文章编号: 1671-6833(2021)03-0053-06

基于混沌初始化和高斯变异的飞蛾火焰优化算法

刘 倩[1], 冯艳红[1,2], 陈嶷瑛[1]

(1.河北地质大学 信息工程学院,河北 石家庄 050031; 2.河北地质大学 河北省智能传感物联网技术工程研究中心,河北 石家庄 050031)

摘 要: 针对飞蛾火焰优化算法(moth-flame optimization algorithm, MFO)在求解最优化问题时存在寻优精度低、易陷入局部最优等问题,提出一种基于混沌初始化和高斯变异的改进飞蛾火焰优化算法。首先,采用立方混沌映射对飞蛾种群进行初始化操作,使飞蛾更均匀地分布于搜索空间;其次,应用高斯变异对种群中少数较差个体进行扰动以增强算法跳出局部最优的能力;最后,通过阿基米德曲线扩大搜索范围,提高算法对未知领域的探索能力。在 CEC14 测试函数及 21 个可扩展 Benchmark 函数上进行了一系列实验,与标准飞蛾火焰优化算法、遗传算法、人工蜂群算法、粒子群算法、差分进化算法、花授粉算法和蝴蝶优化算法进行比较,结果表明,该算法能明显提高解的精度和算法的收敛速度。

关键词: 混沌初始化; 高斯变异; 阿基米德曲线; 飞蛾火焰优化算法; 群体智能

中图分类号: TP301 **文献标志码:** A **doi:** 10.13705/j.issn.1671-6833.2021.03.009

0 引言

飞蛾火焰优化算法(moth-flame optimization algorithm, MFO)[1]是一种新颖的群体智能算法。该算法由飞蛾和火焰 2 部分构成,通过横向定位导航机制来解决探索和利用之间的平衡问题。MFO 具有参数简单、容易实现且鲁棒性好等优点,因此,自提出以来,便受到国内外学者的广泛关注,使其在诸多领域得到应用[2-5]。当前,国内外学者对其改进研究主要分为对飞蛾飞行方式的优化和对进化机制的优化。

尽管已有文献对 MFO 算法的改进在一定程度上提高了算法的性能,但是这些改进措施在解决高维多峰复杂问题时效果不尽如人意。

基于此,本文提出一种基于混沌初始化和高斯变异的飞蛾火焰优化算法(moth-flame optimization algorithm based on chaotic initialization and Gaussian mutation, CGMFO)。CGMFO 采用立方混沌映射初始化种群以增强飞蛾勘探未知空间的能力,从而提高求解精度;采用高斯变异对部分较差个体进行局部扰动以提高算法的收敛速度;采用阿基米德曲线飞行机制以增强种群的多样性,并抑制算法的早熟收敛,进而实现全局最优化。CGMFO 对 CEC14 经典测试函数[6]和 21 个可扩展 Benchmark 函数[7]进行求解,实验结果验证了该算法具有良好的收敛能力和竞争实力。

1 飞蛾火焰优化算法的基本原理

在 MFO 算法中,用式(1)表示飞蛾种群规模为 n,维数为 d 的飞蛾所处空间位置,且用式(2)矩阵存储飞蛾个体的适应度值。火焰是算法的另一个核心,其在空间的位置矩阵与飞蛾的空间矩阵类似,用式(3)表示,使用式(4)矩阵存储相应火焰的适应度值。

$$\boldsymbol{M}(n,d)=\begin{bmatrix} m_{11} & m_{12} & \cdots & m_{1d} \\ m_{21} & m_{22} & \cdots & m_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ m_{n1} & m_{n2} & \cdots & m_{nd} \end{bmatrix}; \quad (1)$$

$$\boldsymbol{OM}=[om_1, om_2, \cdots, om_n]^{\mathrm{T}}; \quad (2)$$

$$\boldsymbol{F}(n,d)=\begin{bmatrix} f_{11} & f_{12} & \cdots & f_{1d} \\ f_{21} & f_{22} & \cdots & f_{2d} \\ \vdots & \vdots & \vdots & \vdots \\ f_{n1} & f_{n2} & \cdots & f_{nd} \end{bmatrix}; \quad (3)$$

收稿日期: 2020-10-21; 修订日期: 2020-12-11

基金项目: 国家自然科学基金资助项目(61806069)

通信作者: 冯艳红(1978—),女,河北卢龙人,河北地质大学副教授,硕士,主要从事人工智能、进化算法的研究,E-mail: qinfyh@163.com。

$$\boldsymbol{OF} = [of_1 \quad of_2 \quad \cdots \quad of_n]^{\mathrm{T}}。 \tag{4}$$

火焰减少过程如下式所示:

$$flameNo = round\left(N - t \times \frac{N-1}{T}\right)。 \tag{5}$$

式中: t 和 T 分别为当前迭代次数和最大迭代次数; N 为最大火焰数量。

$$\boldsymbol{D}_i = |\boldsymbol{F}_j - \boldsymbol{M}_i|。 \tag{6}$$

式中: $\boldsymbol{D}_i$ 为飞蛾 $\boldsymbol{M}_i$ 到火焰 $\boldsymbol{F}_j$ 的距离。

假设飞蛾 $\boldsymbol{M}_i$ 会朝着距离自己最近的火焰 $\boldsymbol{F}_j$ 移动,移动的路径选取了对数螺线曲线,并将该曲线作为飞蛾的主要更新机制,算法的对数螺旋曲线定义如下:

$$S(\boldsymbol{M}_i,\boldsymbol{F}_j) = \boldsymbol{D}_i \cdot e^{bt} \cdot \cos 2\pi t + \boldsymbol{F}_j。 \tag{7}$$

式中: $S(\boldsymbol{M}_i,\boldsymbol{F}_j)$ 为更新后的飞蛾位置; $\boldsymbol{D}_i$ 表示第 i 个飞蛾与第 j 个火焰的距离; b 为与螺旋形状相关的常量; t 为随机数,取值区间为 $[-1,1]$; $e^{bt} \cdot \cos 2\pi t$ 为对数螺旋曲线表达式。

总结前文所述,飞蛾火焰优化算法的一般步骤如下:

步骤 1 初始化算法参数: 飞蛾数量 n、维数 d、最大迭代次数 T 等,随机初始化种群;

步骤 2 计算种群中飞蛾的适应度值,并按适应度值升序排序;

步骤 3 利用式(5)更新火焰数量;

步骤 4 利用式(6)计算飞蛾到火焰的距离;

步骤 5 利用式(7)更新飞蛾位置;

步骤 6 若满足给定的迭代次数,算法结束,获得最优解; 否则,返回步骤 2。

2 CGMFO 原理

2.1 混沌初始化

在原始 MFO 中,飞蛾的位置通过随机初始化的方式产生,会造成飞蛾的位置分布不均匀,进而降低求解精度。而基于混沌理论的混沌序列具有伪随机性和边界性等特点,故许多学者提出了嵌入混沌序列的多种算法[8-9]。在诸多混沌映射中,立方映射的性能较优[10],因此,本文采用立方混沌映射对 MFO 进行初始化:

$$\begin{cases} y(n+1) = 4y(n)^3 - 3y(n), \\ -1 \leqslant y(n) \leqslant 1, \ n = 0,1,\cdots,n。 \end{cases} \tag{8}$$

利用立方混沌映射初始化飞蛾种群的 3 个步骤如下:

步骤 1 按照式(8)随机产生 d 维空间中的 n 只飞蛾,即 $\boldsymbol{Y} = (y_1,y_2,y_3,\cdots,y_d)$, $y_i \in [-1,1]$, $i = 1,2,\cdots,d$;

步骤 2 将每只飞蛾的每一维迭代 n 次,从而产生 n 只飞蛾;

步骤 3 在所有的飞蛾迭代完成后,按照式(9)映射到解空间中。

$$x_{id} = L_d + (1 + y_{id}) \times \frac{U_d - L_d}{2}。 \tag{9}$$

式中: U_d、L_d 为搜索空间的上、下界; y_{id} 是利用式(8)产生的第 i 只飞蛾的第 d 维坐标; x_{id} 是第 i 只飞蛾在搜索空间第 d 维的坐标。

本文将基于立方混沌映射初始化的飞蛾火焰优化算法记为 CMFO。

2.2 高斯变异机制

观察 MFO 算法中解的更新公式(式(5)和式(7)),可以发现每只飞蛾空间矢量位置的更新与离其最近的火焰空间位置有直接的关系。而此火焰空间位置仅靠适应度值来确定,这就导致对当前全局最优飞蛾的有效利用能力不强。

因此,在对飞蛾按适应度值进行排序后,选取适应度值最差的 $n \times \omega$ 个个体,并对其应用高斯变异 $Gaussian(\mu,\sigma^2)$,其中, n 是飞蛾种群规模; ω 是变异的比例; μ 表示均值; σ^2 表示方差。根据经验,本文 ω 取值为 1/6,便于控制和缩小更新范围,并适度地提高算法的种群多样性。改进后的飞蛾产生公式描述如下:

$$\boldsymbol{M}_{\mathrm{new}}^t = \boldsymbol{M}_j^{*\,t} + Gaussian(\mu,\sigma^2)。 \tag{10}$$

式中: $\boldsymbol{M}_j^{*\,t}$ 和 $\boldsymbol{M}_{\mathrm{new}}^t$ 分别表示变异前和变异后的飞蛾。

本文将基于 CMFO,对种群每一代中少数较差个体用高斯变异进行微小扰动的飞蛾火焰优化算法记为 GCMFO。

2.3 基于阿基米德曲线的飞行机制

由于 MFO 中飞蛾的飞行机制会影响算法的性能,与对数螺旋曲线相比,阿基米德螺旋曲线使飞蛾扩大了搜索范围,且阿基米德螺旋曲线在诸多优化领域得以应用[11-12]。为了增强 MFO 算法的种群多样性,加快算法的收敛速度,本文将原 MFO 算法中的对数螺旋曲线替换为阿基米德螺旋曲线。其平面笛卡尔坐标方程式为

$$\begin{cases} x = (\alpha + \beta\theta)\cos\theta; \\ y = (\alpha + \beta\theta)\sin\theta。 \end{cases} \tag{11}$$

其中,当 $\theta = 0$ 时, α 为起点到极坐标原点的距离, β 为螺旋线每增加单位角度随之对应增加的数值。当 $\theta > 0$ 时, α 相当于旋转螺线,而 β 则控制相邻两条曲线之间的距离。

则飞蛾位置的移动如下式所示:

$$S(\boldsymbol{M}_i,\boldsymbol{F}_j)=\boldsymbol{D}_i\cdot e^{bt}\cdot(\alpha\times\gamma)\cdot\sin 2\pi\beta(\gamma-\alpha)+\boldsymbol{F}_j。\tag{12}$$

其中,$b=1$,$\beta=10$,α、γ 定义如下:

$$\alpha=t\times\frac{-1}{T}-1;\tag{13}$$

$$\gamma=(\alpha-1)\times rand+1。\tag{14}$$

式中: $rand$ 为随机数,取值区间为 [-1, 1]。

本文将基于 GCMFO,应用阿基米德螺旋曲线的飞行方式的飞蛾火焰优化算法记为 CGMFO。

2.4　CGMFO 算法流程

根据 2.1~2.3 节描述的算法改进思想,提出的 CGMFO 算法步骤如下:

步骤 1　按式(8)和式(9)初始化种群;

步骤 2　计算种群中飞蛾的适应度值,按适应度值升序排序;

步骤 3　选择排序后适应度较差的 1/6 飞蛾,按式(10)进行高斯变异,然后求解飞蛾和火焰的适应度值,选择其中最好飞蛾个体的位置并将其保存为火焰适应度值矩阵;

步骤 4　利用式(5)更新火焰数量;

步骤 5　利用式(6)计算飞蛾到火焰的距离;

步骤 6　利用式(13)和式(14)初始化螺旋曲线的相关参数,按式(12)更新飞蛾位置;

步骤 7　若满足给定的迭代次数,算法结束,获得最优解; 否则,返回步骤 2。

3　数值实验及分析

3.1　环境与参数

为了比较不同改进策略对 MFO 的影响程度,本文进行了 2 组实验。第 1 组实验对 CGMFO、GCMFO、CMFO 与 MFO 这 4 种算法,选用 CEC14 测试集[6]中 $f_1\sim f_6$ 进行仿真实验比较。第 2 组实验对 CGMFO、MFO、GA[13]、ABC[14]、PSO[15]、DE[16]、FPA[17]和 BOA[18]这 8 种算法,利用 21 个可扩展 Benchmark 函数[7]进行仿真实验比较。

仿真硬件环境为 LENOVO Intel i7-8750H CPU 2.21 GHz, 8.00 GB RAM,操作系统为 Windows 10,利用 MATLAB R2018b 编程实现。

所有算法均采用相同的种群规模 $n=30$,最大迭代次数 $T=1\,000$,第 1 组实验中,维数 $d=100$,第 2 组实验中,$d=10$,重复实验次数 $RepeatCount=30$。各算法的其余参数设置如表 1 所示。

表 1　8 种算法的参数设置

Table 1　Parameter settings for 8 algorithms

算法	参数名称	参数含义	取值
CGMFO	μ	均值	0
	b	螺旋形状常量	1
	β	螺旋形状常量	10
MFO	b	螺旋形状常量	1
GA	a	选择概率	0.3
	b	交叉概率	0.3
	c	变异概率	0.1
ABC	$limit$	限度	15
	fn	食物的数量	15
PSO	$c1$	学习因子	0.9
	$c2$	学习因子	0.3
DE	f	缩放因子	0.5
	cr	交叉概率	0.3
FPA	p	转换概率	0.8
BOA	p	转换概率	0.8
	a	强度增加指数	0.1

3.2　算法性能评价指标

本文的 2 组实验都采用优化均值误差($Average$)和标准方差($Std.Dev$)评价算法的优化能力,其中优化均值误差计算如下:

$$Average=f(x)-f(x^*)。\tag{15}$$

式中: x 表示算法得到的解; x^* 表示每个测试函数的理论最优解。$Average$ 越小,则解的质量越好。

第 2 组实验中增加了 $Best$ 最优解和 $Worst$ 最差解的比较,这 2 个值越小,说明解的精度越高。

同时,为了比较算法在统计意义上的差异性,利用 Wilcoxon 秩和检验(显著水平 $\alpha=0.05$)分别对 21 个函数的实验结果进行统计分析。

3.3　实验结果分析

3.3.1　第 1 组实验结果分析

由于篇幅有限,表 2 给出了 4 种算法对 CEC14 中每个函数进行 30 次独立实验的部分计算结果,由表 2 的计算结果可以得出如下结论:

(1) 3 种改进算法对于 CEC14 中的测试集函数 $f_1\sim f_6$,无论是优化均值误差还是标准方差均优于原始的 MFO。

(2) CMFO 只使用立方混沌映射产生混沌序列,保证了飞蛾个体初始化位置在整个搜索空间的均匀分布,优化均值误差和标准方差在绝大多数测试函数上均有提高。

(3) GCMFO 的改进效果优于 CMFO,这是由于 GCMFO 不仅采用了立方混沌映射初始化,而且通过高斯变异对每代中少数较差的个体进行扰动,帮助算法跳出局部极值区域,提高了算法收敛速度。

(4) CGMFO 的改进效果明显优于其他 2 种

表2 4种算法对于CEC14测试函数的实验结果比较

Table 2 Comparison of the experimental results of the 4 algorithms for the CEC14 test function

算法	f_1		f_2		f_3	
	Average	*Std.Dev*	*Average*	*Std.Dev*	*Average*	*Std.Dev*
MFO	1.52E+06	1.24E+06	1.40E+06	9.17E+06	1.46E+06	9.13E+05
CMFO	3.63E+05	3.00E+05	3.20E+05	2.50E+05	3.84E+05	3.39E+05
GCMFO	5.01E+03	1.92E+01	5.01E+03	2.63E+01	5.01E+03	2.25E+01
CGMFO	**9.80E+01**	**2.04E+00**	**9.74E+01**	**2.21E+00**	**9.84E+01**	**1.32E+00**
算法	f_4		f_5		f_6	
	Average	*Std.Dev*	*Average*	*Std.Dev*	*Average*	*Std.Dev*
MFO	1.23E+06	9.12E+05	1.39E+06	1.15E+06	1.37E+06	1.00E+06
CMFO	3.36E+05	2.85E+05	2.74E+05	1.85E+05	2.00E+05	1.50E+05
GCMFO	5.01E+03	2.15E+01	5.00E+03	2.55E+01	5.00E+03	2.15E+01
CGMFO	**9.82E+01**	**1.70E+00**	**9.83E+01**	**1.96E+00**	**9.80E+01**	**2.05E+00**

改进算法,多个函数的误差均值和标准方差都接近理论最优。

3.3.2 第2组实验结果分析

表3为8种算法对21个可扩展Benchmark函数进行30次独立实验的计算结果,其中*Best*为最优解,*Worst*为最差解,*Average*和*Std.Dev*定义与第1组实验相同。表3中的符号†、≈、▪分别表示CGMFO算法的实验结果优于、相当于和劣于对比算法,其中结果加粗表示最优。为了进一步比较CGMFO与其他7种算法的收敛趋势,图1给出了8种算法求解Benchmark函数f_1~f_3、f_9~f_{11}各30次的平均进化曲线。经多次实验发现,当设定每种算法的迭代次数均为$T=160$时,进化曲线的效果明显,故这里将T设置为160。

由以上实验结果可知:

(1)由表3可以看出,对于21个Benchmark函数,针对4种评价指标,CGMFO在求解几乎全部问题上均优于其他7种比较算法。其中14个函数求解结果达到了理论最优(f1~f8、f11~f13、f18、f20、f21)。其余未达到理论最优值的7个函数,与其他算法相比,CGMFO也最接近理论最优。

(2)由图1的平均迭代进化曲线可以看出,在21个可扩展Benchmark函数中,CGMFO对应

表3 8种算法对于测试函数f_1~f_{21}的部分实验结果比较

Table 3 Comparisons of some experimental results of 8 algorithms for test functions f_1 to f_{21}

算法	f_1				f_4				f_7			
	Best	*Worst*	*Average*	*Std.Dev*	*Best*	*Worst*	*Average*	*Std.Dev*	*Best*	*Worst*	*Average*	*Std.Dev*
CGMFO	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
MFO	4.91E-36	3.63E-30	3.24E-31	8.06E-31†	5.84E-67	1.00E+00	1.34E+01	3.27E+01†	0	0	0	0
GA	3.58E-06	4.32E-05	1.79E-05	1.00E-05†	7.51E-10	6.44E-06	8.00E-07	1.52E-06†	0	0	0	0
ABC	9.52E-17	5.85E-14	1.97E-14	3.36E-14†	0	0	0	0	9.12E-02	4.73E-01	2.50E-01	1.99E-01†
PSO	2.18E-18	1.12E-05	3.75E-07	2.05E-06†	4.96E-16	5.85E-08	3.97E-08	9.49E-08†	0	1.00E+00	3.33E-02	1.83E-01≈
DE	6.97E-43	8.89E-41	1.48E-41	1.72E-41†	5.99E-92	2.66E-85	2.33E-86	6.66E-86†	0	0	0	0
FPA	2.31E-02	2.46E-01	8.22E-02	5.64E-02†	4.64E-09	8.41E-06	1.11E-06	1.77E-06†	0	0	0	0
BOA	2.16E-14	3.90E-14	2.91E-14	4.69E-15†	5.93E-15	2.69E-14	2.07E-14	4.66E-15†	0	0	0	0
算法	f_{10}				f_{16}				f_{19}			
	Best	*Worst*	*Average*	*Std.Dev*	*Best*	*Worst*	*Average*	*Std.Dev*	*Best*	*Worst*	*Average*	*Std.Dev*
CGMFO	**4.35E+00**	**6.77E+00**	**6.00E+00**	**5.83E-04**	2.30E-17	5.76E-09	2.74E-10	1.09E-09	**3.13E-34**	**1.10E-32**	**7.33E-33**	**2.79E-52**
MFO	3.43E-02	9.96E+02	1.12E+02	3.00E+02†	4.71E-32	7.75E+01	5.08E+00	1.43E+01†	1.35E-31	1.00E+01	3.74E-01	1.82E+00†
GA	3.41E-02	7.39E+00	4.68E+00	2.57E+00†	8.69E-01	1.09E+00	1.06E+00	5.61E-02†	3.25E-05	5.54E-03	1.21E-03	1.29E-03†
ABC	7.40E-03	2.47E-02	1.82E-02	9.42E-03†	4.79E-13	3.98E-10	1.34E-10	2.89E-09†	7.83E-01	7.54E+00	7.21E-03	3.91E-12†
PSO	6.70E-01	1.01E+02	1.38E+01	2.32E+01†	3.20E-06	2.36E-00	2.02E-01	4.94E-01†	3.69E-08	6.62E-01	3.38E-02	1.22E-01†
DE	1.43E+00	7.02E+00	4.59E+00	1.56E+00†	**4.71E-32**	**4.71E-32**	**4.71E-32**	**1.67E-47**▪	1.35E-31	1.35E-30	1.34E-31	8.20E-43†
FPA	4.29E+00	1.84E+01	9.09E+00	2.30E+00†	1.96E-01	1.81E+00	7.50E-01	4.40E-01†	2.16E-02	2.56E-01	1.11E-01	5.76E-02†
BOA	8.97E+00	9.00E+00	1.00E+01	6.43E-03†	1.08E+00	2.10E+01	5.21E+00	5.37E+00†	5.78E+00	1.64E+01	1.15E+01	2.48E+00†

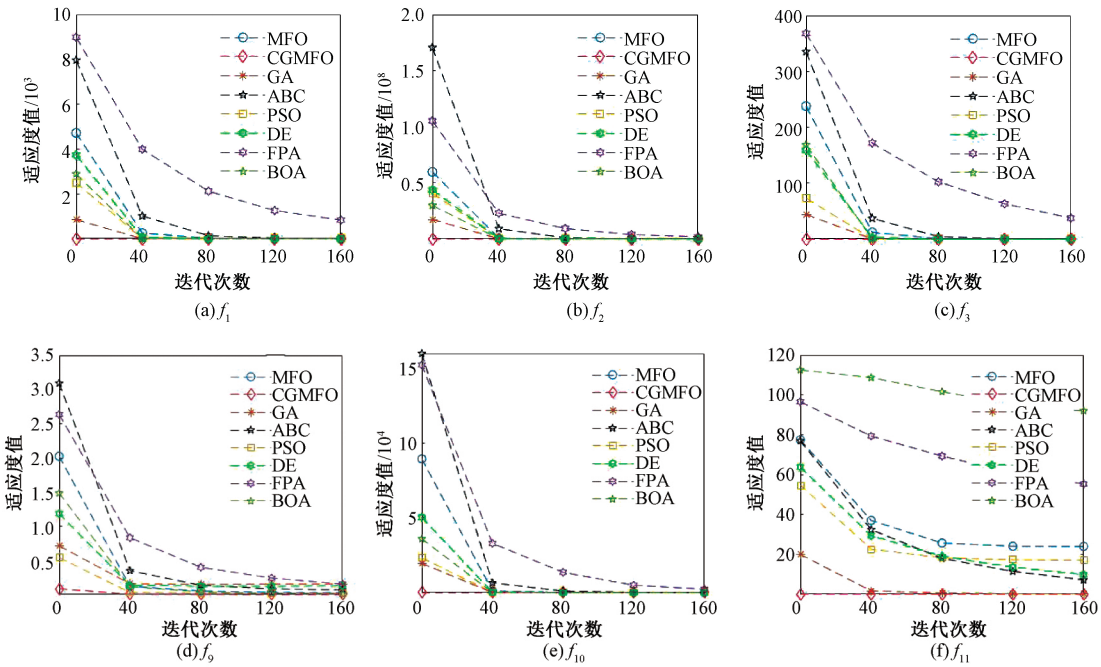

图 1　部分平均迭代进化曲线

Figure 1　Some average iterative evolution curves

的目标函数下降曲线比其他 7 种算法的下降曲线具有更快的下降速度，并很快达到了最优值。这再次表明，CGMFO 在函数全局优化方面具有更好的有效性，收敛速度更快。

4　结论

针对 MFO 求解精度不高及易陷入局部最优等缺陷，本文从 3 个方面对算法进行改进: 立方混沌映射初始化，有效地防止飞蛾的位置分布不均匀; 高斯变异对较差个体的扰动，减少了算法陷入局部最优的可能; 阿基米德曲线飞行机制的应用使种群的多样性有所增加。从实验结果看出，CGMFO 在搜索性能上均优于 CMFO 和 GCMFO，且 CGMFO 在求解精度和收敛速度上与文中的其他进化算法相比，均表现优秀。由于 MFO 提出时间不长，故算法的参数分析和在工程问题中的应用将是下一步的研究课题。

参考文献:

[1] MIRJALILI S. Moth-flame optimization algorithm: a novel nature-inspired heuristic paradigm [J]. Knowledge-based systems，2015，89: 228-249.

[2] 李佳华，马连博，王兴伟，等.基于多目标蜂群进化优化的微电网能量调度方法 [J].郑州大学学报(工学版)，2018，39(6)：50-58.

[3] 程适，王锐，伍国华，等.群体智能优化算法 [J].郑州大学学报(工学版)，2018，39(6)：1-2.

[4] LIU D，ZHANG G D，LI H，et al. Projection pursuit evaluation model of a regional surface water environment based on an Ameliorative Moth-Flame Optimization algorithm [J]. Ecological indicators，2019，107: 105674.

[5] 李志明，莫愿斌.基于 Lévy 飞行的飞蛾扑火优化算法 [J].计算机工程与设计，2017，38(3)：807-813.

[6] GARG V，DEEP K.Performance of Laplacian biogeography-based optimization algorithm on CEC 2014 continuous optimization benchmarks and camera calibration problem [J]. Swarm and evolutionary computation，2016，27: 132-144.

[7] KIRAN M S，HAKLI H，GUNDUZ M，et al. Artificial bee colony algorithm with variable search strategy for continuous optimization [J]. Information sciences，2015，300: 140-157.

[8] 冯艳红，刘建芹，贺毅朝.基于混沌理论的动态种群萤火虫算法 [J].计算机应用，2013，33(3)：796-799，805.

[9] YU H L，ZHAO N N，WANG P J，et al. Chaos-enhanced synchronized bat optimizer [J]. Applied mathematical modelling，2020，77: 1201-1215.

[10] CHEN G，WU X D，ZHU X Q，et al.Efficient string matching with wildcards and length constraints [J].Knowledge and information systems，2006，10(4)：399-419.

[11] CHEN Y D，WEI T H，GONG T X.Research on optimal layout of cutter-head system of rock tunnel-boring machine based on Archimedes spiral theory [J]. Advances in mechanical engineering，2018，10(2)：151-160.

[12] ZHANG B F, HU C B, TANG B J, et al. Chiral plasmonic lens with Archimedes-spiral distributed nanoslits [J]. Journal of nanophotonics, 2019, 13(2): 026008.
[13] KOZA J R. Genetic programming as a means for programming computers by natural selection [J]. Statistics and computing, 1994, 4(2): 87-112.
[14] KARABOGA D, AKAY B. A comparative study of artificial bee colony algorithm [J]. Applied mathematics and computation, 2009, 214(1): 108-132.
[15] CHUANG L Y, TSAI S W, YANG C H. Chaotic catfish particle swarm optimization for solving global numerical optimization problems [J]. Applied mathematics and computation, 2011, 217(16): 6900-6916.
[16] SIVANANAITHAPERUMAL S, AMALI S M J, BASKAR S, et al. Constrained self-adaptive differential evolution based design of robust optimal fixed structure controller [J]. Engineering applications of artificial intelligence, 2011, 24(6): 1084-1093.
[17] NABIL E. A modified flower pollination algorithm for global optimization [J]. Expert systems with applications, 2016, 57: 192-203.
[18] ARORA S, SINGH S. Butterfly optimization algorithm: a novel approach for global optimization [J]. Soft computing, 2019, 23(3): 715-734.

Moth-flame Optimization Algorithm Based on Chaotic Initialization and Gaussian Mutation

LIU Qian[1], FENG Yanhong[1,2], CHEN Yiying[1]

(1. School of Information Engineering, Hebei GEO University, Shijiazhuang 050031, China; 2. Intelligent Sensor Network Engineering Research Center of Hebei Province, Hebei GEO University, Shijiazhuang 050031, China)

Abstract: Moth-flame optimization algorithm (MFO) has some drawbacks in solving optimization problems, such as low precision and high possibility of being trapped in local optimum. A modified MFO algorithm based on chaotic initialization and Gaussian mutation is proposed. Firstly, the cube chaotic map is used to initialize the moth population, which makes the moth more evenly distributed in the search space. Then, Gaussian mutation is adopted to disturb a few poor individuals to enhance the ability of escaping the local optimum. Finally, Archimedes curve is introduced to expand the search scope and strength the exploration ability in the unknown field. A series of experiments are carried out on CEC14 test function set and 21 extensible Benchmark functions. Compared with standard moth-flame optimization algorithm, genetic algorithm, artificial bee colony algorithm, particle swarm algorithm, differential evolution algorithm, flower pollination algorithm, and butterfly optimization algorithm, the results demonstrate that the proposed algorithm is strengthened in obtaining solutions with better quality and convergence.

Key words: chaotic initialization; Gaussian mutation; Archimedes curve; moth-flame optimization algorithm; swarm intelligence