文章编号: 1000−8152(2011)10−1363−08

蛙跳优化算法求解多目标无等待流水线调度

潘玉霞[1], 潘全科[2], 李俊青[2]

(1. 海南大学 三亚学院, 海南 三亚 572000; 2. 聊城大学 计算机学院, 山东 聊城 252059)

摘要: 提出了基于Pareto边界和档案集的改进蛙跳算法, 解决以最大完工时间、最大拖后时间和总流经时间为目标值的无等待流水线调度问题. 首先, 采用NEH(Nawaz-Enscore-Ham)启发式与随机解相结合的初始化方法, 保证了初始群体的质量和分布性; 其次, 采用两点交叉方法生成新解, 使蛙跳算法能够直接用于解决调度问题; 再次, 利用非支配解集动态更新群体, 改善了群体的质量和多样性; 最后, 将基于插入邻域的快速局部搜索算法嵌入到蛙跳算法中, 增强了算法的开发能力和效率. 仿真试验表明了所得蛙跳算法的有效性和高效性.

关键词: Pareto边界; 蛙跳算法; 无等待流水线调度; 多目标; 快速局部搜索

中图分类号: TP18　　**文献标识码:** A

Shuffled frog-leaping algorithm for multi-objective no-wait flowshop scheduling

PAN Yu-xia[1], PAN Quan-ke[2], LI Jun-qing[2]

(1. Sanya College, Hainan University, Sanya Hainan 572000, China;
2. School of Computer Science, Liaocheng University, Liaocheng Shandong 252059, China)

Abstract: An enhanced shuffled frog-leaping algorithm(ESFLA) is presented based on the Pareto front and archive set for solving the no-wait flowshop scheduling problems with makespan, maximum tardiness and total flow time criteria. Firstly, an initialization method based on the NEH(Nawaz-Enscore-Ham) heuristic is designed. Secondly, a two-point crossover operator is used to produce a new individual. Thirdly, a part of the non-dominated solutions are added to the population to improve their diversity and quality. Finally, to further enhance the exploitation capability and efficiency of the algorithm, a fast local search algorithm based on the insert neighborhood is embedded in the proposed shuffled frog-leaping algorithm. The computational results and comparisons show that the proposed ESFLA is effective and efficient in finding better solutions for the problem considered.

Key words: Pareto front; shuffled frog-leaping algorithm; no-wait flowshop scheduling; multi-objective; fast local search

1　引言(Introduction)

无等待流水线调度问题是一类重要的生产调度问题, 在许多工业领域中有广泛的应用背景, 受到了研究者的重视, 提出了许多针对此类问题的有效求解方法, 其中以求解单目标问题为主. 而在实际生产环境中, 往往存在着多个优化目标, 研究多目标流水线调度问题更加符合实际. 因此多目标优化理论与方法一直是理论界和工程界共同关注的重要研究课题. 通常多目标问题的最优解不是一个, 而是一组Pareto最优解集. 无等待调度要求加工的任务从开始到结束必须连续进行. 此类调度广泛存在于食品加工、炼钢、制药等生产领域[1,2]. 随着准时制的到来, 在生产过程中不仅要使生产周期和总流经时间最小, 还要使最大拖后时间指标得到满足.

蛙跳算法(shuffled frog-leaping algorithm, SFLA)是结合memetic算法和微粒群优化算法的优点而产生的一种新兴算法, 具有概念简单、参数少、全局寻优能力强等特点[3]. 近年来, 蛙跳算法被成功应用于求解优化问题: 文献[4]利用蛙跳算法能够跳出局部极值的特点解决成品油管输送网设计问题. 文献[5]采用改进的蛙跳算法对贴片机贴装顺序进行优化. 文献[6]将混合蛙跳算法应用到含风电场电力系统动态优化中, 统一考虑所有时段的动态优化潮流计算. 鉴于蛙跳算法的优点, 提出了一种基于Pareto边界和档案集的高效调度算法, 用于解决此类多目标调度问题. 仿真实验研究表明所得算法有良好的寻

收稿日期: 2010−07−28; 收修改稿日期: 2010−10−28.
基金项目: 国家自然科学基金资助项目(60874075, 70871065); 数字制造装备与技术国家重点实验室开放课题(华中科技大学)资助项目; 博士后科学基金资助项目(20070410791).

优特性和运算效率.

2 无等待流水线调度模型(No-wait flowshop scheduling)

2.1 问题描述(Problem description)

无等待流水线调度问题(no-wait flowshop schedule problem, NWFSP)可描述为: 给定m台机床$K = \{1, 2, \cdots, m\}$和n个工件$J = \{1, 2, \cdots, n\}$, 每个工件在各机床上的加工顺序相同. 同时约定, 每个工件在某一时刻只能够在一台机床上加工, 每台机床在某一时刻只能够加工一个工件, 同一工件相邻的两工序之间没有等待时间, 即要求任意工件在每台机床上的完成时间必须等于其在下一台机床上的开始加工时间. 调度模型如图1所示.

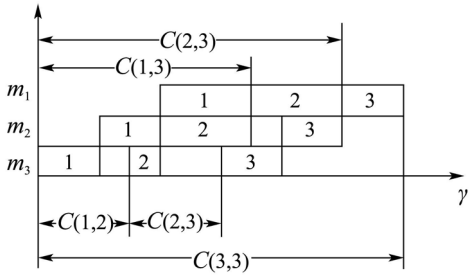

图 1 无等待调度模型

Fig. 1 No-wait scheduling model

假设工件按机床1$\sim$ m的顺序加工, 给定一个工序$\pi = \{\pi_1, \pi_2, \cdots, \pi_n\}$, 工件$j$在机床$k$上的操作时间为$p(\pi_j, k)$, $e(\pi_{j-1}, \pi_j)$代表工件π_{j-1}和π_j在第一台机床上的开始时间差, $d(\pi_j)$为工件π_j的交货期. 各工序的加工时间已知, 问题的求解目标是得到一个满足上述约束条件的可行调度, 使最大完工时间($C_{\max}(\pi)$)、最大拖后时间($T_{\max}(\pi)$)和总流经时间($TF(\pi)$)最小. 工件π_j在机床m上的完成时间$C(\pi_j, m)$的计算公式为

$$C(\pi_1 m) = \sum_{k=1}^{m} p(\pi_1, k), \tag{1}$$

$$C(\pi_j, m) = \sum_{i=2}^{j} e(\pi_{i-1}, \pi_i) + \sum_{k=1}^{m} p(\pi_l, k), \quad j = 2, 3, \cdots, n. \tag{2}$$

其中$e(\pi_{j-1}, \pi_j)$的计算公式如下:

$$e(\pi_{j-1}, \pi_j) = p(\pi_{j-1}, l) + \max[0, \max_{2 \leqslant k \leqslant m} \{\sum_{k=2}^{k} p(\pi_{j-1}, h) - \sum_{k=1}^{k-1} p(\pi_j, h)\}], \tag{3}$$

那么工序的最大完成时间计算公式如下:

$$f_1(\pi) = C_{\max}(\pi) = C(\pi_n, m) = \sum_{j=2}^{n} e(\pi_{j-1}, \pi_j) + \sum_{k=1}^{m} p(\pi_n, k). \tag{4}$$

最大拖后时间的计算公式如下:

$$f_2(\pi) = T_{\max} = \max_{j=1}^{n}\{\max\{0, C(\pi_j, m) - d(\pi_j)\}\}. \tag{5}$$

总流经时间的计算公式如下:

$$f_3(\pi) = TF(\pi) = \sum_{j=1}^{n} C(\pi_j, m) = \sum_{j=2}^{n}(n + 1 - j)e(\pi_{j-1}, \pi_j) + \sum_{j=1}^{n}\sum_{k=1}^{m} p(\pi_j, k). \tag{6}$$

2.2 目标值的快速计算(Fast calculation of objective)

计算目标函数值最大完成时间($C_{\max}(\pi)$)、最大拖后时间($T_{\max}(\pi)$)和总流经时间($TF(\pi)$)时, 式(4)(5)和式(6)的时间复杂度分别为$\mathrm{O}(n)$, $\mathrm{O}(n^2)$和$\mathrm{O}(mn)$, 求得目标值的时间复杂度为$\max \mathrm{O}(n^2)$, $\mathrm{O}(mn)$. 采用以下算法可以使时间复杂度降为$\mathrm{O}(n)$.

步骤 1 γ_i代表第1个和第i个工件在第1台机器上的开工时间差, δ_i代表前i个工件分别与第1个工件在第1台机器上开工时间差之和. $\gamma_1 = 0$, $\delta_1 = 0$, $\gamma_i = \gamma_{i-1} + e(\pi_{i-1}, \pi_i)$, $\delta_i = \delta_{i-1} + (n + 1 - i)e(\pi_{i-1}, \pi_i)$, $i = 2, \cdots, n$.

步骤 2 $TP(\pi_i)$用来存储工件π_i在所有机器上的总加工时间和. 令$TP(\pi_i) = \sum_{j=1}^{m} p(\pi_i, j)$, 则工件$\pi_i$的最大完工时间$C(\pi_i, m) = \gamma_i + TP(\pi_i)$, $i = 1, \cdots, n$.

步骤 3 计算$\lambda(\pi_i) = C(\pi_i, m) - d(\pi_i)$, $i = 1, \cdots, n$.

步骤 4 $C_{\max}(\pi) = C(\pi_n, m)$, $T_{\max}(\pi) = \max_{j=1}^{n}\{\max\{0, \lambda(\pi_j)\}\}$, $TF(\pi) = \delta_n + \sum_{i=1}^{n} TP(\pi_i)$.

2.3 快速插入邻域搜索算法(Fast insertion neighborhood search algorithm)

对于基于工件排列的优化问题, 插入操作的直径为$n - 1$. 相对于互换和逆序操作, 插入操作是直径最短的, 即插入操作产生的邻域解相互之间比较接近, 因此插入操作更适合执行深度搜索[8]. 故邻域搜索选择插入操作. 对于一个含有n个工件的调度π, 其插入邻域解的规模是$(n - 1)^2$. 用2.2部分的方法评价所得算法的时间复杂度为$\mathrm{O}(n^3)$. 为了提高效率, 提出了时间复杂度为$\mathrm{O}(n^2)$的快速插入邻域搜索算法:

步骤 1 设当前调度$\pi = (\pi_1, \pi_2, \cdots, \pi_n)$, 令$k = 1$.

步骤 2 从π中取出π_k工件所得调度记为$\pi'' = (\pi''_1, \pi''_2, \cdots, \pi''_{n-1})$.

执行以下操作:

步 骤 2.1　α_i代表π''的第1个和第i个工件在第1台机器上的开工时间差, β_i代表π''的前i个工件分别与第1个工件在第1台机器上开工时间差之和. 设$\alpha_1=0$, $\beta_1=0$计算$\alpha_i=\alpha_{i-1}+e(\pi''_{i-1},\pi''_i)$, $\beta_i=\beta_{i-1}+e(\pi''_{i-1},\pi''_i)$, $i=2,\cdots,n$.

步骤 2.2　计算π''的最大完工时间$C(\pi''_i,m)=\alpha_i+TP(\pi_i)$, $i=1,2,\cdots,n$.

步骤 2.3　计算$C_{\max}(\pi'')=C(\pi''_{n-1},m)$, $TP(\pi'')=\beta_n+\sum_{i=1}^{n-1}TP(\pi''_i)$.

步骤 2.4　计算$l(\pi''_i)=C(\pi''_i,m)-d(\pi''_i)$, $i=1,\cdots,n-1$.

步骤 2.5　令$\lambda_{n-1}=l(\pi''_{n-1})$, 计算

$$\lambda_{i-1}=\max\{\lambda_i,l(\pi''_{i-1})\},\ i=n-1,n-2,\cdots,2.$$

步骤 3　将工件π_k插入到π''的第k'个位置得到调度π, $k'\in\{1,2,\cdots,n\}\wedge k'\notin\{k,k-1\}$.

步骤 3.1　所得调度的makespan($C_{\max}(\pi')$)计算公式如下:

$$C_{\max}(\pi')=\begin{cases}C_{\max}(\pi'')+e(\pi_k,\pi''_1), & k'=0;\\ C_{\max}(\pi'')+e(\pi''_{k'-1},\pi_k)+\\ e(\pi_k,\pi''_{k'})-e(\pi''_{k'-1},\pi''_{k'}), & 0<k'<n;\\ C_{\max}(\pi'')+e(\pi''_{n-1},\pi_k)+\\ TP(\pi_k)-TP(\pi''_{n-1}), & k'=n.\end{cases}$$

步骤 3.2　最大延迟($T_{\max}(\pi')$)计算公式如下:

$$T_{\max}(\pi')=\begin{cases}\max\{0,\lambda_1+e(\pi_k,\pi''_1),TP(\pi_k)-d(\pi_k)\},\\ \qquad\qquad k'=0;\\ \max\{0,\lambda_1,\alpha_{k'-1}+e(\pi''_{k'-1},\pi_k)+TP(\pi_k)-\\ \qquad d(\pi_k),\lambda_{k'}+C_{\max}(\pi')-C_{\max}(\pi'')\},\\ \qquad\qquad 0<k'<n;\\ \max\{0,\lambda_1,C_{\max}(\pi')-d(\pi_k)\},\ k'=n.\end{cases}$$

步骤 3.3　最大流经时间($TF_{\max}(\pi')$)的计算公式如下:

$$(TF_{\max}(\pi'))=\begin{cases}TF_{\max}(\pi'')+(n-1)e(\pi_k,\pi''_1)+TP(\pi_k),\ k'=0;\\ TF_{\max}(\pi'')+\alpha_{k'-1}+(n+1-k')-\\ e(\pi''_{k'-1},\pi'_{k'-1})-(n-k')e(\pi_k,\pi''_k)-\\ (n-k')e(\pi''_{k'-1},\pi''_k)+TP(\pi_k), & 0<k'<n;\\ TF_{\max}(\pi'')+\alpha_{n-1}+e(\pi''_{n-1},\pi_k)+TP(\pi_k),\\ \qquad\qquad k'=n.\end{cases}$$

步骤 4　如果符合条件, 终止算法，否则, 执行步骤2.

3　基于Pareto档案的蛙跳调度算法(Shuffled frog-leaping algorithm based on the Pareto front and archive set)

根据蛙跳算法的优化机理, 结合NWFSP的特征, 采用基于工件序列的编码方式扩展了传统蛙跳算法的求解模型, 提出了一种动态群体的离散蛙跳算法, 使其能够解决离散化的组合优化问题.

3.1　算法编码方式(Algorithm encoding)

用蛙跳算法求解NWFSP问题的编码方法就是位置矢量的每一维用来表示一个工件, 这样一只青蛙就表示一个序列. 表1为位置矢量和对应的调度之间的关系.

表 1　位置矢量及对应的工件排序

Table 1　The position vector and corresponding sequence

维 数	1	2	3	4	5	6
位置矢量X	2	1	6	3	4	5
工件序列	2	1	6	3	4	5

3.2　算法初始化(Algorithm initialization)

好的初始化群体有助于提高SFLA的求解质量. 在此采用NEH方法生成前3个解, 其余解在离散空间中随机产生. 保证了初始种群一定的质量, 并不失初始种群的多样性, 同时启发式方法的快速性保证了这种初始化的速度. NEH初始化步骤如下:

步骤 1　计算各工件在所有机器上的加工时间总和, 并按递减的顺序排列.

步骤 2　先将前两个工件进行最优化调度, 然后将剩余工件依次插入到已调度好的工序中的所有可能位置, 分别使得子调度的最大完工时间最小, 直到所有工件调度完毕, 得到第一个个体.

步骤 3　先将前两个工件进行最优调度, 然后将剩余工件逐一插入到已调度好的工件排列中的所有可能位置, 分别使得总流经时间最小, 得到第2个个体.

步骤 4　按交货期的递增的顺序排列得到工件的一个排序, 再按步骤2的方法使得子调度最大拖后时间最小. 直到所有工件均调度完毕, 得到第3个个体. 此种初始化方法使得所得初始解具有较高的质量,有效提高了算法性能.

3.3　多目标个体排序及分组(Sort and group individuals)

在SFLA中, 种群由很多个体组成, 每只个体代表一个解. 根据个体的目标值将整个群体分为不同的子群, 并记录下种群个体中最好的解以及每个子群

中性能最好的个体和性能最差的个体. 对多目标群体中所有个体进行Pareto最优个体排序的步骤如下:

步骤 1 计算并记录下每个个体支配的解的个数a以及被支配的解的个数b.

步骤 2 求出$c = a - b$.

步骤 3 按c升值排序得到一个排序.

个体排序完毕后进行分组, 分组方式如下:

$$\begin{cases} U(j)k = U(k + p(k-1)), \\ j = 1, 2, \cdots, s,\ k = 1, 2, \cdots, p, \end{cases} \tag{7}$$

其中: s代表子群中的个体数, p代表子群数. 比如$p=3$, 那么第1个个体属于第1个子群, 第2个个体属于第2个子群, 第3个个体属于第3个子群, 第4个个体属于第1个子群, 依次类推.

3.4 新个体产生(Produce new individuals)

蛙跳算法的更新本质是$p_{\rm worst}$向$p_{\rm best}$和$g_{\rm best}$的学习过程. 为了使蛙跳算法直接应用于调度问题, 该过程可以通过两点交叉操作来实现, 即随机产生两个位置$pt1, pt2$(如$pt1 = 3,\ pt2 = 6$), 取$p_{\rm best}$中位置3和6之间的工件作为交叉区域, 将$p_{\rm best}$的交叉区域放到$p_{\rm worst}$的前面或后面, 分别生成两个个体(记为x_1和x_2), 并删除$p_{\rm worst}$中已在$p_{\rm best}$交叉区域中出现过的数字, 选择性能较好的个体作为新个体. 该方法优点是子串能够有效的继承父串的模式. 如图2所示.

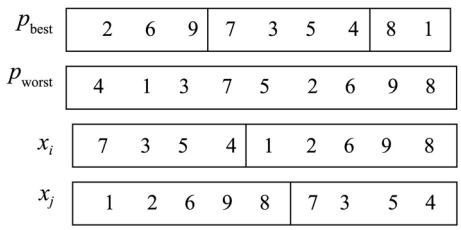

图 2 个体更新图

Fig. 2 Update individuals

3.5 个体矢量更新(Update individual)

为了保持群体的多样性, 若两点交叉操作产生的新解x支配$p_{\rm worst}$, 则用新解代替$p_{\rm worst}$; 若新解x与$p_{\rm worst}$互不支配时, 则以一定的概率代替$p_{\rm worst}$; 若新解x被$p_{\rm worst}$支配则不变. 更新公式如下:

$$p_{\rm worst} = \begin{cases} x, & x\text{支配}p_{\rm worst}, \\ x, & x\text{与}p_{\rm worst}\text{互不支配, } {\rm rand}(\cdot) < r, \\ p_{\rm worst}, & \text{其他}. \end{cases} \tag{8}$$

3.6 档案集更新(Update archive set)

为了防止算法执行过程中非支配解的丢失, 用档案集(archive set, AS)保存算法搜索到的非支配解. 在优化过程中, AS不断更新, 逐渐接近Pareto最优边界. 一旦新解产生了, 将其与AS中的非支配解作比较, 以确定一个新解是否为非支配解. 若新解为非支配解, 则更新AS, 并将AS中被新解支配的解删除.

4 多目标局部搜索(Multi-objective local search)

蛙跳算法具有较强的全局探索能力, 但局部开发能力较差. 为了避免算法陷入局部非支配解集, 对新得到的解执行基于快速插入邻域的局部搜索. 算法步骤如下:

步骤 1 令$\pi = x$.

步骤 2 对所有工件重复执行下列步骤:

步骤2.1 从x中无重复地随机选择一个工件, 把它分别插入到x中所有可能的其他位置, 插入过程中把所有的非支配解放入临时档案集(AS′)中.

步骤2.2 如果AS′中的解被x支配, 则将其删除, 如果此解支配x, 则将这个解赋值给x.

步骤2.3 将AS′中的剩余解与AS中的解合并, 通过排序找出非支配解, 更新AS.

步骤 3 如果x支配π, 返回步骤1.

步骤 4 输出个体π.

5 改进的多目标离散蛙跳算法(Enhanced multi-objective shuffled frog-leaping algorithm)

基于以上设计, 用于解决以最大完工时间、最大拖后时间和总流经时间为指标的无等待流水线调度问题的蛙跳算法步骤如下:

步骤 1 设置参数. 子群体的数量p和每个子群中个体的个数s. 那么整个种群中个体的个数为$F = p * s$. 初始化种群$U_i = (U_i^1, U_i^2, \cdots, U_i^d),\ i = 1, 2, \cdots, F$, 其中$d$为维变量.

步骤1.1 用2.2节的方法评价每个个体的目标值.

步骤1.2 将个体排序并分组.

步骤 2 初始化AS, 在AS中随机选择一个个体记为$g_{\rm best}$.

步骤 3 设置子迭代次数为1, 对各个子群分别执行如下:

步骤3.1 在每个子群中, 记录性能最好和最坏的个体, 按3.4节的新个体产生方法调整最坏个体的位置.

步骤3.2 如果上述过程不能产生一个更好的解, 则用$g_{\rm best}$代替$p_{\rm best}$重复执行上述过程.

步骤3.3 如果上述方法仍不能产生一个更好的解, 那么就随机产生一个新解x.

步骤 4 对产生的新解执行多目标局部搜索.

步骤 5 若满足子迭代条件执行步骤6, 否则执行步骤3.1.

步骤 6 在AS中随机选择p个非支配解加入群体中, $s = s + 1$.

步骤 7　将子群合并, 按指标排序并分组, 更新AS, 并在AS中随机选择一个个体作为g_{best}.

步骤 8　检查终止条件. 若满足终止条件则停止, 否则重新执行步骤3.

6　仿真试验与分析(Simulation and analysis)

6.1　试验设置(Test set)

为了验证所得算法ESFLA的有效性, 将其与文献[7]的DPSO算法和文献[8]中的HDE算法进行比较. 两种算法都在Pentium(R)1.60 GHz的处理器、512 M内存的微机上进行测试. 利用文献[7]的调度实例以及交货期的产生方法, 采用C++进行编码. 对于每个算例, 每种算法独立运行20次, 子群个数$p=5$, 初始群体个数为20, 每迭代依次增加5个, 算法最大运行时间为$T=10{*}m{*}n\,\mu$s. 采用3个性能指标来评价算法所得到的非支配解集: 距Pareto边界的平均距离Dl_{R}、非支配解个数$N_{\text{NDS}}(S_i)$、非支配解的比率$R_{\text{NDS}}(S_i)$[7]. 其中Dl_{R}能够区分不同算法所得到的s_i相对于S^*的优劣程度; 而$N_{\text{NDS}}(S_i)$和$R_{\text{NDS}}(S_i)$则从不同方面来评价s_i相对于S的优劣程度. 因此, 这3个性能指标能够有效地对不同算法所得解集进行评价. 设S^*表示参考非支配解集, 产生方式如下: ESFLA, HDE和DPSO算法对每个调度实例分别执行20次, 从中选出非支配解集作为参考非支配解集S^*. s_i表示第i个算法获得的非支配解集.

表 2　ESFLA, HDE和DPSO算法的Dl_{R}的比较

Table 2　Comparison of Dl_{R} among ESFLA, HDE and DPSO algorithm

算　例		ESFLA			HDE			DPSO		
名称	$n*m$	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN
Car01	$11*5$	0.13	0.00	2.56	6.15	2.56	7.69	13.88	2.56	99.56
Car02	$13*4$	2.19	0.00	4.69	4.53	3.13	4.69	3.05	1.56	4.69
Car03	$12*5$	0.96	0.00	1.89	72.09	1.33	2.28	1.23	0.00	1.89
Car04	$14*4$	2.39	0.00	3.77	3.69	2.21	3.77	1.25	0.00	1.56
Car05	$10*4$	0.77	0.00	5.13	6.15	2.56	7.69	6.92	2.56	7.69
Car06	$8*9$	0.00	0.00	0.00	6.92	0.00	13.41	0.56	0.00	5.56
Car07	$7*7$	0.00	0.00	0.00	4.50	0.00	10.00	30.06	0.00	99.1
Car08	$8*8$	0.00	0.00	0.00	3.92	0.00	8.11	9.46	0.00	97.30
Hel1	$100*10$	0.37	0.00	2.44	9.10	3.45	99.35	4.30	3.45	5.89
Hel2	$20*10$	0.53	0.00	1.05	2.49	1.49	2.54	2.10	1.05	2.54
Rec01	$20*5$	0.12	0.00	0.78	1.68	1.10	1.87	1.52	1.10	1.87
Rec03	$20*5$	0.84	0.00	1.01	0.99	0.67	1.01	0.67	0.00	1.01
Rec05	$20*5$	0.24	0.00	0.80	1.89	1.13	1.93	1.51	0.80	1.93
Rec07	$20*10$	1.93	0.00	2.83	2.64	1.89	2.83	76.93	0.94	222.36
Rec09	$20*10$	0.25	0.00	0.72	1.66	1.02	1.74	5.25	0.00	98.56
Rec11	$20*10$	1.47	0.70	2.10	2.10	2.10	2.10	1.92	1.40	2.10
Rec13	$20*15$	2.54	0.00	5.80	5.58	4.35	5.80	9.92	2.90	99.75
Rec15	$20*15$	4.35	0.00	6.52	6.52	6.52	6.52	39.94	4.35	140.31
Rec17	$20*15$	0.16	0.00	0.35	0.84	0.84	0.84	0.75	0.35	0.84
Rec19	$30*10$	0.10	0.00	0.96	2.03	1.36	2.32	6.06	0.00	99.04
Rec21	$30*10$	2.44	0.00	3.02	3.02	3.02	3.02	23.72	0.00	140.05
Rec23	$30*10$	0.00	0.00	0.00	2.35	1.54	2.62	1.65	1.54	2.62
Rec25	$30*15$	0.07	0.00	0.68	1.57	0.96	1.64	0.61	0.00	0.68
Rec27	$30*15$	0.31	0.00	1.03	2.49	2.49	2.49	1.60	1.03	2.06
Rec29	$30*15$	0.04	0.00	0.85	1.75	1.20	2.05	1.62	1.20	2.05
Rec31	$50*10$	0.38	0.00	1.08	2.38	1.52	2.60	2.00	1.52	2.60
Rec33	$50*10$	0.66	0.00	1.47	3.26	2.08	3.55	2.28	1.47	2.94
Rec35	$50*10$	0.00	0.00	0.00	2.85	2.28	3.89	2.76	2.28	3.89
Rec37	$75*20$	0.23	0.00	1.14	2.23	1.61	2.74	2.06	1.61	2.74
Rec39	$75*20$	0.04	0.00	0.76	1.30	1.07	1.83	1.19	1.07	1.83
Rec41	$75*20$	0.59	0.00	1.96	4.64	2.77	4.73	3.75	2.77	4.73
平均值		0.78	0.02	1.79	3.33	1.88	7.02	8.40	1.21	37.41

6.2 仿真结果比较(Comparison of simulation results)

对文献[7]提出的DPSO算法和文献[8]提出的HDE算法进行修改, 将其应用于多目标无等待流水线调度问题, 参数分别按文献[7]和文献[8]进行设置. 两种算法采用相同的终止条件. AVG, MIN和MAX分别为对应目标值的平均值、最小值和最大值. 结果如表2所示.

1) 由表2知ESFLA算法的平均AVG, MIN和MAX分别为0.78, 0.02和1.79, 远远小于HDE算法得到的3.33, 1.88和7.02和DPSO算法得到的8.40, 1.21, 37.41. 对于所有算例, ESFLA所得到的平均值都远远小于HDE的平均值. 对于大多数算例, ESFLA所得到的平均值都远远小于DPSO的平均值. 因此ESFLA所得到的非支配解更接近Pareto最优解.

2) 由表3知在31个算例中ESFLA所得$R_{\rm NDS}$的AVG, MIN, MAX有29个大于HDE所得到的值, 全部大于DPSO所得到的值. ESFLA的平均AVG值接近HDE的所得值的4倍, 是DPSO所得值的24倍, 说明ESFLA能得到更多的非支配解.

表 3 ESFLA, HDE和DPSO算法的$N_{\rm NDS}$的比较

Table 3 Comparison of $N_{\rm NDS}$ among ESFLA, HDE and DPSO algorithm

算 例		ESFLA			HDE			DPSO		
名称	$n*m$	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN
Car01	$11*5$	37.70	35.00	39.00	15.60	12.00	19.00	0.70	0.00	6.00
Car02	$13*4$	51.10	45.00	57.00	14.15	8.00	20.00	3.65	1.00	9.00
Car03	$12*5$	93.15	83.00	103.00	16.05	11.00	24.00	3.00	1.00	7.00
Car04	$14*4$	40.10	31.00	60.00	9.75	5.00	12.00	0.25	0.00	1.00
Car05	$10*4$	37.50	34.00	39.00	21.55	18.00	25.00	0.35	0.00	3.00
Car06	$8*9$	18.00	18.00	18.00	19.10	17.00	22.00	10.25	6.00	16.00
Car07	$7*7$	20.00	20.00	20.00	23.15	19.00	29.00	1.95	0.00	12.00
Car08	$8*8$	37.00	37.00	37.00	29.40	24.00	37.00	0.25	0.00	5.00
Hel1	$100*10$	28.45	11.00	53.00	3.20	0.00	6.00	2.20	1.00	5.00
Hel2	$20*10$	36.85	26.00	52.00	8.10	2.00	13.00	1.75	1.00	7.00
Rec01	$20*5$	49.85	21.00	71.00	9.25	3.00	14.00	1.60	1.00	4.00
Rec03	$20*5$	97.55	69.00	127.00	8.95	3.00	15.00	2.70	0.00	7.00
Rec05	$20*5$	45.25	27.00	64.00	7.80	4.00	12.00	1.35	1.00	3.00
Rec07	$20*10$	37.75	23.00	51.00	13.25	9.00	21.00	0.40	0.00	2.00
Rec09	$20*10$	40.95	25.00	64.00	11.20	6.00	17.00	1.75	1.00	3.00
Rec11	$20*10$	55.45	34.00	79.00	13.15	7.00	19.00	0.60	0.00	2.00
Rec13	$20*15$	39.60	25.00	49.00	15.00	11.00	20.00	1.40	1.00	4.00
Rec15	$20*15$	30.75	25.00	36.00	11.65	8.00	18.00	0.05	0.00	1.00
Rec17	$20*15$	91.40	67.00	126.00	22.50	13.00	35.00	2.85	1.00	5.00
Rec19	$30*10$	53.10	14.00	83.00	9.55	4.00	22.00	1.90	1.00	4.00
Rec21	$30*10$	34.70	12.00	77.00	9.25	5.00	18.00	1.20	0.00	3.00
Rec23	$30*10$	49.10	19.00	76.00	8.90	2.00	15.00	1.35	1.00	3.00
Rec25	$30*15$	51.70	19.00	79.00	14.40	4.00	24.00	0.70	0.00	2.00
Rec27	$30*15$	36.55	12.00	61.00	10.45	6.00	17.00	2.15	1.00	4.00
Rec29	$30*15$	37.80	25.00	56.00	10.45	4.00	20.00	1.85	1.00	4.00
Rec31	$50*10$	37.95	16.00	72.00	6.75	2.00	10.00	2.45	1.00	4.00
Rec33	$50*10$	43.35	4.00	113.00	7.35	4.00	12.00	1.85	1.00	4.00
Rec35	$50*10$	38.60	16.00	84.00	7.05	3.00	13.00	2.05	1.00	4.00
Rec37	$75*20$	44.05	10.00	93.00	6.55	2.00	11.00	2.85	1.00	6.00
Rec39	$75*20$	64.70	14.00	117.00	7.45	4.00	11.00	2.00	1.00	3.00
Rec41	$75*20$	50.05	2.00	109.00	6.60	1.00	10.00	1.80	1.00	3.00
平均值		46.13	26.42	69.84	12.18	7.13	18.10	1.91	0.81	4.71

表 4 ESFLA, HDE和DPSO算法的R_{NDS}的比较

Table 4 Comparison of R_{NDS} among ESFLA ,HDE and DPSO algorithm

算 例		ESFLA			HDE			DPSO		
名称	$n*m$	AVG	MAX	MIN	AVG	MAX	MIN	AVG	MAX	MIN
Car01	$11*5$	1.00	1.00	1.00	0.90	0.67	1.00	0.07	0.00	0.44
Car02	$13*4$	0.97	0.89	1.00	0.73	0.41	0.93	0.28	0.08	0.58
Car03	$12*5$	0.99	0.93	1.00	0.66	0.46	0.86	0.25	0.10	
Car04	$14*4$	0.92	0.78	1.00	0.77	0.54	0.92	0.07	0.00	0.33
Car05	$10*4$	1.00	0.97	1.00	0.87	0.68	1.00	0.04	0.00	0.33
Car06	$8*9$	1.00	1.00	1.00	0.99	0.94	1.00	0.67	0.56	0.89
Car07	$7*7$	1.00	1.00	1.00	0.99	0.88	1.00	0.21	0.00	0.75
Car08	$8*8$	1.00	1.00	1.00	0.98	0.87	1.00	0.01	0.00	0.28
Hel1	$100*10$	0.87	0.29	0.98	0.86	0.00	1.00	0.39	0.13	0.83
Hel2	$20*10$	0.78	0.51	0.98	0.62	0.14	1.00	0.24	0.10	0.64
Rec01	$20*5$	0.76	0.33	0.96	0.69	0.33	1.00	0.27	0.13	0.50
Rec03	$20*5$	0.86	0.62	0.99	0.50	0.17	0.83	0.32	0.00	0.60
Rec05	$20*5$	0.82	0.56	0.98	0.64	0.35	0.91	0.23	0.10	0.40
Rec07	$20*10$	0.75	0.51	1.00	0.78	0.58	0.95	0.08	0.00	0.50
Rec09	$20*10$	0.70	0.50	0.90	0.67	0.33	0.93	0.23	0.08	0.60
Rec11	$20*10$	0.76	0.44	0.97	0.64	0.32	1.00	0.11	0.00	0.33
Rec13	$20*15$	0.91	0.69	1.00	0.90	0.69	1.00	0.26	0.13	0.67
Rec15	$20*15$	0.97	0.87	1.00	0.93	0.63	1.00	0.01	0.00	0.14
Rec17	$20*15$	0.76	0.58	0.96	0.70	0.47	0.92	0.34	0.20	0.56
Rec19	$30*10$	0.63	0.23	0.91	0.84	0.32	1.00	0.34	0.13	0.80
Rec21	$30*10$	0.58	0.15	0.97	0.77	0.38	1.00	0.24	0.00	0.50
Rec23	$30*10$	0.70	0.20	0.98	0.64	0.15	1.00	0.22	0.08	0.43
Rec25	$30*15$	0.79	0.17	0.89	0.78	0.31	1.00	0.13	0.00	0.33
Rec27	$30*15$	0.77	0.23	0.87	0.75	0.50	1.00	0.37	0.14	0.80
Rec29	$30*15$	0.64	0.44	0.96	0.65	0.29	0.94	0.33	0.14	0.50
Rec31	$50*10$	0.73	0.23	0.95	0.87	0.33	1.00	0.37	0.20	0.80
Rec33	$50*10$	0.52	0.05	0.93	0.85	0.40	1.00	0.38	0.14	0.67
Rec35	$50*10$	0.71	0.19	0.92	0.92	0.53	1.00	0.40	0.14	
Rec37	$75*20$	0.87	0.13	0.98	0.94	0.33	1.00	0.44	0.17	0.86
Rec39	$75*20$	0.78	0.13	0.97	0.99	0.82	1.00	0.38	0.13	0.75
Rec41	$75*20$	0.83	0.03	0.93	0.92	0.45	1.00	0.38	0.11	0.60
平均值		0.82	0.50	0.97	0.80	0.46	0.97	0.26	0.10	0.56

3) 由表4知ESFLA的R_{NDS}平均AVG, MIN和MAX的值分别为0.82, 0.50和0.97, 均大于HDE的0.80, 0.46, 0.9和DPSO的0.26, 0.10和0.56. 31个算例中, ESFLA的AVG大多数大于HDE的AVG值, 全部大于DPSO的AVG值, 即ESFLA的解被HDE和DPSO的解支配的少. 说明ESFLA与HDE和DPSO算法相比较, 得到的非支配解的质量更高. 根据以上分析可知ESFLA在解决此类问题时所得解的质量、多样性和有效性均优于HDE和DPSO.

7 结论(Conclusion)

多目标流水线调度问题较单目标问题更符合车间调度现状, 能更好地起到指导生产实践的作用. 根据多目标无等待流水线调度问题的特点, 提出了一种基于Pareto边界的离散蛙跳算法. 同时给出了性能指标的快速评价方法, 并结合快速局部搜索算法, 对SFLA算法进行改进. 仿真实验表明了所得算法的优越性.

参考文献(References):

[1] 潘全科, 朱剑英. 解决无等待流水线调度问题的变邻域搜索算法[J]. 中国机械工程, 2006, 17(16): 1741 – 1743.
(PAN Quanke, ZHU Jianying. A variable neighborhood search for no-wait flow shop scheduling[J]. *China Mechanical Engineering*, 2006, 17(16): 1741 – 1743.)

[2] 潘全科, 赵保华, 屈玉贵, 等. 一类解决无等待流水车间调度问题的蚁群算法[J]. 计算机集成制造系统. 2007, 9(21): 1801 – 1804.

(PAN Quanke, ZHAO Baohua, QU Yugui, et al. Ant-colony heuristic algorithm for no-wait flow shop problem with makespan criterion[J]. *Computer Integrated Manufacturing Systems*, 2007, 9(21): 1801 – 1804.)

[3] ELBELTAGI E, HEGAZY T, GRIERSON D. Comparison among five evolu -tionary- based optimization algorithms[J]. *Advanced Engineering Informatics*, 2005, 19(1): 43 – 53.

[4] 吴华丽, 汪玉春, 陈坤明, 等. 基于混合蛙跳算法的成品油管网优化设计[J]. 石油工程建设, 2008, 1(11): 14 – 16.
(WU Huali, WANG Yuchun, CHEN Kunming, et al. Optimal design of multi-product pipeline network by shuffled frog leaping algorithm. Petroleum Engineering Construction[J]. *Petroleum Engineering Construction*, 2008, 1(11): 14 – 16.)

[5] 朱光宇, 林蔚清. 基于改进混合蛙跳算法的贴片机贴装顺序优化[J]. 中国工程机械学报, 2008, 4(10): 428 – 432.
(ZHU Guangyu, LIN Weiqing. Mounting sequential optimization on surface mounting machine using improved hybrid frog-jumping algorithm[J]. *Chinese Journal of Construction Machinery*, 2008, 4(10): 428 – 432.)

[6] 陈功贵, 李智欢, 陈金富, 等. 含风电场电力系统动态优化潮流的混合蛙跳算法[J]. 电力系统自动化, 2009, 4(7): 25 – 30.
(CHEN Gonggui, LI Zhihuan, CHEN Jinfu, et al. SFL algorithm based dynamic optimal power flow in wind power integrated system[J]. *Automation of Electric Power Systems*, 2009, 4(7): 25 – 30.)

[7] PAN Q K, WANG L, QIAN B. A novel multi-objective particle swarm optimization algorithm for no-wait flow shop scheduling problems[J]. *Proceedings of the Institution of Mechanical Engineers Part B-Journal of Engineering Manufacture*, 2008, 222(4): 519 – 539.

[8] QIAN B, WANG L, HUANG D X, et al. An effective hybrid DE-based algorithm for multi-objective flowshop scheduling with limited buffers[J]. *Computers & Operations Research*, 2009, 36(1): 209 – 233.

作者简介:

潘玉霞 (1983—), 女, 助教, 从事智能计算及其应用研究, E-mail: panyuxia2008@163.com;

潘全科 (1971—), 男, 教授, 博士, 从事智能计算及其应用研究, E-mail: panquanke@lcu.edu.cn;

李俊青 (1976—), 男, 副教授, 从事智能计算及其应用研究, E-mail: lijunqing@lcu.edu.cn.

下 期 要 目