一种差异工件单机批调度问题的蚁群优化算法①

王栓狮, 陈华平, 程八一, 李 燕
(中国科学技术大学管理学院, 合肥 230026)

摘要: 由于在利用蚁群算法构建差异工件(即工件有尺寸差异)单机批调度问题的解时, 批的加工时间是不确定的, 从而不能类似于经典调度问题的蚁群算法把批加工时间的倒数作为蚁群算法中的启发式信息, 引入批的利用率和批的负载均衡率作为蚁群算法中的启发式信息, 提出了 JACO(ant colony optimization based a job sequence)和 BACO (ant colony optimization based a batch sequence)两种蚁群优化算法. 在算法 JACO 中, 解的编码为工件序列, 它对应着用 BF (best fit)分批规则生成的调度方案, 信息素代表工件间的排列顺序; 在算法 BACO 中, 解的编码为批序列, 信息素代表工件间的批相关性, 由此信息素通过中间信息素量来构造相应的解, 并引入特定的局部优化策略, 提高了算法的搜索效率. 实验表明, 与以往文献中的 SA(simulated annealing)、GA(genetic algorithm)算法以及 FFLPT(first-fit longest processing time)、BFLPT (best-fit longest processing time)启发式规则相比, 算法 JACO和 BACO明显优于它们, 且 BACO 算法比 JACO算法效果更好.

关键词: 调度; 批处理机; 蚁群优化算法; 组合优化
中图分类号: TP29 文献标识码: A 文章编号: 1007- 9807(2009)06- 0072- 11

0 引 言

0.1 差异工件批调度问题

调度问题是组合优化领域中的一类重要问题, 在生产管理与调度、网络通信等方面有着广泛的应用. 批调度问题是它的重要分支, 与经典调度不同的是, 批调度中 1台机器可以同时处理多个工件而非仅仅 1个工件, 批的加工时间等于此批中工件的最大加工时间, 批的完工时间等于批的开始时间加上批的加工时间, 同一批中所有工件的完工时间等于所在批的完工时间. 生产中有很多这样的例子, 如半导体工业中的芯片预烧工序或者金属加工业的热处理工序, 前者, 机器的容量定义为它所能容纳的工件的个数, 后者, 每个工件有其特定的容量需求, 批中工件的总尺寸不能超过批的容量限制, 因此, 包含在每一批中的工件个数可能不同. 在实际生产生活中, 此类问题的例子还有很多, 如陶瓷烧制、港口货物装卸、汽车货运等. 从工件尺寸的角度来讲, 前者可以称为同类工件的批调度问题, 而后者可以称为差异工件的批调度问题. 目前, 关于第一类问题已有相当多的文献研究, 而在差异工件批调度问题上的研究较少. 本文主要研究了目标为极小化最大完工时间 $C_{\max}$ 的差异工件单机批调度问题.

为描述问题, 作如下假设:

(1) 有 n个待加工工件 $\{1, 2, \cdots, n\}$, 工件 i的加工时间为 p_i, 工件 i的尺寸为 s_i;

(2) 机器容量为 B; 每一批中工件的总尺寸小于等于 B, 假设没有尺寸大于机器容量的工件;

(3) 假设一旦 1个批开始加工, 在该批加工完成之前, 不能被中断且不能再添加新的工件; 批

① 收稿日期: 2007- 06- 18; 修订日期: 2008- 03- 17.
基金项目: 国家自然科学基金资助项目: (70671096); 国家杰出青年基金(B类)资助项目 (70629002).
作者简介: 王栓狮(1978—), 男, 山西绛县人, 硕士, 助理研究员. Email: luckywss@mail.ustc.edu.cn

j的加工时间 T_j 等于批中工件的最大加工时间;

(4) 目标是使得最大完工时间 $C_{\max}$ 最小. 其中, 最大完工时间 $C_{\max}$ 等于最后一个离开机器的工件的完工时间.

目标为极小化 $C_{\max}$的差异工件单机批调度问题采用 3参数表示法, 可记为 $1 \mid B, s_j \mid C_{\max}$. 基于上述的假设, 可建立问题 $1 \mid B, s_j \mid C_{\max}$的数学模型如下:

$$\text{Minimize } C_{\max} = \sum_{j=1}^{k} T_j \tag{1}$$

$$\text{s.t.} \sum_{j=1}^{k} x_{ij} = 1, \quad i = 1, \cdots, n \tag{2}$$

$$\sum_{i=1}^{n} s_i x_{ij} \leqslant B, \quad j = 1, \cdots, k \tag{3}$$

$$T_j \geqslant p_i x_{ij}, \quad i = 1, \cdots, n \quad j = 1, \cdots, k \tag{4}$$

$$x_{ij} \in \{0, 1\}, \quad i = 1, \cdots, n \quad j = 1, \cdots, k \tag{5}$$

$$\left\lceil \sum_{i=1}^{n} s_i / B \right\rceil \leqslant k \leqslant n, \quad k \in Z^+ \tag{6}$$

式 (2) 保证了每个工件 i只能被安排在一个批 j 中; 式 (3) 保证了批中工件的总尺寸不能超过机器的容量限制; 式 (4) 为批加工时间约束; 式 (5)、(6) 为决策变量, $x_{ij} = 1$说明工件 i被安排在批 j 中, 否则 $x_{ij} = 0$, k为总批数, 其中, $\left\lceil \sum_{i=1}^{n} s_i / B \right\rceil$ 为最小批数, n 为最大批数.

除了上面模型中出现的一些记号外, 为了表达方便, 再给出下文中将要用到的几个变量和函数:

B_j: 表示批 j中的工件标号集; $\mid B_j \mid$ 表示批 j 中的工件数.

$s(B_j)$: 表示批 j的尺寸, 等于批 j中工件的总尺寸, 是 B_j 的函数; 即 $s(B_j) = \sum_{i \in B_j} s_i$.

$p(B_j)$: 表示批 j的加工时间, 等于批 j中工件加工时间的最大值, 是 B_j 的函数; 即 $p(B_j) = \max_{i \in B_j} \{p_i\}$. 当构成调度方案后, $T_j = p(B_j)$.

0 2　差异工件批调度问题的相关研究结果

Uzsoy[1] 证明了当所有工件的加工时间相同时, 问题 $1 \mid B, s_j \mid C_{\max}$等价于箱子容量为 B、物品尺寸为 s_i 的装箱问题, 由于装箱问题是强 NP难的, 因此, 问题 $1 \mid B, s_j \mid C_{\max}$ 也是强 NP难的, $1 \mid B, s_j \mid \sum C_i$也被证明了是 NP难的.

他还指出, $1 \mid B, s_j \mid C_{\max}$ 问题的下界可以通过求解如下的松弛问题得到: 把工件松弛为可按尺寸拆分且可放在不同的批中加工的批调度问题. 松弛问题的一个实例为: 加工时间为 p_j、尺寸为 s_j 的工件被松弛为 s_j 个加工时间为 p_j 的单位尺寸工件; 此问题是同类工件的单机批调度问题, 由以下方法可得到其最优解: 先以加工时间非增序排列工件, 然后连续 B 个工件组成 1批, 以任意顺序加工批; 由此得到的解为原问题的下界[1]. 上面的下界可以得到如下的改进: 令

$$J = \{ i \mid B - s_i < \min\{s_1, s_2, \cdots, s_n\} \}$$

则集合 J 中的工件只能单独成批; 先求出只能单独成批的工件的加工时间, 再加上其他工件按上述松弛方法得到的加工时间, 由此得到的值为原问题的改进下界, 即:

$$C_n^{\mathrm{LB}} = \sum_{j \in J} p_j + C_{n-|J|}^{\mathrm{relax}} \tag{7}$$

其中, $C_{n-|J|}^{\mathrm{relax}}$ 是按上述方法所得的松弛问题的最优解.

针对问题, $1 \mid B, s_j \mid \sum C_i$, Uzsoy[1] 提出了几个启发式算法; Ghazvini和 Dupont[2] 给出了两个新的启发式算法. 针对问题 $1 \mid B, s_j \mid C_{\max}$, Uzsoy[1] 根据装箱问题的 first-fit[3] 启发式规则给出了几个启发式规则算法, 其中, FFLPT的性能最好, 张国川等在文献[4]中证明了 FFLPT算法的近似比小于等于 2; Dupont[5] 等给出了另外两个启发式算法 BFLPT 和 SKP(successive knapsack problem), BFLPT的性能要优于 FFLPT; Dupont 和 Flipo[6] 提出了该问题的分枝定界算法; Melouk[7] 和 Damodaran[8] 分别利用模拟退火算法 (SA) 和遗传算法 (GA) 求解了此问题, 但只与求解线性规划的商业软件 CPLEX作了比较, 未与具有较好性能的 FFLPT和 BFLPT启发式规则算法作比较.

0 3　蚁群算法在调度问题中的应用

蚁群算法是由 Colomi[9] 等首先提出的一种人工智能算法, 它是基于对自然界蚂蚁觅食过程的观察而产生的类启发式优化算法. 蚁群算法的基本思想及具体的算法描述见文献[10].

自 Dorigo等首次将蚁群算法应用于 TSP问题以来, 国内外学者对其进行了大量研究并将其推

广应用到其他组合优化问题,取得了较好的效果. 利用蚁群算法求解组合优化问题, 关键是确定解的编码、信息素含义及问题的启发式信息,以及给定信息素和启发式信息后如何按解的编码方式随机产生解.

类似于求解 TSP问题, 应用蚁群算法求解调度问题[11~22], 一般需要先将调度问题图形化表示,如 flow shop问题可以表示为结点模式或弧模式的图[11, 12], Job Shop可以用析取图或类似于析取图的模型来表示[12~18], 然后再确定问题的信息素含义和启发式信息. 表 1总结了以往文献中,利用蚁群算法求解调度问题时所采用的解的编码、信息素含义及问题的启发式信息:

表 1 蚁群算法在调度问题中的应用

Table 1 Applications of ACO algorithm in scheduling problems

解的编码	首先将问题图形化,解的编码即为图中的一条路径
信息素含义	1、弧模式: 信息素 τ_{ij} 对应于弧 (i, j) 上的信息素浓度[11~17, 22]. 弧 (i, j) 反映了同一工件的不同工序之间的顺序约束或同一机器上加工的不同工件工序之间的占用约束; 2、结点模式: 信息素 τ_{ij} 对应于结点 O_{ij} 上的信息素浓度[11, 12, 19]. 结点 O_{ij} 反映了工序 j 位于工序处理序列的第 i 个位置, 或者工序 j 在机器 i 上加工.
启发式信息	1、无启发式信息[11, 12, 17, 19, 20] 2、工序加工时间的倒数[12~16, 18]

批调度问题要远远复杂于经典调度问题, 可以把它看作是由分批和批的加工两个子问题组成的. 分批子问题即是在满足批容量约束的前提下把工件分为多个批次; 批加工子问题是在工件分批后, 安排这些批在机器上加工, 此时可以把一个批看作一个工件, 这样, 第二个子问题即是经典调度问题 (与经典调度不同的是, 这里批的加工时间是不确定的, 它由第一个子问题的解给出). 从而, 批调度问题不易直观地用图来描述, 且由于批加工时间的不确定性, 也就不能类似于经典调度问题的蚁群算法把批加工时间的倒数作为蚁群算法中的启发式信息. 文献[18] 和文献[21] 对蚁群算法在批调度问题中的应用做了研究, 但文献[18] 只是利用蚁群算法求解了批调度的第二个子问题 (即经典调度问题), 第一个子问题的解是利用问题的启发式信息给出的; 文献[21] 所采用的解的编码方案和信息素含义与本文提出的三种方案 (见表 2) 中的第二种相同, 且没有用到启发式信息.

1 两种蚁群算法

本文提出了 3种编码方案及相应的信息素含义 (见表 2), 并针对问题的特性, 引入批的利用率和批的负载均衡率两个指标作为蚁群算法中的启发式信息指引蚂蚁的搜索过程; 依照方案 1和方案 3设计了针对问题 $1 \mid B, s_j \mid C_{\max}$ 的 JACO和 BACO两种蚁群算法, 并在 BACO算法中设计了局部优化策略 (下文中, 各算法的信息素矩阵统一用 $(\tau_{ij})_{n\times n}$ 来表示).

1.1 三种编码方案及相应的信息素含义

表 2 三种编码方案及相应的信息素含义

Table 2 Three coding schemes and pheromone trails for a solution

	解的编码	信息素含义	对应的算法
方案 1	工件序列 (代表着用某一特定分批规则生成的调度方案)	τ_{ij} 表示工件序中工件 j 排列在工件 i 之后的信息素浓度	JACO
方案 2	工件 1到 n 对应的批号序列 (一个调度方案中最多有 n 个批, 故批号取值为 1到 n)	τ_{ij} 表示工件 i 被安排到批 j 中的信息素浓度	文献[21] 中算法
方案 3	调度方案 (即批序列)	τ_{ij} 表示工件 i 和工件 j 被安排在同一批中的信息素浓度	BACO

实验表明, 在方案 1和方案 2中, 当问题规模较小时, 方案 2较好, 但随着规模的增大, 方案 2将明显差于方案 1. 其原因在于方案 2没有考虑批的特征, 带有一定的盲目性, 致使其搜索空间随着规模的增大而成指数增长, 其搜索空间的规模为 n^n; 而方案 1的搜索空间规模为 $n!$, 在 n 较小时, 搜索空间相当, 但方案一受所选用的分批规则的限制不能完全遍历整个解空间, 而方案 2更容易遍历整个解空间, 故方案 2较优. 但随着 n 的增

大, 方案 2的搜索空间成指数增长, 在相同时间内, 方案 2找到较优解的概率较小.

方案 1和方案 2都有一定的优势和劣势, 可以将它们的优势互补, 得到考虑分批和工件序相结合的方案 3

1 2　启发式信息 – 批的利用率和批的负载均衡率

显然, 在给定一组工件的前提下, 使得批的总剩余空间越小的方案越优.

其次, 由于批的加工时间等于批中工件加工时间的最大值, 因此, 在批加工完之前, 可能有部分工件已经加工完, 但又不能把这些工件交付使用. 如果把工件已加工完但不能交付使用的时间称为工件的冗余时间, 显然, 这部分时间是被白白浪费掉了, 因此使得总冗余时间越小的方案将可能越优. 由批的加工时间和工件冗余时间的定义可以知道, 批中工件的加工时间越平均, 总冗余时间将越小, 当批中工件的加工时间全相等时, 总冗余时间为零.

定义 **1**　批的利用率定义为批的尺寸 (即批中工件尺寸之和) 所占机器容量的比例. 计算公式如下

$$UR_k = \frac{s(B_k)}{B} = \frac{\sum_{i \in B_k} s_i}{B}$$

定义 **2**　批的负载均衡率定义为批中工件加工时间的变差系数与 1的差值. 计算公式如下

$$BR_k = 1 - \frac{\sigma_k}{\omega_k} = 1 - \frac{\sqrt{\frac{1}{|B_k|}\sum_{i \in B_k}(p_i - \omega_k)^2}}{\omega_k}$$

其中: $\omega_k = \frac{1}{|B_k|}\sum_{i \in B_k} p_i$ 为批 k中工件加工时间的平均值; σ_k 为批 k中工件加工时间的方差. $\frac{\sigma_k}{\omega_k}$ 为批 k中工件加工时间的变差系数. 批的负载均衡率越大, 批中工件加工时间的变异系数越小, 说明批中工件的加工时间分布越均匀.

批的利用率和批的负载均衡率具有以下特点:

1) 不可公度性 因为利用率和负载均衡率是分别从工件尺寸和工件加工时间两个不同的方面定义的, 因此, 它们之间没有统一的衡量标准, 难以比较.

2) 矛盾性 在给定一组工件的前提下, 使批利用率提高的调度方案可能会使其负载均衡率降低.

一个调度方案中各个批之间的利用率和负载均衡率也可能存在矛盾性. 提高了某一个批的利用率和负载均衡率的调度方案可能会降低另一个批的利用率和负载均衡率.

一般而言, 批的利用率和负载均衡率的平均值越大的调度方案将越好, 但并不是对所有实例问题的所有解都成立, 如 10个工件的实例

i	0	1	2	3	4	5	6	7	8	9
p_i	5	3	17	7	5	13	2	4	18	19
s_i	3	9	2	10	2	6	2	1	10	4

存在两个可行调度方案 S1: {9, 5} – {8} – {2, 0, 4, 6, 7} – {3} – {1} 与 S2 {9, 2, 4, 7} – {8} – {5, 0} – {3} – {1} – {6}; S1的平均批利用率为 (1 + 1 + 1 + 1 + 0. 90) /5 = 0. 98, 大于 S2的平均批利用率 (0. 90 + 1 + 0. 90 + 1 + 0. 90 + 0. 20} /6 = 0. 82, S1的平均负载均衡率为 (0. 81 + 1 + 0. 41 + 1 + 1) /5 = 0. 844, 大于 S2的平均负载均衡率 (0. 40 + 1 + 0. 56 + 1 + 1 + 1) /6 = 0. 826, 但 S1的最大完工时间为 64, 大于 S2的最大完工时间 62, 即方案 S1劣于方案 S2 如何更好地综合利用批的利用率和批的负载均衡率来衡量一个批的好坏以及调度方案的优劣, 有待进一步研究.

1 3　两种蚁群算法的数学描述

1. 3. 1　JACO 算法

1) 信息素含义　采用方案 1的信息素表示, 即 τ_{ij} 表示工件序中工件 j排列在工件 i之后的信息素浓度. 分批规则采用已知的具有最好性能的 BF分批规则.

2) 启发式信息　算法中的 BF分批规则, 实质上就是利用了批的利用率最大这个启发式信息.

在生成工件序列后, 利用 BF分批规则进行分批时, 工件序中间隔较近的工件被分在一批的可能性较大, 为了计算方便, 对应于批的负载均衡率, 定义工件序中工件 j排列在工件 i之后的启发式信息为

$$\eta_{ij} = \frac{1}{1 + |p_i - p_j|}$$

3) 状态转移概率公式 记 $allowed_a = \{j \mid$ 工件 j未被蚂蚁 a 访问到$\}$, 设蚂蚁 a 当前所在位置为工件 i, 蚂蚁 a 按如下的状态转移概率选择下一个工件

$$p_{ij}^{a} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\eta_{ij}^{\beta}}{\sum_{l \in allowed_a} \tau_{il}^{\alpha}\eta_{il}^{\beta}}, & j \in allowed_a \\ 0, & j \notin allowed_a \end{cases} \tag{8}$$

4) 信息素更新 采用蚁环 (ant-cycle) 算法中离线式的信息素更新方式, 即在所有蚂蚁都得到一个解之后按如下公式更新信息素

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}$$

$$\Delta\tau_{ij} = \sum_{a=1}^{m} \Delta\tau_{ij}^{a} \tag{9}$$

其中, ρ ($0 < \rho < 1$) 是信息素挥发系数.

$\Delta\tau_{ij}^{k}$ 按下式计算得到

$$\Delta\tau_{ij}^{a} = \begin{cases} \dfrac{Q}{C_{\max}^{a}}, & \text{若第 } a \text{ 只蚂蚁生成的工件} \\ & \text{序中, 工件 } j \text{ 在工件 } i \text{ 后} \\ 0, & \text{其他} \end{cases} \tag{10}$$

其中, $C_{\max}^{a}$ 为第 a 只蚂蚁构造的调度方案的目标值.

5) 局部优化策略 蚁群优化算法中可以加入局部搜索机制提高每次迭代的效率.

可以对每只蚂蚁生成的工件序利用变异交叉等技术进行局部搜索, 但都是随机搜索, 改进效果不太显著, 故此算法中未引入局部优化策略.

1.3.2　BACO算法

1) 信息素含义　采用方案 3的信息素表示, 即 τ_{ij} 表示工件 i和工件 j被安排在同一批中的信息素浓度. 为了利用工件间的信息素构造解, 需要增加中间值

$$\vartheta_{kj} = \frac{1}{|B_k|}\sum_{i \in B_k} \tau_{ij}, \quad j \in \{1, 2, \cdots, n\} / B_k$$

ϑ_{kj} 表示工件 j加入批 k的信息素浓度, 是工件 j与已安排在批 k中的工件间的信息素的平均值.

2) 启发式信息 为了计算方便, 对应于批的利用率和负载均衡率, 定义工件 j加入批 k的两个启发式信息为

$$\gamma_{kj} = s_j \text{ 和 } \kappa_{kj} = \frac{1}{1+|\omega_k - p_j|}$$

其中: $j \in \{1, 2, \cdots, n\} / B_k$; ω_k 为已安排在批 k中的工件加工时间的平均值.

3) 状态转移概率公式 记 $allowed_a = \{j \mid$ 工件 j未被蚂蚁 a 访问到$\}$, 设蚂蚁 a 当前所在位置为批 k, 记 $\Phi_k^a = \{j \mid j \in allowed_a$ 且 $s_j + s(B_k^a) \leqslant B\}$, 表示可以放入当前批 k的工件标号集, 如果 $\Phi_k^a = \Phi$, 则蚂蚁 a 在 $allowed_a$ 中任选一工件开新批, 否则, 蚂蚁 a 按如下的状态转移概率选择下一个要加入当前批 k的工件:

$$p_{kj}^{a} = \begin{cases} \dfrac{\vartheta_{kj}^{\alpha}\gamma_{kj}^{\beta_1}\kappa_{kj}^{\beta_2}}{\sum_{l \in allowed_a} \vartheta_{kl}^{\alpha}\kappa_{kl}^{\beta_1}\gamma_{kl}^{\beta_2}}, & j \in \Phi_k^a \\ 0, & j \notin \Phi_k^a \end{cases} \tag{11}$$

其中, B_k^a 为蚂蚁 a 当前所在批 k 的工件标号集.

4) 信息素更新 同样采用公式 (9) 更新信息素. 此时, $\Delta\tau_{ij}^{a}$ 按下式计算得到

$$\Delta\tau_{ij}^{a} = \begin{cases} \dfrac{Q}{C_{\max}^{a}}, & \text{若第 } a \text{ 只蚂蚁生成的调度方案} \\ & \text{中, 工件 } j \text{ 和工件 } i \text{ 在同一批} \\ 0, & \text{其他} \end{cases} \tag{12}$$

其中, $C_{\max}^{a}$ 为第 a 只蚂蚁构造的调度方案的目标值.

5) 局部优化策略 在算法 BACO中, 依据解的编码与目标函数的关系, 采用了如下的局部优化策略:

假设每个批中的工件是按加工时间非增的顺序排列的, 对每个蚂蚁所得到的调度方案 (设总共有 k个批) 做如下改进:

步骤 **0**　令 $batchpos_1 = 0$;

步骤 **1**　按批加工时间非增序排列各批;

步骤 **2**　如果 $batchpos_1 = k$, 则停止, 输出当前批序列; 否则, 令 $batchpos_2 = batchpos_1 + 1$;

步骤 **3**　如果 $batchpos_2 = k$, 则删除当前调度方案中的空批, 更新批数 k, 令 $batchpos_1 = batchpos_1 + 1$, 转步骤 1; 否则, 如果批 $batchpos_2$中的第一工件能放入批 $batchpos_1$ 中, 则将批 $batchpos_2$中的第一工件移到批 $batchpos_1$ 中; 令

$batchpos_2 = batchpos_2 + 1$, 转步骤 3

1.4 两种蚁群算法的流程图

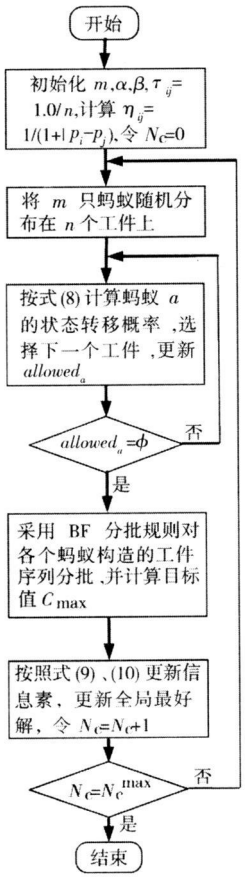

图 1　算法 JACO 流程图

Fig. 1 The flow chart of JACO algorithm

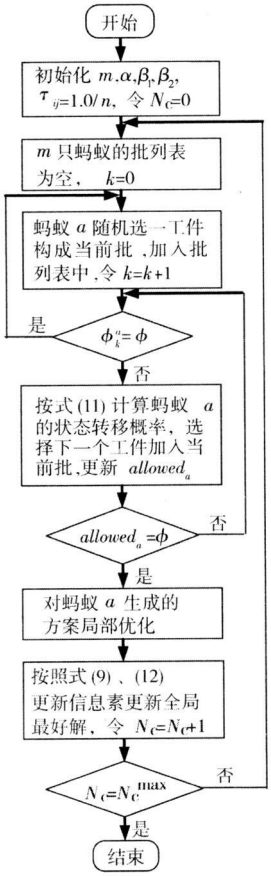

图 2　算法 BACO 流程图

Fig. 2 The flow chart of BACO algorithm

2　仿真实验与分析

2.1 仿真实例

为了检验本文中所提蚁群优化算法的有效性, 类似于 Melouk 等[7] 的方法生成随机测试实例. 为了覆盖不同种类的实例, 需要考虑 3个因素: 工件数的变化、工件加工时间的变化、工件尺寸的变化; 各因素记号、它们的等级划分及取值范围见表 3. 每一类实例的记号为 $Jipjsk(i = 1, 2, 3, 4; j = 1, 2; k = 1, 2, 3)$, 如 10个工件、工件加工时间服从 $[1, 10]$ 离散均匀分布、工件尺寸服从 $[2, 4]$ 离散均匀分布类实例用 J1p1s2表示. 在所有实例中, 假设机器的容量都为 10. 这里所选用的工件加工时间、工件尺寸及机器的容量代表了电子制造服务提供商的产品在环境应力筛选实验箱中的测试活动.

表 3　因素及各因素的等级划分

Table 3 Factors and levels

因素	等级及取值分布
工件数	J1类: 10个工件; J2类: 20个工件; J3类: 50个工件; J4类: 100个工件
工件加工时间	p1类: 服从 [1, 10] 离散均匀分布 p2类: 服从 [1, 20] 离散均匀分布
工件尺寸	s1类: 大小工件混合集 (服从 [1, 10] 离散均匀分布) s2类: 小工件集 (服从 [2, 4] 离散均匀分布) s3类: 大工件集 (服从 [4, 8] 离散均匀分布)

2.2 各算法参数设置

对蚂蚁数 $m = 10, 20, 30, n/1.5, n/2.0$分别做了实验, 当蚂蚁数大于 20时, 并没有明显地改善近似解, 反而增加了算法时间; 对其它参数也做了大量的组合实验, 最终选择了如下的参数组合:

表 4　两种蚁群算法的参数设置

Table 4 Parameter setting for JACO and BACO

参数 / 算法	Q	$\tau_{ij}(0)$	m	ρ	α	β	s1类实例		s2类实例		s3类实例	
							β_1	β_2	β_1	β_2	β_1	β_2
JACO	100	$1/n$	20	0.5	1	1	—	—	—	—	—	—
BACO	下界	$1/n$	20	0.5	1	—	2	1	1	3	3	1

其中, 对 s1类实例, $\beta_1 = 2, \beta_2 = 1$; 对 s2类实例, $\beta_1 = 1, \beta_2 = 3$; 对 s3类实例, $\beta_1 = 3, \beta_2 = 1$. 这是因为, s1类实例是一般情况, 工件由大小工件混合而成, 实验表明, 此时批的利用率比批的负载均衡率的重要程度要高; s2类实例中的工件是小尺寸的, 一个批中最少包含 2个工件, 最多可以包含 5个工件, 即批中的工件比较多, 工件加工时间分布不均匀的可能性较大, 因此批的负载均衡率显得就比较重要; 相反, s3类实例中的工件是大尺寸的, 从而有 40% 的工件单独成批, 另 60% 的工件可以两两成批, 即一个批最多由两个工件构成, 工件加工时间的分布大多数是均匀的, 因此可以进一步减小批的负载均衡率所起的作用.

下文中的 SA算法为 Melouk 等在文献 [7] 中提出的模拟退火算法.

下文中的 IGA 算法, 是在 Damodaran[8] 所提的 GA 算法中的选择部分增加了精英保留策略,

即首先把最好的解保留下来, 然后再基于 C_{max} 以竞争的方式选择父代进行交叉变异, 各参数取值不变. 这样改进的原因在于, Damodaran[8] 提出的GA算法并不能保证所得方案比 FFLPT、BFLPT规则所得到的方案优, 实验表明, 随着问题规模的增大, Damodaran[8] 提出的 GA 算法与 FFLPT、BFLPT 规则相比, 并不占优势.

算法 BACO、JACO、IGA、SA 均以迭代 80次为终止条件.

2.3 实验结果与分析

所有算法 BACO、JACO、IGA、SA、FFLPT、BFLPT 以及求解实例下界的算法均在 VC6.0平台下实现. 每一类 $Jipjsk(i = 1, 2, 3, 4; j = 1, 2; k = 1, 2, 3)$ 各生成 500个随机测试实例, 每个测试实例求解 15次取最好解. 算法 BACO 与其他 5个算法所得解的比较如下:

表 5 BACO与其他算法运行结果的比较

Table 5 Comparison between BACO algorithm and the other algorithms

		s1					s2					s3				
		SA	FFLPT	BFLPT	IGA	JACO	SA	FFLPT	BFLPT	IGA	JACO	SA	FFLPT	BFLPT	IGA	JACO
J1p1	优	**0.296**	**0.196**	**0.142**	**0.026**	**0.002**	**0.422**	**0.128**	**0.124**	**0.034**	**0.000**	**0.068**	**0.120**	**0.090**	**0.004**	**0.000**
	等 > 下	0.380	0.382	0.402	0.482	0.504	0.092	0.112	0.116	0.144	0.158	0.258	0.218	0.224	0.282	0.284
	等 = 下	0.324	0.422	0.456	0.492	0.494	0.486	0.760	0.760	0.822	0.842	0.674	0.662	0.686	0.714	0.716
	劣	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
J2p1	优	**0.882**	**0.518**	**0.394**	**0.248**	**0.024**	**0.812**	**0.386**	**0.358**	**0.176**	**0.012**	**0.462**	**0.374**	**0.274**	**0.158**	**0.000**
	等 > 下	0.114	0.422	0.528	0.654	0.860	0.050	0.190	0.208	0.260	0.338	0.342	0.398	0.474	0.548	0.658
	等 = 下	0.004	0.060	0.078	0.098	0.116	0.138	0.424	0.434	0.564	0.650	0.196	0.228	0.252	0.294	0.342
	劣	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
J3p1	优	**0.998**	**0.894**	**0.794**	**0.648**	**0.488**	**1.000**	**0.840**	**0.820**	**0.630**	**0.368**	**0.988**	**0.706**	**0.588**	**0.474**	**0.150**
	等 > 下	0.002	0.106	0.204	0.346	0.504	0.000	0.090	0.094	0.160	0.276	0.012	0.282	0.392	0.496	0.808
	等 = 下	0.000	0.000	0.000	0.004	0.004	0.000	0.070	0.086	0.210	0.356	0.000	0.012	0.020	0.030	0.042
	劣	0.000	0.000	0.002	0.002	0.004	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
J4p1	优	**1.000**	**0.992**	**0.930**	**0.868**	**0.830**	**1.000**	**0.988**	**0.984**	**0.898**	**0.790**	**1.000**	**0.888**	**0.794**	**0.726**	**0.574**
	等 > 下	0.000	0.008	0.064	0.122	0.160	0.000	0.008	0.012	0.072	0.144	0.000	0.112	0.206	0.272	0.414
	等 = 下	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.002	0.026	0.060	0.000	0.000	0.000	0.002	0.000
	劣	0.000	0.000	0.006	0.010	0.010	0.000	0.002	0.002	0.004	0.006	0.000	0.000	0.000	0.000	0.012
		s1					s2					s3				
		SA	FFLPT	BFLPT	IGA	JACO	SA	FFLPT	BFLPT	IGA	JACO	SA	FFLPT	BFLPT	IGA	JACO
J1p2	优	**0.358**	**0.208**	**0.158**	**0.040**	**0.000**	**0.482**	**0.158**	**0.150**	**0.060**	**0.000**	**0.076**	**0.136**	**0.084**	**0.004**	**0.000**
	等 > 下	0.410	0.424	0.452	0.552	0.588	0.112	0.128	0.130	0.174	0.216	0.318	0.266	0.284	0.346	0.350
	等 = 下	0.232	0.368	0.390	0.408	0.412	0.406	0.714	0.720	0.766	0.784	0.606	0.598	0.632	0.650	0.650
	劣	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
J2p2	优	**0.898**	**0.510**	**0.382**	**0.228**	**0.024**	**0.884**	**0.426**	**0.410**	**0.218**	**0.030**	**0.516**	**0.356**	**0.250**	**0.144**	**0.000**
	等 > 下	0.102	0.452	0.574	0.716	0.918	0.070	0.246	0.254	0.340	0.458	0.332	0.416	0.504	0.588	0.726
	等 = 下	0.000	0.038	0.044	0.056	0.058	0.046	0.328	0.336	0.442	0.512	0.152	0.228	0.246	0.268	0.274
	劣	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
J3p2	优	**1.000**	**0.922**	**0.788**	**0.676**	**0.486**	**1.000**	**0.874**	**0.846**	**0.646**	**0.484**	**0.982**	**0.706**	**0.586**	**0.436**	**0.140**
	等 > 下	0.000	0.078	0.212	0.320	0.486	0.000	0.092	0.114	0.254	0.388	0.018	0.294	0.412	0.558	0.830
	等 = 下	0.000	0.000	0.000	0.000	0.002	0.000	0.032	0.038	0.096	0.124	0.000	0.000	0.002	0.006	0.014
	劣	0.000	0.000	0.000	0.004	0.026	0.000	0.002	0.002	0.004	0.004	0.000	0.000	0.000	0.000	0.016
J4p2	优	**1.000**	**0.988**	**0.908**	**0.858**	**0.818**	**1.000**	**0.988**	**0.986**	**0.914**	**0.848**	**1.000**	**0.886**	**0.794**	**0.736**	**0.516**
	等 > 下	0.000	0.012	0.084	0.124	0.144	0.000	0.010	0.012	0.068	0.112	0.000	0.114	0.206	0.258	0.438
	等 = 下	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.002	0.002	0.000	0.000	0.000	0.000	0.000
	劣	0.000	0.000	0.008	0.018	0.038	0.000	0.002	0.002	0.016	0.038	0.000	0.000	0.000	0.006	0.046

表 5中的“优”表示优势实例所占比例, “等 > 下”表示 I类等效实例所占比例, “等 = 下”表示 II类等效实例所占比例, “劣”表示劣势实例所占比例. 其中, 优势实例用算法 BACO 求得的 C_{max} 小于其他算法所得的 C_{max}; I类等效实例用算法 BACO 求得的 C_{max} 等于其他算法所得的 C_{max} 但此

C_{max}大于实例的下界; Ⅱ类等效实例用算法 BACO求得的 C_{max}等于其他算法所得的 C_{max}且此 C_{max}等于实例的下界, 此时, 所得解一定是最优解; 劣势实例用算法 BACO求得的 C_{max}大于其他算法所得的 C_{max}.

为了描述 BACO算法相对于其他 5个算法的改进效果, 定义算法 BACO相对于算法 A的改进度为

$$I_A = \frac{C_{max}(A) - C_{max}(BACO)}{C_{max}(A)} \times 100$$

其中, A代表算法 JACO、IGA、SA、FFLPT、BFLPT. I_A大于 0表示算法 BACO要优于算法 A, 其值越大, 说明相对算法 A来说, BACO的改进效果越好. 表 6和图 3给出了算法 BACO相对于其他 5个算法的平均改进度.

表 6　BACO算法相对于其他 5个算法的改进度 (500个随机实例的平均值)

Table 6　Improvement ratioes of BACO algorithms and the other algorithms

	s1					s2					s3				
	I_{SA}	I_{FFLPT}	I_{BFLPT}	I_{IGA}	I_{JACO}	I_{SA}	I_{FFLPT}	I_{BFLPT}	I_{IGA}	I_{JACO}	I_{SA}	I_{FFLPT}	I_{BFLPT}	I_{IGA}	I_{JACO}
J1p1	1.747	1.369	0.962	0.097	0.006	3.849	0.857	0.843	0.180	0.000	0.286	0.715	0.591	0.027	0.000
J1p2	1.923	1.296	0.977	0.151	0.000	4.063	0.950	0.892	0.321	0.000	0.381	0.820	0.560	0.018	0.000
J2p1	4.444	2.232	1.548	0.877	0.052	5.128	1.704	1.579	0.717	0.032	1.252	1.314	0.904	0.344	0.000
J2p2	4.229	1.955	1.399	0.649	0.028	5.156	1.484	1.437	0.596	0.045	1.295	1.118	0.764	0.344	0.000
J3p1	7.311	2.515	1.765	1.185	0.563	7.184	2.405	2.295	1.331	0.513	3.081	1.308	0.984	0.674	0.108
J3p2	7.202	2.487	1.670	1.113	0.400	7.042	2.064	1.902	0.966	0.486	3.095	1.328	0.956	0.599	0.054
J4p1	9.105	2.419	1.614	1.280	0.858	7.700	2.515	2.420	1.500	0.913	4.474	1.280	0.996	0.754	0.302
J4p2	8.737	2.285	1.427	1.097	0.665	7.553	2.179	2.069	1.164	0.720	4.306	1.240	0.952	0.752	0.160

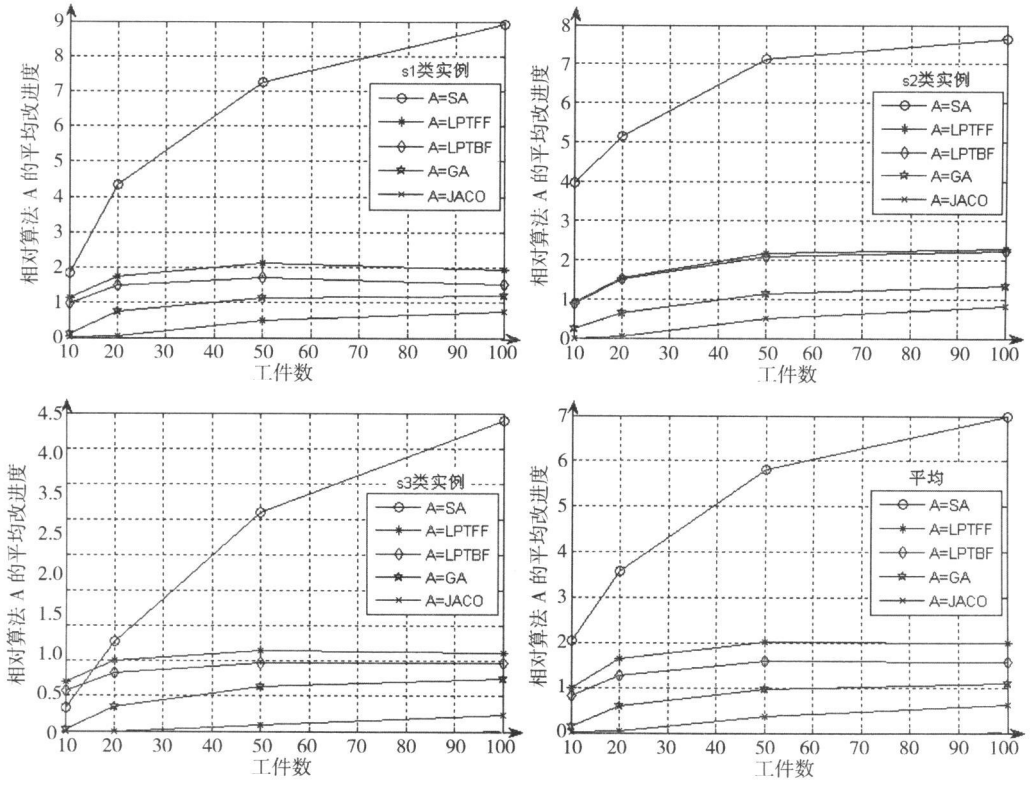

图 3　平均改进度的变化趋势

Fig. 3　Degree of improvement ratios change

从表 5、表 6以及图 3可以看出算法 BACO要优于其他 5个算法. 求解差异工件单机批调度问题的这 6个算法, 按求解效率由高到底依次是 BACO、JACO、IGA、BFLPT、FFLPT、SA.

算法 BACO用于求解 s3类实例时,没有求解 s1、s2类实例时的改进效果那么显著,其原因在于 s3类实例中的工件是大工件,一个批最多由两个工件构成,这样,同样的规模,s3类问题的解空间要比 s1、s2类问题的解空间小的多,从而,其他 5个算法也容易求得问题的较优解.

从表 6和图 3可以看出,随着问题规模的增大,改进度的变化越来越平缓,也就是说,改进度的提升并没有小规模时那么显著,甚至还有某些类实例的改进度出现下降,其原因在于随着问题规模的增大,改进度定义中的分母 $C_{\max}(A)$ 会显著增大.虽然改进度可能下降了,但这并不意味着 BACO算法随着规模的增大效率下降了,相反,由表 5中的优势实例所占比例可以看出,随着问题规模的增大,算法 BACO的优势更加明显.当问题规模为 100时,BACO算法 100% 绝对优于 SA算法,对 s1、s2类问题,BACO算法在 80% 以上的实例求解上绝对优于其他 5个算法,对 s3类问题,BACO算法与其他 5个算法相比,绝对优势比例也在 50% 以上.

由于蚁群算法也是一种随机搜索算法,随着问题规模的增大,解空间成指数地增长,在相同的迭代次数和实验次数的情况下,会出现某些实例的求解结果劣于其他某个算法的求解结果,但如果增加迭代次数或实验次数,这种情况将会减少甚至不出现.

3 总 结

本文中,针对差异工件单机批调度问题,引入批的利用率和批的负载均衡率作为蚁群算法中的启发式信息,给出了基于不同信息素表示的两种基本蚁群算法.实验结果表明,这两种蚁群算法是非常有效的,特别是对大规模问题,两种蚁群算法的优势更明显,而且,两种算法中,BACO算法的求解效果更好.

进一步的研究可以把两种蚁群算法推广应用到工件有交货期、工件动态到达或不相容工件族等差异工件单机批调度问题中,也可以应用到目标为完工时间和的此类问题中;也可以通过采用蚁群算法与其他算法(如模拟退火、遗传算法等)的混合、限制信息素的最大最小值、优化信息素的更新方式等方法进一步提高蚁群算法的效率(但这种改进效果并不太显著).如何更好地综合利用批的利用率和批的负载均衡率来衡量一个批的好坏以及一个调度方案的优劣,以及依据它们,设计出近似比接近于 3/2的启发式规则算法,也将是下一步的研究方向.

参 考 文 献:

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

[2] Jolai G F, Dupont L. Minimizing mean flow time criteria on a single batch processing machine with non-identical job sizes [J]. International Journal of Production Economics, 1998, 55(3): 273—280.

[3] Coffman E G, Garey M R, Johnson D S. Approximation algorithms for bin-packing- an updated survey[A]. In: Algorithm Design for Computer System Design[M]. Austria: Springer-Verlag, 1984. 49—106.

[4] Zhang G C, Cai X Q, Lee C Y, *et al*. Minimizing makespan on a single batch processing machine with non-identical job sizes[J]. Naval Research Logistics, 2001, 48(3): 226—247.

[5] Dupont L, Jolai G F. Minimizing makespan on a single batch processing machine with non-identical job sizes[J]. European Journal of Automation (JESA), 1998, 32: 431—440.

[6] Dupont L, Dhaenens F C. Minimizing the makespan on a batch machine with nonidentical job sizes: An exact procedure[J]. Computers & Operations Research, 2002, 29(7): 807—819.

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

[8] Damodaran P, Kumar M P, Srihari K. Minimizing makespan on a batch-processing machine with non-identical job sizes using genetic algorithms[J]. International Journal of Production Economics, 2006, 103(2): 882—891.

[9] Colorni A, Dorigo M, Maniezzo V, *et al*. Distributed Optimization by Ant Colonies[C]. Proceeding of the 1st European Conference on Artificial Life, 1991: 134—142.

[10]肖人彬, 陶振武. 群集智能研究进展[J]. 管理科学学报, 2007, 10(3): 80—96.
Xiao Ren-bin, Tao Zhen-wu. Research progress of swarm intelligence[J]. Journal of Management Sciences in China, 2007, 10(3): 80—96. (in Chinese)

[11]王笑蓉, 吴铁军. Flow shop问题的蚁群优化调度方法[J]. 系统工程理论与实践, 2003, 23(5): 65—71.
Wang Xiao-rong, Wu Tie-jun. An ant colony optimization algorithm for flowshop scheduling[J]. Systems Engineering—Theory & Practice, 2003, 23(5): 65—71. (in Chinese)

[12]姜　桦, 李　莉, 乔　非, 等. 蚁群算法在生产调度中的应用[J]. 计算机工程, 2005, 31(5): 76—78, 101.
Jiang Hua, Li Li, Qiao Fei, *et al*. Application of ant algorithm in manufacturing scheduling[J]. Computer Engineering, 2005, 31(5): 76—78, 101. (in Chinese)

[13]刘志刚, 李　言, 李淑娟. 基于蚁群算法的 Job-Shop多资源约束车间作业调度[J]. 系统仿真学报, 2007, 19(1): 216—220.
Liu Zhi-gang, Li Yan, Li Shu-juan. Multi-resource constrained Job-Shop optimization scheduling based on ant colony algorithm[J]. Journal of System Simulation, 2007, 19(1): 216—220. (in Chinese)

[14]陈知美, 顾幸生. 基于蚁群算法的不确定条件下的 Job Shop调度[J]. 山东大学学报(工学版), 2005, 35(4): 74—79.
Chen Zhi-mei, Gu Xing-sheng. Job Shop scheduling with uncertain processing time based on ant colony system[J]. Journal of Shandong University (Engineering Science), 2005, 35(4): 74—79. (in Chinese)

[15]杨阿莉. 一种改进蚁群算法在车间作业调度问题中的研究与应用[J]. 机械与电子, 2005, (4): 9—12.
Yang A-li. Study and application of an improved ant algorithm in job shop scheduling problem[J]. Machinery & Electronics, 2005, (4): 9—12. (in Chinese)

[16]孔凡国, 黄　伟. Job Shop调度问题自适应蚁群算法的研究[J]. 新技术新工艺, 2006, (5): 40—42.
Kong Fan-guo, Huang Wei. Research on Job Shop problem based on self-adapt ant colony algorithm[J]. New Technology & New Process, 2006, (5): 40—42. (in Chinese)

[17]王常青, 操云甫, 戴国忠. 用双向收敛蚁群算法解作业车间调度问题[J]. 计算机集成制造系统, 2004, 10(7): 820—824.
Wang Chang-qing, Cao Yun-fu, Dai Guo-zhong. Bi-directional convergence ACO for job-shop scheduling[J]. Computer Integrated Manufacturing Systems, 2004, 10(7): 820—824. (in Chinese)

[18]梁　静, 钱省三, 马　良. 基于双层蚂蚁算法的半导体炉管制程批调度研究[J]. 系统工程理论与实践, 2005, 25(12): 96—101.
Liang Jing, Qian Sheng-san, Ma Liang. Two-level ant algorithm for the furnace batch scheduling in semiconductor furnace operation[J]. Systems Engineering-theory & Practice, 2005, 25(12): 96—101. (in Chinese)

[19]高　尚, 钟　娟, 莫述军. 多处理机调度问题的蚁群算法[J]. 微型电脑应用, 2003, 19(4): 9—10, 16.
Gao Shang, Zhong Juan, Mo Shu-jun. An ant colony algorithm for the multiprocessor scheduling problem[J]. Microcomputer Applications, 2003, 19(4): 9—10, 16. (in Chinese)

[20]宋晓宇, 朱云龙, 尹朝万, 等. 应用混合蚁群算法求解模糊作业车间调度问题[J]. 计算机集成制造系统, 2007, 13(1): 105—109, 125.
Song Xiao-yu, Zhu Yun-long, Yin Chao-wan, *et al*. Hybrid ant colony algorithm for fuzzy Job Shop scheduling[J]. Computer Integrated Manufacturing Systems, 2007, 13(1): 105—109, 125. (in Chinese)

[21]马建华. 单机分批排序问题的变异蚁群算法[J]. 计算机工程与应用, 2006, 42(3): 53—56.
Ma Jian-hua. A mutation ant colony optimization algorithm of single batch machine scheduling problem[J]. Computer Engineering and Applications, 2006, 42(3): 53—56. (in Chinese)

[22]姚建明, 刘丽文, 蒲　云, 等. MC模式下供应链动态调度的蚁群寻优分析[J]. 管理科学学报, 2007, 10(3): 7—14.

Yao Jiarrming, Liu Li-wen, Pu Yun, *et al*. Analysis on ants optimization algorithm for supply chain dynamic scheduling in mass customization[J]. Journal of Management Sciences in China, 2007, 10(3): 7—14. (in Chinese)

Minimizing makespan on a single batch processing machine with non-identical job sizes using ant colony optimization

WANG Shuan-shi, CHEN Hua-ping, CHENG Ba-yi, LI Yan

School of Management, University of Science and Technology of China, Hefei 230026, China

Abstract In this paper, two ant colony optimization (ACO) algorithms are proposed to minimize makespan for scheduling jobs with non-identical sizes on a single batch processing machine. Compared with the traditional ACO algorithm, we design new heuristic information based on utilization ratio and load balance ratio of a batch for this problem. In the first algorithm named JACO (ant colony optimization based a job sequence), the solution is coded as a job sequence which is corresponding to a solution of the problem based on BBF (Batch Best Fit) heuristic, and whose pheromone trail is associated with the sequence of jobs. In the second algorithm named BACO (ant colony optimization based a batch sequence), the solution is coded as a batch sequence which represents a solution of the problem. Its pheromone trail is associated with the extent that jobs are scheduled into the same batch. Computational results show that JACO and BACO significantly outperform other four algorithms addressed in literatures, which are SA (simulated annealing), GA (genetic algorithm), FFLPT (first-fit longest processing time) and BFLPT (best-fit longest processing time). Furthermore, BACO is better than JACO. These results show that BACO is an effective and efficient method for solving scheduling problems to minimize makespan on a single batch processing machine with non-identical job sizes.

Key words scheduling; batch processing machines; ant colony optimization; combinatorial optimization