文章编号：2095-2163(2023)06-0038-12 中图分类号：TP301 文献标志码：A

基于扰动因子和贪心策略的白骨顶优化算法

李雪利[1]，杜逆索[2]，欧阳智[1,2]，朱 鹏[3]
（1 贵州大学 数学与统计学院，贵阳 550025；2 贵州大学 贵州省大数据产业发展应用研究院，贵阳 550025；
3 贵州大学 计算机科学技术学院，贵阳 550025）

摘 要：针对白骨顶优化算法存在收敛精度低、速度慢且鲁棒性较差等问题，提出了基于扰动因子和贪心策略的白骨顶优化算法(PGCOOT)。首先，该算法在种群初始化阶段采用了精英反向学习策略，提高白骨顶种群多样性；其次通过贪心策略在白骨顶移动中选择最优移动方式，以降低搜索的盲目性和低效性；然后引入扰动因子平衡算法的局部和全局探索能力。最后，将改进算法与多个群智能算法在 8 个经典测试函数下进行仿真实验，其结果表明，优化后的白骨顶优化算法拥有更优的寻优精度、速度以及更好的稳定性，Wilcoxon 秩和检验证明了 PGCOOT 算法与其他算法相较存在显著性优异。

关键词：白骨顶优化算法；精英反向学习；贪心策略；扰动因子；函数优化

A COOT optimization algorithm based on perturbation factor and greedy strategy

LI Xueli[1], DU Nisuo[2], OUYANG Zhi[1,2], ZHU Peng[3]
(1 College of Mathematics and Statistics, Guizhou University, Guiyang 550025, China;
2 Guizhou Big Data Academy, Guizhou University, Guiyang 550025, China;
3 College of Computer Science and Technology, Guizhou University, Guiyang 550025, China)

【Abstract】 For the low convergence accuracy and slow convergence speed of COOT optimization algorithm, a coot optimization algorithm based on perturbation factor and greedy strategy (PGCOOT) is proposed. Firstly, the algorithm adopts elite reverse learning strategy to improve the diversity of coot population during the initial stage. Secondly, the greedy strategy is adopted to select the best movement mode in order to reduce the search blindness and inefficiency. Then, the proposed algorithm uses perturbation factor to balance local exploration and global exploration. Finally, the improved algorithm and other swarm intelligence algorithms are simulated under eight classical test functions. The results show that the optimized COOT optimization algorithm has better search accuracy, speed and stability, and has more competitive advantages compared with other algorithms. Wilcoxon rank sum test proves that PGCOOT algorithm is significantly superior to other algorithms.

【Key words】 coot optimization algorithm; elite reverse learning; greedy strategy; perturbation factor; function optimization

0 引 言

群智能优化算法通过模拟生物群体行为所构建的优化模型，帮助解决了实际复杂项目和工程的优化问题。如蚁狮算法[1]、灰狼优化算法[2]、蝗虫优化算法[3]、正余弦算法[4]、多元宇宙优化算法[5]等，这些算法主要研究在复杂的多维解空间里寻得全局最优解。

白骨顶优化算法(Coot Optimization Algorithm, COOT)是 Naruei 等[6]在 2021 年提出的一种新型智能优化算法。该算法通过模拟白骨顶在水中的不同运动方式，并将这些运动方式抽象于数学建模以实现全局寻优。因该算法结构简单、控制参数少、求解精度高等优点，并在测试中试验结果良好，故具有广泛的应用前景。虽然 COOT 算法在优化方面具有一定优势，但寻优过程中依然存在一些不足。如：白骨顶的种群随机初始化和参数随机化会影响算法的寻优性能，白骨顶个体的位置更新机制无法很好地平衡局部探索与全局开发的关系，以及收敛精度欠缺等问题。

基金项目：贵州省科学技术厅重大科技计划项目(黔科合重大专项字[2018]3002)；贵州大学培育项目(贵大培育[2020]41)。

作者简介：李雪利(1996-)，女，硕士研究生，主要研究方向：智能算法、仿真模拟；杜逆索(1986-)，男，博士，讲师，CCF 专业会员，主要研究方向：仿真模拟、数据分析、人工智能等；欧阳智(1987-)，男，博士，讲师，主要研究方向：知识管理、机器学习；朱 鹏(1996-)，男，硕士研究生，主要研究方向：智能计算、机器学习。

通讯作者：欧阳智 Email：zouyang@gzu.edu.cn

收稿日期：2022-05-30

目前，由于群智能算法都存在与 COOT 相似的问题，促使研究者对此开展了相关研究。为了实现更好的寻优性质，研究人员对群智能优化算法的参数进行调整，将群智能优化算法与反向学习[7]、莱维飞行[8]等策略相结合。例如，魏伟一等[9]将精英反向学习与萤火虫算法相结合，并使用了自适应步长平衡算法的探索能力，使算法具有更快的收敛速度，更准确的收敛精度。肖子雅等[10]将鲸鱼优化算法与精英反向学习和黄金分割数相结合，通过平衡算法的局部搜索和全局优化能力，来增强算法的稳定性，得到更优的搜索结果。为了缓解算法容易陷入局部最优，无法寻得全局最优的问题，杨文珍等[11]将扰动机制和强化莱维飞行融入蝗虫优化算法，增强了算法的全局搜索能力。唐海波等[12]在算法中加入模拟退火与贪心策略，有效的提高了算法性能。

综上所述，现已有许多研究针对各种群体智能优化算法的不足之处提出了不同的策略方法，并取得了一定程度的改进效果，而针对白骨顶优化算法不足的研究还较少。本文针对 COOT 算法存在收敛速度慢、收敛精度欠缺的问题，提出了基于扰动因子和贪心策略改进的白骨顶优化算法（PGCOOT）：一方面在白骨顶种群初始化阶段结合精英反向学习进行初始化，另一方面在白骨顶个体移动中融入扰动因子和贪心策略来改变移动机制。因此，本文所提算法不仅改善了白骨顶种群的位置分布，而且能够更好的协调算法的局部搜索和全局优化能力，从而提高算法的收敛性能。

1　白骨顶优化算法

白骨顶优化算法（COOT）是在研究白骨顶在水中两种运动方式的基础上提出的一种新型智能优化算法。在 COOT 优化算法中，整个白骨顶种群由领导者（leader）和普通白骨顶鸟（coot）组成。领导者是指种群前面的少数指引种群走向目标（食物）的白骨顶。白骨顶有许多不同的集体行为，COOT 算法的目标是模拟白骨顶在水面上的集体运动。算法中模拟了白骨顶在水面上 4 种不同的移动，分别是：个体的随机移动、链式移动、根据群体领导者们调整位置以及 leader 带领种群走向最佳区域（领导者运动）。

1.1　初始化阶段

使用公式（1）在空间中随机生成 Coot 种群：

$$CootPos(i) = rand(1,d).*(ub - lb) + lb \quad (1)$$

其中，$CootPos(i)$ 为 $coot_i$ 的位置；d 为一系列的变量或问题维度；lb 为搜索空间的下限；ub 为搜索空间的上限；.* 为点乘运算。

在生成初始群体并确定每个主体的位置后，随机选取 NL 个白骨顶个体作为 leader，最后通过目标函数计算所有白骨顶的适应度。

1.2　*coot* 的移动

在整个移动过程中，*coot* 每次都从个体的随机移动、链式移动、根据群体领导者们调整位置这 3 种移动方式中随机选取一种作为移动方式。

1.2.1　个体随机移动

个体的随机移动能对搜索空间的不同部分进行探索。如果算法陷入局部最优，采用随机移动的方式能够缓解此类问题。为了实现这种移动，首先随机选取一个位置，再将 *coot* 移向这个位置，之后根据公式（2）来更新 *coot* 的位置。

$$CootPos(i) = CootPos(i) + A \times R2 \times (Q - CootPos(i)) \quad (2)$$

其中，$R2$ 为区间［0，1］中的随机数；$CootPos(i)$ 表示 $coot_i$ 的位置；Q 为随机选取的位置；A 值可根据公式（3）计算得出。

$$A = 1 - L \times \left(\frac{1}{Iter}\right) \quad (3)$$

其中，L 表示当前迭代次数，$Iter$ 表示最大迭代次数。

1.2.2　链式移动

实现链移动时，首先计算两个白骨顶位置点之间的距离矢量，然后向其中一个白骨顶的方向移动另一个白骨顶，移动距离为矢量的一半。新位置的计算方法为公式（4）。

$$CootPos(i) = 0.5 \times (CootPos(i-1) + CootPos(i)) \quad (4)$$

其中，$CootPos(i)$ 表示 $coot_i$ 的位置，$CootPos(i-1)$ 表示 $coot_{i-1}$ 的位置。

1.2.3　根据群体领导者们调整位置

当前面的几个 leader 领导族群走向最佳区域时，其余的 coot 则根据 leader 的位置调整自己的位置。由于种群中不止一个 leader，因此 coot 在移动过程中根据公式（5）的机制来选择某一个 leader 作为参考位置。

$$K = 1 + (i \ \mathrm{MOD} \ NL) \quad (5)$$

其中，i 为当前 coot 的索引号；NL 为初始时设置的 leader 数量；K 为 leader 的索引号；MOD 表示取余运算。

在选定某个 leader 作为参考位置后,coot 则朝着参考位置移动。移动后的新位置由公式(6)计算得出。

$$CootPos(i) = LeaderPos(k) + 2 \times R1 \times \cos(2R\pi) \times (LeaderPos(k) - CootPos(i)) \quad (6)$$

其中,$CootPos(i)$ 表示 $coot_i$ 当前位置;$LeaderPos(k)$ 为选定的 $leader_k$ 的位置;$R1$ 为区间[0,1]中的随机数;π 为圆周率;R 为区间[-1,1]中的随机数。

1.3 leader 的移动

COOT 优化寻优过程中,领导者 leader 的移动至关重要,leader 需要朝着目标(最优点)更新位置。有时领导者需要离开当前的最佳位置去寻找更好的位置。更新领导者位置的计算如公式(7)。这个公式展现了接近和远离当前最佳位置的策略。

$$LeaderPos(i) = \begin{cases} B \times R3 \times \cos(2R\pi) \times (gBest - LeaderPos(i)) + gBest, R4 < 0.5 \\ B \times R3 \times \cos(2R\pi) \times (gBest - LeaderPos(i)) - gBest, R4 \geq 0.5 \end{cases} \quad (7)$$

其中,$LeaderPos(i)$ 为 $leader_i$ 的位置;$gBest$ 为迄今为止发现的最佳位置;$R3$、$R4$ 为区间[0,1]中的随机数;π 为圆周率;R 为区间[-1,1]中的随机数;B 可根据公式(8)计算所得。

$$B = 2 - L \times (\frac{1}{Iter}) \quad (8)$$

其中,L 表示当前迭代,$Iter$ 表示最大迭代。

随机选择接近或远离当前最佳位置,并以不同的半径在迄今最佳的位置周围进行搜索,意味着白骨顶优化算法在接近最优点的阶段也在进行探索,以便算法远离陷入局部最优的困境。

2 白骨顶优化算法的改进

2.1 精英反向学习

一般而言,算法的收敛精度和速度均与初始种群中个体的位置分布有关。与其它算法类似,COOT 算法在使用随机方式对 coot 种群位置进行初始化后,才进行优化迭代。但随机初始化会导致 COOT 算法的可行解范围较大,造成算法收敛时间较长,会降低算法的搜索能力,影响算法的搜索结果等问题。

一般而言,精英反向学习策略可以对种群初始化阶段进行优化来提升算法的收敛性能。首先,在种群初始化阶段产生相应的反向种群。生成反向种群的方式表示如公式(9)所示:

$$\bar{CootPos}(i) = k(ub + lb) - CootPos(i) \quad (9)$$

其中,$CootPos(i)$ 为当前个体 $coot_i$ 的位置信息;$\bar{CootPos}(i)$ 为相应的反向种群个体 $\bar{coot}_i$ 的位置信息;lb 为搜索空间的下限;ub 为搜索空间的上限;k 是产生于[0,1]之间的随机数。

其次,根据公式(10),在得到反向种群位置信息后,依序比较初始种群和反向种群的个体适应度,最后将同一位置的适应度更优的个体放入初始种群的对应位置中。

$$CootPos(i) = \begin{cases} \bar{CootPos}(i), & \bar{f}_i < f_i \\ CootPos(i), & \bar{f}_i \geq f_i \end{cases} \quad (10)$$

其中,$CootPos(i)$ 为当前个体 $coot_i$ 的位置信息;$\bar{CootPos}(i)$ 为相应的反向种群个体 $\bar{coot}_i$ 的位置信息;$\bar{f}_i$ 为反向种群中个体 $\bar{coot}_i$ 的适应度值;f_i 为随机初始化种群中个体 $coot_i$ 的适应度值。

2.2 贪心策略移动机制

COOT 优化算法寻优过程中,由于 coot 的移动方式是从上述 3 种移动方式中随机选取的。这种个体移动选择方式容易出现收敛速度慢、收敛精度低等问题。因此,考虑在 coot 的移动方式选取过程中加入贪心策略,来减少移动过程的盲目性。将贪心策略加入到 coot 的移动中表现为:先通过比较上次迭代移动后的位置是否有进步,再决定本次迭代过程中移动方式的选取[14]。当上次移动后的位置有进步时,本次移动过程选取与上次相同的移动方式;当上次移动后的位置表现更差时,本次移动过程选择与上次不同的移动方式,以求更好的移动表现。其基本规则如公式(11)所示:

$$Move(i)_l = \begin{cases} Move(i)_{l-1}, & f(i)_{l-1} < f(i)_{l-2} \\ random[\sim Move(i)_{l-1}], & f(i)_{l-1} \geq f(i)_{l-2} \end{cases} \quad (11)$$

其中,$Move(i)_l$ 为 $coot_i$ 在第 l 次移动时所选择的移动方式;~表示非运算;$random$ 表示随机选取;$f(i)_l$ 为 $coot_i$ 的第 l 次移动的适应度值。

2.3 带扰动因子的领导者

在整个寻优过程中,领导者的位置关乎整个种群的动向[15]。在 leader 的移动过程中添加扰动因子,能够改变 leader 的探索范围,协调局部搜索和全局优化能力,从而得到更优的搜索结果[13]。当 leader 朝着接近最优点移动时,勘探范围由大到小过渡,增强局部搜索能力[16-17];而 leader 朝着远离

最优点移动时，搜索范围由小到大变化，增强了全局探索能力，能有效的避免陷入局部最优的问题。扰动因子 q 定义如公式(12)所示[11,14-15]：

$$q = \left\{\cos\left[\left(1 - \frac{L}{Iter}\right)^2 \times \pi\right] + \alpha\right\} \times \beta \quad (12)$$

其中，L 是当前迭代；$Iter$ 是最大迭代；π 为圆周率。

通过对 α 和 β 在[0,20]区间内依次取值进行组合实验，实验结果表明，当 $\alpha = 4$、$\beta = 5$ 时，全局勘探能力与局部探索能力能够得到较好的平衡，此时寻优效果最好。改进后的领导者移动公式如公式(13)所示：

$$LeaderPos(i) = \begin{cases} (B \times R3 \times \cos(2R\pi) \times (gBest - LeaderPos(i)) + gBest) \times \dfrac{1}{q}, R4 < 0.5 \\ (B \times R3 \times \cos(2R\pi) \times (gBest - LeaderPos(i)) - gBest) \times q, R4 \geqslant 0.5 \end{cases} \quad (13)$$

2.4　PGCOOT 算法描述

综上所述，本文将精英反向学习、贪心策略和扰动因子与 COOT 算法融合得到 PGCOOT 算法。该算法能更好的协调局部搜索和全局优化，具有更优的搜索性能，从而提升算法的收敛效果。PGCOOT 算法实现流程如图 1 所示。算法步骤如下：

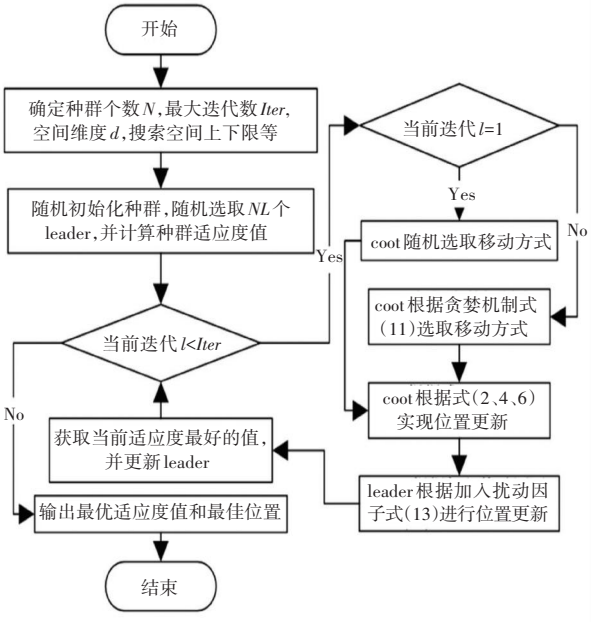


图 1　PGCOOT 算法流程图

Fig. 1　PGCOOT algorithm flow diagram

Step 1　确定白骨顶种群个数 N，最大迭代次数 $Iter$，空间维度 d，搜索空间上下限等参数；

Step 2　随机产生种群个体，初始化种群，随机选取 NL 个 leader，并计算种群适应度值；

Step 3　采用精英反向学习产生反向种群，再选取每个序号下最好的个体作为种群个体；

Step 4　进入主循环，While 当前迭代次数 $l < Iter$；

Step 5　coot 选取移动方式。当 $l = 1$ 时，coot 随机选取移动方式；当 $l > 1$ 时，coot 根据贪婪策略改进的机制(式(11))选取移动方式；

Step 6　coot 根据已选取移动方式，依据式(2)~式(6)实现位置信息更新；

Step 7　leader 根据加入扰动因子的领导者运动式(13)进行位置信息更新；

Step 8　计算种群适应度，获取当前适应度最好的值，并更新 leader；

Step 9　判断是否达到停止条件，满足则迭代过程结束；否则返回 Step 4 继续迭代；

Step 10　输出最优适应度值和最佳位置。

2.5　PGCOOT 算法复杂度分析

在标准 COOT 算法中，假设种群规模为 N，求解空间维数为 d，标准 COOT 进行参数初始化的复杂度为 $O(1)$，则个体适应度为 $O(N)$，种群复杂度为 $O(N \times d)$，此时标准 COOT 算法的整体复杂度如公式(14)所示：

$$O(\text{COOT}) = O(1) + O(N) + O(N \times d) = O(N \times d) \quad (14)$$

PGCOOT 算法在初始化阶段使用精英反向学习策略，替换随机初始化后复杂度为 $O(N \times d)$；在个体适应度计算阶段使用贪心策略后，复杂度为 $O(N \times d)$，干扰因子复杂度为 $O(N \times d)$；算法整体复杂度如公式(15) 所示：

$$O(\text{PGCOOT}) = O(N \times d) + O(N) + O(N \times d) + O(N \times d) = O(N \times d) \quad (15)$$

显然 $O(\text{COOT}) = O(\text{PGCOOT})$。可见，相比于标准 COOT 算法，本文所提算法与标准 COOT 算法时间复杂度一致。

3　仿真实验结果与分析

为证明 PGCOOT 算法加入不同改进策略的有效性，并且具有更好的性能，将其算法与其他算法进行仿真实验对比分析。实验从以下 3 方面进行：

(1)将 PGCOOT 算法与贪心策略下的改进算法 COOT1、精英反向学习初始化与加入扰动因子的领导者运动改进后的算法 COOT2、标准 COOT 算法进行实验对比，验证不同策略的有效性。同时，为了验证

PGCOOT 的时间复杂度,与 COOT 进行了比较分析。

(2)将 PGCOOT 算法与蚁狮算法(ALO)、灰狼优化算法(GWO)、蝗虫优化算法(GOA)、正余弦算法(SCA)、多元宇宙优化算法(MVO) 和原始的 COOT 算法在 d = 30 条件下做实验比较,通过实验数据论证 PGCOOT 算法的可行性、有效性和优越性。再根据寻优能力的表现,选取前三名算法在 d = 100 时进行对比实验。

(3)将 PGCOOT 算法与蚁狮算法(ALO)、灰狼优化算法(GWO)、蝗虫优化算法(GOA)、正余弦算法(SCA)、多元宇宙优化算法(MVO) 和原始的 COOT 算法进行 Wilcoxon 秩和检验,以此验证 PGCOOT 算法相较于其他算法具有显著性差异。

为了检验本文所提 PGCOOT 算法的寻优能力,实验使用 8 种标准测试函数来对算法进行不同寻优特征上的测试。标准测试函数见表 1。其中 F_1、F_2、F_3、F_4 为单峰函数, F_5、F_6、F_7、F_8 是多峰函数,存在多个局部极值,求解难度较高。

实验环境:本文所有实验均在 Intel ® Core™ i5-7500 CPU@3.40 GHz,RAM 8.0 GB,运行环境为 64 位 Window10 操作系统,MATLAB 2018b 软件上进行。为了与其它算法进行公平比较,将种群规模都设为 N = 30,设置领导者数量占比为 0.1,最大迭代次数 $Iter$ 为 500、α 为 4, β 为 5。每个测试函数都进行了 30 次独立实验,以排除随机性对实验结果的影响,并记录最优值(Best)、平均值(Mean)与标准差(Std),来整体评价各算法的性能。其中,平均值用于表示算法的平均寻优能力,最优值用于表示算法的寻优极限,标准差表示算法的鲁棒性。

表 1 标准测试函数

Tab. 1 Standard test function

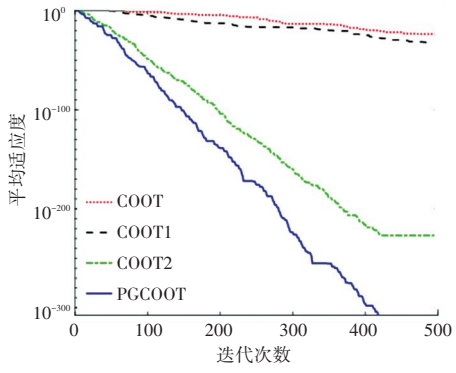
编号	函数表达式	维度	范围	最优值
F_1	$f(x) = \sum_{i=1}^{n} x_i^2$	30,100	[-100,100]	0
F_2	$f(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	30,100	[-10,10]	0
F_3	$f(x) = \sum_{i=1}^{n} \left(\sum_{j-1}^{i} x_j \right)^2$	30,100	[-100,100]	0
F_4	$f(x) = \max\{\lvert x_i \rvert, 1 \leqslant i \leqslant n\}$	30,100	[-100,100]	0
F_5	$f(x) = \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	30,100	[-500,500]	0
F_6	$f(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$	30,100	[-5.12,5.12]	0
F_7	$f(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$	30,100	[-32,32]	0
F_8	$\frac{1}{4\,000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	30,100	[-600,600]	0

3.1 不同策略比较

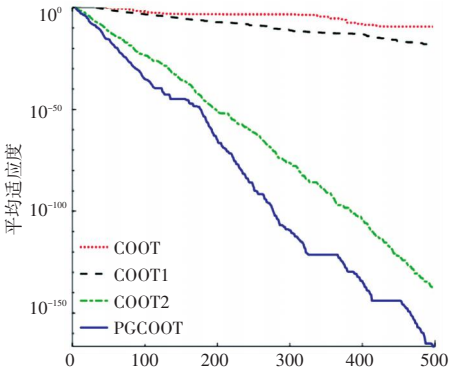
为验证 PGCOOT 算法中各策略的有效性,将原始算法 COOT 与 COOT1、COOT2、PGCOOT 在 d = 30 时,进行实验比较,从而验证各策略的有效性。各策略算法独立运行 30 次后的实验结果见表 2,可视化结果如图 2 所示。从对表 2 和图 2 的分析可知:

(1)对于 F_1 和 F_3, COOT、COOT2 和 COOT1 并未收敛至最优值。COOT1 与原始算法相比,收敛精度略有提升;COOT2 的精度则比原始算法提升了至少 200 个数量级。而融合了 COOT1 和 COOT2 策略的 PGCOOT 算法则在迭代接近 450 次时就收敛至理论最优值,并且 PGCOOT 的收敛平均值 (Mean) 和标准差(Std)均为 0。这表明在 30 次实验中,PGCOOT 能 100%取得理论最优解。

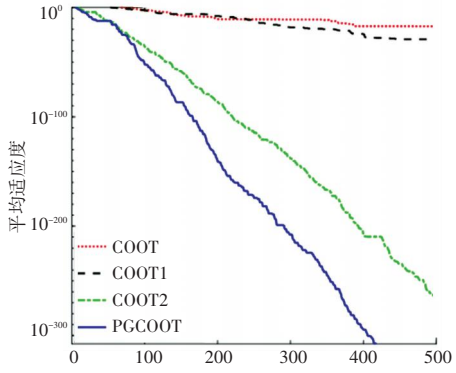
(2)对于 F_2 和 F_4, 各策略算法均未收敛至理论最优值,但 PGCOOT 最接近理论最优值,与原始算法相比提升了 170 个数量级左右,并且 COOT1 和 COOT2 的改进策略结果也优于原始的 COOT。



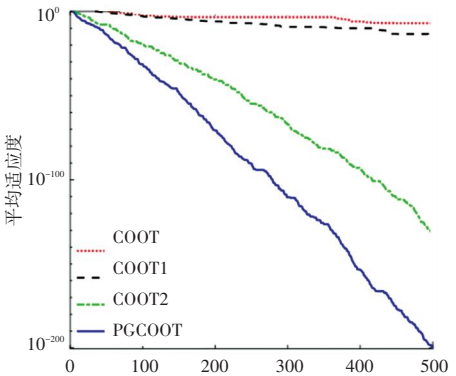
(a) F_1



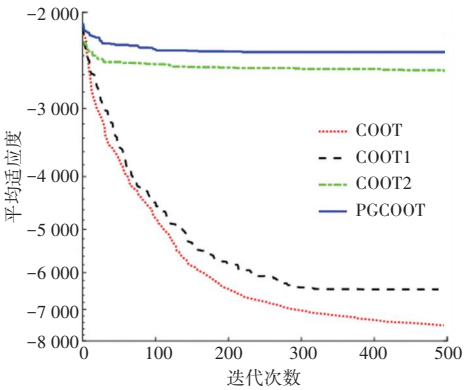
(b) F_2



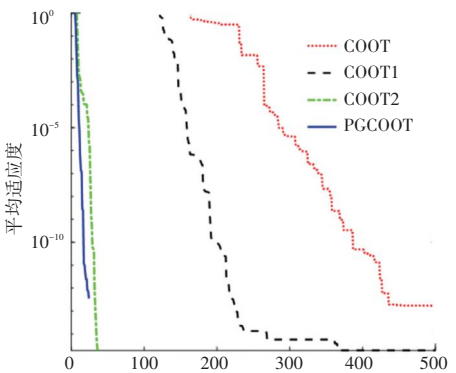
(c) F_3



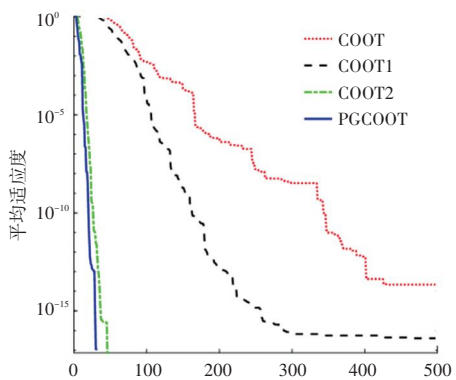
(d) F_4



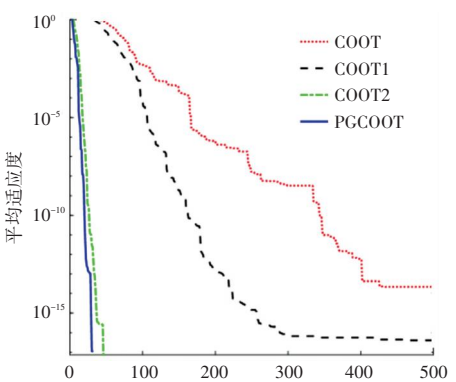
(e) F_5



(f) F_6



(g) F_7



(h) F_8

图 2　各策略算法的平均收敛曲线

Fig. 2　Average convergence curves of several strategy algorithms

表 2 PGCOOT 与 COOT、COOT1、COOT2 结果

Tab. 2 Results of PGCOOT and COOT, COOT1 and COOT2

函数	算法	Mean	Best	Std
F_1	COOT	3.068 7E-24	3.602 3E-50	1.674 8E-23
	COOT1	3.972 1E-34	1.904 1E-54	2.052 7E-33
	COOT2	1.368 2E-227	0	0
	PGCOOT	**0**	**0**	**0**
F_2	COOT	2.624 7E-10	1.719 2E-22	1.373 9E-09
	COOT1	3.328 7E-19	2.703E-26	1.691 4E-18
	COOT2	1.275 1E-139	8.833 4E-173	6.984 2E-139
	PGCOOT	**3.831 5E-167**	**1.045 9E-250**	**0**
F_3	COOT	4.248 9E-18	4.047 1E-52	2.324 2E-17
	COOT1	5.699 8E-30	8.533 6E-57	3.121 9E-29
	COOT2	1.521 8E-264	0	0
	PGCOOT	**0**	**0**	**0**
F_4	COOT	9.879 4E-07	3.597 9E-26	5.411E-06
	COOT1	3.315 1E-17	1.255 2E-28	1.720 1E-16
	COOT2	1.052 8E-136	1.229 9E-169	5.766 2E-136
	PGCOOT	**8.938 3E-173**	**6.610 6E-218**	**0**
F_5	COOT	−7 498.882 3	−6 375.323 2	607.003 4
	COOT1	−6 437.294 1	−3 630.771 5	957.959 72
	COOT2	−2 560.845 6	−1 911.029 1	435.737 3
	PGCOOT	**−2 364.884 9**	**−1 379.522 3**	**434.120 9**
F_6	COOT	1.705 3E-13	0	6.338 6E-13
	COOT1	1.894 8E-15	0	1.037 8E-14
	COOT2	0	0	0
	PGCOOT	**0**	**0**	**0**
F_7	COOT	3.563 9E-11	8.881 8E-16	1.870 2E-10
	COOT1	8.467 3E-15	8.881 8E-16	1.095E-14
	COOT2	8.881 8E-16	8.881 8E-16	0
	PGCOOT	**8.881 8E-16**	**8.881 8E-16**	**0**
F_8	COOT	2.166E-14	0	1.178E-13
	COOT1	4.070 8E-17	0	9.440 1E-17
	COOT2	0	0	0
	PGCOOT	**0**	**0**	**0**

在相同条件下,对于单峰函数 $F_1 \sim F_4$ 的对比分析结果,展现了改进策略 COOT1 和 COOT2 在收敛速度与收敛精度的有效性,而 PGCOOT 算法融合了 COOT1 和 COOT2 的优点,在各函数上都获得了更高的准确性和更快的收敛速度,表明了所提算法的优越性。

在多峰函数时,对于 F_5 来看,各策略算法均未收敛至最优值。收敛平均值(Mean)表明,与 COOT 相比,COOT1 收敛精度相对更高,COOT2 的收敛精度则大幅提升,而融合了 COOT1 和 COOT2 策略的 PGCOOT 收敛精度最高,最接近理论值。通过最优值(Best)和标准差(Std)的比较,表明了 PGCOOT 的寻优极限更高、收敛效果更加稳定。分析 F_6 与 F_8 时的结果,各策略算法均实现最优值(Best)与理论值相等;但收敛平均值(Mean)和标准差(Std)结果则表明了 COOT 和 COOT1 并不能保证 100%取得理论最优解,而 PGCOOT、COOT2 的 30 次运行均能够 100%取得理论最优解。从收敛速度来看,PGCOOT 的收敛速度比 COOT2 更快。

综上,在多峰函数 $F_5 \sim F_8$ 时,融合了 COOT1 和 COOT2 策略的 PGCOOT 在寻优精度、稳定性和收敛速度上都表现出了远优于其他算法的性能。

为了验证 PGCOOT 的时间复杂度,在上述条件下,将 PGCOOT 与 COOT 在相同函数下独立运行 30 次的总时间进行记录,结果见表 3。从表 3 可以看出,PGCOOT 与 COOT 运行 30 次所用时间均在 3 s 左右,不存在较大的差异。说明本文算法在标准 COOT 算法中使用多种策略后,时间复杂度并没有上升,运行效率也未受到影响。

3.2 不同算法比较

本次实验运用 8 个测试函数,在参数设置统一的条件下,测试 PGCOOT 算法的寻优速度与寻优精度。通过最优值(Best)、平均值(Mean)与标准差(Std)3 个指标,评价 PGCOOT 的寻优能力。

本文选取蚁狮算法(ALO)、灰狼优化算法(GWO)、蝗虫优化算法(GOA)、正余弦算法(SCA)、多元宇宙优化算法(MVO)和原始的 COOT 算法等,与改进算法 PGCOOT 在 $d = 30$ 时进行对比实验。实验结果见表 4,收敛曲线如图 3 所示。

由对表 4 和图 3 的分析可知:

(1)对于单峰函数 $F_1 \sim F_4$,PGCOOT 能迅速收敛至最优值或最优值附近,而其余算法收敛精度较低,收敛速度较慢。其中,在 F_1、F_3 上只有本文所提出的 PGCOOT 获得了理论最优值,其余算法均未寻到理论最优值。从平均值与标准差来看,PGCOOT 寻优成功率高达 100%,具有比其他算法更好的稳定性。从对应的收敛曲线中可以看到,虽然 GWO 与 COOT 有着不错的效果,但是 PGCOOT 在收敛速度上占有明显优势。对于 F_2、F_4 而言,虽然所有算法均未寻到理论最优值,但 PGCOOT 的结果最接近理论值;从平均值与标准差来看,PGCOOT 也具有更好的鲁棒性和平均寻优能力。

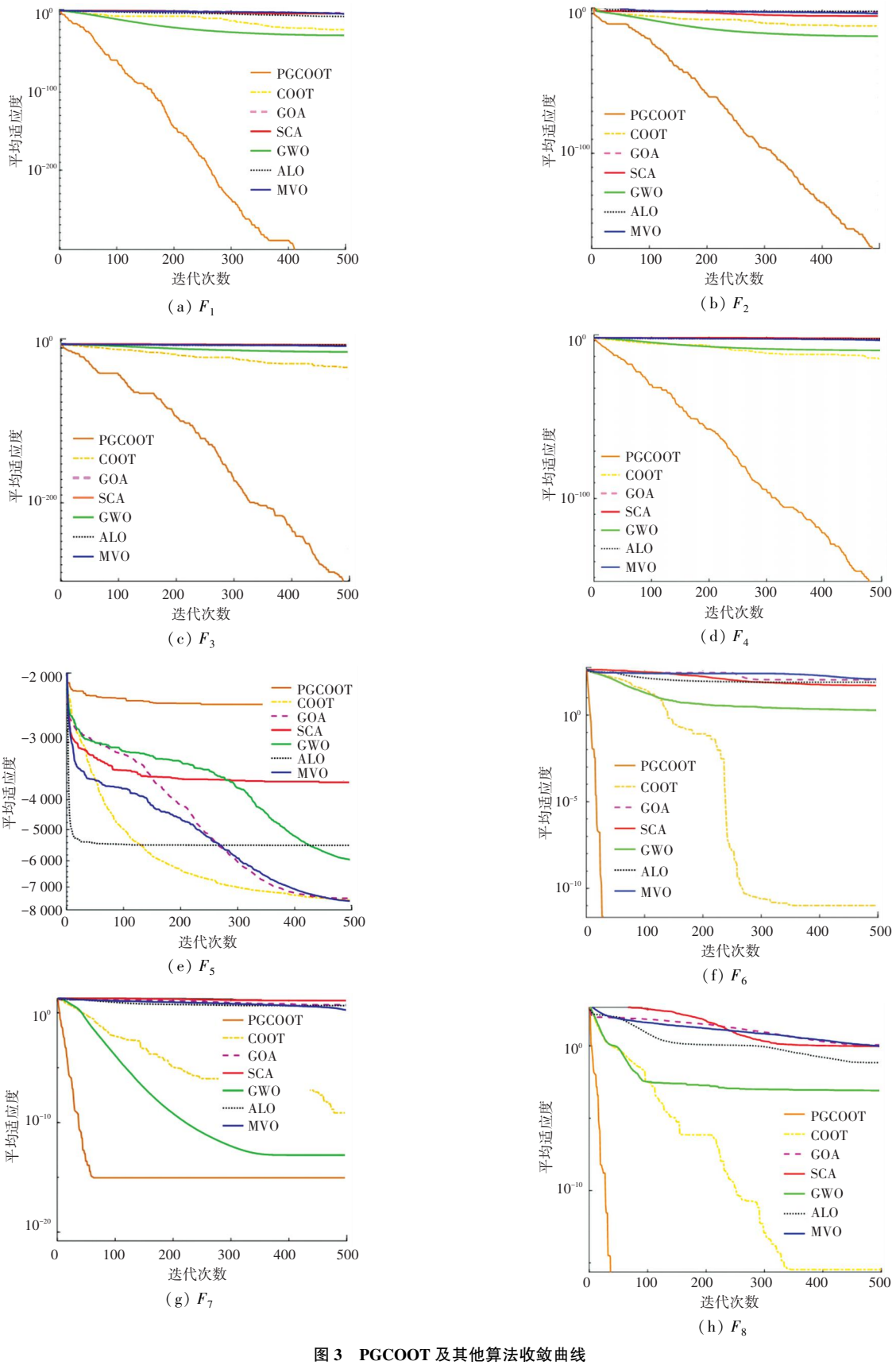


图 3　PGCOOT 及其他算法收敛曲线

Fig. 3　Convergence curves of PGCOOT and other algorithms

表 3 PGCOOT 与 COOT 独立运行 30 次时间(s)比较

Tab. 3 Comparison between PGCOOT and COOT when running independently for 30 times

	PGCOOT	COOT
F_1	3.015 0	3.018 0
F_2	3.175 0	3.009 0
F_3	3.102 0	2.990 0
F_4	2.977 0	3.038 0
F_5	3.059 0	3.020 0
F_6	3.077 0	3.038 0
F_7	2.929 0	3.160 0
F_8	2.996 0	3.111 0

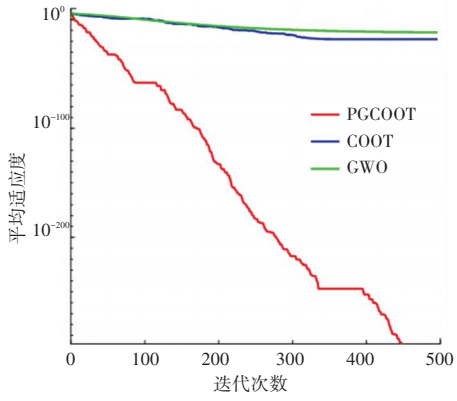
表 4 PGCOOT 及其他算法结果

Tab. 4 Results of PGCOOT and other algorithms

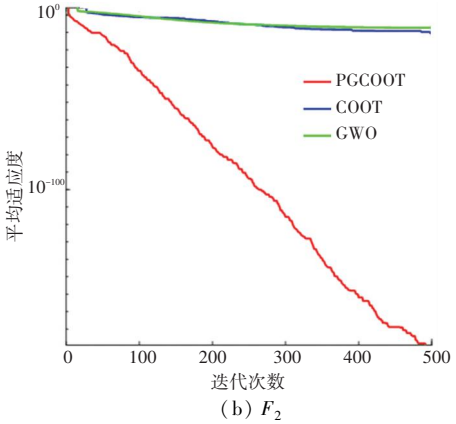
函数	算法	Mean	Best	Std
F_1	**PGCOOT**	**0**	**0**	**0**
	COOT	9.013 5E-25	5.594 4E-44	4.705 8E-24
	GOA	32.020 2	7.513 89	20.337
	GWO	1.518 8E-27	2.460 8E-29	2.673 5E-27
	ALO	0.001 513 6	0.000 129 75	0.001 670 8
	MVO	1.345 8	0.801 34	0.318 73
	SCA	10.359 1	0.009 467 66	32.276 2
F_2	**PGCOOT**	**3.421 4E-189**	**1.033 7E-216**	**0**
	COOT	8.373 7E-13	2.894E-24	4.465 9E-12
	GOA	15.634 3	3.338 97	16.154 7
	GWO	9.412 3E-17	2.125 8E-17	5.321 6E-17
	ALO	59.694	2.456 02	51.234 9
	MVO	0.882 45	0.555 39	0.358 08
	SCA	0.020 075	0.000 676 68	0.032 385
F_3	**PGCOOT**	**0**	**0**	**0**
	COOT	4.067 7E-19	1.709E-52	2.184 3E-18
	GOA	3 200.775 4	898.437 7	1 338.631 2
	GWO	8.158 6E-06	5.316 5E-08	1.470 1E-05
	ALO	4 908.344 8	933.929 93	2 849.760 9
	MVO	235.377 8	111.281 2	84.887 9
	SCA	8 864.576 1	1 798.954 6	5 580.805 5
F_4	**PGCOOT**	**9.662 9E-168**	**4.770 5E-211**	**0**
	COOT	1.605 1E-08	6.943 2E-27	8.761 5E-08
	GOA	13.877 7	7.640 76	4.208 31
	GWO	5.067 9E-07	5.680 1E-08	5.714 9E-07
	ALO	18.325 1	8.276 29	5.100 66
	MVO	2.094 1	0.883 26	0.846 62
	SCA	36.755 5	15.783 1	11.544 5
F_5	**PGCOOT**	**-2 447.532 3**	**-1 582.487 1**	**498.831 82**
	COOT	-7 567.433 4	-5 423.263 2	852.368 05
	GOA	-7 480.268 2	-5 748.576 5	955.575 65
	GWO	-5 960.459 9	-5 120.864 7	507.876 59
	ALO	-5 482.118	-5 417.674 8	100.469 65
	MVO	-7 596.232 3	-5 709.944 4	742.229 95
	SCA	-3 791.965 7	-3 293.497 9	305.864 86
F_6	**PGCOOT**	**0**	**0**	**0**
	COOT	9.492 9E-12	0	3.503E-11
	GOA	106.522 7	51.236 83	37.414 19
	GWO	1.908 3	0	3.263 4
	ALO	78.204 4	40.794	21.547 3
	MVO	116.867 7	48.539 99	30.429 09
	SCA	51.086 3	0.447 146	45.264 3
F_7	**PGCOOT**	**8.881 8E-16**	**8.881 8E-16**	**0**
	COOT	5.596 7E-06	8.881 8E-16	3.038 7E-05
	GOA	5.856 8	3.329 1	1.466 4
	GWO	1.011 9E-13	7.549 5E-14	1.532 2E-14
	ALO	6.825 8	1.341 4	3.852
	MVO	3.049 2	0.521 33	4.537 3
	SCA	15.481 8	0.061 726 5	8.288 31
F_8	**PGCOOT**	**0**	**0**	**0**
	COOT	3.515 7E-16	0	1.229 8E-15
	GOA	1.159 7	0.889 36	0.122 46
	GWO	0.000 819 24	0	0.003 121 6
	ALO	0.069 363	0.027 032	0.035 533
	MVO	0.863 65	0.632 15	0.084 46
	SCA	0.945 64	0.059 149	0.467 64

(2)从多峰函数 F_5 ~ F_8 的结果可见,在 F_5 函数上,虽然所有算法均未寻到理论最优值,但从接近理论最小值的程度来看,PGCOOT 的结果最接近于理论最优值。在 F_6 和 F_8 函数上,PGCOOT、GWO 与 COOT 均跳出局部最优,寻到理论最优值,但是从平均值与标准差来看,PGCOOT 具有更好的稳定性,能够 100%寻得理论最优值。对于 F_7, 虽然 PGCOOT 与其他算法均未寻到理论最优值,但 PGCOOT 获得的最优值 8.881 8E-16 是所有算法中最接近理论值的结果之一。并且从平均值与标准差的结果可以看到,PGCOOT 的稳定性最好。这表明了在 F_7 函数下,PGCOOT 相对于其它算法仍然最具竞争力。

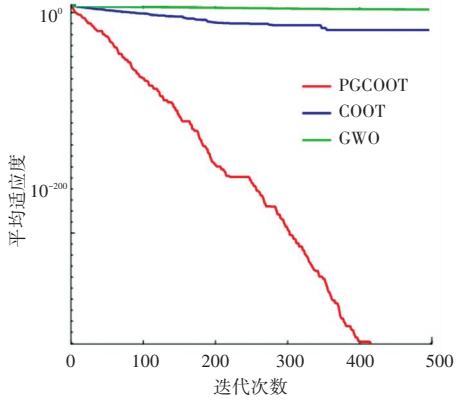
为了进一步验证本文所提 PGCOOT 算法,将与上述结果中表现最好的 GWO 和 COOT 在 $d = 100$ 时再次进行实验。所得实验数据见表5,收敛曲线见图4。



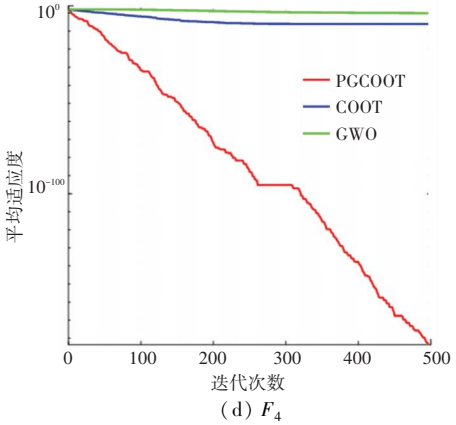
(a) F_1



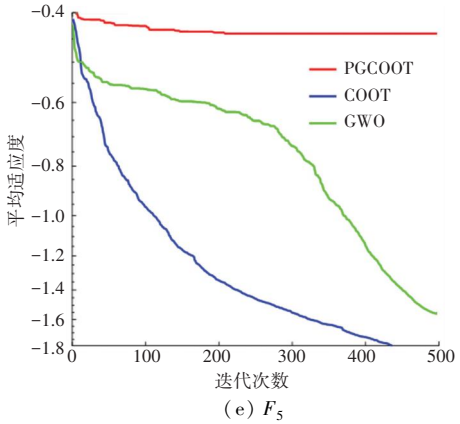
(b) F_2



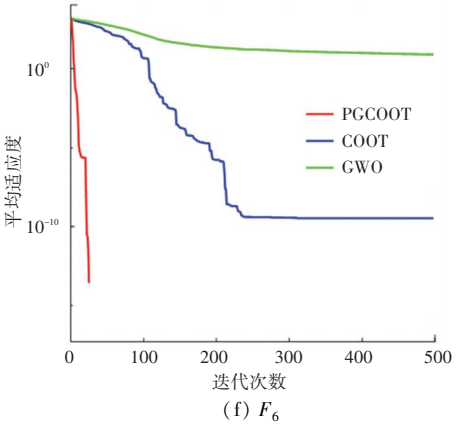
(c) F_3



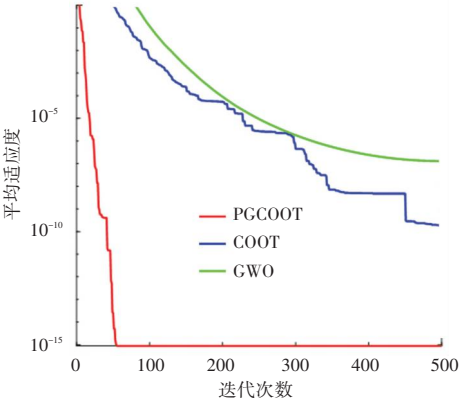
(d) F_4



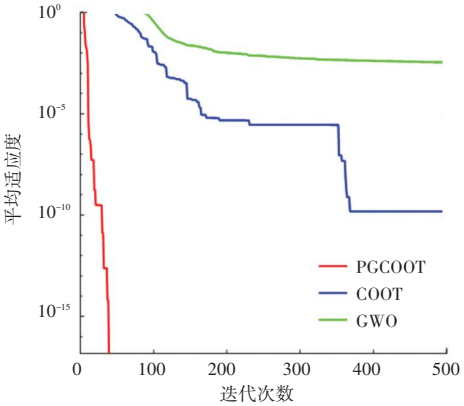
(e) F_5



(f) F_6



(g) F_7



(h) F_8

图 4　PGCOOT、GWO、COOT 收敛曲线

Fig. 4　Convergence curves of PGCOOT, GWO and COOT

表 5 PGCOOT、GWO、COOT 结果

Tab. 5 Results of PGCOOT、GWO、COOT

函数	算法	Mean	Best	Std
F_1	**PGCOOT**	**0**	**0**	**0**
	COOT	8.793 4E-19	8.256 4E-57	4.816 3E-18
	GWO	1.905 4E-12	1.883 2E-13	2.032 9E-12
F_2	**PGCOOT**	**5.155 7E-194**	**1.095 3E-253**	**0**
	COOT	4.824 4E-11	2.650 9E-24	2.635 2E-10
	GWO	4.781 2E-08	2.446 6E-08	1.497 7E-08
F_3	**PGCOOT**	**1.943 7E-308**	**0**	**0**
	COOT	1.607 5E-16	5.12E-50	8.804 5E-16
	GWO	652.942 8	67.007 86	507.981 9
F_4	**PGCOOT**	**2.114 4E-186**	**4.311 4E-247**	**0**
	COOT	9.431 2E-07	2.206 4E-28	5.165 7E-06
	GWO	0.939 5	0.095 852	0.991 53
F_5	**PGCOOT**	**-4 393.034**	**-3 014.643 6**	**992.491 11**
	COOT	-18 862.834	-12 969.379	2 857.586 55
	GWO	-15 602.845 1	-6 190.401 35	2 694.470 98
F_6	**PGCOOT**	**0**	**0**	**0**
	COOT	3.382 3E-10	0	1.849E-09
	GWO	7.991 7	2.106 6E-10	6.070 3
F_7	**PGCOOT**	**8.881 8E-16**	**8.881 8E-16**	**0**
	COOT	1.802 5E-10	8.881 8E-16	9.841 4E-10
	GWO	1.292 9E-07	6.122 4E-08	4.582 4E-08
F_8	**PGCOOT**	**0**	**0**	**0**
	COOT	1.501 4E-10	0	8.223 3E-10
	GWO	0.003 471 2	1.230 1E-13	0.009 272

从图 4 可见,对于 $F_1 \sim F_4$ 来看,PGCOOT、GWO、COOT 的寻优表现结果相似。PGCOOT 的收敛曲线下降迅速,而 GWO、COOT 的收敛曲线则较为缓和。从表 5 和图 4 来看,PGCOOT 在 F_1 能够 100%寻得理论最优值,其余算法均未寻到理论最优值。在 F_2、F_4 函数下,PGCOOT 虽未取得理论最优值,但与 GWO 和 COOT 相比,接近程度更胜一筹。并且 PGCOOT 在平均值与标准差上具有更好的稳定性。在 F_3 函数下,PGCOOT 虽不能够保证 100%寻得理论最优值,但仍保持收敛精度和收敛速度第一。由此可见,在高维单峰函数的实验中,GWO 和 COOT 在寻优能力和收敛速度上都会产生波动,甚至是下降的情况,与之相比 PGCOOT 仍能保持良好的寻优能力,进一步表明了 PGCOOT 整体性能的优越性。

在多峰函数 $F_5 \sim F_8$ 中,PGCOOT 在 F_5 函数上表现了更好的寻优能力,从表 5 和图 4 的结果可以看到,与 GWO 和 COOT 相比 PGCOOT 更接近于理论最优值。在 $F_6 \sim F_8$ 函数上,PGCOOT 的收敛速度比 GWO 与 COOT 快,能够迅速收敛至最优值附近。与 $d = 30$ 时相比,PGCOOT 的收敛精度、速度和稳定性均保持一致,这表明 PGCOOT 具有保持较强的鲁棒性和稳定性。综合 $F_5 \sim F_8$ 函数的结果来看,在高维多峰函数的情况下,PGCOOT 仍保持良好的寻优能力和收敛速度,而 GWO、COOT 在高维度求解复杂函数时会出现稳定性不强,寻优能力骤降的情况。

因此,不管在低维还是高维条件下,PGCOOT 都具有更好的寻优精度和稳定性、更快的寻优速度,这也表明了 PGCOOT 能够有效解决低维和高维的函数优化问题。

3.3 Wilcoxon 秩和检验

为验证 PGCOOT 算法与蚁狮算法(ALO)、灰狼优化算法(GWO)、蝗虫优化算法(GOA)、正余弦算法(SCA)、多元宇宙优化算法(MVO) 和原始的 COOT 算法在全局寻优上是否存在显著性区别,对各算法进行了性能测试比较。考虑到数据为非正态分布,因此使用均值的非参数检验方法,本文采取 Wilcoxon 秩和检验。

Wilcoxon 秩和检验的步骤为:首先,提出原假设 H_0: PGCOOT 算法与其他算法之间性能不存在明显差异。备择假设 H_1: PGCOOT 算法与其他算法之间性能有着明显差异。然后,根据 Wilcoxon 秩和检验原理计算出 P 值。最后,利用检验结果的 P 值来比较两种算法是否存在差异。当 P 值小于 0.05 时,拒绝 H_0, 即认为两种算法在性能上存在明显区别;当 P 值大于 0.05,不拒绝 H_0, 即不认为两种算法之间存在明显差异,两种算法在全局寻优上性能相当。Wilcoxon 秩和检验结果见表 6。

表 6 PGCOOT 算法与其他算法的 Wilcoxon 秩和检验结果

Tab. 6 Wilcoxon rank sum test results of PGCOOT algorithm and other algorithms

	COOT	GOA	SCA	GWO	ALO	SCA
F_1	4.470 8E-134	8.640 8E-161	1.252 6E-160	1.700 6E-122	3.213 6E-158	1.077 5E-161
F_2	2.856 4E-129	8.541 6E-160	1.882 1E-157	**2.495 5E-95**	8.129 5E-160	4.300 6E-161
F_3	3.936 9E-126	1.827 4E-161	4.762 6E-162	5.282 3E-152	2.703 9E-161	1.350 2E-162
F_4	1.832 6E-137	7.917 7E-162	8.947 7E-164	1.946 1E-144	1.941 8E-161	7.187 7E-163
F_5	4.117E-157	1.759 8E-162	2.621 4E-163	1.138 5E-161	5.432 2E-164	8.911 4E-163
F_6	4.353 2E-169	3.658 8E-182	3.277 6E-181	6.332 3E-180	3.039 8E-181	**3.015 3E-183**
F_7	6.165 9E-154	1.074 4E-175	1.072 6E-175	1.801 1E-125	3.656 3E-175	6.597 5E-177
F_8	1.019E-151	1.667 6E-179	1.898 5E-179	1.729 2E-167	2.233E-177	4.593 9E-180

从表 6 Wilcoxon 秩和检验可以看出，PGCOOT 算法与其他算法之间由 Wilcoxon 秩和检验所得的 P 值最大值为 2.495 5E-95，最小值为 3.015 3E-183，大部分值处于 9.35E-162 附近，远远小于 0.05。因此拒绝 H_0，认为 PGCOOT 算法与其他算法之间存在明显差异。结合 PGCOOT 算法的优异表现，即认为本文所提改进算法 PGCOOT 比其他算法具有更好的搜索能力。

4　结束语

本文针对 COOT 优化算法易陷入局部最优、收敛精度低等问题，提出了融合扰动因子和贪心策略的 PGCOOT 算法。将 PGCOOT 与其他多个算法在 8 个经典测试函数上对收敛速度、精度进行比较。仿真实验结果验证了所提出的精英反向学习初始化和贪心策略以及扰动因子相结合的 PGCOOT 算法的有效性，表明了 PGCOOT 算法在全局寻优以及局部搜索具有更优秀的寻优精度与速度。今后的研究将继续改进算法，进一步提高算法的寻优性能，同时将 PGCOOT 算法应用于实际问题。

参考文献

[1] MIRJALILI S. The ant lion optimizer[J]. Advances in engineering software, 2015, 83(5): 80-98.

[2] MIRJALILI S, MIRJALILI S M, LEWIS A. Grey wolf optimizer [J]. Advances in Engineering Software, 2014, 69(3): 46-61.

[3] SAREMI S, MIRJALILI S, LEWIS A. Grasshopper optimisation algorithm: Theory and application[J]. Advances in Engineering Software, 2017, 105(3):30-47.

[4] MIRJALILI S. SCA: A sine cosine algorithm for solving optimization problems[J]. Knowledge-Based Systems, 2016, 96:120-130.

[5] MIRJALILI S, MIRJALILI S M, HATAMLOU A, et al. Multi-verse optimizer: a nature - inspired algorithm for global optimization[J]. Neural Computing and Applications, 2016, 27 (2): 495-513.

[6] NARUEI I, KEYNIA F. A new optimization method based on coot bird natural life model[J]. Expert Systems with Applications, 2021, 183(2):115352.

[7] TIZHOOSH H R. Opposition-Based Learning: A new scheme for machine intelligence [C]// International Conference on International Conference on Computational Intelligence for Modelling, Control & Automation. IEEE, 2005:695-701.

[8] HEIDARI A A, PAHLAVANI P. An efficient modified grey wolf optimizer with lévy flight for optimization tasks[J]. Applied Soft Computing, 2017, 60:115-134.

[9] 魏伟一，文雅宏. 一种精英反向学习的萤火虫优化算法[J]. 智能系统学报，2017，12(5)：710-716.

[10] 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[J]. 电子学报，2019，47(10)：2177-2186.

[11] 杨文珍，何庆，杜逆索. 具有扰动机制和强化莱维飞行的蝗虫优化算法[J]. 小型微型计算机系统，2022，43(2)：247-253.

[12] 唐海波，林煜明，李优，等. 基于模拟退火与贪心策略的平衡聚类算法[J]. 计算机应用，2018，38(11)：3132-3138.

[13] 林杰，何庆. 融合正弦余弦和变异选择的蝗虫优化算法[J]. 小型微型计算机系统，2021，42(4)：706-713.

[14] 孟宪猛，蔡翠翠. 基于精英反向学习和 Lévy 飞行的鲸鱼优化算法[J]. 电子测量技术，2021，44(20)：82-87.

[15] 李爱莲，全凌翔，崔桂梅，等. 融合正余弦和柯西变异的麻雀搜索算法[J]. 计算机工程与应用，2022，58(3)：91-99.

[16] 高晨峰，陈家清，石默涵. 融合黄金正弦和曲线自适应的多策略麻雀搜索算法[J]. 计算机应用研究，2022，39(2)：491-499.

[17] 赵志刚，张振文，石辉磊. 带扰动因子的自适应粒子群优化算法[J]. 计算机科学，2013，40(12)：68-69，103.

(上接第 37 页)

[10] GAVAGSAZ E. Weighted spatial skyline queries with distributed dominance tests[J]. Cluster Computing, 2022, 25: 3249-3264.

[11] CHEN Zijun, GUO Shasha, LIU Wenyuan. Direction - based spatial- textual skyline [J]. International Journal of Innovative Computing, Information and Control, 2017, 13(6): 1813-1828.

[12] 陈子军，郭莎莎，刘文远，等. 基于社交的空间文本 skyline 查询[J]. 高技术通讯，2018，28(3)：194-206.

[13] ATTIQUE M, AFZAL M, ALI F, et al. Geo-social Top-k and skyline keyword queries on road networks[J]. Sensors, 2020, 20 (3): 798.

[14] CHEN Zijun, ZHAO Tingting, LIU Wenyuan. Time - aware collective spatial keyword query [J]. Computer Science and Information Systems, 2021, 18(3): 1077-1100.

[15] HUANG Y K. Continuous dε-skyline queries for objects with time varying attribute in road networks [C]//31st International Conference on Advanced Information Networking and Applications. Taiwan, China: IEEE, 2017: 439-446.

[16] 陈子军，杨蕊，刘文远，等. 基于旅行时间的 Top-k 轨迹查询[J]. 小型微型计算机系统，2019，40(7)：1496-1502.

[17] GUTTMAN A. R-trees: a dynamic index structure for spatial searching[C]//Proceedings of ACM SIGMOD Conference on Management of Data. New York, USA: Association for Computing Machinery, 1984: 47-57.

[18] CONG G, JENSEN C S, WU D. Efficientret rieval of the Top-k most relevant spatial web objects[J]. Proceedings of the Very Large DataBases Endowment, 2009, 2(1): 337-348.

[19] FELIPE I D, HRISTIDIS V, RISHE N. Keyword search on spatial databases[C]//IEEE 24th International Conference on Data Engineering. Cancun, Mexico: IEEE, 2008: 656-665.