求解差异机器平行批调度的双目标协同蚁群算法

贾兆红 [1, 2] 王 燕 [1] 张以文 [1, 2]

摘 要 利用用户的偏好信息, 提出一种基于蚁群的双目标协同优化算法 (Bi-objective synergy ant colony optimization algorithm based on Pareto domination, PDACO) 并用于求解平行批处理机调度问题. 考虑在一组差异容量并带有不同加工功率的平行批处理机器上, 加工带有不同到达时间、尺寸和加工时间的一组工件, 以同时最小化最大完工时间和总能耗. 偏好向量的引入虽然可以提高算法的收敛性, 但会降低解的多样性. 为了弥补这一缺陷, 在本文所提算法中, 利用两个子蚁群分别沿着不同方向, 迭代地进行独立和联合搜索. 最后, 通过大量的仿真实验验证了本文提出算法的有效性.

关键词 协同优化, 批处理机, 用户偏好, 最大完工时间, 总能耗

引用格式 贾兆红, 王燕, 张以文. 求解差异机器平行批调度的双目标协同蚁群算法. 自动化学报, 2020, **46**(6): 1121−1135

DOI 10.16383/j.aas.c180834

A Bi-objective Synergy Optimization Algorithm of Ant Colony for Scheduling on Non-identical Parallel Batch Machines

JIA Zhao-Hong[1, 2] WANG Yan[1] ZHANG Yi-Wen[1, 2]

Abstract The paper proposed a bi-objective synergy optimization algorithm based on ant colony (PDACO) using preference vectors. It is used to solve the batch scheduling problem. These constraints, a set of jobs with different arrival times, sizes, and processing times on a group of parallel batch processing machines (BPMs) with different capacities and different processing powers, are taken into consideration. The objective is to minimize the maximum completion time and the total energy consumption, simultaneously. Although the preference vector is effective to improve the convergence of algorithm, it will deteriorate the diversity of solutions. In order to reduce the adverse effects of the preference vector, two colonies searching solutions in different directions iteratively use the independent and co-operation search approaches in the proposed algorithm. Finally, through extensive simulation experiments, the validity of the algorithm PDACO proposed in this paper is verified.

Key words Synergy optimization, batch processing machines (BPMs), user preference, maximum completion time, total energy consumption

Citation Jia Zhao-Hong, Wang Yan, Zhang Yi-Wen. A bi-objective synergy optimization algorithm of ant colony for scheduling on non-identical parallel batch machines. *Acta Automatica Sinica*, 2020, **46**(6): 1121−1135

批调度问题 (Batch scheduling problem, BSP) 是由经典调度问题延伸出来的一类更加复杂的问题, 在现实中有着广泛的应用, 如铸造业、家具制造业、金属业、航空业、制药业和物流货运[1−2]. 批调度问题源于半导体制造业中芯片生产过程的最后测试阶段. 芯片被分组放置在加热板上, 再将加热板放到烤箱中进行高温检测. 一个加热板可以同时容纳多个芯片, 只要该加热板上的芯片总体积不超过该加热板的大小即可. 批调度问题与传统调度问题的最大区别在于批调度问题能够在一台机器上同时加工多个工件. 多个工件组成的批的加工时间等于该批中所有工件的最长加工时间. 在生产制造过程中, 生产效率往往是影响企业收益的一个至关重要的因素, 所以批调度问题的目标通常与加工时间相关.

随着传统制造业的发展, 环境污染问题日益严重. 许多制造企业消耗大量能源, 如煤炭或电力. 因此近年来, 绿色制造引起了企业管理者和研究人员的关注. 绿色制造综合考虑了生产对环境的影响和资源的利用率.

随着技术的进步, 功能相同的新机器的处理能

收稿日期 2018-12-17 录用日期 2019-08-08
Manuscript received December 17, 2018; accepted August 8, 2019
国家自然科学基金 (71601001, 71671168), 中国教育部人文社会科学青年基金会 (15YJC630041), 安徽省科学基金 (1608085MG154), 安徽省教育厅自然科学基金 (KJ2015A062) 资助
Supported by National Natural Science Foundation of China (71601001, 71671168), Humanity and Social Science Youth Foundation of Ministry of Education of China (15YJC630041), Natural Science Foundation of Anhui Province (1608085MG154), and Natural Science Foundation of Anhui Provincial Department of Education (KJ2015A062)
本文责任编委 周志华
Recommended by Associate Editor ZHOU Zhi-Hua
1. 安徽大学计算机科学与技术学院 合肥 230601 2. 安徽大学计算智能与信号处理教育部重点实验室 合肥 230601
1. School of Computer Science and Technology, Anhui University, Hefei 230601 2. Key Laboratory of Intelligent Computing and Signal Processing of Ministry of Education, Anhui University, Hefei 230601

力通常比旧机器更强, 功耗更大, 但成本也更高. 因此在实际生产环境中, 制造企业里通常部分更新加工设备, 使得工厂会同时存在新旧两种机器, 以兼顾成本和生产效率.

基于上述讨论, 本文考虑如下调度问题. 假设有 m 台能耗和容量不同的平行批处理机. 由于在生产过程中, 工件 (任务) 的完工时间对生产成本有很大的影响, 并且可以反映生产效率. 因此, 在本文的问题中我们考虑同时最小化最大完工时间 ($C_{\max}$) 和总能耗 (TEC) 两个目标, 以提高生产效率和能源使用率.

在批调度问题中, 多个工件以批的形式同时被加工, 只要批的尺寸不超过加工该批的机器容量. 单台机器上带有差异工件尺寸的批调度问题已经被证明是 NP 难问题[3]. 而本文所研究的问题比单机环境更复杂, 并且从单目标拓展成双目标问题, 所以本文研究的问题也是 NP 难问题. 此外, 工件带有不同的加工时间和动态到达时间的约束将进一步增加问题的求解难度.

Ikura 等[4] 首先研究了单台批处理机上 (Batch processing machine, BPM) 的批调度问题. 之后, 赵玉芳等[5] 研究了单机连续型极小化最大完工时间的批调度问题, 给出了一个复杂性为 $\mathrm{O}(n^2)$ 的动态规划算法. Zhou 等[6] 考虑了具有不同工件尺寸和随机到达时间的单机批调度问题, 目标是最小化最大延迟时间并提出了一个改进的粒子群优化算法.

平行机上的批调度问题 (Parallel batch scheduling problem, BSPP) 是单机 BSP 的扩展. Chang 等[7]考虑了不同工件尺寸的 BSPP, 并提出一种模拟退火 (Simulated annealing, SA) 算法. 与 CPLEX 的比较实验结果表明 SA 在解的质量和计算时间方面具有更好的性能. Jia 等[8] 研究了多台平行机上带有差异工件尺寸的 BSPP, 提出了一种有效的蚁群优化算法 (Ant colony optimization, ACO) 并通过实验验证了算法的有效性. Shi 等[9] 研究在单机和多机环境下考虑不相容工件簇的批处理调度问题, 并提出了一个启发式算法. Zhou 等[10] 研究了在多批处理机上带有不同工件尺寸和动态到达时间以最小化最大完工时间的问题, 提出了一个基于随机键的遗传算法. Arroyo 等[11] 提出了几个启发式算法求解带有不同工件尺寸和到达时间的 BSPP, 问题的目标是最小化最大完工时间. 陆志强等[12] 考虑资源转移时间的资源受限项目调度问题, 提出了一种内嵌分支定界寻优搜索的遗传算法和一种基于任务绝对顺序的编码策略.

在实际生产环境中, 为了提高生产系统的整体性能, 通常需要权衡多个目标, 例如, 生产效率、生产成本、时间成本等. Zhou 等[13] 考虑存在动态到达作业的大规模并行批处理机调度问题, 目标是最小化完工时间和电力成本, 提出了一种多目标差分进化算法. Du 等[14] 研究了差异机器容量的流水车间批调度问题, 提出了最小化制造跨度和总电力成本的多目标 ACO 算法. Jia 等[15] 针对同时优化总电力成本和制造跨度的 BSPP, 提出了基于 Pareto 的蚁群优化算法 (Pareto-based ant colony optimization, PACO), 并通过仿真实验验证了算法的有效性. 汪恭书等[16] 研究了连铸–轧制混合流生产模式下轧批调度问题, 以最小化温装钢坯 (热钢锭) 缓冷 (等待) 导致的热能损失和连轧机架切换带来的产能损失为目标并提出了一种分支–定价算法进行求解. 郎劲等[17] 研究了最小化采油成本的大规模油井调度问题并提出了一种基于变量分离的拉格朗日松弛算法.

1 问题模型

根据三参数表示法, 本文研究的问题可表示为 $P_m|p-batch, l_i, r_j, p_j, s_j, S_i, M_j|(C_{\max}, TEC)$. 该问题包含如下假设:

1) 机器的容量不相同. 每台机器 M_i 具有容量 S_i 和加工功率 l_i. 不妨假定 $S_1 \le S_2 \le \cdots \le S_m$, $l_1 \le l_2 \le \cdots \le l_m$, 每个工件的尺寸不超过最大机器容量 S_m.

2) $J=\{J_1, J_2, \cdots, J_n\}$ 表示一组工件, 工件分组成批后在 m 台 BPM 上加工, 记为 $M=\{M_1, M_2, \cdots, M_m\}$. B 表示批集合, B_{bi} 表示机器 M_i 上第 b 个批. 每个工件 J_j 带有加工时间 p_j, 尺寸 s_j 和到达时间 r_j.

3) 直到所有工件被分批后才能确定批集合. 批中所有工件尺寸总和不能超过加工该批的机器容量. 机器 M_i 上的批 $B_{bi} \in B$ 的到达时间和加工时间 (表示为 RT_{bi} 和 PT_{bi}) 由批中工件决定, 分别等于批中所有工件的最迟到达时间和最长加工时间. 即, $RT_{bi}=\max\{r_j|J_j \in B_{bi}\}$, $PT_{bi}=\max\{p_j|J_j \in B_{bi}\}$.

4) 批一旦开始加工就不能被中断, 其他工件不能添加到批中或从批中移走, 直到批加工完成为止. ST_{bi} 和 CT_{bi} 表示批 B_{bi} 的开始加工时间和完工时间. 机器上的批序列一旦被确定, 则有 $ST_{bi}=\max\{R_{bi}, CT_{b-1,i}\}$, $CT_{bi}=ST_{bi}+PT_{bi}$, 其中 $CT_{b-1,i}$ 是在同一台机器上位于 B_{bi} 之前的批的完工时间, 且每台机器上第一个批开始加工时间为 0.

5) C_i 表示机器 M_i 的完工时间, 等于 M_i 上最后一个批的完工时间. 问题的第 1 个目标是最小化 $C_{\max}$, 等于所有机器完工时间的最大值, 即 $C_{\max}=\max_{i=1}^{m}\{C_i\}$. 第 2 个目标是最小化 TEC, 即所有机器的总能耗. 机器 M_i 的能耗 (EC_i) 等于在

M_i 上所有批的加工时间乘以该机器的加工功率 (l_i) 之和.

$$X_{jbi}=\begin{cases}1, & \text{工件}J_j\text{在机器}M_i\text{上的批 }B_{bi}\text{ 内}\\ 0, & \text{其他}\end{cases} \tag{1}$$

$$Y_{bi}=\begin{cases}1, & \text{批}B_{bi}\text{在机器}M_i\text{上加工}\\ 0, & \text{其他}\end{cases} \tag{2}$$

基于以上假设, 问题的数学模型构建如下:

$$\min \quad C_{\max} \tag{3}$$

$$\min \quad TEC=\sum_{i=1}^{m}EC_i=\sum_{i=1}^{m}\sum_{b=1}^{n}l_i(PT_{bi}\times Y_{bi}) \tag{4}$$

$$\text{s.t.} \quad X_{jbi}\le Y_{bi},\ \ j=1,\cdots,n,b=1,\cdots,k,\ i=1,\cdots,m \tag{5}$$

$$\sum_{i=1}^{m}\sum_{b=1}^{n}X_{jbi}=1,\ \ j=1,\cdots,n \tag{6}$$

$$\sum_{i=1}^{m}s_jX_{jbi}\le S_i,\ \ j=1,\cdots,n,b=1,\cdots,k \tag{7}$$

$$PT_{bi}\ge p_jX_{jbi},\quad j=1,\cdots,n,b=1,\cdots,k,\ i=1,\cdots,m \tag{8}$$

$$ST_{bi}\ge r_jX_{jbi},\quad j=1,\cdots,n,b=1,\cdots,k,\ i=1,\cdots,m \tag{9}$$

$$ST_{bi}\ge ST_{(b-1)i}Y_{(b-1)i}+PT_{(b-1)i}Y_{(b-1)i},\ b=1,\cdots,k,i=1,\cdots,m \tag{10}$$

$$CT_{bi}=ST_{bi}Y_{bi}+PT_{bi}Y_{bi},\quad b=1,\cdots,k,\ i=1,\cdots,m \tag{11}$$

$$C_{\max}\ge CT_{bi},\quad b=1,\cdots,k,i=1,\cdots,m \tag{12}$$

$$Y_{bi},X_{jbi}\in\{0,1\},\quad j=1,\cdots,n,b=1,\cdots,k,\ i=1,\cdots,m \tag{13}$$

其中, 式 (3) 表示问题目标是最小化最大完工时间; 式 (4) 表示最小化总能耗目标; 约束条件 (5) 表示工件只能被添加到已创建的批中; 约束条件 (6) 表示任意一个工件只能被分到一个批中并且只能在一台机器上加工; 约束条件 (7) 表示批内工件的总尺寸不能超过加工该批的机器容量; 约束条件 (8) 表示一个批的加工时间等于批内所有工件的最长加工时间; 约束条件 (9) 表示一个批的最早开始加工时间为该批内所有工件的最迟到达时间; 约束条件 (10) 表示批的加工不能被中断, 并且只能在同一台机器上该批之前的一个批加工结束之后才能开始加工; 式 (11) 定义了每台机器上批的完工时间; 约束条件 (12) 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 (13) 表示二进制决策变量, X_{jbi} 表示工件 J_j 是否在机器 M_i 的批 B_{bi} 中, Y_{bi} 表示批 B_{bi} 是否在机器 M_i 上加工.

2　双目标协同蚁群算法

蚁群优化算法是智能优化的研究热点之一, 已被广泛应用于多种 NP 难的离散优化问题[18−20]. 近年来, ACO 算法也被用于求解批调度问题, 利用自组织交互特性进行寻优, 具有良好的搜索性能.

为了接近实际的决策过程和提高算法的全局搜索能力, 有学者研究在双目标 ACO 算法[14−15] 的搜索过程中引入偏好信息来平衡不同目标间的搜索. 然而同时兼顾两个目标间的搜索会使蚁群忽略两个目标各自的优化方向, 从而限制了蚁群的搜索范围.

因此, 本文提出独立搜索与联合搜索相结合的双目标协同蚁群算法, 即两个蚁群分别针对不同的目标独立进行搜索, 在共同感兴趣的区域进行联合搜索. 具体地, 在独立搜索阶段, 两个蚁群分别针对不同目标进行搜索, 并记录获得的非劣解. 独立搜索一定代数后, 两个蚁群利用当前获得的非劣解集进行联合搜索. 由于连续几代间的解的差异性相对较小, 也为了降低算法的复杂度, 联合搜索每隔固定代数执行一次.

2.1　编码机制

编码是应用 ACO 算法求解问题的第一步. 为了降低求解难度, 可以先构建批缩小搜索空间, 再根据启发式在批处理机上调度批. 虽然这种方法可以降低问题求解的复杂度, 但是划分后的子问题的解空间小于原问题的解空间. 因此本文采用 Jia 等[15] 使用的编码方式, 即在一个工件被分到批中之后, 同时获得工件所在批和机器的信息. 表 1 给出了一个解编码的例子.

表 1　编码实例

Table 1　An example of a solution encoding

J_j	b	i
J_1	2	3
J_2	1	4
J_3	3	2
J_4	5	4
J_5	4	1
J_6	2	3
J_7	3	2
J_8	5	5
J_9	1	4

表 1 所示的编码中包含 9 个工件, 表示为一个 9×2 的向量. b 和 j 分别代表工件所在的批号和机器号. 在这种编码方式中, 一旦工件 J_j 被添加到机器 M_i 的批 B_{bi} 中, 那么工件 J_j 的编码就完成了. 当所有工件的位置都确定后, 就构建了一个完整的解.

2.2 信息素更新

在构建可行解的过程中, 信息素表示蚂蚁构建解的经验. 为了区分不同目标之间的影响, 每个蚁群分别针对每个目标设置一个独立的信息素矩阵, 并且每个蚁群的两个信息素矩阵会同时更新. 信息素 τ_{bij} 表示工件 J_j 分配到批 B_{bi} 中的期望值, 定义为

$$\tau^x_{bi,j}=\frac{\sum\limits_{J_v\in B_{bi}}\tau^x_{vj}}{|B_{bi}|},\ J_j\in L^3_{bi} \tag{14}$$

其中, τ^1_{vj} 与 τ^2_{vj} 分别代表 $C_{\max}$ 和 TEC 的信息素路径. $|B_{bi}|$ 表示当前批 B_{bi} 中的工件数, J_v 是批 B_{bi} 中的工件, J_j 表示候选列表 L^3_{bi} 中的工件.

基于蚁群的双目标协同优化算法 (PDACO) 交替使用局部最优解 (即迭代最优解) 和全局最优解 (即自迭代开始到当前代的最优解) 更新信息素. 设 $m_{vj}(t)$ 表示到 t 代为止, J_v 和 J_j 分在同一批中的频数. 引入变量 $\Delta\tau^x_{vj}(t)$: 若在第 t 代 J_v 和 J_j 没有被分在同一批中, 则 $\Delta\tau^x_{vj}(t)=0$; 否则 $\Delta\tau^x_{vj}(t)=Q/P$, 其中 $x=1$ 时, $P=C^*_{\max}(t)$; $x=2$ 时, $P=\sqrt{TEC^*(t)}$. Q 设置为工件数, P 为到第 t 代为止获得的最优解对应的目标值. 信息素更新公式为

$$\tau^x_{vj}\ (t+1)=(1-\rho)\ \tau^x_{vj}(t)+m_{vj}(t)\ \Delta\tau^x_{vj}(t) \tag{15}$$

其中, ρ 表示信息素蒸发率, 用于控制信息素蒸发的速度, 避免信息素无限累积. 同时为避免算法陷入局部最优, 将信息素的范围设为 $[\tau_{\min},\tau_{\max}]$, 其中, $\tau_{\max}=\dfrac{1}{(1-\rho)C^*}$, $\tau_{\min}=\dfrac{\tau_{\max}\ (1-\sqrt[n]{0.05})}{(\frac{n}{2}-1)\ \sqrt[n]{0.05}}$ [21], C^* 为当前所得最优解的目标值. 当信息素取值大于 $\tau_{\max}$ (或小于 $\tau_{\min}$), 则将其设置 $\tau_{\max}$ (或 $\tau_{\min}$).

2.3 启发式信息

启发式信息是在蚁群优化算法构建解的过程中用于指导搜索的重要信息. 启发式信息的设计通常依赖于具体问题. 本文提出的算法中两个蚁群根据偏好信息沿不同的方向进行搜索, 第 1 个蚁群主要优化 $C_{\max}$, 第 2 个蚁群侧重于优化 TEC, 因此分别设计不同的启发式信息. 由于影响总能耗的因素复杂, 并且机器能耗的计算与批的加工时间相关, 因此可以依据加工时间来设计 TEC 的启发式信息. 另外批的浪费空间与 $C_{\max}$ 呈正相关的关系[22], 所以本文基于浪费空间设计 $C_{\max}$ 的启发式信息.

对于可行解 σ, B_{bi} 是机器 M_i 上的任一批. WIS_{bi} 表示批 B_{bi} 的剩余空间.

定义 1. WIS_{bi} 表示机器 M_i 上所有批的总剩余空间, 计算方式为

$$WIS_i=S_i\ CT_{|B^i|i}-\sum_{J_j\in B^i}s_j\ p_j \tag{16}$$

其中, B^i 表示机器 M_i 上的批集合, $\left|B^i\right|$ 表示 M_i 中批的数量, $CT_{|B^i|,i}$ 表示机器 M_i 上最后一个批的完工时间. 假设 WIS'_i 表示工件 J_u 加入批 $B_{|B^i|,i}$ 后 M_i 上的剩余空间, 则有

$$WIS'_i=S_i\ \left(\max\left\{ST_{|B^i|i},r_u\right\}+\max\left\{PT_{|B^i|i},p_u\right\}\right)-\sum_{J_j\in B^i}s_jp_j-s_up_u \tag{17}$$

令 ΔWIS^u_i 表示上述两者之间的差值, 即 $\Delta WIS^u_i=WIS_i-WIS'_i$. 根据候选工件与批的关系, 工件加入指定批后可能会增加或降低 WIS_i 的值, 即 ΔWIS^u_i 可能是正值或负值. 根据 ΔWIS^u_i 和启发式信息的正值特性, 针对 $C_{\max}$ 的启发式信息 $\eta^{11}_{bi,u}$ 为

$$\eta^{11}_{bi,u}=\begin{cases}S_i\ (CT_{bj}-\max\{ST_{bi},r_u\}-\max\{PT_{bi}\})+\\ \quad s_up_u, & \Delta WIS^u_i>0\\ 1, & \Delta WIS^u_i\le 0\end{cases} \tag{18}$$

在第 1 个蚁群中, 同时考虑工件的加工时间和到达时间设计第 2 个启发式信息 $\eta^{12}_{bi,u}$ 为

$$\eta^{12}_{bi,u}=\frac{1}{|PT_{bi}-p_u|+1}+\frac{1}{|ST_{bi}-r_u|+1} \tag{19}$$

根据 TEC 的定义可知, TEC 与机器的加工时间和机器功率相关, 因此第二个蚁群设计的启发式信息 $\eta^{21}_{bi,u}$ 和 $\eta^{22}_{bi,u}$ 分别如下:

$$\eta^{21}_{bi,u}=\frac{1}{|PT_{bi}-p_u|+1}+\frac{1}{s_u} \tag{20}$$

$$\eta^{22}_{bi,u}=\frac{s_u}{S_i-\sum\limits_{j=1}^{d}s_j} \tag{21}$$

其中, $d=\left|B_{|B^i|i}\right|$, 表示批 $B_{|B^i|i}$ 中的工件数.

2.4 候选列表的构建

由于所求问题中某些工件尺寸大于部分机器的容量, 为了提高搜索效率, 采用候选列表来缩小工件选择的范围. 候选列表 L^1_i 定义为满足机器 M_i 容

量约束的工件集合

$$L_i^1 = \{J_j \,|s_j \le S_i\} \tag{22}$$

候选列表 $L_{i,bi}^2$ 为满足机器 M_i 上批 B_{bi} 剩余容量的工件集合:

$$L_{i,bi}^2 = \left\{ J_j \in L_i^1 \left| s_j \le \left(S_i - \sum_{J_l \in B_{bi}} s_l \right) \right. \right\} \tag{23}$$

根据浪费空间的定义可知, 候选工件需要同时考虑提高机器的利用率和减少工件的等待时间, 因此, 当 $\Delta WIS_i^u > 0$ 时, 将未加工工件 J_u 加入候选列表. 候选列表 L_{bi}^3 定义为

$$\begin{aligned} L_{bi}^3 = \{J_u \in L_{i,bi}^2 | s_u\ p_u > S_i\ (\max\{ST_{bi}, r_u\} - \\ ST_{bi} + \max\{PT_{bi} - p_u\} - PT_{bi})\} \end{aligned} \tag{24}$$

2.5　决策规则

为了保证解的质量, 采用分机分批同时考虑的方式构建解. 蚂蚁在构建可行解的过程中, 基于如下规则选择机器:

$$\arg\min_{M_i \in M} \left\{ \sum_{x=1}^{2} \omega^x \cdot f_i^x \right\} \tag{25}$$

其中, f_i^1 表示机器 M_i 的完工时间, f_i^2 表示机器 M_i 的 EC_i 的值. ω^1 和 ω^2 分别对应 f_i^1 和 f_i^2 的权值, 表示用户对不同目标的偏好程度, 且 $\omega^1 + \omega^2 = 1$. 在不同的蚁群中, ω^1 和 ω^2 的取值范围不同.

在构建批的过程中, 如果当前批 B_{bi} 为空, 蚂蚁从候选列表 L_i^1 中随机选择一个未调度的工件加入批; 否则, 蚂蚁根据状态转移概率从 L_{bi}^3 中选择一个工件加入 B_{bi} 中. 其中状态转移概率定义为

$$P_{bi,u} = \begin{cases} \dfrac{\left(\sum\limits_{x=1}^{2} \tau_{bi,u}^b \omega_x \right)^{\alpha_y} \left(\eta_{bi,u}^{y1} \cdot \eta_{bi,u}^{y2} \right)^{\beta_y}}{\sum\limits_{J_j \in L_{bi}} \left(\sum\limits_{x=1}^{2} \tau_{bi,j}^b \omega_x \right)^{\alpha_y} \left(\eta_{bi,j}^{y1} \cdot \eta_{bi,j}^{y2} \right)^{\beta_y}}, J_u \in L_{bi}^3 \\ 0, \qquad \text{其他} \end{cases} \tag{26}$$

其中, ω_x 表示用户对第 x 个目标的偏好程度, 与式 (25) 中 ω^x 的取值相同. α_y 和 β_y 表示在第 y 个蚁群中信息素和启发式信息的重要程度, y 的取值为 1 或 2.

2.6　局部优化策略

在 PDACO 算法中, 当一个工件被选择加入批中, 如果到达时间较晚的工件优先被加工, 会延迟当前批的开始加工时间, 从而影响工件的完工时间 (即 $C_{\max}$). 因此针对 $C_{\max}$ 提出一个局部优化策略. 假设当前生成了一个解 σ_0, 首先将每台机器上的批按到达时间非递减排序, 再根据新生成的批序列进行调度. 局部优化策略的伪代码如下所示.

```
for i to m do
    for j ← 1 to B_{M_i} do
        /* B_{M_i} 表示机器 M_i 上批总数*/;
        for k ← 1 to B_{M_i} do
            if RT_{ji} > RT_{ki} then
                交换批 B_{ji} 和批 B_{ki}, 并更新批
                与机器的相关参数;
            end if
        end for
    end for
end for
输出批序列.
```

为了便于理解, 将给出一个实例说明其中的细节, 具体参数如表 2 所示. 表 2 列出了解 σ_0 中 5 个批的参数信息. 第一行表示批的索引, 第二行和第三行分别表示批的到达时间和加工时间. 解 σ_0 中批调度方案如图 1 所示. 图 2 给出了通过局部优化策略对 σ_0 调整后的解 σ_1.

表 2　实例参数

Table 2　The example of the problem

B_{bi}	R_{bi}	PT_{bi}
B_{11}	3	7
B_{21}	12	5
B_{12}	5	4
B_{22}	2	5
B_{32}	3	4

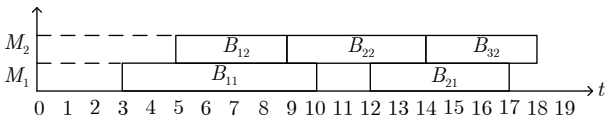

图 1　解 σ_0 甘特图

Fig. 1　The Gantt chart of σ_0

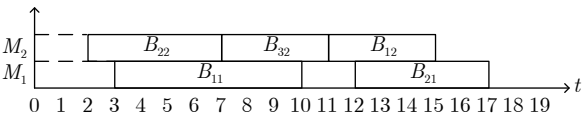

图 2　解 σ_1 甘特图

Fig. 2　The Gantt chart of σ_1

定义 2. 假设存在一个可行解 σ, B_{ki} 为任意批, 机器 M_i 上批 B_{ki} 的开始加工时间为 ST_{ki}, 批 B_{ki} 的到达时间为 R_{ki} ($ST_{ki} \ge R_{ki}$). 若 ST_{ki} 和 R_{ki} 满足

$ST_{ki} = R_{ki}$，则称B_{ki}按时加工，否则称B_{ki}延迟加工.

从图 1 中可以看出，$C_{\max}$ 值为 18. 在机器M_2上批B_{12}按时加工，批B_{22}和B_{32}延迟加工，这是由于到达时间较晚的B_{12}最先加工导致的. 若是先到达的批先加工，则会减少不必要的等待时间. 按照局部优化策略调整解σ_0中批的调度顺序，由$B_{12} \to B_{22} \to B_{32}$调整为$B_{22} \to B_{32} \to B_{12}$，从而得到图 2 中解$\sigma_1$，目标值$C_{\max}$由$\sigma_0$中的 18 降为$\sigma_1$中的 17.

2.7 算法描述

算法流程图如图 3 所示. 在 PDACO 算法中，首先初始化参数和信息素矩阵，从第 1 代开始，在每一代中两个蚁群各自偏好一个目标进行搜索，各自得到一个非支配解集，记为NDS_1和NDS_2. 每隔固定代数K，利用两个蚁群的非支配解集融合形成的非支配解集NDS去更新两个蚁群的信息素矩阵，从而达到联合搜索的目的；否则，两个蚁群利用各自的非支配解集更新信息素矩阵. 直到满足终止条件后，输出非支配解集NDS.

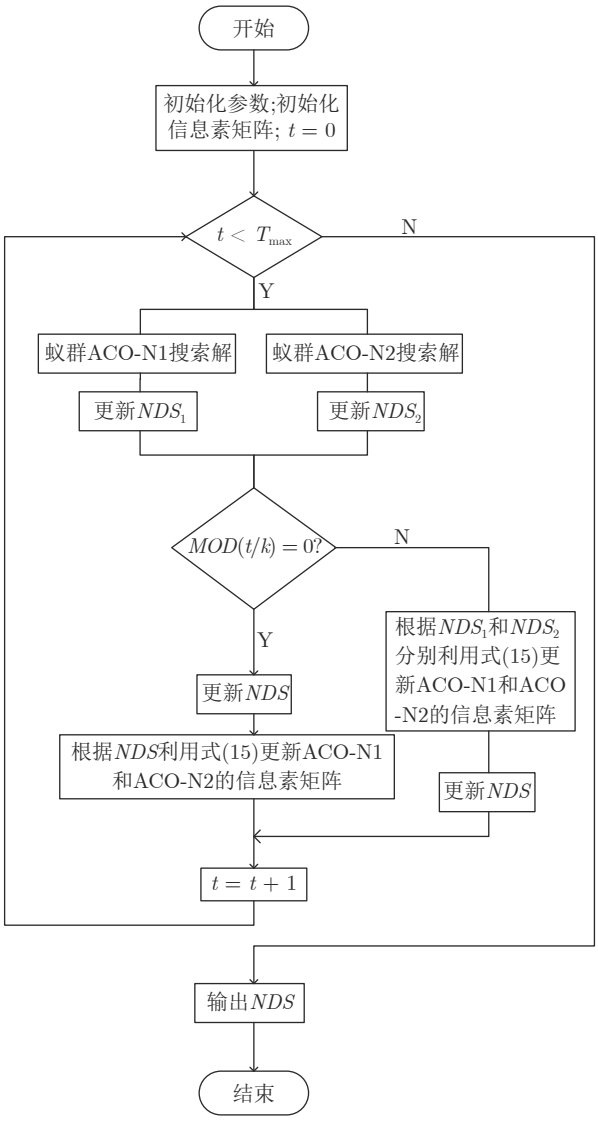

图 3 PDACO 算法总流程图

Fig. 3 The flowchart of algorithm PDACO

为了更好地描述解的构建过程，引入一个初值为 [1, 2, 3, $\cdots$, n] 的n维禁忌表TB，表示初始时工件均未被调度；一旦某个工件被调度，则将TB中对应的该工件编号更新为 0；如果$TB = [0, 0, \cdots, 0]$则表示所有工件均被调度. 在每一代中，蚂蚁根据当前的工件集J和机器集M搜索解.

蚁群 ACO-N1 搜索过程描述如下:

1: **for** $a1 \leftarrow 1$ to N_1 **do**
2: 生成偏好向量$W = (W_{mk}, W_{tec})$;
3: $U^{a1}(t) \leftarrow J$ /* U^{a1} 为蚂蚁$a1$未调度的工件集合*/;
4: **while** $U^{a1}(t) \neq \emptyset$ **do**
5: 蚂蚁$a1$根据式 (22) 构建候选列表L_i^1;
6: 蚂蚁$a1$根据式 (25) 选择机器构建一个新的空批B_{bi}; 从L_i^1中随机选择一个工件加入批B_{bi}; $U^{a1}(t) \leftarrow U^{a1}(t) - B_{bi}$;
7: 蚂蚁$a1$根据式 (23) 和式 (24) 构建候选列表$L_{i,bi}^2$, L_{bi}^3;
8: **while** $L_{bi}^3 \neq \emptyset$ **do**
9: 蚂蚁$a1$根据式 (26) 按概率从L_{bi}^3中选择工件加入当前批, 更新$U^{a1}(t)$; 更新TB;
10: 蚂蚁$a1$根据式 (23) 和式 (24) 构建候选列表$L_{i,bi}^2$, L_{bi}^3;
11: 蚂蚁$a1$根据式 (18) 和式 (19) 更新启发式$\eta_{bi,u}^{11}$, $\eta_{bi,u}^{12}$;
12: **end while**
13: **end while**
14: 调用局部优化策略改进当前解.
15: **end for**

蚁群 ACO-N2 的搜索过程与 ACO-N1 类似，不同点包括：1) 第 1 行：ACO-N2 的规模为N_2；2) 第 2 行：偏好向量的生成方式不同，具体将在下文介绍；3) 第 11 行：蚂蚁根据式 (20) 和式 (21) 更新启发式$\eta_{bi,u}^{11}$, $\eta_{bi,u}^{12}$. 其他步骤均与 ACO-N1 相同. 蚁群 ACO-N1 和 ACO-N2 中机器的选择和状态转移概率的计算基于偏好向量 (w_{mk}, w_{tec})，其中$w_{mk} + w_{tec} = 1$，且偏好分量的值均通过均匀分布生成，具体地，ACO-N1 的w_{mk}和 ACO-N2 的w_{tec}都均匀分布于 [0.8, 1][14].

PDACO 算法的运行时间主要由四个部分构成: 参数初始化、解的构建、局部优化和信息素的更新. 每个部分的时间复杂度如下: 1) 参数初始化的复杂度主要源于初始化信息素矩阵, 即: $\mathrm{O}(n^2)$. 2) 在解的构建过程中, 候选列表的初始化和选择机器的时间复杂度分别为 $\mathrm{O}(n)$和$\mathrm{O}(m)$. 构建候选列表, 计算状态转移概率并根据概率选择一个未调度的工件的总时间复杂度为$\mathrm{O}(n^2)$. 可以得出构建解的时间复杂度为$\mathrm{O}(n^2+mn)$. 因为n的值通常远大于m的值, 所以构建解的时间复杂度可以记为$\mathrm{O}(n^2)$. 3) 局部优化部分, 计算批的到达时间和按批的到达时间排序的时间复杂度分别为$\mathrm{O}(mn)$和$\mathrm{O}(mn^2)$. 因此局部优化策略的时间复杂度为$\mathrm{O}(mn^2)$. 4) 信息素更新的时间复杂度为$\mathrm{O}(n^2)$. 在蚂蚁完成解的构建之后, 调用局部优化策略, 因此每迭代一次的时间复杂度为$\mathrm{O}((N_1+N_2)mn^2)$. 故而算法 PDACO 的时间复杂度为$\mathrm{O}(T_{\max}(N_1+N_2)mn^2)$.

3　仿真实验及结果分析

为了评估本文所提算法的性能, 将其与经典的双目标优化算法 NSGA-II[23] 和 SPEA2[24], 粒子群优化算法 SMPSO[25] 以及一个解决相似问题的算法 PACO[15] 进行比较. 为了将 NSGA-II、SPEA2 和 SMPSO 用于求解我们的问题, 首先将工件集按照机器容量进行分类, 随后根据 Best-Fit-LPT (BFLPT) 规则对每个子工件集按照对应的机器容量进行分批, 再分别利用这两个算法在机器上调度生成的批.

3.1　实验设计

比较算法性能的测试算例采用随机方法生成. 11 组算例的工件数分别为 90, 108, 126, 144, 162, 180, 216, 252, 306, 360, 432, 每组有 20 个测试实例. 工件的加工时间均匀分布于 [8, 48][15]. 令 S^i 表示第 i 类机器容量. 假设机器总数为 10, 机器容量有 3 种: $S^1=10$, $S^2=25$, $S^3=65$. 考虑到实际应用中, 容量大的机器通常价格较高, 因此实验中大容量机器的数量相对较少, 从而设置每种容量对应的机器数分别为 5, 3, 2. J 中工件根据机器容量分为 3 个子集, 即 $J=J^1\cup J^2\cup J^3$, 其中 J^1 的工件可以在所有的机器上加工, J^2 的工件可以在容量不小于 S^2 的 5 台机器上加工, J^3 只能在容量为 S^3 的 2 台机器上加工, $J^1\sim J^3$ 的工件数分别设为 $2n/3$, $2n/9$ 和 $n/9$, 以使$\left|J^1\right|>\left|J^2\right|>\left|J^3\right|$[26].

工件尺寸是测试算例的一个关键参数, 通常基于均匀分布或正态分布来生成, 这样获得小尺寸和大尺寸工件的概率相同. 由于大尺寸工件往往会单独成批使得问题变得相对简单, 因此本文参照文献 [26] 的方式生成工件尺寸, 具体为

$$\left\{s_j|J_j\in J^i\right\}\sim P(\lambda_i),\ \lambda_i=\frac{S^i}{2},\ i=1,2,3$$

从而保证同一组工件中小尺寸工件的数量比大尺寸工件多, 可以将更多的小尺寸工件分配到其他较大尺寸工件所在的批中, 以达到增加求解的复杂度的目的. 进一步, 考虑到工件尺寸需要与机器相容, 则有:

$$s_j=\begin{cases}S^{i-1}, & s_j<S^{i-1}\\ s_j, & S^{i-1}\le s_j\le S^i\\ S^i, & s_j>S^i\end{cases}\tag{27}$$

其中, $S_0=1$. 实验中使用的相关参数设置如表 3.

表 3　实验参数设置

Table 3　Experimental parameter settings

参数	符号	取值
工件数	n	$n\in\{90, 108, 126, 144, 162, 180, 216, 252, 306, 360, 432\}$
工件尺寸	s_j	$s_j\sim P(\lambda_i), \lambda_1=5, \lambda_2=12.5, \lambda_3=32.5$
工件到达时间	r_j	$U[1, R]$
工件加工时间	p_j	$U[8, 48]$
机器数	m_i	$m_1=5$, $m_2=3$, $m_3=2$
机器容量	S^i	$S^1=10$, $S^2=25$, $S^3=65$
机器功率	l^i	$l^1=10$, $l^2=35$, $l^3=85$

表 3 中, U 表示均匀分布, R 表示下界值, R 的计算过程参见文献 [27].

本文提出的算法及对比算法的参数设计如表 4.

表 4　比较算法的参数设置

Table 4　The parameter settings of comparative algorithms

PDACO	PACO	NSGA-II	SPEA2	SMPSO
$N_1:50$	$N:100$	$N:100$	$N:100$	$N:100$
$N_2:50$		$Q_a:100$	$Q_a:100$	$r_1\in[0,1], r_2\in[0,1]$
		交叉概率: 1.0	交叉概率: 1.0	交叉概率: 0.9
		变异概率: 0.01	变异概率: 0.01	变异概率: $1/L$
$T_{\max}=200$	$T_{\max}=200$	$T_{\max}=200$	$T_{\max}=200$	$T_{\max}=200$

NSGA-II 和 SPEA2 的参数值参照原文, 其中, Q_a 表示外部存档集的大小, 保存搜索过程中非支配解的最大个数. PDACO 算法中信息素蒸发率 $\rho = 0.25$, 最大迭代次数设置为 $T_{\max} = 200$. 同时考虑到偏好向量的使用对实验结果的影响, 为了公平地比较不同算法所得解的质量, 对比算法中也引入了偏好向量. 在对比算法中, 将偏好向量作为机器选择的一个参数. PDACO 算法其他参数的设置在实验部分具体介绍, 其中 SMPSO 算法的变异概率为 $1/L$, L 表示工件数, r_1 和 r_2 是输入参数.

3.2 评价指标

为了评价算法的性能, 选用以下常用的性能指标:

1) 非劣解的数量 (NPS): 表示算法所得非支配解的数量.

2) 覆盖率 (C) :这一指标用于区分集合 E 和 F [24]. $C(E,F)$ 的值代表集合 F 中至少被集合 E 中的一个解支配的解所占的比例, $C(E,F)$ 的计算式为

$$C(E,F) = \frac{|\{f \in F \,|\exists e \in E : e \succ f\}|}{|f|} \tag{28}$$

$C(E,F)$ 的取值范围为 $[0,1]$, 其值越接近于 1, 则 F 中有更多的解被 E 中的解支配, 说明集合 E 中解的质量比集合 F 中的更好. 需要注意的是, 这一指标的值是不对称的, 即通常 $C(E,F) \neq 1 - C(F,E)$. 所以 $C(E,F)$ 和 $C(F,E)$ 需要分别计算.

3) 超体指标 (H): 该指标描述的是非支配解集的多样性和收敛性[24]. 对于集合 E 和 F, 如果 E 支配 F, 那么 E 的超体值 $H(E)$ 一定比 F 的超体值 $H(F)$ 大, 且 $H(E)$ 的值越大, 集合 E 越接近 Pareto 前沿.

超体指标的参考点是被所有算法的解支配的点, 通常采用式 (29) 的方式生成[15]

$$x_s = \max_s + \phi(\max_s - \min_s) \tag{29}$$

其中, $s = \{mk, tec\}$, $\max_s$ 和 $\min_s$ 分别代表在目标 s 上所有算法获得的解的最大值和最小值, 且 $\phi = 0.1$.

4) 覆盖区域 (DVR): 该指标代表解的覆盖区域, 可由式 (30) 计算

$$DVR_\Omega = \left(\max_{\xi\in\Omega} C^{\xi}_{\max} - \min_{\xi\in\Omega} C^{\xi}_{\max}\right) \times \left(\max_{\xi\in\Omega} TEC^{\xi} - \min_{\xi\in\Omega} TEC^{\xi}\right) \tag{30}$$

5) 解集均匀性 (SPC) : 该指标描述非支配解沿着 Pareto 前沿的分布情况, 可采用式 (31) 进行计算

$$SPC_\Omega = \frac{\left[\dfrac{1}{|\Omega|}\displaystyle\sum_{\xi\in\Omega}\left(d_\xi - \bar{d}\right)^2\right]^{\frac{1}{2}}}{\bar{d}} \tag{31}$$

其中, d_ξ 是解 ξ 和 Pareto 前沿 Ω 上最近的解之间的欧氏距离, $\bar{d} = \dfrac{1}{\Omega}\displaystyle\sum_{\xi\in\Omega} d_\xi$.

6) 运行时间 (T): 该指标用于比较在相同运行环境下, 不同算法的时间性能.

3.3 实验结果

仿真实验主要包 3 个部分, 即参数值确定及其影响分析、算法策略有效性验证以及不同算法的性能比较. 通过实验确定的实验参数值均在后续实验中采用.

3.3.1 参数确定及影响分析

本文提出的 PDACO 算法主要涉及以下参数: 种群规模 N_1 和 N_2 、信息素更新方式选择参数 K 、信息素蒸发率 ρ 、信息素更新式 (15) 中使用的参数 Q 以及信息素和启发式的相关重要系数 α_1, β_1, α_2, β_2 . 随机选取最大工件规模, 即 432 个工件的 10 个测试实例组成参数测试实例组. 因为超体指标 (H) 能够同时评价解的多样性和收敛性, 所以将其作为参数值确定实验的评价指标. 具体测试过程如下: 1) 每一个测试实例单独运行 20 次, 基于 Pareto 支配关系将 20 次运行的结果合并成为一个非支配解集; 2) 计算每个测试实例在对应参数取值下的超体值; 3) 计算在对应参数取值下, 10 个测试实例获得的平均超体值并将其作为算法在测试集上的超体值.

图 4 给出 N_1 , N_2 , k , ρ 和 Q 在不同取值下得到的 H 值. 为了测试种群规模对算法性能的影响, 将 (N_1, N_2) 分别取值为 (20, 80), (40, 60), (50, 50), (60, 40), (80, 20) 进行测试. 测试所得超体值如图 4 中的 (N_1, N_2) 组从左到右的 5 个条形柱所示. 从图 4 可以看出, 两个种群的规模越接近, 算法的性能越好, 且当两个种群的规模相同时算法的性能最好. 图 4 中 rho 组条形柱分别表示 ρ 取值为 0.05, 0.1, 0.15, 0.25, 0.35 时的结果. 可以看出, 当 $\rho = 0.25$, H 值最大, 此时算法表现出最好的性能. 此外, 当 $\rho < 0.25$ 时, 算法得到的 H 值几乎没有变化. 整体上看, ρ 的取值对算法性能的影响不大. 根据 k 的不同取值结果可以看出, k 的不同取值对算法性能的影响也不大. 图 4 中 K 组 5 个条形柱分别对应 K 取值为 10, 20, 30, 50, 70 时的结果. 由图可

以看出, 当 K 等于 30 或 50 时算法的性能最好, 而且 $K = 30$ 时的超体值略大于 $K = 50$ 时的结果, 即当 $K = 30$ 时, 算法的性能最好. 图 4中的 Q 组条形柱分别对应基于工件数设置的一组 Q 值, 即 216, 432, 468, 864, 1 080, 得到的结果. 由图 4可看出, Q 的取值对算法性能的影响不大, 但当 $Q = 432$ 时, 所得超体值最好. 综上, 实验参数取值如下: $N_1 = N_2 = 50$, $K = 30$, $\rho = 0.25$, $Q = 432$.

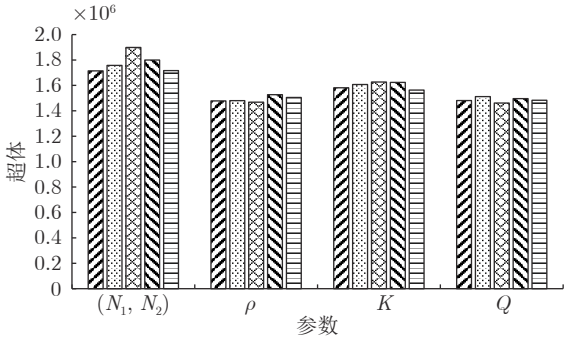

图 4　(N_1 , N_2), K , ρ, Q 不同取值获得的 H 值

Fig. 4　The values of H under different values of (N_1 , N_2), K , ρ and Q

图 5 给出不同 (α_1 , β_1) 和 (α_2 , β_2) 取值得到的超体值. 因为信息素的值小于 1, 为了使信息素能发挥指导作用, α 取值于 [0, 1]. 类似地, 启发式信息的值大于 1, 为了协同发挥启发式信息和信息素的作用, β 取值于 [1, 10]. 选用上述参数测试实例组, 对 α 和 β 的不同取值组合后进行仿真实验. 将 α 和 β 的比值设置为质数进行参数值选择,得到 α 和 β 的取值如下: (1, 1), (1, 3), (1/3, 1), (1, 5), (1/5, 1), (1,7), (1/7, 1), (1, 9), (1/9, 1). 如图 5 所示, (α_1, β_1) 组表示蚁群 ACO-N1 中信息素和启发式信息相对重要系数 (α_1, β_1) 分别取上述 9 种不同取值时获得的超体值. 从实验结果可以看出, 当 $\alpha_1 = 1/7$, $\beta_1 = 1$ 时, 算法获得最大的超体值. 类似地, (α_2, β_2) 组表示蚁群 ACO-N2 中信息素和启发式信息的相关重要程度 (α_2, β_2) 在上述 9 组不同取值下的实验结果. 从图 5 可以看出, 当 $\alpha_2 = 1$, $\beta_2 = 1$ 时, 在测试实例组上获得的超体值相对较大, 而且不同的 α_1和 β_1的取值对超体值影响相对较小. 基于上述讨论, 这 4 个参数在后续实验中分别取值为: $\alpha_1 = 1/7$, $\beta_1 = 1$, $\alpha_2 = 1$, $\beta_2 = 1$. 值得注意的是, 图 5 中 (α_1, β_1) 和 (α_2, β_2) 是两组独立实验. 这两组实验分别得到不同的 Pareto 参考前沿, 根据第 3.2 节中的超体计算公式计算超体值时, 所选的参考点不同, 导致图 5 中两组实验所得超体值的范围明显存在差异.

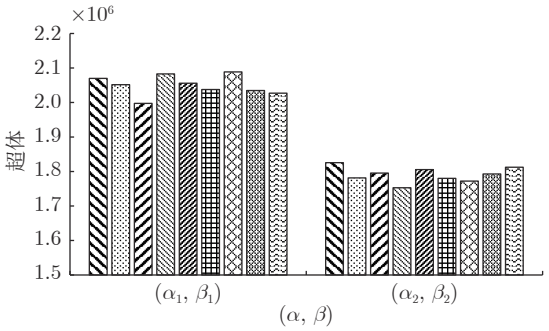

图 5　(α_1, β_1) , (α_2, β_2) 不同取值获得的 H 值

Fig. 5　The values of H under different values of (α_1, β_1) , (α_2, β_2)

3.3.2　策略性能分析

为了分析本文提出的改进策略对算法性能的影响, 本节分别针对算法的编码、信息素更新、启发式信息、候选列表的构建、决策规则以及局部优化 6 个部分进行测试. 在测试某一个策略对算法性能的影响时, 其他策略保持不变. 采用的测试实例组与第 3.3.1 节相同, 且每个测试实例运行 20 次并计算平均结果进行比较. 图 6 给出了采用和不采用优化策略的比较实验结果, 其中, Local、Code、Update、Choice、Heuristic、Candidate 分别表示局部优化策略、编码方式、信息素更新方式、决策规则、启发式信息和候选列表对应的实验结果. 图 6中每组左右两个条形柱分别表示本文算法采用和不采用该策略的超体值, 其计算过程如下: 首先产生每个测试实例 20 次独立运行得到的非支配解集, 计算采用和不采用某种策略 (如采用局部优化策略和不采用局部优化策略) 时在某个测试实例上得到的超体值, 再计算在组中 10 个测试实例的平均超体值, 并以此为最终结果. 如在 Local 组的结果中, 左边和右边的条形柱分别表示采用和不采用本文提出的局部优化策略得到的结果. 从图 6 可以看出, 局部优化策略对算法求解质量的影响较大, 左边的超体值接近右边的 2 倍. 这是因为分批时只针对批的当前状态进行工件选择, 但从整体上来看可能并不是最优的选择. 在整个解构建完成之后, 利用局部优化策略将一些满足条件的工件进行调整可以显著提高解的质量. 为了分析编码方式对算法性能的影响, 将本文算法的编码方式换成工件编码, 利用 BFLPT 规则进行分批得到的结果与本文算法进行比较. 图 6 中 Code 组左边和右边分别是采用本文中基于批的编码和采用工件编码得到的超体值. 可以看出, 采用批编码的算法在本文问题上获得更好的性能. 这可能是因为基于批编码能够同时获得工件所在机器

和批的信息, 不需要利用其他的启发式进行分批. 本文迭代地采用单个蚁群获得的最优解和到目前为止两个蚁群共同得到的最优解更新信息素, 为了分析这种更新方式对算法性能的影响, 将该更新方式与通常只采用当前代最优解更新信息素的方法进行比较, 比较结果如图 6 中 Update 组所示. 实验结果表明, 采用迭代进行更新的方式能够提高算法的性能. 这是因为更新时, 每只蚁群能够根据自己的搜索经验沿自己的搜索方向搜索质量更高的解, 有利于提高算法的多样性. 为了分析本文的机器选择规则的效果, 将机器的当前完工时间与当前能耗相加之和作为另一种机器选择方法来进行对比, 实验结果如图 6 的 Choice 组, 其中左边和右边分别为本文算法和对比策略的实验结果. 从图 6 可以看出, 采用本文的决策规则获得了更大的超体值, 说明本文采用的决策规则能够提高算法的性能. 启发式信息用于指导蚂蚁搜索解, 这里比较带有和不带有与浪费空间相关的启发式信息的两个算法, 实验结果表明本文设计的启发式信息对算法性能的影响较大, 也说明基于浪费空间的启发式信息能够有效提高解的质量. 为了更好地降低搜索成本, 本文针对问题设计了多个不同的候选列表, 这里将本文设计的候选列表与只考虑容量约束设计的候选列表进行比较, 实验结果如图 6 中 Candidate 组所示. 实验结果表明, 本文采用的候选列表能够有效提高算法的性能.

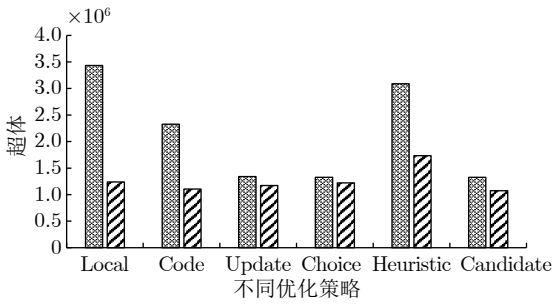

图 6 采用不同策略时获得的 H 值

Fig. 6 The values of H using different strategies

根据图 6 的实验结果及其分析, 可以看出本文提出的策略有效提高了算法的整体性能, 其中对算法性能影响最大的是编码、启发式信息和局部优化策略的设计.

3.3.3 比较实验结果与分析

表 5 列出了 5 种算法得到的每组实例的非支配解个数的平均数. 表中第 1 列对应测试实例的工件数, 第 2~4, 5~7, 8~10, 11~13, 14~16 列分别表示由算法 PDACO、PACO、NSGA-II、SPEA2 和 SMPSO 在每组实例上获得的非支配解的最大值 MAX , 最小值 MIN 和平均值 AVG 的平均值. 每组算例包含 20 个测试实例, 每个测试实例运行 20 次. 首先计算每个测试实例 20 次运行的非支配解数量的最大 (最小或平均) 值, 然后计算同一组内的 20 个实例的非支配解的平均最大 (最小或平均) 值. 非支配解的数量越多, 表明多目标优化算法的性能越好. 表 5 中每组的最佳平均结果 (每行中的最大值) 均以粗体显示.

从表 5 可以看出, 除了在工件数为 144 的算例组上 PDACO 的 *NPS* 值比 PACO 的大之外, 在所有测试实例上, PDACO 的 *NPS* 值比 NSGA-II、SPEA2 和 SMPSO 的都要大. 随着工件数的增大,

表 5 *NPS* 指标值比较结果

Table 5 Comparison of the five algorithms using the *NPS* metric

工件数	PDACO			PACO			NSGA-II			SPEA2			SMPSO		
	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN	AVG
90	**11.10**	**3.20**	**6.80**	10.40	2.65	6.30	7.75	1.00	2.74	4.25	1.00	1.22	8.4	1.2	4.26
108	**11.45**	**3.00**	**7.00**	10.05	2.75	6.36	7.85	1.00	2.92	5.65	1.00	1.51	7.95	1.4	4.07
126	**11.40**	**3.80**	**7.24**	10.50	2.75	6.28	7.75	1.05	2.84	6.95	1.00	1.73	8.2	1.3	3.99
144	12.85	3.65	7.62	**19.15**	**7.40**	**13.04**	8.80	1.00	3.02	8.05	1.00	1.92	7.9	1.35	4
162	**11.50**	**3.60**	**7.39**	10.50	3.30	6.63	9.70	1.00	3.05	6.95	1.00	1.73	7.35	1.25	3.93
180	**12.35**	**3.45**	**7.40**	11.40	2.55	6.64	8.95	1.00	2.95	7.70	1.00	2.11	8.35	1.25	4.15
216	**12.40**	**3.90**	**7.75**	11.40	3.20	7.04	8.55	1.00	2.71	8.20	1.00	2.11	7.95	1.2	4.03
252	**12.45**	**3.75**	**7.87**	11.35	3.55	7.16	9.25	1.00	2.71	8.60	1.00	2.01	7.6	1.4	4.02
306	**12.35**	**3.60**	**7.72**	11.30	3.20	7.00	8.45	1.00	2.55	8.50	1.00	2.00	8.1	1.3	4.11
360	**12.75**	**3.60**	**7.79**	11.85	3.35	7.40	7.75	1.00	2.42	9.05	1.00	2.06	8.15	1.3	4.21
432	**13.15**	**4.00**	**8.10**	12.15	3.25	7.61	8.05	1.00	2.39	7.65	1.00	1.76	8.2	1.25	4.04

在 MAX , MIN 和 AVG 上, PDACO 算法与NSGA-II、SPEA2 和 SMPSO 算法相比, 优势越来越明显, 尤其是在 MIN 和 AVG 上更为显著. 同时, 从表 5 的实验结果可以看出 PACO 算法的性能也优于 NSGA-II、SPEA2 和 SMPSO 算法. 且从整体的实验统计数据来看, PDACO 算法与 PACO 算法相比, 在 NPS 指标上的结果相近, 然而从所有测试实例组的平均值来看, PDACO 优于 PACO 算法, 差值分别为 2.8%, 4.2% 和 1.4%. 由此可知, PDACO 的 NPS 指标明显优于其他四种算法, 这是因为 PDACO 算法中两个蚁群分别针对不同方向搜索, 搜索范围更广泛, 在一定程度上降低了陷入局部最优的可能性, 从而提高了解的多样性.

表 6 给出了 5 种算法的平均覆盖率, 其中每个算法的非劣解集由每个实例运行 20 次得到的非支配解组合而成. 表 6 的第 1 列的定义与表 5 相同, 每一列的数值是对应每个实例组 20 个实例, 每个实例运行 20 次的结果的 C 指标的平均值. 从表 6 的统计数据可以看出, PDACO 算法明显优于其他的算法. 虽然在小规模测试集上, PDACO 算法中的解存在被其他算法支配的情况, 但随着工件数的增加, 所得的解几乎完全占优其他四个算法的解. 因为 PDACO 算法分别针对两个目标值进行搜索, 进一步扩大了搜索范围, 从而保证算法搜索到质量更好的解的可能性. 表 6 的实验结果也验证了本文算法的有效性.

表 7 给出了 5 个算法在每组实例上的超体值. 超体指标是一个评价算法的多样性、收敛性、与真实 Pareto 前沿近似程度的综合性指标. 表 7 中第 1 列是工件规模, 第 2~6 列分别对应算法 PDACO, PACO, NSGA-II, SPEA2 和 SMPSO 算法的超体指标值. 由表 7 可以看出, 在所有的测试实例中,

表 6　覆盖率 (C) 比较结果

Table 6　Comparison of the five algorithms using the C metric

工件数	C (PDACO, PACO)	C (PACO, PDACO)	C (PDACO, NSGA-II)	C (NSGA-II, PDACO)	C (PDACO, SPEA2)	C (SPEA2, PDACO)	C (PDACO, SMPSO)	C (SMPSO, PDACO)
90	**0.988**	0.001	**0.801**	0.001	**0.757**	0	**0.833**	0.021
108	**0.971**	0.003	**0.608**	0.001	**0.564**	0.001	**0.743**	0.004
126	**0.986**	0.002	**0.656**	0	**0.599**	0	**0.647**	0
144	**0.994**	0	**0.504**	0	**0.476**	0	**0.561**	0
162	**0.997**	0	**0.421**	0	**0.677**	0	**0.587**	0
180	**0.999**	0	**0.554**	0	**0.533**	0	**0.533**	0
216	**0.995**	0	**0.470**	0	**0.577**	0	**0.516**	0
252	**0.997**	0	**0.421**	0	**0.426**	0	**0.452**	0
306	**0.997**	0	**0.529**	0	**0.645**	0	**0.513**	0
360	**0.989**	0	**0.606**	0	**0.553**	0	**0.459**	0
432	**0.997**	0	**0.501**	0	**0.439**	0	**0.466**	0

表 7　超体 (H) 指标比较结果

Table 7　Comparison of the five algorithms using the H metric

工件数	PDACO	PACO	NSGA-II	SPEA2	SMPSO
90	**469 603**	277 319	110 110	111 396	121 352
108	**650 187**	403 180	146 403	148 960	147 856
126	**933 105**	539 363	256 380	236 164	271 332
144	**1 232 318**	690 126	275 805	303 290	288 495
162	**1 472 585**	793 386	318 719	358 388	341 864
180	**1 814 061**	1 000 950	413 532	465 956	428 793
216	**2 577 210**	1 385 263	560 037	671 642	635 549
252	**3 489 851**	1 874 024	749 649	888 686	815 992
306	**4 969 991**	2 614 455	1 079 066	1 314 258	1 179 523
360	**6 638 616**	3 478 054	1 555 105	1 878 950	1 638 628
432	**8 805 782**	4 469 622	2 106 786	2 439 180	2 266 835

PDACO的平均 H 值均大于其他三种算法的 H 值, 并且 PDACO 算法的 H 指标值几乎是 PACO 算法 H 指标值的两倍, 是 NSGA-II、SPEA2 和 SMPSO 算法 H 指标值的 4 倍. 显然 PDACO 的超体指标比其他四种算法更好, 这意味着 PDACO 算法得到的解更接近实际的 Pareto 前沿.

表 8 给出 5 个算法的 DVR , SPC 和 T 指标值. 类似地, 表 8 每一行的数据是每组 20 个实例, 每个实例运行 20 次的平均值. 对于每一个指标, 每组的最好值均以粗体显示. DVR 和 SPC 的值越大, 说明解的质量越好. 表 8 中, 第 2, 5, 8, 11 和 14 列是算法的平均 DVR 值. 可以看出, PDACO 算法得到的平均 DVR 值比 PACO、NSGA-II、SPEA2 和 SMPSO 都要好, 表明在所有的测试实例上 PDACO 算法得到的解均优于这四种算法.

表 8 的第 3, 6, 9, 12 和 15 列是 SPC 指标值. PDACO 算法除了在工件数为 360 的测试实例组上得到的 SPC 值比 PACO 的小之外, 在其他的测试实例上, PDACO 算法得到的 SPC 的值均大于 PACO 算法的 SPC 值. 与 NSGA-II、SPEA2 和 SMPSO 的比较结果可以看出, PDACO 算法在工件规模为 126 的测试实例组上得到的平均 SPC 值小于 NSGA-II. 除此之外, 在其他算例组上, PDACO 算法的 SPC 值比 NSGA-II 算法的 SPC 值大. 总体来说, PDACO 算法在 DVR 和 SPC 的两个指标上的性能均优于另外四个算法. 表 8 中第 4, 7, 10, 13 和 16 列给出了 T 指标值. 可以看出, PDACO 除了在 90 和 144 个工件的测试实例上的运行时间比 PACO 的运行时间长之外, 在其他测试实例组上, PDACO 的运行时间都要短. 此外, PDACO 算法在所有测试实例组上的平均计算时间都比 NSGA-II、SPEA2 和 SMPSO短. 因此可以得出, 随着工件规模的增大, PDACO算法的时间优势越明显. 因为在 PDACO 算法中通过候选列表有效减小搜索范围, 达到了降低计算时间的目的.

通过实验结果可以看出, 本文提出的基于蚁群的多目标协同优化算法 PDACO 能够搜索到质量更好的解, 针对两个目标分别进行搜索的方式可以保证解的多样性.

图 7 中给出了 4 组不同工件数的测试实例的目标函数图, 以 "1-1" 的形式表示, 第 1 个数值表示工件数, 第 2 个值表示对应实例组中的编号. 从图 7 可以看出, 随着工件数的增加, PDACO 算法的解的多样性和收敛性较其他算法更好.

为了验证本文所提算法的正确性, 图 8 给出本文算法求解测试集中含有 90 个工件的一个算例的调度解的甘特图, 其中横坐标表示时间, 纵坐标表示 10 台机器, 不同机器对应的纵坐标宽度表示机器的容量. 从图 8 可以看出, 该解的所有批均满足机器容量约束, 且批的加工过程满足机器同一时刻只能加工一个批的约束, 同时也满足一个批只能在一台机器上加工的约束. 图 8 中 90 个工件分为 39

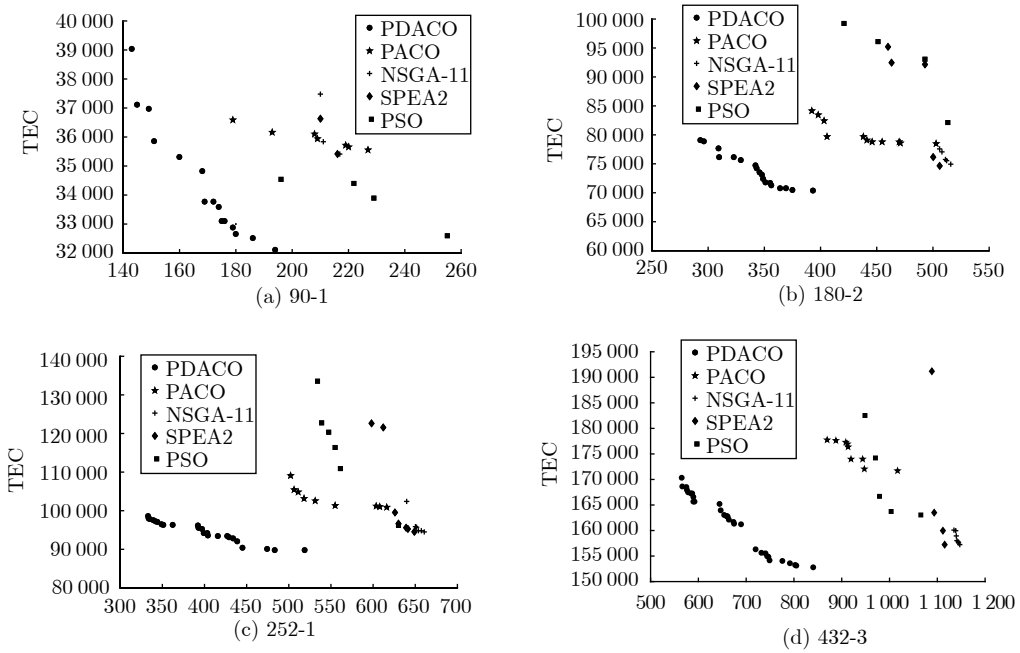

图 7 4 组不同工件数测试实例目标函数

Fig. 7 Solution distribution on four instances with different number of jobs

表 8　多样性 (DVR)、Spacing(SPC) 和时间 (T) 指标比较结果

Table 8　Comparison of the five algorithms using the DVR, SPC and T metrics

工件数	PDACO			PACO			NSGA-II			SPEA2			SMPSO		
	DVR	*SPC*	*T*	*DVR*	*SPC*	*T*	*DVR*	*SPC*	*T*	*DVR*	*SPC*	*T*	*DVR*	*SPC*	*T*
90	**137 262**	**0.89**	0.86	89 010	0.82	**0.83**	17 167	0.26	1.87	4 121	0.04	2.17	21 362	0.36	1.14
108	**164 060**	**0.89**	**1.08**	110 360	0.84	1.13	20 772	0.29	2.39	17 299	0.11	2.96	26 815	0.33	1.28
126	**219 865**	0.92	**1.44**	141 594	0.86	1.49	46 860	**1.01**	2.88	28 803	0.13	3.93	52 619	0.52	1.5
144	**256 752**	**0.91**	1.80	166 727	0.84	**1.65**	24 153	0.29	3.61	59 443	0.22	4.65	39 526	0.23	1.7
162	**294 754**	**0.88**	**2.30**	197 631	0.86	2.53	24 276	0.28	4.43	54 872	0.14	5.62	43 511	0.22	1.83
180	**328 732**	**0.92**	**2.67**	213 904	0.83	3.17	27 072	0.26	5.07	102 915	0.27	6.59	27 342	0.24	2.28
216	**435 913**	**0.94**	**3.83**	290 015	0.90	4.40	21 623	0.25	17.00	133 770	0.27	9.01	25 113	0.24	2.29
252	**520 652**	**0.93**	**5.05**	370 128	0.90	5.94	23 940	0.22	9.43	179 343	0.23	12.87	34 619	0.25	2.57
306	**664 941**	**0.94**	**7.36**	451 475	0.89	8.53	24 801	0.24	20.53	293 842	0.27	21.12	33 428	0.24	3.07
360	**800 856**	0.91	**10.33**	535 670	**0.93**	12.33	23 838	0.19	27.58	431 781	0.28	27.80	49 734	0.26	3.46
432	**1 010 000**	**0.96**	**14.77**	789 834	0.94	17.54	23 959	0.17	36.70	374 310	0.19	40.10	66 253	0.21	4.03

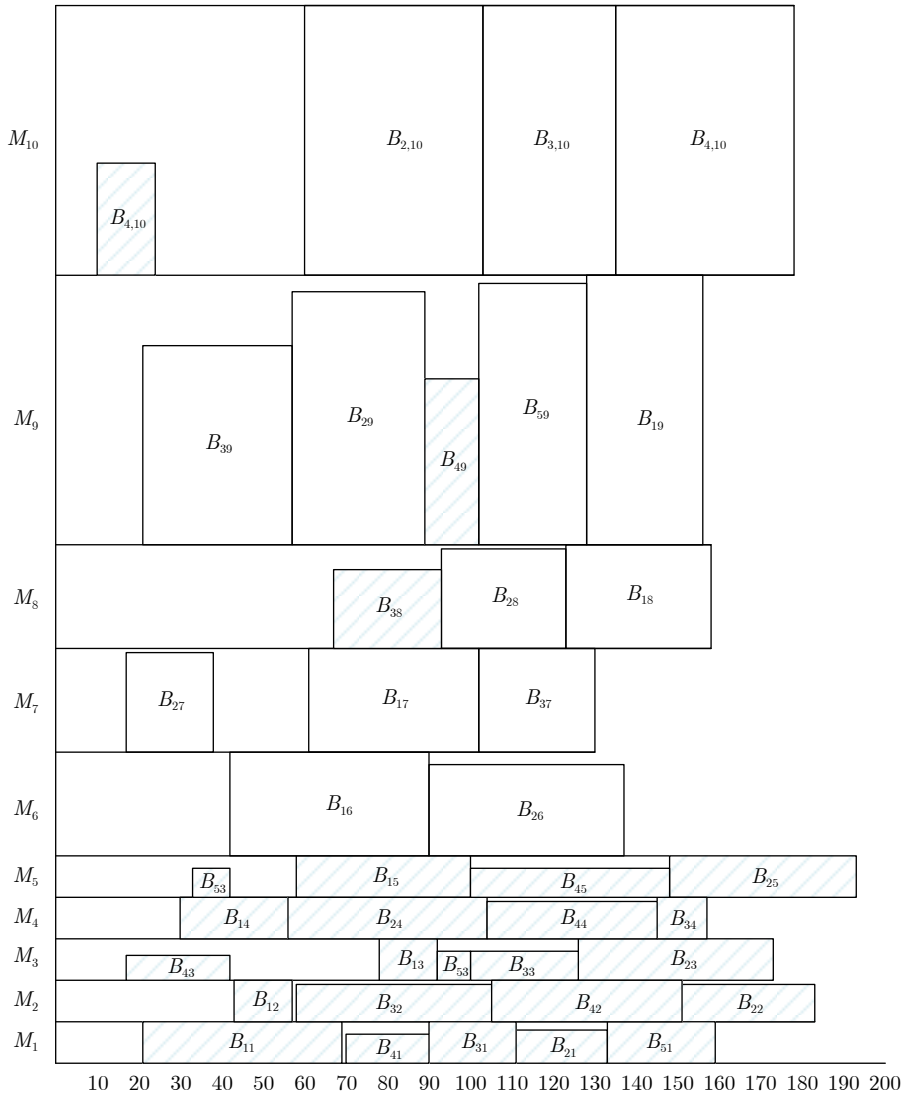

图 8　90 个工件实例调度甘特图

Fig. 8　The Gantt chart of the example with 90 jobs

个批, 调度结束得到的两个目标值分别为: $C_{\max} = 193$, $TEC = 39\,125$.

4 结论

本文讨论了具有动态到达时间、作业尺寸不同的差异容量并行 BPM 调度问题, 并提出 PDACO 算法来最小化完工时间和总能耗. 提出的算法针对每个目标值设置一个蚁群进行搜索, 同时利用候选列表缩小搜索空间. 为了提高搜索效率, 在每个蚁群内针对不同的目标设置启发式信息来指导搜索. 为了验证算法的性能, 从不同角度如非支配解的多样性、运行时间和非支配解的分布性等, 将所提算法与其余三个算法进行比较. 仿真实验结果表明, 所提的 PDACO 算法明显优于其他算法, 且算法 PDACO 能够在更加合理的时间内获得质量较好的解, 从而验证了 PDACO 的有效性. 在后续的研究中, 我们将从如何平衡解的多样性和分布均匀性方面着手进一步提高解的质量, 同时也将考虑如何更好地实现蚁群间的协同进化, 更好地实现独立搜索与联合搜索的平衡. 另一方面我们也考虑将协同进化的方法应用于求解其他类型的问题. 例如, 流水车间调度问题、云制造环境中资源分配等问题.

References

1 Ahmadi J H, Ahmadi R H, Dasu S, Tang C S. Batching and scheduling jobs on batch and discrete processors. *Operations research*, 1992, **40**(4): 750−763

2 Yuan Hao-Nan, Guo Ge. Vehicle cooperative optimization scheduilng intransportation cyber physical systems. *Acta Automatica Sinica*, 2019, **45**(1): 143−152
(原豪男, 郭戈. 交通信息物理系统中的车辆协同运行优化调度. 自动化学报, 2019, **45**(1): 143−152)

3 Uzsoy R. Scheduling a single batch processing machine with non-identical job sizes. *International Journal of Production Research*, 1994, **32**(7): 1615−1635

4 Ikura Y, Gimple M. Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, 1986, **5**(2): 61−65

5 Zhao Yu-Fang, Tang Li-Xin. Scheduling a single continuous batch processing machine to minimize makespan. *Acta Acta Automatica Sinica*, 2006, **32**(5): 730−737
(赵玉芳, 唐立新. 极小化最大完工时间的单机连续型批调度问题. 自动化学报, 2006, **32**(5): 730−737)

6 Zhou H M, Pang J H, Chen P K, Chou F D. A modified particle swarm optimization algorithm for a batch-processing machine scheduling problem with arbitrary release times and non-identical job sizes. *Computers and Industrial Engineering*, 2018, **123**: 67−81

7 Chang P Y, Damodaran P, Melouk S. Minimizing makespan on parallel batch processing machines. *International Journal of Production Research*, 2004, **42**(19): 4211−4220

8 Jia Z H, Wang Y, Wu C, Yang Y, Zhang X Y, Chen H P. Multi-objective energy-aware batch scheduling using ant colony optimization algorithm. *Computer and Industrial Engineering*, 2019, **131**: 41−56

9 Shi Z S, Huang Z W, Shi L Y. Customer order scheduling on batch processing machines with incompatible job families. *International Journal of Production Research*, 2018, **56**(1-2): 795−808

10 Zhou S C, Xie J H, Du N, Pang Y. A random-keys genetic algorithm for scheduling unrelated parallel batch processing machines with different capcities and arbitrary job sizes. *Applied Mathematics and Computation*, 2018, **334**: 254−268

11 Arroyo J E C, Leung J Y T. Scheduling unrelated parallel batch processing machines with non-identical job sizes and unequal ready times. *Computers and Operations Research*, 2017, **78**: 117−128

12 Lu Zhi-Qiang, Liu Xin-Yi. Algorithm for resource-constrained project scheduling problem with resource transfer time. *Acta Automatica Sinica*, 2018, **44**(6): 1028−1036
(陆志强, 刘欣仪. 考虑资源转移时间的资源受限项目调度问题的算法. 自动化学报, 2018, **44**(6): 1028−1036)

13 Zhou S C, Li X L, Du N, Pang Y, Chen H P. A multi-objective differential evolution algorithm for parallel batch processing machine scheduling considering electricity consumption cost. *Computers and Operations Research*, 2018, **96**: 55−68

14 Du B, Chen H P, Huang G Q, Yang H D. Preference vector ant colony system for minimising make-span and energy consumption in a hybrid flow shop. *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing*. Springer-London, 2011. 279–304

15 Jia Z H, Zhang Y L, Leung J Y T, Li K. Bi-criteria ant colony optimization algorithm for minimizing makespan and energy consumption on parallel batch machines. *Applied Soft Computing*, 2017, **55**: 226−237

16 Wang Gong-Shu, Liu Jing-Yi, Tang Li-Xin. Branch-and-price algorithm for rolling batch scheduling problem in continuous-casting and rolling processes with hybrid production mode. *Acta Automatica Sinica*, 2017, **43**(7): 1178−1189
(汪恭书, 刘静宜, 唐立新. 连铸–轧制混流生产模式下轧批调度问题的分支–定价算法. 自动化学报, 2017, **43**(7): 1178−1189)

17 Lang Jin, Tang Li-Xin. Batch scheduling problem of oil well considering ramping constraints in oilfield production. *Acta Automatica Sinica*, 2019, **45**(2): 388−397
(郎劲, 唐立新. 考虑爬坡约束的油井间抽批调度问题. 自动化学报, 2019, **45**(2): 388−397)

18 Zhao B X, Gao J M, Chen K, Guo K. Two-generation Pareto ant colony algorithm for multi-objective job shop scheduling problem with alternative process plans and unrelated parallel machines. *Journal of Intelligent Manufacturing*, 2018, **29**(1): 93−108

19 Wu Y, Gong M G, Ma W P, Wang S F. High-order graph

matching based on ant colony optimization. *Neurocomputing*, 2019, **328**: 97−104

20 Eskandari L, Jafarian A, Rahimloo P, Baleanu D. A modified and enhanced ant colony optimization algorithm for traveling salesman problem. *Mathematical Methods in Engineering*. Springer-Cham, 2019. 257–265

21 Stützle T, Hoos H H. MAX-MIN ant system. *Future Generation Computer Systems*, 2000, **16**(8): 889−914

22 Jia Z H, Leung J Y T. A meta-heuristic to minimize makespan for parallel batch machines with arbitrary job sizes. *European Journal of Operational Research*, 2015, **240**(3): 649−665

23 Deb K, Agrawal S, Pratap A, Meyarivan T. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Proceedings of the 2000 International Conference on Parallel Problem Solving from Nature. Berlin, Heidelberg: Springer, 2000. 849–858

24 Zitzler E, Laumanns M, Thiele L. SPEA2: Improving the strength Pareto evolutionary algorithm. *TIK-Report*, 2001, 103

25 Nebro A J, Durillo J J, Garcia-Nieto J, Coello C C A, Luna F, Alba E. SMPSO: A new PSO-based metaheuristic for multi-objective optimization. In: Proceedings of the 2009 IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making. IEEE, 2009. 66–73

26 Wang J Q, Leung J Y T. Scheduling jobs with equal-processing-time on parallel machines with non-identical capacities to minimize makespan. *International Journal of Production Economics*, 2014, **156**: 325−331

27 Jia Z H, Li X H, Leung J Y T. Minimizing makespan for arbitrary size jobs with release times on P-batch machines with arbitrary capacities. *Future Generation Computer Systems*, 2017, **67**: 22−34

贾兆红 安徽大学计算机科学与技术学院教授. 2008 年获得中国科技大学博士学位. 主要研究方向为计算智能, 多目标优化, 决策支持. 本文通信作者.
E-mail: jiazhaohong001@163.com
(**JIA Zhao-Hong** Professor at the School of Computer Science and Technology, Anhui University. She received her Ph.D. degree from University of Science and Technology of China in 2008. Her research interest covers intelligent computation, multi-objective optimization, and decision support. Corresponding author of this paper.)

王 燕 安徽大学硕士研究生. 主要研究方向为多目标优化, 进化计算.
E-mail: yanwang0501@outlook.com
(**WANG Yan** Master student at the School of Computer Science and Technology, Anhui University. Her research interest covers intelligent optimization and evolutionary computation.)

张以文 安徽大学计算机科学与技术学院教授. 主要研究方向为服务计算, 云计算, 电子商务.
E-mail: zhangyiwen@ahu.edu.cn
(**ZHANG Yi-Wen** Professor at the School of Computer Science and Technology, Anhui University. His research interest covers service computing, cloud computing, and e-commerce.)