灰狼与郊狼混合优化算法及其聚类优化

张新明[1,2] 姜 云[1] 刘尚旺[1,2] 刘国奇[1,2] 窦 智[1,2] 刘 艳[1,2]

摘 要 郊狼优化算法 (Coyote optimization algorithm, COA) 是最近提出的一种新颖且具有较大应用潜力的群智能优化算法, 具有独特的搜索机制和能较好解决全局优化问题等优势, 但在处理复杂优化问题时存在搜索效率低、可操作性差和收敛速度慢等不足. 为弥补其不足, 并借鉴灰狼优化算法 (Grey wolf optimizer, GWO) 的优势, 提出了一种 COA 与 GWO 的混合算法 (Hybrid COA with GWO, HCOAG). 首先提出了一种改进的 COA (Improved COA, ICOA), 即将一种高斯全局趋优成长算子替换原算法的成长算子以提高搜索效率和收敛速度, 并提出一种动态调整组内郊狼数方案, 使得算法的搜索能力和可操作性都得到增强; 然后提出了一种简化操作的 GWO (Simplified GWO, SGWO), 以提高算法的可操作性和降低其计算复杂度; 最后采用正弦交叉策略将 ICOA 与 SGWO 二者融合, 进一步获得更好的优化性能. 大量的经典函数和 CEC2017 复杂函数优化以及 K-Means 聚类优化的实验结果表明, 与 COA 相比, HCOAG 具有更高的搜索效率、更强的可操作性和更快的收敛速度, 与其他先进的对比算法相比, HCOAG 具有更好的优化性能, 能更好地解决聚类优化问题.

关键词 优化算法, 灰狼优化算法, 郊狼优化算法, 混合算法, 聚类优化

引用格式 张新明, 姜云, 刘尚旺, 刘国奇, 窦智, 刘艳. 灰狼与郊狼混合优化算法及其聚类优化. 自动化学报, 2022, **48**(11): 2757−2776

DOI 10.16383/j.aas.c190617

Hybrid Coyote Optimization Algorithm With Grey Wolf Optimizer and Its Application to Clustering Optimization

ZHANG Xin-Ming[1,2] JIANG Yun[1] LIU Shang-Wang[1,2] LIU Guo-Qi[1,2] DOU Zhi[1,2] LIU Yan[1,2]

Abstract Coyote optimization algorithm (COA) is a novel swarm intelligence optimization algorithm with great application potential, which was proposed recently. It has a unique search mechanism and the advantages to solve global optimization problems well and so on. But when dealing with the complex optimization problems, it has some defects, such as low search efficiency, poor operability, slow convergence speed and so on. To make up for COA′s disadvantages and utilize the advantages of grey wolf optimizer (GWO), a hybrid COA with GWO (HCOAG) is proposed. Firstly, an improved COA (ICOA) is proposed. A Gaussian global-best growing operator replaces the growing operator of the original algorithm to improve the search efficiency and convergence speed, and a dynamic adjustment scheme of coyote number in each group is proposed to enhance the search ability and operability. Secondly, in order to improve the operability and reduce the computational complexity of the algorithm, a simplified GWO (SGWO) is proposed. Finally, ICOA and SGWO are integrated by a sinusoidal crossover strategy to further get better optimization performance. A large number of experimental results on classical benchmark functions and CEC2017 complex functions and K-Means clustering show that, compared with COA, HCOAG has higher search efficiency, stronger operability and faster convergence speed. Compared with other state-of-the-art comparison algorithms, HCOAG has better optimization performance and can solve clustering optimization problems better.

Key words Optimization algorithm, grey wolf optimizer (GWO), coyote optimization algorithm (COA), hybrid algorithm, clustering optimization

Citation Zhang Xin-Ming, Jiang Yun, Liu Shang-Wang, Liu Guo-Qi, Dou Zhi, Liu Yan. Hybrid coyote optimization algorithm with grey wolf optimizer and its application to clustering optimization. *Acta Automatica Sinica*, 2022, **48**(11): 2757−2776

收稿日期 2019-09-02 录用日期 2020-03-11
Manuscript received September 2, 2019; accepted March 11, 2020
国家自然科学基金 (61901160, U1904123), 河南省高等学校重点科研项目 (19A520026) 资助
Supported by National Natural Science Foundation of China (61901160, U1904123) and Key Research Project of Higher Education Institutions of Henan Province (19A520026)

本文责任编委 金连文
Recommended by Associate Editor JIN Lian-Wen
1. 河南师范大学计算机与信息工程学院 新乡 453007 2. 智慧商务与物联网技术河南省工程实验室 新乡 453007
1. College of Computer and Information Engineering, Henan Normal University, Xinxiang 453007 2. Engineering Laboratory of Intelligence Business and Internet of Things, Xinxiang 453007

在现实世界中, 无时无处无不面临着优化问题. 随着社会的发展, 急需处理的优化问题越来越多样化并越来越复杂. 而诸如牛顿法、梯度下降法等传统的方法不能够很好地处理这些急需解决的优化问题. 因此, 许多研究者建立了各种仿生类的智能计算模型, 提出了许多智能优化算法. 群智能优化算法 (Swarm intelligence optimization algorithm, SIOA) 是模拟自然界社会群体集体行为的一种智能优化方法[1], 包括粒子群优化 (Particle swarm optimization, PSO) 算法[2]、灰狼优化算法 (Grey wolf optimizer, GWO)[3]、郊狼优化算法 (Coyote optimization algorithm, COA)[4] 等. SIOA 具有易于实现、能有效处理全局优化和大规模优化问题等优势, 广泛应用于多目标优化、数据聚类以及模式识别等诸多领域[5]. 但根据无免费午餐定理[6], 没有任何一种 SIOA 能独立地解决所有的优化问题, 每种 SIOA 都有自身的优势和局限性. 因此许多学者提出了大量新奇和改进的 SIOA, 其中包括算法的混合改进. 两种或多种算法的混合可以达到优势互补, 以获得最佳的优化性能[7].

GWO 是由 Mirjalili 等[3] 于 2014 年提出的一种新颖的 SIOA, 具有原理简单、开采能力强、可调参数少等优点, 但存在易陷入局部最优、解决复杂优化问题性能差等缺点. 因此许多学者对其进行了研究和改进, 其中包括混合改进. 例如张新明等[7] 提出一种 GWO 与人工蜂群算法的混合算法, 其中在人工蜂群观察蜂阶段自适应融合 GWO, 以便增强开采能力和提高优化效率. Zhang 等[8] 提出一种基于生物地理学优化算法和 GWO 的混合算法, 将改进的基于生物地理学优化算法和基于反向学习的 GWO 混合, 使得优化性能最大化. Teng 等[9] 提出一种 PSO 与 GWO 结合的混合算法, 该算法具有更好的寻优能力和更强的鲁棒性. Arora 等[10] 提出一种乌鸦搜索算法与 GWO 的混合算法, 更有效地实现了全局优化.

COA 是由 Pierezan 等[4] 于 2018 年提出的一种模拟郊狼群居生活、成长、生死等现象的新型 SIOA. COA 具有独特的搜索模型和结构以及出色的优化能力, 尤其在解决复杂优化问题优势明显, 但仍存在搜索效率低、可操作性不强以及收敛速度慢等缺陷, 且由于 COA 提出的时间较短, 需改进和完善并拓展其应用领域. 鉴于 GWO 和 COA 各有优势和不足, 本文将两种算法分别进行改进和简化, 得到改进的 COA (Improved COA, ICOA) 和简化的 GWO (Simplified GWO, SGWO), 然后通过正弦交叉策略将 ICOA 和 SGWO 有机融合在一起, 从而获得了一种性能优越且高效的混合 COA (Hybrid COA with GWO, HCOAG). 另外, 混合算法的研究一直是优化领域的热点, 因此研究 GWO 与 COA 的混合是一项较有意义的工作.

1 灰狼优化算法

GWO 模拟了自然界中灰狼种群严格的社会等级制度和群体捕食行为. 在其社会等级制度中, 灰狼从高到低依次为 α, β, δ, ω 狼. 其捕食行为是追踪和接近猎物、追捕和包围猎物、攻击和捕杀猎物. GWO 将狼群中的 α, β, δ, ω 狼的位置分别代表第一最优解、第二最优解、第三最优解和候选解. 灰狼包围行为的数学模型为

$$\boldsymbol{Dis} = |\boldsymbol{C} \otimes \boldsymbol{X}_p(t) - \boldsymbol{X}(t)| \tag{1}$$

$$\boldsymbol{X}(t+1) = \boldsymbol{X}_p(t) - \boldsymbol{A} \otimes \boldsymbol{Dis} \tag{2}$$

$$\boldsymbol{A} = 2a \times \boldsymbol{r}_1 - a \tag{3}$$

$$\boldsymbol{C} = c_1 \times \boldsymbol{r}_2 \tag{4}$$

$$a = 2 - \frac{2t}{MaxDT} \tag{5}$$

其中, $\boldsymbol{Dis}$ 是灰狼与猎物间的距离, t 是当前迭代次数, $\boldsymbol{X}_p$ 是猎物的位置向量, $\boldsymbol{X}$ 是灰狼的位置向量. $\boldsymbol{A}$ 和 $\boldsymbol{C}$ 是系数向量. 系数 a 在迭代过程中从 2 线性递减到 0, $MaxDT$ 是最大迭代次数, $\boldsymbol{r}_1$ 和 $\boldsymbol{r}_2$ 是 [0, 1] 中的均匀分布随机向量, c_1 为可调参数[11], $\otimes$ 表示两个向量各个对应的分量相乘. 狩猎行为的数学模型为

$$\boldsymbol{Dis}_\alpha = |\boldsymbol{C}_1 \otimes \boldsymbol{X}_\alpha(t) - \boldsymbol{X}(t)| \tag{6}$$

$$\boldsymbol{Dis}_\beta = |\boldsymbol{C}_2 \otimes \boldsymbol{X}_\beta(t) - \boldsymbol{X}(t)| \tag{7}$$

$$\boldsymbol{Dis}_\delta = |\boldsymbol{C}_3 \otimes \boldsymbol{X}_\delta(t) - \boldsymbol{X}(t)| \tag{8}$$

$$\boldsymbol{X}_1 = \boldsymbol{X}_\alpha(t) - A_1 \otimes \boldsymbol{Dis}_\alpha \tag{9}$$

$$\boldsymbol{X}_2 = \boldsymbol{X}_\beta(t) - A_2 \otimes \boldsymbol{Dis}_\beta \tag{10}$$

$$\boldsymbol{X}_3 = \boldsymbol{X}_\delta(t) - A_3 \otimes \boldsymbol{Dis}_\delta \tag{11}$$

$$\boldsymbol{X}(t+1) = \frac{\boldsymbol{X}_1 + \boldsymbol{X}_2 + \boldsymbol{X}_3}{3} \tag{12}$$

其中, $\boldsymbol{Dis}_\alpha$, $\boldsymbol{Dis}_\beta$ 和 $\boldsymbol{Dis}_\delta$ 分别表示当前狼与 3 头最优灰狼间的距离. 式 (12) 代表 ω 狼在 α, β 和 δ 狼的引导下移动的新位置, 即 GWO 产生的新解. GWO 的流程图见图 1.

从上述描述和图 1 可知, 与 PSO 等经典的 SIOA 相比, GWO 的主要特点有: 1) GWO 采用 3 头最优狼 (最优解) 引导 ω 狼围猎, 有较强的局部搜索能力, 但在解决复杂优化问题时, 容易陷入局部最优; 2) GWO 仅有两个参数 a 和 c_1, 前者采用动态调整

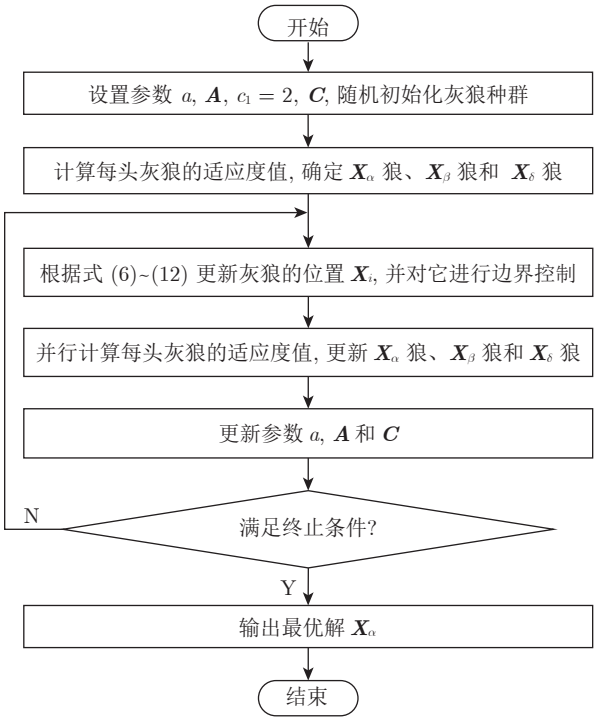

图 1　GWO 的流程图

Fig. 1　Flow chart of GWO

方式, 后者 c_1 常取常数 2, 调整的参数少, 故可操作性强; 3) 原理简单、易于实现, 但更新是基于维的计算, 需计算式 (6)~(12), 故计算复杂度较高; 4) 目标函数采用并行计算方式, 故运行速度较快.

2　郊狼优化算法

COA 包含初始化参数和狼群、组内郊狼成长、郊狼生与死和被组驱离与接纳[4] 等 4 个主要步骤.

2.1　参数初始化和随机初始化郊狼群组

首先设置参数, 如郊狼群规模 N, 郊狼组数 N_p, 组内郊狼数 N_c 和 $MaxDT$ 等, 其中, $N=N_c\times N_p$. 然后随机初始化郊狼群组, 第 p 组第 c 个郊狼第 j 维的随机化操作如式 (13). 最后计算每头郊狼 $\boldsymbol{soc}$ 的社会适应度值 fit, 如式 (14).

$$\boldsymbol{soc}_{c,\,j}=lb_j+r\times(ub_j-lb_j) \tag{13}$$

$$fit_c=f(\boldsymbol{soc}_c) \tag{14}$$

其中, lb_j 和 ub_j 分别表示郊狼第 j 维社会状态因子的下界和上界, $j=1,\,2,\,\cdots,\,D$, D 为搜索空间的维度, r 为 [0, 1] 内均匀分布的随机数, f 表示适应度函数.

2.2　组内郊狼成长

组内最优狼 $\boldsymbol{alpha}$、组文化趋势 $\boldsymbol{cult}$ 以及随机选取的两头郊狼 cr_1 和 cr_2 影响组内郊狼的成长过程, 即组内郊狼成长受 $\boldsymbol{\delta}_1$ 和 $\boldsymbol{\delta}_2$ 的影响. 其中 $\boldsymbol{cult}$ 的计算如式 (15) 所示, 即其每一个因素是组内所有郊狼对应社会因子的中位数 (O 为排名后的社会因子序列), 故 $\boldsymbol{cult}$ 又称中值郊狼. $\boldsymbol{\delta}_1$ 和 $\boldsymbol{\delta}_2$ 计算见式 (16) 和式 (17), 郊狼的成长方式见式 (18).

$$\boldsymbol{cult}_j=\begin{cases}O_{\frac{N_c+1}{2},\,j}, & N_c \text{ 是奇数}\\ \dfrac{O_{\frac{N_c}{2},\,j}+O_{\frac{N_c}{2}+1,\,j}}{2}, & N_c \text{ 是偶数}\end{cases} \tag{15}$$

$$\boldsymbol{\delta}_1=\boldsymbol{alpha}-\boldsymbol{soc}_{cr_1} \tag{16}$$

$$\boldsymbol{\delta}_2=\boldsymbol{cult}-\boldsymbol{soc}_{cr_2} \tag{17}$$

$$\boldsymbol{new\_soc}_c=\boldsymbol{soc}_c+r_3\times\boldsymbol{\delta}_1+r_4\times\boldsymbol{\delta}_2 \tag{18}$$

其中, $\boldsymbol{new\_soc}_c$ 是组内第 c 头郊狼成长获得的新解, r_3 和 r_4 是 [0, 1] 内均匀分布的随机数. 组内的郊狼成长后评估其社会适应能力如式 (19) 所示, new_fit_c 为新适应度值.

$$new\_fit_c=f(\boldsymbol{new\_soc}_c) \tag{19}$$

最后采用迭代贪心算法进行优胜劣汰 (式 (20)), 如此新产生的更优郊狼参与组内其余郊狼成长以加快收敛速度.

$$\boldsymbol{soc}=\begin{cases}\boldsymbol{new\_soc}_c, & new\_fit_c>fit_c\\ \boldsymbol{soc}_c, & \text{其他}\end{cases} \tag{20}$$

2.3　郊狼生与死

幼郊狼 (pup) 的诞生受随机选择父母郊狼的遗传基因和环境因素的影响, 见式 (21) 所示.

$$pup_j=\begin{cases}\boldsymbol{soc}_{cr_1,\,j}, & r_j<P_s \text{ 或 } j=j_1\\ \boldsymbol{soc}_{cr_2,\,j}, & r_j\geq P_s+P_a \text{ 或 } j=j_2\\ R_j, & \text{其他}\end{cases} \tag{21}$$

其中, r_j 是均匀分布在 [0, 1] 内的随机数; j_1 和 j_2 是两个随机选择的维度标号, 以确保两个父郊狼的基因能够遗传给幼狼; R_j 是在第 j 维决策变量范围内随机产生的变异值; P_s 和 P_a 分别是分散概率和关联概率, 它们决定着幼狼被遗传和变异的情况, 如式 (22) 和式 (23) 所示.

$$P_s=\frac{1}{D} \tag{22}$$

$$P_a=\frac{1-P_s}{2} \tag{23}$$

幼狼出生后, 评估其社会适应能力, 并依照其能力决定生与死, 具体描述如下: 在社会适应能力上, 1) 组内只有一个郊狼比幼狼差时, 则此郊狼死亡, 幼狼存活, 并设它的年龄为 0, 即 $age=0$; 2) 组

内有多个郊狼比幼狼差时, 能力差且年龄最大的郊狼死亡, 幼狼存活, 并设 $age = 0$; 3) 组内所有郊狼都比幼狼强时, 幼狼死亡.

2.4 郊狼被驱离和接纳

在 COA 中, 郊狼以 P_e 的概率被组驱离和接纳. 郊狼初始被随机分配到组群中, 但有时某些郊狼会离开它所在组群而加入另一组群, 这种随机驱离和接纳用来保证组内郊狼的多样性和组间信息共享, P_e 的计算方式为

$$P_e = 0.005 \times {N_c}^2 \tag{24}$$

COA 的伪代码如算法 1 所示.

算法 1. COA 算法

1. 设置 N_p 和 N_c 等参数, 随机初始化郊狼种群并随机分组, 评估每个郊狼的社会适应能力, 确定全局最优郊狼
2. **For** $t = 1$ **To** $MaxDT$ **Do**
3. **For** $p = 1$ **To** N_p **Do**
4. 确定组内最优的 ***alpha*** 狼
5. 根据式 (15) 计算组文化趋势 ***cult***
6. **For** $c = 1$ **To** N_c **Do**
7. 根据 ***alpha*** 和 ***cult*** 进行郊狼成长, 如式 (16)~(18)
8. 使用贪心算法优胜劣汰保留更优郊狼和全局最优郊狼
9. **End For**
10. 幼狼出生与死亡, 出生如式 (21), 按其社会能力决定其生死
11. **End For**
12. 郊狼采用如式 (24) 的概率被组驱离和接纳
13. 更新郊狼的年龄, $age = age + 1$
14. **End For**
15. 输出全局最优郊狼

从以上描述可知, 与 PSO 等经典的 SIOA 相比, COA 的主要优势有: 1) 具有很好的搜索框架, 即多组结构, 郊狼随机被组驱离和接纳等使郊狼群具有多样性, 有较强的探索能力, 能够更好地解决复杂优化问题[12]; 2) 组内最优郊狼和组内文化趋势引导组内每个郊狼的成长, 增强了局部搜索能力; 3) 郊狼组内的生与死受随机选择父郊狼的遗传因子和环境变异因子影响, 郊狼生与死算子使得 COA 具有一定的全局搜索能力. 其主要不足有: 1) 结构复杂, 计算复杂度高; 2) 采用迭代贪心算法, 稳定性差, 效率低; 3) 需调整的参数多, 如 P_e, N_p 和 N_c 等需调整, 可操作性差; 4) 多组结构等虽然增强了探索能力, 但组与组之间信息共享不足, 导致搜索后期收敛速度慢.

3 灰狼与郊狼混合优化算法

3.1 提出 HCOAG 的动机

COA 与 GWO 虽然同属狼群算法, 但二者有许多不同. 如在搜索方式上, 前者模拟郊狼的成长、生与死, 而后者模拟灰狼的等级制度和狩猎模式; 在引导上, 前者仅仅使用一头最好的狼来引导其它狼成长, 而后者使用 3 头最优狼来引导其它狼搜索; 在结构上, 前者复杂, 后者简单; 在产生新解方式上, 前者有两种方式, 后者只有一种方式等等. 因此, COA 和 GWO 各有优势和不足. 为了弥补二者不足, 本文将 COA 和 GWO 两种狼群算法经过改进后进行了有机结合, 从而获得一种性能优越且高效的混合算法 (HCOAG).

3.2 改进的郊狼优化算法

ICOA 主要包括高斯全局趋优成长算子和动态调整组内郊狼数方案.

3.2.1 高斯全局趋优成长算子

为了提高郊狼成长后的社会适应能力、组与组之间的信息共享程度及算法收敛速度, 受 Omran 等[13] 提出的全局最优和声搜索算法中趋优策略及高斯分布的启发, 对其组内郊狼成长方式进行改进, 提出一种高斯全局趋优成长算子. 全局趋优算子充分利用郊狼种群当前全局最优郊狼的状态信息, 使郊狼成长向当前全局最优郊狼趋近, 提高开采能力, 也使得组内信息经全局最优解达到组间共享. 高斯分布又称为正态分布, 与 COA 的均匀随机分布 [0, 1] 相比, 可以增加搜索范围, 在一定程度上增强全局搜索能力.

在组内郊狼成长过程中, 在式 (18) 中引入趋优算子和高斯分布随机因子, 具体见式 (25) 和式 (26).

$$\boldsymbol{\delta}_3 = \boldsymbol{GP} - \boldsymbol{soc}_{cr_1} \tag{25}$$

$$\boldsymbol{new\_soc}_1 = \boldsymbol{soc} + rn_1 \times \boldsymbol{\delta}_3 + rn_2 \times \boldsymbol{\delta}_2 \tag{26}$$

式 (25) 中, $\boldsymbol{GP}$ 为当前全局最优郊狼的状态, $\boldsymbol{\delta}_3$ 表示组内随机选取的一个郊狼状态 (cr_1) 与 $\boldsymbol{GP}$ 的差值. 式 (26)中, rn_1 和 rn_2 是均值为 0、方差为 1 的高斯 (正态) 分布产生的随机数, $\boldsymbol{new\_soc}_1$ 表示每头郊狼在 $\boldsymbol{\delta}_2$ 和 $\boldsymbol{\delta}_3$ 的共同作用下成长产生的新状态 (新解).

从式 (25) 和式 (26) 可以看出, 与 COA 相比, 在搜索前期, 个体间差异大, 缩放因子 (高斯随机数) 变化范围大, 故增强了探索能力. 在搜索后期, 虽然还是采用高斯随机数, 但个体间差异小, $\boldsymbol{\delta}_2$ 和 $\boldsymbol{\delta}_3$ 的值变小, 故搜索范围变小, 强化开采能力. 且由于采用全局最优解引导, 故更增强了开采能力, 同

时上一组获得的全局最优解, 会作用于下一组的郊狼成长, 如此形成一种信息共享的正反馈作用, 大幅度加快了收敛速度. 另外, 所有组内郊狼成长后并行计算它们的适应度值和优胜劣汰, 提高了运行速度和稳定性.

3.2.2　动态调整组内郊狼数方案

如上文所述, COA 有多个参数需要调整, 可操作性差. 对于以上改进的 COA, 主要有两个参数 N_c 和 N_p 对优化性能影响较大. 在 N 固定下, 如果 N_c 确定, 则 $N_p = N/N_c$, 即 N_p 越大, N_c 越少, 成长操作少, 但全局解的作用逐组增强, 开采强; 反之, 成长操作多, N_p 少, 全局解的作用减弱, 开采弱. 为了提高 COA 的可操作性, 本文对 N_p 和 N_c 参数进行动态调整. 设 $N = 100$, 则 N_p 和 N_c 必须为 100 的因子, 依据文献 [4], 每组郊狼数不能超过 14, 所以 N_c 只能为 4、5 和 10. 由于 N_c 不能少于 3, 这是因为组内郊狼成长至少需要 3 头郊狼, 包括随机选取的两头郊狼和组内最优郊狼. 当 $N_c = 4$ 时可选郊狼范围受限制, 所以 N_c 最可能的取值为 5 和 10, 故具体的分配方式如图 2 所示, 动态调整参数方案如算法 2 所示.

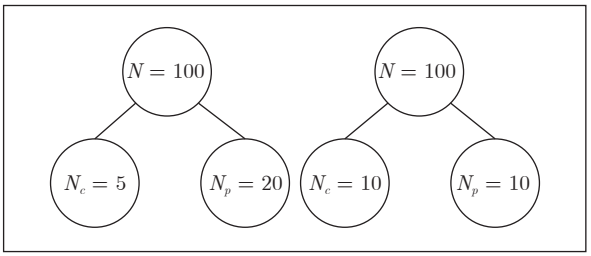

图 2　组数 N_p 与组内郊狼数 N_c 的分配图

Fig. 2　Disposition graph of two parameters N_c and N_p

算法 2. 动态调整参数 N_c 和 N_p

1. **If** 在搜索后期
2. $N_c = 5$
3. **Else** 在搜索前期
4. $N_c = 10$
5. **End If**
6. $N_p = N/N_c$ 和随机分组

在搜索后期, $N_c = 5$, 则 $N_p = 20$, 组数多, 增强全局解的正反馈作用, 局部搜索能力增强; 在搜索前期, $N_c = 10$, 则 $N_p = 10$, 组数少, 减弱全局解的正反馈作用, 全局搜索能力增强. 因此, 动态调整组内郊狼数参数不仅提高了可操作性, 而且可以更好地平衡探索与开采能力. 另外, 在动态调整参数之后, 随机分组, 如此可以省去郊狼被组驱离与接纳这个过程, 从而不需要调整参数 P_e, 因此提高了可操作性.

3.3　简化的灰狼优化算法

为了进一步解决 COA 存在搜索效率低和易陷入局部最优的问题, 本文引入 GWO 搜索方式. 首先提出一种精简的 GWO 搜索方式, 即将式 (6)~(11) 与 COA 组内郊狼进行融合并简化, 具体如式 (27)~(29).

$$\boldsymbol{NX}_1 = \boldsymbol{GP} - \boldsymbol{A}_1 \otimes |\boldsymbol{GP} - \boldsymbol{temp}_c| \tag{27}$$

$$\boldsymbol{NX}_2 = \boldsymbol{alpha} - A_2 \otimes |\boldsymbol{alpha} - \boldsymbol{temp}_c| \tag{28}$$

$$\boldsymbol{NX}_3 = \boldsymbol{cult} - A_3 \otimes |\boldsymbol{cult} - \boldsymbol{temp}_c| \tag{29}$$

其中, $\boldsymbol{temp}_c$ 表示当前组第 c 个郊狼的状态. $\boldsymbol{NX}_1$, $\boldsymbol{NX}_2$ 和 $\boldsymbol{NX}_3$ 表示组内郊狼分别在 $\boldsymbol{GP}$, $\boldsymbol{alpha}$ 和 $\boldsymbol{cult}$ 的引导下获得成长的情况. 从式 (27)~(29) 可以看出, 这 3 个式子去掉了式 (6) ~ (8) 中的可调参数向量 $\boldsymbol{C}$, 保留 GWO 的优势并克服其不足, 即在 SGWO 中, 去掉 $\boldsymbol{C}$, 无需调整 c_1, 也省略了向量 $\boldsymbol{C}$ 的相关计算. 所以, 这种简化的 GWO 在保证其有较强开采能力的同时, 提高了可操作性并降低了计算复杂度. 为了更进一步简化计算, SGWO 直接采用 COA 中的当前全局最优郊狼、组内最优郊狼和中值郊狼 (组内文化趋势) 的引导寻找最优解, 无需寻找组内的第二和第三最优郊狼, 如此 SGWO 与 COA 达到了一种高效融合. 为方便理解 SGWO, 关于 GWO 中的灰狼等级情况与 SGWO 中的等级情况的对比见图 3 所示.

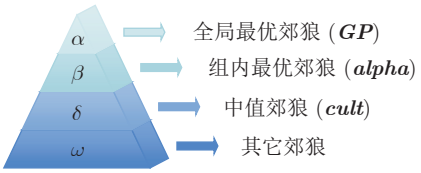

图 3　GWO 与 SGWO 的等级情况对比

Fig. 3　Comparison of hierarchies of GWO and SGWO

3.4　正弦交叉混合策略

为了平衡 COA 组内郊狼成长的探索与开采能力, 本文采用正弦交叉策略将 ICOA 与 SGWO 有机混合. 其中交叉策略是指在一定概率的情况下, 两个解进行交叉得到一个新解. 当概率为零时, 两个解的维值不交换; 当概率为 1 时, 两个解的维值全部交换, 这两种情况都不能产生新解. 因此只有概率为一个适当的值时, 两个解的交叉效果才会达到最好. 其中使用正弦函数概率模型使得探索和开采都有良好的表现, 即使用正弦函数自适应控制交叉概率 CR (在 0.5 附近前期小幅度波动, 后期大幅

度跳动) 可以兼顾组内郊狼的多样性和收敛性[14], 其计算式为

$$CR = 0.5\times\left(\sin\left(2\pi\times 0.25\times t+\pi\right)\times\frac{t}{MaxDT}+1\right) \tag{30}$$

本文采用的正弦交叉策略见算法 3 所示, 在 $rand < CR$ 的概率下, 组内第 c 个郊狼 D 维中一些维的值采用 SGWO 搜索方式获取; 反之, 组郊狼的第 c 个郊狼 D 维中的其余维值采用高斯全局趋优成长算子获取. 这种交叉策略具有如下特点: 1) 增强了组内各类郊狼的信息交流; 2) 交叉两个新解, 二者的信息融合获得有别二者的新解, 增强了新解的多样性, 降低陷入局部最优的概率; 3) 前期在 CR 为 0.5 的附近小幅度波动, 高斯全局趋优操作和 GWO 操作以近似等概率的方式产生新解, 多样性强, 强化探索能力; 后期在 CR 为 0.5 的附近大幅度跳动, 如此产生的新解以其中一种操作为主, 多样性降低, 强化了开采能力.

算法 3. 正弦交叉策略

1. 计算正弦交叉概率, 见式 (30)
2. 选定第 p 组第 c 个郊狼
3. **For** $j = 1$ **To** D **Do**
4. **If** $rand < CR$
5. 计算式 (27)~(29)
6. $\boldsymbol{new\_c}_j = (\boldsymbol{NX}_{1,j} + \boldsymbol{NX}_{2,j} + \boldsymbol{NX}_{3,j})/3$
7. **Else**
8. 采用高斯全局趋优成长算子的式 (26)
9. **End If**
10. **End For**

正弦交叉策略有机融合了两种搜索方式, 很好地平衡了成长过程的探索与开采能力. HCOAG 的流程图见图 4.

从图 4 可以看出, 与 COA 相比, HCOAG 主要有如下不同: 1) 成长方式采用正弦交叉策略融合高斯全局趋优方式和 SGWO 方式; 2) 组内郊狼的适应度值采用并行计算; 3) 动态调整参数方案; 4) 丢弃了郊狼被组驱离与接纳的过程.

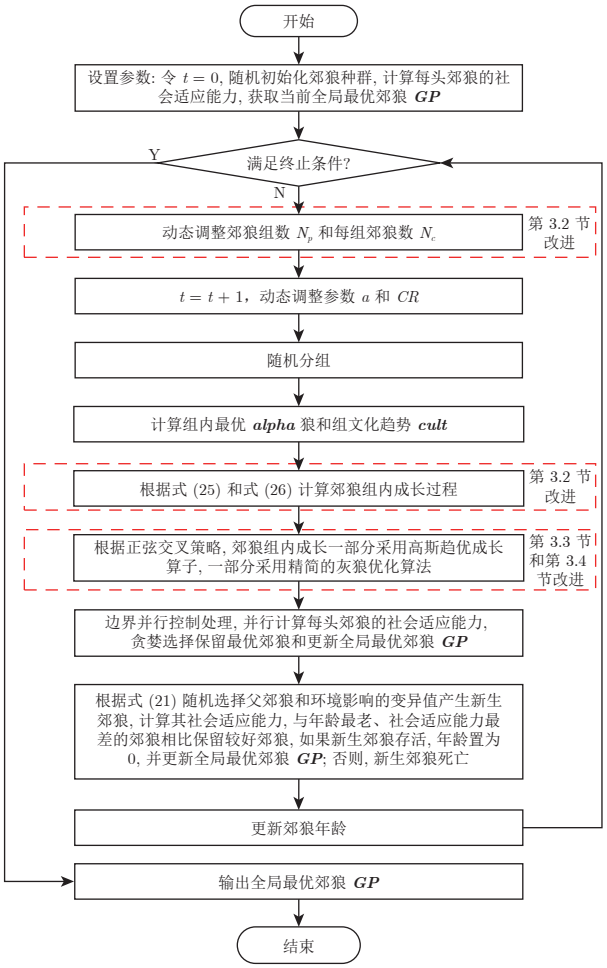

图 4 HCOAG 流程图

Fig. 4 Flow chart of HCOAG

4 函数优化实验结果与分析

4.1 实验环境配置及参数设置

为验证 HCOAG 的有效性, 本文采用经典基准函数和来自 CEC2017 的复杂函数进行优化实验. CEC2017 包括单峰函数 (F_1~F_3)、多峰函数 (F_4~F_{10})、混合函数 (F_{11}~F_{20}) 和复合函数 (F_{21}~F_{30}), 详细情况见文献 [15]. 实验环境采用主频 3.00 GHz 的 Inter(R) Core(TM) i5-7400 CPU 和内存 8 GB 的 PC 机, 操作系统采用 64 位的 Windows 10, 编程语言采用 MATLAB2017A. 选取的对比算法包括 GWO[3]、COA[4]、MEGWO (Multi-strategy ensemble GWO)[16]、DEBBO (Differential evolution and biogeography-based optimization)[17]、HFPSO (Hybrid firefly with PSO)[18]、SaDE (DE with strategy adaptation)[19]、SE04 (Spherical evolution)[20]、FWA (Fireworks algorithm)[21] 和 TLBO (Teaching-learning based optimization)[22]. MEGWO 是最近提出的 GWO 改进算法, 文献 [16] 表明它优于 GWO 及其改进版以及其他的 SIOA. DEBBO 是 DE 与 BBO 的混合算法, 文献 [17] 表明它优于 DE 和 BBO 及其他优秀的 SIOA. HFPSO 是 PSO 和 FA 的混合算法, 文献 [18] 表明它优于 PSO 和 FA 等 SIOA. SaDE 具有显著优秀性能, 文献 [19] 表明 SaDE 优于 DE 等 SIOA. SE04 是球形进化算法

的变体之一, 文献 [20] 表明 SE04 显著优于 GWO 等算法. FWA 和 TLBO 也是目前较为新型算法的代表. 总之, 这些对比算法具有较强的竞争性.

为公平起见, 所有算法的公共参数设置相同. 在 CEC2017 上, 依据文献 [15] 的参数最佳推荐, $D = 30$, 最大函数评价次数 10 000 × D, 独立运行 51 次. 在经典函数的 $D = 10$ 和 $D = 30$ 上, $MaxDT$ 分别为 100 和 500, 独立运行 30 次. 依据文献 [4] 的推荐, COA 的最佳参数设置为: $N = 100$, $N_c = 5$, $N_p = 20$. HCOAG 的 N 也设置为 100, N_p 和 N_c 无需调整, 算法其他参数采用相应文献中推荐的最佳设置.

一般单峰函数用来考察一个算法的局部搜索能力, 多峰函数考察其全局搜索能力, 混合和复合函数考察其处理复杂问题的能力. 本文采用均值 (Mean) 和方差 (Std) 分别评估一个算法的优化性能和稳定性. 在解决极小值问题中, 均值越小表示算法性能越好, 方差越小表示算法稳定性越好. 另外, 还采用了排名 (Rank) 方法. 其评价标准是先比较算法获得的均值, 均值越小算法名次越好; 在均值相同的情况下, 再比较方差, 方差越小算法名次越好. 结果表中的 "Count"表示排名为第一的总次数, "Ave rank"表示平均排名情况, "Total rank"表示在 "Ave rank"基础上的总排名情况, 最优者用加粗字体表示.

4.2　与不完全算法的对比分析

为验证每个改进对 HCOAG 优化性能的贡献, 将 HCOAG 与其不完全算法 HCOAG_5 ($N_c = 5$ 和 $N_p = 20$ 的 HCOAG)、HCOAG_{10} ($N_c = 10$ 和 $N_p = 10$ 的 HCOAG)、ICOA、SGWO、GWO 和 COA 在 CEC2017 的 30 维函数上进行实验, 实验结果如表 1 所示.

从表 1 中可以看出, HCOAG_5 在单峰函数上获

表 1　HCOAG 与其不完全算法的结果对比

Table 1　Comparison results of HCOAG and its incomplete algorithms

函数	标准	HCOAG	COA	GWO	HCOAG_5	HCOAG_{10}	ICOA	SGWO
F_1	Mean	7.4494×10^{-4}	1.2099×10^{3}	1.2813×10^{9}	$\mathbf{4.1072\times10^{-4}}$	1.9800×10^{-3}	1.0737×10^{2}	3.3279×10^{3}
	Std	1.4801×10^{-3}	1.2998×10^{3}	9.6388×10^{8}	$\mathbf{8.5916\times10^{-4}}$	2.9438×10^{-3}	1.0569×10^{2}	4.3271×10^{3}
	Rank	2	5	7	1	3	4	6
F_2	Mean	$\mathbf{1.1941\times10^{1}}$	2.9013×10^{21}	3.1831×10^{32}	4.8580×10^{3}	1.8078×10^{1}	8.6764×10^{15}	3.3582×10^{14}
	Std	$\mathbf{2.4077\times10^{1}}$	1.1462×10^{22}	1.5894×10^{33}	3.3985×10^{4}	5.0208×10^{1}	3.5675×10^{16}	1.3200×10^{15}
	Rank	1	6	7	3	2	5	4
F_3	Mean	9.5410×10^{-1}	6.0573×10^{4}	2.8342×10^{4}	$\mathbf{7.6972\times10^{-1}}$	1.0995×10^{0}	3.3032×10^{4}	8.7276×10^{2}
	Std	1.9288×10^{0}	1.0177×10^{4}	9.2323×10^{3}	$\mathbf{9.8032\times10^{-1}}$	1.6794×10^{0}	6.8409×10^{3}	7.2376×10^{2}
	Rank	2	7	5	1	3	6	4
F_4	Mean	$\mathbf{1.8113\times10^{1}}$	8.4041×10^{1}	2.0825×10^{2}	2.0841×10^{1}	2.8446×10^{1}	4.8248×10^{1}	1.0495×10^{2}
	Std	$\mathbf{2.7696\times10^{1}}$	8.5306×10^{0}	8.4445×10^{1}	3.0540×10^{1}	3.1826×10^{1}	3.3517×10^{1}	2.4806×10^{1}
	Rank	1	5	7	2	3	4	6
F_5	Mean	$\mathbf{2.8433\times10^{1}}$	5.2890×10^{1}	9.6116×10^{1}	3.5884×10^{1}	3.0204×10^{1}	3.4844×10^{1}	3.1488×10^{1}
	Std	$\mathbf{6.8886\times10^{0}}$	1.5025×10^{1}	3.2690×10^{1}	1.0115×10^{1}	8.8983×10^{0}	1.0983×10^{1}	9.2242×10^{0}
	Rank	1	6	7	5	2	4	3
F_6	Mean	$\mathbf{1.7483\times10^{-7}}$	1.6399×10^{-5}	6.3664×10^{0}	9.4452×10^{-7}	1.5005×10^{-6}	2.8782×10^{-4}	2.0381×10^{-2}
	Std	$\mathbf{4.7524\times10^{-7}}$	9.6428×10^{-6}	3.1596×10^{0}	2.4080×10^{-6}	5.9643×10^{-6}	1.6254×10^{-4}	2.7102×10^{-2}
	Rank	1	4	7	2	3	5	6
F_7	Mean	6.1055×10^{1}	7.5148×10^{1}	1.4460×10^{2}	6.7082×10^{1}	$\mathbf{5.9300\times10^{1}}$	6.7675×10^{1}	6.4025×10^{1}
	Std	1.0851×10^{1}	1.3762×10^{1}	4.6314×10^{1}	1.1241×10^{1}	$\mathbf{9.4998\times10^{0}}$	1.1520×10^{1}	1.1856×10^{1}
	Rank	2	6	7	4	1	5	3
F_8	Mean	3.2489×10^{1}	5.6110×10^{1}	8.4662×10^{1}	3.6085×10^{1}	$\mathbf{2.9446\times10^{1}}$	3.6138×10^{1}	3.1775×10^{1}
	Std	1.2272×10^{1}	1.8774×10^{1}	2.5270×10^{1}	8.8063×10^{0}	$\mathbf{7.9048\times10^{0}}$	1.0081×10^{1}	7.8884×10^{0}
	Rank	3	6	7	4	1	5	2
F_9	Mean	2.7362×10^{-1}	5.6225×10^{-1}	5.5392×10^{2}	5.2270×10^{-1}	2.5931×10^{-1}	$\mathbf{8.8559\times10^{-2}}$	7.4159×10^{0}
	Std	4.8298×10^{-1}	1.0209×10^{0}	3.2695×10^{2}	8.7374×10^{-1}	4.6957×10^{-1}	$\mathbf{1.5656\times10^{-1}}$	6.7112×10^{0}
	Rank	3	5	7	4	2	1	6

表 1 HCOAG 与其不完全算法的结果对比 (续表)

Table 1 Comparison results of HCOAG and its incomplete algorithms (continued table)

函数	标准	HCOAG	COA	GWO	$HCOAG_5$	$HCOAG_{10}$	ICOA	SGWO
	Mean	2.2671×10^{3}	2.7575×10^{3}	3.1862×10^{3}	2.5574×10^{3}	2.3435×10^{3}	$\mathbf{2.1380\times10^{3}}$	2.1424×10^{3}
F_{10}	Std	6.1427×10^{2}	4.6685×10^{2}	9.7886×10^{2}	5.3524×10^{2}	6.0670×10^{2}	$\mathbf{5.6098\times10^{2}}$	4.0691×10^{2}
	Rank	3	6	7	5	4	1	2
	Mean	$\mathbf{2.1678\times10^{1}}$	4.1143×10^{1}	4.9771×10^{2}	2.9822×10^{1}	2.6698×10^{1}	2.2685×10^{1}	1.0908×10^{2}
F_{11}	Std	$\mathbf{2.0907\times10^{1}}$	2.7367×10^{1}	6.4235×10^{2}	2.6128×10^{1}	2.4059×10^{1}	2.0893×10^{1}	3.8218×10^{1}
	Rank	1	5	7	4	3	2	6
	Mean	$\mathbf{9.8943\times10^{3}}$	1.2532×10^{5}	4.0285×10^{7}	1.2577×10^{4}	1.0657×10^{4}	1.3660×10^{5}	2.0716×10^{5}
F_{12}	Std	$\mathbf{6.0932\times10^{3}}$	1.2555×10^{5}	7.3849×10^{7}	6.4424×10^{3}	6.4606×10^{3}	9.3092×10^{4}	1.9967×10^{5}
	Rank	1	4	7	3	2	5	6
	Mean	1.9749×10^{3}	2.0357×10^{4}	2.8073×10^{6}	3.0265×10^{3}	3.1829×10^{3}	$\mathbf{3.6293\times10^{2}}$	1.3271×10^{4}
F_{13}	Std	3.8565×10^{3}	2.6333×10^{4}	1.6225×10^{7}	6.2901×10^{3}	8.0719×10^{3}	$\mathbf{8.9975\times10^{1}}$	1.5283×10^{4}
	Rank	2	6	7	3	4	1	5
	Mean	8.6436×10^{1}	8.0070×10^{1}	1.3112×10^{5}	7.7150×10^{1}	1.0134×10^{2}	$\mathbf{5.6726\times10^{1}}$	1.4132×10^{4}
F_{14}	Std	4.3766×10^{1}	1.9915×10^{1}	2.3335×10^{5}	6.0071×10^{1}	9.2585×10^{1}	$\mathbf{1.4850\times10^{1}}$	1.7944×10^{4}
	Rank	4	3	7	2	5	1	6
	Mean	1.8396×10^{3}	2.0792×10^{3}	3.3658×10^{5}	7.1579×10^{2}	1.7386×10^{3}	$\mathbf{6.9111\times10^{1}}$	6.6116×10^{3}
F_{15}	Std	2.9044×10^{3}	7.9984×10^{3}	7.9125×10^{5}	1.2272×10^{3}	2.9477×10^{3}	$\mathbf{1.9083\times10^{1}}$	8.3961×10^{3}
	Rank	4	5	7	2	3	1	6
	Mean	$\mathbf{3.0243\times10^{2}}$	7.9869×10^{2}	8.1416×10^{2}	3.3715×10^{2}	3.0883×10^{2}	4.6416×10^{2}	5.0252×10^{2}
F_{16}	Std	$\mathbf{2.0550\times10^{2}}$	2.8651×10^{2}	2.6440×10^{2}	2.1415×10^{2}	1.8878×10^{2}	2.6962×10^{2}	2.4545×10^{2}
	Rank	1	6	7	3	2	4	5
	Mean	4.7111×10^{1}	2.2439×10^{2}	2.7004×10^{2}	6.4809×10^{1}	5.3120×10^{1}	$\mathbf{3.7365\times10^{1}}$	1.3984×10^{2}
F_{17}	Std	4.0925×10^{1}	1.3518×10^{2}	1.3820×10^{2}	5.3991×10^{1}	4.7801×10^{1}	$\mathbf{4.0654\times10^{1}}$	8.0664×10^{1}
	Rank	2	6	7	4	3	1	5
	Mean	6.1013×10^{4}	6.9910×10^{4}	7.1643×10^{5}	5.1875×10^{4}	5.0376×10^{4}	$\mathbf{3.9930\times10^{4}}$	1.8454×10^{5}
F_{18}	Std	5.7031×10^{4}	1.0210×10^{5}	8.2799×10^{5}	3.6270×10^{4}	3.7592×10^{4}	$\mathbf{2.0034\times10^{4}}$	1.7045×10^{5}
	Rank	4	5	7	3	2	1	6
	Mean	3.4042×10^{1}	4.4886×10^{3}	4.6400×10^{5}	3.4163×10^{1}	3.0105×10^{2}	$\mathbf{2.4678\times10^{1}}$	5.4815×10^{3}
F_{19}	Std	2.0528×10^{1}	1.3325×10^{4}	5.4998×10^{5}	3.9687×10^{1}	1.1322×10^{3}	$\mathbf{7.1259\times10^{0}}$	4.9859×10^{3}
	Rank	2	5	7	3	4	1	6
	Mean	$\mathbf{9.6665\times10^{1}}$	2.4290×10^{2}	3.6059×10^{2}	1.0084×10^{2}	1.1637×10^{2}	1.0389×10^{2}	2.0165×10^{2}
F_{20}	Std	$\mathbf{7.7834\times10^{1}}$	1.4995×10^{2}	1.0264×10^{2}	6.6726×10^{1}	7.5890×10^{1}	8.7694×10^{1}	9.6673×10^{1}
	Rank	1	6	7	2	4	3	5
	Mean	$\mathbf{2.3023\times10^{2}}$	2.5626×10^{2}	2.8298×10^{2}	2.3713×10^{2}	2.3135×10^{2}	2.3724×10^{2}	2.3289×10^{2}
F_{21}	Std	$\mathbf{8.5095\times10^{0}}$	1.6800×10^{1}	2.5684×10^{1}	1.0815×10^{1}	9.8453×10^{0}	1.1261×10^{1}	9.9428×10^{0}
	Rank	1	6	7	4	2	5	3
	Mean	1.0010×10^{2}	1.9999×10^{3}	8.0434×10^{2}	$\mathbf{1.0005\times10^{2}}$	1.0005×10^{2}	1.0005×10^{2}	1.2974×10^{2}
F_{22}	Std	4.8096×10^{-1}	1.5970×10^{3}	1.1113×10^{3}	$\mathbf{3.4354\times10^{-1}}$	3.4444×10^{-1}	3.4443×10^{-1}	2.0703×10^{2}
	Rank	4	7	6	1	3	2	5
	Mean	$\mathbf{3.7755\times10^{2}}$	4.1635×10^{2}	4.7029×10^{2}	3.8706×10^{2}	3.7831×10^{2}	3.8441×10^{2}	3.8854×10^{2}
F_{23}	Std	$\mathbf{1.0911\times10^{1}}$	1.6898×10^{1}	2.9324×10^{1}	1.3271×10^{1}	8.1817×10^{0}	1.0543×10^{1}	1.3692×10^{1}
	Rank	1	6	7	4	2	3	5
	Mean	4.4827×10^{2}	5.4044×10^{2}	5.2489×10^{2}	4.5484×10^{2}	$\mathbf{4.4530\times10^{2}}$	4.5862×10^{2}	4.5671×10^{2}
F_{24}	Std	1.0757×10^{1}	4.5778×10^{1}	3.3902×10^{1}	1.1783×10^{1}	$\mathbf{1.1461\times10^{1}}$	1.2203×10^{1}	1.1956×10^{1}
	Rank	2	7	6	3	1	5	4

表 1 HCOAG 与其不完全算法的结果对比 (续表)

Table 1 Comparison results of HCOAG and its incomplete algorithms (continued table)

函数	标准	HCOAG	COA	GWO	$HCOAG_5$	$HCOAG_{10}$	ICOA	SGWO
F_{25}	Mean	3.8777×10^{2}	3.8706×10^{2}	4.7784×10^{2}	3.8747×10^{2}	3.8729×10^{2}	$\mathbf{3.8698\times10^{2}}$	4.0239×10^{2}
	Std	5.4462×10^{0}	8.0647×10^{-1}	2.3819×10^{1}	1.5625×10^{0}	1.2735×10^{0}	$\mathbf{5.4095\times10^{-1}}$	1.5135×10^{1}
	Rank	5	2	7	4	3	1	6
F_{26}	Mean	1.2578×10^{3}	1.6520×10^{3}	2.0116×10^{3}	1.3249×10^{3}	$\mathbf{1.2449\times10^{3}}$	1.3138×10^{3}	1.5024×10^{3}
	Std	1.9807×10^{2}	1.7070×10^{2}	5.7618×10^{2}	3.1093×10^{2}	$\mathbf{3.4947\times10^{2}}$	1.9599×10^{2}	2.8691×10^{2}
	Rank	2	6	7	4	1	3	5
F_{27}	Mean	5.1091×10^{2}	$\mathbf{5.0430\times10^{2}}$	5.9279×10^{2}	5.1349×10^{2}	5.1088×10^{2}	5.0560×10^{2}	5.3331×10^{2}
	Std	7.7116×10^{0}	$\mathbf{8.2707\times10^{0}}$	3.8462×10^{1}	8.6401×10^{0}	8.6837×10^{0}	7.3827×10^{0}	1.1175×10^{1}
	Rank	4	1	7	5	3	2	6
F_{28}	Mean	3.3558×10^{2}	4.0555×10^{2}	5.9941×10^{2}	$\mathbf{3.2930\times10^{2}}$	3.3793×10^{2}	3.4828×10^{2}	4.5710×10^{2}
	Std	5.3866×10^{1}	3.6156×10^{1}	6.9788×10^{1}	$\mathbf{5.1585\times10^{1}}$	5.1950×10^{1}	5.3564×10^{1}	2.3392×10^{1}
	Rank	2	5	7	1	3	4	6
F_{29}	Mean	4.5991×10^{2}	6.6978×10^{2}	8.5036×10^{2}	4.8821×10^{2}	4.6287×10^{2}	$\mathbf{4.5683\times10^{2}}$	6.2453×10^{2}
	Std	4.4075×10^{1}	1.7459×10^{2}	1.8235×10^{2}	5.5772×10^{1}	4.3839×10^{1}	$\mathbf{4.4800\times10^{1}}$	1.1861×10^{2}
	Rank	2	6	7	4	3	1	5
F_{30}	Mean	2.9823×10^{3}	6.0618×10^{3}	4.0643×10^{6}	3.1036×10^{3}	$\mathbf{2.9323\times10^{3}}$	1.9586×10^{4}	6.1880×10^{3}
	Std	6.1135×10^{2}	4.7022×10^{3}	3.1688×10^{6}	8.4665×10^{2}	$\mathbf{5.9332\times10^{2}}$	7.3086×10^{3}	2.8250×10^{3}
	Rank	2	4	7	3	1	6	5
Count		10	1	0	4	5	10	0
Ave rank		2.20	5.23	6.87	3.10	2.60	3.07	4.93
Total rank		1	6	7	4	2	3	5

得最好的结果次数最多, 表明 N_p 增大, 在提高郊狼生与死的全局搜索能力的同时, 大幅度提高了 COA 的开采能力, 即 $HCOAG_5$ 中高斯趋优成长和 GWO 搜索都采用当前全局最优郊狼引导, 不仅获得更好的全局最优解, 这种全局最优解又作用于下一组郊狼的成长过程, 如此形成正反馈机制: N_p 越大, 开采越强, 收敛速度越快. 但容易陷入局部最优, 如在多峰函数上的表现不尽人意. $HCOAG_{10}$ 在多峰函数上表现出了良好的优化性能, 表明 N_p 变少, 在提高组内郊狼成长的局部搜索能力的同时, 正反馈减弱, 开采能力降低, 即陷入局部最优的概率也降低了. ICOA 中没有精简的 GWO, 整体来说优于 COA, 表明对 COA 的改进是有效的. SGWO 在整体上优于 GWO, 表明 SGWO 提高了 GWO 搜索能力. 与 GWO 相比, COA 的优化性能更好, 即 COA 能更好地处理复杂优化问题. 在 7 个算法中, HCOAG 的平均排名为 2.20, COA、GWO、$HCOAG_5$、$HCOAG_{10}$、ICOA 和 SGWO 的平均排名分别为 5.23、6.87、3.10、2.60、3.07 和 4.93, 总排名的名次分别为 6、7、4、2、3 和 5. 在 HCOAG 与 6 种对比算法中, HCOAG 获得了最好的优化性能, 这也说明本文提出的算法是有效的.

4.3 在经典函数上的优化性能对比

为验证 HCOAG 的性能, 首先将其用于经典函数的优化实验, 并将其结果与 COA、GWO、HFPSO 和 DEBBO 的结果进行对比, 实验结果见表 2.

为了能说明问题, 选择 6 个经典的、具有代表性的基准函数的实验结果作为示例进行分析和讨论, 这 6 个函数的信息如表 3 所示, 其中 $f_1 \sim f_3$ 和 $f_4 \sim f_6$ 分别作为单峰函数和多峰函数的代表. 在此实验中, 由于 HCOAG 是 COA 和 GWO 的混合算法, 故选择 COA 和 GWO 作为对比算法. HFPSO 和 DEBBO 是目前两种较为优秀的混合算法, 故选为 HCOAG 的对比算法.

从表 2 可以看出, 在 $D = 10$ 和 $D = 30$ 上, COA 无论是在单峰还是在多峰函数上优化性能最差, 说明 COA 在处理这些经典函数优化问题上无优势. 在单峰函数的均值和方差上, 除了 f_3 外, GWO 均优于 HCOAG 以及其他 3 种对比算法, 证明了 GWO 具有较好的开采能力. 在多峰函数上, HCOAG 的性能最佳, 说明 GWO 与全局搜索能力强的 COA 混合有效.

从整体上看, 在 $D = 10$ 和 $D = 30$ 上, HCO-

表 2 在 6 个经典函数上的实验结果对比

Table 2 Comparison results on the 6 classic functions

函数	标准	HCOAG	COA	GWO	HFPSO	DEBBO
				$D = 10$		
f_1	Mean	6.0684×10^{-9}	1.7833×10^{2}	$\mathbf{9.0799\times10^{-15}}$	3.4157×10^{-5}	6.7086×10^{-2}
	Std	4.8458×10^{-9}	6.3524×10^{1}	$\mathbf{2.4849\times10^{-14}}$	2.4485×10^{-5}	3.1056×10^{-2}
	Rank	2	5	1	3	4
f_2	Mean	8.4133×10^{-6}	2.3737×10^{0}	$\mathbf{1.3222\times10^{-9}}$	1.3703×10^{-3}	2.8483×10^{-2}
	Std	3.7531×10^{-6}	4.0964×10^{-1}	$\mathbf{1.1382\times10^{-9}}$	6.5582×10^{-4}	7.0993×10^{-3}
	Rank	2	5	1	3	4
f_3	Mean	**0**	1.6180×10^{2}	1.0000×10^{-1}	**0**	**0**
	Std	**0**	5.0787×10^{1}	3.0513×10^{-1}	**0**	**0**
	Rank	1	5	4	1	1
f_4	Mean	$\mathbf{1.2498\times10^{-10}}$	4.0253×10^{0}	1.5325×10^{-6}	3.5760×10^{-7}	1.8575×10^{-3}
	Std	$\mathbf{2.0501\times10^{-10}}$	1.6104×10^{0}	9.4192×10^{-7}	4.1539×10^{-7}	1.0158×10^{-3}
	Rank	1	5	3	2	4
f_5	Mean	$\mathbf{2.0046\times10^{-8}}$	7.1619×10^{2}	2.4093×10^{-5}	4.6770×10^{-6}	2.6132×10^{-2}
	Std	$\mathbf{5.9686\times10^{-8}}$	1.5819×10^{3}	1.4121×10^{-5}	5.6661×10^{-6}	1.0613×10^{-2}
	Rank	1	5	3	2	4
f_6	Mean	$\mathbf{4.1921\times10^{-10}}$	1.7228×10^{0}	1.2593×10^{-2}	4.9377×10^{-7}	3.5149×10^{-3}
	Std	$\mathbf{4.3501\times10^{-10}}$	5.1829×10^{-1}	6.8779×10^{-2}	4.8665×10^{-7}	1.5826×10^{-3}
	Rank	1	5	4	2	3
				$D = 30$		
f_1	Mean	1.3966×10^{-17}	3.2554×10^{1}	$\mathbf{5.4432\times10^{-41}}$	7.2595×10^{-9}	2.7076×10^{-4}
	Std	3.2255×10^{-17}	5.9567×10^{0}	$\mathbf{7.1605\times10^{-41}}$	7.3446×10^{-9}	1.1010×10^{-4}
	Rank	2	5	1	3	4
f_2	Mean	2.8862×10^{-10}	1.3998×10^{0}	$\mathbf{6.0158\times10^{-24}}$	5.3463×10^{-5}	1.3264×10^{-3}
	Std	4.6435×10^{-10}	1.9835×10^{-1}	$\mathbf{6.2049\times10^{-24}}$	3.9178×10^{-5}	2.3103×10^{-4}
	Rank	2	5	1	3	4
f_3	Mean	**0**	3.3700×10^{1}	3.3333×10^{-2}	**0**	**0**
	Std	**0**	7.9877×10^{0}	1.8257×10^{-1}	**0**	**0**
	Rank	1	5	4	1	1
f_4	Mean	$\mathbf{1.0451\times10^{-17}}$	3.8002×10^{0}	1.5129×10^{-2}	1.7278×10^{-2}	6.4886×10^{-5}
	Std	$\mathbf{3.0738\times10^{-17}}$	1.3163×10^{0}	1.0471×10^{-2}	3.9296×10^{-2}	2.7878×10^{-5}
	Rank	1	5	3	4	2
f_5	Mean	$\mathbf{5.3309\times10^{-17}}$	1.8376×10^{1}	1.6587×10^{-1}	3.2962×10^{-3}	5.0150×10^{-4}
	Std	$\mathbf{1.5184\times10^{-16}}$	5.8919×10^{0}	1.1940×10^{-1}	5.1211×10^{-3}	2.1237×10^{-4}
	Rank	1	5	4	3	2
f_6	Mean	$\mathbf{1.2484\times10^{-18}}$	1.8049×10^{0}	7.9738×10^{-1}	8.2480×10^{-3}	8.5930×10^{-5}
	Std	$\mathbf{1.7135\times10^{-18}}$	4.9152×10^{-1}	7.4565×10^{-1}	4.5176×10^{-2}	3.9292×10^{-5}
	Rank	1	5	4	3	2
Count		8	0	4	2	2
Ave rank		1.33	5.00	2.75	2.50	2.92
Total rank		1	5	3	2	4

AG 的总体平均排名为 1.33, 其他算法的平均排名依次为: HFPSO (2.50)、GWO (2.75)、DEBBO (2.92) 和 COA (5.00). 这些结果表明, HCOAG 不是对所有优化问题都有绝对优势, 在单峰函数上优化性能稍差. 而 GWO 在单峰函数上具有优势, 对 HCOAG 的性能具有一定的贡献. 从整体上看,

表 3　6 个经典函数的情况

Table 3　Details of 6 classical benchmark functions

类型	函数名称	函数表达式	搜索范围	最小值
单峰函数	Sphere	$f_1(x)=\sum_{i=1}^{D}x_i^2$	[−100, 100]	0
	Schwefel 2.22	$f_2(x)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D}\lvert x_i\rvert$	[−10, 10]	0
	Step	$f_3(x)=\sum_{i=1}^{D}(\lfloor x_i+0.5\rfloor)^2$	[−100, 100]	0
多峰函数	Penalized 1	$f_4(x)=\frac{\pi}{D}\Big\{10\sin^2(\pi y_i)+\sum_{i=1}^{D-1}(y_i-1)^2\left[1+10\sin^2(\pi y_{i+1})\right]+(y_D-1)^2\Big\}+\sum_{i=1}^{D}u(x_i,10,100,4)$ $y_i=1+\frac{1}{4}(x_i+1)$ $u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m, & x_i>a\\ 0, & -a\le x_i\le a\\ k(-x_i-a)^m, & x_i<-a\end{cases}$	[−50, 50]	0
	Penalized 2	$f_5(x)=0.1\Big\{\sin^2(\pi x_1)+\sum_{i=1}^{D-1}(x_i-1)^2\left[1+\sin^2(3\pi x_{i+1})\right]+(x_D-1)\left[1+\sin^2(2\pi x_D)\right]\Big\}+\sum_{i=1}^{D}u(x_i,5,100,4)$	[−50, 50]	0
	Levy	$f_6(x)=\sum_{i=1}^{D-1}(x_i-1)^2\left[1+\sin^2(3\pi x_{i+1})\right]+\sin^2(3\pi x_1)+\lvert x_D-1\rvert\left[1+\sin^2(3\pi x_D)\right]$	[−10, 10]	0

HCOAG 在经典函数上具有较好的优化性能.

为直观显示 HCOAG、COA、GWO、HFPSO 和 DEBBO 的收敛性能, 给出了这 5 个算法的收敛图. 限于篇幅, 本节在 $D=30$ 上选取 2 个单峰函数 (f_1 和 f_2) 和 2 个多峰函数 (f_5 和 f_6), 如图 5 所示. 在单峰函数上, 与 HCOAG、COA、HFPSO 和 DEBBO 算法相比, GWO 收敛速度更快, 表现出更好的收敛性能. 在多峰函数上, 与 COA、GWO、HFPSO 和 DEBBO 算法相比, HCOAG 表现出了更好的收敛性能. 综上所述, 表 2 和图 5 说明了 GWO 具有较好的局部搜索能力和收敛性能, 也部分证明了利用 GWO 局部搜索能力强的优势对 COA 的混合改进是必要和可行的.

4.4　在 CEC2017 复杂函数上的优化性能对比

为了进一步验证 HCOAG 的搜索能力, 将其用在 CEC2017 复杂函数上实验, 并将其结果与 9 个代表性的先进算法 COA、GWO、MEGWO、HFPSO、DEBBO、SaDE、SE04、FWA 和 TLBO 的实验结果进行比较, 其结果见表 4, 其中 SaDE 和 SE04 的实验数据直接来自文献 [20].

在均值和方差上, 与 MEGWO 相比, HCOAG 优于 23 次. 与同为混合算法的 HFPSO 和 DEBBO 相比, HCOAG 分别优于 29 次和 23 次. 与 SaDE 和 SE04 相比, HCOAG 分别优于 28 次和 29 次. 与 FWA 和 TLBO 相比, HCOAG 分别优于 30 次和 29 次. 在总排名上, HCOAG 排名第一, 接下来依次为 MEGWO、DEBBO、SaDE、SE04、COA、TLBO、HFPSO、FWA、GWO. 因此, HCOAG 的性能优于 9 个先进对比算法的性能.

4.5　收敛性能分析

为了验证 HCOAG 的收敛性能, 给出 HCOAG 与 COA、MEGWO、DEBBO、TLBO 和 HFPSO 在 CEC2017 测试集中 30 维函数的收敛图. 受篇幅所限, 本文仅选取有代表性的函数作为示例进行对比分析. 在 $F_1\sim F_3$ 中选取 1 个单峰函数, 即 F_1, 在 $F_4\sim F_{10}$ 中选取 F_5, F_7, F_8 这 3 个多峰函数, 在 $F_{11}\sim F_{20}$ 中选取 F_{11}, F_{12}, F_{16}, F_{17} 这 4 个混合函数, 在 $F_{21}\sim F_{30}$ 中选取 F_{21}, F_{23}, F_{24}, F_{29} 这 4 个复合函数, 其收敛图见图 6.

从图 6 中可以看出, 在 3 个函数 F_1, F_5 和 F_8 上, 随着函数评价次数的增加, 在收敛速度上, HCOAG 较对比算法要快得多, 优势明显. 在其余

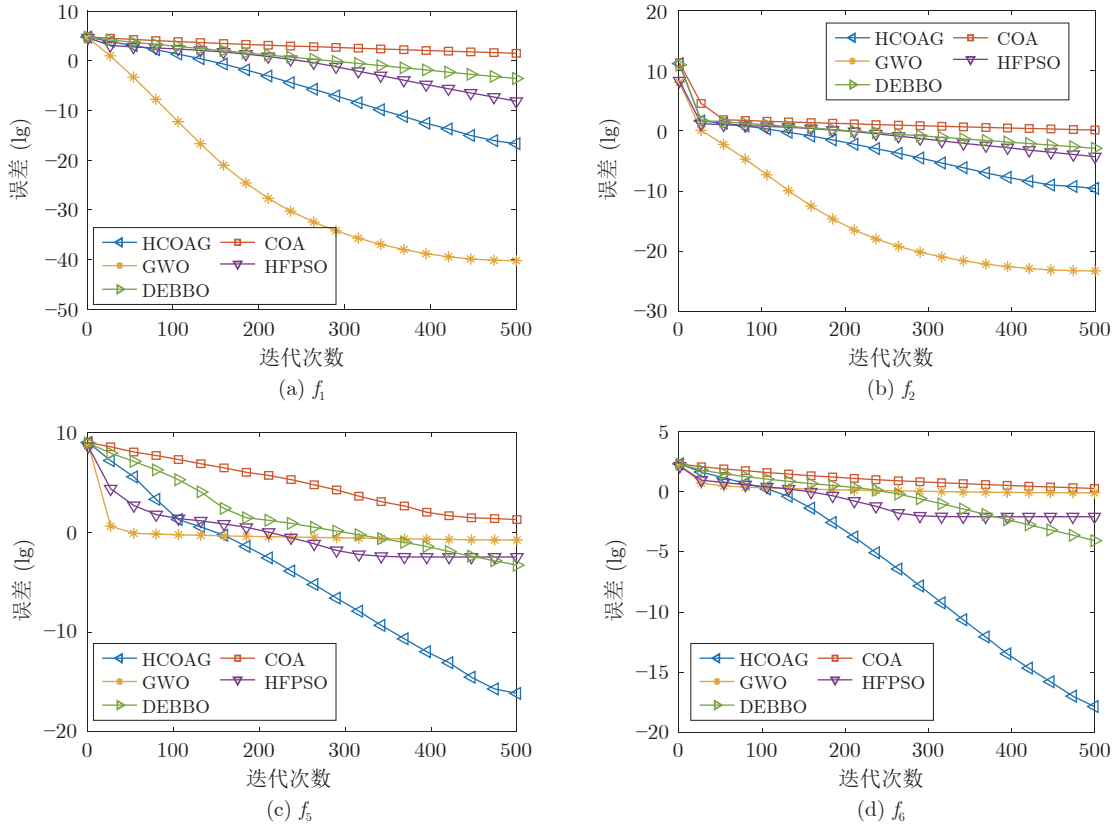

图 5 HCOAG 与对比算法在 4 个经典函数上的收敛图

Fig. 5 Convergence curves of HCOAG and the comparison algorithms on the 4 classical benchmark functions

表 4 在 30 维 CEC2017 复杂函数上的优化结果对比

Table 4 Comparison results on the 30-dimensional complex functions from CEC2017

函数	标准	HCOAG	COA	GWO	MEGWO	HFPSO	DEBBO	SaDE	SE04	FWA	TLBO
	Mean	$\mathbf{7.4494\times10^{-4}}$	1.2099×10^{3}	1.2813×10^{9}	4.5517×10^{3}	3.9338×10^{3}	2.7849×10^{3}	3.0714×10^{3}	3.2930×10^{3}	4.3987×10^{6}	2.9846×10^{3}
F_1	Std	$\mathbf{1.4801\times10^{-3}}$	1.2998×10^{3}	9.6388×10^{8}	1.0677×10^{3}	5.3689×10^{3}	4.0364×10^{3}	3.5072×10^{3}	4.2328×10^{3}	1.4055×10^{6}	3.1471×10^{3}
	Rank	1	2	10	8	7	3	5	6	9	4
	Mean	1.1941×10^{1}	2.9013×10^{21}	3.1831×10^{32}	2.8884×10^{8}	5.3485×10^{4}	1.5057×10^{17}	$\mathbf{8.6275\times10^{-1}}$	3.0802×10^{13}	4.1397×10^{15}	1.0448×10^{16}
F_2	Std	2.4077×10^{1}	1.1462×10^{22}	1.5894×10^{33}	8.0571×10^{8}	3.2847×10^{5}	3.5179×10^{17}	$\mathbf{4.9357\times10^{0}}$	1.1694×10^{14}	1.5680×10^{16}	4.7082×10^{16}
	Rank	2	9	10	4	3	8	1	5	6	7
	Mean	9.5410×10^{-1}	6.0573×10^{4}	2.8342×10^{4}	2.2633×10^{2}	$\mathbf{1.5595\times10^{-7}}$	3.6772×10^{4}	3.0045×10^{2}	9.7974×10^{3}	2.4748×10^{4}	4.0488×10^{-4}
F_3	Std	1.9288×10^{0}	1.0177×10^{4}	9.2323×10^{3}	1.7031×10^{2}	$\mathbf{2.3334\times10^{-7}}$	5.8394×10^{3}	7.3017×10^{2}	3.4377×10^{3}	6.3467×10^{3}	1.6647×10^{-3}
	Rank	3	10	8	4	1	9	5	6	7	2
	Mean	$\mathbf{1.8113\times10^{1}}$	8.4041×10^{1}	2.0825×10^{2}	2.4815×10^{1}	6.9386×10^{1}	8.4851×10^{1}	6.0423×10^{1}	8.5881×10^{1}	1.1370×10^{2}	5.9054×10^{1}
F_4	Std	$\mathbf{2.7696\times10^{1}}$	8.5306×10^{0}	8.4445×10^{1}	2.8995×10^{1}	2.1364×10^{1}	2.2848×10^{-1}	2.9825×10^{1}	1.1251×10^{1}	1.7315×10^{1}	3.0429×10^{1}
	Rank	1	6	10	2	5	7	4	8	9	3
	Mean	$\mathbf{2.8433\times10^{1}}$	5.2890×10^{1}	9.6116×10^{1}	5.6912×10^{1}	8.5624×10^{1}	5.8216×10^{1}	5.6192×10^{1}	4.1688×10^{1}	1.8456×10^{2}	8.5717×10^{1}
F_5	Std	$\mathbf{6.8886\times10^{0}}$	1.5025×10^{1}	3.2690×10^{1}	1.0725×10^{1}	1.7427×10^{1}	6.5957×10^{0}	1.4216×10^{1}	8.1545×10^{0}	3.3933×10^{1}	1.8601×10^{1}
	Rank	1	3	9	5	7	6	4	2	10	8
	Mean	1.7483×10^{-7}	1.6399×10^{-5}	6.3664×10^{0}	2.4470×10^{-1}	1.0170×10^{0}	$\mathbf{1.1369\times10^{-13}}$	8.9317×10^{-2}	7.5481×10^{-6}	5.1770×10^{0}	7.2903×10^{0}
F_6	Std	4.7524×10^{-7}	9.6428×10^{-6}	3.1596×10^{0}	8.1620×10^{-2}	2.3644×10^{0}	**0**	1.3955×10^{-1}	3.9880×10^{-5}	3.1351×10^{0}	4.4538×10^{0}
	Rank	2	4	9	6	7	1	5	3	8	10

表 4 在 30 维 CEC2017 复杂函数上的优化结果对比 (续表)

Table 4 Comparison results on the 30-dimensional complex functions from CEC2017 (continued table)

函数	标准	HCOAG	COA	GWO	MEGWO	HFPSO	DEBBO	SaDE	SE04	FWA	TLBO
	Mean	**6.1055×10^{1}**	7.5148×10^{1}	1.4460×10^{2}	8.9106×10^{1}	1.0407×10^{2}	9.9725×10^{1}	9.4945×10^{1}	7.2448×10^{1}	2.0998×10^{2}	1.3661×10^{2}
F_7	Std	**1.0851×10^{1}**	1.3762×10^{1}	4.6314×10^{1}	1.0935×10^{1}	1.9791×10^{1}	6.4285×10^{0}	1.9879×10^{1}	7.3495×10^{0}	4.4817×10^{1}	2.4846×10^{1}
	Rank	1	3	9	4	7	6	5	2	10	8
	Mean	**3.2489×10^{1}**	5.6110×10^{1}	8.4662×10^{1}	5.9398×10^{1}	7.2842×10^{1}	5.9299×10^{1}	5.3942×10^{1}	4.4194×10^{1}	1.4508×10^{2}	7.1339×10^{1}
F_8	Std	**1.2272×10^{1}**	1.8774×10^{1}	2.5270×10^{1}	1.0663×10^{1}	1.7967×10^{1}	6.0788×10^{0}	1.2792×10^{1}	6.5834×10^{0}	2.1470×10^{1}	1.4852×10^{1}
	Rank	1	4	9	6	8	5	3	2	10	7
	Mean	2.7362×10^{-1}	5.6225×10^{-1}	5.5392×10^{2}	7.8267×10^{0}	3.2733×10^{1}	**4.0125×10^{-14}**	8.3556×10^{1}	3.0839×10^{-1}	3.5295×10^{3}	2.4197×10^{2}
F_9	Std	4.8298×10^{-1}	1.0209×10^{0}	3.2695×10^{2}	1.1815×10^{1}	1.3249×10^{2}	**5.4870×10^{-14}**	6.2643×10^{1}	8.4139×10^{-1}	9.5511×10^{2}	1.4491×10^{2}
	Rank	2	4	9	5	6	1	7	3	10	8
	Mean	**2.2671×10^{3}**	2.7575×10^{3}	3.1862×10^{3}	2.4369×10^{3}	2.9908×10^{3}	3.2911×10^{3}	2.3253×10^{3}	2.3267×10^{3}	3.7800×10^{3}	6.0667×10^{3}
F_{10}	Std	**6.1427×10^{2}**	4.6685×10^{2}	9.7886×10^{2}	4.4542×10^{2}	5.9210×10^{2}	2.7284×10^{2}	4.9247×10^{2}	2.8457×10^{2}	5.9660×10^{2}	1.0625×10^{3}
	Rank	1	5	7	4	6	8	2	3	9	10
	Mean	**2.1678×10^{1}**	4.1143×10^{1}	4.9771×10^{2}	2.9612×10^{1}	1.1553×10^{2}	3.7430×10^{1}	1.0032×10^{2}	4.1343×10^{1}	1.6164×10^{2}	1.2672×10^{2}
F_{11}	Std	**2.0907×10^{1}**	2.7367×10^{1}	6.4235×10^{2}	1.0347×10^{1}	3.9628×10^{1}	2.3672×10^{1}	4.3101×10^{1}	2.7994×10^{1}	4.5263×10^{1}	4.5717×10^{1}
	Rank	1	4	10	2	7	3	6	5	9	8
	Mean	**9.8943×10^{3}**	1.2532×10^{5}	4.0285×10^{7}	1.5983×10^{4}	9.9670×10^{4}	1.3866×10^{5}	6.8629×10^{4}	1.1143×10^{6}	4.6351×10^{6}	3.3042×10^{4}
F_{12}	Std	**6.0932×10^{3}**	1.2555×10^{5}	7.3849×10^{7}	4.0434×10^{3}	1.0658×10^{5}	9.2097×10^{4}	3.8252×10^{4}	8.1422×10^{5}	3.1121×10^{6}	2.8646×10^{4}
	Rank	1	6	10	2	5	7	4	8	9	3
	Mean	1.9749×10^{3}	2.0357×10^{4}	2.8073×10^{6}	**2.0450×10^{2}**	3.0927×10^{4}	8.1265×10^{3}	1.1211×10^{4}	4.6063×10^{3}	3.7320×10^{4}	1.4857×10^{4}
F_{13}	Std	3.8565×10^{3}	2.6333×10^{4}	1.6225×10^{7}	**2.7028×10^{1}**	2.7301×10^{4}	7.8066×10^{3}	1.0535×10^{4}	4.8590×10^{3}	2.6480×10^{4}	1.7072×10^{4}
	Rank	2	7	10	1	8	4	5	3	9	6
	Mean	8.6436×10^{1}	8.0070×10^{1}	1.3112×10^{5}	**6.1985×10^{1}**	6.7377×10^{3}	4.9240×10^{3}	4.3238×10^{3}	7.1204×10^{4}	2.6955×10^{5}	3.5454×10^{3}
F_{14}	Std	4.3766×10^{1}	1.9915×10^{1}	2.3335×10^{5}	**8.6647×10^{0}**	5.5695×10^{3}	3.2902×10^{3}	5.7159×10^{3}	5.9323×10^{4}	2.4525×10^{5}	4.1276×10^{3}
	Rank	3	2	9	1	7	6	5	8	10	4
	Mean	1.8396×10^{3}	2.0792×10^{3}	3.3658×10^{5}	**5.1634×10^{1}**	9.7487×10^{3}	4.9944×10^{3}	2.1676×10^{3}	2.2013×10^{3}	3.2784×10^{3}	3.9091×10^{3}
F_{15}	Std	2.9044×10^{3}	7.9984×10^{3}	7.9125×10^{5}	**1.0713×10^{1}**	1.2114×10^{4}	6.6468×10^{3}	3.0178×10^{3}	1.9756×10^{3}	1.9819×10^{3}	4.3347×10^{3}
	Rank	2	3	10	1	9	8	4	5	6	7
	Mean	**3.0243×10^{2}**	7.9869×10^{2}	8.1416×10^{2}	4.4823×10^{2}	7.7229×10^{2}	3.9643×10^{2}	5.6072×10^{2}	4.9392×10^{2}	1.2266×10^{3}	5.0039×10^{2}
F_{16}	Std	**2.0550×10^{2}**	2.8651×10^{2}	2.6440×10^{2}	1.3443×10^{2}	2.2590×10^{2}	1.1932×10^{2}	2.0850×10^{2}	1.7309×10^{2}	3.0034×10^{2}	2.7575×10^{2}
	Rank	1	8	9	3	7	2	6	4	10	5
	Mean	**4.7111×10^{1}**	2.2439×10^{2}	2.7004×10^{2}	6.9544×10^{1}	2.5591×10^{2}	8.1642×10^{1}	8.7684×10^{1}	1.4116×10^{2}	5.5825×10^{2}	2.3994×10^{2}
F_{17}	Std	**4.0925×10^{1}**	1.3518×10^{2}	1.3820×10^{2}	1.7296×10^{1}	1.2971×10^{2}	2.2037×10^{1}	9.1289×10^{1}	8.5026×10^{1}	2.3401×10^{2}	8.8703×10^{1}
	Rank	1	6	9	2	8	3	4	5	10	7
	Mean	6.1013×10^{4}	6.9910×10^{4}	7.1643×10^{5}	**2.0505×10^{2}**	1.1409×10^{5}	3.2225×10^{5}	1.0034×10^{5}	2.1361×10^{5}	9.8409×10^{5}	2.0609×10^{5}
F_{18}	Std	5.7031×10^{4}	1.0210×10^{5}	8.2799×10^{5}	**4.7536×10^{1}**	1.1535×10^{5}	1.2197×10^{5}	1.1019×10^{5}	1.3261×10^{5}	1.1184×10^{6}	1.5131×10^{5}
	Rank	2	3	9	1	5	8	4	7	10	6
	Mean	3.4042×10^{1}	4.4886×10^{3}	4.6400×10^{5}	**2.9977×10^{1}**	8.6631×10^{3}	8.3686×10^{3}	5.9612×10^{3}	2.0723×10^{3}	5.2207×10^{3}	6.3203×10^{3}
F_{19}	Std	2.0528×10^{1}	1.3325×10^{4}	5.4998×10^{5}	**3.3897×10^{0}**	1.9974×10^{4}	9.2795×10^{3}	7.1112×10^{3}	2.1685×10^{3}	3.9175×10^{3}	1.0793×10^{4}
	Rank	2	4	10	1	9	8	6	3	5	7
	Mean	9.6665×10^{1}	2.4290×10^{2}	3.6059×10^{2}	1.1363×10^{2}	2.6516×10^{2}	**5.5205×10^{1}**	1.2989×10^{2}	1.7303×10^{2}	4.6345×10^{2}	2.4392×10^{2}
F_{20}	Std	7.7834×10^{1}	1.4995×10^{2}	1.0264×10^{2}	5.2411×10^{1}	1.1737×10^{2}	**3.5413×10^{1}**	7.0970×10^{1}	7.2015×10^{1}	1.7129×10^{2}	8.4432×10^{1}
	Rank	2	6	9	3	8	1	4	5	10	7
	Mean	**2.3023×10^{2}**	2.5626×10^{2}	2.8298×10^{2}	2.5458×10^{2}	2.7446×10^{2}	2.5950×10^{2}	2.4896×10^{2}	2.5047×10^{2}	3.7768×10^{2}	2.6988×10^{2}
F_{21}	Std	**8.5095×10^{0}**	1.6800×10^{1}	2.5686×10^{1}	3.3247×10^{1}	1.9517×10^{1}	7.6690×10^{0}	1.3195×10^{1}	8.4442×10^{0}	7.9462×10^{1}	1.9589×10^{1}
	Rank	1	5	9	4	8	6	2	3	10	7

表 4 在 30 维 CEC2017 复杂函数上的优化结果对比 (续表)

Table 4 Comparison results on the 30-dimensional complex functions from CEC2017 (continued table)

函数	标准	HCOAG	COA	GWO	MEGWO	HFPSO	DEBBO	SaDE	SE04	FWA	TLBO
	Mean	1.0010×10^{2}	1.9999×10^{3}	8.0434×10^{2}	1.0022×10^{2}	1.4532×10^{3}	$\mathbf{1.0000\times10^{2}}$	1.0228×10^{2}	1.0211×10^{3}	2.1380×10^{3}	1.0232×10^{2}
F_{22}	Std	4.8096×10^{-1}	1.5970×10^{3}	1.1113×10^{3}	4.3917×10^{-2}	1.8286×10^{3}	$\mathbf{2.3100\times10^{-13}}$	3.2279×10^{0}	1.2872×10^{3}	2.2149×10^{3}	4.0114×10^{0}
	Rank	2	9	6	3	8	1	4	7	10	5
	Mean	$\mathbf{3.7755\times10^{2}}$	4.1635×10^{2}	4.7029×10^{2}	3.8959×10^{2}	4.8447×10^{2}	4.0323×10^{2}	4.1472×10^{2}	4.0247×10^{2}	5.8963×10^{2}	4.5003×10^{2}
F_{23}	Std	$\mathbf{1.0911\times10^{1}}$	1.6898×10^{1}	2.9324×10^{1}	6.8787×10^{1}	4.4709×10^{1}	5.6348×10^{0}	1.8742×10^{1}	8.1687×10^{0}	8.8792×10^{1}	3.0546×10^{1}
	Rank	1	6	8	2	9	4	5	3	10	7
	Mean	$\mathbf{4.4827\times10^{2}}$	5.4044×10^{2}	5.2489×10^{2}	4.8972×10^{2}	5.6079×10^{2}	4.7430×10^{2}	4.8169×10^{2}	4.9840×10^{2}	7.9489×10^{2}	5.0152×10^{2}
F_{24}	Std	$\mathbf{1.0757\times10^{1}}$	4.5778×10^{1}	3.3902×10^{1}	1.6597×10^{1}	5.7847×10^{1}	6.0055×10^{0}	2.0610×10^{1}	1.3899×10^{1}	8.6391×10^{1}	2.3700×10^{1}
	Rank	1	8	7	4	9	2	3	5	10	6
	Mean	3.8777×10^{2}	3.8706×10^{2}	4.7784×10^{2}	$\mathbf{3.8374\times10^{2}}$	3.8818×10^{2}	3.8691×10^{2}	4.0124×10^{2}	3.8779×10^{2}	4.1099×10^{2}	4.0877×10^{2}
F_{25}	Std	5.4462×10^{0}	8.0647×10^{-1}	2.3819×10^{1}	$\mathbf{1.8246\times10^{-1}}$	3.4076×10^{0}	7.5524×10^{-2}	1.9489×10^{1}	1.1319×10^{0}	2.1657×10^{1}	2.2401×10^{1}
	Rank	4	3	10	1	6	2	7	5	9	8
	Mean	1.2578×10^{3}	1.6520×10^{3}	2.0116×10^{3}	$\mathbf{2.5051\times10^{2}}$	1.4922×10^{3}	1.4821×10^{3}	1.7344×10^{3}	1.5337×10^{3}	2.2418×10^{3}	1.8645×10^{3}
F_{26}	Std	1.9807×10^{2}	1.7070×10^{2}	5.7618×10^{2}	$\mathbf{4.1112\times10^{1}}$	9.6940×10^{2}	7.2015×10^{1}	7.1347×10^{2}	1.9051×10^{2}	1.7373×10^{3}	1.0276×10^{3}
	Rank	2	6	9	1	4	3	7	5	10	8
	Mean	5.1091×10^{2}	5.0430×10^{2}	5.9279×10^{2}	5.1286×10^{2}	5.3523×10^{2}	$\mathbf{4.9807\times10^{2}}$	5.4289×10^{2}	5.0744×10^{2}	5.7919×10^{2}	5.3827×10^{2}
F_{27}	Std	7.7116×10^{0}	8.2707×10^{0}	3.8462×10^{1}	6.1632×10^{0}	2.1095×10^{1}	$\mathbf{4.7270\times10^{0}}$	1.7086×10^{1}	3.6242×10^{0}	3.5317×10^{1}	1.6907×10^{1}
	Rank	4	2	10	5	6	1	8	3	9	7
	Mean	3.3558×10^{2}	4.0555×10^{2}	5.9941×10^{2}	3.6492×10^{2}	3.5331×10^{2}	$\mathbf{3.2281\times10^{2}}$	3.3257×10^{2}	4.1364×10^{2}	4.6256×10^{2}	3.6806×10^{2}
F_{28}	Std	5.3866×10^{1}	3.6156×10^{1}	6.9788×10^{1}	3.2477×10^{1}	5.9179×10^{1}	$\mathbf{3.7880\times10^{1}}$	5.2165×10^{1}	2.5577×10^{1}	2.3601×10^{1}	5.3388×10^{1}
	Rank	3	7	10	5	4	1	2	8	9	6
	Mean	$\mathbf{4.5991\times10^{2}}$	6.6978×10^{2}	8.5036×10^{2}	5.4385×10^{2}	6.7006×10^{2}	5.1851×10^{2}	5.5826×10^{2}	5.4778×10^{2}	1.0120×10^{3}	7.8922×10^{2}
F_{29}	Std	$\mathbf{4.4075\times10^{1}}$	1.7459×10^{2}	1.8235×10^{2}	5.4241×10^{1}	1.4475×10^{2}	3.4859×10^{1}	1.0040×10^{2}	8.1960×10^{1}	2.1872×10^{2}	1.3560×10^{2}
	Rank	1	6	9	3	7	2	5	4	10	8
	Mean	$\mathbf{2.9823\times10^{3}}$	6.0618×10^{3}	4.0643×10^{6}	3.6855×10^{3}	1.8733×10^{4}	5.9405×10^{3}	5.0147×10^{3}	4.9671×10^{3}	1.5965×10^{4}	5.9572×10^{3}
F_{30}	Std	$\mathbf{6.1135\times10^{2}}$	4.7022×10^{3}	3.1688×10^{6}	3.3042×10^{2}	3.4470×10^{4}	2.3158×10^{3}	1.9712×10^{3}	2.0934×10^{3}	8.7877×10^{3}	3.9139×10^{3}
	Rank	1	7	10	2	9	5	4	3	8	6
Count		15	0	0	7	1	6	1	0	0	0
Ave rank		1.73	5.27	9.10	3.17	6.67	4.37	4.53	4.63	9.03	6.50
Total rank		1	6	10	2	8	3	4	5	9	7

9 个函数上, 虽然 HCOAG 的收敛速度优势不是很明显, 但在收敛性能上也优于其他对比算法. 总之, 无论在单峰函数和多峰函数上, 还是在混合函数和复合函数上, HCOAG 的收敛速度都比其他对比算法的收敛速度快, 表明 HCOAG 具有更优秀的收敛性能. 这些都说明, 本文提出的高斯全局趋优成长算子、动态调整组内郊狼数方案以及简化操作的 GWO 搜索算子等的采用都为收敛速度的提升做出了贡献, 这些策略的融合使用是有效可行的.

4.6 时间对比分析

为了考察 HCOAG 的运行时间, 直接采用第 4.4 节的实验. 因为 HCOAG 是 COA 和 GWO 的混合算法, 故仅记录 HCOAG、COA 和 GWO 在 30 维 CEC2017 函数上的耗时, 它们在独立运行 30 次获得不同函数类型的平均耗时结果见图 7. 横坐标为不同的函数类型, 纵坐标为平均时间 (s).

从图 7 可以看出, 在单峰函数上, HCOAG 耗时 (2.4443 s) 是 COA 耗时 (2.9988 s) 的 81.51%, GWO 耗时 (2.4222 s) 是 HCOAG 耗时 (2.4443 s) 的 99.10%; 在多峰函数上, HCOAG 耗时 (2.9211 s) 分别是 COA(3.5503 s) 和 GWO(2.9233 s) 耗时的 84.25% 和 99.92%; 在混合函数上, HCOAG 耗时 (3.6246 s) 是 COA 耗时 (4.2034 s) 的 86.23%, GWO 耗时 (3.4981 s) 是 HCOAG 耗时 (3.6246 s) 的 96.51%; 在复合函数上, HCOAG 耗时 (6.1480 s) 是 COA 耗时 (7.0897 s) 的 86.72%, GWO 耗时 (6.0822 s) 是 HCOAG 耗时 (6.1480 s) 的 98.93%.

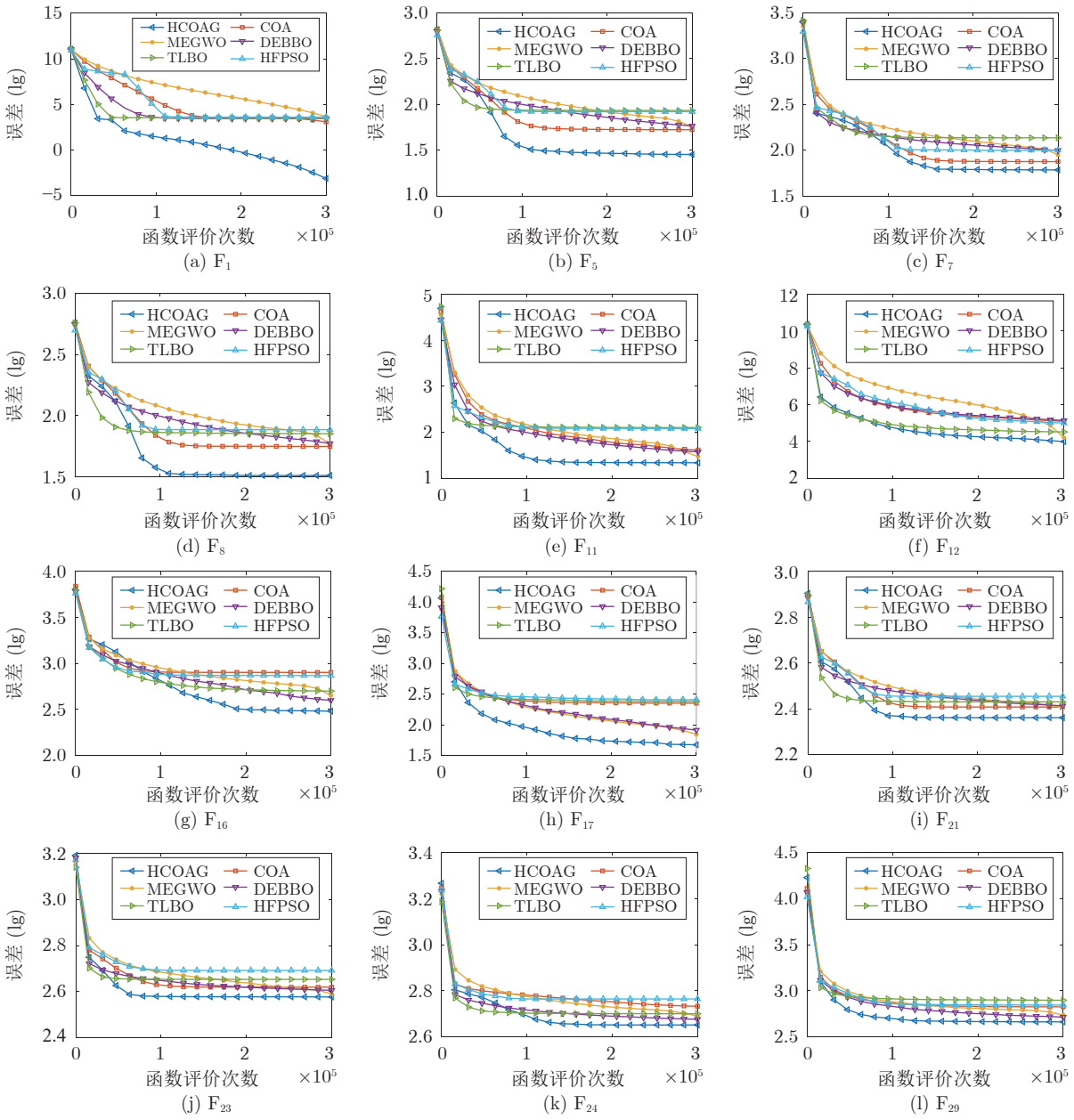

图 6　HCOAG、COA、MEGWO、DEBBO、TLBO 和 HFPSO 的收敛图

Fig. 6　Convergence curves of HCOAG, COA, MEGWO, DEBBO, TLBO, and HFPSO

故无论是单峰函数和多峰函数, 还是混合函数和复合函数, HCOAG 的耗时都与 GWO 耗时大致持平. 其主要原因是: 1) HCOAG 与 GWO 都采用并行计算模式, 但 GWO 结构简单, 而其产生新解的计算复杂度 (见式 (12)) 较高; 2) HCOAG 虽然采用精简 GWO, 但需要分组并在组内寻优, 结构复杂, 故二者耗时大致持平 (HCOAG 耗时稍多于 GWO 的耗时). 在耗时上, HCOAG 较 COA 少, 主要原因是: 1) 在成长过程的目标函数评价上, 后者采用串行计算, 前者采用并行计算减少了运行时间; 2) 虽然在成长过程中嵌入了 GWO 搜索, 但采用精简的 GWO 搜索方式并未增加多少计算成本; 3) 前者没有郊狼被驱离和接纳操作, 节省了运行时间; 4) 动态调整方案减少了生与死操作次数.

综合第 4.3 ~ 4.6 节, HCOAG 在较短的时间内能获得最好的优化性能, 说明了其优化效率高.

4.7　上下界分析

为了能够更充分地评价 HCOAG 的性能, 本节对其进行上下界分析, 即在一个优化问题上, 独立运行一定次数后考察其最坏结果和最优结果.

因为本文将 HCOAG 应用于 CEC2017 复杂函

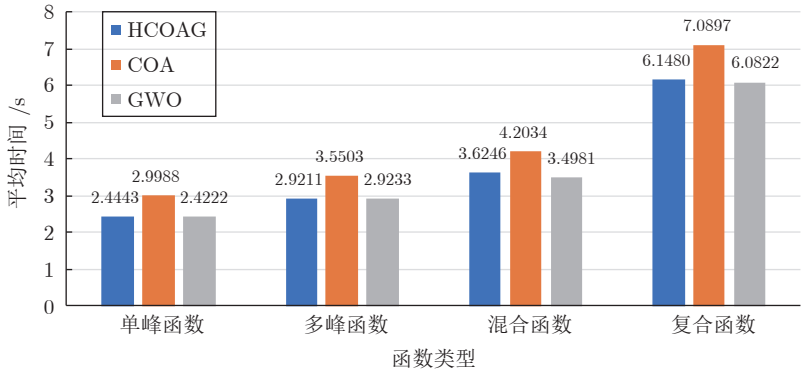

图 7 HCOAG 与 COA、GWO 在不同类别函数上的平均时间对比图

Fig. 7 Comparison bars of average time of HCOAG, COA, and GWO on different kinds of functions

数集上, 此函数集有 30 个不同的优化问题, 其全局最优解对应的最优值不同, 故其获得的上下界也不同. 限于篇幅, 在 CEC2017 四类函数上分别选取了同样的一个函数 (F_1, F_5, F_{11}, F_{29}) 进行上下界讨论, 即在 51 次的运行结果中选择最好值和最差值分别作为本文提出算法在该函数上的上下界, 并与对比算法中结果最好的算法 (在 F_1 和 F_5 上选用 COA; 在 F_{11} 和 F_{29} 上选用 DEBBO) 获得的上下界进行对比.

为了直观和方便对比, 对 3 个算法获得的结果进行了统一的处理: 即将每个算法每次独立运行结果减去理想的最优函数值作为最终结果, 让每个函数的理想最优值为 0. 在 4 个函数上的上下界结果见表 5.

表 5 在 30 维 CEC2017 复杂函数上的上下界结果对比

Table 5 Comparison of upper and lower bounds on the 30-dimensional complex functions from CEC2017

函数	HCOAG		COA/DEBBO	
	下界	上界	下界	上界
F_1	3.8942×10^{-9}	7.8898×10^{-3}	3.5001×10^{0}	5.0055×10^{3}
F_5	1.2935×10^{1}	4.1788×10^{1}	2.6865×10^{1}	9.2083×10^{1}
F_{11}	4.0954×10^{0}	7.5899×10^{1}	1.3819×10^{1}	8.8108×10^{1}
F_{29}	3.7200×10^{2}	6.0977×10^{2}	4.4500×10^{2}	6.2345×10^{2}

从表 5 可以看出, 在 4 个函数上, 无论上界还是下界, HCOAG 都比对比算法小, 说明本文提出的算法比对比算法好, 也证明了本文提出算法的有效性.

4.8 渐进收敛性分析

依据文献 [23] 的随机泛函分析方法原理, 本节对 HCOAG 的收敛性进行分析. 从算法 3 的描述和 HCOAG 流程图 (图 4) 可以看出, HCOAG 由 3 个算子完成新解的产生, 即高斯全局趋优成长算子、简化 GWO 搜索算子以及生与死算子. 高斯全局趋优成长算子类似 DE (Differential evolution) 中的 DE/rand-to-best/1/bin 变异算子; 简化 GWO 搜索算子类似 DE 中的 DE/best/1/bin 变异算子; 生与死算子相当于遗传算法中的变异算子. 3 个算子产生的新解都采用贪心算法严格基于优胜劣汰策略, 通过淘汰新解与原解中较差者产生更优的新一代个体. 故对于最小优化问题, HCOAG 评价函数序列为单调非递增序列. 为了证明 HCOAG 的渐近收敛性, 首先做如下定义.

定义 1. 设 $\boldsymbol{Q}(t)$ 表示 $\boldsymbol{X}(t)$ 的中间群体, $\boldsymbol{V}_{c,j}(t+1)\in\boldsymbol{Q}(t+1)$, 则 HCOAG 搜索算子定义如下:

高斯全局趋优成长与简化 GWO 搜索混合算子定义为

$$\boldsymbol{V}_{c,j}(t+1)=\begin{cases}\dfrac{\boldsymbol{NX}_{1,j}+\boldsymbol{NX}_{2,j}+\boldsymbol{NX}_{3,j}}{3}, & rand<CR\\ \boldsymbol{X}_{c,j}+rn_1\times\boldsymbol{\delta}_{3,j}+rn_2\times\boldsymbol{\delta}_{2,j}, & rand\geq CR\end{cases}\tag{31}$$

生与死算子定义为

$$\boldsymbol{V}_j(t+1)=\begin{cases}\boldsymbol{X}_{cr_1,j}, & r_j<P_s\ or\ j=j_1\\ \boldsymbol{X}_{cr_2,j}, & r_j\geq P_s+P_a\ 或\ j=j_2\\ R_j, & 其他\end{cases}\tag{32}$$

假设 $f(\boldsymbol{X})$为最小优化函数, 解空间 $\boldsymbol{S}=\boldsymbol{X}|\boldsymbol{X}=\{x_1,x_2,\cdots,x_D\}$ 且 $L_j\leq x_j\leq U_j,\ j=1,2,\cdots,D\}$ 为 D 维欧氏空间 $\mathbf{R}^D$ 的有界子空间. 为了在进行数值计算时不受计算精度的限制, 设定 HCOAG 的计算精度保留到小数点后第 k 位数字.

定义 2. 两种算子的混合策略是一种按照概率 CR/D 和 $(1-CR/D)$ 对群体中个体向量的每一维分量分别采用简化 GWO 搜索算子和高斯全局趋优成长算子以及在每一组中最差郊狼上采用生与死

算子进行重组变换的过程, 它是解空间上的一种随机映射 $\boldsymbol{\psi}: \Omega \times \boldsymbol{S} \rightarrow \boldsymbol{S}^2$, 可定义为

$$\mu\{\omega \,|\boldsymbol{\psi}(\omega, \boldsymbol{X}) = \langle \boldsymbol{X}, \boldsymbol{V}\rangle\} = \mu\{\boldsymbol{V} = F(\boldsymbol{X}, \boldsymbol{X}_c, \boldsymbol{X}_j, \boldsymbol{X}_k, \boldsymbol{X}_{\text{mod}})\} \tag{33}$$

其中, (Ω, M, μ) 是完全概率测度空间, Ω 是非空抽象集合, 其元素 ω 为基本事件, M 是 Ω 的某些子集所构成的 σ 代数, μ 是 M 上的概率测度. X_{mod} 表示郊狼为 $\boldsymbol{GP}$, $\boldsymbol{alpha}$ 或 $\boldsymbol{cult}$, $F(\boldsymbol{X}, \boldsymbol{X}_c, \boldsymbol{X}_j, \boldsymbol{X}_k, \boldsymbol{X}_{\text{mod}})$ 由式 (31) 和式 (32) 确定.

由于 HCOAG 与 DE 一样采用贪心算法更新种群, 因此在每次迭代过程中相当于 3 种算子所对应的随机映射 $\boldsymbol{\psi}$ 和贪心选择所对应的映射逆序 $\boldsymbol{\psi}_2$ 合成的映射 $\boldsymbol{\psi}'$ 作用于当前群体, 即 $\boldsymbol{X}(t+1) = \boldsymbol{\psi}'(\omega, \boldsymbol{X}(t)) = \boldsymbol{\psi}_2(\boldsymbol{\psi}(\omega, \boldsymbol{X}(t)))$. 这样在映射 $\boldsymbol{\psi}'$ 的作用下, HCOAG 每次迭代种群产生的 $\boldsymbol{GP}$ 的适应度值构成的序列 $\{f(X_{GP}(t))\}_{1 \le t \le MaxDT}$ 是一个单调非递增序列. 同样, $\boldsymbol{\psi}'$ 可重新定义为 $\boldsymbol{B}_{t+1} = \boldsymbol{\psi}'(\omega, \boldsymbol{B}_t) = \boldsymbol{\psi}_2(\boldsymbol{\psi}(\omega, \boldsymbol{B}_t))$, $\boldsymbol{B}_{t+1}$ 和 $\boldsymbol{B}_t$ 分别为 $\boldsymbol{X}(t+1)$ 和 $\boldsymbol{X}(t)$ 的最优个体. 因此类似于文献 [23]中的定理 1, 可得到如下定理.

定理 1. HCOAG 的一次迭代所形成的随机映射 $\boldsymbol{\psi}'$ 是一个随机压缩算子.

再依据文献 [23] 引理 2, 即可得到 HCOAG 渐进收敛的结论, 即定理 2.

定理 2. 设 $\boldsymbol{\psi}'$ 为 HCOAG 形成的随机压缩算子, 则 $\boldsymbol{\psi}'$ 具有唯一的随机不动点, 即 HCOAG 是渐进收敛的.

4.9 Wilcoxon 符号秩检验和 Friedman 检验

Wilcoxon 符号秩检验方法[24] 是一种非参数统计性检验方法, 目的在于检验两个样本均值之间的显著差异. 其中 p 值由 R^+ 和 R^- 计算可得, R^+ 表示正秩总和, R^- 表示负秩总和, 具体见式 (34) 和式 (35), d_i 为两种算法在 n 个问题中第 i 个问题上的性能分数的差值. 当 HCOAG 算法与对比算法性能相同时, 对应的秩平分给 R^+ 和 R^-. 为检验 HCOAG 与表 4 中对比算法的显著性差异, 在软件 IBM SPSS Statistics 24 上实现 Wilcoxon 检验, 结果如表 6 所示. 表 6 中, $n/w/t/l$ 表示在 n 个函数上 HCOAG 分别优于对比算法 w 次, 与对比算法相等 t 次, 劣于对比算法 l 次.

$$R^+ = \sum_{d_i>0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i) \tag{34}$$

$$R^- = \sum_{d_i<0} \text{rank}(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}(d_i) \tag{35}$$

从表 6 可以看出, 与 COA 和 GWO 相比, p 值分别为 1.3039×10^{-7} 和 1.8626×10^{-9}, 均小于 0.05, 表明 HCOAG 显著优于 COA 和 GWO, 再一次说明 COA 和 GWO 二者的改进与混合是有效的. HCOAG 与 MEGWO、HFPSO、DEBBO、SaDE、SE04、FWA 和 TLBO 相比的 p 值均小于 0.05, 表明 HCOAG 也显著优于这些对比算法.

Friedman 检验是一种非参数双向方差分析方法[24], 目的在于检测两个或多个观测数据之间的显著性差异. 具体实现过程分为 3 步: 1) 收集每个算法或者问题的观测结果; 2) 对于每个问题 i 的从 1 (最好结果) 到 k (最差结果) 的排名值, 定义为 r_i^j $(1 \le j \le k)$; 3) 在所有问题中求出每个算法的平均排名, 得到最后排名 $R_j = \frac{1}{n}\sum_{i=1}^{n} r_i^j$. 在零假设下所有算法的行为相似 (它们的秩 R_j 相等), Friedman 统计值 F_f 的计算方式如式 (36), 当 n 和 k 足够大时 (根据经验 $n > 10$, $k > 5$), 它是按照 $k-1$ 自由度的 x^2 分布的. 为了再次验证 HCOAG 的显著性, 依据表 4 对 HCOAG 和对比算法在软件 IBM

表 6　Wilcoxon 符号秩检验结果

Table 6　Wilcoxon sign rank test results

	p	$a = 0.05$	R^+	R^-	$n/w/t/l$
HCOAG vs COA	1.3039×10^{-7}	YES	453	12	30/27/0/3
HCOAG vs GWO	1.8626×10^{-9}	YES	465	0	30/30/0/0
HCOAG vs MEGWO	2.7741×10^{-2}	YES	339	126	30/23/0/7
HCOAG vs HFPSO	5.5879×10^{-9}	YES	463	2	30/29/0/1
HCOAG vs DEBBO	9.0000×10^{-6}	YES	429	36	30/23/0/7
HCOAG vs SaDE	3.5390×10^{-8}	YES	458	7	30/28/0/2
HCOAG vs SE04	1.3039×10^{-8}	YES	461	4	30/29/0/1
HCOAG vs FWA	1.8626×10^{-9}	YES	465	0	30/30/0/0
HCOAG vs TLBO	3.7253×10^{-9}	YES	464	1	30/29/0/1

SPSS Statistics 24 上实现 Friedman 检验, 其结果如表 7 所示.

$$F_f = \frac{12n}{k\,(k+1)}\left[\sum\nolimits_j R_j^2 - \frac{k(k+1)^2}{4}\right] \tag{36}$$

从表 7 可以看出, 在 30 维复杂函数上, 通过 Friedman 检验获得的渐进显著性 p 值为 6.3128×10^{-31}, 由于得到的渐进显著性小于 0.01, 所以在 30 维上 HCOAG 与对比算法之间存在显著性的差异. HCOAG 算法的秩均值 (1.73) 最小, 随后依次是 MEGWO、DEBBO、SaDE、SE04、COA、TLBO、HFPSO、FWA 和 GWO, 表明 HCOAG 的优化性能最好. 结合 Wilcoxon 符号秩检验和 Friedman 检验结果可以得出, HCOAG 总体上明显优于先进的对比算法.

5 HCOAG 在 K-Means 聚类优化上的应用

聚类在数据挖掘领域发挥着十分重要的作用, 通过对聚类的数据分析可以得到数据的具体分布情况以及掌握数据类型的特点[25]. 其中, 最常用的聚类方法是 K-Means 方法[26], 其在 n 个样本数据中的具体实现过程如下: 首先随机产生 K 个初始聚类中心, 然后计算每个样本与各聚类中心的距离, 接着将样本与距离最近的聚类中心划分到一个组内, 形成 K 个组, 最后重新计算每个组内所有样本的均值作为新聚类中心并进行下一次划分, 如此迭代执行划分过程直到满足终止条件为止. K-Means 方法具有原理简单、可伸缩性好以及效率高等优势, 但存在对初始点敏感和易陷于局部最优等问题. 目前已有研究将 SIOA 运用到 K-Means 聚类中, 很好地解决了 K-Means 算法存在的一些问题[25, 27]. 本节采用 HCOAG 优化 K-Means 聚类以解决其对初始点敏感等问题, 其中, 目标函数的定义为

$$E = \sum_{k=1}^{K}\sum_{x_i\in C_k}\|\boldsymbol{x}_i - \boldsymbol{c}_k\|^2 \tag{37}$$

其中, E 是数据集中所有数据点到所属聚类的聚类中心的距离和, K 是聚类中心个数, $\boldsymbol{c}_k$ 是第 k 个聚类中心位置, C_k 是第 k 个聚类簇, $\boldsymbol{x}_i$ 是聚类族 C_k 中第 i 个数据点. 将 HCOAG 应用到 K-Means上, 首先假设每头郊狼由 k 个聚类中心组成, 则解向量的维数应等于 $k\times$ 数据样本的特征数, 目标函数采用式 (37); 接着执行 HCOAG, 直到满足算法的终止条件, 输出最优聚类中心.

实验采用 UCI 数据库 (http://archive.ics.uci.edu/ml/datasets.php) 中 7 个标准数据集 (见表 8 第 1 列) 来验证 HCOAG 在 K-Means 聚类优化上的有效性, 数据集名称后的括号内的数字为样本数、属性数和聚类数. 选取的 5 个对比算法包括: COA、MEGWO、HFPSO、擅长聚类优化的改进的粒子群优化算法 (Improved PSO, IPSO) 和改进的遗传算法 (Improved genetic algorithm, IGA). 其中, IPSO 和 IGA 来自 "http://yarpizcom/64/ypml101-evolutionary-clustering". 所有算法的公共参数设置相同: $N = 50$, $MaxDT = 200$, $Num = 30$. 表 8 是 6 种算法独立运行 30 次获得的均值、方差和排名结果.

从表 8 可知, HCOAG 在 7 个数据集上的均值和方差得到排名第一 5 次, 其次是 HFPSO 和 IPSO 均为 1 次, COA、MEGWO 和 IGA 均获得 0 次第一. HCOAG 的平均排名为 1.43, 其他算法的平均排名顺序依次为: IPSO、IGA、MEGWO、HFPSO 和 COA. 以上结果都表明 HCOAG 在 K-Means 聚类上的优化性能最好. 总之, HCOAG 在 K-Means 聚类优化中获得竞争性的优化性能, 能够更好地处理聚类优化问题.

6 结束语

针对 COA 存在的不足, 本文提出了一种 COA 与 GWO 的混合算法 (HCOAG). 首先, 改进 COA (ICOA). 1) 在成长过程中, 提出一种高斯全局趋优成长算子, 提高了搜索能力; 2) 提出一种动态调整组数方案, 搜索前期采用较少组数, 减弱全局最优解的正反馈作用, 强化探索能力, 后期采用较多组数, 增强全局最优解的正反馈作用, 强化开采能力, 并提高可操作性. 然后, 对 GWO 进行改进, 提出了一种精简的 GWO (SGWO), 在发挥其局部搜索能力强的优势同时, 提高了可操作性. 最后, 采用正弦交叉策略将 ICOA 与 SGWO 有机融合, 很好地平衡了组内郊狼的探索与开采能力, 从而获得了最佳优化性能. 大量经典函数与 CEC2017复杂函数优化的实验结果表明, 在经典函数上, COA不及 GWO; 在复杂函数上, GWO 不及 COA; 而 HCOAG 在两

表 7 Friedman 检验结果

Table 7 Friedman test results

D	p	HCOAG	COA	GWO	MEGWO	HFPSO	DEBBO	SaDE	SE04	FWA	TLBO
30	6.3128×10^{-31}	1.73	5.27	9.10	3.17	6.67	4.37	4.53	4.63	9.03	6.50

表 8 6 种算法在 K-Means 聚类优化上的结果对比

Table 8 Comparison results of the 6 algorithms on K-Means clustering optimization

数据集		HCOAG	COA	MEGWO	HFPSO	IPSO	IGA
Wine (178, 13, 3)	Mean	**88.6271**	116.7307	91.5916	93.622	89.8617	89.564
	Std	$\mathbf{3.4479\times10^{-2}}$	2.9398×10^{0}	2.5237×10^{0}	6.7353×10^{0}	3.9148×10^{0}	2.0321×10^{0}
	Rank	**1**	6	4	5	3	2
Heart (270, 13, 2)	Mean	**283.7680**	295.3786	284.5731	284.7653	285.0072	284.4112
	Std	$\mathbf{3.9989\times10^{-3}}$	2.3404×10^{0}	2.3896×10^{-1}	2.3804×10^{0}	5.2425×10^{0}	2.1715×10^{0}
	Rank	**1**	6	3	4	5	2
Iris (150, 4, 3)	Mean	**29.2053**	31.0511	29.2659	29.3578	29.3578	29.2607
	Std	$\mathbf{8.8033\times10^{-2}}$	5.7768×10^{-1}	1.3448×10^{-1}	1.0048×10^{0}	1.0048×10^{0}	9.2414×10^{-2}
	Rank	1	6	3	4	4	2
Glass (214, 9, 6)	Mean	**55.0255**	75.4621	68.8816	62.7114	57.3102	60.8651
	Std	$\mathbf{2.2242\times10^{0}}$	2.1402×10^{0}	2.8975×10^{0}	3.7975×10^{0}	3.5444×10^{0}	3.5855×10^{0}
	Rank	**1**	6	5	4	2	3
Newthyroid (215, 5, 3)	Mean	**40.0538**	42.0033	40.4736	41.8087	40.8213	41.9155
	Std	$\mathbf{9.8154\times10^{-3}}$	7.6493×10^{-1}	4.0104×10^{-1}	2.8501×10^{0}	2.0086×10^{0}	2.8122×10^{0}
	Rank	**1**	6	2	4	3	5
Liver disorders (345, 6, 2)	Mean	90.3443	93.5344	90.3849	91.0246	**90.3365**	90.3698
	Std	3.8530×10^{-4}	1.0160×10^{0}	2.8148×10^{-2}	2.6310×10^{0}	$\mathbf{2.1424\times10^{-2}}$	2.0447×10^{-2}
	Rank	2	6	4	5	**1**	3
Balance (625, 4, 3)	Mean	356.1247	357.6041	356.502	**356.0165**	356.0802	356.4092
	Std	2.3618×10^{-1}	4.0943×10^{-1}	3.3785×10^{-1}	$\mathbf{1.5930\times10^{-1}}$	2.0303×10^{-1}	1.4966×10^{-1}
	Rank	3	6	5	**1**	2	4
Count		5	0	0	1	1	0
Ave rank		1.43	6.00	3.71	3.86	2.86	3.00
Total rank		1	6	4	5	2	3

类优化问题上都有更好的性能, 说明二者混合有必要和有效; 与其他先进的对比算法相比, HCOAG 具有更好的优化性能. K-Means 聚类优化结果表明, 与对比算法相比, HCOAG 获得了竞争性的优化性能.

总之, 与 COA 相比, HCOAG 具有如下优势: 1) 普适性强, 经典函数和复杂函数优化以及聚类优化 3 组实验结果表明, HCOAG 都有更好的表现; 2) 耗时少, 因此有更好的搜索效率; 3) 更好的收敛质量; 4) 更强的稳定性和鲁棒性, 5) 可操作性更强.

COA 是最近提出的一种 SIOA, 尚有许多地方需要探讨和完善, 本文仅是一种混合改进研究的尝试. 未来将进一步改进 COA, 对其进行深入的理论研究, 并拓展其应用领域.

References

1 Liu San-Yang, Jin An-Zhao. A co-evolutionary teaching-learning-based optimization algorithm for constrained optimization problems. *Acta Automatica Sinica*, 2018, **44**(9): 1690−1697
(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): 1690−1697)

2 Lv Bai-Quan, Zhang Jing-Jing, Li Zhan-Pei, Liu Ting-Zhang. Fuzzy particle swarm optimization based on filled function and transformation function. *Acta Automatica Sinica*, 2018, **44**(1): 74−86
(吕柏权, 张静静, 李占培, 刘廷章. 基于变换函数与填充函数的模糊粒子群优化算法. 自动化学报, 2018, **44**(1): 74−86)

3 Mirjalili S, Mirjalili S M, Lewis A. Grey wolf optimizer. *Advances in Engineering Software*, 2014, **69**: 46−61

4 Pierezan J, Coelho L D S. Coyote optimization algorithm: A new metaheuristic for global optimization problems. In: Proceedings of the 2018 IEEE Congress on Evolutionary Computation (CEC). Rio de Janeiro, Brazil, USA: IEEE, 2018.

5 Peng Xi-Yuan, Peng Yu, Dai Yu-Feng. Swarm intelligence theory and applications. *Acta Electronica Sinica*, 2003, **12A**(31): 1982−1988
(彭喜元, 彭宇, 戴毓丰. 群智能理论及应用. 电子学报, 2003, **12A**(31): 1982−1988)

6 Wolpert D H, Macready W G. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1997, **1**(1): 67−82

7 Zhang Xin-Ming, Wang Xia, Kang Qiang, Cheng Jin-Feng. Hybrid grey wolf optimizer with artificial bee colony and its application to clustering optimization. *Acta Electronica Sinica*, 2018, **46**(10): 2430−2442
(张新明, 王霞, 康强, 程金凤. GWO与ABC的混合优化算法及其聚类优化. 电子学报, 2018, **46**(10): 2430−2442)

8 Zhang X M, Kang Q, Cheng J F, Wang X. A novel hybrid algorithm based on biogeography-based optimization and grey wolf optimizer. *Applied Soft Computing*, 2018, **67**: 197−214

9 Teng Z J, Lv J L, Guo L W. An improved hybrid grey wolf optimization algorithm. *Soft Computing*, 2019, **23**(15): 6617−6631

10 Arora S, Singh H, Sharma M, Sharma S, Anand P. A new hybrid algorithm based on grey wolf optimization and crow search algorithm for unconstrained function optimization and feature selection. *IEEE Access*, 2019, **7**: 26343−26361

11 Long Wen, Wu Tie-Bin, Tang Ming-Zhu, Xu Ming, Cai Shao-Hong. Grey wolf optimizer 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)

12 Zhang Xin-Ming, Wang Dou-Dou, Chen Hai-Yan, Mao Wen-Tao, Dou Zhi, Liu Shang-Wang. Best and worst coyotes strengthened coyote optimization algorithm and its application to QAP. *Journal of Computer Applications*, 2019, **39**(10): 2985−2991
(张新明, 王豆豆, 陈海燕, 毛文涛, 窦智, 刘尚旺. 强化最优和最差狼的郊狼优化算法及其QAP应用. 计算机应用, 2019, **39**(10): 2985−2991)

13 Omran M G H, Mahdavi M. Global-best harmony search. *Applied Mathematics and Computation*, 2008, **198**(2): 643−656

14 Draa A, Bouzoubia S, Boukhalfa I. A sinusoidal differential evolution algorithm for numerical optimization. *Applied Soft Computing*, 2015, **27**: 99−126

15 Awad N H, Ali M Z, Liang J J, Qu B Y, Suganthan P N. Problem definitions and evaluation criteria for the CEC 2016 special session and competition on single objective bound constrained real-parameter numerical optimization. In: Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Technical Report, Nanyang Technological University, Singapore, 2016.

16 Tu Q, Chen X C, Liu X C. Multi-strategy ensemble grey wolf optimizer and its application to feature selection. *Applied Soft Computing*, 2019, **76**: 16−30

17 Gong W Y, Cai Z H, Ling C X. DE/BBO: A hybrid differential evolution with biogeography-based optimization for global numerical optimization. *Soft Computing*, 2010, **15**(4): 645−665

18 Avdilek I B. A hybrid firefly and particle swarm optimization algorithm for computationally expensive numerical problems. *Applied Soft Computing*, 2018, **66**: 232−249

19 Qin A K, Huang V L, Suganthan P N. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 2009, **13**(2): 398−417

20 Tang D Y. Spherical evolution for solving continuous optimization problems. *Applied Soft Computing*, 2019, **81**: 1−20

21 Tan Y, Zhu Y C. Fireworks algorithm for optimization. *International Conference in Swarm Intelligence*, 2010: 355−364

22 Rao R V, Savsani V J, Vakharia D P. Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 2011, **43**(3): 303−315

23 He Yi-Chao, Wang Xi-Zhao, Liu Kun-Qi, Wang Yan-Qi. Convergent analysis and algorithmic improvement of differential evolution. *Journal of Software*, 2010, **21**(5): 875−885
(贺毅朝, 王熙照, 刘坤起, 王彦祺. 差分演化的收敛性分析与算法改进. 软件学报, 2010, **21**(5): 875−885)

24 Derrac J, García S, Molina D, Herrera F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 2011, **1**(1): 3−18

25 Das P, Das D K, Dey S. A modified bee colony optimization (MBCO) and its hybridization with k-means for an application to data clustering. *Applied Soft Computing*, 2018, **70**: 590−603

26 Jain A K. Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 2010, **31**(8): 651−666

27 Luo Ke, Li Lian, Zhou Bo-Xiang. Artificial bee colony rough clustering algorithm based on mutative precision search. *Control and Decision*, 2014, **29**(5): 838−842
(罗可, 李莲, 周博翔. 基于变异精密搜索的蜂群聚类算法. 控制与决策, 2014, **29**(5): 838−842)

张新明 河南师范大学教授. 主要研究方向为智能优化算法, 图像去噪, 图像增强和图像分割. 本文通信作者.
E-mail: xinmingzhang@126.com
(**ZHANG Xin-Ming** Professor at Henan Normal University. His research interest covers intelligence optimization algorithm, image denoising, image enhancement, and image segmentation. Corresponding author of this paper.)

姜 云 河南师范大学硕士研究生. 主要研究方向为智能优化算法和图像分割.
E-mail: jiangyun951120@163.com
(**JIANG Yun** Master student at Henan Normal University. Her research interest covers intelligence optimization algorithm and image segmentation.)

刘尚旺 河南师范大学副教授. 主要研究方向为图像处理和计算机视觉.
E-mail: shwl08@126.com
(**LIU Shang-Wang** Associate professor at Henan Normal University. His research interest covers image processing and computer vision.)

刘国奇 河南师范大学副教授. 主要研究方向为图像分割和偏微分方程.
E-mail: liuguoqi080408@163.com
(**LIU Guo-Qi** Associate professor at Henan Normal University. His research interest covers image segmentation and partial differential equation.)

窦 智 河南师范大学讲师. 主要研究方向为算法及数字图像处理.
E-mail: 619534345@163.com
(**DOU Zhi** Lecturer at Henan Normal University. His research interest covers algorithms and digital image processing.)

刘 艳 河南师范大学实验师. 主要研究方向为优化算法和图像分割.
E-mail: liu_yan122@sina.com
(**LIU Yan** Laboratory teacher at Henan Normal University. Her research interest covers optimization algorithm and image segmentation.)