Computer Science and Application 计算机科学与应用, 2024, 14(11), 91-106
Published Online November 2024 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2024.1411219

Hans汉斯

多策略融合改进的蜣螂优化算法及工程设计应用

海明威[1,2]，王　淼[1,2*]

[1]黑龙江省水利科学研究院，黑龙江　哈尔滨
[2]哈尔滨理工大学建筑工程学院，黑龙江　哈尔滨

收稿日期：2024年10月15日；录用日期：2024年11月13日；发布日期：2024年11月21日

摘　要

文章聚焦于蜣螂优化算法所固有的局限性问题，具体表现为其易于陷入局部最优解、在全局搜索能力上有所欠缺，以及收敛速度相对缓慢。针对这些不足，文章提出了一种创新性的改进策略——多策略融合的改进型蜣螂优化算法(简称MSIDBO)。在该改进方案中，首先于算法的初始化阶段引入了Logistic混沌映射机制，旨在有效提升种群分布的均匀程度；其次，采用鱼鹰优化算法替换原有蜣螂算法中的滚球位置更新机制，以解决原算法仅依赖最差值进行位置更新、缺乏个体间即时交流及参数冗余的问题；最后，实施了自适应t分布扰动策略，旨在迭代初期强化全局探索能力，而在迭代末期则加强局部搜索效率，并加速了算法的收敛进程。为了验证MSIDBO算法的有效性，对14个经典测试函数和工程应用问题进行测试，结果表明，引入的3种策略能有效提升蜣螂优化算法的性能。

关键词

蜣螂优化算法，混沌映射，融合鱼鹰优化算法，自适应t分布扰动策略

Improved Dung Beetle Optimization Algorithm with Multi-Strategy Fusion and Applications in Engineering Design

Mingwei Hai[1,2], Miao Wang[1,2*]

[1]Heilongjiang Province Hydraulic Research Institute, Harbin Heilongjiang
[2]College of Civil Engineering and Architecture, Harbin University of Science and Technology, Harbin Heilongjiang

Received: Oct. 15th, 2024; accepted: Nov. 13th, 2024; published: Nov. 21st, 2024

*通讯作者。

文章引用: 海明威, 王淼. 多策略融合改进的蜣螂优化算法及工程设计应用[J]. 计算机科学与应用, 2024, 14(11): 91-106. DOI: 10.12677/csa.2024.1411219

Abstract

This study examines the intrinsic limitations of the dung beetle optimization algorithm, particularly its propensity to converge on local optima, its insufficient global search capabilities, and its relatively slow convergence rate. To mitigate these issues, the paper introduces a novel enhancement strategy termed the Improved Dung Beetle Optimization Algorithm with Multi-Strategy Fusion (MSIDBO). This enhancement involves several key modifications: first, a Logistic Chaos mapping mechanism is incorporated during the initialization phase of the algorithm to enhance the uniformity of population distribution. Second, the Fishhawk optimization algorithm is employed to replace the original rolling ball position update mechanism of the dung beetle algorithm. This substitution addresses the original algorithm's reliance on the worst value for position updates, the absence of instantaneous communication among individuals, and the presence of parameter redundancy. Lastly, an adaptive t-distribution perturbation strategy is introduced to bolster global exploration during the initial iterations while simultaneously improving local search efficiency in the later stages, thereby accelerating the overall convergence of the algorithm. To evaluate the efficacy of the MSIDBO algorithm, a series of tests involving 14 classical benchmark functions and engineering application problems were conducted. The results indicate that the three strategies implemented significantly enhance the performance of the dung beetle optimization algorithm.

Keywords

Dung Beetle Optimization Algorithm, Chaotic Mapping, Fusion Fishhawk Optimization Algorithm, Adaptive t-Distribution Perturbation Strategy

1. 引言

受自然界物理法则与生物行为特性的启迪，研究人员创造了一系列群智能优化方法，涵盖了粒子群优化算法(PSO) [1]、灰狼优化算法(GWO) [2]、蛇优化算法(SO) [3]、鲸鱼优化算法(WOA) [4]、鹈鹕优化算法(POA) [5]等。这些算法具有收敛速度快、搜索精度高、操作简便等共同特点，已被广泛应用于各个领域。

Xue 与 Shen 两位学者在 2022 年首次提出了一种新颖的元启发式优化方法——蜣螂优化算法(DBO) [6]，此算法依据蜣螂个体的角色差异，将种群细分为四种类型。为了综合衡量蜣螂优化算法(DBO)的性能表现，Xue 和 Shen [6]挑选了一个广泛的测试函数集合，该集合包括了 23 个经典的基准测试函数以及 29 个来自 CEC-2017 的复杂测试函数。经过一系列严格的实验验证，结果显示，在收敛速度、解的精确度以及算法运行的稳定性等多个关键性能指标上，DBO 算法与当前广泛采用的主流优化算法相比，展现出了显著的竞争优势。尽管如此，在应用 DBO 算法解决复杂优化问题时，如何确保获得理想结果仍是一大难题。具体而言，DBO 算法面临的主要局限包括：首先，现有的 DBO 在搜索过程中种群的初始位置是随机分布的，由此产生的初始种群位置并不能使其均匀分布于搜索空间；其次，现有的 DBO 在种群位置更新过程中不够灵活，无法动态调整算法在迭代前期与迭代后期的搜索范围，容易使得算法早熟；最后，在处理某些复杂的优化问题时，现有的 DBO 表现不够出色，例如在处理高维度、多约束或动态变化等问题时，收敛速度较慢。随着实际应用场景的不断变化和拓展，对优化算法的收敛精度、稳定性、适

用性等方面提出了更高的要求，需要通过改进来提升其性能，以适应更广泛的应用需求。

为了优化蜣螂优化算法的表现，众多研究者展开了深入探索。Zhang 等人[7]提出了一种基于维度学习的觅食搜索策略，通过向邻近蜣螂学习并选取代理更新位置，增强了算法对局部解的探寻能力，并成功应用于提升反向传播(BP)神经网络的预测精度。Shen 等人[8]则引入了个体与维度交叉机制来更新偷窃蜣螂的位置，有效提高了算法的收敛速度。Tu 等人[9]采用自适应权重因子结合 Levy 扰动，解决了蜣螂优化算法易早熟及搜索性能不佳的问题，同时结合帕累托支配理论筛选出优质解，赋予了算法解决多目标问题的能力。郭琴等[10]受麻雀搜索算法启发，采纳了其追随者位置更新策略以优化蜣螂种群质量，并进一步引入了柯西高斯变异策略，这一举措显著增强了算法摆脱局部最优困境的能力，进而促进了其全局寻优性能的提升。另一方面，潘劲成等人[11]在蜣螂种群的初始化阶段巧妙地融合了 Bernoulli 混沌映射技术，此举有效拓宽了算法的搜索空间。在迭代过程中，他们创新性地应用了自适应高斯－柯西混合变异扰动方法，此举不仅丰富了种群的多样性，还显著提高了算法的收敛精度和稳定性。隋东等人[12]则融合了偏移估计策略与鲸鱼算法中的变螺旋搜索策略，显著增强了算法的全局搜索与局部开发能力。

上述各项研究均采用了多样化的策略对蜣螂优化算法(DBO)进行了改良，从而在各自关注的方面不同程度地提升了其寻优性能。尽管如此，DBO 算法仍然蕴含着巨大的优化潜力和改进空间。鉴于 DBO 算法目前存在的缺陷与局限性，本文提出了一种创新性的解决方案——基于多策略融合的改进蜣螂优化算法(简称 MSIDBO)。MSIDBO 算法的核心宗旨在于，通过巧妙地整合多种优化策略，来显著增强原始 DBO 算法的全局搜索能力，并同步提升其收敛精度与收敛速度。为了全面评估 MSIDBO 算法的性能，我们开展了一系列深入而全面的实验验证。总体而言，本文的主要贡献可以归纳如下：

• 在种群初始化阶段，我们引入了 Logistic 混沌映射，以实现种群的更均匀分布，进而加速算法的收敛进程。

• 我们将鱼鹰优化算法融入其中，替换了原始蜣螂算法在滚球阶段的位置更新机制，从而解决了原算法仅依赖最差值、缺乏个体间即时通讯以及参数冗余的问题。

• 采用了自适应 t 分布扰动策略，确保算法在迭代初期具备出色的全局探索能力，而在迭代后期则展现出良好的局部挖掘能力，同时进一步提高了算法的收敛速度。

• 通过对 14 个经典测试函数的严格测试，我们验证了所提出的 MSIDBO 算法在性能上相较于其他经典元启发式算法具有显著优势。

• 此外，MSIDBO 成功应用于三个实际工程优化问题，验证了其解决复杂工程问题的卓越能力。

本文的结构如下。第 2 节介绍了原始蜣螂优化算法。第 3 节针对蜣螂优化算法的不足，提出了多策略融合改进蜣螂优化算法(MSIDBO)。第 4 节将多策略融合改进蜣螂优化算法与其他算法进行多方面的实验比较，以验证改进措施的有效性。第 5 节在实际工程应用中使用改进算法，进一步探讨改进算法的实际应用性。第 6 节总结了全部工作。

2. 原始蜣螂优化算法

2.1. 滚球蜣螂

蜣螂优化算法深受蜣螂一系列独特且高效的自然习性启发，这些习性包括其滚球行为、舞蹈交流方式、觅食技巧、策略性偷窃行动以及繁殖策略等，旨在通过模拟这些复杂的生存策略来有效解决各类优化问题。基于这些行为特征，设计了五种不同的位置更新规则，以期能够精准地模拟蜣螂的各种行为模式。同时，蜣螂种群细分为四种不同类型的代理个体，即滚球型蜣螂、繁殖型蜣螂、觅食型蜣螂以及偷窃型蜣螂，这些代理个体分别对应着蜣螂种群中的不同行为类型。

在自然环境中，蜣螂以其将粪便滚成球体并利用天体信号作为导航线索，确保球体沿直线行进的特

点而著称。然而，一旦失去光源指引，蜣螂的行进路径便不再保持直线。在蜣螂优化算法(DBO)的框架内，负责模拟蜣螂滚球行为的位置更新公式被设定为：

$$x_i(t+1)=x_i(t)+\alpha\times k\times x_i(t-1)+b\times\Delta x \tag{1}$$

$$\Delta x=\left|x_i(t)-X^w\right| \tag{2}$$

式中，t 表示当前迭代次数；x_i 表示第 i 个蜣螂在搜索空间中的位置信息；k 表示一个偏转系数，用来调整蜣螂移动方向的作用；α 是一个介于 0 和 1 之间的随机数；a 是一个自然系数，其取值限定为−1 或 1；X^w 则代表全局最差位置。

当遇到障碍物阻碍前进时，蜣螂会展现出一种独特的跳舞行为，以此作为调整行进方向的手段，进而探索可能的替代路径。这种跳舞行为在算法中通过切线函数进行模拟，并且我们特别关注该函数在[0, π]区间内的特性。一旦蜣螂通过跳舞行为成功确定了新的行进方向，它就会坚定不移地继续推动粪球沿该方向前进。基于这一行为特征，相应地更新滚动过程中蜣螂的位置信息，并将这一过程具体定义为：

$$x_i(t+1)=x_i(t)+\tan(\theta)\left|x_i(t)-x_i(t-1)\right| \tag{3}$$

式中，tan (θ)为偏转系角。

2.2. 繁殖蜣螂

蜣螂会小心翼翼地将粪球搬运到安全隐蔽的地点藏匿，随后开始进行产卵。对蜣螂种群而言，选定适宜的产卵区域对其繁殖至关重要。为了更精确的模拟蜣螂选择产卵地的行为，引入了边界选择策略，该策略的具体数学表达式为：

$$Lb^*=\max\left(X^*\times(1-R),Lb\right) \tag{4}$$

$$Ub^*=\min\left(X^*\times(1+R),Ub\right) \tag{5}$$

式中，X^*为当前最佳位置；Lb 表示下界，Ub 表示上界。

当蜣螂种群确定了最为理想的产卵地点后，雌性蜣螂便会在该地点特定的孵化球上进行产卵，且每次迭代过程中仅产下一枚卵。由于蜣螂的最佳产卵区会随着种群迭代而动态变化，因此蜣螂产卵区的位置也会随着种群迭代而变化。这种动态变化可以定义为

$$B_i(t+1)=X^*+b_1\times\left(B_i(t)-Lb^*\right)+b_2\times\left(B_i(t)-Ub^*\right) \tag{6}$$

式中，B_i 表示卵球在空间中的位置；b_1 和 b_2 和为 1 × D 的随机向量，它们可以使算法在搜索空间中更加全面地探索可能的解。

2.3. 觅食蜣螂

小蜣螂在从产卵球中孵化成虫后，会出来寻找食物来源。为了确保小蜣螂能在最理想的觅食地点获取食物，需要构建一个最优觅食区域模型，用以指导小蜣螂的觅食行为并模拟其觅食过程。该最优觅食区域的具体定义公式为：

$$Lb^b=\max\left(X^b\times(1-R),Lb\right) \tag{7}$$

$$Ub^b=\min\left(X^b\times(1+R),Ub\right) \tag{8}$$

式中，X^b 为全局最佳位置。由此，小蜣螂的位置更新为

$$x_i(t+1)=x_i(t)+C_1\times\left(x_i(t)-Lb^b\right)+C_2\times\left(x_i(t)-Ub^b\right) \tag{9}$$

式中，C_1是服从正态分布的随机数；$C_2 \in (0, 1)$是一个随机向量。

2.4. 偷窃蜣螂

在蜣螂种群内部，存在着一类特殊的个体—偷窃蜣螂，它们会窃取其他蜣螂的粪球作为自己的食物。假设这些偷窃行为发生在争夺食物资源的最佳区域，那么偷窃蜣螂的位置更新过程可以描述为：

$$x_i(t+1) = X^b + S \times g \times \left(\left| x_i(t) - X^* \right| + \left| x_i(t) - X^b \right| \right) \tag{10}$$

在给出的公式中，引入了一个随机向量g，该向量服从均值为 0 且方差为 1 的正态分布。此外，定义了一个常数S，它为算法的随机性和多样性提供了保障。

3. 多策略融合改进蜣螂优化算法(MSIDBO)

3.1. Logistic 混沌映射

在算法设计的整体框架中，初始种群的质量扮演着至关重要的角色，它对算法的整体性能具有决定性的影响。一个优质的初始种群能够为算法提供更有利的起点，从而有助于算法更快地收敛到全局最优解。传统的随机初始化手段往往造成种群多样性欠缺且分布不均衡，进而可能引发初始解的不均匀分布，显著削弱算法的寻优效能。针对蜣螂优化算法而言，其初始蜣螂种群是随机生成的，且种群的初始化方式会对算法的寻优路径产生一定的影响。混沌现象因其独特的遍历性和内在随机性特质，在提升群智能算法的寻优性能方面展现出了显著的优势。这一特性使得混沌现象成为了一种有效的工具，被广泛地融入到多种群智能算法中，以增强其搜索能力和效率。例如，混沌现象已经成功地被应用于麻雀优化算法[13]以及蚁群优化算法[14]等，取得了良好的优化效果。特别是 Logistic 混沌映射，因其出色的随机性和长周期特性，在群智能优化算法的初始化阶段得到了广泛应用。

Logistic 混沌映射数学表达式如下：

$$x_{n+1} = u x_n (1 - x_n) \tag{11}$$

其中，$u \in [0, 4]$，$x_n \in (0, 1)$，$n = 1, 2, \ldots$。

3.2. 融合鱼鹰优化算法

鱼鹰优化算法(Osprey Optimization Algorithm, OOA)是一种模拟鱼鹰捕食行为的优化算法，由 Mohammad Dehghani 和 Pavel Trojovský 于 2023 年提出[15]。该算法通过模拟鱼鹰的两个主要行为—定位鱼类并捕捉，以及将捕获的鱼带到安全的地方食用—来进行寻优。算法的设计涵盖了两个核心阶段：全局探索与局部开发。

在全局探索阶段，算法模拟了鱼鹰识别并捕捉鱼类的行为。鱼鹰凭借其卓越的视觉能力锁定水下鱼类的位置，随后发起攻击并捕获。这一行为在算法中被抽象为种群更新的首要阶段，通过大幅度调整鱼鹰在搜索空间内的位置，强化了算法在定位最优区域及规避局部最优方面的探索效能。进入局部开发阶段，鱼鹰会将捕获的鱼类带至适宜地点享用。这一行为在算法中被映射为种群更新的第二阶段，通过在搜索空间内进行小幅度的位置调整，增强了算法在局部搜索中的挖掘能力，并促使算法在已发现的解附近逐步收敛至更优解。

鉴于鱼鹰优化算法在全局探索方面的优势，它能够弥补蜣螂算法在滚球阶段仅依赖最差值、缺乏即时通讯机制且参数设置复杂的不足。鉴于此，本文决定采纳鱼鹰优化算法在第一阶段所展现的全局探索策略，以此作为替代，来优化原始蜣螂算法在滚球阶段的位置更新机制。具体而言，鱼鹰优化算法在第一阶段的全局探索策略，其数学表达形式可以详细阐述如下：

$$x_{i,j}^{p1} = x_{i,j} + r_{i,j} \cdot \left(SF_{i,j} - I_{i,j} \cdot x_{i,j} \right) \tag{12}$$

式中，$r_{i,j}$ 是区间[0, 1]中的随机数，$SF_{i,j}$ 是第 i 只鱼鹰从它的鱼群中随机选定的鱼，$I_{i,j}$ 是集合中的任一数。

3.3. 自适应 t 分布扰动策略

标准蜣螂优化算法(DBO)的运行机制建立在蜣螂觅食行为的基础之上，并依赖于四种不同类型的代理蜣螂之间的协同合作，共同寻找最优解。然而，这一过程中存在一个潜在的问题：一旦蜣螂在觅食时误入局部最优解的陷阱，就可能导致整个算法过早地收敛，从而无法找到全局最优解。为了解决这个问题，我们可以借鉴统计学中的 t 分布，也被称为“student’s t 分布”。t 分布是高斯分布(正态分布)与柯西分布的一种广义形式，具有独特的数学特性。特别值得注意的是，当 t 分布的自由度参数设定为 1 时，其形态会趋近于柯西分布；而当自由度参数逐渐增大并趋于无穷大时，t 分布则会逐渐逼近高斯分布。这一特性使得 t 分布在处理具有不同自由度的数据时，能够展现出灵活多变的统计性质[16] [17]。

为了优化蜣螂的觅食行为，本文在蜣螂觅食阶段创新性地引入了基于 t 分布的扰动机制。具体而言，我们设计了一个以迭代次数为自变量的公式，用以动态地确定 t 分布的自由度参数。随后，我们利用这个参数对小蜣螂的觅食行为进行 t 分布变异扰动。这一策略的核心目的，是在算法迭代的初期阶段，赋予蜣螂算法强大的全局探索能力，以便广泛搜索潜在的解空间；而在迭代的后期阶段，则逐步提升算法的局部挖掘能力，确保算法能够精确地收敛到最优解。以下是详细的位置更新策略公式：

$$x(t+1) = x(t) + t(iter) \times x(t) \tag{13}$$

式中，定义了一个特定的 t 分布，其自由度参数由当前的迭代次数 $iter$ 确定，将其表示为 t ($iter$)；x (t)为蜣螂当前位置，x (t+1)为变异后的蜣螂位置。

3.4. MSIDBO 算法

算法 1 MSIDBO 算法框架名称

输入：初始化种群数 N，最大迭代次数 T_{max}
输出：最佳位置 X^b，适应度 f_b

1：根据 Logistic 混沌映射进行种群初始化；
2：**While**($t \le T_{max}$)；
3：**for** i=1→N;
4：　　**for** i=1→N_1;
5：　　按照公式(1)(2)(3)(12)更新滚球蜣螂位置及适应度值；
6：　　**end for；**
7：　　**for** i=1→N_2;
8：　　按照公式(4)(5)(6)更新繁殖蜣螂位置及适应度值；
9：　　**end for；**
10：　　**for** i=1→N_3;
11：　　按照公式(7)(8)(9)(13)更新觅食蜣螂位置及适应度值；
12：　　**end for；**
13：　　**for** i=1→N_4;
14：　　按照公式(10)更新偷窃蜣螂位置及适应度值；
15：　　**end for；**
16：**end for；**
17：完成最佳位置X^b，适应度 f_b 的更新；
18：**end while；**
Return X^b，适应度 f_b

3.5. MSIDBO 算法时间复杂度分析

原始 DBO 算法的时间复杂度为$O(T_{\max} * N * D)$，其中 N 为种群规模，$T_{\max}$为最大迭代次数，D 为维度。当我们采用经过优化的 Logistic 混沌映射来进行种群初始化时，时间复杂度为$O(N * D)$。加入融合鱼鹰优化算法和自适应 t 分布扰动策略之后，没有引入新的循环，也没有改变顺序，因此 MSIDBO 算法的时间复杂度为$O(T_{\max} * N * D)$，与原始 DBO 算法的时间复杂度一致。

4. 模拟实验和结果分析

4.1. 实验环境

为了确保公平性和一致性，我们将所有待比较的算法都部署在相同的硬件平台和软件环境中进行运行。这一做法旨在消除因硬件差异或软件环境不同而对算法性能评估产生的潜在影响，从而能够更准确地评估各算法的实际表现。实验环境为 Windows 11 64bit 操作系统，内存为 32.00 GB，CPU 为 12th Gen Intel (R) Core (TM) i7-12650H 2.30 GHz，仿真软件为 Matlab R2024a。

4.2. 实验设计

Table 1. Test functions
表 1. 测试函数

	名称	维度	范围	最优值
F1	Sphere	30	[−100, 100]	0
F2	Schwefe2.22	30	[−10, 10]	0
F3	Schwefel1.2	30	[−100, 100]	0
F4	Schwefel2.21	30	[−100, 100]	0
F5	Rosenbrock	30	[−30, 30]	0
F6	STEP	30	[−100, 100]	0
F7	Quartic	30	[−1.28, 1.28]	0
F8	Schwefel	30	[−500, 500]	−12569.5
F9	Rastrigin	30	[−5.12, 5.12]	0
F10	Ackley	30	[−32, 32]	0
F11	Griewank	30	[−600, 600]	0
F12	Penalized	30	[−50, 50]	0
F13	Penalized2	30	[−50, 50]	0
F14	Foxholes	30	[−65, 65]	0.998

在本节中，我们精心挑选了五种在智能优化领域具有广泛影响力的经典算法，包括蜣螂优化算法(DBO) [6]、灰狼优化算法(GWO) [2]、鲸鱼优化算法(WOA) [4]、金豺优化算法(GJO) [18]以及黑翅鸢优化算法(BKA) [19]，并将它们与本文新提出的 MSIDBO 算法进行了深入的对比分析。为了全面而准确地评估 MSIDBO 算法的改进效果，我们采用了 14 个经典的测试函数[20]，这些测试函数的具体细节已在表 1

中详细列出。具体而言，我们选择了七个单峰函数(F1 至 F7)，这些函数主要用于检验算法在开发(或挖掘)能力方面的表现。此外，我们还选择了六个多峰函数(F8 至 F13)，这些函数含有一个全局最优解和多个局部最优解，因此能够有效地评估算法的全局搜索能力和局部逃逸能力。最后，我们还选择了一个固定维度的多峰函数(F14)，该函数可用于验证算法在探索和开发能力之间的平衡性和稳定性。在实验设置方面，我们统一设定了所有算法的种群规模为 30，迭代次数为 500，以确保实验条件的一致性。为了消除随机误差对实验结果的影响，我们针对每个模拟实验均独立运行了 30 次，并计算了每个算法的最优值(min)、标准差(std)、平均值(avg)、最差值(worse)以及中位数(median)等五个关键性能指标。此外，我们还进行了秩和检验，以进一步验证算法之间的性能差异。

4.3. 实验结果分析

表 2 详细列出了模拟结果的具体数据。为了全面评估各算法的性能，我们计算了每个算法的最优解(即最小值 min)、标准差(std)、平均值(avg)、最差解(worse)以及中位数(median)等指标。在测试函数方面，我们采用了一系列单峰测试函数(F1 至 F7)，这些函数具有一个显著的特点，即它们仅存在一个全局最优解。因此，这些函数非常适合用于评估算法的开发(或称为挖掘)性能。通过在这些函数上进行测试，我们可以直观地观察到算法在搜索过程中的收敛速度和精度，从而判断其开发性能的好坏。根据表格 2 中的数据，可以观察到 MSIDBO 算法在处理测试函数 F1 至 F6 时展现出了相较于其他算法的优势。针对 F7 函数，尽管 MSIDBO 算法的适应度值未达最优，但仍处于领先地位。为了评估算法的勘探能力，我们采用了一个多模态函数集，即 F8 至 F13。这些函数具有一个显著的特征，即它们包含了大量的局部最小值。更为复杂的是，随着维度的不断增加，这些局部最小值的数量会呈现出指数级的增长趋势。因此，这些函数对于检验算法在复杂搜索空间中的勘探效能具有极高的挑战性。通过在这些函数上进行测试，我们可以深入地了解算法在面临大量局部最优解时的表现，从而准确地评估其勘探能力。对于函数 F9 至 F11 以及 F13，MSIDBO 算法展现出了超越其他算法的搜索强度。综上所述 MSIDBO 在绝大多数的测试函数上取得的最优收敛值和稳定性均优于其他 5 种算法。

Table 2. Simulated results
表 2. 模拟结果

		MSIDBO	DBO	GWO	WOA	GJO	BKA
F1	min	0	1.6318E−176	5.59506E−60	1.13152E−86	6.5924E−116	4.8232E−109
	std	0	1.97926E−99	1.50325E−56	8.74101E−77	8.7486E−108	1.05796E−74
	avg	0	3.67546E−100	7.29981E−57	2.12644E−77	2.6139E−108	1.93298E−75
	median	0	1.168E−135	4.88802E−58	6.12358E−84	4.6822E−111	8.94743E−100
	worse	0	1.08456E−98	5.90548E−56	4.67972E−76	4.6172E−107	5.79483E−74
F2	min	2.5449E−289	1.31727E−84	8.2768E−35	6.49339E−60	7.30426E−63	5.47192E−54
	std	0	3.16335E−59	7.33885E−33	1.34027E−51	2.75752E−60	6.49164E−41
	avg	4.4575E−253	5.86792E−60	6.13781E−33	4.43535E−52	1.71222E−60	1.18521E−41
	median	2.2739E−261	3.63397E−68	3.71896E−33	1.2099E−55	5.07748E−61	4.70341E−51
	worse	1.3013E−251	1.7334E−58	2.98211E−32	5.28074E−51	1.02273E−59	3.55562E−40

续表

	min	0	7.6805E−146	6.38145E−30	0.000726735	1.61973E−64	1.0296E−102
	std	0	1.76419E−84	3.16794E−23	164.2813995	3.85625E−56	6.54981E−73
F3	avg	0	3.22096E−85	6.02529E−24	142.0842022	1.4267E−56	1.19583E−73
	median	0	1.8268E−123	6.17437E−27	66.04950499	1.93306E−60	2.57059E−99
	worse	0	9.66289E−84	1.73722E−22	562.1154687	1.88639E−55	3.58748E−72
	min	8.7698E−294	4.23344E−81	1.69362E−20	4.87262E−06	1.08366E−43	2.21516E−53
	std	0	7.16731E−53	1.18141E−17	3.342377498	8.13317E−40	1.2707E−39
F4	avg	3.7586E−241	1.65231E−53	3.52498E−18	2.096663554	2.44525E−40	2.74078E−40
	median	2.3481E−266	1.54137E−66	6.27638E−19	0.92339951	1.6681E−41	3.30196E−50
	worse	1.1276E−239	3.88666E−52	6.53664E−17	12.62441754	4.3354E−39	6.8749E−39
	min	0.000408923	4.916158339	5.709289844	4.540800829	6.229844156	3.950182102
	std	0.837663733	3.087681112	0.648885269	0.834642314	0.530238318	1.412813056
F5	avg	4.284751776	5.899588558	6.710064462	6.851630693	7.127827938	6.320492649
	median	4.404044709	5.393438255	6.294549865	6.828114685	7.195977818	6.13773147
	worse	4.803969775	22.19170461	8.518302887	8.777013782	8.100719072	8.948063013
	min	7.70372E−32	3.95971E−30	1.76795E−06	0.000150467	5.01126E−06	8.71974E−07
	std	1.23245E−27	1.10124E−22	1.46104E−06	0.001045251	0.160432971	0.267422461
F6	avg	4.3038E−28	4.20768E−23	3.97904E−06	0.001079024	0.17015947	0.067628029
	median	5.64683E−30	7.7788E−26	3.72648E−06	0.00077782	0.249132898	5.50225E−06
	worse	6.28484E−27	3.98864E−22	9.03118E−06	0.004494862	0.502265784	1.291247008
	min	8.54976E−06	0.000207731	7.53757E−05	0.000127901	6.95642E−06	1.17813E−05
	std	0.000292525	0.001014405	0.000484447	0.002465513	0.000116437	0.000601523
F7	avg	0.000376119	0.001281034	0.000727868	0.002085999	0.000146259	0.000286483
	median	0.000335527	0.000876648	0.000627668	0.001024855	0.000110455	0.000169891
	worse	0.001322001	0.003937947	0.00196891	0.011826643	0.000554176	0.003407412
	min	−4189.82852	−4188.265059	−3439.505063	−4188.756929	−3203.98894	−4189.032923
	std	511.2761011	455.3116838	309.1493868	595.3793993	429.9285956	348.1944774
F8	avg	−3560.15093	−3500.748535	−2696.685935	−3265.5958	−2265.419765	−3358.56852
	median	−3497.402355	−3512.474126	−2674.044153	−3044.370353	−2221.624858	−3288.457762
	worse	−2349.406827	−2284.196102	−2151.988756	−2050.779474	−1507.85367	−2763.528069
	min	0	0	0	0	0	0
	std	0	5.687882326	1.12135456	2.59454E−15	0	0
F9	avg	0	2.10493997	0.236494561	4.73695E−16	0	0
	median	0	0	0	0	0	0
	worse	0	24.87393104	6.094493297	1.42109E−14	0	0

续表

F10	min	4.44089E−16	4.44089E−16	3.9968E−15	4.44089E−16	4.44089E−16	4.44089E−16
	std	0	0	1.13631E−15	2.36029E−15	1.08403E−15	0
	avg	4.44089E−16	4.44089E−16	7.43109E−15	3.28626E−15	3.64153E−15	4.44089E−16
	median	4.44089E−16	4.44089E−16	7.54952E−15	3.9968E−15	3.9968E−15	4.44089E−16
	worse	4.44089E−16	4.44089E−16	1.11022E−14	7.54952E−15	3.9968E−15	4.44089E−16
F11	min	0	0	0	0	0	0
	std	0	0.087236012	0.029903915	0.12791314	0.001426015	0
	avg	0	0.037319973	0.030165694	0.061807322	0.000260354	0
	median	0	0	0.022140478	0	0	0
	worse	0	0.364345978	0.116506577	0.542917728	0.007810607	0
F12	min	4.80844E−32	3.55937E−29	6.6225E−07	0.000498274	1.03586E−05	1.62841E−07
	std	0.056781861	6.16134E−23	0.006855396	0.009393626	0.0516	0.055133049
	avg	0.010366902	1.42638E−23	0.002644962	0.00760913	0.047097533	0.013743869
	median	2.07808E−29	2.11462E−26	1.09626E−06	0.002598686	0.03916324	8.02171E−07
	worse	0.311007061	3.33401E−22	0.019946799	0.035764577	0.275393043	0.276909594
F13	min	6.89646E−32	2.24809E−28	2.04503E−06	0.000775755	1.92691E−05	4.72247E−06
	std	0.020048194	0.050857176	0.043047575	0.059944183	0.095522566	0.166331417
	avg	0.006902893	0.02941213	0.013169153	0.043373095	0.11892333	0.126156414
	median	3.72398E−28	5.84935E−24	8.5615E−06	0.014569494	0.105967009	0.050811869
	worse	0.09737116	0.197739754	0.199787347	0.202041407	0.299237679	0.76218286
F14	min	0.998003838	0.998003838	0.998003838	0.998003838	0.998003838	0.998003838
	std	4.282084227	1.0303653	4.136038848	3.338627879	4.619407071	0.181483682
	avg	3.866597835	1.559707067	4.131335685	2.861340837	5.952025116	1.031138073
	median	0.998003838	0.998003838	2.982105157	0.998003839	2.982105157	0.998003838
	worse	10.76318067	3.968250106	12.67050581	10.7631812	12.67050581	1.9920309

4.4. 算法收敛曲线的对比分析

为了更直观地凸显 MSIDBO 算法在动态收敛特性和避免陷入局部最优方面的卓越表现，本文绘制了六种算法在 14 个基准测试函数上的收敛曲线图，具体如图 1 所示。在该图中，横轴代表迭代次数的递进，而纵轴则对应着适应度值的变化。

在单峰测试函数上，MSIDBO 算法的收敛速度和寻优精度相较于其他对比算法展现出了显著的优势。特别是在这些单峰测试函数中，全局最小值往往隐藏在狭窄的抛物线谷底，想精确收敛至最小值非常困难。然而，MSIDBO 算法所获得的最优值比其他算法高出了 2 至 5 个数量级，展示了强大的寻优能力。通过观察某些特定测试函数的收敛曲线，我们发现尽管 MSIDBO 与其他算法在收敛精度上相当，但其收

敛速度却明显更快。在多峰测试函数上，MSIDBO 算法能够更准确地逼近全局最优解，收敛过程中的波动性和误差范围也更小。在面对具有多个局部最优解和复杂搜索空间的优化问题时，MSIDBO 算法也能够更有效地跳出局部最优，从而找到全局最优解。

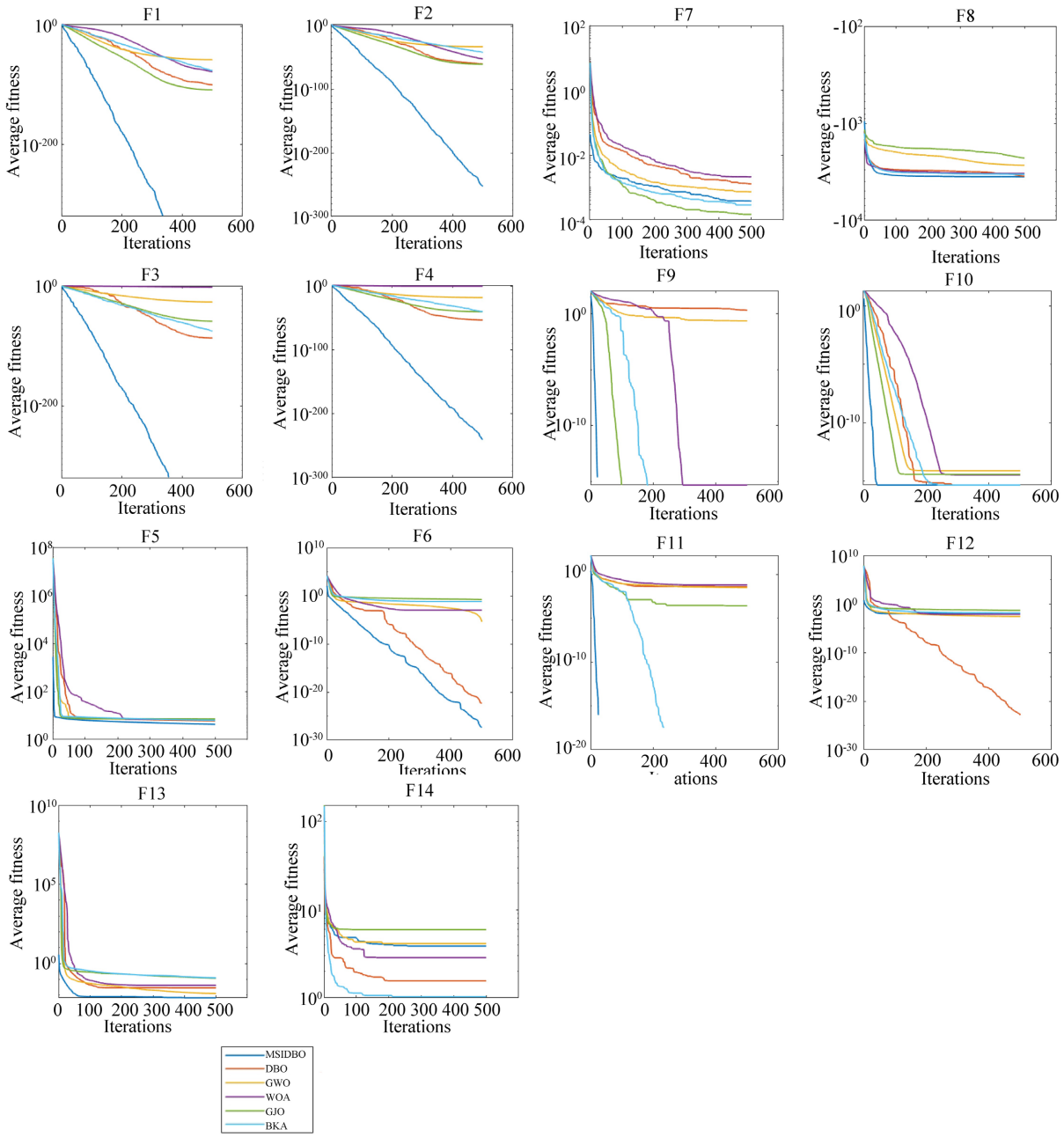

Figure 1. Algorithm convergence graphs
图 1. 算法收敛图

4.5. 统计分析：秩和检验

为了验证 MSIDBO 与其他优化算法是否存在性能差异，采纳一种名为 Wilcoxon 秩和检验的统计方法行对比分析[21]。该方法的核心在于通过计算 p 值来评估统计结果的显著性水平。具体而言，当 p 值小

于 0.05 时，可以合理推断两种算法之间存在统计学上的显著差异。相关的计算结果已详细列于表 3 中，可以清晰地观察到 MSIDBO 算法所得的优化结果与优化算法相比呈现出较大的差异性。

Table 3. Wilcoxon rank sum test
表 3. Wilcoxon 秩和检验

	DBO	GWO	WOA	GJO	BKA
F1	1.21178E−12	1.21178E−12	1.21178E−12	1.21178E−12	1.21178E−12
F2	3.01986E−11	3.01986E−11	3.01986E−11	3.01986E−11	3.01986E−11
F3	1.21178E−12	1.21178E−12	1.21178E−12	1.21178E−12	1.21178E−12
F4	3.01986E−11	3.01986E−11	3.01986E−11	3.01986E−11	3.01986E−11
F5	3.01986E−11	3.01986E−11	7.38908E−11	3.01986E−11	1.07018E−09
F6	1.28704E−09	3.01986E−11	3.01986E−11	3.01986E−11	3.01986E−11
F7	2.31679E−06	0.003848068	1.60621E−06	0.000110577	0.005569939
F8	0.529771056	3.35195E−08	0.036438856	4.19968E−10	0.037782018
F9	0.011035086	0.160802121	0.333710696	1	1
F10	1	6.13374E−14	8.725E−08	3.94103E−12	1
F11	0.005584312	6.24703E−10	0.001370186	0.333710696	1
F12	1.01877E−05	5.57265E−10	5.57265E−10	5.57265E−10	5.57265E−10
F13	0.000721942	0.000117344	5.17627E−07	1.19954E−08	2.43244E−09
F14	0.18516828	0.001546738	0.016633297	2.99206E−05	0.003174175

5. 工程应用分析

为了检验 MSIDBO 算法在实际工程应用中的潜力，本文使用所提出的 MSIDBO 算法对三个工程设计问题进行了优化。分别是三杆桁架设计问题[22]、焊接梁设计问题[23]、减速器问题[24]，并将优化结果进行了比较。测试方法是将这三个工程问题转化为数学模型，使用惩罚函数处理不等式约束，然后使用各算法求出最优解。在寻优过程中，每个算法独立运行 30 次，最大迭代次数为 300，取最优结果。

5.1. 三杆桁架设计问题

三杆桁架设计问题是土木工程领域中常见的非线性分式优化问题[22]。为了使三杆桁架的体积最小，需要对每个桁架构件施加约束，它包括三种类型的约束，如屈曲约束、挠度约束和应力约束。这个问题共有两个核心参数：A1 和 A2。其目标函数和约束条件表述如下：

(1) 设计变量 $\vec{x}=\left[x_1\ x_2\right]=\left[A_1\ A_2\right]$

(2) 目标函数 $Min\,\mathrm{f}\left(\vec{x}\right)=\left(2\sqrt{2}x_1+x_2\right)\cdot l$

(3) 约束条件
$$\begin{cases} g_1\left(\vec{x}\right)=\dfrac{\sqrt{2}x_1+x_2}{\sqrt{2}x_1^2+2x_1x_2}P-\sigma\le 0 \\ g_2\left(\vec{x}\right)=\dfrac{x_2}{\sqrt{2}x_1^2+2x_1x_2}P-\sigma\le 0 \\ g_3\left(\vec{x}\right)=\dfrac{1}{\sqrt{2}x_2+x_1}P-\sigma\le 0 \end{cases}$$

(4) 取值范围 $0\le x_1,x_2\le 1$

(5) 常数变量 $l=100cm, P=2\,kN/cm^2, \sigma=2\,kN/cm^2$

我们将 MSIDBO 算法应用于三杆桁架问题，并将所得实验结果与多种来自其他文献的改进算法进行了对比。在表 4 中，我们列出了不同算法所获得的最优解以及相应的决策变量值。通过对比分析，我们

发现 MSIDBO 算法相较于其他算法表现出了更强的竞争力。

Table 4. Comparison of test results of each algorithm for solving the threE−bar truss problem
表 4. 各算法求解三杆桁架问题的测试结果比较

算法	A_1	A_2	最优值
MSIDBO	0.788809685	0.407867857	263.8958567
DBO	0.788106244	0.409859738	263.8960817
GWO	0.787673406	0.411097937	263.8974765
WOA	0.782474986	0.426073704	263.924718
GJO	0.788142132	0.409795675	263.8998259
BKA	0.788671957	0.40825728	263.8958434

从表 4 中各算法的优化结果可以明显的看出，MSIDBO 算法能够找到更好的控制参数值和目标函数值。总的来说，当参数 A_1 和 A_2 的值分别为 0.788809685 与 0.407867857 时，三杆桁架的体积达到最小值 263.8958567。

5.2. 焊接梁设计问题

焊接梁问题的主要目标是优化焊接梁的重量[23]。焊接梁的设计问题是找到满足剪应力(τ)、弯曲应力(σ)、梁的弯曲载荷(Pc)和终端偏差(δ)等约束条件的四个设计参数。这些设计参数分别为梁的长度(l)、宽度(t)、厚度(b)和焊接点的厚度(h)。最后，这些设计参数用于计算焊接梁的制造成本。焊接梁设计问题的公式如下：

(1) 设计变量 $\vec{x}=\left[x_1\ x_2\ x_3\ x_4\right]=\left[h\ l\ t\ b\right]$

(2) 目标函数 $\mathrm{f}\left(\vec{x}\right)=1.10471x_1^2x_2+0.04811x_3x_4\left(14.0+x_2\right)$

(3) 约束条件

$$
\begin{cases}
g_1\left(\vec{x}\right)=\tau\left(\vec{x}\right)-\tau_{\max}\le 0\\
g_2\left(\vec{x}\right)=\sigma\left(\vec{x}\right)-\sigma_{\max}\le 0\\
g_3\left(\vec{x}\right)=\delta\left(\vec{x}\right)-\delta_{\max}\le 0\\
g_4\left(\vec{x}\right)=x_1-x_4\le 0\\
g_5\left(\vec{x}\right)=p-p_c\left(\vec{x}\right)\le 0\\
g_6\left(\vec{x}\right)=0.125-x_1\le 0\\
g_7\left(\vec{x}\right)=0.10471x_1^2+0.04811x_3x_4\left(14.0+x_2\right)-5.0\le 0\\
\tau\left(\vec{x}\right)=\sqrt{\left(\tau'\right)^2+2\tau'\tau''\dfrac{x_2}{2R}+\left(\tau''\right)^2},\tau'=\dfrac{p}{\sqrt{2}x_1x_2},\tau''=\dfrac{MR}{J},M=p\left(L+\dfrac{x_2}{2}\right)\\
R=\sqrt{\dfrac{x_2^2}{4}+\left(\dfrac{x_1+x_3}{2}\right)^2},J=2\left\{\sqrt{2}x_1x_2\left[\dfrac{x_2^2}{12}+\left(\dfrac{x_1+x_3}{2}\right)^2\right]\right\},\sigma\left(\vec{x}\right)=\dfrac{6pL}{x_4x_3^2}\\
\delta\left(\vec{x}\right)=\dfrac{4pL^3}{Ex_3^3x_4},p_c\left(\vec{x}\right)=\dfrac{4.013E\sqrt{\dfrac{x_3^2x_4^6}{36}}}{L^2}\left(1-\dfrac{x_3}{2L}\sqrt{\dfrac{E}{4G}}\right)\\
p=6000lb,L=14in,\delta_{\max}=0.25in\\
E=30\times10^6\ psi,G=12\times10^6\ psi\\
\tau_{\max}=13600psi,\sigma_{\max}=30000psi
\end{cases}
$$

(4) 取值范围 $0.1 \le x_1 \le 2, 0.1 \le x_2 \le 10, 0.1 \le x_3 \le 10, 0.1 \le x_4 \le 2$

利用 MSIDBO 算法对焊接梁问题进行优化并得出最优值，并将求解结果与 5 种算法数据进行对比。表 5 显示了各算法得出的相关参数的值。

Table 5. Comparison of test results of algorithms for solving the welded beam problem
表 5. 各算法求解焊接梁问题的测试结果比较

算法	h	l	t	b	最优值
MSIDBO	0.198733201	3.339233366	9.192056879	0.198832215	1.670322918
DBO	0.199100231	3.33399329	9.18583492	0.199101111	1.671199494
GWO	0.197263339	3.366985556	9.197773334	0.198971147	1.673826354
WOA	0.17425565	3.908364381	9.187388467	0.199033017	1.706568345
GJO	0.197566703	3.356296869	9.229642542	0.198801468	1.676857042
BKA	0.198171654	3.350458274	9.19199264	0.198856863	1.671153175

5.3. 减速器问题

减速器问题的优化目的是优化齿轮的重量和轴的轴向变形[24]，使得减速器自身的重量达到最小化。减速器共包含 7 个约束变量：宽度(b)，齿模数量(m)，小齿轮中的齿数(z)，轴承之间的第一根轴的长度($l1$)、轴承之间的第二根轴的长度($l2$)、第一根轴的直径($d1$)和第二根轴的直径($d2$)。通过求解减速器中的 11 个约束条件，得出相关设计变量的值和减速器的重量。减速器的数学模型如下所示：

(1) 设计变量 $\vec{x} = [x_1\ x_2\ x_3\ x_4\ x_5\ x_6\ x_7] = [b\ m\ z\ l_1\ l_2\ d_1\ d_2]$

(2) 目标函数

$$\begin{aligned} f(\vec{x}) = {} & 0.7894 x_2^2 x_1 \left(14.9334 x_3 - 43.0934 + 3.3333 x_3^2\right) + \\ & 0.7854\left(x_7^2 x_5 + x_6^2 x_4\right) - 1.508 x_1 \left(x_7^2 + x_6^2\right) + 7.477\left(x_7^3 + x_6^3\right) \end{aligned}$$

(3)约束条件

$$\begin{cases} g_1(\vec{x}) = -x_1 x_2^2 x_3 + 27 \le 0 \\ g_2(\vec{x}) = -x_1 x_2^2 x_3^2 + 397.5 \le 0 \\ g_3(\vec{x}) = -x_2 x_6^4 x_3 x_4^{-3} + 1.93 \le 0 \\ g_4(\vec{x}) = -x_2 x_7^4 x_3 x_5^{-3} + 1.93 \le 0 \\ g_5(\vec{x}) = 10 x_6^{-3} \sqrt{16.91 \times 10^6 + \left(745 x_4 x_2^{-1} x_3^{-1}\right)^2} - 1100 \le 0 \\ g_6(\vec{x}) = 10 x_7^{-3} \sqrt{157.5 \times 10^6 + \left(745 x_5 x_2^{-1} x_3^{-1}\right)^2} - 850 \le 0 \\ g_7(\vec{x}) = x_2 x_3 - 40 \le 0 \\ g_8(\vec{x}) = -x_1 x_2^{-1} + 5 \le 0 \\ g_9(\vec{x}) = x_1 x_2^{-1} - 12 \le 0 \\ g_{10}(\vec{x}) = 1.5 x_6 - x_4 + 1.9 \le 0 \\ g_{11}(\vec{x}) = 1.1 x_7 - x_5 + 1.9 \le 0 \end{cases}$$

(4) 取值范围

$$2.6 \le x_1 \le 3.6, 0.7 \le x_2 \le 0.8, 17 \le x_3 \le 28, 7.3 \le x_4, x_5 \le 8.3, 2.9 \le x_6 \le 3.9, 5 \le x_7 \le 5.5$$

Table 6. Comparison of test results of the algorithms for solving the speed reducer problem
表 6. 各算法求解减速器问题的测试结果比较

算法	b	m	z	l_1	l_2	d1	d2	最优值
MSIDBO	3.5	0.7	17	7.3	7.715319912	3.350540949	5.286654465	2994.424466
DBO	3.5	0.7	17	7.3	7.715319912	3.350540949	5.286654465	2994.424466
GWO	3.506731037	0.7	17	7.358334334	7.87969366	3.355749193	5.28753044	3003.079007
WOA	3.502310029	0.7	17	7.3	8.088418175	3.478078982	5.286783667	3037.351444
GJO	3.50708475	0.7	17	7.3	7.940089649	3.358400103	5.295416248	3009.738061
BKA	3.500071368	0.7	17	7.3	7.927283161	3.350569303	5.289362582	3000.838299

从表 6 中的数据可以明显地看出，MSIDBO 算法处理问题时得到的减速器重量最小，有效节省了工程设计费用。与其他算法相比，MSIDBO 算法在处理减速器问题时所需要的 7 个核心参数值都得到了提高，从而使得减速器的重量得到优化。

6. 主要结论

鉴于蜣螂优化算法(DBO)在种群多样性保持、局部最优解逃逸能力、收敛精度提升以及全局搜索范围扩展等方面存在的局限性，本文提出了一种集多策略改进于一体的新型蜣螂优化算法，即多策略融合改进的蜣螂优化算法(MSIDBO)。首先于算法的初始化阶段引入了 Logistic 混沌映射机制，旨在有效提升种群分布的均匀程度；其次，采用鱼鹰优化算法替换原有蜣螂算法中的滚球位置更新机制，以解决原算法仅依赖最差值进行位置更新、缺乏个体间即时交流及参数冗余的问题；最后，实施了自适应 t 分布扰动策略，旨在迭代初期强化全局探索能力，而在迭代末期则加强局部搜索效率，并加速了算法的收敛进程。为了验证上述改进策略的有效性，我们进行了模拟实验，选取了 14 个经典测试函数作为实验对象，并将 MSIDBO 算法与五种知名的传统优化算法进行了对比。实验结果显示，MSIDBO 算法在多个方面均取得了显著改进。此外，我们还基于这 14 个经典测试函数的实验数据，利用 Wilcoxon 秩和检验方法验证了 MSIDBO 算法与传统 DBO 算法以及其他传统算法之间的显著差异。实验结果表明，MSIDBO 算法与其他算法相比存在显著差异。为了进一步检验 MSIDBO 算法的实际应用潜力，我们选取了三个工程设计问题作为测试案例，并利用 MSIDBO 算法进行了优化测试。同时，我们还将 MSIDBO 算法与其他知名算法进行了对比。对比结果表明，MSIDBO 算法在处理此类工程设计问题时展现出了更强的竞争力。

基金项目

黑龙江省省属科研院所科研业务费项目(CZKYF2025-1-B008)；黑龙江省水利科学研究院重点实验室开放基金(DT2024A01)。

参考文献

[1] Kennedy, J. and Eberhart, R. (1995) Particle Swarm Optimization. *Proceedings of ICNN'95—International Conference on Neural Networks*, Perth, 27 November-1 December 1995, 1942-1948. https://doi.org/10.1109/icnn.1995.488968

[2] Mirjalili, S., Mirjalili, S.M. and Lewis, A. (2014) Grey Wolf Optimizer. *Advances in Engineering Software*, **69**, 46-61. https://doi.org/10.1016/j.advengsoft.2013.12.007

[3] Hashim, F.A. and Hussien, A.G. (2022) Snake Optimizer: A Novel Meta-Heuristic Optimization Algorithm. *KnowledgE−Based Systems*, **242**, Article ID: 108320. https://doi.org/10.1016/j.knosys.2022.108320

[4] Mirjalili, S. and Lewis, A. (2016) The Whale Optimization Algorithm. *Advances in Engineering Software*, **95**, 51-67. https://doi.org/10.1016/j.advengsoft.2016.01.008

[5] Trojovský, P. and Dehghani, M. (2022) Pelican Optimization Algorithm: A Novel NaturE−Inspired Algorithm for Engineering Applications. *Sensors*, **22**, Article 855. https://doi.org/10.3390/s22030855

[6] Xue, J. and Shen, B. (2022) Dung Beetle Optimizer: A New Meta-Heuristic Algorithm for Global Optimization. *The Journal of Supercomputing*, **79**, 7305-7336. https://doi.org/10.1007/s11227-022-04959-6

[7] Zhang, R. and Zhu, Y. (2023) Predicting the Mechanical Properties of Heat-Treated Woods Using Optimization-Algorithm-Based Bpnn. *Forests*, **14**, Article 935. https://doi.org/10.3390/f14050935

[8] Shen, Q., Zhang, D., Xie, M. and He, Q. (2023) Multi-Strategy Enhanced Dung Beetle Optimizer and Its Application in ThreE−Dimensional UAV Path Planning. *Symmetry*, **15**, Article 1432. https://doi.org/10.3390/sym15071432

[9] Tu, N. and Fan, Z. (2023) IMODBO for Optimal Dynamic Reconfiguration in Active Distribution Networks. *Processes*, **11**, Article 1827. https://doi.org/10.3390/pr11061827

[10] 郭琴, 郑巧仙. 多策略改进的蜣螂优化算法及其应用[J]. 计算机科学与探索, 2024, 18(4): 930-946.

[11] 潘劲成, 李少波, 周鹏, 等. 改进正弦算法引导的蜣螂优化算法[J]. 计算机工程与应用, 2023, 59(22): 92-110.

[12] 隋东, 杨振宇, 丁松滨, 等. 基于 EMSDBO 算法的无人机三维航迹规划[J]. 系统工程与电子技术, 2024, 46(5): 1756-1766.

[13] Chai, Y., Sun, X. and Ren, S. (2023) Chaotic Sparrow Search Algorithm Based on Multi-Directional Learning. *Computer Engineering and Applications Journal*, **59**, 81-91.

[14] Luo, Y. (2014) Critical Chain Project Management Based on the Improved Ant Colony Algorithm. *Computer Engineering & Science*, **36**, 1722.

[15] Dehghani, M. and Trojovský, P. (2023) Osprey Optimization Algorithm: A New Bio-Inspired Metaheuristic Algorithm for Solving Engineering Optimization Problems. *Frontiers in Mechanical Engineering*, **8**, Article 1126450. https://doi.org/10.3389/fmech.2022.1126450

[16] Yang, X., Liu, J., Liu, Y., Xu, P., Yu, L., Zhu, L., et al. (2021) A Novel Adaptive Sparrow Search Algorithm Based on Chaotic Mapping and T-Distribution Mutation. *Applied Sciences*, **11**, Article 11192. https://doi.org/10.3390/app112311192

[17] 郑婷婷, 刘升, 叶旭. 自适应 t 分布与动态边界策略改进的算术优化算法[J]. 计算机应用研究, 2022, 39(5): 1410-1414.

[18] Chopra, N. and Mohsin Ansari, M. (2022) Golden Jackal Optimization: A Novel NaturE−Inspired Optimizer for Engineering Applications. *Expert Systems with Applications*, **198**, Article ID: 116924. https://doi.org/10.1016/j.eswa.2022.116924

[19] Wang, J., Wang, W., Hu, X., Qiu, L. and Zang, H. (2024) Black-winged Kite Algorithm: A NaturE−Inspired Meta-Heuristic for Solving Benchmark Functions and Engineering Problems. *Artificial Intelligence Review*, **57**, Article No. 98. https://doi.org/10.1007/s10462-024-10723-4

[20] Yao, X., Liu, Y. and Lin, G.M. (1999) Evolutionary Programming Made Faster. *IEEE Transactions on Evolutionary Computation*, **3**, 82-102. https://doi.org/10.1109/4235.771163

[21] Wilcoxon, F. (1992) Individual Comparisons by Ranking Methods. In: Kotz, S. and Johnson, N.L., Eds., *Breakthroughs in Statistics*, Springer New York, 196-202. https://doi.org/10.1007/978-1-4612-4380-9_16

[22] Ray, T. and Saini, P. (2001) Engineering Design Optimization Using a Swarm with an Intelligent Information Sharing among Individuals. *Engineering Optimization*, **33**, 735-748. https://doi.org/10.1080/03052150108940941

[23] Coello Coello, C.A. (2000) Use of a Self-Adaptive Penalty Approach for Engineering Optimization Problems. *Computers in Industry*, **41**, 113-127. https://doi.org/10.1016/s0166-3615(99)00046-9

[24] Zhou, Y., Zhang, S., Luo, Q. and Abdel-Baset, M. (2019) CCEO: Cultural Cognitive Evolution Optimization Algorithm. *Soft Computing*, **23**, 12561-12583. https://doi.org/10.1007/s00500-019-03806-w