文章编号: 1001-0920 (2014) 10-1758-07 DOI: 10.13195/j.kzyjc.2013.1112

求解差异工件批调度问题的改进型蚁群算法

贾兆红[a,b], 李 丹[b], 李龙澍[a,b]

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

摘 要: 针对最小化制造跨度的差异工件尺寸单批处理机调度问题, 通过将其转化为最小化浪费空间的问题, 采用候选集策略构建分批以减少搜索空间, 利用基于浪费空间的启发式更新信息素, 提出一种改进的最大最小蚁群算法. 此外, 在算法中还引入了一种局部优化策略, 以进一步提高算法的性能. 仿真实验结果表明, 所提出的算法优于其他几种已有算法, 验证了所提出算法的有效性和鲁棒性.

关键词: 调度; 批处理机; 制造跨度; 最大最小蚁群算法; 局部优化

中图分类号: TP278 **文献标志码:** A

Improved ant colony algorithm for solving batch scheduling problem with non-identical job sizes

JIA Zhao-hong[a,b], *LI Dan*[b], *LI Long-shu*[a,b]

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

Abstract: By formulating the problem of minimizing the makespan on a single batch machine with non-identical job sizes as a problem of minimizing the wasted space minimization, a candidate set strategy is designed to narrow the search space when constructing the batches, combined with a heuristic based on the wasted space to update the pheromone information. Incorporated with a local optimization method to further improve the performance, an improved max-min ant system algorithm is presented. Moreover, the parameter setting in the proposed algorithm is analyzed by experimental designs to determine the appropriate parameter values. The simulation results show the effectiveness and robustness of the proposed algorithm that outperforms several previously studied algorithms.

Key words: scheduling; batch processing machine; makespan; max-min ant system(MMAS) algorithm; local optimization

0 引 言

并行批处理机一次可以同时加工多个工件, 这类机器已广泛应用于半导体加工、铸造、金属、航空、制药和物流运输等行业[1]. 例如, 在半导体工业中, 芯片加工的测试阶段需要在满足高温炉的容量约束下, 对带有不同尺寸和灼烧时间的芯片分组, 然后将同组的多个芯片同时放入炉中进行灼烧, 每组的加工时间为该组芯片的最长加工时间. 此类问题通常被称为差异工件尺寸单批处理机调度问题(SPBN). 与其他工序相比, 灼烧工序的加工时间冗长, 往往成为芯片测试阶段的瓶颈. 因而, 有效的调度方案对于提高半导体加工业的整体性能至关重要.

Uzsoy[2]最先证明了最小化制造跨度的SPBN问题强NP难, 并提出几种启发式算法. 其中: FFLPT先对工件按加工时间降序排序, 然后根据机器容量对工件分批, 再采用FF规则对批进行调度; 而FFDECR则是先对工件按工件尺寸降序排序. 实验结果表明, FFLPT的性能较优. Zhang等[3]证明FFLPT的最差性能比不超过2, 而FFDECR的最差性能比可能任意大. Dupont等[4]提出了另外几种改进的启发式算法, 并证明BFLPT性能最优. 针对带有任意到达时间的SPBN问题, Li等[5]提出了性能比为$2+\varepsilon$(ε为任意小

收稿日期: 2013-08-14; **修回日期:** 2014-01-06.

基金项目: 国家自然科学基金项目(71171184); 国家留学基金委项目(201206505002); 安徽大学自然科学基金项目(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; 李丹(1988−), 男, 硕士生, 从事智能调度算法的研究.

的正数) 的近似算法. Kashan 等[6]提出了工件尺寸与其加工时间成比例的假设, 设计了近似性能比为 4/3 的算法. 启发式算法通常基于一定的排序规则来求解问题, 排序结束后不能对所得解作进一步的改善, 往往解的质量不高, 因而一些学者致力于研究基于智能优化的元启发式算法以获得更好的解. 基于编码方法, 元启发式算法可以分为领域搜索和构建型 2 种, 前者以某些初始解为起点, 通过某种邻域搜索机制迭代地对其改进, 以获得最终解, 而后者以空解为起点, 迭代地向其增加解成分直到构建出可行解[7]. SPBN 的解是批序列且不同解中批的个数不同, 因而采用邻域搜索的方法, 难以直接对批序列进行编码, 而通常先对工件序列编码, 再采用某种启发式对工件分组[8-10]. 这样基于邻域的算法性能会因部分依赖于生成批序列的启发式而受到限制, 而且额外的启发式必定增加计算时间, 因此构建型算法的求解效率更高. 蚁群优化算法 (ACO) 是一种具有代表性的构建型算法, 表示一类在多个简单个体 (即人工蚂蚁) 间存在类似于自组织交互特性的分布式算法[11],目前也是智能优化算法领域的研究热点之一, 已被成功地应用于多种 NP 难的离散优化问题[12-14]. 针对 SPBN 问题, 王栓狮等[15] 提出采用不同编码方法的 2 种 ACO 算法. Cheng 等[16]结合 Metropolis 准则提出了一种改进的 ACO 算法来求解模糊制造系统下的 SPBN 问题. Xu 等[17]采用 ACO 算法对单位尺寸工件且带有到达时间的 SPBN 的制造跨度进行优化. 李小林等[18]采用最大最小蚁群 (MMAS) 算法求解了同类平行机批调度问题. Xu 等[19]基于 ACO 求解了双目标平行机批调度问题.

本文提出一种改进的 MMAS 算法来求解最小化制造跨度的 SPBN 问题. 与已有方法相比, 本文所提出的算法的不同之处主要体现在: 以往的元启发式通常基于未调度工件构建解, 而本文从可以减少当前批浪费空间的未调度工件中进行选择; 基于隐性加工时间设计局部优化策略以获得更好的解.

1　问题模型

在 SPBN 问题中, 假设由 n 个工件组成工件集 $J=\{1,2,\cdots,n\}$. 其中: 每个工件 j 带有加工时间 p_j 和尺寸 s_j; J 中的工件分组后在最大容量为 B 的批处理机上加工, 加工的批次记为 $B_k\,(k=1,2,\cdots,m)$, 任一批中所有工件的尺寸之和不大于机器容量 B, 即 $\sum_{j=1}^{|B_k|}\leqslant B$; 批的加工不允许中断, 批 B_k 的加工时间 P_k 等于批中工件的最大加工时间, 即 $P_k=\max\limits_{j\in B_k}p_j$; 问题目标最大完工时间是所有批的加工时间之和.

基于上述假设, 可建立问题 $1|s_j,B|C_{\max}$ 的数学模型如下:

$$\min C_{\max}=\sum_{k=1}^{m}P_k. \tag{1}$$

$$\text{s.t.}\ \sum_{k=1}^{m}x_{jk}=1; \tag{2}$$

$$\sum_{j=1}^{n}s_jx_{jk}\leqslant B,\ k=1,2,\cdots,m; \tag{3}$$

$$0\leqslant p_jx_{jk}\leqslant P_k,\ j=1,2,\cdots,n,\quad k=1,2,\cdots,m; \tag{4}$$

$$|B_k|=\sum_{j=1}^{n}x_{jk},\ k=1,2,\cdots,m; \tag{5}$$

$$\sum_{j=1}^{n}\left\lceil\frac{s_j}{B}\right\rceil\leqslant m\leqslant n; \tag{6}$$

$$x_{jk}=\begin{cases}1, j\in B_k,\\ 0, j\notin B_k,\end{cases}\quad j=1,2,\cdots,n,\ k=1,2,\cdots,m. \tag{7}$$

其中: 式 (1) 表示模型目标; 式 (2) 表示每个工件只能被分配到一个批中; 式 (3) 表示批中工件的总尺寸不超过机器容量; 式 (4) 表示批的加工时间约束; 式 (5) 为批中工件数的计算公式; 式 (6) 表示最小批数的约束; 式 (7) 定义决策变量, $x_{jk}=1$ 表示工件 j 被分到批 k, 否则, 表示 j 没有被分组到 k 中.

2　基于 MMAS 的求解算法

2.1　MMAS 算法

ACO 算法是有效求解离散优化问题的一种元启发式算法[20]. 继最初的蚂蚁系统 (AS) 被提出之后, 又出现多种改进算法[21]. MMAS 算法在 AS 的基础上通过引入最优路径开发和信息素限制等措施, 有效地避免了在搜索初期出现停滞, 从而提高了算法的搜索性能. MMAS 算法在许多应用中表现出较好的性能, 是目前应用最成功的蚁群算法之一, 其收敛性能在理论上已经得到证明[22].

2.2　基于 MMAS 的 SPBN 求解算法

为了不依赖于额外的启发式规则, 利用蚁群算法构建型的特点, 直接对批序列进行编码. 根据机器容量约束选择未调度工件构建批时, 新加入的工件会因增加了批的浪费空间 (WS) 而降低了解的质量. 本文基于 WS 设计候选工件集和启发式信息.

2.2.1 基于浪费空间的候选工件集

图1给出一个含有3个工件的可行批k. 由于工件具有加工时间和尺寸, 可以从二维空间的角度来考虑这个问题.

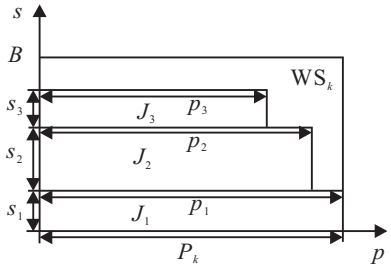

图 1 批的浪费空间

定义 1 假定k为某可行解的任一批, 批k的浪费空间WS_k定义为

$$\mathrm{WS}_k = BP_k - \sum_{j\in B_k} s_j p_j. \tag{8}$$

定义 2 假定O为一个可行解, O的浪费空间WS^O为所有批的浪费空间之和, 即

$$\mathrm{WS}^O = \sum_{k=1}^{m} \mathrm{WS}_k. \tag{9}$$

基于上述定义, 可以得到如下命题.

命题 1 对于问题$1|s_j, B|C_{\max}$, 最小化$C_{\max}$等价于最小化解的浪费空间WS^O.

证明 假设O为一个可行解, k为其任一批, 则

$$\mathrm{WS}^O = \sum_{k=1}^{m} \mathrm{WS}_k =$$
$$\sum_{k=1}^{m} \Big(BP_k - \sum_{j\in B_k} s_j p_j\Big) =$$
$$B\sum_{k=1}^{m} P_k - \sum_{k=1}^{m}\sum_{j\in B_k} s_j p_j =$$
$$BC_{\max} - \sum_{k=1}^{m}\sum_{j\in B_k} s_j p_j.$$

由于各工件的s_j和p_j均事先已知, 上式中第2项可预先确定且为固定值, 第1项中B也是已知的, 故$\mathrm{WS}^O \propto C_{\max}$, 即$\min C_{\max}$等价于$\min \mathrm{WS}^O$. □

由命题1可知, 最优解将是浪费空间最小的调度解. 虽然只有在解构建完时才能得到解的浪费空间, 但可通过尽量减少当前批的浪费空间, 使得解的浪费空间也尽量小. 因此, 蚂蚁应选择最大程度减小当前批WS的工件来生成批. 令U_k表示批k的未调度工件集, 则有如下命题.

命题 2 假定当前批为B_k, $y\in U_k$且$s_y \leqslant B - \sum_{j\in B_k} s_j$. 如果工件$y$满足如下条件:

$$s_y p_y > B(\overline{P}_k - P_k), \tag{10}$$

则加入y将减少B_k的浪费空间. 其中

$$\overline{P}_k = \begin{cases} P_k,\ p_y \leqslant P_k; \\ p_y,\ p_y > P_k. \end{cases}$$

证明 下面采用反证法证明. 令$\overline{B}_k$表示加入y得到的新批, 其浪费空间为$\overline{\mathrm{WS}}_k$. 定义这2个批浪费空间的差异为$\Delta\mathrm{WS}_k^y$, 即

$$\Delta\mathrm{WS}_k^y = \overline{\mathrm{WS}}_k - \mathrm{WS}_k =$$
$$\Big(B\overline{P}_k - \sum_{j\in B_k} s_j p_j - s_y p_y\Big) -$$
$$\Big(BP_k - \sum_{j\in B_k} s_j p_j\Big) = B(\overline{P}_k - P_k) - s_y p_y.$$

在B_k中加入y后不会减少WS_k, 故$\Delta\mathrm{WS}_k^y \geqslant 0 \Rightarrow s_y p_y \leqslant B(\overline{P}_k - P_k)$, 与假设矛盾. □

定义 3 当前批k的候选工件集CS_k定义为可减少B_k浪费空间的工件组成的集合, 即

$$\mathrm{CS}_k = \Big\{y | y\in U_k \bigwedge s_y \leqslant B - \sum_{j\in B_k} s_j \bigwedge$$
$$s_y p_y > B(\overline{P}_k - P_k)\Big\}. \tag{11}$$

2.2.2 信息素定义

在MMAS算法的优化过程中, 蚂蚁根据工件与当前批之间的信息素和启发式信息, 迭代地从候选集中逐个选择工件加入批来构建可行解. 第t代的信息素轨迹$\tau_k^y(t)$反映了候选集中工件y加入批$B_k(t)$的期望度, 定义如下:

$$\tau_k^y(t) = \frac{\sum\limits_{x\in B_k(t)} \varepsilon_{xy}(t)}{|B_k(t)|}. \tag{12}$$

其中: $\varepsilon_{xy}(t)$反映了x和y分在同一批中的期望度, 且$\varepsilon_{xy}(0) = ((1-\rho)\mathrm{LB})^{-1}$.

2.2.3 启发式信息

由命题1和命题2可知, 较大程度地减少浪费空间的工件加入批将有利于问题目标, 所以启发式信息η_k^y定义如下, 使得$\Delta\mathrm{WS}_k^y$越小的工件将更有可能被选择:

$$\eta_k^y = \begin{cases} 1 - B(\overline{P}_k - P_k) + s_y p_y,\ \Delta\mathrm{WS}_k^y < 0; \\ 1,\ \Delta\mathrm{WS}_k^y \geqslant 0. \end{cases} \tag{13}$$

2.2.4 解的构建

蚂蚁$d(d = 1, 2, \cdots, n_d)$按如下步骤构建可行解, 直到$U_k = \varnothing$.

Step 1: 构建一个空批, 并随机选择一个未调度工件作为该批的第一个工件.

Step 2: 根据如下概率p_k^y, 从CS_k中逐个选择下

一个工件加入当前批, 直至 CS_k 为空:

$$p_k^y=\begin{cases}\dfrac{\tau_k^y(\eta_k^y)^\beta}{\sum\limits_{i\in \mathrm{CS}_k}\tau_k^i(\eta_k^i)^\beta},\ y\in \mathrm{CS}_k;\\ 0,\ \text{otherwise.}\end{cases}\tag{14}$$

其中 β 为表示 τ_k^y 和 η_k^y 相对重要性的参数.

2.2.5　信息素更新

MMAS 算法的一个重要特性是只利用当代最优解或全局最优解进行信息素更新[21]. 如果只利用全局最优解更新信息素, 则可能导致搜索过于集中而陷于较差的解, 降低搜索其他更好解的可能性. 由于每代所得的当代最优解可能不同, 用其更新信息素会使得大部分的解成分只能偶尔得到加强, 可以改善算法的搜索性能. 令 $m_{xy}(t)$ 记录 t 代工件 x 和 y 分在同一批的频率, 信息素更新定义为

$$\varepsilon_{xy}(t+1)=(1-\rho)\varepsilon_{xy}(t)+m_{xy}(t)\Delta\varepsilon_{xy}(t).\tag{15}$$

其中: $\rho\in(0,1)$ 是控制信息素挥发速度的参数, 以避免信息素无限累积. 若第 t 代的所有解中, x 和 y 都没有被分在同一批, 则 $\Delta\varepsilon_{xy}(t)=0$, 否则 $\Delta\varepsilon_{xy}(t)=Q/C_{\max}^*(t)$. 其中: Q 是输入参数, 当 t 是 μ 的整数倍时, $C_{\max}^*(t)$ 为全局最优解的目标值 $C_{\max}^{gb}$, 否则, $C_{\max}^*(t)$ 为当代最优解的目标值.

虽然利用当代或全局最优解更新信息素可以改善搜索性能, 但是当某个路径的信息素特别高或特别低时, 搜索仍可能会停滞, 解决方法之一是动态调整 p_k^y. 而 η_k^y 在搜索过程中是不变的, 于是可调整 τ_k^y. 在每代更新信息素后, 采用如下公式进行调整:

$$\varepsilon_{xy}(t+1)=\begin{cases}\varepsilon_{\min},\ \varepsilon_{xy}(t+1)<\varepsilon_{\min};\\ \varepsilon_{xy}(t),\ \varepsilon_{\min}\leqslant\varepsilon_{xy}(t+1)\leqslant\varepsilon_{\max};\\ \varepsilon_{\max},\ \varepsilon_{xy}(t+1)>\varepsilon_{\max}.\end{cases}\tag{16}$$

其中

$$\varepsilon_{\max}=((1-\rho)C_{\max}^{gb})^{-1},$$

$$\varepsilon_{\min}=\frac{\varepsilon_{\max}(1-\sqrt[n]{0.05})}{(n/2-1)\sqrt[n]{0.05}}\ [23].$$

2.2.6　信息素重新初始化

在迭代过程中, 某些轨迹的概率可能会变得非常小, 从而降低了解的多样性, 容易导致搜索停滞, 此时可以通过重新初始化信息素来提高解的多样性[21]. 当目标函数值在 L 代内都没有改进时, 可将信息素重新初始化为 $\varepsilon_{\max}$.

2.2.7　局部优化策略

MMAS 算法在信息素更新过程中利用全局最优解 (即每代局部最优解中的最好解) 和局部最优解, 这种方法可以在更少的迭代中获得更好的全局最优解. 因而, 进一步优化局部最优解无疑可以获得更好的全局最优解. 于是本文引入一种新的基于隐性加工, (RPT) 的局部优化算法, 对每只蚂蚁构建的可行解进行改进.

定义 4　令 B_k 为可行解的任一批, 隐性加工时间 RPT_k 定义为已分配至 B_k, 但对 P_k 没有影响的工件的加工时间之和, 即

$$\mathrm{RPT}_k=\sum_{j\in B_k}p_j-\max_{j\in B_k}\{p_j\}.\tag{17}$$

定义 5　定义任一可行解 O 的 RPT 为 O 中所有批的 RPT 之和, 即

$$\mathrm{RPT}^O=\sum_{k=1}^{m}\mathrm{RPT}_k.\tag{18}$$

命题 3　增加一个调度解的 RPT 将减少该解的制造跨度.

证明　令 O 为一个可行解. 基于上述公式可得

$$\mathrm{RPT}^O=\sum_{k=1}^{m}\Big(\sum_{j\in B_k}p_j-\max_{j\in B_k}\{p_j\}\Big)=\sum_{k=1}^{m}\sum_{j\in B_k}p_j-\sum_{k=1}^{m}\max_{j\in B_k}\{p_j\}.$$

其中: 第 1 项为常量, 即所有工件的加工时间之和; 第 2 项为 $C_{\max}$. 故增加 RPT^O 将减少 O 的制造跨度. □

针对每个调度解, 采用图 2 所示的局部优化算法 (LRA) 进行改进.

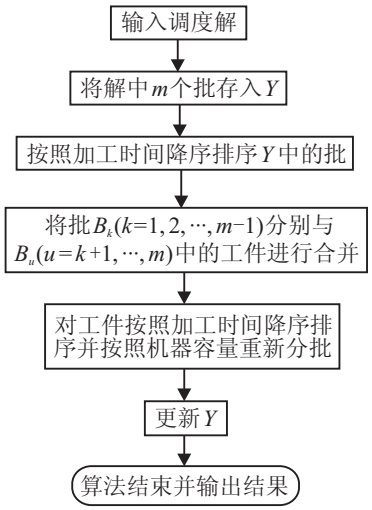

图 2　算法 LRA 的流程图

2.2.8　算法描述

基于 LRA 局部优化策略的 MMAS 算法 (LRAS) 流程如图 3 所示.

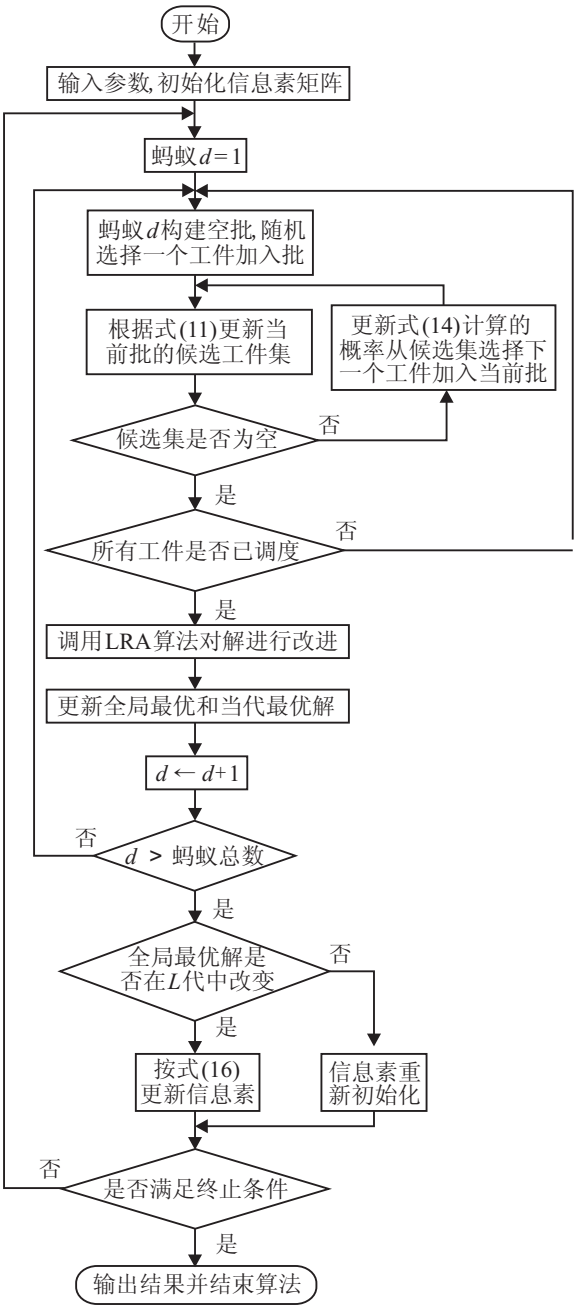

图 3 算法 LRAS 流程

3 实验与分析

3.1 实验数据

采用 Melouk[10]的方法生成随机测试实例. 为了在不同类型的问题上进行验证, 考虑工件数, 工件加工时间和工件尺寸等因素的变化, 设计均匀分布于[1, 15]和[15,35]的小尺寸 S_1 和大尺寸 S_2 两类工件集, 且每类的工件数分别为 10, 20, 50, 100 (记为 $J_1 \sim J_4$), p_j 均匀分布于[8,48], B 均设置为 40.

3.2 参数设置

ACO 算法的性能对影响搜索行为和收敛速度的参数敏感, 为了得到高质量的解, 有必要通过一些准备实验确定合适的参数值. 通过准备实验及其结果分析, 对影响 LRAS 算法性能的实验参数取值如表 1 所示.

表 1 参数设置

n_d	β	ρ	Q	$I_{\max}$	L
20	10	0.5	1	200	50

3.3 实验结果

将本文提出的算法 LRAS 和其他算法在测试实例上分别进行测试. 由于测试实例是随机产生的, 对每一类测试实例分别生成 5 个不同的实例, 以获得每种算法在每类测试实例上的平均结果. 所有算法均在 Inter Core 2 处理器, 2 G RAM 环境下, 采用 C++ 编程实现. 为了更有效地体现各算法对于下界的改进程度, 令 R_{A} 表示算法 A 相对于下界的距离, 对于测试实例 I, $R_{\mathrm{A}}^{\mathrm{I}}$ 定义为

$$R_{\mathrm{A}}^{\mathrm{I}} = \frac{C_{\max}^{\mathrm{I}}(\mathrm{A}) - \mathrm{LB}^{\mathrm{I}}}{\mathrm{LB}^{\mathrm{I}}} \times 100\%,$$

其中 LB 根据文献[2]中的下界计算方法获得. $R_{\mathrm{A}}^{\mathrm{I}}$ 值越小则解距离下界越近, 表明 A 的优化效果越好. 为了比较算法的平均性能, 计算算法 A 在每类实例上 $R_{\mathrm{A}}^{\mathrm{I}}$ 的平均值, 即

$$R_{\mathrm{A}} = \sum_{\mathrm{I}=1}^{5} \frac{R_{\mathrm{A}}^{\mathrm{I}}}{5}.$$

表 2 给出了不同算法的平均实验结果, 并且每类实例上每一项指标对应的最好性能值均加粗表示. 在表 2 中, H 代表启发式 FFLPT[2], FFDECR[2]和 BFLPT[4]中的最好结果; GA[9], BACO[15], MACO[16]和 ACO[17]为 4 种元启发式的比较算法. 对于每一实例, 元启发式均运行 10 次以得到平均结果.

表 2 不同算法的平均实验结果比较

问题类型	LB	H	GA			BACO			MACO			ACO			LRAS		
		R_{H}	R_G	SD_G	T_G	R_B	SD_B	T_B	R_M	SD_M	T_M	R_A	SD_A	T_A	R_L	SD_L	T_L
J_1S_1	111	6.14	8.92	0.36	**0.69**	3.25	0.57	1.39	3.94	2.19	6.85	2.98	0.93	1.38	**2.29**	**0.31**	1.37
J_2S_1	241	15.18	12.85	1.33	**0.72**	7.67	1.57	2.91	10.79	4.42	14.68	7.64	1.94	2.97	**6.52**	**0.76**	2.69
J_3S_1	574	8.70	10.97	5.21	**0.92**	8.49	5.18	9.20	10.56	8.07	39.95	8.27	5.45	9.06	**7.43**	**3.10**	7.73
J_4S_1	1 153	9.14	10.88	6.85	21.66	10.82	7.99	23.73	11.36	16.68	85.04	10.78	8.22	24.02	**7.13**	**4.44**	**21.31**
J_1S_2	175	15.26	25.42	**0.00**	0.69	15.26	0.00	0.65	15.36	0.06	8.65	15.28	**0.08**	0.72	**15.09**	**0.00**	0.60
J_2S_2	381	21.85	25.53	**0.00**	**0.73**	20.77	0.00	1.40	21.01	1.32	15.26	20.97	**1.21**	1.46	**20.39**	**0.00**	1.26
J_3S_2	927	24.47	25.21	9.87	**1.00**	21.86	2.06	3.12	22.20	5.68	39.86	22.07	2.45	3.81	**21.64**	**1.22**	3.07
J_4S_2	1 745	23.43	24.62	5.44	9.31	21.82	5.20	11.23	22.88	11.83	85.17	22.16	5.54	11.37	**21.15**	**4.51**	**8.01**

表 2 的第 1 列给出问题类型的编码, 如 J_1S_1 表示包含 10 个 (J_1) 小尺寸 (S_1) 工件的问题实例; 第 2 列的 LB 表示每类问题 5 个实例的下界平均值; 第 3 列的 R_H 代表 3 种启发式的最好解, 即

$$R_H = \min\{R_{\mathrm{FFLPT}}, R_{\mathrm{FFDECR}}, R_{\mathrm{BFLPT}}\};$$

第 4 ~ 6 列给出了 GA 的平均性能, 其中 R_G 表示算法 GA 所得解距离下界的平均值, SD_G 表示算法 GA 在每类问题中每个实例 I 上运行 10 次所得标准偏差 $\mathrm{SD}_{\mathrm{A}}^{\mathrm{I}}$ 的平均值, 即

$$\mathrm{SD}_{\mathrm{A}} = \frac{\sum_{\mathrm{I}=1}^{5} \mathrm{SD}_{\mathrm{A}}^{\mathrm{I}}}{5},$$

SD_G 的值越小表示算法的鲁棒性越好, T_G 表示 GA 在每类问题测试实例运行 10 次的平均耗时 $T_{\mathrm{A}}^{\mathrm{I}}$ 的平均值 (单位为秒), 即

$$T_{\mathrm{A}} = \frac{\sum_{\mathrm{I}=1}^{5} T_{\mathrm{A}}^{\mathrm{I}}}{5}.$$

类似地, 第 7 ~ 9 列, 第 10 ~ 12 列, 第 13 ~ 15 列, 第 16 ~ 18 列分别给出了算法 BACO, MACO, ACO 和 LRAS 的平均性能. 由于启发式算法是确定性算法, 运行时间也很短, 故表 2 没有列出启发式算法的运行时间.

由表 2 可知, 每种算法在小尺寸工件问题上所得的解在质量上总体均优于在大尺寸问题上所得的解, 对于相同工件数的问题, 基于蚁群优化的算法 BACO, MACO, ACO 和 LRAS 在小尺寸问题上的计算时间要多于对应的大尺寸问题. 另一方面, 4 种元启发式算法在大尺寸工件问题上表现出来的鲁棒性优于小尺寸工件的问题 (除了在工件数为 50 的小工件问题上, 算法 GA 的鲁棒性相对于大工件问题较好). 这可能是因为在大工件尺寸问题空间中, 解的分布相对于小尺寸问题更加离散化, 从而增加了搜索到更好解的难度.

对于小尺寸工件问题, 算法 LRAS 所得解与问题下界的距离最小, 因而解的质量最好, 算法 ACO 次之. 同时, LRAS 算法的平均标准偏差均小于其他元启发式算法, 因而其鲁棒性也是相对最好的, 而 MACO 的鲁棒性相对最差. 在小规模的小尺寸问题上, GA 算法的运行时间最少, LRAS 次之, MACO 耗时最长, 而对于工件数为 100 的小尺寸工件问题, LRAS 算法需要的计算时间最少, 运行效率最高.

对于大尺寸的 4 类问题, LRAS 算法所得解的质量均是最好的. 对于小规模的大尺寸问题 J_1S_2 和 J_2S_2, 启发式 H 与基于 ACO 的元启发算法得到质量相似的解, 且都优于 GA 所得的解, 而元启发算法 GA, BACO 和 LRAS 的平均标准偏差均为零, 它们的鲁棒性均优于 MACO 和 ACO. 在大规模的大尺寸问题 J_3S_2 和 J_4S_2 上, LRAS 表现出最好的鲁棒性, BACO 次之. 而在运行时间方面, 算法 GA 与 LRAS 相当, 而 MACO 在 4 类大尺寸问题实例上依然是相对耗时最长的算法.

综上所述, LRAS 算法的综合性能优于其他算法, 不仅所得解的质量较其他算法好, 而且鲁棒性也较好. 这是由于在 LRAS 中采用了有效的搜索机制, 利用问题特征的候选工件集, 显著减小了搜索空间, 使得用较少的运行时间可获得质量较好的解. 此外, 有效的局部优化策略进一步提高了解的质量.

4 结　论

本文针对最小化制造跨度的差异工件尺寸单机批调度问题, 提出了一种新的基于 ACO 的元启发式算法 LRAS, 通过利用最小化浪费空间与最小化制造跨度之间的等价关系, 引入候选集策略以减小搜索空间, 提高搜索性能. 此外, 在算法中引入了一种基于隐性时间的局部优化机制来进一步提高解的质量. 实验结果表明, 所提出的算法能够在合理的运行时间内找到质量较好的解, 且算法的鲁棒性也较其他几种比较算法更优. 进一步的研究可以将所提出的算法推广到相同平行机问题, 以及带有不相容工件族、准备时间或其他约束的更为复杂的问题中. 此外, 还可以考虑其他问题目标, 如最大延迟、平均加权流水时间等.

参考文献(References)

[1] Mathirajan M, Sivakumar A I. A literature review, classification and simple meta-analysis on scheduling of batch processors in semiconductor[J]. Int J of Advanced Manufacturing Technology, 2006, 29(9/10): 990-1001.

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

[3] Zhang G, Cai X, Lee C Y, et al. Minimizing makespan on a single batch processing machine with nonidentical job sizes[J]. Naval Research Logistics, 2001, 48(3): 226-240.

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

[5] Li S G, Li G J, Wang X L, et al. Minimizing makespan on a single batching machine with release times and non-identical job sizes[J]. Operations Research Letters, 2005, 33(2): 157-164.

[6] Kashan A H, Karimi B, Fatemi Ghomi S M T. A note on minimizing makespan on a single batch processing

machine with nonidentical job sizes[J]. Theoretical Computer Science, 2009, 410(27/28/29): 2754-2758.

[7] Dorigo M, Stutzle T. Ant colony optimization[M]. Cambridge: MIT Press, 2004: 36-39.

[8] Kashan A H, Karimi B, Jolai F. Effective hybrid genetic algorithm for minimizing makespan on a single-batch-processing machine with non- identical job sizes[J]. Int J of Production Research, 2006, 44(12): 2337-2360.

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

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

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 王栓狮, 陈华平, 程八一, 等. 一种差异工件单机批调度问题的蚁群优化算法[J]. 管理科学学报, 2009, 12(6): 72-82.
(Wang S S, Chen H P, Cheng B Y, et al. Minimizing makespan on a single batch processing machine with non-identical job sizes using ant colony optimization[J]. J of Management Science, 2009, 12(6): 72-82.)

[16] Cheng B Y, 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.

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

[18] 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): 102-110.
(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): 102-110.)

[19] Xu R, Chen H, Li X. A bi-objective scheduling problem on batch machines via a pareto-based ant colony system[J]. Int J of Production Economics, 2013, 145(1): 371-386.

[20] Lee Z J, Su S F, Chuang C C, et al. Genetic algorithm with ant colony optimization(GA-ACO) for multiple sequence alignment[J]. Applied Soft Computing, 2008, 8(1): 55-78.

[21] Stutzle T, Hoos H. Max-min ant system[J]. Future Generation Computer Systems, 2000, 16(7): 889- 914.

[22] Stutzle T, Dorigo M. A short convergence proof for a class of ACO algorithms[J]. IEEE Trans on Evolutionary Computation, 2002, 6(4): 358-365.

[23] Kashan A H, Karimi B, Jolai F. Minimizing makespan on a single batch processing machine with non-identical job sizes: A hybrid genetic approach[C]. Proc of the 6th Conf on Evolutionary Computation in Combinatorial Optimization. Hungary: Springer, 2006: 135-146.

(责任编辑: 闫 妍)