工程科学学报, 第 45 卷, 第 3 期: 431−445, 2023 年 3 月
Chinese Journal of Engineering, Vol. 45, No. 3: 431−445, March 2023
https://doi.org/10.13374/j.issn2095-9389.2021.10.27.002; http://cje.ustb.edu.cn

面向全局和工程优化问题的混合进化 JAYA 算法

刘景森[1,2)], 杨　杰[2)], 李　煜[3)✉]

1) 河南大学智能网络系统研究所, 开封 475004　2) 河南大学软件学院, 开封 475004　3) 河南大学管理科学与工程研究所, 开封 475004
✉通信作者, E-mail: leey@henu.edu.cn

摘　要　为了更好求解复杂函数优化和工程约束优化问题, 进一步增强 JAYA 算法的寻优能力, 提出一种面向全局优化的混合进化 JAYA 算法. 首先在计算当前最优和最差个体时引入反向学习机制, 提高最优和最差个体跳离局部极值区域的可能性; 然后在个体位置更新中引入并融合正弦余弦算子和差分扰动机制, 不仅增加了种群的多样性, 而且较好平衡与满足了算法在不同迭代时期对探索和挖掘能力的不同需求; 最后在算法结构上采用奇偶不同的混合进化策略, 有效利用不同演化机制的优势结果, 进一步提升了算法的收敛性和精度. 之后给出了算法流程伪代码, 理论分析证明了改进算法的时间复杂度与基本 JAYA 相同, 而通过 6 种代表性算法在包含和组合了 30 个基准函数的 CEC2017 测试套件上进行的多维度函数极值优化测试, 以及对拉伸弹簧、波纹舱壁、管柱设计、钢筋混凝土梁、焊接梁和汽车侧面碰撞 6 个具有挑战性的工程设计问题的优化求解, 都清楚地表明改进后算法的寻优精度、收敛性能和求解稳定性均有显著提升, 在求解 CEC 复杂函数和工程约束优化问题上有着明显优势.

关键词　JAYA 算法; CEC2017; 正弦余弦算子; 奇偶进化策略; 工程设计优化问题
分类号　TP18

Hybrid evolutionary JAYA algorithm for global and engineering optimization problems

LIU Jing-sen[1,2)], *YANG Jie*[2)], *LI Yu*[3)✉]

1) Institute of Intelligent Networks System, Henan University, Kaifeng 475004, China
2) College of Software, Henan University, Kaifeng 475004, China
3) Institute of Management Science and Engineering, Henan University, Kaifeng 475004, China
✉ Corresponding author, E-mail: leey@henu.edu.cn

ABSTRACT　A swarm intelligence optimization algorithm is an effective method to rapidly solve large-scale complex optimization problems. The JAYA algorithm is a new swarm intelligence evolutionary optimization algorithm, which was proposed in 2016. Compared with other active evolutionary algorithms, the JAYA algorithm has several advantages, such as a clear mechanism, concise structure, and ease of implementation. Further, it has guiding characteristics, obtains the best solution, and avoids the worst solution. The JAYA algorithm has an excellent optimization effect on many problems, and it is one of the most influential algorithms in the field of swarm intelligence. However, when dealing with the CEC test suite, which contains and combines shifted, rotation, hybrid, combination, and other composite characteristics, and the complex engineering constrained optimization problems with considerable difficulty and challenges, the JAYA algorithm has some flaws, that is, it easily falls into the local extremum, its optimization accuracy is sometimes low, and its solution is unstable. To better solve complex function optimization and engineering constrained optimization problems and further enhance the optimization capability of the JAYA algorithm, a global optimization-oriented hybrid evolutionary JAYA algorithm is proposed. First, opposition-based learning is introduced to calculate the current best and worst individuals, which improves the possibility of the best and worst individuals jumping out of the local extremum region. Second, the sine–cosine operator and differential

收稿日期: 2021−10−27
基金项目: 河南省重点研发与推广专项资助项目(182102310886, 222102210065); 国家自然科学基金资助项目(71601071)

disturbance mechanism are introduced and integrated into individual position updating, which not only improves the diversity of the population but also better balances and meets the different requirements of the algorithm for exploration and mining in different iteration periods. Finally, in the algorithm structure, the hybrid evolution strategy with different parity states is adopted and the advantages of different evolution mechanisms are effectively used, which further improves the convergence and accuracy of the algorithm. Then, the pseudocode of the improved algorithm is given, and the theoretical analysis proves that the time complexity of the improved algorithm is consistent with the basic JAYA algorithm. Through the simulation experiment of function extremum optimization of six representative algorithms on multiple dimensions of the CEC2017 test suite, which contains and combines 30 benchmark functions and the optimal solution of six challenging engineering design problems, such as tension/compression spring, corrugated bulkhead, tubular column, reinforced concrete beam, welded beam, and car side impact. The optimal solution of the test results shows that the improved algorithm has significantly improved the optimization accuracy, convergence performance, and solution stability, and it has obvious advantages in solving CEC complex functions and engineering constrained optimization problems.

KEY WORDS JAYA algorithm; CEC2017; sine–cosine operator; parity evolution strategy; engineering design optimization problems

优化问题普遍存在于生产生活的诸多领域, 近年来, 全局优化问题越来越复杂, 规模越来越庞大, 具有高维、非线性、目标函数不可导等特点, 传统的优化方法已经很难有效地进行求解, 而基于群体搜索的元启发式智能优化算法却获得了较好效果, 引起学者们的广泛青睐和研究. 如: 受鸟群捕食过程启发提出的粒子群算法[1−2], 受正余弦函数数学模型启发而提出的正弦余弦算法[3−4], 受座头鲸捕食行为启发提出的鲸鱼优化算法[5], 受量子力学原理和基于量子的原子模型启发提出的原子轨道搜索算法[6] 等. 这些智能优化算法的不断提出、改善和优胜劣汰, 为求解大规模复杂全局优化问题提供了新思路.

函数极值优化问题是测试算法寻优性能的主要方法, 而工程设计约束优化问题则是智能优化算法的重要应用领域. 工程设计问题广泛存在又难以求解, 其目标函数与约束条件的非线性以及决策变量搜索空间不可行域的出现, 使得传统优化方法束手无策, 而群智能优化算法的研究与应用则为解决这类问题提供了实用且有效的方法. 如: Kar 等[7] 提出了一种基于引力搜索算法和粒子群算法的有效混合方法; 刘三阳和靳安钊[8] 提出了一种求解约束优化问题的协同进化教与学优化算法; Banaie-Dezfoul 等[9] 提出了一种狩猎策略的灰狼优化算法; 肖子雅和刘升[10] 提出了一种精英反向学习的黄金正弦鲸鱼算法; Nadimi-Shahraki 等[11] 提出一种基于维度学习狩猎搜索策略的改进灰狼优化算法; 汪逸晖和高亮[12] 提出一种引入动态感知概率、莱维飞行策略以及变异更新机制的改进乌鸦搜索算法. 这些算法均被应用于求解工程约束优化问题, 并取得了良好效果, 但求解的多为几个经典老问题, 种类较少且比较简单, 解决工程设计优化问题仍需探索求解能力更强、寻优精度更高、稳定性和普适性更好的算法.

JAYA 算法是 2016 年由 Rao[13] 提出的一种新型启发式智能优化算法, 该算法控制参数少、易于实现, 且具有趋优避差的导向性特征, 很适于求解全局优化和工程设计优化问题, 成为最近几年优化计算领域重要的研究和改进算法之一, 已被成功应用于旅行商, 文本聚类, 特征选择, 柔性车间调度, 水电站水库优化调度等问题的求解之中.

虽然 JAYA 算法趋优避差的机制特点契合于复杂函数和工程设计中全局优化的思想和需求, 但 JAYA 算法与其他基础性智能优化算法一样, 其本身也存在着容易陷入局部极值、寻优精度有时不高和收敛速度较慢等问题. 为此, 许多学者针对 JAYA 算法的不足之处做了相应改进. Yu 等[14] 引入自适应惯性权重、基于经验的学习策略和混沌精英学习方法, 提高了 JAYA 算法的寻优精度和稳定性. Kang 等[15] 将 JAYA 算法与支持向量机相结合, 并成功应用于桥梁结构健康监测中的温度效应预测问题. Ingle 等[16] 引入莱维飞行和贪婪选择策略, 丰富了 JAYA 算法种群的多样性, 增强了算法的勘探能力, 并成功应用于信道均衡问题. Iacca 等[17] 引入莱维飞行用于 JAYA 算法位置更新产生随机数, 有效提高了算法跳出局部极值的能力. Zhao 等[18] 引入自适应学习策略, 提高了 JAYA 算法的全局搜索能力, 并成功应用于多目标混合零空闲置换流水车间调度问题. Kang 等[19] 构建了基于高斯过程代理模型的 JAYA 算法, 并成功应用于混凝土坝动力参数反演分析, 拓宽了 JAYA 算法的应用领域. Zhang 等[20] 引入局部开发和全局探索策略, 有效增强了 JAYA 算法逃离局部最优的能力. Nayak 等[21] 引入变异策略, 增强了 JAYA

算法全局收敛能力. Zhang 等[22−23]引入包含三种不同学习策略的综合学习机制来更新个体位置，提高了 JAYA 算法的全局搜索能力.

这些改进使 JAYA 算法在各自应用领域的优化性能得以提升，但 JAYA 算法求解全局优化和工程设计优化问题的能力仍有进一步改进的空间. 本文提出一种面向复杂函数和工程设计优化问题的混合进化 JAYA 算法（Hybrid evolutionary JAYA algorithm, H-JAYA）. 首先在计算当前最优和最差个体位置时引入反向学习机制，增强最优和最差个体跳离局部极值区域的可能性. 然后在个体位置更新中引入并融合正弦余弦算子和差分扰动机制，不仅增加了种群的多样性，而且有效平衡和较好满足了算法在不同迭代时期对探索和挖掘能力的不同需求. 最后采用奇偶不同的混合进化策略，有效利用不同演化机制的优势结果，进一步提高了算法的收敛性和精度. 随后给出了算法流程伪代码，用理论分析证明了 H-JAYA 没有增加算法的时间复杂度. 通过对基于多个不同寻优特征基准函数的 CEC2017 测试函数集套件进行多维度、多算法极值优化求解对比测试，结果表明，H-JAYA 的收敛性能、寻优精度和求解稳定性均有明显提升，求解 CEC2017 复杂函数的效果相当优越，全局优化能力出色，非参数统计检验结果也显示了 H-JAYA 与其他对比算法的差异具有显著性. 而对拉伸弹簧、波纹舱壁、管柱设计、钢筋混凝土梁、焊接梁和汽车侧面碰撞 6 个具有挑战性的工程设计约束优化问题的求解，也显示了 H-JAYA 算法在处理不同类型工程优化设计问题时有着明显的优越性和适应性.

1 改进算法 H-JAYA

1.1 基本 JAYA 算法

Step1 设置算法初始参数：种群个体数量 N、最大进化代数 Max_iter、个体维度 D, 并在寻优范围内随机生成每个个体的初始位置 x_i (i=1, 2,…,N).

Step2 根据目标函数计算种群个体的适应度值 $f(x_i)$.

Step3 根据种群中个体的适应度值 $f(x_i)$, 找出最好和最差适应度值 $f_{\min}$ 和 $f_{\max}$, 并记录其位置 x_{best} 和 x_{worst}.

Step4 由式 (1) 对个体每一维的位置进行更新.

$$x_i^j(t+1)=x_i^j(t)+r_1\cdot\left(x_{\text{best}}^j(t)-\left|x_i^j(t)\right|\right)-r_2\cdot\left(x_{\text{worst}}^j(t)-\left|x_i^j(t)\right|\right) \tag{1}$$

其中，$x_{\text{best}}^j(t)$是当前全局最优位置的第 j 维值，$x_{\text{worst}}^j(t)$为当前全局最差位置的第 j 维值，$x_i^j(t)$是当前代中第 i 个个体在第 j 维的位置值，$x_i^j(t+1)$是更新后下一代中第 i 个个体在第 j 维的位置值，r_1、r_2 均为 [0,1] 之间均匀分布的随机数.

Step5 由目标函数 $f(x)$ 求出新个体的适应度值，并对新旧解进行对比，若新解较优，则替换上代的个体位置，否则保留原来的个体位置.

Step6 判断当前迭代次数 t 是否达到最大迭代次数，若$t\leqslant$Max_iter，返回 Step2；

Step7 确定最终的最优值并输出.

1.2 引入反向学习机制

JAYA 算法中种群的当前最优和最差位置具有重要作用，用来引导种群个体向全局最优解进化，但若种群的当前最优和最差位置陷入局部极值区域，就容易导致群体出现搜索停滞的现象，无法获得更优的全局最优解. 基于以上原因，将反向学习机制引入到 JAYA 算法的当前最优和最差位置中.

本文使用的反向学习机制是将基本反向学习与随机反向学习相融合，使 JAYA 算法种群中当前最优和最差个体位置随机进行基本反向学习或随机反向学习，从而增强最优和最差个体跳离局部极值区域的可能性，更好地引导 JAYA 算法种群寻找到全局最优解. 而只对个体位置更新计算中起关键作用的当前最优和最差个体进行反向学习而不是对所有个体再逐一进行反向学习，既有利于避免 JAYA 算法陷于局部最优，又不至于影响算法的收敛速度. 引入反向学习机制的数学模型如下：

$$\text{sign}(\eta)=\begin{cases}1 & \eta<0.5;\\ \text{rand} & \eta\geqslant 0.5.\end{cases} \tag{2}$$

$$x'_{\text{best}}(t)=X_{\min}+(X_{\max}-x_{\text{best}}(t))\cdot\text{sign}(\eta) \tag{3}$$

if $\left(f\left(x'_{\text{best}}(t)\right)<f(x_{\text{best}}(t))\right)$

$$x_{\text{best}}(t+1)=x'_{\text{best}}(t) \tag{4}$$

else

$$x_{\text{best}}(t+1)=x_{\text{best}}(t) \tag{5}$$

end

$$x'_{\text{worst}}(t)=X_{\min}+(X_{\max}-x_{\text{worst}}(t))\cdot\text{sign}(\eta) \tag{6}$$

if $(f(x'_{\text{worst}}(t))>f(x_{\text{worst}}(t)))$

$$x_{\text{worst}}(t+1)=x_{\text{worst}}(t) \tag{7}$$

else

$$x_{\text{worst}}(t+1)=x'_{\text{worst}}(t) \tag{8}$$

end

其中: η为 [0,1] 之间均匀分布的随机数, rand 为 [0, 1] 之间均匀分布的随机数, $X_{\max}$ 和 $X_{\min}$ 分别为种群中个体位置空间的上界和下界. $x_{\text{best}}(t)$ 和 $x_{\text{worst}}(t)$ 分别是当前个体最优和最差位置, $x_{\text{best}}(t+1)$ 和 $x_{\text{worst}}(t+1)$ 分别是更新后的下一代个体最优和最差位置.

1.3　引入正弦余弦算子和差分扰动机制

基本 JAYA 算法, 采用了一个统一的位置更新公式, 机制清晰简单、易于实现, 且具有趋优避差的导向性特征, 对一些问题有着较好的寻优效果. 但对于一些复杂多极值优化问题, 由于缺少不同迭代时期需要不同全局探索和局部挖掘能力与平衡的机制, 导致算法前期的全局搜索有时不够充分, 容易陷入局部极值, 而后期则存在最优解附近局部精细挖掘能力不强的问题, 造成算法有时寻优精度不高和收敛速度较慢的情况. 为此, 在 JAYA 算法的个体位置更新中引入并融合正弦余弦算子和差分扰动机制, 正弦余弦算子通过迭代自适应因子和正弦余弦函数的变化改变 JAYA 算法中种群的个体状态, 增加种群的多样性, 并有效平衡和较好满足了算法在不同迭代时期对探索和挖掘能力的不同需求. 差分扰动机制源于差分进化算法的变异思想, 通过引入差分扰动, 可以增强算法的局部搜索能力, 提高收敛速度, 而采用的双随机差分策略也较好保持了种群的活跃性, 降低算法陷入局部极值的风险. 将上述两种机制产生的位置更新公式通过转换概率 m_4, 分别对应于算法的全局搜索和局部搜索阶段, 有效提高了算法的寻优精度和收敛性能, 数学模型如下:

$m_4 = \text{rand}$

if ($m_4 < 0.5$)

$$x_i^j(t+1) = x_i^j(t) + m_1 \cdot \sin(m_2) \cdot \left(m_3 \cdot x_{\text{best}}^j(t) - \left|x_i^j(t)\right|\right) - m_1 \cdot \cos(m_2) \cdot \left(m_3 \cdot x_{\text{worst}}^j(t) - \left|x_i^j(t)\right|\right) \tag{9}$$

else

if ($f(x_a(t)) < f(x_b(t))$)

$$x_i(t+1) = x_{\text{best}}(t) + F \cdot (x_a(t) - x_b(t)) \tag{10}$$

else

$$x_i(t+1) = x_{\text{best}}(t) - F \cdot (x_a(t) - x_b(t)) \tag{11}$$

end if

end if

其中: m_4 为 [0,1] 之间均匀分布的随机数, $x_{\text{best}}^j(t)$ 是当前全局最优位置的第 j 维值, $x_{\text{worst}}^j(t)$为当前全局最差位置的第 j 维值, $x_i^j(t)$是当前代中第 i 个个体在第 j 维的位置值, $x_i^j(t+1)$是更新后下一代中第 i 个个体在第 j 维的位置值, x_a、x_b 是种群中两个随机个体, 满足 $a \in [1,N]$和 $b \in [1,N]$, 且 $a \neq b \neq i$, 缩放因子 F 为 [0,1] 之间的 D 维随机向量. 在正弦余弦算子机制中, 控制搜索距离的参数 m_2 为 [0,π] 之间均匀分布的随机数, 控制距离对解影响的参数 m_3 为 [0,1] 之间均匀分布的随机数, 而对于控制搜索步长的迭代自适应因子 m_1, 采用了带有随机性的非线性递减策略, 其计算公式为:

$$m_1 = 1 - \ln\left(1 + \frac{\text{rand} \cdot (\text{e} - 1) \cdot t}{\text{Max\_iter}}\right) \tag{12}$$

分析式 (12) 可知, m_1 起着平衡全局搜索和局部搜索能力的重要作用. 在算法迭代前期, m_1 的值较大, 全局搜索步幅较大, 使算法具有较强的全局探索能力, 而在算法迭代后期, m_1 的值较小, 增加了算法在最优解附近区域深度挖掘的能力, 提高了算法的收敛精度. 而 m_1 在保持从 1 到 0 整体非线性递减趋势的同时, 也呈现出一定的随机性, 这种随机性降低了算法如在中前期未能搜索到全局最优值附近, 而在后期因控制因子的单调递减直接陷入局部极值无法跳出的风险.

1.4　奇偶不同的混合进化策略

为了更好发挥和融合改进前后两种不同 JAYA 机制的进化特点, 采用奇偶不同的混合进化策略, 具体描述为: 当迭代次数为奇数时, 使用融合正弦余弦算子和差分扰动的个体位置更新公式进行搜索; 当迭代次数为偶数时, 使用基本 JAYA 算法的个体位置更新公式进行搜索. 同时, 在这两种机制中都引入上面所述的反向学习机制, 而且当更新后所产生个体的适应度值优于上一代个体的适应度值时, 新的个体位置将被接受, 否则按一定概率接受更新后的个体位置. 接受概率 p 的计算公式为:

$$p = \lambda \cdot \left(1 - \text{e}^{\frac{t}{\text{Max\_iter}} - 1}\right) \tag{13}$$

其中, 缩放因子 λ 为 [0,0.1] 之间均匀分布的随机数, 接受概率 p 利用 e 的负指数函数特性随迭代次数增加而非线性递减, 在算法的迭代前期, 接受概率 p 较大, 可以接受较多的差解, 有利于进化的多样性, 随着迭代次数的增加, 接受概率 p 不断减小, 接受较差解越来越少, 最后在接受概率 p 趋于 0 时, 就不再接受任何较差解了, 这在迭代后期有利于算法在最优值附近利用两种不同机制反复进行深度挖掘, 提高了算法的收敛性和精度.

2 H-JAYA 算法流程与时间复杂度分析

2.1 H-JAYA 算法流程

H-JAYA 算法描述如下：

适应度函数 $f(x)$

初始化种群 x_i, i=(1,2,…N)

初始化各参数 Max_iter、D、N

计算种群中个体的适应度值 $f(x_i)$

t=1

while ($t \leqslant$ Max_iter)

根据种群中个体的适应度值 $f(x_i)$, 比较并记录最好和最差解的位置 x_{best} 和 x_{worst}

根据式 (2) ~ (8) 以反向学习机制对最好位置 x_{best} 和最差位置 x_{worst} 进行反向学习

由式 (12) 计算迭代自适应因子 m_1

if (mod (t,2) ≠ 0) /*使用奇偶不同的混合进化策略*/

for i=1:N

m_4 = rand

if (m_4 < 0.5)

for j=1:D

由式 (9) 使用正弦余弦算子对个体第 j 维的位置进行更新

end for j

else

从种群中随机选取两个个体 x_a、x_b

if ($f(x_a(t))$< $f(x_b(t))$)

由式 (10) 通过差分扰动机制对个体的位置进行更新

else

由式 (11) 通过差分扰动机制对个体的位置进行更新

end if

end if

计算更新后个体适应度值 $f(x_{\text{new}})$

if ($f(x_{\text{new}}) < f(x_i)$)

$x_i = x_{\text{new}}$

else

由式 (13) 计算接受概率 p

if (p>rand)

$x_i = x_{\text{new}}$

end if

end if

end for i

else

for i=1:N

for j=1:D

生成 [0,1] 上的随机数 r_1, r_2

由式 (1) 对个体第 j 维的位置进行更新

end for j

计算更新后个体适应度值 $f(x_{\text{new}})$

if ($f(x_{\text{new}}) < f(x_i)$)

$x_i = x_{\text{new}}$

else

由式 (13) 计算接受概率 p

if (p>rand)

$x_i = x_{\text{new}}$

end if

end if

end for i

end if

t=t+1

end while

根据新的适应度值比较并输出最优解位置

2.2 时间复杂度证明

时间复杂度是体现算法性能的关键因素，它反映了算法的运算效率. 文献 [24] 和文献 [25] 分别对蝴蝶优化算法和萤火虫算法的时间复杂度进行了分析，本文采用同样的思想对 H-JAYA 算法的时间复杂度进行分析.

对于群智能优化算法而言，当算法的进化代数和种群大小取值在一个合理范围内时，再增加进化代数和种群大小的值，对提高算法的寻优能力和求解精度已基本没有作用，因此将迭代次数和种群规模设置为一个固定值是群智能优化算法求解问题时普遍采用的方法，而决定算法时间复杂度的基本要素就是代表问题规模的个体空间维度.

在基本 JAYA 算法中，若种群规模为 N，个体位置的维度为 n，设：设置初始参数的时间为 t_0，初始化 JAYA 算法个体位置中每一维的时间为 t_1，求给定目标函数适应度值的时间为 $f(n)$. 则初始化种群阶段的时间复杂度为：

$$T_1 = O(t_0 + N \cdot (n \cdot t_1 + f(n))) = O(n + f(n))$$

进入迭代后，总迭代次数为 Max_iter.

在个体位置更新阶段，设：根据种群中个体的适应度值找出并记录最好和最差个体位置的时间为 t_2，产生均匀分布随机数 r_1、r_2 的时间分别为 t_3，由式 (1) 对个体位置每一维进行更新的时间为 t_4. 则该阶段的时间复杂度为：

$$T_2 = O(t_2 + N \cdot (2 \cdot t_3 + n \cdot t_4)) = O(n)$$

在边界处理和新旧解比较阶段, 设: 每个个体每一维边界处理的时间为 t_5, 计算新个体适应度值的时间为f(n), 新旧解适应度值的比较替换的时间为 t_6, 则此阶段的时间复杂度为:

$$T_3 = O(N\cdot(n\cdot \mathrm{t}_5 + f(n) + \mathrm{t}_6)) = O(n + f(n))$$

综上所述, 基本 JAYA 算法总的时间复杂度为:

$$T(n) = T_1 + \mathrm{Max\_iter}\cdot(T_2 + T_3) = O(n + f(n))$$

在 H-JAYA 改进算法中, 算法的种群规模 N、个体维度 n、参数设置时间、初始化种群和求给定目标函数适应度值的时间均与基本 JAYA 算法一致, 两者初始化过程一样, 因此 H-JAYA 算法在初始化种群阶段的时间复杂度与基本 JAYA 算法相同, 为:

$$T_1' = T_1 = O(n + f(n))$$

进入迭代后, 总迭代次数仍为 Max_iter.

根据种群中个体的适应度值找出并记录最好和最差个体位置的时间仍为 t_2, 设: 由式 (2) 求sign(η)的时间为ε_1, 由式 (3)、式 (6) 分别对最优解、最差解进行反向学习的时间均为 ε_2, 分别对反向学习前后新旧最优解、最差解进行比较替换的时间均为 ε_3, 用式 (13) 产生接受概率 p 的时间为 ε_4, 则这一阶段的时间复杂度为:

$$T_2' = O(t_2 + \varepsilon_1 + 2\cdot\varepsilon_2 + 2\cdot\varepsilon_3 + \varepsilon_4) = O(1)$$

在迭代次数为奇数时的个体位置更新阶段, 设: 由式 (12) 产生 m_1 的时间为 ε_5, m_2、m_3 和转换概率 m_4 都是均匀分布的随机数, 产生它们的时间分别为 t_3, 设 m_4<0.5 执行全局搜索的个体数 k ($0 \leqslant k \leqslant N$), 由式 (9) 对个体位置每一维进行更新的时间为 ε_6; 而$m_4 \geqslant 0.5$执行局部搜索的个体数为 N–k, 此时, 从种群中随机选取两个个体的时间为 ε_7, 计算这两个随机选取个体适应度值的时间分别为 $f(n)$, 比较这两个个体适应度值的时间为 t_6, 用式 (10) 或 (11) 对个体位置进行更新的时间为 ε_8, 则这一阶段的时间复杂度为:

$$T_3' = O(\varepsilon_5 + N\cdot 3\cdot t_3 + k\cdot n\cdot\varepsilon_6 + (N-k)\cdot (\varepsilon_7 + 2\cdot f(n) + t_6 + \varepsilon_8)) = O(n + f(n))$$

在迭代次数为奇数时的边界处理和新旧解比较阶段, 每个个体每一维边界处理的时间、计算更新后个体适应度值的时间、新旧解适应度值的比较替换时间均与基本 JAYA 算法在边界处理和新旧解比较阶段的时间一致, 故该阶段的时间复杂度为:

$$T_4' = T_3 = O(n + f(n))$$

在迭代次数为偶数时的个体位置更新阶段, 产生均匀分布随机数 r_1、r_2 的时间分别为 t_3、由式 (1) 进行每一维个体位置更新的时间仍为 t_4. 故该阶段的时间复杂度为:

$$T_5' = O(N\cdot(2\cdot t_3 + n\cdot t_4)) = O(n)$$

在迭代次数为偶数时的边界处理和新旧解比较阶段, 每个个体每一维边界处理的时间、计算更新后个体适应度值的时间、新旧解适应度值的比较替换时间均与迭代次数为奇数时的边界处理和新旧解比较阶段相同. 故该阶段的时间复杂度为:

$$T_6' = T_4' = O(n + f(n))$$

综上所述, H-JAYA 算法总的时间复杂度为:

$$T'(n) = T_1' + \mathrm{Max\_iter}\cdot T_2' + \frac{\mathrm{Max\_iter}}{2}\cdot (T_3' + T_4' + T_5' + T_6') = O(n + f(n))$$

由此可知, 改进算法 H-JAYA 和基本 JAYA 算法的时间复杂度相同, 没有降低算法的运行效率.

3　仿真实验

为了全面检验改进算法 H-JAYA 的寻优能力, 本文的仿真实验分为两部分进行. 3.1 节将算法在 CEC2017 测试函数集[26] 上进行多维度函数极值优化测试, 用以验证算法的寻优性能与收敛能力. 3.2 节研究了 6 个具有挑战性的工程设计约束优化问题, 用以检验算法在不同类型工程优化设计问题上的求解能力和应用潜力. 两部分实验都将本文算法 H-JAYA 与基本 JAYA 算法 (2016)[13]、Improved JAYA Optimization Algorithm(IJAYA, 2017)[14]、Comprehensive Learning Jaya Algorithm(CLJAYA, 2020)[22−23]、鲸鱼优化算法(WOA, 2016)[5] 和 Hybrid Firefly and Particle Swarm Optimization Algorithm (HFPSO, 2018)[27] 共 6 种性能优越的代表性算法进行了对比实验.

为了保证实验的公平性与客观性, 6 种对比算法采用相同的软、硬件平台在相同条件下独立运行 50 次, 运行环境为 Windows10、编程语言为 MATLAB R2019a, 种群大小均为 30, 最大进化代数 Max_iter=1000. 算法参数设置方面, 4 种 JAYA 类算法无需另设参数, 而 WOA 算法只需设置用于定义对数螺旋的常数 b=1, HFPSO 算法中学习因子 c1 与 c2 均为 1.49445, 这些参数的取值都与各自算法的原文献和源代码取值相同.

3.1　函数极值优化仿真实验

3.1.1　寻优精度分析

本文对 CEC2017 测试集中 30 个函数都进行

了测试和分析，因篇幅限制下面只讨论按序给出的混合测试函数 $f_{11}(x)$、$f_{12}(x)$、$f_{19}(x)$ 和 $f_{20}(x)$，组合测试函数 $f_{21}(x)$、$f_{22}(x)$、$f_{25}(x)$、$f_{26}(x)$、$f_{29}(x)$ 和 $f_{30}(x)$，其他函数的测试结果与之类似，不再一一赘述.

表 1 统计了 6 种算法分别在 10 维、50 维和 100 维时，对于不同测试函数各自独立运行 50 次，得到的最佳值、平均值和方差.

由表 1 的数据可以看出，除个别函数外，改进算法 H-JAYA 在各函数不同维度下的求解精度和稳定性均明显优于其他 5 种对比算法，显示出 H-JAYA 算法对于 JAYA 机制改进的显著性和有效性.

在 10 维条件下，对于函数 $f_{11}(x)$、$f_{19}(x)$ 和 $f_{20}(x)$，H-JAYA 在这 3 个函数上的寻优结果最佳值均为理论最优值，求解效果十分出色；CLJAYA、IJAYA、JAYA、HFPSO 和 WOA 算法的最佳值、平均值和方差都差于 H-JAYA 算法. 对于函数 $f_{11}(x)$、$f_{26}(x)$、$f_{29}(x)$ 和 $f_{30}(x)$，H-JAYA 在这 4 个函数上的最佳值、平均值和方差虽未达到理论最优值，但全部优于其他 5 种算法，求解能力明显突出和优越. 对于函数 $f_{21}(x)$，H-JAYA 的最佳值与 CLJAYA 算法

表 1 6 种算法在固定迭代次数下的寻优结果比较

Table 1 Comparison of the optimization results of six representative algorithms under fixed iteration times

Functions	Algorithms	D=10			D=50			D=100		
		Best	Mean	Variance	Best	Mean	Variance	Best	Mean	Variance
$f_{11}(x)$	H-JAYA	1.10×10^{3}	1.11×10^{3}	1.40×10^{1}	1.37×10^{3}	1.74×10^{3}	1.65×10^{5}	1.50×10^{4}	3.94×10^{4}	2.47×10^{8}
	IJAYA	1.11×10^{3}	1.12×10^{3}	2.62×10^{1}	3.87×10^{3}	7.80×10^{3}	2.20×10^{6}	1.68×10^{5}	2.55×10^{5}	1.32×10^{9}
	CLJAYA	1.11×10^{3}	1.18×10^{3}	4.57×10^{3}	5.95×10^{3}	1.27×10^{4}	1.37×10^{7}	9.30×10^{4}	1.32×10^{5}	3.71×10^{8}
	HFPSO	1.11×10^{3}	1.16×10^{3}	2.65×10^{3}	5.19×10^{3}	1.43×10^{4}	3.54×10^{7}	1.12×10^{5}	2.37×10^{5}	5.10×10^{9}
	JAYA	1.14×10^{3}	1.19×10^{3}	1.19×10^{3}	5.59×10^{3}	9.84×10^{3}	6.79×10^{6}	1.66×10^{5}	2.68×10^{5}	2.61×10^{9}
	WOA	1.12×10^{3}	1.22×10^{3}	1.04×10^{4}	2.93×10^{3}	5.25×10^{3}	1.65×10^{6}	1.26×10^{5}	2.31×10^{5}	6.78×10^{9}
$f_{12}(x)$	H-JAYA	2.80×10^{3}	1.01×10^{4}	3.15×10^{7}	6.03×10^{6}	4.80×10^{7}	5.94×10^{15}	4.18×10^{8}	1.46×10^{9}	1.74×10^{18}
	IJAYA	8.60×10^{4}	4.01×10^{5}	1.35×10^{11}	1.18×10^{9}	1.85×10^{9}	1.55×10^{17}	9.17×10^{9}	1.20×10^{10}	2.80×10^{18}
	CLJAYA	3.44×10^{3}	4.28×10^{5}	1.67×10^{12}	7.36×10^{9}	2.36×10^{10}	6.40×10^{19}	8.88×10^{10}	1.35×10^{11}	3.73×10^{20}
	HFPSO	3.02×10^{3}	1.60×10^{4}	1.33×10^{8}	1.01×10^{7}	8.26×10^{7}	6.55×10^{15}	7.82×10^{9}	1.24×10^{10}	1.99×10^{19}
	JAYA	6.17×10^{5}	8.03×10^{6}	4.38×10^{13}	5.39×10^{9}	7.86×10^{9}	1.77×10^{18}	2.86×10^{10}	4.13×10^{10}	5.16×10^{19}
	WOA	1.11×10^{4}	4.99×10^{6}	2.17×10^{13}	4.90×10^{8}	1.52×10^{9}	4.04×10^{17}	6.85×10^{9}	1.30×10^{10}	9.73×10^{18}
$f_{19}(x)$	H-JAYA	1.90×10^{3}	1.91×10^{3}	5.47×10^{1}	2.84×10^{3}	3.03×10^{4}	3.91×10^{9}	7.24×10^{4}	9.21×10^{5}	1.60×10^{12}
	IJAYA	1.95×10^{3}	2.89×10^{3}	9.08×10^{5}	1.00×10^{6}	7.99×10^{6}	2.24×10^{13}	1.78×10^{8}	4.39×10^{8}	1.88×10^{16}
	CLJAYA	1.91×10^{3}	1.95×10^{3}	1.61×10^{3}	1.16×10^{6}	2.96×10^{7}	1.00×10^{15}	1.97×10^{9}	7.38×10^{9}	9.59×10^{18}
	HFPSO	1.94×10^{3}	1.10×10^{4}	8.86×10^{7}	3.75×10^{5}	3.69×10^{6}	1.30×10^{13}	3.11×10^{7}	1.72×10^{8}	3.85×10^{16}
	JAYA	1.94×10^{3}	3.13×10^{3}	8.75×10^{6}	4.27×10^{6}	1.36×10^{8}	6.37×10^{15}	1.32×10^{9}	2.58×10^{9}	3.29×10^{17}
	WOA	2.37×10^{3}	7.89×10^{4}	4.84×10^{10}	3.42×10^{5}	1.07×10^{7}	8.86×10^{13}	2.99×10^{7}	1.25×10^{8}	4.01×10^{15}
$f_{20}(x)$	H-JAYA	2.00×10^{3}	2.02×10^{3}	1.49×10^{2}	3.03×10^{3}	3.37×10^{3}	2.51×10^{4}	5.01×10^{3}	5.78×10^{3}	6.10×10^{4}
	IJAYA	2.04×10^{3}	2.07×10^{3}	2.05×10^{2}	3.74×10^{3}	4.11×10^{3}	3.54×10^{4}	7.21×10^{3}	7.76×10^{3}	7.01×10^{4}
	CLJAYA	2.02×10^{3}	2.07×10^{3}	2.09×10^{3}	3.06×10^{3}	3.55×10^{3}	8.07×10^{4}	5.66×10^{3}	6.82×10^{3}	3.74×10^{5}
	HFPSO	2.03×10^{3}	2.14×10^{3}	4.44×10^{3}	2.86×10^{3}	3.69×10^{3}	1.84×10^{5}	5.65×10^{3}	7.25×10^{3}	3.76×10^{5}
	JAYA	2.05×10^{3}	2.09×10^{3}	9.96×10^{2}	3.83×10^{3}	4.28×10^{3}	2.81×10^{4}	7.30×10^{3}	7.94×10^{3}	8.05×10^{4}
	WOA	2.07×10^{3}	2.19×10^{3}	6.29×10^{3}	3.19×10^{3}	3.91×10^{3}	1.40×10^{5}	5.37×10^{3}	7.12×10^{3}	3.94×10^{5}
$f_{21}(x)$	H-JAYA	2.20×10^{3}	2.33×10^{3}	2.87×10^{1}	2.50×10^{3}	2.55×10^{3}	1.04×10^{3}	3.30×10^{3}	3.47×10^{3}	5.39×10^{3}
	IJAYA	2.21×10^{3}	2.33×10^{3}	6.44×10^{2}	2.67×10^{3}	2.79×10^{3}	1.69×10^{3}	3.45×10^{3}	3.61×10^{3}	5.55×10^{3}
	CLJAYA	2.20×10^{3}	2.33×10^{3}	4.23×10^{2}	2.74×10^{3}	2.96×10^{3}	6.65×10^{3}	3.85×10^{3}	4.16×10^{3}	2.72×10^{4}
	HFPSO	2.21×10^{3}	2.33×10^{3}	3.11×10^{3}	2.71×10^{3}	2.82×10^{3}	3.30×10^{3}	3.48×10^{3}	3.65×10^{3}	1.09×10^{4}
	JAYA	2.33×10^{3}	2.34×10^{3}	3.50×10^{1}	2.78×10^{3}	2.86×10^{3}	1.17×10^{3}	3.61×10^{3}	3.79×10^{3}	8.65×10^{3}
	WOA	2.21×10^{3}	2.33×10^{3}	2.79×10^{3}	2.81×10^{3}	3.05×10^{3}	1.16×10^{4}	3.91×10^{3}	4.36×10^{3}	5.14×10^{4}

表 1 (续)

Table 1 (Continued)

Functions	Algorithms	D=10			D=50			D=100		
		Best	Mean	Variance	Best	Mean	Variance	Best	Mean	Variance
$f_{22}(x)$	H-JAYA	2.30×10^{3}	2.31×10^{3}	1.69×10^{1}	2.58×10^{3}	1.09×10^{4}	3.02×10^{6}	2.09×10^{4}	2.50×10^{4}	5.31×10^{6}
	IJAYA	2.30×10^{3}	2.31×10^{3}	1.79×10^{0}	1.41×10^{4}	1.63×10^{4}	2.41×10^{5}	3.27×10^{4}	3.44×10^{4}	4.14×10^{5}
	CLJAYA	2.22×10^{3}	2.29×10^{3}	1.40×10^{3}	1.27×10^{4}	1.48×10^{4}	7.48×10^{5}	2.87×10^{4}	3.27×10^{4}	2.48×10^{6}
	HFPSO	2.31×10^{3}	2.39×10^{3}	1.02×10^{5}	1.24×10^{4}	1.50×10^{4}	1.82×10^{6}	2.83×10^{4}	3.24×10^{4}	3.61×10^{6}
	JAYA	2.23×10^{3}	2.32×10^{3}	5.18×10^{2}	1.54×10^{4}	1.66×10^{4}	2.06×10^{5}	3.32×10^{4}	3.48×10^{4}	3.48×10^{5}
	WOA	2.24×10^{3}	2.51×10^{3}	2.09×10^{5}	1.10×10^{4}	1.43×10^{4}	2.11×10^{6}	2.65×10^{4}	3.09×10^{4}	3.67×10^{6}
$f_{25}(x)$	H-JAYA	2.60×10^{3}	2.89×10^{3}	1.23×10^{4}	3.09×10^{3}	3.22×10^{3}	5.72×10^{3}	4.33×10^{3}	5.20×10^{3}	3.38×10^{5}
	IJAYA	2.90×10^{3}	2.90×10^{3}	8.93×10^{1}	3.34×10^{3}	3.59×10^{3}	1.92×10^{4}	7.41×10^{3}	9.95×10^{3}	1.60×10^{6}
	CLJAYA	2.90×10^{3}	2.95×10^{3}	2.57×10^{2}	6.91×10^{3}	9.93×10^{3}	2.01×10^{6}	1.68×10^{4}	2.16×10^{4}	4.38×10^{6}
	HFPSO	2.90×10^{3}	2.93×10^{3}	8.41×10^{2}	3.59×10^{3}	4.20×10^{3}	1.93×10^{5}	4.97×10^{3}	5.79×10^{3}	4.79×10^{5}
	JAYA	2.93×10^{3}	2.96×10^{3}	1.27×10^{2}	3.93×10^{3}	4.58×10^{3}	1.61×10^{5}	1.05×10^{4}	1.48×10^{4}	3.79×10^{6}
	WOA	2.90×10^{3}	2.95×10^{3}	8.12×10^{2}	3.75×10^{3}	4.22×10^{3}	1.04×10^{5}	6.55×10^{3}	8.12×10^{3}	6.34×10^{5}
$f_{26}(x)$	H-JAYA	2.79×10^{3}	2.99×10^{3}	8.70×10^{2}	7.08×10^{3}	9.01×10^{3}	3.21×10^{5}	2.23×10^{4}	2.54×10^{4}	2.05×10^{6}
	IJAYA	2.90×10^{3}	3.11×10^{3}	1.76×10^{5}	8.98×10^{3}	1.10×10^{4}	9.90×10^{5}	2.58×10^{4}	3.01×10^{4}	5.24×10^{6}
	CLJAYA	2.91×10^{3}	3.05×10^{3}	2.59×10^{4}	1.04×10^{4}	1.32×10^{4}	1.82×10^{6}	2.97×10^{4}	4.02×10^{4}	1.82×10^{7}
	HFPSO	2.81×10^{3}	3.01×10^{3}	1.15×10^{5}	5.62×10^{3}	9.75×10^{3}	5.34×10^{6}	7.77×10^{3}	1.82×10^{4}	1.41×10^{7}
	JAYA	3.00×10^{3}	3.52×10^{3}	3.11×10^{5}	1.01×10^{4}	1.14×10^{4}	3.74×10^{5}	2.58×10^{4}	2.97×10^{4}	4.55×10^{6}
	WOA	2.83×10^{3}	3.61×10^{3}	3.86×10^{5}	1.03×10^{4}	1.41×10^{4}	1.63×10^{6}	2.82×10^{4}	3.62×10^{4}	9.11×10^{6}
$f_{29}(x)$	H-JAYA	3.14×10^{3}	3.17×10^{3}	6.75×10^{2}	5.10×10^{3}	5.89×10^{3}	1.33×10^{5}	8.34×10^{3}	9.85×10^{3}	6.71×10^{5}
	IJAYA	3.15×10^{3}	3.20×10^{3}	8.47×10^{2}	5.31×10^{3}	6.26×10^{3}	1.39×10^{5}	1.18×10^{4}	1.37×10^{4}	9.48×10^{5}
	CLJAYA	3.15×10^{3}	3.19×10^{3}	1.44×10^{3}	5.28×10^{3}	8.55×10^{3}	2.60×10^{6}	1.89×10^{4}	3.32×10^{4}	1.12×10^{8}
	HFPSO	3.17×10^{3}	3.27×10^{3}	4.25×10^{3}	4.88×10^{3}	6.25×10^{3}	4.44×10^{5}	9.86×10^{3}	1.15×10^{4}	1.20×10^{6}
	JAYA	3.16×10^{3}	3.22×10^{3}	1.79×10^{3}	6.14×10^{3}	6.91×10^{3}	2.38×10^{5}	1.44×10^{4}	1.83×10^{4}	5.09×10^{6}
	WOA	3.19×10^{3}	3.39×10^{3}	1.59×10^{4}	5.54×10^{3}	8.56×10^{3}	3.26×10^{6}	1.39×10^{4}	1.85×10^{4}	1.34×10^{7}
$f_{30}(x)$	H-JAYA	4.40×10^{3}	1.06×10^{4}	3.57×10^{8}	7.80×10^{5}	9.97×10^{6}	1.83×10^{15}	3.44×10^{6}	1.82×10^{7}	4.14×10^{14}
	IJAYA	7.46×10^{3}	6.66×10^{5}	1.22×10^{12}	5.80×10^{7}	1.38×10^{8}	2.57×10^{15}	3.95×10^{8}	6.93×10^{8}	2.14×10^{16}
	CLJAYA	4.48×10^{3}	1.98×10^{6}	2.45×10^{12}	5.31×10^{7}	3.07×10^{8}	2.70×10^{16}	3.09×10^{9}	1.32×110	3.27×10^{19}
	HFPSO	9.86×10^{3}	7.09×10^{5}	6.11×10^{11}	6.77×10^{7}	1.49×10^{8}	2.60×10^{15}	2.62×10^{8}	9.42×10^{8}	3.74×10^{17}
	JAYA	9.69×10^{3}	3.75×10^{5}	1.18×10^{11}	6.08×10^{7}	1.95×10^{8}	1.71×10^{16}	2.18×10^{9}	4.27×10^{9}	5.93×10^{17}
	WOA	8.17×10^{3}	1.11×10^{6}	1.65×10^{12}	9.36×10^{7}	2.54×10^{8}	1.27×10^{16}	6.34×10^{8}	1.35×10^{9}	3.02×10^{17}

相同, 优于其他 4 种算法, 平均值与 CLJAYA、IJAYA、HFPSO 和 WOA 算法相同, 优于基本 JAYA 算法, 而且方差是 6 种算法中最小的. 对于函数 $f_{22}(x)$, H-JAYA 的最佳值与 IJAYA 算法相同, 略逊于 WOA、JAYA 和 CLJAYA 算法, 优于 HFPSO 算法, 平均值与 IJAYA 算法相同, 略逊于 CLJAYA 算法, 优于其他 3 种算法, 方差略逊于 IJAYA 算法, 优于其他 4 种算法. 对于函数 $f_{25}(x)$, H-JAYA 的最佳值和平均值均优于其他 5 种算法, 方差略逊于其他 5 种算法.

在 50 维和 100 维的高维条件下, 6 种算法的求解精度都会随着维度的增加有所降低, 但相较于其他 5 种算法, H-JAYA 算法仍表现出良好的寻优精度和稳定性. 在 50 维条件下, 对于函数 $f_{11}(x)$、$f_{12}(x)$、$f_{19}(x)$、$f_{21}(x)$、$f_{25}(x)$ 和 $f_{30}(x)$, H-JAYA 在这 6 个函数上的最佳值、平均值和方差都优于其他 5 种算法. 对于函数 $f_{22}(x)$, H-JAYA 的最佳值和平均值都优于其他 5 种算法, 方差略逊于其他 5 种算法. 对于函数 $f_{20}(x)$、$f_{26}(x)$ 和 $f_{29}(x)$, H-JAYA 在这 3 个函数上的最佳值略逊于 HFPSO 算法, 优于其

他 4 种算法，平均值和方差优于 5 种对比算法. 在 100 维条件下，对于函数 $f_{11}(x)$、$f_{12}(x)$、$f_{19}(x)$、$f_{20}(x)$、$f_{21}(x)$、$f_{25}(x)$、$f_{29}(x)$ 和 $f_{30}(x)$,H-JAYA 在这 8 个函数上的最佳值、平均值和方差均优于其他 5 种算法，表现出优越的高维求解能力. 对于函数 $f_{22}(x)$, H-JAYA 的最佳值和平均值都优于其他 5 种算法，方差略逊于其他 5 种算法. 对于函数 $f_{26}(x)$, H-JAYA 的最佳值和平均值略逊于 HFPSO 算法，但方差优于该算法，而且 H-JAYA 的最佳值、平均值和方差均优于其他 4 种算法.

以上求解结果和分析表明，相比于 5 种寻优性能较为优秀的代表性对比算法，H-JAYA 算法整体呈现出更为优越的求解性能，较好解决了 JAYA 算法在求解复杂函数全局优化时易陷入局部极值、寻优性能不稳定，寻优精度不高的问题.

3.1.2 收敛曲线分析

一个算法性能的优劣，可以直观地通过收敛曲线展现出来，收敛曲线显示了算法在寻优过程中陷入局部最优的次数和收敛速度. 下面给出对于 3.1.1 节 $f_{21}(x)$、$f_{22}(x)$、$f_{25}(x)$、$f_{26}(x)$、$f_{29}(x)$ 和 $f_{30}(x)$ 共 6 个求解难度较大的组合函数的收敛曲线图，其他函数的收敛曲线对比结果与之相似，不再一一赘述. 图 1 是 6 种算法在维度 D=100 时对于这 6 个函数的收敛曲线对比图.

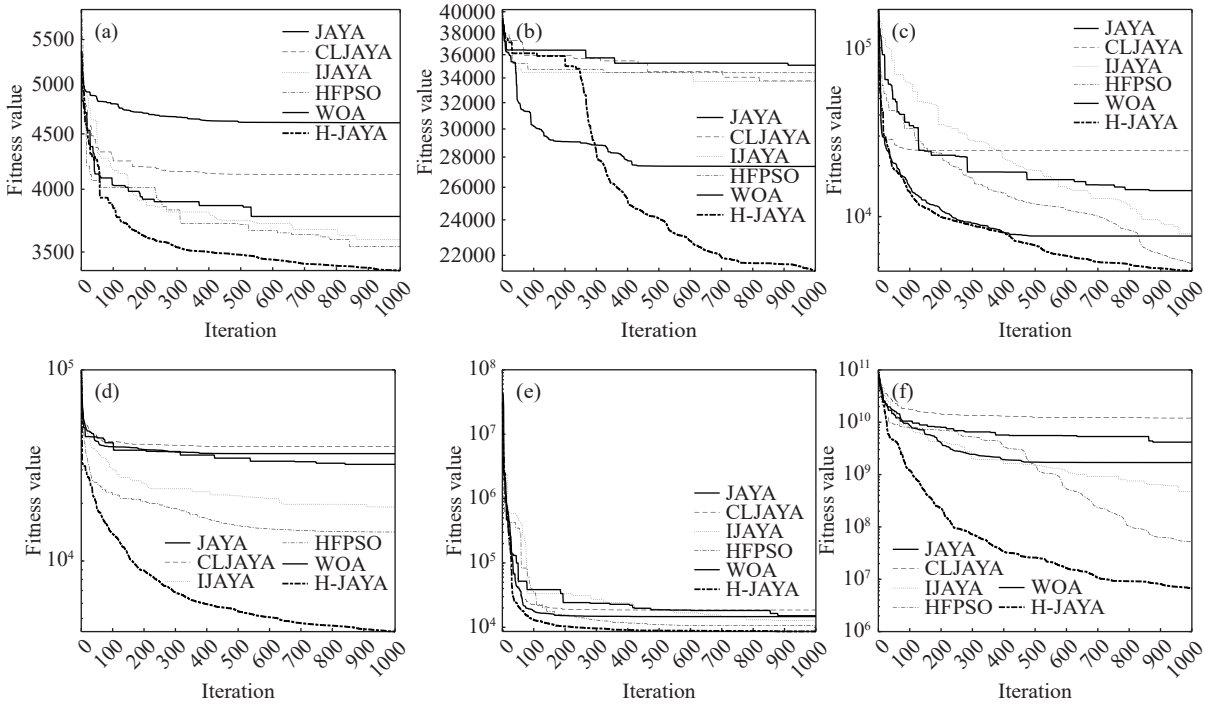

图 1 收敛曲线. (a) $f_{21}(x)$; (b) $f_{22}(x)$; (c) $f_{25}(x)$; (d) $f_{26}(x)$; (e) $f_{29}(x)$; (f) $f_{30}(x)$

Fig.1 Convergence curves: (a) $f_{21}(x)$; (b) $f_{22}(x)$; (c) $f_{25}(x)$; (d) $f_{26}(x)$; (e) $f_{29}(x)$; (f) $f_{30}(x)$

以上图 1 清晰地展现了 H-JAYA、JAYA、CLJAYA、IJAYA、WOA 和 HFPSO 算法在进化过程中适应度值的变化趋势. 从这些图中可以看出 H-JAYA 收敛速度快于其他 5 种算法，且收敛曲线也总体比较光滑，陷入局部极值的次数较少，寻优能力较强.

在图 1(a) 中，CLJAYA 和 WOA 算法收敛速度一直慢于 JAYA，更明显慢于 H-JAYA; IJAYA 在迭代前期收敛速度略慢于 JAYA，160 代以后则快于 JAYA，但一直明显慢于 H-JAYA; HFPSO 收敛速度在迭代前期与 JAYA 互有快慢，260 代以后则快于 JAYA，且在迭代早期快于 H-JAYA，但在 60 代以后明显慢于 H-JAYA. 在图 1(b) 中，CLJAYA 收敛速度在迭代前期与 JAYA 十分相近，在 480 代以后快于 JAYA，但始终明显慢于 H-JAYA; IJAYA 收敛速度一直快于 JAYA，且在迭代前期略快于 H-JAYA，但在 280 代以后明显慢于 H-JAYA; WOA 收敛速度一直明显快于 JAYA，且在迭代前期快于 H-JAYA，但在 300 代以后，随着 H-JAYA 收敛速度加快，WOA 已明显慢于 H-JAYA; HFPSO 收敛速度一直快于 JAYA，且在迭代前期快于 H-JAYA，但在 290 代以后则明显慢于 H-JAYA. 在图 1(c) 中，CLJAYA 收敛速度前期快于 JAYA，但在 170 代以后慢于 JAYA，更是一直明显慢于 H-JAYA; IJAYA 收敛速度在迭代前期慢于 JAYA，570 代以后快于 JAYA，但一直明显慢于 H-JAYA; WOA 收敛速度一直明

显快于 JAYA, 且在迭代早期与 H-JAYA 相近, 但在 50 代以后随着 H-JAYA 收敛速度加快, WOA 慢于 H-JAYA; HFPSO 收敛速度在迭代前期与 JAYA 相近, 在 200 代以后明显快于 JAYA, 但始终慢于 H-JAYA. 在图 1(d) 中, CLJAYA 和 WOA 算法收敛速度慢于 JAYA, 更明显慢于 H-JAYA; IJAYA 和 HFPSO 算法收敛速度快于 JAYA, 但明显慢于 H-JAYA. 在图 1(e) 中, CLJAYA 收敛速度在迭代前期快于 JAYA, 460 代以后却相近并最终慢于 JAYA, 更是始终慢于 H-JAYA; IJAYA 收敛速度在迭代前期与 JAYA 快慢交替基本相近, 530 代以后则快于 JAYA, 但一直慢于 H-JAYA; WOA 收敛速度从迭代开始到 880 代一直快于 JAYA, 880 代以后与 H-JAYA 相近, 但始终慢于 H-JAYA; HFPSO 收敛速度在迭代前期慢于 JAYA, 100 代以后则快于 JAYA, 但一直慢于 H-JAYA. 在图 1(f) 中, CLJAYA 收敛速度慢于 JAYA, 更明显慢于 H-JAYA; IJAYA、WOA 和 HFPSO 算法收敛速度一直快于 JAYA, 但都慢于 H-JAYA.

综上所述, 本文提出的 H-JAYA 算法在低、中、高三种维度下对于具有较高求解难度的 CEC-2017 测试函数集都有较好的寻优性能, 其求解精度、收敛速度和寻优稳定性均优于 JAYA、CLJAYA、IJAYA、WOA 和 HFPSO 等 5 种代表性对比算法, 表现出较为显著的求解优越性.

3.1.3　实验结果的秩和检验统计分析

为了验证改进算法 H-JAYA 与其他对比算法在实验结果上的差异具有显著性, 进一步评价算法的寻优性能, 采用非参数统计检验方法–Wilcoxon 秩和检验, 进行统计分析. 表 2 给出了对于 CEC2017 测试函数集套件, H-JAYA 分别与其他对比算法在 D=100 时求解 3.1.1 节具有代表性的 10 个函数的统计检验结果. 其中, 用符号“+”、“–”和“=”分别表示 H-JAYA 的寻优结果优于、劣于和相当于其他对比算法. 由文献 [28] 可知, 那些 p<0.05 的结果就可被认为是拒绝零假设具有显著性的有力验证.

从表 2 的统计检验结果来看, 改进算法 H-JAYA 与 JAYA、IJAYA、CLJAYA、WOA 和 HFPSO 算法相比, 在全部 10 个函数上的检验 p 值都小于 0.05 且符号为“+”, 拒绝零假设. 由此可见, H-JAYA 与其他 5 种对比算法的计算结果之间具有显著差异, 且 H-JAYA 显著更优.

3.2　H-JAYA 求解工程设计约束优化问题

为了检验改进算法 H-JAYA 求解工程约束优化问题的能力, 将 H-JAYA 算法和上述 5 种代表性对比算法应用于拉伸弹簧、波纹舱壁、管柱设计、钢筋混凝土梁、焊接梁和汽车侧面碰撞 6 个工程优化设计问题. 这 6 个工程问题具有各自不同的约束条件和求解难度, 能够很好地测试出各算法求解工程设计优化问题的能力和适用性. 在求解这些工程优化问题时, 将每种算法独立运行 50 次, 得到设计结果的最佳值、平均值和方差作为评价各算法求解能力的指标.

3.2.1　求解拉伸弹簧设计问题

拉伸弹簧设计是一个最小化约束问题, 其目的是设计一种重量最轻且满足挠度、剪切应力、波动频率、外径 4 个约束条件的拉伸弹簧. 该问

表 2　各算法求解 CEC2017 测试集套件 Wilcoxon 秩和检验的 p 值

Table 2　p-value for Wilcoxon's rank-sum test on each algorithm used to solve the CEC2017 test suite

Functions	H-JAYA *vs* JAYA	H-JAYA *vs* IJAYA	H-JAYA *vs* CLJAYA	H-JAYA *vs* WOA	H-JAYA *vs* HFPSO
	p-value win	p-value win	p-value win	p-value win	p-value win
$f_{11}(x)$	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)
$f_{12}(x)$	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)
$f_{19}(x)$	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)
$f_{20}(x)$	2.6734×10^{-4}(+)	2.6727×10^{-4}(+)	2.6727×10^{-4}(+)	2.6727×10^{-4}(+)	2.6727×10^{-4}(+)
$f_{21}(x)$	2.6693×10^{-4}(+)	2.6700×10^{-4}(+)	2.6727×10^{-4}(+)	2.6720×10^{-4}(+)	2.6720×10^{-4}(+)
$f_{22}(x)$	2.6720×10^{-4}(+)	2.6727×10^{-4}(+)	2.6734×10^{-4}(+)	2.6734×10^{-4}(+)	2.6734×10^{-4}(+)
$f_{25}(x)$	2.6734×10^{-4}(+)	2.6734×10^{-4}(+)	2.6734×10^{-4}(+)	2.6727×10^{-4}(+)	2.6727×10^{-4}(+)
$f_{26}(x)$	2.6734×10^{-4}(+)	2.6734×10^{-4}(+)	2.6727×10^{-4}(+)	2.6734×10^{-4}(+)	2.6734×10^{-4}(+)
$f_{29}(x)$	2.6734×10^{-4}(+)	2.6720×10^{-4}(+)	2.6734×10^{-4}(+)	2.6734×10^{-4}(+)	2.6734×10^{-4}(+)
$f_{30}(x)$	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)	2.6917×10^{-13}(+)
(+/–/=)	10/0/0	10/0/0	10/0/0	10/0/0	10/0/0

题包含 3 个设计变量，分别是线径 $x_1(0.05 \leqslant x_1 \leqslant 2.00)$、平均线圈直径 $x_2(0.25 \leqslant x_2 \leqslant 1.30)$、活性线圈数量 $x_3(2.00 \leqslant x_3 \leqslant 15.0)$. 该设计问题的数学模型如下：

目标函数：$f(x) = (x_3 + 2) \cdot x_2 \cdot x_1^2$;

约束条件：$g_1(x) = 1 - \dfrac{x_2^3 \cdot x_3}{71785 \cdot x_1^4} \leqslant 0$,

$$g_2(x) = \frac{4 \cdot x_2^2 - x_1 \cdot x_2}{12566 \cdot (x_2 \cdot x_1^3 - x_1^4)} + \frac{1}{5108 \cdot x_1^2} - 1 \leqslant 0,$$

$$g_3(x) = 1 - \frac{140.45 \cdot x_1}{x_2^2 \cdot x_3} \leqslant 0, g_4(x) = \frac{x_1 + x_2}{1.5} - 1 \leqslant 0.$$

表 3 是改进算法 H-JAYA 和其他 5 种对比算法各自独立运行 50 次求解拉伸弹簧设计问题得到的最佳值、平均值和方差. 从表中数据可以看出，H-JAYA 算法的最佳值、平均值和方差都优于其他 5 种算法，表现出较优的求解能力和稳定性.

表 3 6 种算法求解拉伸弹簧设计问题的寻优结果比较

Table 3 Comparison of the optimization results of six representative algorithms used to solve the tension/compression spring design problem

Algorithms	Best	Mean	Variance
H-JAYA	$1.2665237000\times10^{-2}$	$1.2806874020\times10^{-2}$	$2.7477865853\times10^{-8}$
IJAYA	$1.2666032600\times10^{-2}$	$1.3032429118\times10^{-2}$	$2.1994218985\times10^{-6}$
CLJAYA	$1.2666232800\times10^{-2}$	$1.3068941340\times10^{-2}$	$1.2273508194\times10^{-6}$
JAYA	$1.2666917100\times10^{-2}$	$1.3185287610\times10^{-2}$	$1.9559284633\times10^{-6}$
HFPSO	$1.2665806000\times10^{-2}$	$1.2982082240\times10^{-2}$	$3.0289473131\times10^{-7}$
WOA	$1.2671936200\times10^{-2}$	$1.3894943310\times10^{-2}$	$2.4615922288\times10^{-6}$

3.2.2 求解波纹舱壁设计问题

波纹舱壁设计问题的目标是最小化波纹舱壁的重量，该问题包含 6 个约束条件和 4 个设计变量，设计变量分别是波纹舱壁的宽度 $x_1(0 \leqslant x_1 \leqslant 100)$、波纹舱壁的高度 $x_2(0 \leqslant x_2 \leqslant 100)$、波纹舱壁的长度 $x_3(0 \leqslant x_3 \leqslant 100)$、波纹舱壁的厚度 $x_4(0 \leqslant x_4 \leqslant 5)$，其数学模型如下：

目标函数：$f(x) = \dfrac{5.885\, x_4 (x_1 + x_3)}{x_1 + \sqrt{|x_3^2 - x_2^2|}}$;

约束条件：$g_1(x) = -x_4 x_2 \left(0.4 x_1 + \dfrac{x_3}{6}\right) + 8.94\left(x_1 + \sqrt{|x_3^2 - x_2^2|}\right) \leqslant 0$,

$$g_2(x) = -x_4 x_2^2 \left(0.2 x_1 + \frac{x_3}{12}\right) + 2.2\left(8.94\left(x_1 + \sqrt{|x_3^2 - x_2^2|}\right)\right)^{4/3} \leqslant 0,$$

$$g_3(x) = -x_4 + 0.015 x_1 + 0.15 \leqslant 0, g_4(x) = -x_4 + 0.0156 x_3 + 0.15 \leqslant 0, g_5(x) = -x_4 + 1.05 \leqslant 0, g_6(x) = -x_3 + x_2 \leqslant 0.$$

表 4 是 6 种算法 50 次求解波纹舱壁设计问题得到的最佳值、平均值和方差. 由表中数据可知，H-JAYA 算法的最佳值与 IJAYA、CLJAYA 算法相同，且优于其余 3 种算法，而平均值和方差则优于所有 5 种对比算法，表现出更好的寻优精度和稳定性.

表 4 6 种算法求解波纹舱壁设计问题的寻优结果比较

Table 4 Comparison of the optimization results of six representative algorithms used to solve the corrugated bulkhead design problem

Algorithms	Best	Mean	Variance
H-JAYA	6.8429580101	6.8574579730	$3.5011426525\times10^{-4}$
IJAYA	6.8429580101	7.5431661442	1.0303170764
CLJAYA	6.8429580101	8.2229580101	1.4648979592
JAYA	6.9429580101	8.6689580101	$7.9828979592\times10^{-1}$
HFPSO	6.8924274599	7.0360704247	$2.4342757860\times10^{-1}$
WOA	6.8589881925	7.1845962530	$2.1316652625\times10^{-1}$

3.2.3 求解管柱设计问题

管柱设计问题的目标是以最低成本设计一个均匀的管状截面柱来承载压缩载荷，该设计问题包含 6 个约束条件和 2 个设计变量，设计变量分别为管柱的平均直径 $x_1(2 \leqslant x_1 \leqslant 14)$，管壁的厚度 $x_2(0.2 \leqslant x_2 \leqslant 0.8)$，其数学模型如下：

目标函数：$f(x) = 9.8 x_1 x_2 + 2 x_1$;

约束条件：$g_1(x) = \dfrac{P}{\pi\, x_1 x_2 \sigma_y} - 1 \leqslant 0$,

$$g_2(x) = \frac{8 p L^2}{\pi^3 E x_1 x_2 \left(x_1^2 + x_2^2\right)} - 1 \leqslant 0,$$

$$g_3(x) = \frac{2}{x_1} - 1 \leqslant 0,$$

$$g_4(x) = \frac{x_1}{14} - 1 \leqslant 0, g_5(x) = \frac{0.2}{x_2} - 1 \leqslant 0, g_6(x) = \frac{x_2}{8} - 1 \leqslant 0.$$

其中：管柱弹性模量 $E = 0.85 \times 10^6$，屈服强度 σ_y= 500.

表 5 是 6 种算法求解管柱设计问题时得到的最佳值、平均值和方差. 由表中数据可以清楚地看出，H-JAYA 算法的最佳值与 CLJAYA 相同，且优于其余 4 种算法，而平均值和方差则优于其他全部 5 种算法，展现出更为优越的寻优能力和求解稳定性.

3.2.4 求解钢筋混凝土梁设计问题

钢筋混凝土梁是用钢筋混凝土材料制成的梁，其形式多种多样，是房屋建筑、桥梁建筑等工

表 5　6 种算法求解管柱设计问题的寻优结果比较

Table 5　Comparison of the optimization results of six representative algorithms used to solve the tubular column design problem

Algorithms	Best	Mean	Variance
H-JAYA	2.6486361472×10^{1}	2.6486976485×10^{1}	$2.7492263055\times10^{-6}$
IJAYA	2.6486361473×10^{1}	2.6770361473×10^{1}	$3.8922448975\times10^{-2}$
CLJAYA	2.6486361472×10^{1}	2.6840361472×10^{1}	$2.8248979592\times10^{-2}$
JAYA	2.6686361472×10^{1}	2.6910363872×10^{1}	$1.3289800884\times10^{-2}$
HFPSO	2.6491554294×10^{1}	2.6524256607×10^{1}	$4.0282119604\times10^{-4}$
WOA	2.6491740158×10^{1}	2.6831750349×10^{1}	$1.0260601687\times10^{-1}$

程结构中最基本的承重构件, 应用范围极广. 该问题的目标是以最低成本设计一个有最大承载力的钢筋混凝土梁, 问题中包含 2 个约束条件和 3 个变量, 这些变量分别是钢筋的面积$x_1(x_1=\{6,6.16,6.32,6.6,7,7.11,7.2,7.8,7.9,8,8.4\})$, 梁的宽度$x_2(x_2=\{28,29,30,\ldots,40\})$, 梁的长度$x_3(5\leqslant x_3\leqslant 10)$, 其数学模型如下:

目标函数: $f(x)=2.9x_1+0.6x_2x_3$;

约束条件: $g_1(x)=\dfrac{x_2}{x_3}-4\leqslant 0$, $g_2(x)=180+7.375\dfrac{x_1^2}{x_3}-x_1x_2\leqslant 0$.

表 6 统计了 6 种算法运行 50 次求解钢筋混凝土梁设计问题得到的最佳值、平均值和方差. 由表中数据可知, H-JAYA 算法求得的最佳值、平均值和方差都是 6 种算法中最好的, 求解性能更加出色.

表 6　6 种算法求解钢筋混凝土梁问题的寻优结果比较

Table 6　Comparison of the optimization results of six representative algorithms used to solve the reinforced concrete beam design problem

Algorithms	Best	Mean	Variance
H-JAYA	3.5920800000×10^{2}	3.6080773841×10^{2}	2.8418472149×10^{2}
IJAYA	3.6225000000×10^{2}	3.7488260888×10^{2}	1.5910386068×10^{2}
CLJAYA	3.6225000000×10^{2}	3.7447556000×10^{2}	1.3625233588×10^{2}
JAYA	3.6225000000×10^{2}	3.9080088880×10^{2}	2.4313100946×10^{2}
HFPSO	3.6925001569×10^{2}	3.8520700009×10^{2}	7.3955644549×10^{1}
WOA	3.6225106247×10^{2}	3.6700943742×10^{2}	4.3808572351×10^{1}

3.2.5　求解焊接梁设计问题

焊接梁设计的目标是在剪切应力、弯曲应力、屈曲载荷、端部挠度和侧面约束下找到最低制造成本. 该问题有 7 个约束条件和 4 个设计变量, 设计变量分别是焊缝厚度$x_1(0.125\leqslant x_1\leqslant 2)$、焊缝长度$x_2(0.1\leqslant x_2\leqslant 10)$、梁宽度$x_3(0.1\leqslant x_3\leqslant 10)$、梁厚度$x_4(0.1\leqslant x_4\leqslant 2)$, 剪切应力 τ, 横梁弯曲应力 σ, 屈曲载荷 P_c, 横梁挠度 δ 以及各设计变量之间的尺寸约束, 具体数学模型如下:

目标函数: $f(x)=1.10471x_1^2\cdot x_2+0.04811\cdot x_3\cdot x_4(14.0+x_2)$;

约束条件: $g_1(x)=\tau(x)-136000\leqslant 0$,

$$g_2(x)=\sigma(x)-30000\leqslant 0,\ g_3(x)=x_1-x_4\leqslant 0,$$

$$g_4(x)=0.10471\cdot x_1^2+0.04811\cdot x_3\cdot x_4(14.0+x_2)-5.0\leqslant 0,$$

$$g_5(x)=0.125-x_1\leqslant 0,\ g_6(x)=\delta(x)-0.25\leqslant 0,$$

$$g_7(x)=6000-P_c(x)\leqslant 0,\ \tau(x)=\sqrt{(\tau')^2+2\tau'\cdot\tau''\frac{x_2}{2\cdot R}+(\tau'')^2},\ \tau'=\frac{6000}{\sqrt{2}x_1\cdot x_2},\ \tau''=\frac{M\cdot R}{J},$$

$$M=6000\left(14+\frac{x_2}{2}\right),\ R=\sqrt{\frac{x_2^2}{4}+\left(\frac{x_1+x_2}{2}\right)^2},$$

$$J=2\left\{\sqrt{2}x_1\cdot x_2\left[\frac{x_2^2}{12}+\left(\frac{x_1+x_3}{2}\right)^2\right]\right\},$$

$$\sigma(x)=\frac{784000}{x_4\cdot x_3^2},\ \delta(x)=\frac{4\times600\times14^3}{30\times10^6\cdot x_3^3\cdot x_4},$$

$$P_c(x)=\frac{2.0065\cdot\sqrt{x_3^2\cdot x_4^6}}{14^2}\left(1-\frac{50\cdot x_3}{28}\right).$$

表 7 统计了 6 种算法求解焊接梁设计问题得到的最佳值、平均值和方差. 观察表中数据可以看出, 改进算法 H-JAYA 的最佳值、平均值和方差都是 6 种算法最好的, 而且 H-JAYA 算法 50 次寻优结果的最佳值与文献 [29] 中给出的最优值相同, 达到理论最优, 求解效果相当出色.

表 7　6 种算法求解焊接梁设计问题的寻优结果比较

Table 7　Comparison of the optimization results of six representative algorithms used to solve the welded beam design problem

Algorithms	Best	Mean	Variance
H-JAYA	1.6702177263	1.6808983776	$1.2670965769\times10^{-3}$
IJAYA	1.6702258303	1.6902400352	$2.0000233481\times10^{-2}$
CLJAYA	1.6702177264	1.6902198653	$1.9999912854\times10^{-2}$
JAYA	1.6702177328	1.8165708743	$1.2530675483\times10^{-1}$
HFPSO	1.6702682457	2.0454210525	$8.4760880179\times10^{-2}$
WOA	1.7177501400	2.3609633625	$5.8101787131\times10^{-1}$

3.2.6　求解汽车侧面碰撞设计问题

汽车侧面碰撞设计问题的目标是研究车门部件对汽车侧面碰撞安全性的影响, 通过改进汽车的结构, 以最大限度地提高汽车的碰撞安全性, 从

而降低交通事故的伤害. 该问题包括 11 个设计变量和 10 个约束条件, 设计变量分别是 B 柱的内板厚度 $x_1(0.5 \leqslant x_1 \leqslant 1.5)$、B 柱的加强板厚度 $x_2(0.5 \leqslant x_2 \leqslant 1.5)$、车顶横梁长度 $x_3(0.5 \leqslant x_3 \leqslant 1.5)$、B 柱的外板材料屈服强度 $x_4(0.5 \leqslant x_4 \leqslant 1.5)$、门槛梁长度 $x_5(0.5 \leqslant x_5 \leqslant 1.5)$、抗侧撞梁长度 $x_6(0.5 \leqslant x_6 \leqslant 1.5)$、车顶边梁长度 $x_7(0.5 \leqslant x_7 \leqslant 1.5)$、B 柱的内板材料屈服强度 $x_8(x_8 \in \{0.192, 0.345\})$、车底部横梁长度 $x_9(x_9 \in \{0.192, 0.345\})$、碰撞位置最小角度 $x_{10}(x_{10} \geqslant -30)$、碰撞位置最大角度 $x_{11}(x_{11} \leqslant 30)$, 具体数学模型如下:

目标函数: $f(x) = 1.98 + 4.90x_1 + 6.67x_2 + 6.98x_3 + 4.01x_4 + 1.78x_5 + 2.37x_7$;

约束条件:

$$g_1(x) = 1.16 - 0.3717x_2x_4 - 0.00931x_2x_{10} - 0.484x_3x_9 + 0.01343x_6x_{10} - 1 \leqslant 0,$$

$$g_2(x) = 46.36 - 9.9x_2 - 12.9x_1x_2 + 0.1107x_3x_{10} - 32 \leqslant 0,$$

$$g_3(x) = 33.86 + 2.95x_3 + 0.1792x_3 - 5.057x_1x_2 - 11.0x_2x_8 - 0.0215x_5x_{10} - 9.98x_7x_8 + 22.0x_8x_9 - 32 \leqslant 0,$$

$$g_4(x) = 28.98 + 3.818x_3 - 4.2x_1x_2 + 0.0207x_5x_{10} + 6.63x_6x_9 - 7.7x_7x_8 + 0.32x_9x_{10} - 32 \leqslant 0,$$

$$g_5(x) = 0.261 - 0.0159x_1x_2 - 0.188x_1x_8 - 0.019x_2x_7 + 0.0144x_3x_5 + 0.0008757x_5x_{10} + 0.08045x_6x_9 + 0.00139x_8x_{11} + 0.00001575x_{10}x_{11} - 0.32 \leqslant 0,$$

$$g_6(x) = 0.214 + 0.00817x_5 - 0.131x_1x_8 - 0.0704x_1x_9 + 0.03099x_2x_6 - 0.018x_2x_7 + 0.0208x_3x_8 + 0.121x_3x_9 - 0.00364x_5x_6 + 0.0007715x_5x_{10} - 0.0005354x_6x_{10} + 0.00121x_8x_{11} + 0.00184x_9x_{10} - 0.02x_2^2 - 0.32 \leqslant 0,$$

$$g_7(x) = 0.74 - 0.61x_2 - 0.163x_3x_8 + 0.001232x_3x_{10} - 0.166x_7x_9 + 0.227x_2^2 - 0.32 \leqslant 0,$$

$$g_8(x) = 4.72 - 0.5x_4 - 0.19x_2x_3 - 0.0122x_4x_{10} + 0.009325x_6x_{10} + 0.000191x_{11}^2 - 4 \leqslant 0,$$

$$g_9(x) = 10.58 - 0.674x_1x_2 - 1.95x_2x_8 + 0.02054x_3x_{10} - 0.0198x_4x_{10} + 0.028x_6x_{10} - 9.9 \leqslant 0,$$

$$g_{10}(x) = 16.45 - 0.489x_3x_7 - 0.843x_5x_6 + 0.0432x_9x_{10} - 0.0556x_9x_{11} - 0.000786x_{11}^2 - 15.7 \leqslant 0.$$

表 8 统计了 6 种算法 50 次求解汽车侧面碰撞设计问题得到的寻优结果最佳值、平均值和方差. 从表中数据可以看出, H-JAYA 算法的最佳值、平均值和方差都是 6 种算法最好的, 表现出优越的求解能力和稳定性.

以上求解结果分析以及相关文献[30−32]的求解结果都表明, 智能优化算法对于求解工程设计约束优化问题是有效的, 而本文所使用的代表性对比算法在求解工程问题时都具有良好的求解性能, 其中改进算法 H-JAYA 在面对不同类型和复杂程度的工程优化设计问题时, 更是展现出十分优越的求解精度和稳定性, 对于求解工程设计约束优化问题有着明显可见的优越性和适用性.

表 8 6 种算法求解汽车侧面碰撞问题的寻优结果比较

Table 8 Comparison of the optimization results of six representative algorithms used to solve the car side impact design problem

Algorithms	Best	Mean	Variance
H-JAYA	2.2848738268×10^{1}	2.3445600597×10^{1}	$1.0419208431\times10^{-1}$
IJAYA	2.2857969385×10^{1}	2.3815013650×10^{1}	$2.7491744948\times10^{-1}$
CLJAYA	2.3008501929×10^{1}	2.3956603034×10^{1}	$6.5386322803\times10^{-1}$
JAYA	2.3094202105×10^{1}	2.3872171530×10^{1}	$7.0916777331\times10^{-1}$
HFPSO	2.3207806520×10^{1}	2.3570163892×10^{1}	$7.7055620394\times10^{-1}$
WOA	2.3612634156×10^{1}	2.5726584934×10^{1}	2.0363936631

4 结论

本文提出一种面向全局优化的混合进化 JAYA 算法 H-JAYA, 在最优和最差个体位置中引入反向学习机制, 增强了算法跳离局部极值区域的能力; 引入正弦余弦算子和差分扰动机制对个体位置进行更新, 不仅增加了种群的多样性, 而且较好平衡了算法的全局探索和局部挖掘能力; 在算法结构上采用奇偶不同的混合进化策略, 使算法的收敛性和精度得到了进一步提高. 理论分析证明该算法的时间复杂度与基本 JAYA 算法相同, 没有降低执行效率. 将 H-JAYA 算法应用于 CEC 2017 复杂函数的全局优化求解中, 其 50 次运行得到的寻优结果最佳值、平均值和方差明显优于其他 5 种性能优越的代表性对比算法; 而 Wilcoxon 秩和检验则验证了 H-JAYA 算法与各对比算法之间的优势差异具有显著性. 最后, 将该算法成功运用于 6 个工程设计优化问题, 取得了很好的优化效果. 上述研究表明, H-JAYA 算法可以获得更好的全局搜索和局部搜索能力, 算法的寻优能力具有明显的优越性, 求解全局复杂函数和工程设计约束优化问题有着显著的可行性和有效性. 在后续的研究中, 考虑将改进的 JAYA 算法应用到更多实际问题求解中, 以进一步验证算法性能, 完善算法的改进机制, 拓宽和提高算法的应用领域与能力.

参 考 文 献

[1] Kennedy J, Eberhart R. Particle swarm optimization // *Proceedings of ICNN'95 - International Conference on Neural Networks*. Perth,

1995: 1942

[2] Coello C A C, Pulido G T, Lechuga M S. Handling multiple objectives with particle swarm optimization. *IEEE Trans Evol Comput*, 2004, 8(3): 256

[3] Mirjalili S. SCA: A sine cosine algorithm for solving optimization problems. *Knowl Based Syst*, 2016, 96: 120

[4] Liu X J, Wang L G. A sine cosine algorithm based on differential evolution. *Chin J Eng*, 2020, 42(12): 1674
(刘小娟, 王联国. 一种基于差分进化的正弦余弦算法. 工程科学学报, 2020, 42(12): 1674)

[5] Mirjalili S, Lewis A. The whale optimization algorithm. *Adv Eng Softw*, 2016, 95: 51

[6] Azizi M. Atomic orbital search: A novel metaheuristic algorithm. *Appl Math Model*, 2021, 93: 657

[7] Kar D, Ghosh M, Guha R, et al. Fuzzy mutation embedded hybrids of gravitational search and Particle Swarm Optimization methods for engineering design problems. *Eng Appl Artif Intell*, 2020, 95: 103847

[8] Liu S Y, Jin A Z. A co-evolutionary teaching-learning-based optimization algorithm for constrained optimization problems. *Acta Autom Sin*, 2018, 44(9): 1690
(刘三阳, 靳安钊. 求解约束优化问题的协同进化教与学优化算法. 自动化学报, 2018, 44(9): 1690)

[9] Banaie-Dezfouli M, Nadimi-Shahraki M H, Beheshti Z. R-GWO: Representative-based grey wolf optimizer for solving engineering problems. *Appl Soft Comput*, 2021, 106: 107328

[10] Xiao Z Y, Liu S. Study on elite opposition-based golden-sine whale optimization algorithm and its application of project optimization. *Acta Electron Sin*, 2019, 47(10): 2177
(肖子雅, 刘升. 精英反向黄金正弦鲸鱼算法及其工程优化研究. 电子学报, 2019, 47(10): 2177)

[11] Nadimi-Shahraki M H, Taghian S, Mirjalili S. An improved grey wolf optimizer for solving engineering problems. *Expert Syst Appl*, 2021, 166: 113917

[12] Wang Y H, Gao L. Improvement of crow search algorithm and its application in engineering constrained optimization problems. *Comput Integr Manuf Syst*, 2021, 27(7): 1871
(汪逸晖, 高亮. 乌鸦搜索算法的改进及其在工程约束优化问题中的应用. 计算机集成制造系统, 2021, 27(7): 1871)

[13] Rao R. Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *Int J Industrial Eng Comput*, 2016, 7(1): 19

[14] Yu K J, Liang J J, Qu B Y, et al. Parameters identification of photovoltaic models using an improved JAYA optimization algorithm. *Energy Convers Manag*, 2017, 150: 742

[15] Kang F, Li J J, Dai J H. Prediction of long-term temperature effect in structural health monitoring of concrete dams using support vector machines with Jaya optimizer and salp swarm algorithms. *Adv Eng Softw*, 2019, 131: 60

[16] Ingle K K, Jatoth D R K. An efficient JAYA algorithm with lévy flight for non-linear channel equalization. *Expert Syst Appl*, 2020, 145: 112970

[17] Iacca G, Junior V C S, Melo V V. An improved Jaya optimization algorithm with Lévy flight. *Expert Syst Appl*, 2021, 165: 113902

[18] Zhao F Q, Ma R, Wang L. A self-learning discrete jaya algorithm for multiobjective energy-efficient distributed no-idle flow-shop scheduling problem in heterogeneous factory system. *IEEE Trans Cybern*, 6181, PP(99): 1

[19] Kang F, Wu Y R, Li J J, et al. Dynamic parameter inverse analysis of concrete dams based on Jaya algorithm with Gaussian processes surrogate model. *Adv Eng Inform*, 2021, 49: 101348

[20] Zhang Y Y, Chi A N, Mirjalili S. Enhanced Jaya algorithm: A simple but efficient optimization method for constrained engineering design problems. *Knowl Based Syst*, 2021, 233: 107555

[21] Nayak D R, Zhang Y D, Das D S, et al. MJaya-ELM: A Jaya algorithm with mutation and extreme learning machine based approach for sensorineural hearing loss detection. *Appl Soft Comput*, 2019, 83: 105626

[22] Zhang Y Y, Ma M D, Jin Z G. Comprehensive learning Jaya algorithm for parameter extraction of photovoltaic models. *Energy*, 2020, 211: 118644

[23] Zhang Y Y, Jin Z G. Comprehensive learning Jaya algorithm for engineering design optimization problems. *J Intell Manuf*, 2021: 1

[24] Liu J S, Ma Y X, Li Y. Improved butterfly algorithm for multi-dimensional complex function optimization problem. *Acta Electron Sin*, 2021, 49(6): 1068
(刘景森, 马义想, 李煜. 改进蝴蝶算法求解多维复杂函数优化问题. 电子学报, 2021, 49(6): 1068)

[25] Liu J S, Mao Y N, Liu X Z, et al. A dynamic adaptive firefly algorithm with globally orientation. *Math Comput Simul*, 2020, 174: 76

[26] Wu G, Mallipeddi R, Suganthan P N. Problem definitions and evaluation criteria for the CEC 2017 competition on constrained real-parameter optimization [EB/OL]. *Technical Repore Online* (2017-09) [2021-10-27]. https://www.researchgate.net/profile/Guohua-Wu-5/publication/317228117_Problem_Definitions_and_Evaluation_Criteria_for_the_CEC_2017_Competition_and_Special_Session_on_Constrained_Single_Objective_Real-Parameter_Optimization/links/5982cdbaa6fdcc8b56f59104/Problem-Definitions-and-Evaluation-Criteria-for-the-CEC-2017-Competition-and-Special-Session-on-Constrained-Single-Objective-Real-Parameter-Optimization.pdf

[27] Aydilek İ B. A hybrid firefly and particle swarm optimization

algorithm for computationally expensive numerical problems. *Appl Soft Comput*, 2018, 66: 232

[28] Derrac J, García S, Molina D, et al. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm Evol Comput*, 2011, 1(1): 3

[29] Kumar A, Wu G H, Ali M Z, et al. A test-suite of non-convex constrained optimization problems from the real-world and some baseline results. *Swarm Evol Comput*, 2020, 56: 100693

[30] Li Y, Zhao Y R, Liu J S. Dimension by dimension dynamic sine cosine algorithm for global optimization problems. *Appl Soft Comput*, 2021, 98: 106933

[31] Salgotra R, Singh U, Singh S, et al. Self-adaptive salp swarm algorithm for engineering optimization problems. *Appl Math Model*, 2021, 89: 188

[32] Liu J S, Ma Y X, Li Y. Improved whale algorithm for solving engineering design optimization problems. *Comput Integr Manuf Syst*, 2021, 27(7): 1884
(刘景森, 马义想, 李煜. 改进鲸鱼算法求解工程设计优化问题. 计算机集成制造系统, 2021, 27(7): 1884)