

# 融合多策略的黄金正弦黑猩猩优化算法

刘成汉<sup>1,2</sup> 何庆<sup>1,2</sup>

**摘要** 针对黑猩猩优化算法 (Chimp optimization algorithm, ChOA) 存在收敛速度慢、精度低和易陷入局部最优值问题, 提出一种融合多策略的黄金正弦黑猩猩优化算法 (Multi-strategy golden sine chimp optimization algorithm, IChOA). 引入 Halton 序列初始化种群, 提高初始化种群的多样性, 加快算法收敛, 提高收敛精度; 考虑到收敛因子和权重因子对于平衡算法勘探和开发能力的重要作用, 引入改进的非线性收敛因子和自适应权重因子, 平衡算法的搜索能力; 结合黄金正弦算法相关思想, 更新个体位置, 提高算法对局部极值的处理能力. 通过对 23 个基准测试函数的寻优对比分析和 Wilcoxon 秩和统计检验以及部分 CEC2014 测试函数寻优结果对比可知, 改进的算法具有更好的鲁棒性; 最后, 通过 2 个实际工程优化问题的实验对比分析, 进一步验证了 IChOA 在处理现实优化问题上的优越性.

**关键词** 黑猩猩优化算法, Halton 序列, 非线性收敛因子, 自适应权重因子, 黄金正弦算法

**引用格式** 刘成汉, 何庆. 融合多策略的黄金正弦黑猩猩优化算法. 自动化学报, 2023, 49(11): 2360–2373

**DOI** 10.16383/j.aas.c210313

## Golden Sine Chimp Optimization Algorithm Integrating Multiple Strategies

LIU Cheng-Han<sup>1,2</sup> HE Qing<sup>1,2</sup>

**Abstract** Chimp optimization algorithm (ChOA) has the problems of slow convergence speed, low accuracy and easy to fall into local optimal value. A multi-strategy golden sine chimp optimization algorithm (IChOA) was proposed. The Halton sequence is introduced into the algorithm to initialize the population, which improves the diversity of the initialized population, accelerates the convergence of the algorithm, and improves the convergence accuracy. Considering the important role of convergence factor and weight factor on the exploration and exploitation of the balance algorithm, the improved nonlinear convergence factor and adaptive weighting factor are introduced to balance the search ability of the algorithm. The golden sine algorithm is used to update the individual position and improve the ability of the algorithm to deal with the local extreme value. Through comparative analysis of the optimization of 23 benchmark test functions, Wilcoxon rank sum statistical test and optimization results of some CEC2014 test functions, it can be seen that the improved algorithm has better robustness. Finally, through comparative analysis of two practical engineering optimization problems, the superiority of IChOA in dealing with realistic optimization problem is further verified.

**Key words** Chimp optimization algorithm (ChOA), Halton sequence, nonlinear convergence, adaptive weighting factor, golden sine algorithm

**Citation** Liu Cheng-Han, He Qing. Golden sine chimp optimization algorithm integrating multiple strategies. *Acta Automatica Sinica*, 2023, 49(11): 2360–2373

在过去的 20 年中, 元启发式算法由于其简单性、灵活性和较高的鲁棒性被广泛关注. 常见的元

启发式算法有遗传算法 (Genetic algorithm, GA)<sup>[1]</sup>、粒子群优化 (Particle swarm optimization, PSO) 算法<sup>[2]</sup>、灰狼优化 (Grey wolf optimization, GWO) 算法<sup>[3]</sup> 和蝴蝶优化算法<sup>[4]</sup> 等. 随着技术不断发展和更新, 元启发式算法在复杂优化问题上表现卓越, 如路径规划<sup>[5]</sup>、电力系统控制<sup>[6]</sup> 等工程设计问题.

2020 年, Khishe 等<sup>[7]</sup> 提出黑猩猩优化算法 (Chimp optimization algorithm, ChOA), 该算法是一种基于黑猩猩种群社会行为的启发式智能优化算法, ChOA 与其他智能优化算法相比, 具有参数少、易实现、稳定性高等优点. 尽管不同的智能优化算法存在不同的搜索方式, 但其目标大多都是探索种群多样性和搜索能力之间的平衡, 在保证收敛精度

收稿日期 2021-04-12 录用日期 2021-09-17

Manuscript received April 12, 2021; accepted September 17, 2021

国家自然科学基金 (62166006), 贵州省科技计划项目重大专项项目 ([2021] 335), 公共大数据国家重点实验室开放课题 (2017BDKFJJ004) 资助

Supported by National Natural Science Foundation of China (62166006), Major Special Project of Guizhou Science and Technology Planning Project ([2021] 335), and Open Project of State Key Laboratory of Public Big Data (2017BDKFJJ004)

本文责任编辑 张俊

Recommended by Associate Editor ZHANG Jun

1. 贵州大学大数据与信息工程学院 贵阳 550025 2. 贵州大学公共大数据国家重点实验室 贵阳 550025

1. College of Big Data and Information Engineering, Guizhou University, Guiyang 550025 2. State Key Laboratory of Public Big Data, Guizhou University, Guiyang 550025

和速度的同时, 避免早熟现象的发生<sup>[8]</sup>. 针对上述思想, 众多学者对其研究的智能优化算法做了改进, 例如王坚浩等<sup>[9]</sup>提出一种基于混沌搜索策略的鲸鱼优化算法, 采用混沌反向学习策略初始化种群, 保证了初始化种群的质量; 宁杰琼等<sup>[10]</sup>利用混沌映射初始化蝴蝶个体的位置, 以此增加算法初期种群的多样性; 王秋萍等<sup>[11]</sup>引入基于余弦规律变化的收敛因子和基于欧氏距离的比例权重因子, 平衡算法的探索和开发能力; Ewees 等<sup>[12]</sup>通过将蝗虫优化算法与对立学习策略相结合, 同时引入深度探索搜索策略, 从而加快了算法的收敛速度; Dinkar 等<sup>[13]</sup>提出一种基于对立的拉普拉斯蚁狮搜索算法, 采用对立的学习模型确保在搜索空间的同时, 探索原始和相反的候选解, 从而在进化过程中评估更好的候选解; 吕鑫等<sup>[14]</sup>提出一种混沌麻雀搜索算法, 通过产生混沌序列扰动陷入局部最优的解, 促使其跳出局部最优值; Sayed 等<sup>[15]</sup>提出一种二进制克隆花授粉算法, 通过将克隆选择算法和花授粉算法相结合, 构成二元克隆花授粉算法, 以解决原始花粉克隆算法易陷入局部最优解的问题.

综上所述, 智能优化算法和改进机制层出不穷, 各具优势, 适用于解决一些优化问题, 但在另外一些问题的求解上, 会显露出不足. 因此, 为了提高 ChOA 的寻优性能和适用性, 探索更适合解决实际优化问题的方法, 本文提出一种融合多策略的黄金正弦黑猩猩优化算法 (Multi-strategy golden sine chimp optimization algorithm, IChOA). 该算法采用 Halton 序列初始化种群, 生成更加均匀个体, 保证算法初期的多样性, 提高算法收敛速度和精度; 同时, 引入非线性收敛因子和自适应权重因子, 平衡算法的搜索能力, 加快算法收敛速度; 最后, 根据黄金正弦算法的相关思想更新个体位置, 避免算法过早收敛. 仿真实验采用 23 个标准测试函数以及部分 CEC2014 测试函数进行寻优测试, 并采用 Wilcoxon 秩和检验进行统计分析, 验证了算法的优越性. 通过 2 个工程设计优化问题的实验对比结果表明, IChOA 相较于其他算法, 更具优势.

## 1 基本黑猩猩搜索算法

ChOA 算法是根据黑猩猩种群的捕猎行为提出的一种群智能优化算法, 一般黑猩猩的狩猎过程分为探索和开发两个阶段. 探索阶段即驱动、阻挡和追逐猎物, 开发阶段即攻击猎物. 黑猩猩捕猎行为的仿生学原理描述如下.

为了用数学模型模拟黑猩猩的行为, 假设第 1 个攻击者 (最佳解决方案)、驱赶者、阻碍者和追赶者能发现潜在猎物的位置, 其他黑猩猩被迫根据最

佳黑猩猩的位置更新自己的位置. 黑猩猩驱赶和追逐猎物的数学模型为:

$$d(t) = |cX_P(t) - mX_C(t)| \quad (1)$$

$$X_C(t+1) = X_P(t) - a \cdot d \quad (2)$$

式中,  $d(t)$  为猎物与黑猩猩间的距离,  $t$  为当前迭代次数,  $a$  和  $c$  为系数向量,  $m$  是由混沌映射产生的混沌向量,  $X_P$  为猎物的位置向量,  $X_C$  为黑猩猩的位置向量.  $a$  和  $c$  向量表示如下:

$$a = 2 \cdot f \cdot r_1 - f \quad (3)$$

$$c = 2 \cdot r_2 \quad (4)$$

式中,  $f$  为线性衰减因子, 随着迭代次数的增加,  $f$  值由 2.5 线性衰减到 0,  $r_1$  和  $r_2$  是取值为  $[0, 1]$  的随机数.

参数  $a$  为  $[-2f, 2f]$  的随机变量, 假设  $a$  的值为  $[-1, 1]$  时, 猎物停止移动, 此时黑猩猩必须攻击猎物结束捕猎. 因此, 采取降低  $f$  值的方式迫使黑猩猩结束捕猎, 黑猩猩的下一个位置可以在当前位置与猎物位置之间的任意位置.

黑猩猩攻击猎物数学模型为:

$$\begin{cases} d_A = |c_1 X_A - m_1 X| \\ d_B = |c_2 X_B - m_2 X| \\ d_C = |c_3 X_C - m_3 X| \\ d_D = |c_4 X_D - m_4 X| \end{cases} \quad (5)$$

$$\begin{cases} X_1 = X_A - a_1 \cdot d_A \\ X_2 = X_B - a_2 \cdot d_B \\ X_3 = X_C - a_3 \cdot d_C \\ X_4 = X_D - a_4 \cdot d_D \end{cases} \quad (6)$$

$$X(t+1) = \frac{X_1 + X_2 + X_3 + X_4}{4} \quad (7)$$

由式 (5) ~ (7) 可知, 黑猩猩个体最终的位置是随机分布在一个由攻击者、阻碍者、追逐者和驱赶者黑猩猩位置所确定的圆圈中, 即猎物位置是由 4 个最好的个体来估计的, 而其他黑猩猩则随机更新它们在附近的位置.

种群在捕猎的最后阶段个体获得食物满足, 随后的社会动机会使黑猩猩释放它们的天性, 此时, 黑猩猩个体试图强行混乱地获得食物. 黑猩猩在最后阶段的混乱行为, 有助于进一步缓解解决高维问题时的局部最优陷阱和收敛速度慢这两个问题.

原始黑猩猩算法使用了 6 种具有随机行为的确定性混沌过程映射, 为了模拟这种社会行为, 假设有 50% 的概率在正常的位置更新机制或混沌模型中选择其一更新黑猩猩的位置, 社会性刺激行为为

学模型为:

$$X_C(t+1) = \begin{cases} X_P(t) - a \cdot d, & \mu < 0.5 \\ \text{Chaotic}, & \mu \geq 0.5 \end{cases} \quad (8)$$

式中,  $\mu$  是取值  $[0, 1]$  的随机数, *Chaotic* 是混沌映射, 用来更新解的位置.

## 2 改进的黑猩猩搜索算法

在基本 ChOA 中, 种群初始化处理采用随机分布方式, 这种随机方式导致算法多样性降低, 个体寻优存在一定的盲目性; 其次, 算法用来平衡局部搜索和全局搜索能力的收敛因子  $f$  采用线性规律下降因子, 并不符合算法迭代过程中的非线性寻优规律; 最后, 对于加强算法跳出局部极值能力的混沌扰动 *Chaotic*, 其只有 50% 概率随机触发, 具有很大的不稳定性, 虽然有一定的概率带领个体跳出局部最优值点, 但是并没有考虑寻优过程中个体位置信息的变化.

综上所述, 本文针对上述算法原理上的缺陷, 引入相应策略进行改善, 具体策略如下.

### 2.1 Halton 序列初始化种群

原始黑猩猩优化算法采用 rand 函数随机初始化种群, 所得到的种群随机性高, 但是不一定均匀地分布在解空间, 导致种群搜索速度慢, 算法多样性不足. 针对上述问题, 本文引入 Halton 序列产生伪随机数来初始化种群, 伪随机数的遍历性使个体更加均匀地分布在解空间<sup>[16]</sup>, 提高初始化时算法的多样性, 个体能快速发现优质解的位置, 从而加快算法收敛, 提高算法收敛精度.

对于二维 Halton 序列, 其实现过程为: 选取两个质数作为基础量, 通过对两个基础量不断切分, 从而组合成一系列均匀分布且不重复的点, 其切分过程数学模型为:

$$n = \sum_{i=0}^m b_i \cdot p^i = b_m \cdot p^m + \dots + b_1 \cdot p^1 + b_0 \quad (9)$$

$$\theta(n) = b_0 p^{-1} + b_1 p^{-2} + \dots + b_m p^{-m-1} \quad (10)$$

$$H(n) = [\theta_1(n), \theta_2(n)] \quad (11)$$

式中,  $n \in [1, N]$  为任意整数,  $p$  是大于等于 2 的质数,  $b_i \in \{0, 1, \dots, p-1\}$  为常数,  $p$  表示 Halton 序列基础量,  $\theta(n)$  是定义的序列函数,  $H(n)$  为最后得到的二维均匀 Halton 序列.

图 1 为基本黑猩猩算法产生的种群随机初始化个体分布图. 图 2 为使用 Halton 序列产生的初始种群分布图, 其中 Halton 序列基础量为  $base1 = 2$ ,  $base2 = 3$ .

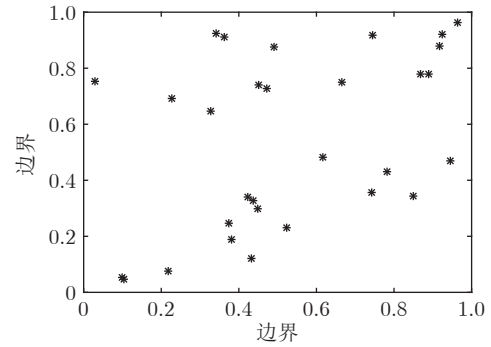


图 1 种群随机初始化个体分布图

Fig.1 Randomly initialized population distribution map

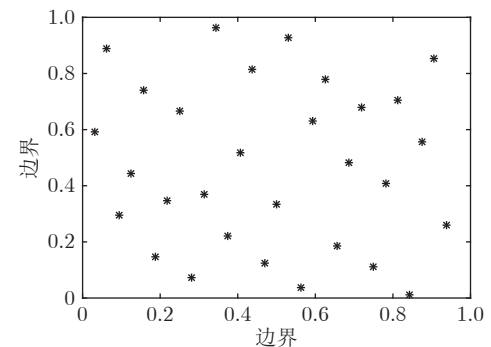


图 2 使用 Halton 序列产生的初始种群分布图

Fig.2 Halton sequence initialized population distribution map

由图 1 和图 2 的对比分析可知, Halton 序列产生的种群分布虽然随机性并不高, 但是分布更为均匀, 并没有出现个体重叠现象. 由此可知, Halton 序列产生的种群质量更高, 算法多样性更好.

### 2.2 非线性收敛因子和自适应权重因子

#### 2.2.1 非线性收敛因子

评价启发式算法性能好坏的重要环节之一是否能平衡算法的全局搜索能力和局部开发能力. 由前面的分析可知, 随机变量  $a = [-2f, 2f]$ , 当  $|a| \leq 1$  时, 个体攻击猎物收敛; 当  $|a| > 1$  时, 个体分散搜寻猎物, 因此收敛因子  $f$  的变化决定了算法局部和全局搜索能力的平衡. 基本 ChOA 算法的收敛因子  $f$  由 2.5 线性下降到 0, 这种线性变化并不能适应算法对于复杂多峰函数的寻优, 导致算法寻优速度缓慢, 甚至陷入局部最优值. 因此, 本文引入一种非线性变化的收敛因子, 在算法迭代前期种群大范围搜索时, 缓慢衰减的收敛因子能让种群更好地搜索全局最优解; 在算法迭代后期种群收敛, 此时快速衰减的收敛因子有利于算法局部寻找最优解. 同时, 加入控制因子  $k$ , 能控制衰减的幅度, 非线性收敛因

子的数学模型描述如下:

$$f = f_m \cdot \left[ 1 - \left( \frac{e^{\frac{t}{Max_{iter}} - 1}}{e - 1} \right)^k \right] \quad (12)$$

式中,  $t$  为当前迭代次数,  $Max_{iter}$  为最大迭代次数,  $f_m$  为收敛因子的初始值.  $k \in [1, 10]$  为控制因子,  $k$  能控制  $f$  衰减幅度,  $k$  越大时, 收敛因子衰减越慢; 反之, 收敛因子衰减越快. 线性收敛因子  $f$  和控制因子  $k$ , 取  $k = 1, 5, 10$  的非线性收敛因子, 对比如图 3 所示.

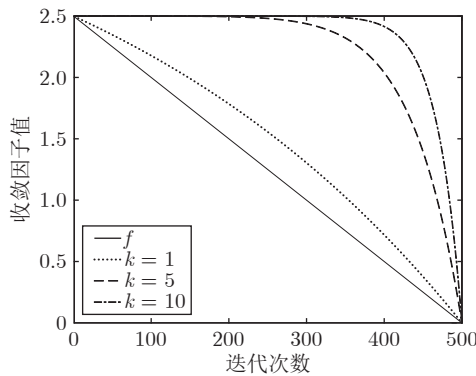


图 3 收敛因子对比图

Fig.3 Contrast diagram of convergence factors

### 2.2.2 自适应权重因子

权重因子在目标函数优化中起着重要的作用, 合适的权重能够加快算法收敛, 提高算法收敛精度. 针对 ChOA 算法寻优过程中收敛速度慢、精度不高问题, 引入一种自适应调整的权重因子  $\omega$ : 在算法迭代初期, 给予一个较大的权重使种群以大步长遍历整个搜索空间, 有利于算法快速搜寻全局最优位置, 加快算法收敛; 在算法迭代中/后期, 算法逐渐收敛, 个体局部搜寻最优解, 此时给予一个较小的权重, 有利于算法以小步长精细探索最优位置, 提高算法收敛精度; 最后, 在算法迭代末期, 针对 ChOA 算法易陷入局部最优问题, 给予个体位置一个相对较大的扰动, 有利于算法跳出局部最优解. 自适应权重因子  $\omega$  数学模型如下:

$$\omega = \begin{cases} \delta_1 \cdot \left( \cos \left( t \cdot \frac{\pi}{\delta_2} \right) + \delta_3 \right), & t \leq \varepsilon \\ \rho_1 \cdot \sin(\rho_2 \cdot t \cdot \pi) + \rho_3, & t > \varepsilon \end{cases} \quad (13)$$

式中,  $\delta_1$ 、 $\delta_2$ 、 $\delta_3$ 、 $\rho_1$ 、 $\rho_2$ 、 $\rho_3$  为常数系数,  $t$  为当前迭代次数,  $t = (1, 2, \dots, Max_{iter})$ ,  $\varepsilon$  为指定迭代次数. 自适应权重因子  $\omega$  曲线见图 4.

### 2.3 黄金正弦策略更新位置

2017 年, Tanyildizi 等<sup>[17]</sup> 根据正弦函数相关思

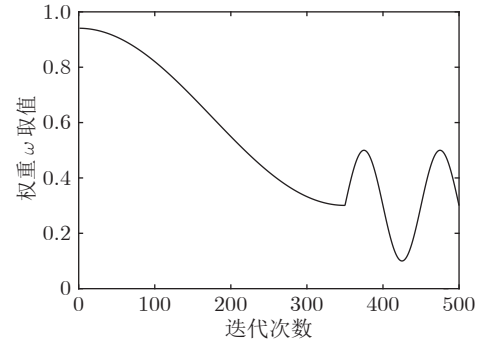


图 4 自适应权重因子  $\omega$  曲线

Fig.4 Adaptive weighting factor  $\omega$  curve

想, 提出黄金正弦算法, 该新型智能算法具有寻优速度快, 调参简单、鲁棒性好等优点. 黄金正弦算法使用正弦函数与单位圆的特殊关系结合黄金分割系数进行迭代搜索, 通过正弦函数扫描单位圆模拟算法探索搜索空间的过程.

公元前 4 世纪, 古希腊数学家 Eudoxus 首次提出黄金分割系数概念, 黄金分割系数不需要梯度信息, 每个步骤仅需要迭代一次, 同时, 黄金分割系数的收缩步骤是固定的. 因此, 将正弦函数与黄金分割率组合, 可以更快地找到函数的最大值或最小值. 同时, 黄金正弦搜索策略的遍历性能有效防止算法陷入局部最优值. 黄金正弦算法数学模型如下:

$$X_i^{t+1} = X_i^t \cdot |\sin(R_1)| + R_2 \cdot \sin(R_1) \cdot |x_1 \cdot P_i^t - x_2 \cdot x_i^t| \quad (14)$$

式中,  $t$  为当前迭代次数;  $R_1$  和  $R_2$  分别为取值  $[0, 2\pi]$  和  $[0, \pi]$  的随机数, 表示下一代个体移动的距离和方向;  $x_1$  和  $x_2$  为黄金分割系数, 用来缩小搜索空间, 引导个体向最优值收敛;  $P_i^t$  是当前最优个体的位置. 黄金分割系数  $x_1$  和  $x_2$  数学模型如下:

$$x_1 = \alpha \cdot (1 - \tau) + \beta \cdot \tau \quad (15)$$

$$x_2 = \alpha \cdot \tau + \beta \cdot (1 - \tau) \quad (16)$$

$$\tau = \frac{\sqrt{5} - 1}{2} \quad (17)$$

式中,  $\alpha$  和  $\beta$  是搜索间隔;  $\tau$  是黄金分割比, 取值约为 0.618033.

### 2.4 IChOA 实现流程和伪代码

IChOA 算法的具体实现步骤如下:

**步骤 1.** 利用 Halton 序列初始化种群, 包括种群个体数  $N$ , 最大迭代次数  $Max_{iter}$ , 维度  $d$ , 种群搜索边界  $lb$ 、 $ub$  等, 并设置相关参数.

**步骤 2.** 计算当前种群的搜索空间, 返回超出搜索空间的个体.

**步骤 3.** 计算种群中每个个体的适应度  $fit$  并排序, 记录当前最好个体  $P_{best}$  和最差个体  $P_{worst}$ .

**步骤 4.** 更新攻击者、追赶者、追逐者和阻碍者位置.

**步骤 5.** 更新非线性扰动因子  $f$  和自适应权重因子  $\omega$ , 以及参数  $A$  和  $C$ .

**步骤 6.** 随机选取部分个体根据式 (12) 更新位置, 其他个体按照式 (5) ~ (7) 更新位置.

**步骤 7.** 根据式 (8) 引入混沌算子.

**步骤 8.** 重复步骤 3 ~ 7, 直到达到最大迭代次数或者算法收敛.

### 算法 1. IChOA 算法伪代码

**初始化.** Halton 序列种群并设置相关参数.

- 1) 计算当前种群的搜索空间, 返回超出搜索空间的个体;
- 2) While  $l < Max_{iter}$ ;
- 3) 计算个体适应度并排序;
- 4) 记录最优和最差个体位置及适应度;
- 5) 更新 4 个领导者位置;
- 6) for  $i = 1: N$ ;
- 7) if  $i < 10$ ;
- 8) 按照式 (12) 更新个体位置;
- 9) else;
- 10) 按照式 (5) ~ (7) 更新个体位置;
- 11)  $m = chaos(3, 1, 1)$ ;
- 12) end;
- 13) end;
- 14) end;
- 15) 算法结束, 返回最优位置及适应度.

## 2.5 IChOA 算法时间复杂度分析

基本 ChOA 算法的时间复杂度为  $O(N \times d \times Max_{iter})$ , 其中  $N$  为种群规模,  $d$  为维度,  $Max_{iter}$  为最大迭代次数. IChOA 算法时间复杂度分析如下:

1) 引入 Halton 序列初始化种群的时间复杂度为  $O(N \times d)$ , 则引入 Halton 序列初始化种群的 ChOA (HChOA) 的时间复杂度为  $O(N \times d \times Max_{iter} + N \times d) = O(N \times d \times Max_{iter})$ ;

2) 假设引入自适应权重和非线性收敛因子所需的时间分别为  $t_1$  和  $t_2$ , 因此引入自适应权重和非线性收敛因子的 ChOA 算法 (WChOA) 的时间复杂度为  $O(N \times d \times Max_{iter} + t_1 + t_2) = O(N \times d \times Max_{iter})$ ;

3) 引入黄金正弦算法的时间复杂度为  $O(N \times d)$ , 因此引入黄金正弦算法更新位置的 ChOA 算法 (GChOA) 的时间复杂度为  $O(N \times d \times Max_{iter} + N \times d) = O(N \times d \times Max_{iter})$ .

综上所述, IChOA 算法的时间复杂度为  $O(N \times d \times Max_{iter})$ . 由此可知, 本文提出的 IChOA 算法时间复杂度与 ChOA 算法时间复杂度一致.

## 3 实验仿真与结果分析

### 3.1 参数设置

仿真实验使用的计算机配置为 Intel Core i5-7500U, 主频为 3.40 GHz, 内存 32 GB, 操作系统 64 bit, 计算环境为 Matlab2016(a). 本文选取基本黑猩猩算法 (ChOA)、粒子群优化算法 (PSO)<sup>[18]</sup>、灰狼优化算法 (GWO)<sup>[19]</sup> 和鲸鱼优化算法 (Whale optimization algorithm, WOA) 与 IChOA 算法进行对比, 基本参数统一设置为: 种群规模  $N = 30$ , 最大迭代次数  $Max_{iter} = 500$ , 维度分为低维  $d = 30$ 、高维  $d = 500$ . 各算法参数设置见表 1.

表 1 算法参数设置  
Table 1 Parameter setting of algorithm

算法	参数
ChOA	$m = chaos(3, 1, 1)$
PSO	$c_1 = 1.5, c_2 = 2.0, \omega = 1, w_{damp} = 0.99$
GWO	—
WOA	$b = 1$
IChOA	$m = chaos(3, 1, 1), \delta_1 = 0.3, \delta_2 = 300.0, \delta_3 = 1.8, \rho_1 = 0.10, \rho_2 = 0.05, \rho_3 = 0.30, \epsilon = 300$

### 3.2 测试函数介绍

为了测试 IChOA 算法的寻优性能, 采用文献 [20] 使用的 23 个基准测试函数进行函数寻优测试, 其中,  $f_1 \sim f_7$  为单峰函数,  $f_8 \sim f_{13}$  为复杂多峰函数,  $f_{14} \sim f_{23}$  为固定维度多峰函数, 基准测试函数相关信息如表 2 所示.

### 3.3 各改进策略寻优性能对比

#### 3.3.1 Halton 序列初始化种群性能分析

为了提高种群初始化的均匀性, 提高种群多样性, 本文引入 Halton 序列相关思想初始化种群. 取 Halton 序列基础量为  $base1 = 2$ 、 $base2 = 3$ , 种群规模  $N = 30$ , 最大迭代次数  $Max_{iter} = 500$ , 基本 ChOA 与加入 Halton 序列的 ChOA (HChOA) 在单峰测试函数  $f_1$ 、 $f_3$  和多峰测试函数  $f_9$ 、 $f_{12}$  上的寻优对比如图 5 所示.

由图 5 可知, 对于单峰测试函数  $f_1$  和多峰测试函数  $f_9$ , HChOA 能收敛到更接近于理论最优值, 说明引入 Halton 序列的 ChOA 算法在基本函数寻优精度上有一定的优势; 对于单峰测试函数  $f_3$  和多

表 2 基准测试函数介绍  
Table 2 Introduction to benchmark functions

编号	函数名	定义域	维度	最优值	绝对精度误差 $\epsilon$
$f_1$	Sphere	$[-100, 100]$	30	0	$1.00 \times 10^{-3}$
$f_2$	Schwefel'problem 2.22	$[-10, 10]$	30	0	$1.00 \times 10^{-3}$
$f_3$	Schwefel'problem 1.2	$[-100, 100]$	30	0	$1.00 \times 10^{-3}$
$f_4$	Schwefel'problem 2.21	$[-100, 100]$	30	0	$1.00 \times 10^{-3}$
$f_5$	Generalized Rosenbrock's function	$[-30, 30]$	30	0	$1.00 \times 10^{-2}$
$f_6$	Step function	$[-100, 100]$	30	0	$1.00 \times 10^{-2}$
$f_7$	Quartic function	$[-1.28, 1.28]$	30	0	$1.00 \times 10^{-2}$
$f_8$	Generalized Schwefel's problem 2.26	$[-500, 500]$	30	-12569.5000	$1.00 \times 10^2$
$f_9$	Generalized Rastrigin's Function	$[-5.12, 5.12]$	30	0	$1.00 \times 10^{-2}$
$f_{10}$	Ackley's function	$[-32, 32]$	30	0	$1.00 \times 10^{-2}$
$f_{11}$	Generalized Ciewank function	$[-600, 600]$	30	0	$1.00 \times 10^{-2}$
$f_{12}$	Generalized penalized function 1	$[-50, 50]$	30	0	$1.00 \times 10^{-2}$
$f_{13}$	Generalized penalized function 2	$[-50, 50]$	30	0	$1.00 \times 10^{-2}$
$f_{14}$	Shekell's foxholes function	$[-65, 65]$	2	1.0000	$1.00 \times 10^{-2}$
$f_{15}$	Kowalik's function	$[-5, 5]$	4	0.0003	$1.00 \times 10^{-2}$
$f_{16}$	Six-hump camel-back function	$[-5, 5]$	2	-1.0300	$1.00 \times 10^{-2}$
$f_{17}$	Branin function	$[-5, 5]$	2	0.3980	$1.00 \times 10^{-2}$
$f_{18}$	Gold stein-price function	$[-2, 2]$	2	3.0000	$1.00 \times 10^{-2}$
$f_{19}$	Hatman's function1	$[0, 1]$	3	-3.8600	$1.00 \times 10^{-2}$
$f_{20}$	Hatman's function 2	$[0, 1]$	6	-3.3200	$1.00 \times 10^{-2}$
$f_{21}$	Shekel's family 1	$[0, 10]$	4	-10.0000	$1.00 \times 10^{-2}$
$f_{22}$	Shekel's family 2	$[0, 10]$	4	-10.0000	$1.00 \times 10^{-2}$
$f_{23}$	Shekel's family 3	$[0, 10]$	4	-10.0000	$1.00 \times 10^{-2}$

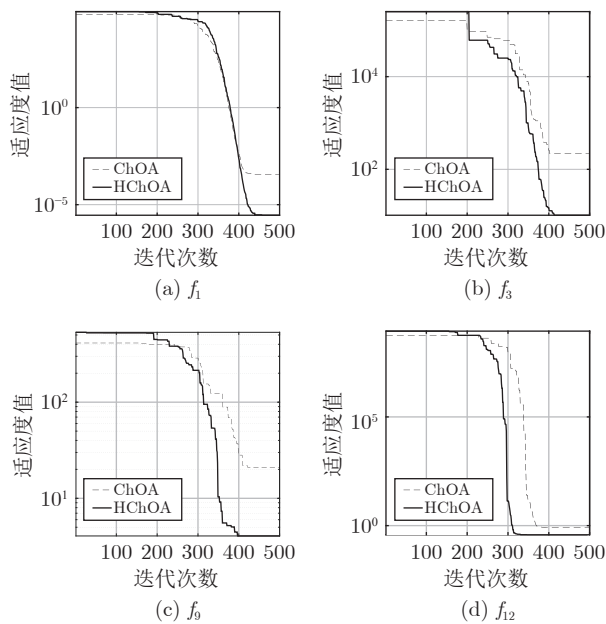


图 5 ChOA 与 HChOA 收敛对比图

Fig.5 Convergence curve of ChOA and HChOA

峰测试函数  $f_{12}$ , HChOA 收敛速度比 ChOA 更快, 说明 Halton 序列初始化种群提高了种群初期多样性, 对于提高算法收敛速度和精度都有一定帮助, 虽然这种优势并不明显, 但是对于后期引入其他策略寻优有较大帮助.

### 3.3.2 引入自适应权重因子和非线性收敛因子性能分析

由第 2.2 节的分析可知, 自适应权重因子和非线性收敛因子对于算法全局搜索和局部开发能力的平衡起着至关重要的作用. 为了平衡算法的勘探和开发能力, 本文引入自适应权重因子  $\omega$  和改进的非线性收敛因子  $f$ . 取维度  $d = 30$ , 最大迭代次数  $Max_{iter} = 500$ , 种群规模  $N = 30$ , 其中权重因子  $\omega$  公式中的相关参数已在第 3.1 节给出, 基本 ChOA 与引入自适应权重因子和非线性收敛因子的 ChOA (WChOA) 对于单峰测试函数  $f_2$ 、 $f_7$  和多峰测试函数  $f_{10}$ 、 $f_{11}$  寻优结果对比如图 6 所示.

由图 6 可以看出, 对于单峰函数  $f_2$  和多峰函数  $f_{11}$ , WChOA 能够直接收敛到理论最优值 0; 对于单峰函数  $f_7$ , WChOA 不管是在收敛速度还是在收

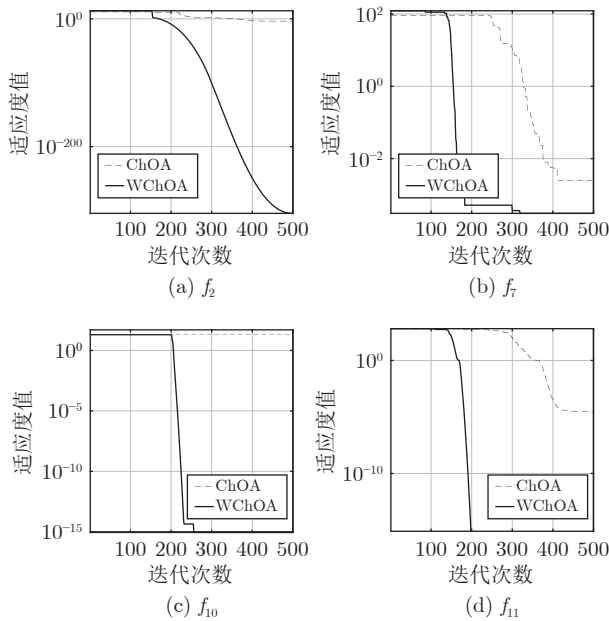


图6 ChOA与WChOA收敛对比图

Fig.6 Convergence curve of ChOA and WChOA

敛精度上都优于 ChOA; 对于多峰函数  $f_{10}$ , WChOA 能收敛到更接近于全局最优解. 综上所述, 引入自适应权重因子和非线性收敛因子对算法寻优速度和精度有一定帮助.

### 3.3.3 引入黄金正弦策略性能分析

黄金正弦算法以正弦函数为基础, 结合了黄金分割算子进行迭代寻优. 为了加快 ChOA 收敛, 提高算法跳出局部最优值的能力, 本文引入黄金正弦策略更新种群位置. 取维度  $d = 30$ , 最大迭代次数  $Max_{iter} = 500$ , 种群规模  $N = 30$ , 搜索间隔  $\alpha = \pi$ ,  $\beta = -\pi$ , 基本 ChOA 与引入黄金正弦策略算法更新位置的 ChOA (GChOA) 对单峰测试函数  $f_4$ 、 $f_6$  和多峰测试函数  $f_8$ 、 $f_{13}$  寻优结果对比如图 7 所示.

由图 7 可知, 对单峰函数  $f_4$ 、 $f_6$  和多峰函数  $f_{13}$ , GChOA 的收敛速度明显快于 ChOA, 并且在收敛精度上也有很大提升; 对复杂多峰函数  $f_8$ , GChOA 能够跳出局部最优值, 收敛到理论最优值  $-12569.48$  附近, 说明引入黄金正弦算法相关思想更新种群位置, 对于算法跳出局部最优解有一定帮助, 同时能加快算法收敛, 提高收敛精度.

### 3.4 IChOA 算法与其他算法性能对比

为了验证 IChOA 对于基本测试函数的寻优性能, 选取基本黑猩猩算法 (ChOA)、粒子群优化算法 (PSO)<sup>[18]</sup>、灰狼优化算法 (GWO)<sup>[19]</sup> 和正弦余弦黑猩猩优化算法 (Sine cosine ChOA, SChOA)<sup>[21]</sup> 进行寻优对比, 各算法统一取维度  $d = 30$ , 最大迭代

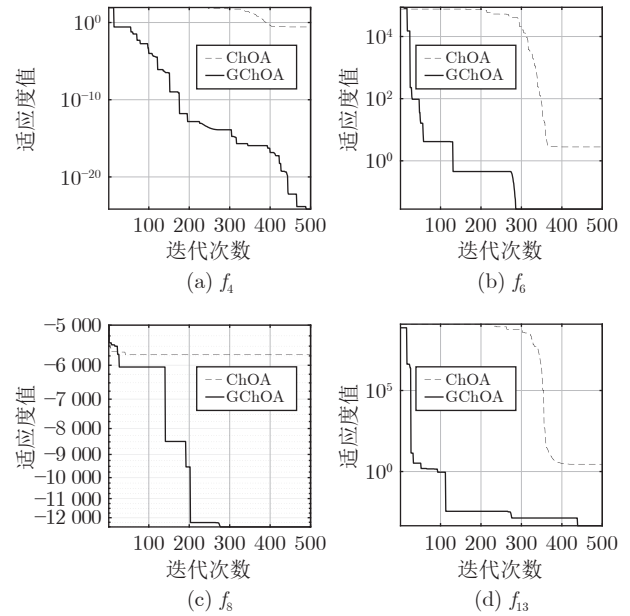


图7 ChOA与GChOA收敛对比图

Fig.7 Convergence curve of ChOA and GChOA

次数  $Max_{iter} = 500$ , 种群规模  $N = 30$ , 各算法其他相关参数由表 1 给出, 利用表 2 给出的 23 个基本测试函数进行算法寻优对比测试, 各算法分别运行 50 次, 取平均值, 对比结果如表 3 所示.

由表 3 对比结果可知, 对于单峰函数, 除了  $f_6$  函数效果比 PSO 算法差外, IChOA 在其他函数上的寻优结果都优于其他智能算法, 其中  $f_1 \sim f_6$  能找到理论最优值. 对于复杂多峰函数, IChOA 表现同样出色, 其中  $f_9$  和  $f_{11}$  能找到最优值. 对于  $f_8$  函数, IChOA 能跳出局部最优解, 更接近于理论最优值  $-12369.49$ . 对于固定维度测试函数, IChOA 基本上都能收敛到最优值附近, 其中  $f_{14}$ 、 $f_{16}$ 、 $f_{18}$  和  $f_{21}$  函数能直接收敛到理论最优值, 说明 IChOA 相比于其他算法, 在基础函数寻优上, 有明显优势.

### 3.5 IChOA 算法与其他算法高维性能对比

为了测试 IChOA 对于高维函数的寻优能力, 选取基本黑猩猩算法 (ChOA)、粒子群优化算法 (PSO)、灰狼优化算法 (GWO) 和鲸鱼优化算法 (WOA) 与 IChOA 进行高维函数寻优对比, 取维度  $d = 500$ , 最大迭代次数  $Max_{iter} = 500$ , 各算法其他相关参数由表 1 给出, 寻优对比结果如图 8 所示.

由图 8 可知, 对于单峰函数  $f_1$ 、 $f_2$ 、 $f_3$ 、 $f_4$  和多峰函数  $f_9$ 、 $f_{11}$ , IChOA 在 500 维时, 仍然能找到最优值 0; 对于单峰函数  $f_6$ , 虽然 IChOA 寻优效果并不是最好的, 但是其收敛速度是最快的; 对于 500 维复杂多峰函数  $f_8$ , IChOA 仍然能跳出局部最优值, 收敛到最优值  $-12569.49$ . 由图 8 可以看出, 对

表 3 各算法寻优结果对比 (30 维)  
Table 3 Comparison of optimization results of each algorithm (30 dim)

函数	ChOA		PSO		GWO		SChOA		IChOA	
	平均值	标准差	平均值	标准差	平均值	标准差	平均值	标准差	平均值	标准差
$f_1$	$1.34 \times 10^{-5}$	$1.19 \times 10^{-20}$	$1.40 \times 10^{-4}$	$2.11 \times 10^{-4}$	$5.95 \times 10^{-28}$	$6.85 \times 10^{-28}$	$5.66 \times 10^{-33}$	$5.68 \times 10^3$	<b>0</b>	<b>0</b>
$f_2$	$1.42 \times 10^{-5}$	$8.55 \times 10^{-21}$	$4.21 \times 10^{-2}$	$4.54 \times 10^{-2}$	$7.95 \times 10^{-17}$	$4.97 \times 10^{-17}$	$1.72 \times 10^{-20}$	$1.91 \times 10^{10}$	<b>0</b>	<b>0</b>
$f_3$	$6.31 \times 10^0$	$1.40 \times 10^{-17}$	$7.01 \times 10^1$	$2.21 \times 10^1$	$2.83 \times 10^{-5}$	$1.12 \times 10^{-4}$	$6.19 \times 10^{-8}$	$2.25 \times 10^4$	<b>0</b>	<b>0</b>
$f_4$	$2.75 \times 10^{-2}$	$5.95 \times 10^{-28}$	$1.08 \times 10^0$	$3.17 \times 10^{-1}$	$5.69 \times 10^{-7}$	$5.55 \times 10^{-7}$	$2.75 \times 10^{-10}$	$1.26 \times 10^0$	<b>0</b>	<b>0</b>
$f_5$	$2.87 \times 10^1$	$2.51 \times 10^{-14}$	$9.67 \times 10^1$	$6.01 \times 10^1$	$2.70 \times 10^1$	$8.26 \times 10^{-1}$	$2.85 \times 10^2$	$1.00 \times 10^7$	<b><math>3.13 \times 10^{-4}</math></b>	<b><math>5.26 \times 10^{-17}</math></b>
$f_6$	$3.72 \times 10^0$	$4.48 \times 10^{-15}$	<b><math>1.10 \times 10^{-4}</math></b>	<b><math>8.28 \times 10^{-5}</math></b>	$7.64 \times 10^{-1}$	$3.58 \times 10^{-1}$	$3.01 \times 10^0$	$5.62 \times 10^3$	$6.51 \times 10^{-3}$	$3.25 \times 10^{-4}$
$f_7$	$1.72 \times 10^{-3}$	$1.09 \times 10^{-18}$	$1.22 \times 10^{-1}$	$4.49 \times 10^{-2}$	$1.72 \times 10^{-3}$	$7.51 \times 10^{-4}$	$1.00 \times 10^{-3}$	$5.77 \times 10^1$	<b><math>7.81 \times 10^{-7}</math></b>	<b><math>1.02 \times 10^{-12}</math></b>
$f_8$	$-5.65 \times 10^3$	$2.75 \times 10^{-12}$	$-4.84 \times 10^3$	$1.15 \times 10^3$	$-6.08 \times 10^3$	$1.02 \times 10^3$	$-9.87 \times 10^3$	$1.80 \times 10^2$	<b><math>-1.26 \times 10^4</math></b>	<b><math>1.83 \times 10^{-11}</math></b>
$f_9$	$1.41 \times 10^1$	0	$4.67 \times 10^1$	$1.16 \times 10^1$	$3.22 \times 10^0$	$4.16 \times 10^0$	<b>0</b>	$7.77 \times 10^1$	<b>0</b>	<b>0</b>
$f_{10}$	$1.96 \times 10^1$	$1.79 \times 10^{-14}$	$2.76 \times 10^{-1}$	$5.09 \times 10^{-1}$	$1.05 \times 10^{-13}$	$2.39 \times 10^{-14}$	$1.50 \times 10^{-14}$	$1.76 \times 10^1$	<b><math>8.88 \times 10^{-16}</math></b>	<b>0</b>
$f_{11}$	$4.79 \times 10^{-2}$	$7.00 \times 10^{-17}$	$9.21 \times 10^{-3}$	$7.74 \times 10^{-3}$	$5.14 \times 10^{-3}$	$9.98 \times 10^{-3}$	<b>0</b>	$8.30 \times 10^1$	<b>0</b>	<b>0</b>
$f_{12}$	$3.98 \times 10^{-1}$	$5.60 \times 10^{-17}$	$6.92 \times 10^{-3}$	$1.19 \times 10^{-2}$	$5.99 \times 10^{-2}$	$9.78 \times 10^{-2}$	$1.62 \times 10^{-1}$	$3.31 \times 10^7$	<b><math>6.46 \times 10^{-4}</math></b>	<b><math>4.39 \times 10^{-18}</math></b>
$f_{13}$	$2.82 \times 10^0$	$1.76 \times 10^{-15}$	$6.68 \times 10^{-3}$	$8.91 \times 10^{-3}$	$6.27 \times 10^{-1}$	$3.06 \times 10^{-1}$	$6.76 \times 10^{-1}$	$5.16 \times 10^7$	<b><math>2.97 \times 10^{-5}</math></b>	<b><math>4.56 \times 10^{-16}</math></b>
$f_{14}$	$1.00 \times 10^0$	$1.12 \times 10^{-15}$	$3.63 \times 10^0$	$2.50 \times 10^0$	$5.09 \times 10^0$	$4.34 \times 10^0$	$1.00 \times 10^0$	$1.04 \times 10^1$	$0.99 \times 10^0$	$4.48 \times 10^{-16}$
$f_{15}$	$1.36 \times 10^{-3}$	$4.39 \times 10^{-19}$	$5.80 \times 10^{-4}$	$2.21 \times 10^{-4}$	$5.73 \times 10^{-3}$	$8.98 \times 10^{-3}$	$6.80 \times 10^{-4}$	$1.70 \times 10^{-3}$	$2.30 \times 10^{-4}$	$5.46 \times 10^{-19}$
$f_{16}$	$-1.03 \times 10^0$	$6.72 \times 10^{-15}$	$-1.03 \times 10^0$	$6.25 \times 10^{-16}$	$-1.03 \times 10^0$	$2.42 \times 10^{-8}$	$-1.03 \times 10^0$	$2.21 \times 10^{-1}$	$-1.03 \times 10^0$	$5.60 \times 10^{-16}$
$f_{17}$	$3.98 \times 10^{-1}$	$3.36 \times 10^{-16}$	$3.98 \times 10^{-1}$	0	$3.98 \times 10^{-1}$	0	$3.99 \times 10^{-1}$	$4.90 \times 10^{-3}$	$3.99 \times 10^{-1}$	$8.98 \times 10^{-16}$
$f_{18}$	$3.00 \times 10^0$	0	$3.18 \times 10^0$	$1.33 \times 10^{-15}$	$5.70 \times 10^0$	$1.47 \times 10^1$	$3.00 \times 10^0$	$1.92 \times 10^{-1}$	$3.00 \times 10^0$	0
$f_{19}$	$-3.85 \times 10^0$	$2.69 \times 10^{-15}$	$-3.86 \times 10^0$	$2.58 \times 10^{-15}$	$-3.86 \times 10^0$	$2.17 \times 10^{-3}$	$-3.86 \times 10^0$	$9.16 \times 10^{-2}$	$-3.72 \times 10^0$	$1.34 \times 10^{-18}$
$f_{20}$	$-1.92 \times 10^0$	$1.12 \times 10^{-15}$	$-3.26 \times 10^0$	$6.05 \times 10^{-2}$	$-3.23 \times 10^0$	$8.43 \times 10^{-2}$	$-3.32 \times 10^0$	$1.24 \times 10^{-1}$	$-2.90 \times 10^0$	$5.23 \times 10^{-15}$
$f_{21}$	$-4.92 \times 10^0$	$2.69 \times 10^{-15}$	$-6.87 \times 10^0$	$3.01 \times 10^0$	$-8.80 \times 10^0$	$2.20 \times 10^0$	$-1.01 \times 10^1$	$3.34 \times 10^{-1}$	$-1.01 \times 10^1$	$2.38 \times 10^{-15}$
$f_{22}$	$-4.99 \times 10^0$	$2.69 \times 10^{-15}$	$-8.46 \times 10^0$	$3.08 \times 10^0$	$-10.22 \times 10^0$	$9.70 \times 10^{-1}$	$-5.18 \times 10^0$	$5.74 \times 10^{-2}$	$-9.78 \times 10^0$	$8.79 \times 10^{-15}$
$f_{23}$	$-5.02 \times 10^0$	$4.48 \times 10^{-16}$	$-8.95 \times 10^0$	$1.78 \times 10^0$	$-9.90 \times 10^0$	$1.96 \times 10^0$	$-1.05 \times 10^1$	$4.95 \times 10^{-2}$	$-9.93 \times 10^0$	$8.97 \times 10^{-16}$

于高维测试函数, IChOA 同样具有很大的优势.

### 3.6 Wilcoxon 秩和检测

Wilcoxon 秩和检验是一种非参数统计检验方法, 能够检测更为复杂的数据分布, 一般的数据分析只是针对当前数据的平均值和标准差, 并没有与算法多次运行的数据进行对比, 因此这种数据对比分析是不科学的. 为了全面体现 IChOA 的优越性, 采用统计分析方法对每一次仿真结果进行分析, 从统计学角度分析 IChOA 与其他算法的性能差异. 选取 IChOA 在 12 个测试函数的运行结果与 PSO、GWO、WOA、ChOA 和 GChOA 运行结果进行 Wilcoxon 秩和检验并计算  $p$  值, 当  $p$  小于 5% 时, 可以被认为是拒绝零假设的有力验证<sup>[22]</sup>. NaN 表示没有数据与算法对比, “+”、“=” 和 “-” 分别表示 IChOA 寻优性能好于、等于和差于其他算法, Wilcoxon 秩和检验结果如表 4 所示.

由表 4 可知, IChOA 的 Wilcoxon 秩和检验结果  $p$  值基本上都小于 5%, 从统计学上说明, IChOA 对于基本函数的寻优性能优势明显, 从而进一步体

现了 IChOA 的鲁棒性.

### 3.7 CEC2014 测试函数实验分析

CEC2014 测试函数<sup>[23]</sup>是由多个基本优化测试函数的权值组合而成, 这种基函数的权值组合使测试函数的特征更为复杂, 利用这些特征复杂的测试函数对本文 IChOA 方法进行测试, 一方面体现 IChOA 对于复杂函数优化的优越性能; 另一方面, 多测试函数的组合优化可以体现 IChOA 对于不同复杂优化问题的适用性. 因此, 为了进一步测试 IChOA 的鲁棒性, 本文选取部分 CEC2014 单目标优化函数进行求解分析, 其中包括单峰、多峰、混合和复合类型函数, 函数相关信息如表 5 所示. 本文将 IChOA 与标准 PSO 算法<sup>[24]</sup>、标准正弦余弦算法 (Sine cosine algorithm, SCA)<sup>[25]</sup>、线性自适应差分进化算法 (Linear success-history based adaptive differential evolution, L-SHADE)<sup>[26]</sup>、引入 Halton 序列初始化种群的黑猩猩优化算法 (HChOA) 和引入黄金正弦优化算法更新个体位置的黑猩猩优化算法 (GChOA) 进行对比. 其中, L-SHADE 算法在 CEC2014



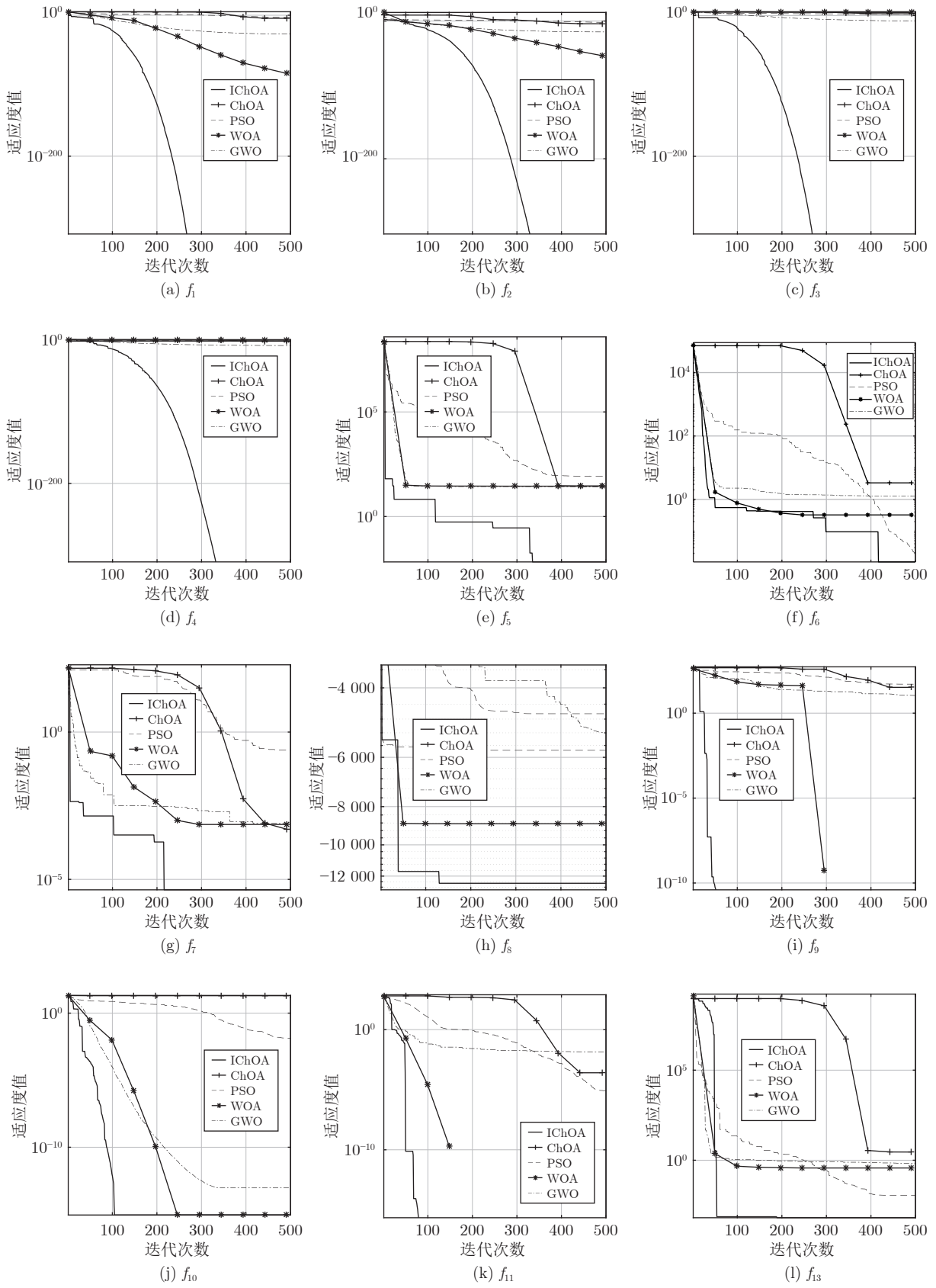


图 8 各算法 500 维寻优对比曲线

Fig. 8 Comparison curves of 500-dimensional optimization of each algorithm

表 4 Wilcoxon 秩和检验结果  
Table 4 Wilcoxon rank sum test results

编号	PSO ( $p_1$ )	GWO ( $p_2$ )	WOA ( $p_3$ )	ChOA ( $p_4$ )	GChOA ( $p_5$ )
$f_1$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$
$f_2$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.25 \times 10^{-20}$
$f_3$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$
$f_4$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$
$f_5$	$1.01 \times 10^{-17}$	$2.47 \times 10^{-17}$	$1.04 \times 10^{-15}$	$2.29 \times 10^{-15}$	$7.96 \times 10^{-18}$
$f_6$	$7.06 \times 10^{-18}$	$1.28 \times 10^{-17}$	$1.38 \times 10^{-15}$	$2.13 \times 10^{-16}$	$7.06 \times 10^{-18}$
$f_7$	$4.20 \times 10^{-17}$	$7.06 \times 10^{-18}$	$6.88 \times 10^{-14}$	$1.36 \times 10^{-17}$	$1.27 \times 10^{-16}$
$f_8$	$7.06 \times 10^{-18}$	$7.06 \times 10^{-18}$	$2.21 \times 10^{-10}$	$7.06 \times 10^{-18}$	$7.06 \times 10^{-18}$
$f_9$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	NaN	$1.17 \times 10^{-19}$	$3.31 \times 10^{-20}$
$f_{10}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$2.39 \times 10^{-16}$	$2.91 \times 10^{-20}$	$2.62 \times 10^{-23}$
$f_{11}$	$3.31 \times 10^{-20}$	$3.31 \times 10^{-20}$	$3.27 \times 10^{-1}$	$2.50 \times 10^{-4}$	$3.31 \times 10^{-20}$
$f_{12}$	$7.06 \times 10^{-18}$	$9.37 \times 10^{-11}$	$1.83 \times 10^{-17}$	$7.96 \times 10^{-18}$	$7.06 \times 10^{-18}$
+/=/-	12/0/0	12/0/0	10/1/1	12/0/0	12/0/0

表 5 部分 CEC2014 函数介绍

Table 5 Introduction of part CEC2014 function

函数	维度	特征	定义域	最佳值
CEC03	30	单峰	$[-100, 100]$	300
CEC05	30	多峰	$[-100, 100]$	500
CEC06	30	多峰	$[-100, 100]$	600
CEC16	30	多峰	$[-100, 100]$	1600
CEC19	30	混合	$[-100, 100]$	1900
CEC22	30	混合	$[-100, 100]$	2200
CEC25	30	复合	$[-100, 100]$	2500
CEC27	30	复合	$[-100, 100]$	2700

测试函数中表现卓越, 常用来作为对比. 实验参数取种群规模为  $N = 50$ , 维度  $d = 30$ , 最大迭代次数  $Max_{iter} = 2000$ , 每个函数独立运行 50 次, 取平均值和标准差. 优化对比结果见表 6.

由表 6 可知, L-SHADE 算法在单峰函数上表现良好, 但是对于多峰、混合和复合函数, IChOA 具有明显优势, 例如对于多峰测试函数 CEC05, IChOA 能够收敛到理论值 500 附近, 而 PSO 算法等其他算法性能相对表现较差; 对于混合和复合测试函数, IChOA 以及其变体优势也比较明显, 在 CEC19 和 CEC22 测试函数上, IChOA 寻优结果更接近理论最优值, 比 PSO 算法、L-SHADE 算法和 SCA 算法更具优势. 此外, 除了 CEC27 函数以外, IChOA 稳定性普遍更高, 验证了 IChOA 的稳定性.

### 3.8 IChOA 平均运行时间和成功率分析

为了验证 IChOA 处理优化问题的速度和成功率, 选取基本 ChOA、HChOA、WChOA 和 GChOA,

与 IChOA 进行基准测试函数平均寻优时间和寻优成功率对比, 实验数据统一设置为: 维度  $d = 30$ , 种群规模  $N = 30$ , 最大迭代次数  $Max_{iter} = 500$ , 各算法运行 30 次, 取运行时间平均值和标准差, 定义算法寻优成功率如下<sup>[27]</sup>.

设适应度误差为  $F(u)$ :

$$F(u) = f(X(u)) - f(X') \quad (18)$$

式中,  $u$  为算法运行次数,  $X(u)$  为算法运行第  $u$  次的实际寻优结果,  $X'$  为理论最优值, 定义变量  $\delta(u)$ :

$$\delta(u) = \begin{cases} 1, & |F(u)| < \varepsilon \\ 0, & |F(u)| \geq \varepsilon \end{cases} \quad (19)$$

式中,  $\varepsilon$  为适应度误差精度, 具体取值见表 2. 算法寻优成功率  $P$  为:

$$P = \frac{1}{30} \sum_{u=1}^{30} \delta(u) \quad (20)$$

各算法对于 23 个基准测试函数的平均寻优时间以及寻优成功率对比结果见表 7.

由表 7 可知, HChOA、WChOA、GChOA 和 IChOA 在优化问题的处理速度上, 相比于 ChOA 都具有一定优势, 除了  $f_9$  和  $f_{14}$  函数外, IChOA 的寻优速度都是最快的, 说明本文提出的改进策略在一定程度上加快了算法收敛速度, 而本文将 3 个策略融合后得到的 IChOA 寻优速度最快; 其次, IChOA 在 16 个基准函数上寻优成功率达到 100% 且在 23 个函数上的寻优成功率相比其他对比算法都是最高的, 说明 IChOA 对于优化问题的处理稳定性较高. 综上所述, 本文提出的 IChOA 在优化问题处理上兼顾速度和稳定性, 具有较高的鲁棒性.

表 6 CEC2014 函数优化对比  
Table 6 CEC2014 function optimization comparison

函数	PSO		SCA		L-SHADE		HChOA		GChOA		IChOA	
	平均值	标准差	平均值	标准差	平均值	标准差	平均值	标准差	平均值	标准差	平均值	标准差
CEC03	$4.87 \times 10^1$	$6.61 \times 10^1$	$8.83 \times 10^0$	$1.36 \times 10^0$	0	0	$7.78 \times 10^4$	$7.44 \times 10^3$	$7.54 \times 10^4$	$6.58 \times 10^3$	$7.35 \times 10^4$	$6.23 \times 10^3$
CEC05	$2.09 \times 10^1$	$8.52 \times 10^{-2}$	$2.21 \times 10^0$	$2.72 \times 10^0$	$2.01 \times 10^1$	$1.70 \times 10^{-2}$	$5.22 \times 10^2$	$6.67 \times 10^{-2}$	$5.26 \times 10^2$	$4.21 \times 10^{-2}$	$5.20 \times 10^2$	$1.02 \times 10^{-2}$
CEC06	$1.08 \times 10^1$	$2.53 \times 10^0$	$6.63 \times 10^1$	$3.74 \times 10^1$	$1.67 \times 10^{-2}$	$9.17 \times 10^{-2}$	$6.33 \times 10^2$	$2.42 \times 10^0$	$6.33 \times 10^{-5}$	$2.42 \times 10^0$	$6.31 \times 10^2$	$2.39 \times 10^0$
CEC16	$1.13 \times 10^1$	$7.05 \times 10^{-1}$	$2.27 \times 10^1$	$1.66 \times 10^{-1}$	$8.48 \times 10^0$	$2.97 \times 10^{-1}$	$1.62 \times 10^3$	$2.81 \times 10^{-1}$	$1.61 \times 10^3$	$1.88 \times 10^{-1}$	$1.61 \times 10^3$	$1.24 \times 10^{-1}$
CEC19	$7.76 \times 10^0$	$1.87 \times 10^0$	$2.88 \times 10^2$	$2.99 \times 10^1$	$3.59 \times 10^0$	$7.22 \times 10^{-1}$	$2.56 \times 10^3$	$2.46 \times 10^0$	$1.76 \times 10^3$	$2.39 \times 10^0$	$2.32 \times 10^3$	$1.96 \times 10^0$
CEC22	$2.31 \times 10^2$	$1.04 \times 10^2$	$2.43 \times 10^1$	$3.03 \times 10^1$	$3.69 \times 10^1$	$3.36 \times 10^1$	$3.57 \times 10^3$	$9.65 \times 10^1$	$4.21 \times 10^3$	$1.48 \times 10^2$	$3.56 \times 10^3$	$7.48 \times 10^1$
CEC25	$2.09 \times 10^2$	$1.65 \times 10^0$	$2.69 \times 10^2$	$2.71 \times 10^1$	$2.03 \times 10^2$	$4.97 \times 10^{-2}$	$2.71 \times 10^3$	$9.49 \times 10^0$	$2.71 \times 10^3$	$3.21 \times 10^0$	$2.70 \times 10^3$	0
CEC27	$5.36 \times 10^2$	$8.15 \times 10^3$	$2.08 \times 10^2$	$1.89 \times 10^1$	$3.00 \times 10^2$	$1.34 \times 10^{-13}$	$2.93 \times 10^3$	$5.36 \times 10^0$	$2.91 \times 10^3$	$8.12 \times 10^0$	$2.90 \times 10^3$	0

表 7 基准函数寻优平均时间及成功率对比  
Table 7 Comparison of average time and success rate for optimization of benchmark function

函数	ChOA			HChOA			WChOA			GChOA			IChOA		
	平均值	标准差	成功率 (%)	平均值	标准差	成功率 (%)	平均值	标准差	成功率 (%)	平均值	标准差	成功率 (%)	平均值	标准差	成功率 (%)
$f_1$	1.9732	0.0136	100	1.9824	0.0088	100	1.9302	0.0155	100	1.4113	0.0181	100	1.3916	0.0101	100
$f_2$	1.9546	0.0101	100	1.9841	0.0181	100	1.7962	0.0063	100	1.4176	0.0101	100	1.4086	0.0081	100
$f_3$	2.2999	0.0083	0	2.2988	0.0077	100	2.0616	0.0906	100	2.1272	0.0128	100	2.1150	0.0275	100
$f_4$	2.0456	0.0107	0	1.9884	0.0309	100	1.9955	0.0676	100	1.4381	0.0330	100	1.4208	0.0446	100
$f_5$	2.0696	0.0663	0	2.0691	0.0649	0	1.9966	0.0326	33.3	1.4670	0.0530	96.6	1.4574	0.0244	100
$f_6$	2.0127	0.0435	0	1.9617	0.0107	36.6	1.9604	0.0138	16.6	1.4122	0.0111	43.3	1.4006	0.0121	90.0
$f_7$	2.0567	0.0089	100	2.0364	0.0190	100	2.0520	0.0147	100	1.5407	0.0070	100	1.5356	0.0098	100
$f_8$	2.0246	0.0127	0	2.0195	0.0310	0	1.9784	0.0294	0	1.4638	0.0086	73.3	1.4625	0.0132	83.3
$f_9$	2.0138	0.0343	0	2.0203	0.0112	96.6	1.9855	0.0096	100	1.4316	0.0184	100	1.4169	0.0141	100
$f_{10}$	2.0011	0.0113	0	2.0045	0.0143	73.3	1.9999	0.0086	100	1.4521	0.0113	100	1.4395	0.0089	100
$f_{11}$	2.0293	0.0081	63.3	2.0301	0.0144	100	2.0155	0.0095	100	1.4776	0.0119	100	1.4735	0.0116	100
$f_{12}$	2.2040	0.0114	0	2.1930	0.0142	33.3	2.1697	0.0169	86.6	1.8354	0.0110	43.3	1.8422	0.0530	100
$f_{13}$	2.1873	0.0112	0	2.1792	0.0115	36.6	2.1811	0.0184	56.6	1.8180	0.0140	66.6	1.8140	0.0084	100
$f_{14}$	0.7893	0.0090	100	0.7916	0.0068	100	0.7898	0.0073	100	1.3830	0.0100	100	1.3571	0.0064	100
$f_{15}$	0.3131	0.0039	0	0.3141	0.0034	50.0	0.3174	0.0024	63.3	0.2792	0.0027	90.0	0.2788	0.0024	96.6
$f_{16}$	0.1730	0.0038	100	0.1743	0.0034	100	0.1728	0.0037	100	0.1647	0.0037	100	0.1647	0.0013	100
$f_{17}$	0.1690	0.0012	100	0.1709	0.0043	100	0.1689	0.0013	100	0.1556	0.0061	100	0.1524	0.0022	100
$f_{18}$	0.1673	0.0025	100	0.1699	0.0035	100	0.1689	0.0021	100	0.1516	0.0012	100	0.1510	0.0020	100
$f_{19}$	0.2804	0.0045	100	0.2847	0.0031	100	0.2849	0.0019	100	0.3021	0.0064	100	0.2580	0.0017	100
$f_{20}$	0.4751	0.0042	46.6	0.4737	0.0024	70.0	0.4724	0.0028	63.3	0.4304	0.0053	63.3	0.4278	0.0033	76.6
$f_{21}$	0.4046	0.0251	16.6	0.4068	0.0035	20.0	0.4012	0.0028	70.0	0.4581	0.0029	76.6	0.4035	0.0055	83.3
$f_{22}$	0.4447	0.0043	0	0.4458	0.0030	36.6	0.4403	0.0050	53.3	0.5391	0.0048	73.3	0.4349	0.0051	80.0
$f_{23}$	0.5087	0.0029	0	0.5089	0.0036	43.3	0.5080	0.0021	36.6	0.6610	0.0034	76.6	0.6347	0.0039	86.6

#### 4 IChOA 算法工程算例应用分析

近年来, 工程算例应用作为优化问题中的一个热门领域, 受到广泛关注. 为了进一步验证 IChOA 在处理实际工程应用案例的优越性, 选取文献 [21]

中的罚函数作为非线性约束条件, 对焊接梁设计问题与拉力/压力弹簧优化设计问题进行优化处理, 并将 IChOA 优化结果与基本遗传算法 (GA)、粒子群优化算法 (PSO)、鲸鱼优化算法 (WOA)、灰狼优化算法 (GWO)、黑猩猩优化算法 (ChOA) 和文

献 [21] 中其他算法进行对比.

### 4.1 焊接梁设计问题

焊接梁设计问题的目的在于降低焊接梁制造成本, 数学模型如式 (21)、式 (22) 所示:

$$\begin{cases} \tau' = \frac{P}{2x_1x_2} \\ \tau'' = MRJ, M = P\left(L + \frac{x_2}{2}\right) \\ J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + \left(\frac{x_1+x_3}{2}\right)^2\right]\right\} \\ R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1+x_3}{2}\right)^2} \\ P = 6000 \\ L = 14 \\ E = 3 \times 10^7 \\ G = 1.2 \times 10^7 \\ \tau_{\max} = 13600 \\ \sigma_{\max} = 30000 \\ \delta_{\max} = 0.25 \end{cases} \quad (21)$$

$$\min f(X) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14 + x_2)$$

$$\text{s.t. } g_1(X) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2} - \tau_{\max} \leq 0$$

$$g_2(X) = \frac{6PL}{x_3^2x_4} - \sigma_{\max} \leq 0$$

$$g_3(X) = x_1 - x_4 \leq 0$$

$$g_4(X) = 0.10471x_1^2 + 0.04811x_3x_4(14 + x_2) - 5 \leq 0$$

$$g_5(X) = 0.125 - x_1 \leq 0$$

$$g_6(X) = \frac{4PL^3}{Ex_3^3x_4} - \delta_{\max} \leq 0$$

$$g_7(X) = P - \frac{4.013Ex_3x_4^3}{6L^2} \left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right) \leq 0$$

$$0.1 \leq x_i \leq 2, i = 1, 4$$

$$0.1 \leq x_i \leq 10, i = 2, 3 \quad (22)$$

式中,  $x_1$ 、 $x_2$ 、 $x_3$  和  $x_4$  分别表示焊接梁的焊缝宽度  $h$ 、横梁宽度  $d$ 、长度  $l$  和厚度  $b$  四个基本属性. 约束条件包括剪切应力  $\tau$ 、横梁弯曲应力  $\delta$ 、屈曲载荷  $J$ 、横梁挠度  $\delta$  和其他内部参数约束. 图 9 为焊接梁模型的基本构造.

表 8 为各算法求解焊接梁设计问题的寻优结果

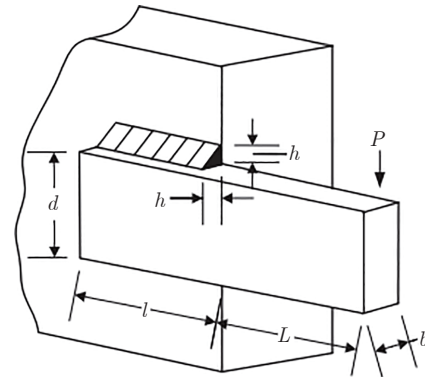


图 9 焊接梁模型

Fig. 9 Welding beam model

表 8 焊接梁设计问题结果对比

Table 8 Comparative results of welding beam design problems

算法	$h$	$l$	$t$	$b$	平均值
GA	0.2455	6.1986	8.1264	0.2247	2.4412
PSO	0.2027	3.4705	9.0366	0.2057	1.7249
WOA	0.2024	3.4772	9.0435	0.2189	1.7299
GWO	0.2022	3.4893	9.0541	0.2155	1.7265
RO	0.2036	3.5284	9.0042	0.2072	1.7353
MVO	0.2054	3.4731	9.0445	0.2056	1.7246
HSSAHHO	0.2057	3.4705	9.0367	0.2057	1.7248
ChOA	0.2214	3.5358	8.9115	0.2127	1.7737
SChOA	0.2057	3.4705	9.0306	0.2056	1.7229
IChOA	0.2038	3.4713	9.0300	0.2060	1.7228

的平均值, 包括射线优化算法 (Ray optimization, RO)<sup>[21]</sup>、多元宇宙算法 (Multi-verse optimizer, MVO)<sup>[21]</sup>、正余弦哈里斯鹰算法 (An efficient hybrid sine-cosine Harris hawks optimization, HSSAHHO)<sup>[21]</sup> 和 SChOA 算法<sup>[21]</sup>. 各算法独立运行 50 次, 取平均值.

由表 8 可以看出, IChOA 对于焊接梁设计问题的优化效果明显高于其他算法, 虽然 IChOA 对于单个焊接梁属性的优化结果不都是最优的, 但其总体造价相比于其他算法优化效果最佳.

### 4.2 拉力/压力弹簧优化设计问题

拉力/压力弹簧优化设计问题的目的是减小弹簧压力/拉力的质量, 该优化设计包含的约束条件有剪应力、振动频率和最小振动挠度, 变量分别用  $x_1$ 、 $x_2$  和  $x_3$  表示, 分别代表弹簧线圈直径  $d$ 、弹簧圈直径  $D$  和绕圈数量  $P$ . 弹簧优化设计问题模型如图 10 所示.

拉力/压力弹簧优化设计问题表示如下:

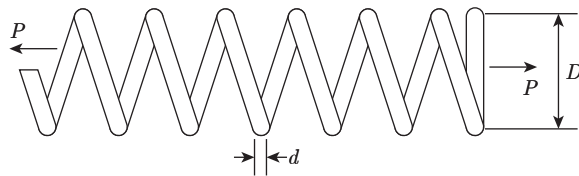


图 10 拉力/压力弹簧优化设计问题模型

Fig.10 The model of tension/pressure spring optimization design

$$\begin{aligned} \min f(x) &= (x_3 + 2)x_1^2 x_2 \\ \text{s.t.} \\ g_1(x) &= 1 - \frac{x_2^3 x_3}{71\,785 x_1^4} \leq 0 \\ g_2(x) &= \frac{4x_2^2 - x_1 x_2}{12\,566(x_2 x_3^3 + x_1^4)} + \frac{1}{5\,108 x_1^2} \leq 0 \\ g_3(x) &= 1 - \frac{140.45 x_1}{x_2^2 x_3} \leq 0 \\ g_4(x) &= \frac{x_1 + x_2}{1.5} \leq 0 \\ 0.05 \leq x_1 \leq 2.0, 0.25 \leq x_2 \leq 1.3, 2.0 \leq x_3 \leq 15.0 \end{aligned} \quad (23)$$

表 9 为各算法求解焊接梁设计问题的寻优结果的平均值, 包括 RO 算法<sup>[21]</sup>、飞蛾火焰优化 (Moth-flame optimization, MFO) 算法<sup>[21]</sup>、HSSAHHO 算法<sup>[21]</sup> 和 SChOA 算法<sup>[21]</sup>. 各算法独立运行 50 次, 取平均值.

表 9 拉力/压力弹簧优化设计问题结果对比  
Table 9 Comparison of tension/compression spring design

算法	$d$	$D$	$P$	平均值
GA	0.0528	0.3523	11.5980	0.01250
PSO	0.0500	0.3174	14.0278	0.01270
WOA	0.5119	0.3452	12.0052	0.01260
GWO	0.5156	0.3562	11.5560	0.01250
RO	0.0413	0.3490	11.7620	0.01260
MFO	0.0510	0.3641	10.8684	0.01260
HSSAHHO	0.0514	0.3535	11.3546	0.01240
ChOA	0.0500	0.3159	14.2629	0.01280
SChOA	0.0524	0.3489	10.6543	0.01187
IChOA	0.0510	0.3374	11.5068	0.01185

由表 9 可知, 虽然 IChOA 对于弹簧的绕圈数量要求略高于 SChOA, 但是对于弹簧线圈直径和弹簧圈直径优化结果都低于 SChOA, 并且总体优化结果略低于 SChOA, 说明 IChOA 对于工程设计问题的优化效果具有优势.

综上所述, 通过以上两个工程设计优化问题实验对比结果可以看出, 本文提出的 IChOA 对于实际工程设计问题的优化, 同样具有一定的优越性, 进一步体现了 IChOA 的鲁棒性.

## 5 结束语

本文在原始 ChOA 基础上, 提出一种融合多策略的黄金正弦黑猩猩优化算法, 通过分析种群初始化分布, 引入 Halton 序列, 提高了算法初期的种群多样性, 引入改进的非线性收敛因子和自适应权重因子, 提高了算法的搜索能力, 结合黄金正弦算法思想更新个体位置, 防止算法过早收敛. 通过对 23 个基本测试函数和部分 CEC2014 测试函数进行寻优测试以及 Wilcoxon 秩和统计检验, 验证了该算法的优越性. 最后通过两个实际工程设计优化问题, 进一步验证了 IChOA 在实际工程应用中的有效性. 通过对本文改进策略的分析可以看出, 改进算法所需参数量较多且算法的寻优性能受非线性收敛因子影响较大, 对于收敛因子的选取依然存在改进的空间. 因此, 今后工作将继续研究改进的优化策略, 以提高 ChOA 的可操作性和适用性, 并将其应用到更为复杂的优化问题中.

## References

- He Dong-Xiao, Zhou Xu, Wang Zuo, Zhou Chun-Guang, Wang Zhe, Jin Di. Complex network community mining: Genetic algorithm based on clustering fusion. *Acta Automatica Sinica*, 2010, **36**(8): 1160–1170 (何东晓, 周栩, 王佐, 周春光, 王喆, 金弟. 复杂网络社区挖掘: 基于聚类融合的遗传算法. *自动化学报*, 2010, **36**(8): 1160–1170)
- Parsopoulos K E, Vrahatis M N. Particle swarm optimization method for constrained optimization problems. *Intelligent Technologies — Theory and Application: New Trends in Intelligent Technologies*, 2002, **76**(1): 214–220.
- Long Wen, Wu Tie-Bin, Tang Ming-Zhu, Xu Ming, Cai Shao-Hong. A Grey wolf optimization algorithm based on lens imaging learning strategy. *Acta Automatica Sinica*, 2020, **46**(10): 2148–2164 (龙文, 伍铁斌, 唐明珠, 徐明, 蔡绍洪. 基于透镜成像学习策略的灰狼优化算法. *自动化学报*, 2020, **46**(10): 2148–2164)
- Arora S, Singh S. Butterfly optimization algorithm: A novel approach for global optimization. *Soft Computing*, 2019, **23**(3): 715–734
- Tharwat A, Elhoseny M, Hassanien A E, Gabel T, Kumar A. Intelligent Bézier curve-based path planning model using chaotic particle swarm optimization algorithm. *Cluster Computing*, 2019, **22**(2): 4745–4766
- Guha D, Roy P K, Banerjee S. Load frequency control of interconnected power system using grey wolf optimization. *Swarm and Evolutionary Computation*, 2016, **27**: 97–115
- Khishe M, Mosavi M R. Chimp optimization algorithm. *Expert Systems With Applications*, 2020, **149**: Article No. 113338
- Lu C, Gao L, Yi J. Grey wolf optimizer with cellular topological structure. *Expert Systems With Applications*, 2018, **107**: 89–114
- Wang Jian-Hao, Zhang Liang, Shi Chao. Whale optimization algorithm based on chaotic search strategy. *Control and Decision*, 2019, **34**(9): 1893–1900

- (王坚浩, 张亮, 史超. 基于混沌搜索策略的鲸鱼优化算法. 控制与决策, 2019, **34**(9): 1893–1900)
- 10 Ning Jie-Qiong, He Qing. Butterfly optimization algorithm improved by hybrid strategy. *Application Research of Computers*, 2021, **38**(6): 1718–1723, 1738  
(宁杰琼, 何庆. 混合策略改进的蝴蝶优化算法. 计算机应用研究, 2021, **38**(6): 1718–1723, 1738)
  - 11 Wang Qiu-Ping, Wang Meng-Na, Wang Xiao-Feng. Grey wolf optimization algorithm with improved convergence factor and proportional weight. *Computer Engineering and Applications*, 2019, **55**(21): 60–65  
(王秋萍, 王梦娜, 王晓峰. 改进收敛因子和比例权重的灰狼优化算法. 计算机工程与应用, 2019, **55**(21): 60–65)
  - 12 Ewees A A, Abd Elaziz M, Houssein E H. Improved grasshopper optimization algorithm using opposition-based learning. *Expert Systems With Applications*, 2018, **112**: 156–172
  - 13 Dinkar S K, Deep K. Opposition based Laplacian ant lion optimizer. *Journal of Computational Science*, 2017, **23**: 71–90
  - 14 Lv Xin, Mu Xiao-Dong, Zhang Jun, Wang Zhen. Chaos sparrow search optimization algorithm. *Journal of Beijing University of Aeronautics and Astronautics*, 2020, **16**(5): 1–10  
(吕鑫, 慕晓冬, 张钧, 王震. 混沌麻雀搜索优化算法. 北京航空航天大学学报, 2020, **16**(5): 1–10)
  - 15 Sayed S A E F, Nabil E, Badr A. A binary clonal flower pollination algorithm for feature selection. *Pattern Recognition Letters*, 2016, **77**: 21–27
  - 16 Bangyal W H, Batool H, Ahmed J, Hafiz P. An improved particle swarm optimization algorithm with chi-square mutation strategy. *International Journal of Advanced Computer Science and Applications*, 2019, **10**(3): 481–491
  - 17 Tanyildizi E, Demir G. Golden sine algorithm: A novel math-inspired algorithm. *Advances in Electrical and Computer Engineering*, 2017, **17**(2): 71–78
  - 18 Xing Yan-Zhen, Wang Dong-Hui. Weight decision gray wolf optimization algorithm based on fuzzy control. *Computer Systems & Applications*, 2018, **27**(10): 202–208  
(邢燕祯, 王东辉. 基于模糊控制的权重决策灰狼优化算法. 计算机系统应用, 2018, **27**(10): 202–208)
  - 19 Guo Wen-Yan, Wang Yuan, Dai Fang, Liu Ting. Alternating sine and cosine algorithm based on elite chaotic search strategy. *Control and Decision*, 2019, **34**(8): 1654–1662  
(郭文艳, 王远, 戴芳, 刘婷. 基于精英混沌搜索策略的交替正弦余弦算法. 控制与决策, 2019, **34**(8): 1654–1662)
  - 20 Wang Chao, Wang Bing-Zhu, Cen Yu-Wan, Xie Neng-Gang. Ion motion algorithm based on diversity global optimal guidance and reverse learning. *Control and Decision*, 2020, **35**(7): 1584–1596  
(汪超, 王丙柱, 岑豫皖, 谢能刚. 基于多样性全局最优引导和反向学习的离子运动算法. 控制与决策, 2020, **35**(7): 1584–1596)
  - 21 Kaur M, Kaur R, Singh N, Dhiman G. SChOA: An newly fusion of sine and cosine with chimp optimization algorithm for HLS of data-paths in digital filters and engineering applications. *Engineering With Computers*, 2021, **42**: 1–29
  - 22 Zhang X M, Wang X, Kang Q. Improved grey wolf optimizer and its application to high dimensional function and FCM optimization. *Control and Decision*, 2019, **10**(8): 1–10
  - 23 Liang J J, Qu B Y, Suganthan P N. Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Single Objective Real-parameter Numerical Optimization, Technical Report, Computational Intelligence Laboratory, Zhengzhou University, China, 2013.
  - 24 Xu G P, Cui Q L, Shi S H, Ge H W, Zhan Z H, Lee H P, et al. Particle swarm optimization based on dimensional learning strategy. *Swarm and Evolutionary Computation*, 2019, **45**: 33–51
  - 25 Chen H L, Wang M J, Zhao X H. A multi-strategy enhanced sine cosine algorithm for global optimization and constrained practical engineering problems. *Applied Mathematics and Computation*, 2020, **369**: Article No. 124872
  - 26 Tanabe R, Fukunaga A S. Improving the search performance of SHADE using linear population size reduction. In: Proceedings of the IEEE Congress on Evolutionary Computation. Beijing, China: IEEE, 2014. 1658–1665
  - 27 Fu Wen-Yuan. Cuckoo search algorithm with gravity acceleration mechanism. *Journal of Software*, 2021, **32**(5): 1480–1494  
(傅文渊. 具有万有引力加速机理的布谷鸟搜索算法. 软件学报, 2021, **32**(5): 1480–1494)



刘成汉 贵州大学大数据与信息工程学院硕士研究生. 主要研究方向为智能优化算法, 深度学习.

E-mail: lzttytm@163.com

(LIU Cheng-Han Master student at the College of Big Data and Information Engineering, Guizhou

University. His research interest covers intelligent optimization algorithm and deep learning.)



何庆 贵州大学大数据与信息工程学院教授. 主要研究方向为认知无线电, 智能算法. 本文通信作者.

E-mail: qhe@gzu.edu.cn

(HE Qing Professor at the College of Big Data and Information Engineering, Guizhou University. His

research interest covers cognitive radio and intelligent algorithms. Corresponding author of this paper.)