Computer Science and Application 计算机科学与应用, 2024, 14(5), 206-218
Published Online May 2024 in Hans. https://www.hanspub.org/journal/csa
https://doi.org/10.12677/csa.2024.145129

Hans汉斯

具有逃脱机制的蚁狮算法及其应用

梁　晨，张立溥*

浙江传媒学院媒体工程学院，浙江　杭州

收稿日期：2024年4月26日；录用日期：2024年5月24日；发布日期：2024年5月31日

摘　要

本文介绍了一种受蚂蚁逃脱蚁狮陷阱方式启发而设计的具有逃逸机制的蚁狮算法。该算法的特点是在蚂蚁的随机游走中引入了曲线位移，增强了它们逃离陷阱的能力。我们将本算法与其他三种蚁狮算法进行了比较，并进行了大量测试，结果显示新构建的蚁狮算法具有更优越的性能。此外，我们将该算法与K-Means聚类算法的质心选择过程相结合，以提高其聚类性能。进一步，我们将这种新算法应用于玻璃产品的分类，并取得了良好的结果。本研究展示了将自然机制融入算法设计的有效性，并在现实问题中得到了实际的应用。

关键词

蚁狮算法，逃脱机制，聚类算法，玻璃制品分类

The Antlion Algorithm with Escape Mechanism and It's Applications

Chen Liang, Lipu Zhang*

College of Media Engineering, Communication University of Zhejiang, Hangzhou Zhejiang

Received: Apr. 26th, 2024; accepted: May 24th, 2024; published: May 31st, 2024

Abstract

In this paper, we present an antlion algorithm with an escape mechanism inspired by the way ants avoid antlions' traps. Our algorithm features a curved displacement in the random walk path of ants, which allows them to escape more efficiently. We compared our algorithm with three other antlion algorithms and conducted extensive testing, which showed that our modified antlion algorithm has superior performance. Furthermore, we integrated our algorithm with the centroid se-

*通讯作者。

文章引用: 梁晨, 张立溥. 具有逃脱机制的蚁狮算法及其应用[J]. 计算机科学与应用, 2024, 14(5): 206-218.
DOI: 10.12677/csa.2024.145129

lection process of the K-Means clustering algorithm to improve its clustering performance. We applied this new algorithm to the classification of glass products and achieved good results. Our research demonstrates the effectiveness of incorporating natural mechanisms into algorithm design and how it can lead to practical applications in real-world problems.

Keywords

Antlion Algorithm, Escape Mechanism, Clustering Algorithm, Classification of Glass Products

1. 引言

群智能优化算法[1]是一种仿生优化算法，它模拟自然种群的行为和规则，利用合作和进化机制，通过简单个体之间的相互作用来完成给定任务。随着时间的推移，群智能优化算法经历了一系列的发展和演变。遗传算法(Genetic Algorithm, GA) [2]于 1975 年问世，是最早提出的一种群智能算法。GA 通过模拟遗传演化和自然选择过程来实现全局搜索，然而，在收敛速度和解的质量方面存在局限性。随后，蚁群优化算法(Ant Colony Optimization, ACO) [3]于 1992 年提出，模拟了蚂蚁的觅食和通信行为，具有良好的全局搜索能力，特别适用于离散优化问题，但在连续优化问题上可能性能受限。粒子群优化算法(Particle Swarm Optimization, PSO) [4]于 1995 年提出，通过模拟群体中粒子的移动来寻找最优解，虽然收敛快速并具有鲁棒性，但容易陷入局部最优。人工鱼群算法(Artificial Fish Swarm Algorithm, AFSA) [5]于 2002 年出现，模拟了鱼类的觅食和集群行为，并在解决连续优化问题上表现出一定的优势。然而，在高维问题中，AFSA 的性能可能受到影响。人工蜂群算法(Artificial Bee Colony, ABC) [6]于 2005 年提出，模拟了蜜蜂的觅食行为。ABC 具有鲁棒性和多样性，但对初始条件敏感。有关群体智能优化算法的更多文章可参考[7] [8]。

蚁狮优化算法(ALO) [9]是一种新型的群智能算法，最初由 Seyedali Mirjalili 提出，模拟了蚁狮在沙坑中捕捉蚂蚁的特殊狩猎方法。ALO 具有参数少、收敛速度快等优点，并广泛应用于工程领域。然而，在 ALO 算法中，随机游走的蚂蚁根据轮盘赌策略选择蚁狮，其中选择概率由每个蚁狮的适应度确定。这种选择策略可能导致更多的蚂蚁随机在适应度较高的蚁狮周围徘徊。随着蚁狮适应度逐渐增加，蚂蚁大概率会被限制在一些精英蚁狮附近，导致搜索范围减少，算法陷入局部最优解。为了解决这些问题，对 ALO 算法提出了几项改进。例如，Kilic、Yüzgeç 和 Karakuzu [10]通过将轮盘赌选择方法替换为锦标赛选择方法，提高了蚂蚁随机漫步模型的效率，显著减少了算法的运行时间。此外，Emary 和 Zawbaa [11]将 Levy 飞行随机游动方法应用于 ALO 算法，取代了蚂蚁使用的正态分布随机游走，极大地提高了算法的优化效率。

该算法的灵感源于观察到蚂蚁在被蚁狮捕捉时的逃逸行为。在捕猎时，蚁狮会向坑穴附近抛掷沙子，以阻止蚂蚁逃跑。而蚂蚁为了躲避蚁狮的捕捉，除了会向坑穴外进行垂直方向的移动，还会本能地进行横向游走，如此通过垂直和横向移动的组合形成的曲线逃逸路线可以有效地帮助蚂蚁避免被捕捉。这种行为启发了我们将曲线位移纳入蚂蚁随机行走路径的设计中，从而设计了一个具有逃逸机制的蚁狮算法。

本文结构如下：第 1 节介绍了本文的研究背景和动机；第 2 节详细介绍了经典蚁狮算法的关键步骤；第 3 节解释了如何实现蚂蚁的逃逸机制并将其嵌入到经典蚁狮算法中。我们还使用 CEC2005 和 CEC2017

测试函数将新设计的算法与三种现有蚁狮算法进行了比较。第 4 节将新设计的算法与 K-Means 聚类算法结合，以提高质心选择的效率。我们使用标准 UCI 数据集将此算法与类似的聚类算法进行了比较。第 5 节提供了一个具体的应用示例，我们使用新设计的聚类算法解决了古玻璃产品分类问题，并取得了令人满意的结果。最后，我们对文章进行了总结，并对之后的研究方向提出设想。

2. 经典蚁狮算法

蚁狮一般生活在干燥、多沙的地区，主要的猎物是蚂蚁。它们会挖出类似于漩涡状的锥形沙坑，当作自己的巢穴和狩猎用的陷阱。这些沙坑的边缘十分陡峭，使猎物很容易掉进去。故蚁狮大多时间会在坑底等待猎物接近，当猎物掉进去时，它们会立即试图抓住它。为了防止猎物逃跑，蚁狮会向陷阱边缘投掷沙子，使猎物滑入坑底，最终被蚁狮食用。完成进食后，蚁狮会修缮陷阱，并准备下一次捕猎。

蚁狮算法是一种创新优化方法[9]，它模拟了蚂蚁的觅食过程和蚂蚁的狩猎行为。该算法设置了两组集群，蚂蚁和蚁狮。在每次迭代中，会通过轮盘赌策略选择出蚁狮，并将其作为临时最优解，蚂蚁会围绕着选中的蚁狮进行随机游走，寻找更优解。当搜索到优于选中蚁狮的解后，蚁狮会将蚂蚁吃掉并取代其位置，更新最优解。随着迭代次数的增加，蚂蚁随机游走的范围会逐渐缩小，并逐渐向全局最优解靠近，算法的精确度也会逐渐提高。

蚁狮算法主要包括两个关键策略：蚂蚁随机游走和轮盘赌选择精英策略。现在，让我们深入研究这两种策略的细节。

2.1. 蚂蚁的随机游走

蚁狮算法模拟了随机游走的蚂蚁在搜索过程中探索未知搜索空间的关键行为。在该算法中，每只蚂蚁在搜索空间内随机选择一个起点，并根据特定的概率规则确定下一步的移动方向。在随机游走过程中，蚂蚁会根据自己的经验和信息素不断调整自己的移动方向，直到找到最佳解决方案或达到搜索时间的上限。蚁狮算法利用蚂蚁的随机游走，有效避免了局域最优解的困点，提高了搜索效率，在各种优化问题中取得了较为理想的结果。

在蚁狮算法中，蚂蚁随机游走[9]是通过一个基本的统计函数实现的，该函数表示为：

$$X(t)=\left[0,ssum\left(2\gamma(t_1)-1\right),\cdots ssum\left(2\gamma(t_n)-1\right)\right], \tag{1}$$

其中， $X(t)$ 为随机游走的步数集合； t 为随机游走的步数； $ssum$ 为计算累加和； $\gamma(t)$ 是定义的一个随机函数，公式如下：

$$\gamma(t)=\begin{cases}1, rand>0.5\\ 0, rand\le 0.5\end{cases}, \tag{2}$$

其中， $rand$ 是在[0, 1]间取的随机数。

而在蚁狮优化算法中，由于蚁狮设置陷阱，限制了蚂蚁随机游走的范围，故对公式(1)进行了修改，以满足行走范围的局限性[9]。修改后的公式如下：

$$X_i^t=\frac{\left(X_i^t-a_i\right)\left(d_i^t-c_i^t\right)}{\left(b_i-a_i\right)}+c_i^t, \tag{3}$$

其中， X_i^t 为蚂蚁在第 t 次迭代中第 i 维的位置； a_i 和 b_i 分别指第 i 维中随机游走的最小值和最大值； c_i^t 和 d_i^t 为第 t 次迭代时在第 i 维中随机游走的最小值和最大值。

因蚂蚁游走的路线会受到蚁狮位置和陷阱尺寸的影响，故参数 c_i^t 和 d_i^t 的值根据特定的规则确定[9]。

$$c_i^t = Antlion_j^t + c^t \text{ and } \mathrm{d}_i^t = Antlion_j^t + d^t, \tag{4}$$

其中，c^t 和 d^t 分别是第 t 次迭代中最小和最大的向量；$Antlion_j^t$ 是在第 t 次迭代中第 j 只蚁狮的位置。

当蚂蚁掉进陷阱后，蚁狮会向沙坑边缘抛掷沙子，并把蚂蚁的向下拖拽，导致蚂蚁随机游走的范围会迅速缩小。我们采用如下方程模拟这种现象：

$$c_i^t = \frac{c^t}{I}, d_i^t = \frac{d^t}{I} \text{ and } I = 10^{\omega}\frac{t}{T}, \tag{5}$$

其中，变量 I 表示比例系数；T 表示允许的最大迭代次数；ω 是一个参数，其值由 t 的大小决定。ω 和 t 之间的关系如表 1 所示。

Table 1. The value of parameters
表 1. 参数 ω 的值

t	ω
$t \le 0.5T$	2
$0.5T < t \le 0.75T$	3
$0.75T < t \le 0.9T$	4
$0.9T < t \le 0.95T$	5
$0.95T < t \le T$	6

当表 1 中的迭代次数增加时，蚂蚁随机游走的半径减小，保证了 ALO 算法的收敛性。

2.2. 精英策略和轮盘赌策略

每次迭代后，算法都会选择适应度最高的蚁狮作为精英蚁狮，之后蚂蚁会通过轮盘赌策略选择一只蚁狮，并围绕该蚁狮进行随机游走。蚂蚁的位置由下述公式确定[9]。

$$Ant_i^t = \frac{R_A^t(m) + R_E^t(m)}{2}, \tag{6}$$

其中，Ant_i^t 是指第 i 个蚂蚁在第 t 次迭代中的位置；$R_A^t(m)$ 表示在 t 次迭代中，蚂蚁跟据轮盘赌策略选中蚁狮后，围绕其走 m 步时得出的值；类似地，$R_E^t(m)$ 表示在第 t 次迭代中，蚂蚁围绕精英蚁狮走 m 步得出的值；变量 m 取自蚂蚁在随机游走期间所走步数范围内的任意值。

然而，由于轮盘赌策略是根据蚁狮的适应度为其分配概率，这可能会导致更多的蚂蚁围绕较高适应度的蚁狮进行游走，让算法有陷入局部最优的危险。

蚁狮优化算法将这两种策略相结合，对解空间进行更为有效地探索与开发，以求找到可能的最佳解。

2.3. ALO 算法主要步骤

ALO 算法的具体步骤如下：

1) 初始化数据：设置两个种群的参数，包括个体数、维度和迭代次数，并在指定范围内初始化蚁狮和蚂蚁的位置，计算它们各自的适应度。

2) 确定精英蚁狮：选择初始化后蚁狮种群中适应度最好的蚁狮作为精英蚁狮。

3) 轮盘选择：使用轮盘赌策略为每只蚂蚁选择一只蚁狮，根据选中蚁狮的位置更新蚂蚁的参数值，并让蚂蚁围绕所选蚁狮随机游走，更新蚂蚁位置。

4) 计算适应度：计算蚁狮和蚂蚁的适应度值，根据蚂蚁的位置和适应度更新蚁狮位置，两个种群中适应度最好的成为新精英蚁狮。

5) 判断是否终止：检查是否满足算法终止条件。如果满足，输出结果，返回精英蚁狮的位置及其适应度值，并结束迭代；否则，返回步骤 3。

下面我们将介绍如何实现蚂蚁的随机游走。

3. 具有逃脱机制的蚁狮算法

为了避免过多蚂蚁在某些蚁狮周围进行无意义地游走，我们考虑修改它们的行走策略，以扩大搜索范围。

3.1. 逃脱机制

在现实捕猎的过程中，蚂蚁落入陷阱后会尽最大努力远离蚁狮并爬出陷阱。同许多其他动物相似，蚂蚁在垂直方向上移动的同时，本能地会在水平方向上进行游走，从而形成曲线状的运动轨迹。为了实现对这一现象的模拟，我们将蚂蚁的随机游走和曲线位移进行结合，使用正弦和余弦函数对其逃脱策略进行建模。

下面，我们将详细描述这个过程，并用数学公式来展现蚂蚁挣扎逃脱的过程。首先，由于蚂蚁具有二维视觉感知，故它们逃跑的方向是随机的。我们用 $\gamma_2 \in \left(0,360^\circ\right)$ 来表示蚂蚁的逃跑方向。一般来说，当蚂蚁被陷阱困住后，会有两种挣扎逃脱的方式。一种方法是在逃跑时逐渐增加力量，直到达到极限，再慢慢耗尽；另一种方法则是在最初进行最有力的挣扎，然后逐渐削弱。我们可以分别用 $\sin(\gamma)$ 和 $\cos(\gamma)$ 来表示这两种方式。至于具体选择哪种方法，我们通过使用随机变量 γ_3，采用概率的形式来模拟蚂蚁选择方式的随机性。此外，由于蚂蚁在逐渐陷入困境的过程中，逃跑的力量会逐渐减弱，我们可以在逃避规则里乘上一个衰减系数，用 $\gamma_1 = a - a*(t/T)$ 来表示，此处 a 代表初始强度，t 代表当前迭代次数，T 代表迭代总数。考虑到不同的蚂蚁可能有着不同的斗争强度，我们在公式的末尾乘一个随机参数 $D = \left|\gamma_4 * Ant_i^t - Ant_i^t\right|$，$\gamma_4$ 为一个随机数，Ant_i^t 表示蚂蚁 i 在第 t 次迭代中的状态。以上为蚂蚁挣扎逃避过程的描述，具体的参数设置和算法实现可能会根据实际的应用与问题而有所不同。

在下文中，我们将使用公式(7)至(10)来描述蚂蚁逃跑的过程。这种方法对蚂蚁的逃跑行为进行了更为真实的模拟，可以让蚂蚁的探索范围更广泛，从而降低陷入局部最优的风险。

$$\mathrm{Ant}_i^{t+1} = \begin{cases} Ant_i^t + \gamma_1 * \sin\gamma_2 * D & \gamma_3 < 0.5 \\ Ant_i^t + \gamma_1 * \cos\gamma_2 * D & \gamma_3 \geq 0.5 \end{cases}, \tag{7}$$

$$D = \left\{\left|\gamma_4 * Ant_i^t - Ant_i^t\right| \middle| i = 1,2,\cdots,n\right\}, \tag{8}$$

$$\gamma_1 = a - a*\left(\frac{t}{T}\right), \tag{9}$$

$$\gamma_2 \in \left(0^\circ, 360^\circ\right), \tag{10}$$

其中，γ_1~γ_4 为参数，γ_1 会随着迭代次数的变化而变化，γ_3 和 γ_4 为 0 到 1 之间均匀分布的随机数；a 为常数，实验中取值为 1。

值得注意的是，正弦余弦策略可以直接应用于算法设计[12] [13]，对此我们将其作为搜索方向来模拟蚂蚁的曲线运动。

3.2. 具有逃脱机制的 ALO 算法主要步骤

改进的算法包括以下步骤：

1) 初始化数据：在规定的范围内设置参数，如蚁狮和蚂蚁的数量、维度和迭代次数。随机为蚁狮和

蚂蚁分配位置，并计算它们各自的适应度值。

2) 确定精英蚁狮：从初始种群中选择适应度最高的蚁狮作为精英蚁狮。

3) 轮盘选择：使用轮盘赌策略为每只蚂蚁选择一只蚁狮，并根据蚁狮的位置更新参数值。蚂蚁围绕所选蚁狮进行随机游走，更新位置。

4) 引入正弦余弦策略：跟据公式(7)更新蚂蚁位置。

5) 计算适应度：计算蚂蚁和蚁狮的适应度值，并依据蚂蚁的位置和适应度更新蚁狮位置，选择所有个体中适应度最好的成为新精英蚁狮。

6) 判断是否终止：判断是否满足算法的终止条件。如果满足，输出结果，返回精英蚁狮位置及其适应度值，并结束迭代；否则，返回步骤 3。

3.3. 实验与分析

在本节中，我们比较了四种蚁狮优化算法在 CEC2005 和 CEC2017 部分测试函数上的性能。这四种算法分别为：新提出的具有逃跑机制的蚁狮优化算法(ALO-E)，结合柯西分布的蚁狮优化算法(CALO) [14]，自适应蚁狮优化算法(PSALO) [15]和经典蚁狮优化算法(ALO)。我们通过评估这些算法对测试函数的求解结果来分析它们的有效性。实验的运行平台为 PyCharm Community Edition 2022.3.3，操作系统为 Windows 10，内存大小为 16 G，CPU 为 i5-1035G1。

两组测试函数如表 2 和表 3 所示。

Table 2. Some test functions from CEC2017
表 2. 部分 CEC2017 测试函数

NO.	Functions	$F_i^* = F_i(x^*)$
F16	Shifted and Rotated Bent Cigar Function	100
F17	Shifted and Rotated Zakharov Function	200
F18	Shifted and Rotated Rosenbrock's Function	300
F19	Shifted and Rotated Rastrigin's Function	400
F20	Shifted and Rotated Expended Scaffer's F6 Function	500
F21	Shifted and Rotated Lunacek Bi_Rastrigin's Function	600
F22	Shifted and Rotated Non-Continuous Rastrigin's Function	700
F23	Shifted and Rotated Levy Function	800
F24	Shifted and Rotated Schwefel's Function	900

Table 3. Some test functions from CEC2005
表 3. 部分 CEC2005 测试函数

NO	Functions	D	Search space	f_{min}
F1	$F_1(x)=\sum_{i=1}^{n}x_i^2$	30	$[-100,100]^D$	0
F2	$F_2(x)=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	30	$[-10,10]^D$	0
F3	$F_3(x)=\sum_{i=1}^{n}\left(\sum_{j=1}^{i}x_j\right)^2$	30	$[-100,100]^D$	0
F4	$F_4(x)=\max_i\{\lvert x_i\rvert,1\le i\le n\}$	30	$[-100,100]^D$	0
F5	$F_5(x)=\sum_{i=1}^{n-1}\left[100(x_{i+1}-x_i)^2+(x_i-1)^2\right]$	30	$[-30,30]^D$	0
F6	$F_6(x)=\sum_{i=1}^{n}([x_i+0.5])^2$	30	$[-100,100]^D$	0
F7	$F_7(x)=\sum_{i=1}^{n}ix_i^4+random[0,1)$	30	$[-1.28,1.28]^D$	0

续表

F8	$F_8(x)=\sum_{i=1}^{n}\left[x_i^2-10\cos(2\pi x_i)+10\right]$	30	$[-5.12,5.12]^D$	0
F9	$F_9(x)=-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)+20+e$	30	$[-32,32]^D$	0
F10	$F_{10}(x)=\frac{1}{4000}\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	30	$[-600,600]^D$	0
F11	$F_{11}(x)=\sum_{i=1}^{n}u(x_i,5,100,4)$ $+0.1\left\{\sin^2(3\pi x_1)+\sum_{i=1}^{n-1}(x_i-1)^2\left[1+\sin^2(3\pi x_{1i+1})\right]+(x_n-1)^2\left[1+\sin^2(3\pi x_n)\right]\right\}$	30	$[-50,50]^D$	0
F12	$F_{12}(x)=\left(\frac{1}{500}+\sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ij})^6}\right)^{-1}$	2	$[-65.536,65.536]^D$	0.998
F13	$F_{13}(x)=4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_1x_2-4x_2^2+4x_2^4$	2	$[-5,5]^D$	−1.031 6285
F14	$F_{14}(x)=\left(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6\right)^2+10\left(1-\frac{1}{8\pi}\right)\cos x_1+10$	2	$[-5,10]\times[0,15]$	0.398
F15	$F_{15}(x)=\left[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)\right]$ $\times\left[30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)\right]$	2	$[-2,2]^D$	3

在本节实验中，我们设置的参数如下：两个种群的大小均为 20，最大迭代次数为 10000。为了确保实验的准确性，我们对每个算法进行了 20 次独立运行，并计算每个基准函数的平均值和标准差。实验结果如表 4 所示。

Table 4. Comparison of results for solving test functions
表 4. 求解测试函数结果对比

F		ALO	ALO-E	CALO	PSALO
$F_1(x)$	mean	5.9142e-09	**9.9392e-36**	6.4911e-09	5.0791e-09
	std	7.0441e-18	**1.8772e-69**	1.4942e-17	2.0195e-17
$F_2(x)$	mean	0.3562	**4.1272e-17**	0.4009	0.3587
	std	0.0564	**3.1104e-32**	0.0542	0.0314
$F_3(x)$	mean	0.2152	**5.0922e-30**	0.2334	0.2490
	std	0.0131	**3.9943e-58**	0.0131	0.0170
$F_4(x)$	mean	0.1612	**8.0806e-13**	0.1464	0.1511
	std	3.1954e-03	**1.2403e-23**	2.5162e-03	2.3087e-03
$F_5(x)$	mean	1.4540	**0.0**	1.3863	**0.0**
	std	40.1694	**0.0**	36.5178	**0.0**
$F_6(x)$	mean	5.2497e-09	1.3237e-05	**5.2017e-09**	5.4318e-09
	std	**1.1620e-17**	1.9473e-10	1.2743e-17	1.9766e-17
$F_7(x)$	mean	9.1821e-02	**1.0554e-04**	0.1014	0.0890
	std	1.6104e-03	**7.4075e-09**	1.3771e-03	1.8421e-03
$F_8(x)$	mean	93.7746	**3.2693e-13**	91.6354	75.4177
	std	521.1842	**6.9514e-26**	1.052e + 03	618.6692
$F_9(x)$	mean	3.5153e-04	**1.6778e-13**	3.7426e-04	3.4196e-04
	std	9.0499e-09	**5.3082e-25**	1.0448e-08	1.7344e-08
$F_{10}(x)$	mean	1.5635e-02	**1.8012e-06**	1.8968e-02	1.4343e-03
	std	4.7831e-04	**5.8440e-12**	1.4096e-03	1.7817e-06
$F_{11}(x)$	mean	4.5370	**4.6233e-05**	4.4605	2.9738
	std	6.5412	**2.2411e-09**	3.6082	2.0936

续表

$F_{12}(x)$	mean	0.0633	0.0730	0.0487	**0.0195**
	std	2.1570e-03	1.7784e-03	2.3703e-03	**1.5170e-03**
$F_{13}(x)$	mean	0.2720	0.4068	0.4203	**0.2260**
	std	0.1160	0.2909	0.3035	**0.0961**
$F_{14}(x)$	mean	0.3979	0.3933	**0.3979**	0.3969
	std	6.5029e-27	0.0298	1.0619e-26	**2.2989e-27**
$F_{15}(x)$	mean	3.0010	**3.0000**	3.0001	**3.0000**
	std	3.1338e-25	3.4628e-25	5.9927e-25	**4.0530e-26**
$F_{16}(x)$	mean	2.9295e + 11	**2.8995e + 11**	2.9175e + 11	2.9085e + 11
	std	4.6716e + 17	**1.8559e + 17**	7.8751e + 17	1.1619e + 18
$F_{17}(x)$	mean	8.6187e + 04	7.4335e + 04	6.1174e + 04	**3.4994e + 04**
	std	8.7787e + 08	2.1095e + 08	1.2089e + 09	**7.1123e + 07**
$F_{18}(x)$	mean	5.7089e + 03	5.7154e + 03	5.6739e + 03	5.6394e + 03
	std	422.9201	**61.6644**	457.0389	369.7308
$F_{19}(x)$	mean	710.0171	710.8291	708.0873	**706.2653**
	std	5.0799	**1.2499**	4.2630	3.2240
$F_{20}(x)$	mean	709.0734	**704.5987**	708.4356	708.6477
	std	8.4226	**5.9179**	23.0464	8.2936
$F_{21}(x)$	mean	748.0431	**743.5674**	748.8433e	747.7458
	std	62.0121	28.1113	**27.5246**	30.2064
$F_{22}(x)$	mean	931.0921	**922.8259**	925.8775	923.2593
	std	**1.4813**	7.5416	9.2972	4.6186
$F_{23}(x)$	mean	2.5679e + 3	**2.2298e + 3**	2.2387e + 3	2.2623e + 3
	std	1.3263e + 4	3.5461e + 4	**3.2139e + 3**	3.2863e + 4

下面，我们对四种算法在求解上述 23 个函数时取得最佳均值和标准差的次数进行统计分析，如图 1 所示。其中，ALO-E 算法的求解大部分最优且最为稳定，表明了具有逃跑机制的蚁狮算法的有效性。

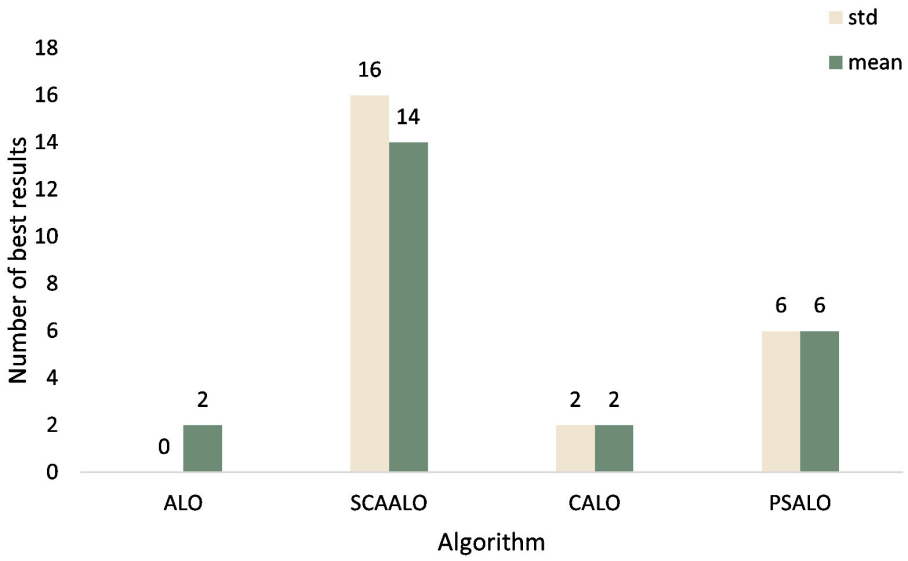

Figure 1. The statistical analysis for the number of times with best results
图 1. 最佳结果次数的统计分析

为了更好地展示 ALO-E 算法的优越性，我们给出了 ALO-E、CALO、PSALO 和 ALO 算法在 1000 次迭代下求解上述部分测试函数结果的收敛图，如图 2 所示。

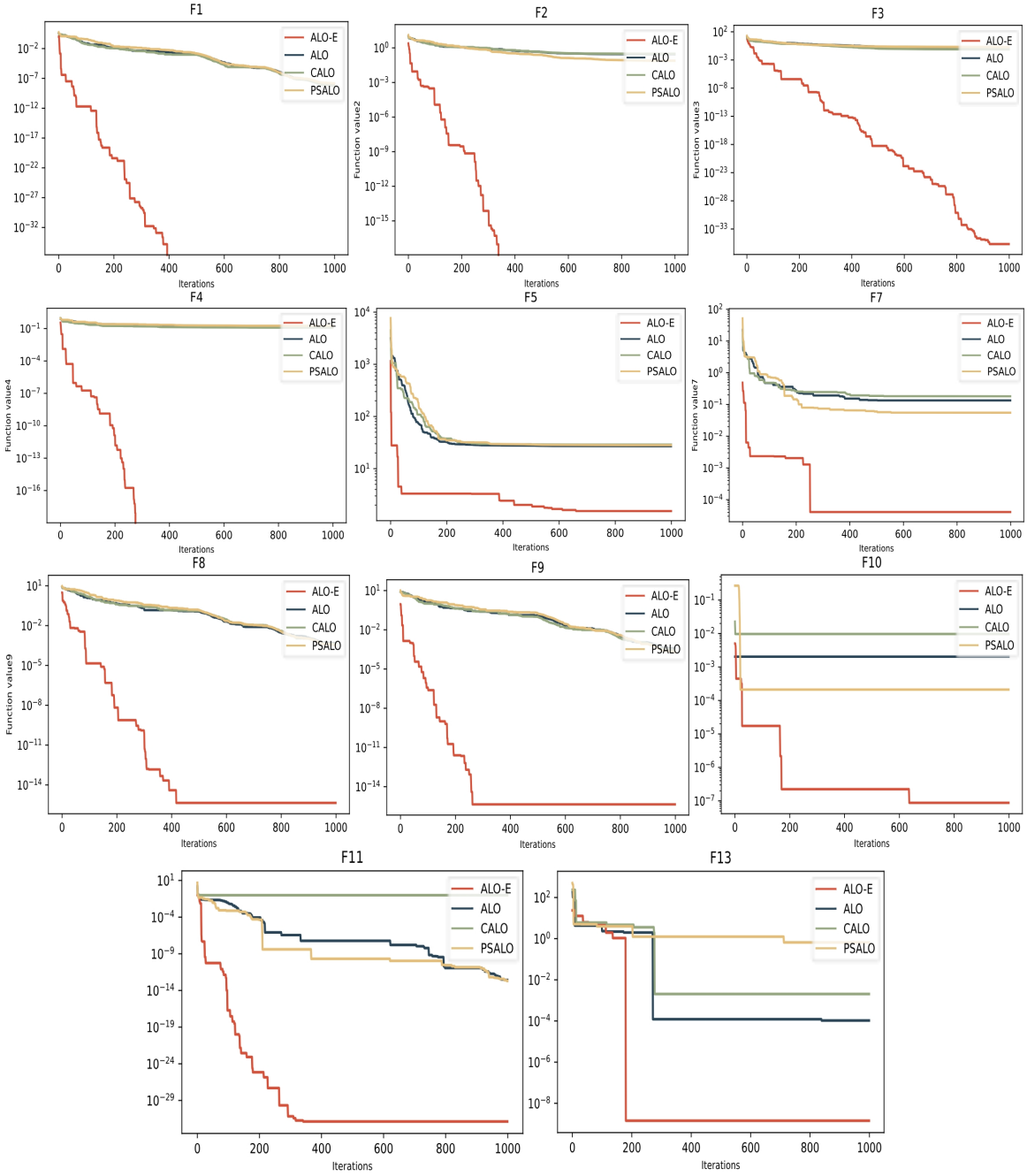

Figure 2. Partial test function convergence result
图 2. 部分测试函数收敛结果

4. 结合 ALO-E 的 K-Means 聚类算法

K-Means 算法[16]是最常用的聚类算法之一，其核心为选择簇的数量并随机分配初始质心，再基于每个数据点和每个质心之间的欧式距离，将所有数据点分配到其对应的簇。通过一次次迭代，把各个簇的质心放在其他数据点的平均值处，并基于相同的原理对数据点进行重新分类，直到获得最佳聚类结果。值得注意的是，更新质心需要计算对应聚类中所有数据点的坐标平均值，这可能导致算法效率较低。

4.1. 改进 K-Means 聚类算法

为了提高 K-Means 聚类中质心更新的效率，我们将 ALO-E 算法应用到该过程中，即在初始化质心之后，采用 ALO-E 算法来获得新的质心。

以下是改进聚类算法的主要步骤：

1) 随机选取 k 个质心。
2) 计算每个数据点和每个质心之间的欧式距离，并根据该距离将样本分配到相应的聚类。
3) 运用改进的蚁狮算法更新质心。
4) 判断是否已达到最大迭代次数。如果达到，则结束迭代并输出聚类结果；否则，返回步骤 3。

4.2. 实验与分析

为了证明改进聚类算法的有效性和优越性，我们将 ALO-E 算法结合 K-Means (Kmeans-ALO-E)与传统的 K-Means 算法、粒子群优化算法结合 K-Means (Kmeans-PSO) [17]进行了比较分析。本次实验从 UCI 中选择了四个数据集：Iris、Wine、Seeds 和 Thyroid，具体信息如表 5 所示。

Table 5. The information for the four datasets
表 5. 四个数据集的信息

Datasets	Number of samples	Number of categories	Number of features
Iris	150	3	4
Wine	178	3	13
Seeds	210	3	7
Thyroid	215	3	5

为了对算法性能进行评估，我们使用了三个评估指标：召回率(recall)、准确度(precision)和 F1 精度，计算公式如下：

$$recall = \frac{TP}{P}, \tag{11}$$

$$precision = \frac{TP}{TP + FP}, \tag{12}$$

$$F1 = \frac{2 * precision * recall}{precisiom + recall}。 \tag{13}$$

表 6 至表 9 为三个聚类算法应用在上述四个数据集的结果，展示了每种算法对数据点进行分组的准确性和有效性。我们可以看到，Kmeans-PSO 在四个数据集上的三个评估指标绝大部分优于传统方法；而 Kmeans-ALO-E 算法的评估指标明显优于传统方法，且略优于 Kmeans-PSO。由此我们可以得出，本文所提出的 Kmeans-ALO-E 算法为数据聚类提供了一种稳健有效的替代方案。

Table 6. Results of three clustering algorithms on the Iris dataset
表 6. Iris 数据集上三种聚类算法的结果

	Recall	Precision	F1
K-Means	0.8867	0.8978	0.8853
Kmeans-PSO	0.8400	0.8555	0.8390
Kmeans-ALO-E	0.8933	0.9072	0.8918

Table 7. Results of three clustering algorithms on the Wine dataset
表 7. Wine 数据集上三种聚类算法的结果

	Recall	Precision	F1
K-Means	0.8483	0.8483	0.8483
Kmeans-PSO	0.8665	0.8741	0.8703
Kmeans-ALO-E	0.8905	0.8932	0.8918

Table 8. Results of three clustering algorithms on the Seeds dataset
表 8. Seeds 数据集上三种聚类算法的结果

	Recall	Precision	F1
K-Means	0.8884	0.9038	0.8960
Kmeans-PSO	0.8714	0.9556	0.9116
Kmeans-ALO-E	0.9323	0.8940	0.9127

Table 9. Results of three clustering algorithms on the Thyroid dataset
表 9. Thyroid 数据集上三种聚类算法的结果

	Recall	Precision	F1
K-Means	0.7805	0.8233	0.8013
Kmeans-PSO	0.8757	0.8789	0.8773
Kmeans-ALO-E	0.8884	0.9038	0.8960

为了更直观地展现 Kmeans-ALO-E 算法的聚类性能，我们绘制了该算法在四个数据集上的聚类情况，如图 3 和图 4 所示。

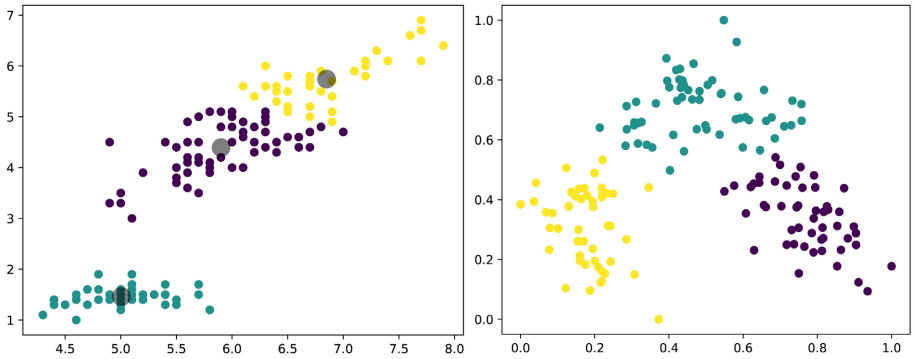

Figure 3. Clustering of the Kmeans-ALO-E algorithm on the Iris and Wine datasets
图 3. Kmeans-ALO-E 算法在 Iris 和 Wine 数据集上的聚类

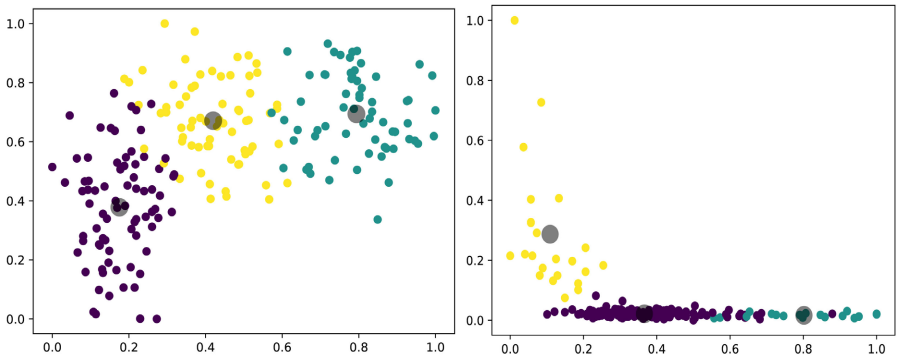

Figure 4. Clustering of the Kmeans-ALO-E algorithm on the Thyroid and Seeds datasets
图 4. Kmeans-ALO-E 算法在 Thyroid 和 Seeds 数据集上的聚类

5. Kmeans-ALO-E 算法在古代玻璃制品分类中的应用

古代玻璃制品的分类源于“古代玻璃制品成分分析和鉴定”问题，关于这个问题的具体描述和数据可以在网站上找到：(www.mcm.edu.cn)。我们利用 Kmeans-ALO-E 算法通过选择合适的化学成分来对每个玻璃类进行亚类划分。

考虑到数据的复杂性，我们首先使用公式(14)对数据集进行归一化，为后续处理和分析数据提供便利。

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} 。 \tag{14}$$

我们使用准确率(ACC)来评估实验结果，其定义公式为：

$$ACC = \frac{TP + TN}{ALL} 。 \tag{15}$$

图 5 展示了 Kmeans-ALO-E 算法在高钾玻璃数据集上的聚类结果。由图可得，高钾玻璃可分为两组：高钾低硅玻璃和高钾高硅玻璃。改进的算法在该数据集上实现了 100%的准确率(ACC)。

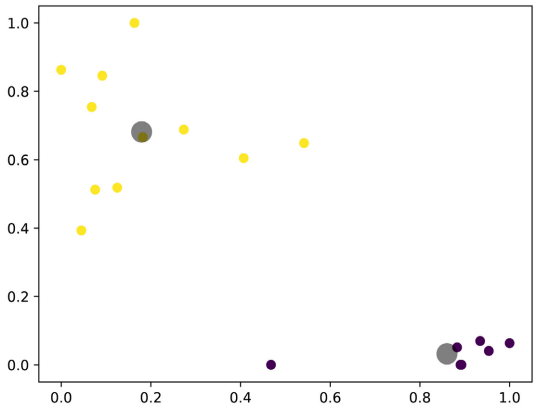

Figure 5. Kmeans-ALO-E clustering in a high-potassium glass dataset
图 5. Kmeans-ALO-E 在高钾玻璃数据集上的聚类

同样，图 6 展示了 Kmeans-ALO-E 算法在铅钡玻璃数据集上的聚类结果。由图可得，铅钡玻璃根据铅含量可分为三个子类：低铅铅钡玻璃、中铅铅钡和高铅铅钡。改进算法在该数据集上的准确率(ACC)为 91%。

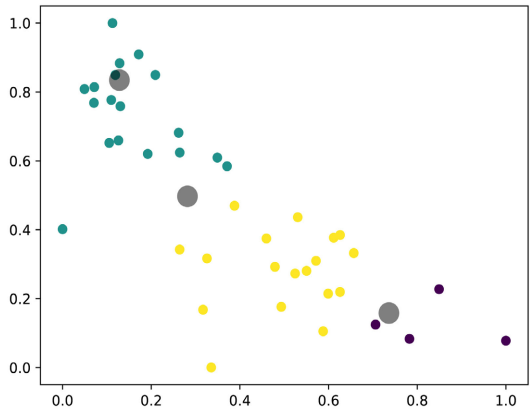

Figure 6. Kmeans-ALO-E clustering in the lead-barium glass dataset
图 6. Kmeans-ALO-E 在铅钡玻璃数据集上的聚类

上述两个古老玻璃制品的分类问题证明了 Kmeans-ALO-E 算法在实际应用中的良好性能。

6. 总结

我们通过模拟蚂蚁被蚁狮追捕时的逃跑路线，将蚂蚁的逃跑策略融入蚁狮优化算法，对算法进行了改进。在进行了大量的数值实验后，我们观察到这种策略对算法的全局优化产生了积极影响。此外，我们将这种新设计的 ALO 算法引入到经典的 K-Means 聚类算法中，对质心更新的过程进行了优化，提高了算法的整体性能。随后，我们成功地将这种新的聚类算法应用于解决实际的问题。

我们的下一个研究目标是检验种群大小对算法性能的影响，并探索该改进算法在其他工程问题中的应用。

参考文献

[1] Chakraborty, A. and Kar, A.K. (2017) Swarm Intelligence: A Review of Algorithms. In: Patnaik, S., Yang, X.S. and Nakamatsu, K. Eds., *Nature-Inspired Computing and Optimization. Modeling and Optimization in Science and Technologies*, Springer, Cham, 475-494. https://doi.org/10.1007/978-3-319-50920-4_19

[2] Goldberg, D.E. (1989) Genetic Algorithms in Search, Optimization, and Machine Learning. Addison-Wesley, Boston.

[3] Dorigo, M., Maniezzo, V. and Colorni, A. (1996) Ant System: Optimization by a Colony of Cooperating Agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B* (*Cybernetics*), **26**, 29-41. https://doi.org/10.1109/3477.484436

[4] Kennedy, J. and Eberhart, R. (1995) Particle Swarm Optimization. *Proceedings of ICNN*'95—*International Conference on Neural Networks*, Perth, WA, Australia, 27 November-01 December 1995, 1942-1948. https://doi.org/10.1109/ICNN.1995.488968

[5] Yazdani, D., NadjaranToosi, A. and Meybodi, M.R. (2010) Fuzzy Adaptive Artificial Fish Swarm Algorithm. *AI* 2010: *Advances in Artificial Intelligence*—23*rd Australasian Joint Conference*, Adelaide, Australia, 7-10 December 2010, 334-343. https://doi.org/10.1007/978-3-642-17432-2_34

[6] Karaboga, D. (2005) An Idea Based On Honey Bee Swarm for Numerical Optimization. Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department. https://api.semanticscholar.org/CorpusID:267873429

[7] Engelbrecht, A.P. (2006) Fundamentals of Computational Swarm Intelligence. John Wiley & Sons, Hoboken, NJ, United States.

[8] Xu, Y., Li, X. and Zhang, L. (2015) The Particle Swarm Shooting Method for Solving the Bratu's Problem. *Journal of Algorithms & Computational Technology*, **3**, 291-302. https://doi.org/10.1260/1748-3018.9.3.291

[9] Seyedali, M. (2015) The Ant Lion Optimizer. *Advances in Engineering Software*, **83**, 80-98. https://doi.org/10.1016/j.advengsoft.2015.01.010

[10] Kilic, H., Yuzgec, U. and Karakuzu, C. (2020) A Novel Improved Antlion Optimizer Algorithm and Its Comparative Performance. *Neural Computing and Applications*, **32**, 3803-3824. https://doi.org/10.1007/s00521-018-3871-9

[11] Emary, E. and Zawbaa, H.M. (2019) Feature Selection via Lèvy Antlion Optimization. *Pattern Analysis and Applications*, **22**, 857-876. https://doi.org/10.1007/s10044-018-0695-2

[12] Seyedali, M. (2016) SCA: A Sine Cosine Algorithm for Solving Optimization Problems. *Knowledge-Based Systems*, **96**, 120-133. https://doi.org/10.1016/j.knosys.2015.12.022

[13] Abualigah, L. and Diabat, A. (2021) Advances in Sine Cosine Algorithm: A Comprehensive Survey. *Artificial Intelligence Review*, **54**, 2567-2608. https://doi.org/10.1007/s10462-020-09909-3

[14] Chenwen, W., Shasha, W. and Xuetong, C. (2023) Fuzzy Clustering Algorithm Combined with Cauchy Distribution and Ant Lion Algorithm. *Computer Engineering and Applications*, **59**, 91-98. https://doi.org/10.3778/j.issn.1002-8331.2205-0436

[15] Wu, C.W., Wang, S.S. and Cao, X.T. (2020) Preferred Strategy Based Self-adaptive Ant Lion Optimization Algorithm. *Pattern Recognition and Artificial Intelligence*, **33**, 121-132. https://doi.org/10.16451/j.cnki.issn1003-6059.202002004

[16] Bock, H.H. (2007) Clustering Methods: A History of K-Means Algorithms. In: Brito, P., Cucumel, G., Bertrand, P. and De Carvalho, F., Eds., *Selected Contributions in Data Analysis and Classification. Studies in Classification, Data Analysis, and Knowledge Organization*, Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-73560-1_15

[17] Kamel, N., Ouchen, I. and Baali, K. (2014) A Sampling-PSO-K-Means Algorithm for Document Clustering. In: Pan, J.S., Krömer, P. and Snášel, V., Eds., *Genetic and Evolutionary Computing. Advances in Intelligent Systems and Computing*, Springer, Cham, 238. https://doi.org/10.1007/978-3-319-01796-9_5