基于离散蛙跳算法的零空闲流水线调度问题求解

王亚敏[1,2]，冀俊忠[1]，潘全科[2]
（1. 北京工业大学 多媒体与智能软件技术北京市重点实验室，北京 100124；
2. 聊城大学 计算机学院，山东 聊城 252059）

摘 要：针对零空闲流水线调度问题，提出了一种新的离散化蛙跳求解算法. 该算法借助蛙跳算法优化机理，采用基于工件序列的编码方式和新的个体产生方法扩展了传统蛙跳算法的求解模型，并结合简化邻域搜索算法给出了3种改进策略. 仿真实验表明了所提算法及策略的有效性.

关键词：零空闲流水线调度；离散蛙跳算法；邻域搜索
中图分类号：TP 18 **文献标志码**：A **文章编号**：0254－0037(2010)01－0124－07

车间调度问题的求解及其优化技术是制造业实现生产过程合理化、自动化、集成化的基础和关键. 其中，零空闲流水线调度(no_idle flow shop，记为NIFS)是一类组合优化问题[1]；Baptiste等[2]提出了解决NIFS问题的分枝限界法；Kalczynski等[3]提出了解决NIFS问题的构造式启发方法；Bzraz等[4]提出了有效的改进贪心算法；一些基于离散粒子群优化算法[5]、差分进化算法[6]等相继被提出；Eusuff等[7]首次提出了蛙跳算法(shuffled frog leaping algorithm，记为SFLA)，并应用于水管网络扩充中管道尺寸最小化问题的求解. 由于蛙跳算法能结合生物种群的遗传行为和群落间信息交流的社会行为，是解决组合优化问题的有效工具[8].

本文提出了基于离散蛙跳算法的NIFS问题的求解方法. 采用基于工件序列的编码方式和新的个体产生方法扩展了传统蛙跳算法的求解模型，提出了解决NIFS问题的蛙跳算法，并给出了3种提高性能的改进策略.

1 NIFS问题描述及邻域搜索算法

1.1 NIFS问题描述

n个工件在m台机器上流水加工，每个工件在机器上的加工顺序相同，同时，约定每个工件在每台机器上只加工1次，而在同一机器上加工的相邻工件之间没有等待时间，且机器之间存在无限缓冲区. 在已知各工件在各台机器上所需加工时间的前提下，问题的求解目标是得到满足上述约束条件的可行调度，使得最大完工时间最短.

文献[5]提出了通过完工时间差或开工时间差来计算最大完工时间的方法. 给定单个工件的排序$\pi=\{\pi_1,\pi_2,\cdots,\pi_n\}$，而$\pi^e(i)=\{\pi_1,\pi_2,\cdots,\pi_i\}$和$\pi^j(i)=\{\pi_i,\pi_{i+1},\cdots,\pi_n\}$表示其中的2个部分排序. 令$F(\pi^e(i),j,j+1)$表示排序$\pi^e(i)$中机器$j$和$j+1$的完工时间差，$E(\pi^f(i),j,j+1)$表示排序$\pi^f(i)$中机器$j$和$j+1$的开工时间差，而$p(\pi_i,j)$表示工件$\pi_i$在机器$j$上的加工时间.

如图1(a)所示，当从前往后依次增加工件时最大完工时间

$$F(\pi^e(1),j,j+1)=p(\pi_1,j+1),\ j=1,2,\cdots,m-1 \tag{1}$$

收稿日期：2008-05-26.
基金项目：北京市自然科学基金资助项目(4083034)，北京市教育委员会科技发展资助项目(KM 200610005020).
作者简介：王亚敏(1979—)，女，山东菏泽人，讲师.

$$F(\pi^e(i),j,j+1)=\max\{F(\pi^e(i-1),j,j+1)-p(\pi_i,j),0\}+p(\pi_i,j+1)$$
$$i=2,3,\cdots,n,j=1,2,\cdots,m-1 \tag{2}$$

从前往后计算方式排列 π 的最大完工时间为

$$C_{\max}(\pi)=\sum_{j=1}^{m-1}F(\pi,j,j+1)+\sum_{i=1}^{n}p(\pi_i,1) \tag{3}$$

如图 1(b)所示，当从后往前依次增加工件时最大完工时间

$$E(\pi^f(n),j,j+1)=p(\pi_n,j),\quad j=1,2,\cdots,m-1 \tag{4}$$
$$E(\pi^f(i),j,j+1)=\max\{E(\pi^f(i+1),j,j+1)-p(\pi_i,j+1),0\}+p(\pi_i,j)$$
$$i=n-1,n-2,\cdots,1,j=1,2,\cdots,m-1 \tag{5}$$

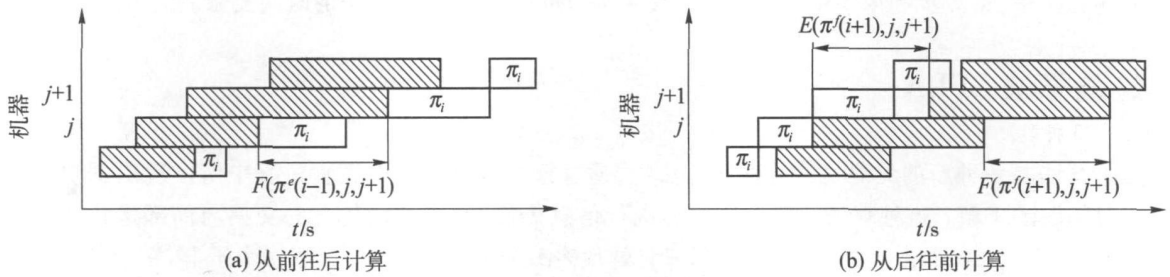

图 1　最大完工时间计算方法示意

Fig. 1　Sketch map of computing the maximum completion time for both cases

从后往前计算方式排列 π 的最大完工时间

$$C_{\max}(\pi)=\sum_{j=1}^{m-1}E(\pi,j,j+1)+\sum_{i=1}^{n}p(\pi_i,m) \tag{6}$$

求最大完工时间的复杂度为 $O(mn)$.

如图 1(b)所示，在一个排列之前增加工件时完工时间差

$$F(\pi^f(n),j,j+1)=p(\pi_n,j+1),\ j=1,2,\cdots,m-1 \tag{7}$$
$$F(\pi^f(i),j,j+1)=\max\{p(\pi_i,j+1)-E(\pi^f(i+1),j,j+1),0\}+F(\pi^f(i+1),j,j+1)$$
$$i=n-1,n-2,\cdots,1,j=1,2,\cdots,m-1 \tag{8}$$

1.2　邻域搜索算法

在排列 π 中，随机选择不同的 2 个位置 k、j，将 j 处的零件移动到 k 处，称为插入移动 $v(j,k)$，其中 $j,k\in\{1,2,\cdots,n\}$. 由移动可以得到原排列的一个邻居. 所有这样的邻居构成插入邻域，则该邻域的规模为 $n(n-1)$，评价该邻域的时间复杂度为 $O(mn^3)$.

如果将排列 π 看作是2 个排列 π_1 和 π_2 合并在一起得到的排列，其中 π_2 在 π_1 之后，如图 2 所示. 则

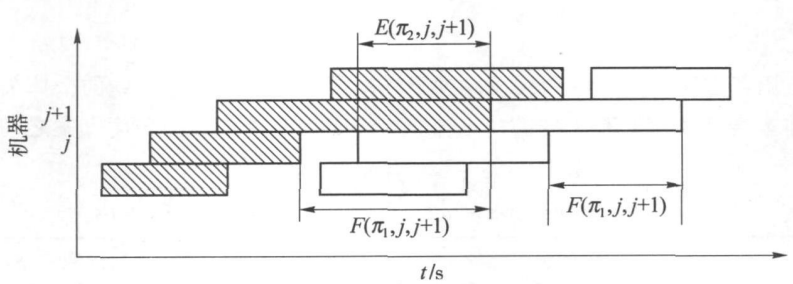

图 2　2 个排列合并在一起的最大完工时间计算方法示意

Fig. 2　Sketch map of computing the maximum completion time for combination two permutations

$$F(\pi,j,j+1)=\max\{F(\pi_1,j,j+1)-E(\pi_2,j,j+1),0\}+F(\pi_2,j,j+1)$$
$$j=1,2,\cdots,m-1 \tag{9}$$

若记排列 π 进行插入移动 $v(j,k)$ 之后的新排列为 π'，对比 π' 和 π 可得到邻域搜索算法.

第 1 步，按照式(1)和(2)从前往后依次求出每个 $\pi^e(i)$ 的完工时间差，按照式(4)、(5)和(7)、(8)从后往前依次求出每个 $\pi^f(i)$ 的完工时间差和开工时间差，进而得到排列 π 的 $C_{\max}(\pi)$.

第 2 步，取出 π 中的第 j 个工件，将其依次插入 π 的 k 处，其中 $k=1,2,\cdots,n$，且 $k\neq j$. 得到新 π'，则此时 π'可以看作由 k 之前的 $\pi^e(k-1)$、k 之后的 $\pi^f(k+1)$和 k 处的工件 j 3 部分组成.

第 3 步，在 $\pi^e(k-1)$后面增加工件 j 得到 π'的 $\pi'^e(k)$，由式(2)可以求出 $\pi'^e(k)$的完工时间差；π'的另一部分 $\pi'^j(k+1)$与 $\pi^f(k+1)$相同. 由式(9)可求出 π'的完工时间差 $F(\pi',j,j+1)$，由式(3)得 π'的 $C_{\max}(\pi')$.

第 4 步，重复第 2 和第 3 步，直到将排列 π 的插入邻域搜索一遍.

上述算法中，第 2 步和第 3 步的时间复杂度为 $O(mn)$. 所以整个算法的时间复杂度为 $O(mn^2)$.

2 蛙跳算法 SFLA

蛙跳算法是一种新的元启发式的搜索算法[6]，通过模拟青蛙群体在觅食过程中所体现出的协同行为来完成对问题的求解. 这种算法按照族群分类进行信息传递，并将全局信息的交换与局部进化搜索相结合. 在蛙跳算法中，种群由很多青蛙组成，每只青蛙代表 1 个解. 种群被分成了多个子群，每个子群包括一定数量的青蛙，称为 1 个 memeplex. 在每个子群内分别执行局部搜索. 每只青蛙都受自己和其他青蛙想法的影响，并通过 memetic 进化来调整位置. 经过一定数量的进化后，不同子群间的青蛙通过跳跃过程来传递信息. 这种局部进化和跳跃过程相间进行，直到满足收敛的条件为止.

首先，由随机初始化一组解组成青蛙的初始种群，然后，将所有青蛙按照它们的适配值降序排列，并分别放入各个 memeplex 中，用 P_b 和 P_w 分别表示该子群中位置(适应值)最好和最坏的青蛙. 另外，用 P_g 表示整个种群中最好的青蛙. 在每一轮的进化中，通过与 PSO 算法近似的方法改善最坏青蛙 P_w 的位置.

青蛙移动的距离

$$D_i = \text{rand}() * (P_b - P_w) \tag{10}$$

新的位置

$$P_w = P_w(\text{当前位置}) + D_i,\ (D_{\max} \geqslant D_i \geqslant -D_{\max}) \tag{11}$$

式中，rand() 为 0 到 1 之间的随机数；$D_{\max}$为允许青蛙移动的最大距离. 如果这个过程能产生一个较好解，那么就用新位置的青蛙取代原来的青蛙 P_w；否则，用 P_g 代替 P_b，重复上述过程. 如果上述方法仍不能生成更好的青蛙，那么就随机生成一个新解取代原来最坏的青蛙 P_w. 按照这种方法执行一定次数的进化.

最后，将所有的青蛙重新排序、子群划分，继续进化、跳跃，重复上述过程直到收敛为止.

3 NIFS 问题的离散蛙跳求解算法 DSFLA

3.1 个体矢量编码

建立个体矢量与调度方案之间的映射关系. 对于零空闲流水线编码方案就的个体矢量的每一维表示 1 个工件. 这样，个体本身就表示所有工件的一个排列. 表 1 为个体矢量与调度方案之间的对应关系.

表 1 个体矢量及对应的工件排列

Table 1 Individual vector and corresponding job permutations

个体矢量维数	个体矢量	工件序列	个体矢量维数	个体矢量	工件序列	个体矢量维数	个体矢量	工件序列
1	3	3	3	2	2	5	6	6
2	1	1	4	4	4	6	5	5

3.2　个体矢量的更新

青蛙所代表的解向量在连续解空间跟踪其局部极值或全局极值的向量运算. 换句话说，式(10)、(11)表示 P_w 向 P_b 学习逼近的过程，其中 rand()表示学习的程度，将式(10)、(11)合并可知

$$P_w = P_w(\text{当前位置}) + \text{rand}() * (P_b - P_w(\text{当前位置})) \tag{12}$$

式中，rand()代表青蛙 P_w 从局部极值 P_b 的信息继承度，反映了对 P_b 信息的置信指标.

当 rand() = 1 时，$P_w = P_b$，当 rand() = 0 时，$P_w = P_w$(当前位置). 定义 $f(P_w, P_b)$ 为 P_w 向 P_b 的学习过程，该过程可以通过交叉操作实现.

1) 在 P_b 中随机选择 1 个交叉区域，其中 rand()决定着交叉区域的大小.

2) 将 P_b 的交叉区域加到 P_w 的前面或后面，并删除 P_w 中已在 P_b 的交叉区中出现过的数字.

采用这种更新策略，子串能继承父串的有效模式，实现了从局部极值 P_b 获得更新信息的目的.

3.3　工序编码的 DSLFA 调度算法

Step 0　初始化. 选择 M 和 N，M 表示 memeplex 的数量，即子群的数量，N 表示每个 memeplex 中青蛙的个数. 那么，整个种群的数量 $F = M \times N$.

Step 1　生成初始种群. 在可行解空间 $\Omega \subset \mathscr{R}^n$，生成 F 个青蛙 $U(1), U(2), \cdots, U(F)$，其中 n 为工件数量. 每个青蛙(Frog)原本代表青蛙的当前位置，对于 NIFS 问题则表示解空间的 1 个候选解. 第 i 个青蛙可以表示为 $U(i) = (U_i^1, U_i^2, \cdots, U_i^n)$. 计算出工件序列 $U(i)$ 的最大完工时间 makespan，用 $f(i)$ 表示. 则 $f(i)$ 的值越小，表示该解的适应度越高.

Step 2　对青蛙划分等级. 将 F 只青蛙按照适应度的降序排列，生成数组 $X = \{U(i), f(i), i = 1, 2, \cdots, F\}$，这样的话，$i = 1$ 表示这只青蛙的位置(适应度)最好，记录下种群中位置最好青蛙 $P_g = U(1)$.

Step 3　将青蛙分组，放入不同的 memeplex. 将数组 X 分成 M 个 memeplex：$Y_1, Y_2, \cdots, Y_M$. 每个 memeplex 中包含 N 只青蛙，即：$Y_k = \{U(j)_k, f(j)_k \mid U(j)_k = U(k + M(j-1)), f(j)_k = f(k + M(j-1)), j = 1, \cdots, N\}$，其中 $k = 1, 2, \cdots, M$；比如 $M = 3$，那么第 1 只青蛙属于子群 memeplex 1，第 2 只青蛙属于子群 memeplex 2，第 3 只青蛙属于子群 memeplex 3，第 4 只青蛙属于子群 memeplex 1，等等.

Step 4　在每个 memeplex 中执行 memetic 进化. 在每个 memeplex 中，每只青蛙受到其他青蛙想法的影响，通过 memetic 进化，使得各个青蛙朝目标位置逼近.

Step 4 - 0 设 $i_M = 0$，i_M 表示对 memeplex 的计数，在 0 到 M 之间变化，与 memeplex 的数量 M 比较. 设 $i_N = 0$，i_N 表示进化次数，与每个 memeplex 中允许的最大进化次数 $N_{\max}$ 比较. 用 P_b 和 P_w 分别表示每个 memeplex 中位置(适应度)最好的和最坏的青蛙，用 P_g 表示整个种群中最好的青蛙. 在每轮的进化中，改善最坏青蛙 P_w 的位置，注意，并非对所有的青蛙都优化.

Step 4 - 1　$i_M = i_M + 1$.

Step 4 - 2　$i_N = i_N + 1$.

Step 4 - 3　采用式(12)调整最坏青蛙的位置.

Step 4 - 4　如果 Step 4 - 3 能使得青蛙有一个更好的位置，即能产生一个更好的解，那么就用新位置的青蛙取代原来的青蛙，执行 step 4 - 6；否则，用 P_g 代替 P_b，利用式(12)计算 $f(P_w, P_g)$，如果能产生一个更好的解，取代原来的青蛙，执行 step 4 - 6.

Step 4 - 5　随机生成一个新解取代原来最坏的青蛙 P_w.

Step 4 - 6　如果 $i_N < N_{\max}$，那么执行 4 - 2.

Step 4 - 7　如果 $i_M < M$，那么执行 4 - 1，否则执行 Step 5.

Step 5　青蛙在 memeplex 之间跳跃移动. 在每个 memeplex 中执行了一定次数的 memetic 进化之后，将各个子群 $Y_1, Y_2, \cdots, Y_M$ 合并到 X，即 $X = \{Y_k, k = 1, 2, \cdots, M\}$. 将 X 重新按降序排列，并更新种群中最好的青蛙 P_g.

Step 6　检查终止条件. 如果迭代终止条件满足，则停止. 否则，重新执行 step 3. 一般情况下，当执

行了一定次数的循环进化,代表最好解的青蛙不再改变的时候,算法停止. 有时也定义最大进化次数作为停止标准.

3.4 蛙跳算法的改进

3.4.1 引入邻域搜索算法的改进策略

蛙跳算法中青蛙从局部极值或全局极值中获得更新信息,其信息共享机制中信息的流动是单向的,信息流动的目的性更强,效率更高. 同时,搜索过程受局部极值 P_b 以及全局极值 P_g 的影响较大. 加强 P_b 和 P_g 的局部探测能力可提高算法性能. 改进 SFLA 的方法是在每组 memeplex 进化过程中,对 P_b 执行简化插入邻域搜索算法,如果搜索之后发现更好的解,则用该解代替原来的 P_b,如果新得到的 P_b 优于 P_g,更新 P_g. 并且在整个种群的每次迭代中,对 P_g 也执行简化邻域搜索算法. 通过对 P_b 和 P_g 的细搜索,使得整个种群能更快的向最优解移动,有利于增加算法的收敛速度,记为 DSFLA1.

3.4.2 增加随机扰动的改进策略

DSFLA1 虽然增强了局部搜索能力,但细搜索范围仅限于 P_b 和 P_g 的最近邻居,范围较小,有可能使 P_b 和 P_g 长时间"徘徊"在若干旧状态上,容易陷入局部最优. 为了增强算法跳出局部最优的能力,可以在 1 次邻域搜索之后,对 P_b 和 P_g 执行 1 次随机插入移动,对所得结果再执行邻域搜索算法,如此将随机插入和邻域搜索过程重复几次,即通过增加扰动来扩大细搜索的范围. 虽然细搜索的时间有所增加,有利于优化过程中状态的局部小范围趋化性移动,从而增强了算法在解空间的探索能力和效率. 记为 DSFLA2.

3.4.3 结合模拟退火机理的改进策略

正是因为种群进化受 P_b 和 P_g 影响,一旦为局部最优,青蛙将难以摆脱局部极值,其信息将覆盖整个邻域种群,导致该种群以较大概率陷入局部收敛. 由此,还可以借鉴模拟退火中的概率接收[9]准则优化 P_b 和 P_g,即在 DSFLA2 的基础上,在对 P_b 和 P_g 进行多次邻域搜索后,如果找到更好的解,那么就接受这个解,如果找不到更好的解,就以一定概率接受次优解. 这种在 DSFLA2 中 SA 的嵌入,赋予优化过程在各状态具有可控的概率突跳特性,尤其在高温时使得算法具有较大的突跳性,是避免 P_b 和 P_g 陷入局部极小和算法"早熟收敛"的有力手段,记为 DSFLA3.

4 仿真试验

采用包含 120 个流水线调度的 Taillard Benchmark 问题作为实例,每个实例计算 5 次,求算法所得解的平均相对偏差(PRD)和平均方差(SD). 设种群规模为 20,因算法受初始解情况的影响,如果初始解先通过某种智能优化算法优化生成,算法的收敛性及其他性能会有所提高. 但是,为了检验该算法的鲁棒性,初始解采用随机生成的方法. 对局部极值 P_b 和全局极值 P_g 执行 3 次随机插入移动和邻域搜索,模拟退火接受标准的温度系数 $t = 0.05 \sum_{i=1}^{n}\sum_{j=1}^{m} P_{ij}/(mn)$. 并与基于离散粒子群优化 DPSO 算法进行了比较. 实验的运行环境为:操作系统 Windows XP,CPU 为 PⅣ 1.5 GHz,内存为 256 MB,算法用 C++语言编程. 各算法采用相同的终止条件,即最大运行时间为 10 ms. 结果如表 2 所示.

DSFLA 算法所得的平均相对偏差和平均方差近似于 DPSO 算法,由此表明具有信息共享、协同进化机制的 DSFLA 能在较短时间内较快得向最优解逼近,是一种有效的解决 NIFS 问题的方法. 由 DSFLA 和 DSFLA1、DSFLA2 和 DSFLA3 各个算法的对比分析可以看出,通过简化邻域搜索算法提高局部极值和全局极值之后,平均偏差减少,解的质量有所提高,尤其是加入随机扰动对算法性能提高有很大作用. 并可以看出,DSFLA2 和 DSFLA3 的平均方差明显减少,表明这 2 种改进策略使得算法的稳定性有所提高. 原因在于 P_b 和 P_g 对种群的进化搜索过程起着引导作用,其性能对算法结果影响很大,所以 DSFLA 和 DSFLA1 算法性能对 P_b 和 P_g 的依赖性较大,而对于改进策略 DSFLA2 和 DSFLA3,由于在对 P_b 和 P_g 邻域搜索时增加了扰动,扩大了搜索范围,从而使得算法在每一代进化过程中对 P_b 和 P_g 的初始性能的依赖性降低,进而使得算法的稳定性提高.

表 2　不同算法的比较

Table 2　Comparison of different algorithms

Instance	DPSO		DSFLA		DSFLA1		DSFLA2		DSFLA3	
$n \times m$	PRD	SD	PRD	SD	PRD	SD	PRD	SD	PRD	SD
20 × 5	9. 7	1. 51	9. 56	1. 91	9. 66	2. 2	5. 41	1. 21	5. 36	1. 48
20 × 10	23. 92	2. 71	23. 64	2. 96	23. 35	2. 44	15. 38	1. 6	15. 68	1. 71
20 × 20	48. 49	2. 17	49. 3	2. 83	49. 16	3. 33	37. 35	2. 57	38. 4	1. 9
50 × 5	6. 94	1. 69	7. 32	1. 44	5. 99	1. 33	3. 7	0. 95	3. 53	1. 16
50 × 10	22. 82	2. 24	23. 85	2. 19	20. 95	2. 29	17. 04	2. 09	16. 61	1. 96
50 × 20	55. 24	3. 34	55. 9	3. 24	50. 21	3. 53	41. 72	3. 61	40. 6	3. 2
100 × 5	4. 15	1. 01	4. 75	1. 12	3	0. 88	1. 95	0. 89	1. 81	0. 74
100 × 10	20. 05	2. 29	21. 01	2. 11	16. 26	1. 45	13. 7	1. 64	13. 2	1. 66
100 × 20	46. 69	2. 63	48. 85	2. 6	40. 07	3. 03	35. 5	2. 97	34. 45	2. 87
200 × 10	15. 13	1. 11	16. 03	1. 29	11. 31	1. 46	10. 09	1. 21	10. 24	1. 22
200 × 20	38. 16	1. 55	40. 3	2. 18	30. 31	2. 76	29. 83	1. 95	28. 57	2. 12
500 × 20	26. 25	1. 09	27. 91	1. 25	23. 2	1. 13	23. 69	1. 37	23. 3	1. 13

分别选取一个 50 × 20 的中等规模和大规模的 NIFS 问题进行分析，各算法的进化过程如图 3 所示.

图 3(a)为 50 × 20 的中等规模 NIFS 问题 Ta051 的各算法进化曲线. 对于中等规模的问题，DSFLA 具有较好的计算性能和较强的全局搜索能力. 与 DSFLA 相比较，DSFLA1 明显加快了简化邻域搜索的收敛速度，由于 P_b 和 P_g 搜索范围不大，易陷入局部最优，进而整个种群也更易陷入局部极值. 加入随机扰动后的 DSFLA2 和 DSFLA3，扩大了 P_b 和 P_g 的搜索范围，不仅收敛速度更快，而且易于跳出局部极值，并且融合了模拟退火机理的改进策略对于跳出局部最优也有一定作用. 由此可以看出，DSFLA2 和 DSFLA3 在搜索广度和深度上能达到较好的均衡，是解决 NIFS 这类组合优化问题的有效工具.

图 3(b)为 500 × 20 的大规模 NIFS 问题 Ta111 的各算法进化曲线. 对于大规模问题，虽然 DSFLA 收敛速度比 DPSO 稍慢一些，但是从试验中看出相对不易陷入局部极值，依然是一种解决 NIFS 问题的有效工具. 这是由于 DSFLA 中对于较差解可能有变异操作，这对保持种群多样性可以起到一定的作用，能在一定程度上减少陷入局部极值的可能. 而对于 3 种改进策略，尽管较大规模问题的邻域搜索算法本身需要占用较多的时间，对整体的收敛进度依然起到了极大的促进作用，使得收敛速度有较大提高.

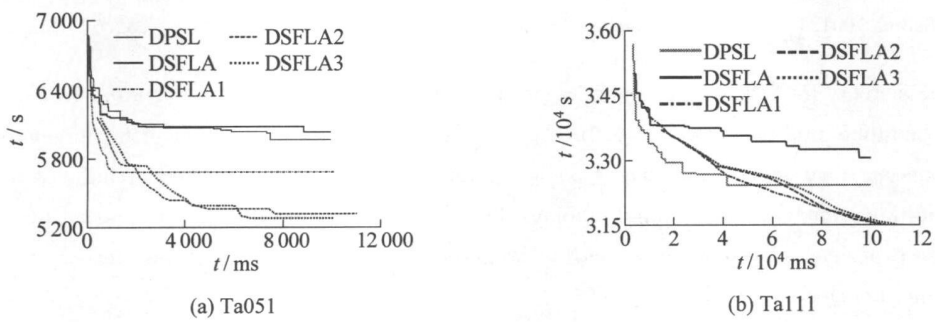

图 3　各算法进化曲线比较

Fig. 3　Evolution curves of different algorithms

5 结论

蛙跳算法结合了粒子群优化算法的社会特性和 memetic 算法的遗传特性，具有连续性和较强的全局搜索能力. 本文基于该算法的优化机理，提出了适合于解决零空闲流水线调度问题的离散蛙跳算法，并研究了几种改进策略，仿真试验表明，离散蛙跳算法是解决零空闲流水线调度问题的有效方法，几种改进策略都有助于改善解的质量和提高算法的收敛速度.

参考文献：

[1] SAADANI N E I，GUINET A，MOALA M. A traveling salesman approach to solve the f/no_idle/cmax problem[J]. European Journal of Operation Research，2005，161：11-20.

[2] SAADANI N E H，BAPTISETE P，MOALLA M. The simple F2 // Cmax with forbiddden tasks in first or last position：A problem more complex than it seems[J]. Eur J Oper Res，2005，161：21-31.

[3] KALCZYNSKI P J，KAMBUROWSKI J. A heuristic for minimizing the makespan in no-idle permutation flow shop[J]. Comput Ind Eng，2005，49：146-154.

[4] BZRAZ D，MOSHEIOV G. A note on a greedy heuristic for the flow-shop makespan minimization with no machine idle-time[J]. Eur J Oper Res，2008，184(2)：810-813.

[5] PAN Quan-ke，WANG Ling. No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm[J]. The International Journal of Advanced Manufacturing Technology，2008，39(7-8)：796-807.

[6] PAN Quan-ke，WANG Ling. A novel differential evolution algorithm for no-idle permutation flow-shop scheduling problems[J]. European J. of Industrial Engineering，2008，2(3)：279-297.

[7] EUSUFF M M，LANSEY K E. Optimization of water distribution network design using the shuffled frog leaping algorithm[J]. Water Resour Plan Manage，2003，129(3)：210-225.

[8] ELBELTAGI E，HEGAZY T，GRIERSON D. Comparison among five evolutionary-based optimization algorithm[J]. Advanced Engineering Informatics，2005，19(1)：43-53.

[9] 王凌. 智能优化算法及其应用[M]. 北京：清华大学出版社，2001，10.

An Algorithm Based on Discrete Shuffled Frog Leaping for No _ Idle Permutation Flow Shop Scheduling Problem

WANG Ya-min[1,2]，JI Jun-zhong[1]，PAN Quan-ke[2]

(1. Beijing Municipal Key Laboratory of Multimedia and Intelligent Software Technology，Beijing University of Technology，Beijing 100124；2. College of Computer Science，Liaocheng University，Liaocheng，252059，China)

Abstract：A Discrete Shuffled Frog Leaping Algorithm (DSFLA) is proposed to solve the No_Idle permutation Flow Shop scheduling problems (NIFS). In the light of the optimization mechanism of general SFLA，the algorithm adopts an encoding scheme based on job permutation and a new method of individual production to extend the traditional model of SFLA，and employs the simple neighborhood search to present three improvement strategies. The experimental results show that the proposed algorithm and its strategies are effective and efficient in different scale benchmarks of NIFS.

Key words：no_idle flow shop；discrete shuffled frog leaping algorithm；neighborhood search

(责任编辑 张士瑛)