 108$ 时各算法性能比较

实例编号	LB	PSO 算法		H 算法		ACO 算法	
		$\overline{R}$	$\overline{t}$	$\overline{R}$	$\overline{t}$	$\overline{R}$	$\overline{t}$
*J*2_1	103	38.83	1.12	22.33	0.05	**15.30**	7.18
*J*2_2	115	36.17	1.08	19.13	0.03	**13.30**	7.49
*J*2_3	111	38.74	1.08	25.23	0.03	**14.87**	7.20
*J*2_4	118	37.37	1.08	21.19	0.04	**15.63**	7.30
*J*2_5	117	36.75	1.11	14.53	0.03	**14.18**	7.27
*J*2_6	127	31.57	1.06	22.05	0.03	**14.34**	7.37
*J*2_7	122	33.52	1.06	23.77	0.03	**16.75**	7.05
*J*2_8	114	36.84	1.09	18.42	0.03	**14.96**	7.00
*J*2_9	113	37.79	1.09	23.01	0.04	**14.35**	6.94
*J*2_10	119	33.11	1.24	23.53	0.02	**14.64**	7.42
AVG	115.9	36.07	1.10	21.32	0.03	**14.83**	7.22

表 4 $n = 126$ 时各算法性能比较

实例编号	LB	PSO 算法		H 算法		ACO 算法	
		$\overline{R}$	$\overline{t}$	$\overline{R}$	$\overline{t}$	$\overline{R}$	$\overline{t}$
*J*3_1	125	36.40	1.40	17.60	0.03	**13.08**	10.42
*J*3_2	139	33.09	1.51	16.55	0.04	**12.55**	10.12
*J*3_3	152	30.07	1.38	21.71	0.03	**15.19**	10.65
*J*3_4	137	34.89	1.42	23.36	0.03	**12.77**	10.36
*J*3_5	128	36.95	1.38	23.44	0.03	**13.98**	10.34
*J*3_6	139	35.83	1.37	16.55	0.03	**13.95**	10.10
*J*3_7	147	31.43	1.35	14.29	0.03	**12.43**	10.44
*J*3_8	144	32.43	1.38	15.97	0.03	**12.73**	10.51
*J*3_9	129	40.62	1.36	17.83	0.05	**12.65**	10.53
*J*3_10	141	37.45	1.36	34.75	0.04	**12.25**	10.61
AVG	138.1	34.92	1.39	20.20	0.03	**13.16**	10.41

表 5　$n = 144$ 时各算法性能比较

实例编号	LB	PSO 算法		H 算法		ACO 算法	
		$\overline{R}$	$\overline{t}$	$\overline{R}$	$\overline{t}$	$\overline{R}$	$\overline{t}$
*J*4_1	168	31.31	1.68	27.98	0.01	**15.09**	13.88
*J*4_2	151	32.65	1.69	**11.92**	0.02	12.36	14.68
*J*4_3	163	32.52	1.70	12.88	0.01	**12.79**	14.97
*J*4_4	149	36.31	1.69	18.12	0.01	**12.82**	14.89
*J*4_5	148	34.12	1.76	18.24	0.01	**12.03**	13.93
*J*4_6	150	33.33	1.69	16.67	0.01	**11.23**	14.18
*J*4_7	155	34.71	1.68	18.06	0.01	**12.68**	15.14
*J*4_8	165	34.30	1.75	20.00	0.01	**12.30**	14.91
*J*4_9	146	37.12	1.70	18.49	0.02	**11.91**	14.58
*J*4_10	146	34.25	1.67	19.18	0.01	**11.92**	14.64
AVG	154.1	34.06	1.70	18.15	0.01	**12.51**	14.58

表 6　$n = 162$ 时各算法性能比较

实例编号	LB	PSO 算法		H 算法		ACO 算法	
		$\overline{R}$	$\overline{t}$	$\overline{R}$	$\overline{t}$	$\overline{R}$	$\overline{t}$
*J*5_1	160	36.63	2.23	13.75	0.01	**10.81**	19.04
*J*5_2	174	34.37	2.27	22.99	0.02	**11.46**	19.30
*J*5_3	180	34.17	2.17	14.44	0.02	**11.01**	18.29
*J*5_4	167	34.19	2.15	**13.17**	0.01	13.41	19.42
*J*5_5	171	36.32	2.24	23.98	0.01	**11.24**	19.76
*J*5_6	179	32.57	2.19	14.53	0.01	**11.90**	19.63
*J*5_7	173	33.87	2.19	**15.03**	0.05	15.96	19.26
*J*5_8	177	32.54	2.15	16.95	0.01	**11.73**	19.40
*J*5_9	195	28.72	2.21	15.38	0.01	**10.38**	18.60
*J*5_10	175	36.40	2.16	14.29	0.01	**10.26**	18.95
AVG	175.1	33.98	2.20	16.45	0.02	**11.82**	19.17

表 7　$n = 180$ 时各算法性能比较

实例编号	LB	PSO 算法		H 算法		ACO 算法	
		$\overline{R}$	$\overline{t}$	$\overline{R}$	$\overline{t}$	$\overline{R}$	$\overline{t}$
*J*6_1	216	28.56	2.51	21.30	0.03	**11.57**	23.73
*J*6_2	204	32.25	2.53	15.20	0.01	**10.66**	25.16
*J*6_3	171	39.71	2.47	15.79	0.01	**10.98**	25.09
*J*6_4	183	33.39	2.47	19.13	0.01	**10.53**	24.01
*J*6_5	215	28.28	2.49	12.56	0.01	**10.36**	24.67
*J*6_6	186	36.24	2.55	12.90	0.02	**10.21**	24.61
*J*6_7	189	35.34	2.50	12.70	0.01	**10.30**	25.66
*J*6_8	204	31.23	2.51	12.25	0.01	**10.66**	24.48
*J*6_9	190	35.74	2.50	15.26	0.01	**11.56**	25.88
*J*6_10	208	30.24	2.56	12.98	0.02	**10.85**	24.89
AVG	196.6	33.10	2.51	15.01	0.02	**10.77**	24.82

在表 2 ～表 7 中: 第 1 列为对应问题的实例编号, LB 为针对所求问题实例由算法 LB 得到的下界, $\overline{R}$ 和 $\overline{t}$ (单位为 s) 分别为算法运行 10 次的平均性能指标值和平均时间, 表格最后 1 行为各列数值的平均值, 每个测试实例对应的最好性能指标值均加粗显示. 由表 2 ～表 7 可以看出, 除了在测试实例 *J*4_2、*J*5_4 和 *J*5_7 上 ACO 算法的性能指标值较 H 算法略差之外, ACO 算法在其他测试实例上所得解的质量均是最好的, H 算法次之, PSO 算法最差. 而且在每组 10 个测试实例的平均结果中, ACO 算法的结果均最优, 即该算法能找到最接近下界的解, 表明该算法所得解的质量最好.

另一方面, 由于 ACO 算法采用了局部优化等策略, 使其相对于其他算法需要更长的运行时间, 但其所耗时间均在可接受范围. 例如, 对于工件数 180 的问题实例, ACO 算法的平均运行时间在 25 s 以内. 此外, H 算法是确定性算法, 运行一次即可获得解, 因此运行时间最短, PSO 算法的运行时间次之.

各算法在不同工件数的各组实例上的平均性能比较如图 4 所示.

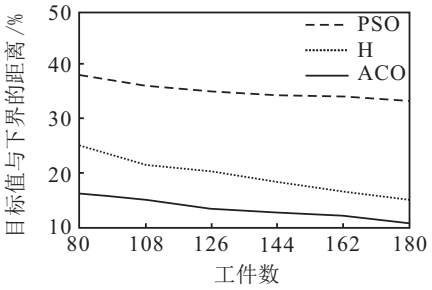

图 4　算法的平均性能比较

在图 4 中, 横坐标为每组实例的工件数, 纵坐标为由式 (17) 计算得到的目标值与下界的差距. 由图 4 可以看出, ACO 算法所得解的质量明显优于其他两种算法, 而 H 算法优于 PSO 算法. 随着工件数的增多以及问题规模的增大, 3 种算法所得解的目标值与下界的差距均有减小的趋势, 其中启发式算法接近速度较 PSO 算法快, 但 ACO 算法所得解的质量依然是 3 种算法中最好的, 且当工件数大于 100 时, ACO 算法所得解的目标值不超过问题下界的 15%.

由实验结果可知, PSO 算法所得解的质量较 H 算法差, 其原因可能是这里 PSO 算法采用的参数值均与文献 [14] 相同, 且没有进行参数优化, 而本文实验中生成测试实例的方法与文献 [14] 不同, 故而其实验结果不是很理想.

综上所述, 可以看出, 本文所提 ACO 算法的综合性能最优, 这可能是由于所提出算法一方面通过构建候选列表减小搜索空间, 另一方面基于浪费空间的启发式信息可以有效提高算法的搜索性能, 具有针对性的局部优化策略则进一步提高了解的质量.

5　结　论

本文主要研究了尺寸差异的工件在容量不同的平行批处理机上加工的调度问题. 首先, 给出了问题的数学模型, 分析了问题的特征, 给出了问题的一个下界, 并证明了下界的正确性; 然后, 给出了一个启发式算法, 并提出了基于 ACO 的元启发式来求解问题; 最后, 通过仿真实验表明, 本文提出的 ACO 算法是非常有效的, 不仅其求解性能超过了现有算法, 而且当问题规模增大时, 其求解性能更加明显.

下一步将研究如何将本文所提算法推广到更复杂的调度模型上, 如带有工件动态到达、工件加工可中断、带有不相容工件簇等约束条件的情况. 此外, 针对本文所研究的平行机批调度问题, 可以设计性能更好的元启发式算法.

参考文献(References)

[1] Uzsoy R. Scheduling a single batch processing machine with non-identical job sizes[J]. Int of Production Research, 1994, 32(7): 1615-1635.

[2] Dupont L, Ghazvini F. Minimizing makespan on a single batch processing machine with non-identical job sizes[J]. European J of Automation, 1998, 32(4): 431-440.

[3] Melouk S, Damodaran P, Chang P. Minimizing makespan for single machine batch processing with non-identical job sizes using simulated annealing[J]. Int J of Production Economics, 2004, 87(2): 141-147.

[4] Damodaran P, Kumar Manjeshwar P, Srihari K. Minimizing makespan on a batch-processing machine with non-identical job sizes using genetic algorithms[J]. Int J of Production Economic, 2006, 103(2): 882-891.

[5] 许瑞, 陈华平, 邵浩, 等. 极小化总完工时间批调度问题的两种蚁群算法[J]. 计算机集成制造系统, 2010, 16(6): 1255-1264.
(Xu R, Chen H P, Shao H, et al. Two kinds of ant colony algorithms to minimize the total completion time for batch scheduling problem[J]. Computer Integrated Manufacturing Systems, 2010, 16(6): 1255-1264.)

[6] 许瑞, 陈华平. 含不同到达时间和尺寸的批调度优化算法[J]. 计算机集成制造系统, 2011, 17(9): 1944-1953.
(Xu R, Chen H P. Optimization algorithm on batch scheduling with different release time and non-identical job sizes[J]. Computer Integrated Manufacturing Systems, 2011, 17(9): 1944-1953.)

[7] Chang P, Damodaran P, Melouk S. Minimizing makespan on parallel batch processing machines[J]. Int J of Production Research, 2004, 42(19): 4211-4220.

[8] Damodaran P, Chang P. Heuristics to minimize makespan of parallel batch processing machines[J]. Int J of Advanced Manufacturing Technology, 2008, 37(9/10): 1005-1013.

[9] Shao H, Chen H P, Huang G, et al. Minimizing makespan for parallel batch processing machines with non-identical job sizes using neural nets approach[C]. Proc of the 3rd IEEE Conf on Industrial Electronics and Applications. Piscataway: IEEE Press, 2008: 1921-1924.

[10] Chen H, Du B, Huang G. Metaheuristics to minimise makespan on parallel batch processing machines with dynamic job arrivals[J]. Int J of Computer Integrated Manufacturing, 2010, 23(10): 942-956.

[11] 李小林, 杜冰, 许瑞, 等. 同类机环境下不同尺寸工件的分批调度问题[J]. 计算机集成制造系统, 2012, 18(1): 102-111.
(Li X L, Du B, Xu R, et al. Batch scheduling on uniform parallel machines with non-identical job sizes[J]. Computer Integrated Manufacturing Systems, 2012, 18(1): 102-111.)

[12] Xu S, Bean J. A genetic algorithm for scheduling parallel non-identical batch processing machines[C]. Proc of the IEEE Symposium on Computational Intelligence in Scheduling. Piscataway: IEEE Press, 2007: 143-150.

[13] Wang H, Chou F. Solving the parallel batch-processing machines with different release times, job sizes, and capacity limits by metaheuristics[J]. Expert Systems with Applications, 2010, 37(2): 1510-1521.

[14] Damodaran P, Diyadawagamage D, Ghrayeb O, et al. A particles swarm optimization algorithm for minimizing makespan of nonidentical parallel batch processing machines[J]. Int J of Advanced Manufacturing Technology, 2012, 58(9/10/11/12): 1131-1140.

[15] Graham R, Lawler E, Lenstra J, et al. Optimization and approximation in deterministic sequencing and scheduling: A survey[J]. Annals of Discrete Mathematics, 1979, 5(2): 287-326.

[16] Coffman E, Garey M, Johnson D. An application of bin-packing to multiprocessor scheduling[J]. SIAM J on Computing, 1978, 7(1): 1-17.

[17] Loiola E M, Abreu N M M, Boaventura-netto P O, et al. A survey for the quadratic assignment problem[J]. European J of Operational Research, 2007, 176(2): 657-690.

[18] Ghafurian S, Javadian N. An ant colony algorithm for solving fixed destination multi-depot multiple traveling salesmen problems[J]. Applied Soft Computing, 2011, 11(1): 1256-1262.

[19] Yang J, Zhuang Y. An improved ant colony optimization algorithm for solving a complex combinatorial optimization problem[J]. Applied Soft Computing, 2010, 10(2): 653-660.

[20] Ding Q, Hu X, Sun L, et al. An improved ant colony optimization and its application to vehicle routing problem with time windows[J]. Neurocomputing, 2012, 98(3): 101-107.

[21] Cheng B, Li K, Chen B. Scheduling a single batch-processing machine with non-identical job sizes in fuzzy environment using an improved ant colony optimization[J]. J of Manufacturing Systems, 2010, 29(1): 29-34.

[22] Xu R, Chen H, Li X. Makespan minimization on single batch-processing machine via ant colony optimization[J]. Computers & Operations Research, 2012, 39(3): 582-593.

[23] Wang J, Leung J. Scheduling jobs with equal-processing-time on parallel machines with non-identical capacities to minimize makespan[J]. Int J of Production Economics, 2014, 156(1): 325-331.

[24] Jia Z, Leung J. An improved meta-heuristic for makespan minimization of a single batch machine with non-identical job sizes[J]. Computers & Operations Research, 2014, 46(6): 49-58.

(责任编辑：闫 妍)