

文章编号:1006-2467(2020)12-1291-09

DOI: 10.16183/j.cnki.jsjtu.2019.321

求解零空闲流水车间调度问题的离散正弦优化算法

赵 芮, 顾幸生

(华东理工大学 化工过程先进控制和优化技术教育部重点实验室, 上海 200237)

摘 要: 针对以最小化最大完工时间(makespan)为目标的零空闲流水车间调度问题(NIFSP),提出一种离散正弦优化算法(DSOA)进行求解.受正弦波形的启发,原始的正弦优化算法(SOA)是一种利用正弦函数对个体位置进行更新的全局优化算法.首先,重新定义了适应组合优化问题的位置更新策略,采用一种去除工件数大小可变的迭代贪婪算法来对个体位置进行更新,以提高算法的探索能力.其次,采用了交叉操作和保留精英解的选择策略,避免算法陷入局部最优.最后,为了提高局部搜索的开发能力和算法精度,引入了一种基于插入的局部搜索方法,以便于在当前最优解的周围寻找更好的解.此外,基于 Taillard 基准,给出了算法性能比较的仿真结果,实验结果验证了所提出的 DSOA 算法求解 NIFSP 的有效性.

关键词: 生产调度;正弦优化算法;零空闲流水车间调度问题;迭代贪婪算法;最大完工时间;智能优化算法;局部搜索

中图分类号: TP 278 **文献标志码:** A

A Discrete Sine Optimization Algorithm for No-Idle Flow-Shop Scheduling Problem

ZHAO Rui, GU Xingsheng

(Key Laboratory of Advanced Control and Optimization for Chemical Process of the Ministry of Education, East China University of Science and Technology, Shanghai 200237, China)

Abstract: Aimed at the no-idle flow-shop scheduling problem (NIFSP) with minimized makespan, a discrete sine optimization algorithm (DSOA) is proposed. Inspired by sine waveforms, the original sine optimization algorithm (SOA) is a global optimization algorithm, which uses the sine function to update the position of search agents. First, the update position strategy to adapt to the combinatorial optimization problem is redefined. An iterated greedy algorithm with a variable removing size is employed to update the position to enhance the exploration ability. Then, a crossover strategy and a selection strategy are applied to avoid the algorithm falling into local optimum. Next, to improve the exploitation ability of local search and the accuracy of the algorithm, an insertion-based local search scheme is applied in DSOA to search for a better solution around the current optimal solution. Finally, based on the Taillard benchmark, the simulation results of performance comparisons are presented. The experimental results demonstrate the effectiveness of the proposed DSOA algorithm for solving NIFSP.

Key words: production scheduling; sine optimization algorithm; no-idle flow-shop scheduling problem

收稿日期: 2019-11-11
基金项目: 国家自然科学基金(61973120,61773165,61673175,61573144)资助项目
作者简介: 赵 芮(1994-),女,重庆市人,硕士生,研究方向为生产调度.
通信作者: 顾幸生,男,教授,博士生导师,电话(Tel.):021-64253463;E-mail:xsgu@ecust.edu.cn.

(NIFSP); iterated greedy algorithm; makespan; intelligent optimization algorithm; local search

流水车间调度问题(FSP)是被广泛研究的组合优化问题,在制造系统和工业过程中起着非常重要的作用^[1].在Johnson^[2]的开创性工作之后,学者们提出了许多解决FSP的方法.作为FSP的重要分支,零空闲流水车间调度问题(NIFSP)假定机器不间断地进行工作,而不必在相邻的工件之间等待时间,广泛存在于实际生产过程^[3-4].在传统行业如陶瓷熔块生产、玻璃纤维加工、铸造及集成电路生产中,由于某些制造环境中的技术要求或是使用了昂贵的机械设备,使得正在加工工件的机器不允许被停止.

早在1997年,Baptiste等^[5]就证明了当机器数大于2时,使makespan准则最小的NIFSP是一个NP(非确定性多项式)难问题.因此如果只用精确算法和构造性启发式算法进行求解,往往难以获得理想的求解效果.在之前的研究中,国内外的学者提出了一些智能优化算法来求解NIFSP.为最小化最大完工时间,文献[6]提出了一种混合离散粒子群(HDPSO)算法来求解NIFSP.文献[7]提出混合离散差分进化(HDDE)算法.二者都采用了评估插入邻域的加速方法,以降低时间复杂度.针对最大完工时间和总流水时间(TFT)两个目标准则,文献[8]利用差分进化算法来优化可变迭代贪婪算法的参数,提出了一种被称为vIG_DE(variable Iterated Greedy Algorithm with Differential Evolution)的算法来求解NIFSP,并在Ruiz基准测试集上进行了测试.结果表明,vIG_DE算法与HDDE、HDPSO算法相比性能有所提高.文献[9]利用离散人工蜂群(DABC)算法来求解以总拖期时间(TTd)为目标的NIFSP,并首次在Taillard基准测试上为TTd准则提供了已知最优解.文献[10]提出入侵杂草优化(IWO)算法,实验结果表明,在Taillard测试集上,IWO算法的性能优于迭代贪婪(IG)算法和HDP-
SO算法,但其没有使用任何局部搜索方法.文献[11]提出双种群分布估计算法(Bi-population EDA, BEDA),以最小化TTd,BEDA由两个子种群协同工作,通过对不同的概率模型进行采样来生成两个子种群.实验表明,BEDA优于DABC算法.最近,针对以最小化最大完工时间为目标的NIFSP,文献[12]提出了一种带有局部搜索的离散烟花算法(DFWA-LS),实验结果表明DFWA-LS算法在寻优精度和稳定性上均不劣于HDPSO和IWO

算法.文献[13]提出具有混合节点和边缘直方图矩阵的模因算法,以最小化最大完工时间;文献[14]提出基于元启发式的混合离散学习算法,以最小化TTd.

受正弦余弦波形的启发,Mirjalili^[15]提出了一种基于智能群体的正弦余弦算法(SCA),用以求解优化问题.此后,为了提高SCA的性能,Meskat等^[16]在SCA的基础上提出了一种新的位置更新策略,该方法仅用正弦函数对个体位置进行更新,因此称为正弦优化算法.研究表明,相比SCA,正弦优化算法(SOA)具有更快的收敛速度以及更高的寻优精度.到目前为止,对SOA的研究还很少,尚未应用于实际优化问题.但SCA在各种优化问题上的成功应用,例如参数优化^[17]、风速预测^[18]、电力系统的最优潮流问题^[19],可以揭示出SOA也有解决优化问题的潜力.

据知,目前尚无关于SOA求解NIFSP的研究成果.本文提出一种有效的基于IG算法的离散正弦优化算法来求解NIFSP,其目标是最小化最大完工时间.本文特别之处在于,将IG与SOA的位置更新策略相结合,并提出了新的位置更新策略来提高算法的搜索能力.加入了个体与当前最优解之间的交叉操作以增加算法多样性,并根据种群数的增加定义了选择策略,保留了种群中优秀的个体信息.此外,还采用了一种基于插入的局部搜索算法以增强局部开发能力.实验表明,采用本文提出的离散正弦优化算法(DSOA)解决NIFSP是可行且有效的.

1 零空闲流水车间调度问题

NIFSP描述如下: n 个工件集合 $J = \{1, 2, \dots, n\}$ 按照相同的顺序在 m 台机器集合 $M = \{1, 2, \dots, m\}$ 上进行加工.在任意时刻,每台机器最多加工一个工件,每个工件最多只能被一台机器加工^[20].为了遵循零空闲的约束,机器 i 加工第 j 个工件的开始时间必须等于第 $j-1$ 个工件的加工完成时间,即每台机器加工任意两相邻工件时没有空闲时间.

若用 $\pi = \{\pi(1), \pi(2), \dots, \pi(n)\}$ 来表示一个工件排序,用 $\pi^E(j) = \{\pi(1), \pi(2), \dots, \pi(j)\}$ 来表示工件排序 π 的部分序列,其中 $1 < j < n$.用 $p(\pi(j), i+1)$ 来表示工件 $\pi(j)$ 在机器 i 上的加工时间,用 $F(\pi^E(j), i, i+1)$ 来表示序列 $\pi^E(j)$ 在相邻机器 i 和 $i+1$ 上加工完成后的完工时间之差.最大完工时间

可以用以下公式计算:

$$F(\pi^E(1), i, i+1) = p(\pi(1), i+1)$$
$$i = 1, 2, \cdots, m-1$$

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

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

式中:最大完工时间 $C_{\max} = p(\pi(1), 1) + p(\pi(2), 1) + F(\pi^E(2), 1, 2) + F(\pi^E(2), 2, 3)$. 为了便于理解,将 2 个工件、3 台机器的 C_{\max} 计算过程在图 1 中给出,图中 t 为机器加工工件所花费的时间.

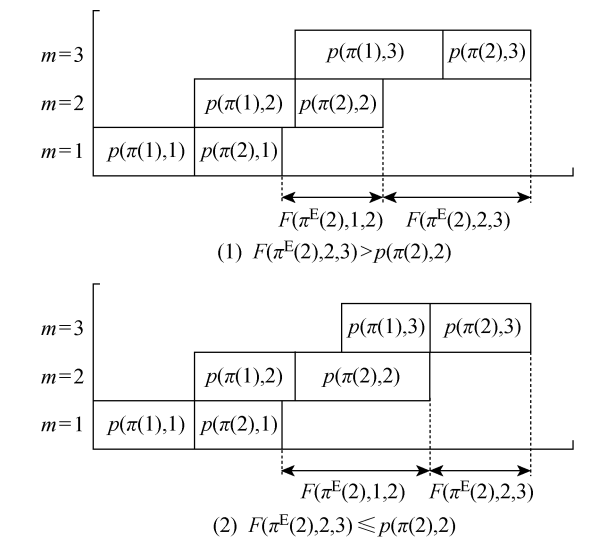


图 1 C_{\max} 计算示意图
Fig. 1 Computation of C_{\max}

2 基本正弦优化算法

Meshkat 等^[16]在 2017 年提出的 SOA 是一种新颖的基于智能群体的随机优化算法,单目标实参数数值优化的结果表明,与 SCA 相比,SOA 的收敛速度更快,精度更高. SOA 构造多个随机的初始候选解,并使用基于正弦函数的数学模型对它们进行优化. 该算法包括一些随机变量和自适应变量,以平衡优化过程中的全局探索和局部开发. 由于随机优化算法将优化问题视为黑箱,使得 SOA 具有更大的灵活性,这意味着 SOA 可轻松应用于不同领域的问题.

SOA 使用一组随机解开始优化过程,假设在 n 维搜索空间中有 NP 个个体,则第 i 个个体的位置表示为

$$X_i = (x_i^1, x_i^2, \cdots, x_i^k, \cdots, x_i^n) \quad i = 1, 2, \cdots, \text{NP}$$

在 SOA 中,使用正弦函数来更新个体的位置:

$$x_i^k(T+1) = \begin{cases} x_{\text{rand}}^k(T) + r_1 \sin r_2, & r_3 < 0.5 \\ x_{\text{best}}^k(T) + r_1 \sin r_2, & r_3 \geq 0.5 \end{cases}$$

式中: $x_{\text{rand}}^k(T)$ 表示第 T 次迭代时的一个随机个体的第 k 维位置分量; $x_{\text{best}}^k(T)$ 表示第 T 次迭代后种群中最优个体的第 k 维位置分量; r_1 和 r_2 决定了当前个体下一位置移动方向上的移动速度, r_1 是线性递减函数,表达式为

$$r_1 = a \left(1 - \frac{T}{T_{\max}} \right)$$

式中: T_{\max} 为最大迭代次数; a 为常数,一般取值为 2. 优化过程开始时, r_1 的值较大,导致 $r_1 \sin r_2$ 有较强的波动,使位置的随机变化量更大,可以更好地进行全局探索. 随着时间的推移, r_1 的值减小,相应个体位置的随机变化量也会减小,可以更好地进行局部开发; r_2 是 $[0, 2\pi]$ 之间的随机数,它规定了个体下一个位置移动的方向; r_3 是 $[0, 1]$ 之间的随机数,决定了当前个体下一位置的移动起点. 当 $r_3 \in [0, 0.5]$ 时,使用式 (5) 中第一个公式更新位置,这时个体的下一位置将等于随机个体的修改位置. 换言之,在位置更新时首先随机地选择一个个体作为移动起点,然后用 $r_1 \sin r_2$ 对其进行修改,得到新的位置. 这样一来,就能够很好地覆盖所有搜索区域,从而增强了算法的全局探索能力. 当 $r_3 \in [0.5, 1]$ 时,使用式 (5) 中第二个公式更新位置,这时个体的下一位置将等于最优个体的修改位置. 如此,能够在最优个体周围的区域进行搜索,从而增强了算法的局部开发能力.

基于以上分析,SOA 的伪代码为

```
Initialize a set of search agents
While ( $T < T_{\max}$ )
    Evaluate each of the search agents
    Update the best solution
    Update  $r_1$ ,  $r_2$  and  $r_3$ 
    Update the position of search by Formula (5)
End while
Return the best solution as the global optimum
```

3 离散正弦优化算法

3.1 编码及初始化评价

DSOA 采用基于工件排序的自然数编码,例如,个体 $X_i = \{3, 5, 1, 4, 2\}$ 表示工件在机器上的加工顺序为 $\pi = \{3, 5, 1, 4, 2\}$. 初始化种群均随机产生.

3.2 位置更新策略

在 SOA 中,每一次的迭代过程,都需要对所有个体的位置进行更新. 而个体位置的更新可以看作

是对随机个体(或最优个体)位置的一次修改,位置修改的变化量取决于 $r_1 \sin r_2$ 波动的幅度. $r_1 \sin r_2$ 波动的幅度越大,其邻域搜索空间越大.

为了解决组合优化问题,IG 算法作为一种简单、有效且易于适应的算法是非常可取的. 基于 NEH(Nawaz-Ensore-Ham)的基本原理,IG 算法可以通过不断地进行破坏操作和重构操作来获得更好的解. 在破坏(Destruction)阶段,首先从 π 中移除随机选择的 d 个工件,产生两个排序: π^R 和 π^D ,其中 π^R 包含 d 个移除工件, π^D 包含剩余的 $n-d$ 个工件. 在重构(Construction)阶段,将 π^R 的第一个工件插入 π^D 中,产生 $n-d+1$ 个部分解,选择其中最优化用于下一次迭代;接下来将第二个工件插入 π^D 中,……,直至 π^D 中包含 n 个工件,形成完整的调度解.

在 DSOA 中,将 IG 算法的思想用于个体的位置更新,把 IG 的迭代过程视为个体的位置修改操作. 每次迭代后,参数 d 更新如下:

$$d = \lfloor \lfloor r_1 \sin r_2 + 0.5 \rfloor \rfloor$$

(7)

$$r_1 = \alpha n \left(1 - \frac{T}{T_{\max}}\right)$$

(8)

式(7)中, d 为对 $r_1 \sin r_2$ 四舍五入取整、再取绝对值后的值. 式(8)中, n 为工件数, α 为控制位置更新变化量大小的参数,由于参数 $d \leq n$,因此 $\alpha \in (0, 1)$.

因此,在 DSOA 中,个体的位置更新公式转变为

$$X_i(T+1) = \begin{cases} \text{Destruct Construct}(X_{\text{rand}}, d), & r_3 < 0.5 \\ \text{Destruct Construct}(X_{\text{best}}, d), & r_3 \geq 0.5 \end{cases}$$

(9)

完整的位置更新策略的伪代码为

```
Input: removing size: d
1   $r_3 = \text{a random number in } [0, 1]$ 
2  if ( $r_3 < 0.5$ )
3     $X_{\text{rand}} = \text{select one solution from } X \text{ randomly}$ 
4     $\pi = X_{\text{rand}}$ 
5  else
6     $\pi = X_{\text{best}}$ 
7  end if
8   $\pi^R = \text{remove } d \text{ job from } \pi \text{ randomly}$ 
9   $\pi^D = \pi - \pi^R$ 
10 for  $i = 1:d$ 
11   insert job  $\pi^R(i)$  into the optimal location
      which gives the minimum makespan in all
      possible positions of  $\pi^D$ 
12 end for
13  $X_i = \pi^D$ 
```

Output: solution X_i

3.3 交叉操作

为了避免个体过早地陷入局部收敛状态,需要对个体 X_i 实施交叉操作^[21]. 此交叉操作的基本思想是:利用种群中的当前最优解 X_{best} 作为参考,产生包含有最优解部分信息的新解,并用新解替换当前解. 具体地,首先随机地选择两个位置 P_1 和 P_2 ,并将 X_i 中 P_1 和 P_2 之间的工件序列作为子序列 sub_1 取出,再另外规定 sub_2 是 X_{best} 去除序列 sub_1 中工件后所得的子序列. 最后生成一个 $[0, 1]$ 之间的随机数 r ,若 $r < 0.5$,则新解 Y_i 由 $[\text{sub}_1, \text{sub}_2]$ 组成;否则, Y_i 由 $[\text{sub}_2, \text{sub}_1]$ 组成. 交叉操作如图 2 所示.

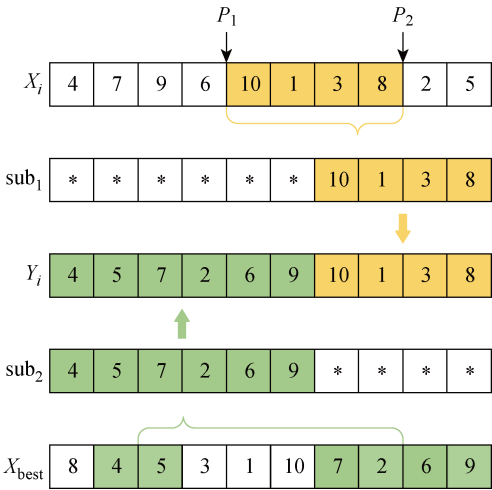


图 2 交叉操作
Fig. 2 Crossover operation

交叉操作伪代码为

Input: current solution $X_i = \{X_i^1, X_i^2, \dots, X_i^n\}$; the best solution X_{best}

```
1  generate two positions  $P_1$  and  $P_2$  randomly,
     $1 \leq P_1 \leq P_2 \leq n$ 
2   $\text{sub}_1 = \{X_i^{P_1+1}, \dots, X_i^{P_2}\}$ 
3   $\text{sub}_2 = X_{\text{best}} - \text{sub}_1$ 
4  if ( $r < 0.5$ )
5     $Y_i = \{\text{sub}_1, \text{sub}_2\}$ 
6  else
7     $Y_i = \{\text{sub}_2, \text{sub}_1\}$ 
8  end if
```

Output: solution Y_i

3.4 选择策略

为了将种群中优秀的信息传递到下一代种群中,在交叉操作产生新的 NP 个个体 Y_i 后,由原种群 X 与新种群 Y 共同组成候选解集合 S ,从 S 中选择 NP 个个体组成下一代的种群 X . 选择策略包括

两部分:① 采用精英保留策略保留最优个体,即候选解集合中适应度值最小的个体会被确定性地保留到下一代种群中;② 对剩下的 NP-1 个个体采用轮盘赌方式进行选择,对于候选解中个体 S_i , 其被选择的概率满足:

$$P(S_i) = \frac{f(S_i)}{\sum_{i=1}^{2NP} f(S_i)} \tag{10}$$

式中: $f(S_i)$ 为第 i 个个体的适应度值. 在候选解集合中,若个体适应度值较高,该个体被选择的概率会提高,反之,被选择的概率会降低.

3.5 迭代局部搜索

当找到的最优解不够理想时,需要对最优解进行局部搜索,其目的是在相对最优解附近增强密集搜索,以期望找到更好的解. 具体使用迭代局部搜索 (ILS)算法^[9]:

Input: sequence π ;
1 $\pi'=\pi$; $s=1$; counter=1
2 while (counter< n)

3 insertjob= $\pi'(s)$
4 remove insertjob from π'
5 π'' = best sequence obtained by inserting
insertjob in all possible positions of π'
6 if $C_{\max}(\pi'') < C_{\max}(\pi)$
7 $\pi=\pi''$
8 counter=1
9 else
10 counter=counter+1
11 end if
12 $s=\text{mod}(s+1,n)$
13 end while
Output: sequence π

3.6 DSOA 的框架

DSOA 的总体框架如图 3 所示,终止条件由迭代次数进行设置. 算法经过初始化、位置更新策略、交叉操作、选择策略以及 ILS 局部搜索等步骤之后,最终返回最优解.

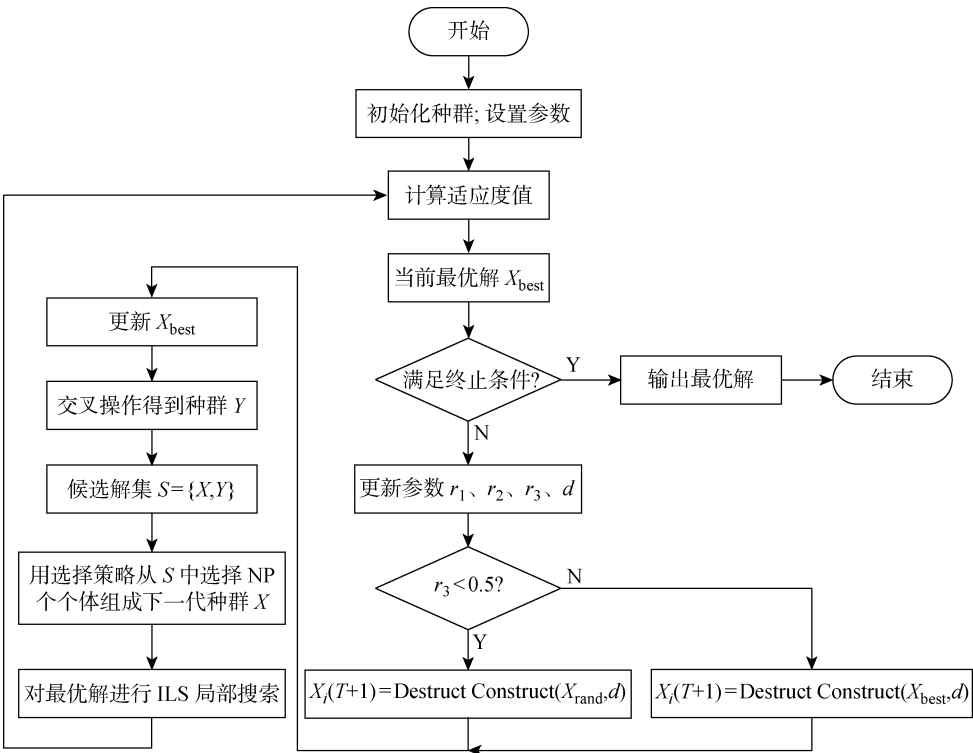


图 3 DSOA 框架

Fig. 3 Framework of DSOA

4 仿真实验

算法基于 MATLAB 2017a 实现,在处理器为 Intel Xeon E5-2640,主频为 2.40 GHz,内存为 64.0 GB 的 PC 机下运行. 为了检验 DSOA 算法求解

NIFSP 的性能,本文采用 Taillard Benchmark 问题^[22]中的 11 种不同规模的共 110 个典型算例进行测试. 为了证明所提出的算法的有效性并与其他方法进行性能比较,采用平均相对百分比偏差 (ARPD)和标准偏差(SD)来衡量算法性能:

$$ARPD = \frac{1}{N} \sum_{i=1}^N \frac{C_{\max}(i) - C^*}{C^*} \tag{11}$$

$$SD = \sqrt{\frac{1}{N} \sum_{i=1}^N [C_{\max}(i) - \bar{C}]^2} \tag{12}$$

式中： N 为独立仿真测试的次数； $C_{\max}(i)$ 为算法经第 i 次仿真测试得到的解， C^* 为迄今为止找到的最优解， \bar{C} 为 N 次仿真得到的解的平均值。

4.1 参数设置

算法的性能很大程度上取决于参数的设置，因

此通过实验来找到最佳的参数至关重要. 为确定 DSOA 中参数 α 的取值，在算例 Ta051、Ta061、Ta071 及 Ta081 上进行参数测试， α 取 0.1、0.3 及 0.5，各进行 10 次仿真，每次仿真的最大迭代次数设为 300，测试结果如表 1 所示，表中 avg 和 min 分别为算法运行结果的平均值和最优值. 此外，为了更直观地表示 3 种参数设置下 DSOA 的运行结果，用迭代收敛图进行对比，如图 4 所示。

由参数测试结果可见， $\alpha=0.1$ 时，DSOA 的运

表 1 参数 α 在 4 个不同算例上的测试结果

Tab. 1 Test results of parameter α in four different instances

算例	$\alpha=0.1$			$\alpha=0.3$			$\alpha=0.5$		
	avg	min	ARPD	avg	min	ARPD	avg	min	ARPD
Ta051	5442.1	5398	1.14	5401.6	5381*	0.38	5399.3	5386	0.34
Ta061	5834.1	5833*	0.02	5833.0	5833*	0.00	5833.0	5833*	0.00
Ta071	6751.5	6736	0.35	6742.8	6728*	0.22	6740.1	6728*	0.18
Ta081	9337.6	9327	0.21	9327.8	9318*	0.11	9324.2	9319	0.07

注：带* 的值是每个算例在所有参数取值情况下找到的最优解，加粗数据为 ARPD 的最小值。

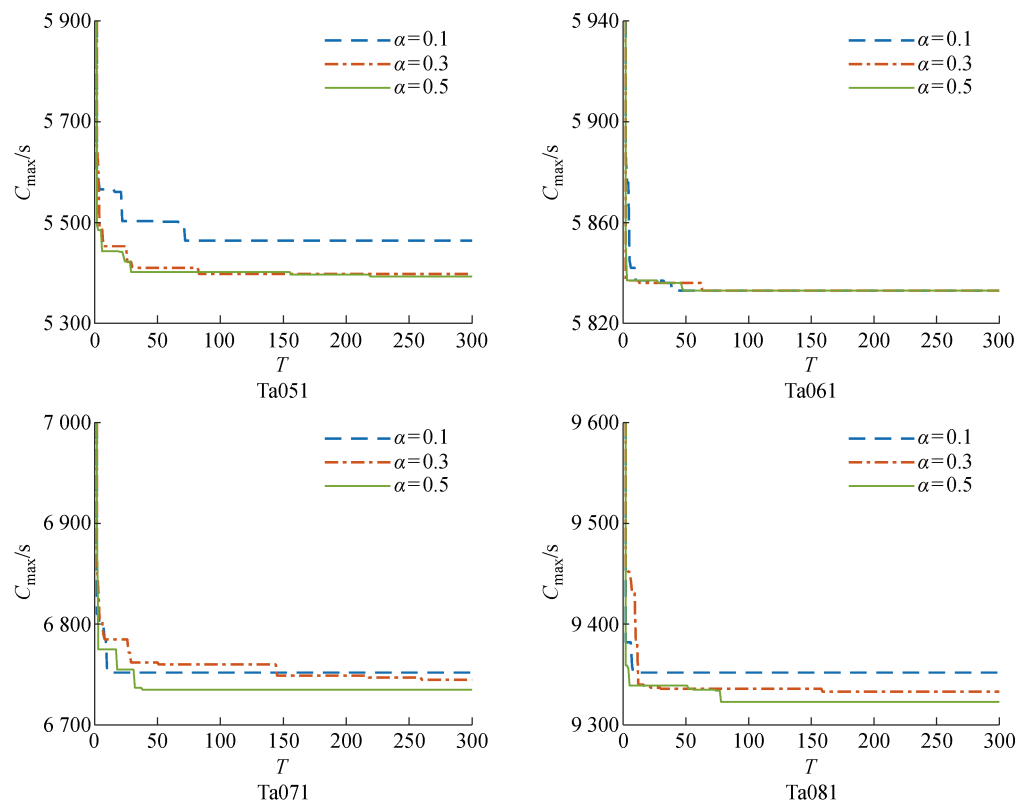


图 4 α 的测试结果
Fig. 4 Test results of α

行结果较差，特别是当问题规模的机器数较大时，算法性能的差距更为明显； $\alpha=0.3$ 时，算法虽然在 4 个算例上都可以得到最优解，但其运行结果的平均

值并不是最优的；而 $\alpha=0.5$ 时，算法在算例 Ta061、Ta071 上能获得最优解，在算例 Ta051、Ta081 上虽然没有获得最优解，但运行结果的最小值与最优解

的值相差不大,且算法运行结果的平均值最小,平均相对百分比偏差也最小,这表明 $\alpha=0.5$ 时算法的平均寻优精度最高,能使算法性能达到最优.基于以上分析,在后续实验中参数 α 设置为 0.5.确定了 α 的取值后, d 和 r_1 的取值可由式(7)、(8)得到.

DSOA 的其余参数设置如下:NP 为 30, T_{\max} 设置为 300, r_2 为 $[0,2\pi]$ 之间的随机数, r_3 为 $[0,1]$ 之间的随机数.

4.2 结果分析

为了验证 DSOA 中各个策略的有效性,采用 SOA、DSOA¹、DSOA²、DSOA³ 及 DSOA 分别对 11 个规模大小不同的算例进行求解,每个算例独立求解 10 次,最大迭代次数为 300. DSOA¹ 表示只改变了位置更新策略的离散正弦优化算法;而 DSOA²

不仅改变了位置更新策略,还加入了交叉操作和选择策略;DSOA³ 则表示改变了位置更新策略,并使用了 ILS 局部搜索,但没有加入交叉操作和选择策略的离散正弦优化算法.需要说明的是,SOA 求解 NIFSP 的编码方法采用的是基于最大值排序(LRV)规则.

表 2 列出了这 11 个算例的运行结果,选取了 10 次运行结果中的平均值进行对比,其中性能提高百分比(PIP)定义为

$$PIP=\frac{\bar{C}_{\max}(\text{SOA})-\bar{C}_{\max}(\text{DSOA}^i)}{\bar{C}_{\max}(\text{SOA})}$$

(13)

显然,当 i 的取值为 1 时,表示的是 DSOA¹ 相比于 SOA 提高的性能百分比.

为了更直观地观察各算法的求解效果,以

表 2 SOA 与 DSOAⁱ 的比较
Tab. 2 Comparison of SOA and DSOAⁱ

算例	规模		SOA	DSOA ¹		DSOA ²		DSOA ³		DSOA	
	n	m	\bar{C}_{\max}	\bar{C}_{\max}	PIP	\bar{C}_{\max}	PIP	\bar{C}_{\max}	PIP	\bar{C}_{\max}	PIP
Ta001	20	5	1 439.2	1 384.5	3.801%	1 383.5	3.870%	1 381.4	4.016%	1 380.6	4.072%
Ta011	20	10	2 331.0	2 198.7	5.676%	2 196.6	5.766%	2 198.6	5.680%	2 196.4	5.774%
Ta021	20	20	3 621.5	3 222.9	11.006%	3 215.0	11.225%	3 213.1	11.277%	3 212.2	11.302%
Ta031	50	5	3 082.6	3 014.0	2.225%	3 014.0	2.225%	3 014.0	2.225%	3 014.0	2.225%
Ta041	50	10	3 935.8	3 435.7	12.706%	3 438.1	12.645%	3 427.2	12.922%	3 423.0	13.029%
Ta051	50	20	6 383.3	5 430.2	14.931%	5 450.0	14.621%	5 411.1	15.230%	5 399.3	15.415%
Ta061	100	5	6 092.9	5 833.4	4.259%	5 833.1	4.264%	5 833.0	4.266%	5 833.0	4.266%
Ta071	100	10	7 403.3	6 756.7	8.734%	6 764.6	8.627%	6 750.0	8.824%	6 740.1	8.958%
Ta081	100	20	10 735.4	9 355.5	12.854%	9 353.2	12.875%	9 327.7	13.113%	9 324.2	13.145%
Ta091	200	10	12 826.8	11 683.4	8.914%	11 691.2	8.853%	11 651.6	9.162%	11 645.4	9.210%
Ta101	200	20	17 351.8	14 918.5	14.023%	14 982.0	13.657%	14 857.5	14.375%	14 841.8	14.465%
均值					9.012%		8.966%		9.190%		9.260%

注:加粗数据为最优的 PIP 值.

Ta051 为例,画出各个算法求解的收敛曲线,如图 5 所示.结合表 2 和图 5,不难看出,相比于 SOA, DSOA¹、DSOA²、DSOA³ 以及 DSOA 的性能都有大幅度提升.对比 SOA 和 DSOA¹ 的实验结果可以看出,重新定义的位置更新策略更适合于求解调度问题,算法的探索能力明显增强.相比于 SOA,DSOA² 和 DSOA¹ 的性能提升相差不大,但交叉操作和选择策略却并非无效的,因为从 DSOA³ 来看,DSOA³ 没有经过交叉操作和选择策略,在种群更新策略之后直接进行局部搜索,求解效果却不如 DSOA,导致这一结果的原因可能是种群多样性被破坏,算法难以跳出局部最优.而 DSOA² 和 DSOA 的实验结果表明,加入 ILS 局部搜索后能够加快收敛速度,提

高寻优精度.

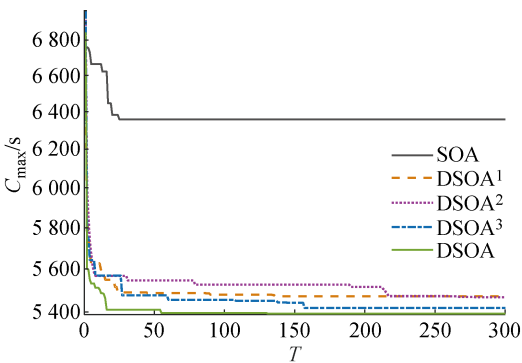


图 5 算例 Ta051 的收敛曲线
Fig. 5 Convergence curve for instance of Ta051

从表 2 中还可以看出,对每一个算例,DSOA 都能找到比 SOA 更令人满意的最优解,算例 Ta051 的 PIP 达到最大,为 15.415%,平均性能提高了 9.260%。这表明对 SOA 的改进总体来说是有效的,对于 NIFSP 的求解,DSOA 相比于 SOA,具有更好的寻优精度。

为了进一步测试所提出的 DSOA 求解 NIFSP 的有效性,将 DSOA 与 IWO^[10] 和 DFWA-LS^[12] 算法进行了比较。由于在另一台计算机上实现同一算法的结果可能不同,所以,为了确保操作环境的一致性,算法比较的公平性,重新实现了对比算法,并在相同的 PC 机上运行。对比算法的参数设置与文献[10,12]中一致。此外,为了避免测试的随机性带来的误差,每次不同的仿真都独立运行 10 次。测试结果见表 3,其中计算 ARPD 的式(11)中, C^* 采用这 3 种算法所能找到的最优解进行计算。

表 3 3 种算法的结果对比
Tab.3 Comparison of three algorithms

规模		DSOA		DFWA-LS		IWO	
n	m	ARPD	SD	ARPD	SD	ARPD	SD
20	5	0.05	0.008	0.15	0.013	0.92	0.083
	10	0.21	0.034	0.62	0.060	2.37	0.156
	20	0.21	0.057	0.51	0.093	2.33	0.272
50	5	0.00	0.000	0.01	0.004	0.49	0.103
	10	0.29	0.048	0.52	0.085	2.76	0.311
	20	0.37	0.113	0.64	0.174	4.04	0.497
100	5	0.00	0.002	0.03	0.011	0.61	0.141
	10	0.05	0.016	0.13	0.056	1.39	0.361
	20	0.22	0.091	0.35	0.154	3.66	0.651
200	10	0.05	0.041	0.13	0.098	1.85	0.397
	20	0.20	0.146	0.27	0.244	3.93	0.891
均值		0.15	0.050	0.31	0.090	2.21	0.351

注:粗体为由 IWO、DFWA-LS 和 DSOA 计算得出的最佳 ARPD 和 SD。

表 3 显示了 3 种算法求解不同规模 Taillard 算例的性能。如表 3 所示,所提出的 DSOA 的平均 ARPD 值为 0.15,小于 DFWA-LS 和 IWO 的对应值 0.31、2.21;DSOA 的平均 SD 值为 0.050,也小于由 DFWA-LS 和 IWO 得到的 0.090、0.351。以上结果说明,在最小化最大完工时间的目标上,DSOA 求解不同规模问题的寻优精度和稳定性均优于两种对比算法。此外,观察可发现 DSOA 在 $n=20, m=5$ 、 $n=50, m=5$ 以及 $n=100, m=5$ 这 3 个问题上的 SD 几近于 0,随着问题规模的增大,其 SD 值相对于 IWO 和 DFWA-LS 的更小,这表明提出的

DSOA 在解决相对大规模的问题时仍然具有出色的稳定性。这些性能受益于 DSOA 的搜索策略以及 IG 算法的结合。因此,DSOA 可较好地应用于 NIF-SP,能在保证算法稳定性的情况下找到比 IWO 和 DFWA-LS 更优的解。

图 6 显示了 3 种算法在求解算例 Ta058 时的收敛性。可以看出,与其他算法相比,DSOA 具有相对较快的收敛速度和更高的精度,并且能够快速跳出局部最优。

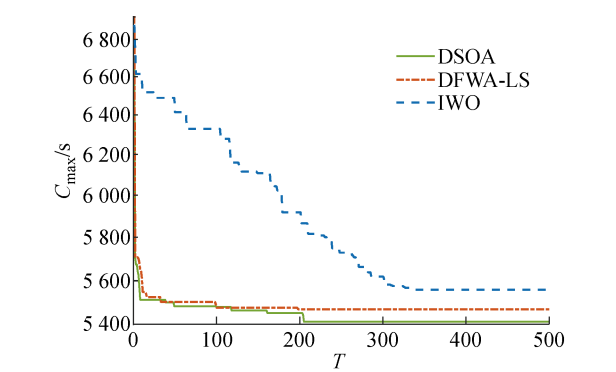


图 6 算例 Ta058 的收敛曲线
Fig.6 Convergence curve for instance of Ta058

5 结语

本文提出了一种新颖的 DSOA 来求解以最小化最大完工时间为目标的零空闲置换流水车间调度问题。所提出算法的主要思想是将 IG 算法嵌入到 SOA 的位置更新策略中。在 DSOA 中,随机选择的个体和不断变化的去除工件数提高了算法的全局搜索能力,避免了算法陷入局部最优状态。同时,IG 算法增强了算法的局部搜索能力。此后,交叉操作可以增加算法多样性,帮助算法跳出局部最优,选择策略能够选择相对优解构成新的种群。最后,ILS 局部搜索的重点在于围绕最优解进行搜索,从而提高了算法的收敛速度。这 4 个基本策略的集成为算法有效性提供了保证。

在 Taillard Benchmark 算例上测试了提出的 DSOA,并通过与 IWO 和 DFWA-LS 的对比,验证了 DSOA 在求解 NIFSP 上的有效性和优越性。下一步的研究将包括:① ILS 是局部搜索的一种基本策略,考虑用更高效的局部搜索,算法的性能将更为优越;② 考虑用 DSOA 求解其他复杂的调度问题。

参考文献:

[1] LIN J, ZHANG S. An effective hybrid biogeography-based optimization algorithm for the distributed as-

- sembly permutation flow-shop scheduling problem [J]. **Computers & Industrial Engineering**, 2016, 97: 128-136.
- [2] JOHNSON S M. Optimal two- and three-stage production schedules with setup times included[J]. **Naval Research Logistics Quarterly**, 1954, 1(1): 61-68.
- [3] GONCHAROV Y, SEVASTYANOV S. The flow shop problem with no-idle constraints: A review and approximation[J]. **European Journal of Operational Research**, 2009, 196(2): 450-456.
- [4] MAKUCHOWSKI M. Permutation, no-wait, no-idle flow shop problems[J]. **Archives of Control Sciences**, 2015, 25(2): 189-199.
- [5] BAPTISTE P, HGUNY L K. A branch and bound algorithm for the $F/\text{no-idle}/C_{\max}$ [C]// **Proceedings of the International Conference on Industrial Engineering and Production Management**. Lyon, France: IEEE, 1997: 429-438.
- [6] PAN Q K, WANG L. 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.
- [7] DENG G L, GU X S. A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion[J]. **Computers & Operations Research**, 2012, 39(9): 2152-2160.
- [8] FATIH TASGETIREN M, PAN Q K, SUGANTHAN P N, *et al.* A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem[J]. **Computers & Operations Research**, 2013, 40(7): 1729-1743.
- [9] TASGETIREN M F, PAN Q K, SUGANTHAN P N, *et al.* A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion[J]. **Applied Mathematical Modelling**, 2013, 37(10/11): 6758-6779.
- [10] ZHOU Y Q, CHEN H, ZHOU G. Invasive weed optimization algorithm for optimization no-idle flow shop scheduling problem[J]. **Neurocomputing**, 2014, 137: 285-292.
- [11] SHEN J N, WANG L, WANG S Y. A bi-population EDA for solving the no-idle permutation flow-shop scheduling problem with the total tardiness criterion [J]. **Knowledge-Based Systems**, 2015, 74: 167-175.
- [12] 刘翱, 冯骁毅, 邓旭东, 等. 求解零空闲置换流水车间调度问题的离散烟花算法[J]. **系统工程理论与实践**, 2018, 38(11): 2874-2884.
- LIU Ao, FENG Xiaoyi, DENG Xudong, *et al.* A discrete fireworks algorithm for solving no-idle permutation flow shop problem[J]. **Systems Engineering-Theory & Practice**, 2018, 38(11): 2874-2884.
- [13] SHAO W S, PI D C, SHAO Z S. Memetic algorithm with node and edge histogram for no-idle flow shop scheduling problem to minimize the makespan criterion[J]. **Applied Soft Computing**, 2017, 54: 164-182.
- [14] SHAO W S, PI D C, SHAO Z S. A hybrid discrete teaching-learning based meta-heuristic for solving no-idle flow shop scheduling problem with total tardiness criterion [J]. **Computers & Operations Research**, 2018, 94: 89-105.
- [15] MIRJALILI S. SCA: A Sine Cosine Algorithm for solving optimization problems[J]. **Knowledge-Based Systems**, 2016, 96: 120-133.
- [16] MESHKAT M, PARHIZGAR M. Sine Optimization Algorithm (SOA): A novel optimization algorithm by change update position strategy of search agent in Sine Cosine Algorithm[C]// **2017 3rd Iranian Conference on Intelligent Systems and Signal Processing (ICS-PIS)**. Piscataway, NJ, USA: IEEE, 2017: 11-16.
- [17] LI S, FANG H J, LIU X Y. Parameter optimization of support vector regression based on sine cosine algorithm[J]. **Expert Systems With Applications**, 2018, 91: 63-77.
- [18] WANG J Z, YANG W D, DU P, *et al.* A novel hybrid forecasting system of wind speed based on a newly developed multi-objective sine cosine algorithm [J]. **Energy Conversion and Management**, 2018, 163: 134-150.
- [19] ATTIA A F, EL SEHIEMY R A, HASANIEN H M. Optimal power flow solution in power systems using a novel Sine-Cosine algorithm[J]. **International Journal of Electrical Power & Energy Systems**, 2018, 99: 331-343.
- [20] LIU W B, JIN Y, PRICE M. A new Nawaz-Enscore-Ham-based heuristic for permutation flow-shop problems with bicriteria of makespan and machine idle time[J]. **Engineering Optimization**, 2016, 48(10): 1808-1822.
- [21] ZHAO F Q, LIU H, ZHANG Y, *et al.* A discrete Water Wave Optimization algorithm for no-wait flow shop scheduling problem [J]. **Expert Systems With Applications**, 2018, 91: 347-363.
- [22] TAILLARD E. Benchmarks for basic scheduling problems[J]. **European Journal of Operational Research**, 1993, 64(2): 278-285.