文章编号：2095-2163(2022)11-0216-08 中图分类号：TP301.6 文献标志码：A

# 融合自适应权重与 Levy 飞行的拉丁超立方体海鸥优化算法及应用

梁 静
(贵州大学 大数据与信息工程学院，贵阳 550025)

**摘 要：**针对海鸥优化算法(Seagull optimization algorithm，SOA)收敛速度慢、寻优精度低以及搜索能力差等缺陷，提出一种融合自适应权重与 Levy 飞行的拉丁超立方体海鸥优化算法(Latin Hypercube Seagull Optimization Algorithm based on adaptive Weights and Levy flight，ALLSOA)。首先使用拉丁超立方体初始化海鸥种群，使海鸥种群全空间填充，分布更加均匀；其次在海鸥迁徙阶段，添加自适应权重因子，提高算法的搜索能力，加快算法收敛速度；最后在海鸥觅食阶段，采用 Levy 飞行策略，增加算法的多样性与跳出局部最优的能力，提高寻优精度。本文采用 23 个基准测试函数对改进算法进行测试，并利用图像分割来检验算法的有效性。试验结果表明，ALLSOA 在收敛速度、寻优能力等方面表现更优。

**关键词：**海鸥优化算法；自适应权重；Levy 飞行；拉丁超立方体；图像分割

# A Latin hypercube seagull optimization algorithm and application combining adaptive weights and Levy flight

LIANG Jing
(College of Big Data and Information Engineering, Guizhou University, Guiyang 550025, China)

**【Abstract】**Aiming at the shortcomings of the Seagull optimization algorithm (SOA), such as slow convergence speed, low optimization accuracy and poor search ability, this paper proposes a Latin Hypercube Seagull Optimization Algorithm based on adaptive weights and Levy flight (ALLSOA). Firstly, by using the Latin hypercube to initialize the seagull population, the whole space of the seagull population is filled and the distribution is more uniform; secondly, in the seagull migration stage, an adaptive weight factor is added to improve the global search ability of the algorithm and speed up the convergence of the algorithm; finally, in the seagull foraging stage , using the Levy flight strategy can increase the diversity of the algorithm and the ability to jump out of the local optimum, which also can improve the optimization accuracy. This paper uses 23 benchmark functions to test the improved algorithm, and applies the algorithm to image segmentation. The experimental results show that ALLSOA has better performance in terms of convergence speed and optimization ability.

**【Key words】**seagull optimization algorithm; adaptive weights; Levy flight; Latin hypercube; images segmentation

## 0 引 言

近年来，随着群智能优化算法的发展与应用，越来越多的优化算法也陆续得以提出，例如蜻蜓算法(Dragonfly algorithm，DA)[1]、蝴蝶优化算法(butterfly optimization algorithm，BOA)[2]、灰狼优化算法(Grey Wolf Optimization，GWO)[3]、水循环算法(Water Cycle algorithm，WCA)[4]、樽海鞘算法(Slap swarm algorithm，SSA)[5]、鲸鱼优化算法(Whale Optimization algorithm，WOA)[6]、黏菌优化算法(Slime mould algorithm，SMA)[7]等。这些通过模仿自然界中生物规律来构建不同搜索机制的群智能算法，被广泛运用到各种优化问题中去寻找最优解或者次优解。

Dhiman 等人[8]受海鸥迁徙行为以及迁徙过程中觅食行为的启发，在 2019 年提出海鸥优化算法(Seagull optimization algorithm，SOA)。在 SOA 算法中，海鸥的迁徙行为代表算法的全局搜索能力，觅食行为则代表算法的局部开发能力。该算法原理简单，在工业问题以及分类问题上部署难度低。虽然 SOA 算法在一些方面表现优异，但是海鸥的群体觅食行为会降低算法的多样性以及算法搜索能力，此外该算法还存在收敛速度慢，容易陷入局部最优等缺陷。针对这些问题，文献[9]引入记忆功能提高海鸥算法的搜索能力以及规避局部最优的能力。文献[10]利用混沌初始化、多方向飞行路径等策略改进算法，增加算法的多样性与寻优能力。文献[11]引入自适应 $t$ 分布变异策略，提高算法跳出局部最优的能力。文献[12]使用 *Sigmoid* 函数与黄金正弦机制引导海鸥完成搜寻以及位置更新，提高算法的

基金项目：贵州省科技计划资助项目(黔科合 SY 字[2011]3111)。

作者简介：梁 静(1977-)，女，硕士，讲师，主要研究方向：智能优化算法、无线通信系统。

收稿日期：2022-03-16

收敛速度和精度。

上述改进策略对该算法的多样性、寻优精度以及收敛速度等方面均有一定的提升，为了进一步提高该算法的性能并检验该算法在实际应用中的效果，本文提出了融合自适应权重与 Levy 飞行的拉丁超立方体海鸥优化算法。首先，采用拉丁超立方体抽样的方法产生海鸥种群的初始位置，使海鸥种群在空间上分布更加均匀、避免碰撞，能够加快海鸥寻找目标的速度；其次，在海鸥迁徙阶段引入自适应权重因子，有利于算法在当前位置使用精确的搜索策略，加快算法收敛；最后，在海鸥觅食阶段使用 Levy 飞行策略[13]，使个体位置变化更加灵活多样，提高算法的多样性与跳出局部最优的能力。本文使用 23 个基准测试函数和图像分割问题检验改进算法的效果，并将实验结果与原算法以及其它优化算法做比较，验证了 ALLSOA 算法的先进性。

## 1　海鸥优化算法

海鸥优化算法是通过模拟海鸥 2 个最重要的特征：迁徙和觅食而提出的算法。海鸥是群居生活动物，在季节更替时，海鸥会进行迁徙活动。迁徙时海鸥会保证自己的位置与其它海鸥的位置不同，避免发生碰撞。海鸥是杂食动物，在觅食时会以螺旋形的运动形态攻击其它候鸟。海鸥的迁徙行为与觅食行为可以用以下数学模型来表示。

### 1.1　迁徙

在该过程中，算法模拟海鸥从当前位置移动到下一个位置。在移动过程中，海鸥会满足 3 个条件：避免碰撞，最佳位置方向，靠近最佳位置。

为了避免碰撞，算法通过添加变量 $A$ 来计算海鸥的新位置。对此可表示：

$$C_s(t) = A * P_s(t) \tag{1}$$

$$A = f_c - (t * (f_c / Max_{iteration})) \tag{2}$$

其中，$C_s(t)$ 表示不与其它海鸥发生碰撞的新位置；$P_s(t)$ 表示海鸥当前的位置；$t$ 表示当前迭代数；$f_c$ 表示控制 $A$ 的频率，取值为 2；$Max_{iteration}$ 表示最大迭代次数。

满足避免碰撞的条件之后，海鸥会向最佳位置方向运动。研究推出的数学公式可写为：

$$M_s(t) = B * (P_{bs}(t) - P_s(t)) \tag{3}$$

$$B = 2 * A^2 * r_d \tag{4}$$

其中，$M_s(t)$ 表示最佳位置方向；$P_{bs}(t)$ 表示海鸥的最优位置；$r_d$ 是[0,1]内的随机数。

在海鸥确定最佳位置方向后，就会向该方向移动，到达最佳位置。数学公式具体如下：

$$D_s(t) = | C_s(t) + M_s(t) | \tag{5}$$

其中，$D_s(t)$ 表示海鸥的最佳位置。

### 1.2　觅食

海鸥在觅食过程中会以螺旋形运动状态攻击猎物，因此海鸥在 $x$、$y$、$z$ 平面中的数学模型如下：

$$x = r * \cos(\theta) \tag{6}$$

$$y = r * \sin(\theta) \tag{7}$$

$$z = r * \theta \tag{8}$$

$$r = u * e^{\theta v} \tag{9}$$

其中，$r$ 表示螺旋半径；$[0,2\pi]$ 是 $[0,2\pi]$ 内的随机数；$u$ 为常数。

因此，海鸥觅食行为的数学描述为：

$$P_s(t) = D_s(t-1) * x * y * z + P_{bs}(t-1) \tag{10}$$

其中，$P_s(t)$ 表示海鸥当前攻击的位置；$D_s(t-1)$ 表示海鸥上一时刻的位置；$P_{bs}(t-1)$ 表示海鸥上一时刻的最优位置。

## 2　融合自适应权重与 Levy 飞行的拉丁超立方体海鸥优化算法

### 2.1　拉丁超立方体初始化

海鸥算法的初始化方法是在规定界限内随机生成海鸥的位置，这种初始化方法很容易导致海鸥个体在空间中分布不均，进而影响算法的寻优精度。为解决该问题，本文采用拉丁超立方体抽样的方法初始化种群。

拉丁超立方体抽样的关键是将输入概率分布进行分层，在每一层中抽取一个样本。这样可以保证海鸥初始位置的分布更加均匀，避免早熟。拉丁超立方体抽样的步骤如下：

**Step 1**　确定抽样规模 $H$，即海鸥种群规模。

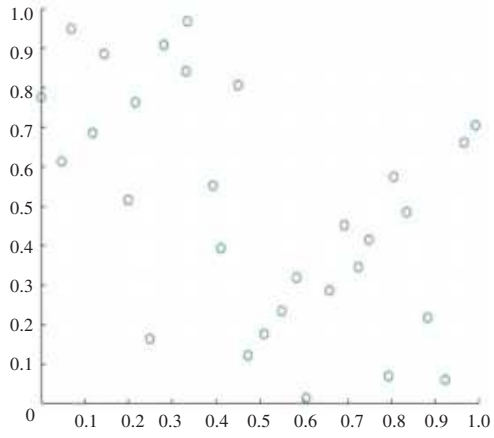
**Step 2**　将每个海鸥位置变量 $d^i$ 的定义域区间 $[d_l^i, d_u^i]$ 划分为 $H$ 个相等的小区间，即：

$$d_l^i = d_0^i < d_1^i < \cdots < d_H^i = d_u^i$$

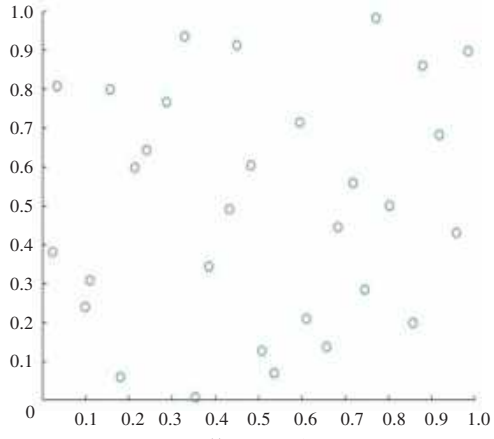
**Step 3**　生成 $H \times n$ 的矩阵 $\boldsymbol{A}$，其中每列均为数列 1,2,…,$H$ 的随机排序。

**Step 4**　$\boldsymbol{A}$ 的每行对应一个超立方体，每个超立方体产生一个样本。

初始化位置分布如图 1 所示。由图 1(a)可知，随机初始化位置在上半部比较密集，下半部比较稀疏；图 1(b)中位置分布则比较均匀。因此可以得出，使用超立方体初始化种群相较于随机初始化种群位置分布更为均匀，更有利于算法寻优。



(a) 随机初始化位置分布图



(b) 超立方体初始化位置分布图

**图 1 初始化位置分布图**

**Fig. 1 Map of initializing the location**

## 2.2 自适应权重因子

在标准的海鸥算法中,海鸥的位置更新公式主要是由海鸥当前位置、最佳方向与随机参数组成。随机参数的大小对算法的搜索能力、寻优能力都有着较大的影响。因为当该参数越大时,海鸥位置更新的步长越大,算法对全局搜索的能力越强;当这个参数越小时,海鸥位置更新的步长越小,算法对局部的搜索能力越强。但是海鸥算法中该参数是随机生成的,这将导致该算法在计算海鸥个体位置时盲目性较高,无法很好地分配算法在初期与末期的搜索能力。

针对该问题,本文在海鸥算法的群体行为中加入自适应权重因子。在更新最佳位置方向时,使用自适应权重因子代替海鸥算法中的随机数。自适应权重因子的数学表达式为:

$$\omega = \sin\left(\frac{\pi \times t}{2 \times Max_{iteration}} + \pi\right) + 1 \tag{11}$$

在海鸥算法的位置更新中引入自适应权重因子的个体位置更新公式为:

$$B = 2 * A^2 * \omega \tag{12}$$

$$M_s(t) = B * (P_{bs}(t) - P_s(t)) \tag{13}$$

算法迭代开始时,自适应权重因子最大,海鸥个体位置更新步长最大,代表算法此时的全局搜索能力最强。此后,自适应权重因子随着时间的增大而减小,海鸥个体位置更新步长逐渐减小,代表算法的局部搜索能力逐渐加强。

## 2.3 Levy 飞行

Levy 飞行模拟自然界中动物觅食的游走过程,并且服从莱维分布,是一种短距离搜索与长距离行走相间的飞行方式。这种飞行机制由法国数学家莱维提出,可以用来缓解海鸥算法早熟的问题以及增加算法的多样性。Levy 飞行的数学表达式为:

$$Le(\beta) \approx \alpha - 1 - \varphi \tag{14}$$

$$\alpha = \frac{D}{|G|/\varphi} \tag{15}$$

$$\upsilon = \left\{\frac{\Gamma(1+\varphi)}{\varphi\Gamma((1+\varphi)/2)} \frac{\sin(\pi\varphi/2)}{2(1+\varphi)/2}\right\}^{\frac{2}{\varphi}} \tag{16}$$

其中, $0 < \varphi \leqslant 2$; $D$、$G \sim N(0,\upsilon)$; $\Gamma(x)$ 是 Gamma 函数; $\alpha$ 表示步长; $\varphi = 2/3$。

使用 Levy 飞行后的海鸥位置更新部分的公式更新为:

$$P_s(t) = \begin{cases} D_s(t-1) * x * y * z * Le(\beta) + P_{bs}(t-1) & t/Max_{iteration} < rand \\ D_s(t-1) * x * y * z + P_{bs}(t-1) & t/Max_{iteration} > rand \end{cases} \tag{17}$$

其中, $rand$ 为[0,1] 之间的随机数。

改进后的位置更新公式,可以使海鸥的位置变化更加灵活,不仅增强算法的寻优能力,还减少算法出现局部最优的现象。

## 2.4 ALLSOA 算法步骤

结合上述改进方法,本文研发的 ALLSOA 算法伪代码的设计表述见如下。

begin

设置初始参数:种群规模 $N$, 超参数 $f_c$ ,最大迭代次数 $Max_{iteration}$

获取测试函数 $F$ 的边界与维度信息

拉丁超立方体初始化种群,计算种群每个搜索代理的适应度

while $t < Max_{iteration}$ do

for $i = 1$ to $N$ do

根据式(1)计算不与其它海鸥碰撞的位置

根据式(13)计算海鸥最佳位置方向
根据式(5)计算海鸥的迁徙位置
根据式(17)计算海鸥的攻击位置
根据式(5)更新海鸥的迁徙位置
end for
end while
end

### 2.5　时间复杂度计算

算法复杂度是算法运算速度的一种体现，也可以反映出算法的效率。在改进算法时算法的时间复杂度如果变大，将会增加算法的运行时间，降低算法效率。在此，本文比较 SOA 算法与 ALLSOA 算法的时间复杂度。

SOA 算法的时间复杂度的组成部分是：随机初始化参数 $O(1)$，计算适应度 $O(N)$，迭代计算最优值 $O(NT)$，则 $SOA$ 算法的时间复杂度为：

$$O(1) + O(N) + O(NT) = O(NT) \tag{18}$$

其中，$N$ 为种群大小，$T$ 为迭代次数。

ALLSOA 算法的时间复杂度的组成部分是：拉丁超立方体初始化参数 $O(NT)$，计算适应度 $O(N)$，迭代计算最优值 $O(NT)$，自适应权重因子 $O(NT)$，Levy 飞行 $O(NT)$，故 ALLSOA 的时间复杂度如式(19)所示：

$$O(NT) + O(N) + O(NT) + O(NT) = O(NT) \tag{19}$$

其中，$N$ 为种群大小，$T$ 为迭代次数。

综上，本文改进后的算法 ALLSOA 时间复杂度与原算法相比未见增加，故 ALLSOA 算法的改进部分并未影响到算法的计算效率。

## 3　图像分割

本文使用不同的智能算法对 K-means 聚类算法[14]进行优化，并使用各算法的最优解完成图像分割任务。

K-means 聚类算法的主要思想是将给定的样本集，按照样本个体之间的距离大小将样本划分为规定的簇数，令簇内的点尽量连接紧密，簇间距离尽量远。根据聚类中心分簇的度量方式为平方差：

$$a^{(i)} = \min_j \| x^{(i)} - y_j \|^2 \tag{20}$$

其中，$x^{(i)}$ 为样本点；$a^{(i)}$ 是与 $x^{(i)}$ 最相似的类别；$y_j$ 为聚类中心。

使用智能算法优化 K-means 聚类算法进行图像分割的步骤为：

**Step 1**　加载图像，将图像像素点转化为样本数据。

**Step 2**　初始化簇心。

**Step 3**　使用智能算法优化 K-means 聚类算法，更新簇心坐标，计算样本数据与簇心的距离，取距离最小簇心的类作为该样本的类别。

**Step 4**　计算更新后的簇心坐标与当前簇心坐标的差距，若大于规定阈值，重复 Step3，反之结束。

## 4　仿真实验与结果分析

### 4.1　实验环境与参数设置

本文实验运行环境为：64 位 Windows10 操作系统，Intel Corei5-7300HQ 的处理器，Matlab R2020b 的仿真软件。

为验证 ALLSOA 算法的性能与图像分割效果，分别进行了 2 种实验。实验 1 使用 23 个基准函数检验 ALLSOA 的算法性能并与其它算法做比较；实验 2 使用图像分割案例验证 ALLSOA 的先进性，并与其它算法做比较。23 个基准函数见表 1，前 7 个函数为单峰值函数，其余为多峰值函数。

### 4.2　算法性能对比实验

为了检验本文算法的性能，本文选择了不同的智能优化算法：鲸鱼优化算法(WOA)、樽海鞘优化算法(SSA)、标准海鸥优化算法(SOA)以及改进算法(ALLSOA)进行对比实验。为了提高实验的准确性，对每个智能优化算法进行 30 次独立实验后，取实验结果的平均值与标准差进行对比。本次实验设置参数为：种群规模 $N = 30$，最大迭代次数 $T = 1\,000$。实验结果见表 2，加粗字体表示 4 种算法求解的最优解。

由表 2 可知，ALLSOA 求解函数 $f_1$、$f_2$、$f_7$、$f_8$、$f_{15}$、$f_{19}$ 的最优值时，比其它优化算法差一点。但是对于函数 $f_8$，ALLSOA 求解的最优值精度虽差，但是标准差更小，算法更加稳定。对于函数 $f_{15}$ 和 $f_{19}$，ALLSOA 求解的最优值与其它算法相差甚微，只是标准差稍大。

对于单峰值函数 $f_3 \sim f_6$，ALLSOA 算法的寻优能力明显高于 WOA、SSA 与 SOA 算法，收敛精度更高。对于多峰值函数 $f_9 \sim f_{14}$，虽然只有函数 $f_9$ 和 $f_{11}$ 被 ALLSOA 算法找到了最优值，但是 ALLSOA 算法求解得到值均为 4 个算法中的最优解，且明显优于 SOA 算法求解值。例如 $f_{12}$ 函数，ALLSOA 算法的解比 SOA 算法的解相差 6 个数量级。对于函数 $f_{18}$，虽然 4 个算法均找到了最优值，但是 ALLSOA 算法的标准差最小。对于函数 $f_{20} \sim f_{23}$，ALLSOA 在寻优精度与标准差均小于其它 3 种算法，与 SOA 算法相比有着大幅提升。

**表 1 基准函数**

**Tab. 1 Benchmark functions**

| 函数 | 维度 | 范围 | 最优值 |
|---|---|---|---|
| $f_1(x) = \sum_{i=1}^{n} x_i^2$ | 30 | $[-100,100]$ | 0 |
| $f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$ | 10 | $[-10,10]$ | 0 |
| $f_3(x) = \sum_{i=1}^{n} (\sum_{j=1}^{i} x_j^2)$ | 10 | $[-100,100]$ | 0 |
| $f_4(x) = \max_i \{\lvert x_i \rvert, 1 \leqslant i \leqslant n\}$ | 10 | $[-100,100]$ | 0 |
| $f_5(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$ | 10 | $[-30,30]$ | 0 |
| $f_6(x) = \sum_{i=1}^{n} (x_i + 0.5)^2$ | 10 | $[-100,100]$ | 0 |
| $f_7(x) = \sum_{i=1}^{n} i x_i^4 + rand(0,1)$ | 10 | $[-1.28,1.28]$ | 0 |
| $f_8(x) = \sum_{i=1}^{n} [x_i^2 \sin(\sqrt{\lvert x_i \rvert})]$ | 10 | $[-500,500]$ | $-418.983 \times n$ |
| $f_9(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$ | 10 | $[-5.12,5.12]$ | 0 |
| $f_{10}(x) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i}^{n} \cos(2\pi x_i)) + 20 + e$ | 10 | $[-32,32]$ | 0 |
| $f_{11}(x) = 1 + \frac{1}{4\,000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}})$ | 10 | $[-600,600]$ | 0 |
| $f_{12}(x) = \frac{\pi}{n}\{10\sin(\pi y_1) + \sum_{i=1}^{n-1} (y_i - 1)^2 [1 - 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\} + \sum_{i=1}^{n} u(x_i,10,100,4)$ | 10 | $[-50,50]$ | 0 |
| $f_{13}(x) = 0.1\sin^2(3\pi x_1) + \sum_{i=1}^{n-1} (x_i - 1)^2 [1 + 10\sin^2(3\pi x_{i+1} + 1)] + (x_n - 1)^2 \times [1 + \sin^2(2\pi x_n)] + \sum_{i=1}^{n} u(x_i,5,100,4)$ | 10 | $[-50,50]$ | 0 |
| $f_{14}(x) = (\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6})^{-1}$ | 2 | $[-65,65]$ | 1 |
| $f_{15}(x) = \sum_{j=1}^{11} [a_i - \frac{x_i(b_i^2 + b_i x_2)}{b_j^2 + b_j x_3 + x_4}]$ | 4 | $[-5,5]$ | 0.000 3 |
| $f_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_1^4$ | 2 | $[-5,5]$ | $-1.031\,6$ |
| $f_{17}(x) = (x^2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6)^2 + 10(1 - \frac{1}{8\pi})\cos x_1 + 10$ | 2 | $[-5,5]$ | 0.398 |
| $f_{18}(x) = [1 + (x_1 + x_2 + 1)^2 \times (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1 x_2 + 3x_2^2)] \times [30 + (2x_1 - 3x_2)^2 \times (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1 x_2 + 27x_2^2)]$ | 2 | $[-2,2]$ | 3 |
| $f_{19}(x) = -\sum_{i=1}^{4} c_i \exp(\sum_{j=1}^{3} a_{ij}(x_i - p_{ij})^2)$ | 3 | $[0,1]$ | $-3.86$ |
| $f_{20}(x) = -\sum_{i=1}^{4} c_i \exp(\sum_{j=1}^{6} a_{ij}(x_i - p_{ij})^2)$ | 6 | $[0,1]$ | $-3.32$ |
| $f_{21}(x) = -\sum_{i=1}^{5} [x - a_i](x - a_i)^{\mathrm{T}} + c_i]^{-1}$ | 4 | $[0,10]$ | $-10.153\,2$ |
| $f_{22}(x) = -\sum_{i=1}^{7} [x - a_i](x - a_i)^{\mathrm{T}} + c_i]^{-1}$ | 4 | $[0,10]$ | $-10.402\,8$ |
| $f_{23}(x) = -\sum_{i=1}^{5} [x - a_i](x - a_i)^{\mathrm{T}} + c_i]^{-1}$ | 4 | $[0,10]$ | $-105\,364$ |

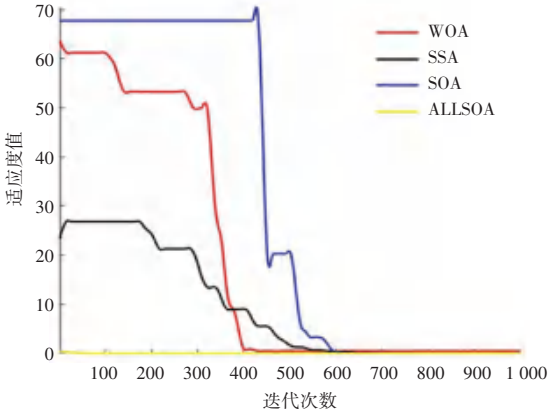
**表 2　测试结果**

**Tab. 2　Test results**

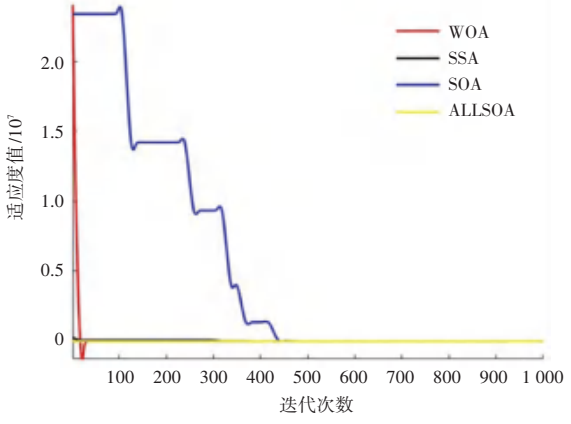
| | | WOA | SSA | SOA | LSOA |
|---|---|---|---|---|---|
| $f_1$ | *Mean* | **2.020 5e−150** | 1.302 4e−08 | 1.592 8e−11 | 1.291 6e−17 |
| | *Std* | **1.057 0e−149** | 2.766 1e−09 | 4.283 7e−11 | 3.410 9e−17 |
| $f_2$ | *Mean* | **9.980 2e−107** | 1.420 8e−04 | 1.108 6e−49 | 1.146 9e−39 |
| | *Std* | **4.817 3e−106** | 7.184 7e−04 | 2.745 6e−49 | 2.633 2e−39 |
| $f_3$ | *Mean* | 5.276 30 | 1.625 0e−09 | 3.660 8e−23 | **2.036 8e−29** |
| | *Std* | 16.156 40 | 5.558 5e−10 | 1.828 1e−22 | **6.761 3e−29** |
| $f_4$ | *Mean* | 2.114 50 | 1.454 3e−05 | 6.289 1e−21 | **9.652 3e−26** |
| | *Std* | 9.508 10 | 3.657 5e−06 | 2.316 9e−20 | **1.528 0e−25** |
| $f_5$ | *Mean* | 6.142 00 | 147.251 3 | 7.279 40 | **3.574 60** |
| | *Std* | 0.308 32 | 304.870 3 | 0.254 22 | **0.599 16** |
| $f_6$ | *Mean* | 5.687 8e−05 | 5.934 0e−10 | 2.419 8e−02 | **7.046 7e−19** |
| | *Std* | 4.015 3e−05 | 2.311 5e−10 | 7.231 7e−02 | **4.991 8e−19** |
| $f_7$ | *Mean* | **9.444 0e−04** | 6.599 7e−03 | 1.993 0e−03 | 1.705 0e−03 |
| | *Std* | **9.197 5e−04** | 4.649 9e−03 | 1.445 8e−03 | 1.140 6e−03 |
| $f_8$ | *Mean* | **−3 552.873 50** | −2 747.651 5 | −2 377.240 4 | −2 836.170 10 |
| | *Std* | **620.436 90** | 297.338 0 | 133.906 60 | 204.129 80 |
| $f_9$ | *Mean* | **0** | 14.393 70 | **0** | **0** |
| | *Std* | **0** | 6.494 10 | **0** | **0** |
| $f_{10}$ | *Mean* | 4.204 0e−15 | 0.779 59 | 1.480 3e−15 | **1.326 6e−15** |
| | *Std* | 2.072 3e−15 | 0.949 93 | 1.346 7e−15 | **1.225 1e−15** |
| $f_{11}$ | *Mean* | 4.463 0e−02 | 0.231 47 | 0 | **0** |
| | *Std* | 0.105 73 | 0.119 88 | 0 | **0** |
| $f_{12}$ | *Mean* | 7.556 4e−02 | 0.432 75 | 8.694 4e−03 | **3.618 7e−09** |
| | *Std* | 0.409 20 | 0.841 11 | 1.232 5e−02 | **2.941 5e−09** |
| $f_{13}$ | *Mean* | 7.525 4e−03 | 0.002 197 5 | 0.387 49 | **2.264 6e−05** |
| | *Std* | 1.293 1e−02 | 0.004 470 1 | 5.725 8e−02 | **3.848 4e−05** |
| $f_{14}$ | *Mean* | 2.535 70 | 1.998 0 | 2.093 60 | **1.067 10** |
| | *Std* | 3.011 90 | 2.656 2e−16 | 1.557 90 | **1.613 0e−17** |
| $f_{15}$ | *Mean* | **6.485 3e−04** | 1.474 0e−03 | 1.186 9e−03 | 3.074 9e−03 |
| | *Std* | **4.474 7e−04** | 3.576 1e−03 | 1.118 4e−04 | 5.498 8e−04 |
| $f_{16}$ | *Mean* | −1.031 60 | −1.031 60 | −1.031 60 | **−1.031 60** |
| | *Std* | 5.049 8e−11 | 8.512 1e−15 | 6.172 2e−08 | **6.251 2e−13** |
| $f_{17}$ | *Mean* | 0.397 89 | 0.397 89 | 0.397 89 | **0.397 89** |
| | *Std* | 7.699 8e−07 | 1.693 8e−16 | 2.295 5e−06 | **1.680 5e−11** |
| $f_{18}$ | *Mean* | 3.000 00 | 3.000 00 | 3.000 00 | **3.000 00** |
| | *Std* | 3.154 8e−05 | 6.044 1e−14 | 9.342 8e−06 | **1.833 1e−16** |
| $f_{19}$ | *Mean* | −3.859 80 | **−3.862 80** | −3.854 90 | −3.862 50 |
| | *Std* | 2.563 8e−03 | **5.698 1e−14** | 2.189 1e−06 | 1.437 6e−03 |
| $f_{20}$ | *Mean* | −3.258 80 | −3.241 10 | −2.183 50 | **−3.384 50** |
| | *Std* | 8.678 2e−02 | 5.828 3e−02 | 0.694 33 | **3.191 2e−02** |
| $f_{21}$ | *Mean* | −9.048 40 | −8.301 60 | −0.915 73 | **−10.052 20** |
| | *Std* | 2.274 00 | 2.723 70 | 0.425 32 | **2.370 9e−09** |
| $f_{22}$ | *Mean* | −8.973 40 | −9.364 10 | −0.959 78 | **−10.087 70** |
| | *Std* | 2.702 00 | 2.411 10 | 0.522 01 | **1.647 4e−09** |
| $f_{23}$ | *Mean* | −9.090 60 | −9.606 40 | −1.032 80 | **−10.128 50** |
| | *Std* | 2.724 60 | 2.443 80 | 0.362 89 | **2.907 7e−09** |

综上,实验结果表明 ALLSOA 算法相比较其它算法有着较好的寻优能力以及收敛精度,但是在某些函数上的表现并不比其它算法要更好,这也表示海鸥算法还有进一步改进的空间。

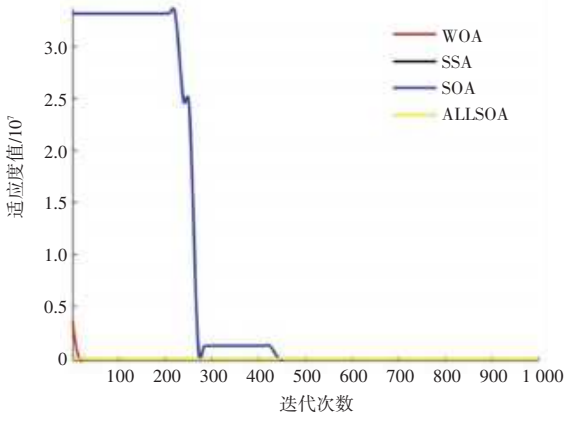
图 2 给出了 ALLSOA 与其它 3 种优化算法在求解基准函数最优值时,随着迭代次数的增长,其适应值的变化曲线。从图 2 中可以看出,ALLSOA 算法在寻优精度与收敛速度上均优于其它算法。



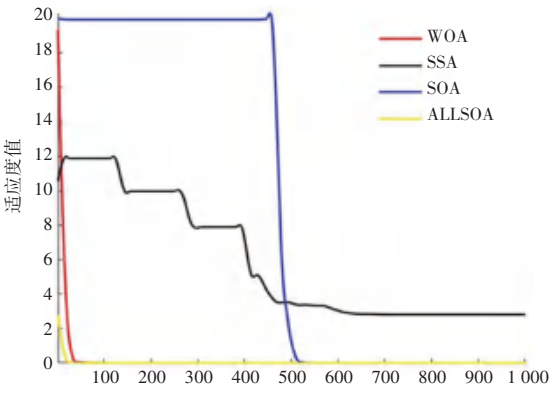

(a) $f_4$ 收敛曲线




(b) $f_5$ 收敛曲线




(c) $f_{12}$ 收敛曲线




(d) $f_{13}$ 收敛曲线

**图 2 算法收敛曲线图**

**Fig. 2 Convergence curve of the algorithms**

### 4.3 图像分割对比实验

将 ALLSOA 算法应用到图像分割中,并与其它 2 种算法做对比,实验参数设置为:种群规模 $N$ = 30,最大迭代次数 $T$ = 50,簇心数 $Z$ = 3。

原图与分割后的图像如图 3 所示。从图 3 中可以明显看出,ALLSOA 算法的分割效果最好,图片中人物与景物的纹理最清晰,最能反映现实景象。

(a) 原图 (b) SOA 分割图

(c) WOA 分割图 (d) ALLSOA 分割图

**图 3 原图与分割图**

**Fig. 3 Original image and split images**

为了进一步验证 ALLSOA 算法在图像分割中的性能,本文绘制 3 种算法在分割过程中的收敛曲线如图 4 所示。

从图 4 中可以看出,ALLSOA 算法的寻优能力与精度均高于其它算法,在迭代第 10 次左右时接近收敛,且寻优精度最高。另外,3 种算法的平均运行时间、最佳适应度与初始聚类中心见表 3。

由表 3 得出,ALLSOA 适应度值最低,虽然比 WOA 慢 0.06 s,但是 ALLSOA 解出的最佳适应度比 WOA 低 159.387,且聚类中心中没有零值,聚类效果更好。

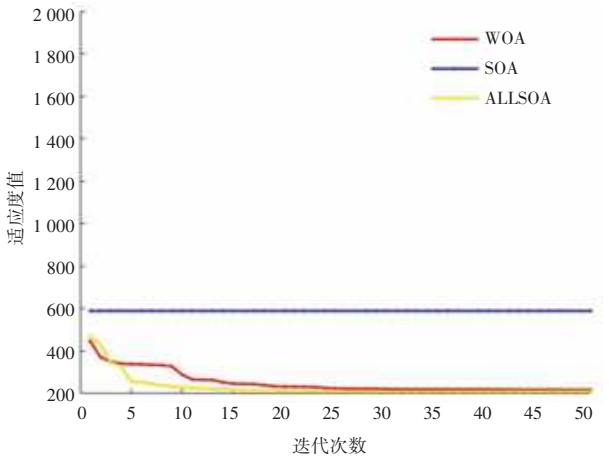



**图 4　WOA、SOA 与 ALLSOA 收敛曲线**

**Fig. 4　Convergence curves of WOA, SOA and ALLSOA**

**表 3　算法运行结果**

**Tab. 3　Running results of the algorithms**

| 算法名称 | 运行时间/s | 最佳适应度 | 初始聚类中心 |
|---|---|---|---|
| WOA | 4.073 941 | 364.594 5 | 0.238 8, 0.000 0, 0.000 0;<br>0.723 6, 0.710 4, 0.699 3;<br>0.262 5, 0.248 1, 0.239 2 |
| SOA | 4.225 572 | 791.373 7 | 0.022 2, 0.011 3, 0.000 0;<br>0.638 5, 0.557 5, 0.559 1;<br>0.178 7, 0.000 0, 0.000 0 |
| ALLSOA | 4.130 199 | 205.207 5 | 0.172 1, 0.165 6, 0.202 1;<br>0.468 0, 0.463 5, 0.467 2;<br>0.778 1, 0.777 1, 0.775 9 |

综合前文研究可知，通过基准函数测试与图像分割测试，均可以证明本文提出的 ALLSOA 算法的性能对比其它算法有着较好的提升，具有更好的收敛速度和寻优精度。

## 5　结束语

本文针对海鸥优化算法(SOA)的各项不足，提出了融合自适应权重与 Levy 飞行的拉丁超立方体海鸥优化算法(ALLSOA)。首先，在初始化时利用拉丁超立方体抽样方法让海鸥分布更加均匀，使海鸥更加容易找到最优目标；其次，在海鸥位置更新时引入自适应权重因子，更好地控制算法的全局寻优与局部搜索能力；最后，在海鸥的觅食阶段使用 Levy飞行策略，增加海鸥位置的灵动性，提升算法跳出局部最优的能力。通过 23 个基准函数和图像分割的仿真实验结果表明，本文提出的算法具有更好的稳定性、寻优能力与收敛精度。

未来的研究方向是进一步提升海鸥算法的收敛精度，并应用于更多的工程问题当中。

## 参考文献

[1] MIRJALILI S. Dragonfly algorithm: A new meta-heuristic optimization technique for solving single - objective, discrete, and multi - objective problems [J]. Neural Computing and Applications, 2016, 27(4):1053-1073.

[2] ARORA S, SINGH S. Butterfly optimization algorithm: a novel approach for global optimization [J]. Soft Computing, 2019, 23 (3): 715-734.

[3] MIRJALILI S, MIRJALILI S M, LEWIS A. Grey wolf optimizer [J].Knowledge-Based Systems,2014,69(3):46-61.

[4] ESKANDAR H, SADOLLAH A, BAHREININEJAD A, et al. Water cycle algorithm - A novel metaheuristic optimization method for solving constrained engineering optimization problems [J]. Computers & Structures, 2012, 110 - 111:151-166.

[5] MIRJALILI S, GANDOMI A H, MIRJALILI S Z, et al. Salp swarm algorithm: a bio-inspired optimizer for engineering design problems[J]. Advances in Engineering Software,2017,114: 163-191.

[6] MIRJALILI S, LEWIS A. The whale optimization algorithm [J]. Advances in Engineering Software,2016,95:51-67.

[7] LI Shimin, CHEN Huiling, WANG Mingjing, et al. Slime mould algorithm: a newmethod for stochastic optimization [J]. Future Generation Computer Systems,2020, 111(1): 300-323.

[8] DHIMAN G, KUMAR V. Seagull optimization algorithm: theory and its applications for large-scale industrial engineering problems [J]. Knowledge-Based Systems, 2019, 165(2019): 169-196.

[9] 许乐，莫愿斌，卢彦越. 具有记忆功能的海鸥优化算法求解方程组[J]. 计算机工程与设计,2021,42(12):3428-3437.

[10] 张冰洁，何庆，戴松利，等. 多方向螺旋搜索的混沌海鸥优化算法[J/OL]. 小型微型计算机系统:1-10[2022-01-21].http://kns.cnki.net/kcms/ detail/21.1106.TP.20211213. 1750.028. html.

[11] 毛清华，王迎港. 融合改进 Logistics 混沌和正弦余弦算子的自适应 $t$ 分布海鸥算法[J/OL]. 小型微型计算机系统:1-9[2021-11-09]. http://kns. cnki. net/kcms/detail/ 21. 1106. TP. 20211019. 1549.006.html.

[12] 王宁，何庆. 融合黄金正弦与 sigmoid 连续化的海鸥优化算法[J]. 计算机应用研究, 2022,39(01):157-162,169.

[13] 梁建明，何庆. 莱维飞行和反馈策略的自适应被囊群算法[J/OL]. 小型微型计算机系统:1-9[2021-11-09]. https://kns.cnki.net/kcms/detail/21.1106.tp.

[14] ARTHUR D, VASSILVITSKII S. k-means++: The advantages of careful seeding [C]//Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms. New Orleans: ACM, 2007: 1027-1035.