文章编号: 1001-0920(2018)03-0557-08 DOI: 10.13195/j.kzyjc.2016.1622

基于自适应步长的改进蝙蝠算法

吕石磊[1,2], 黄永霖[1], 陈海强[1], 李 震[1,2†], 王卫星[1,2]

(1. 华南农业大学 电子工程学院，广州 510642；2. 广东省农情信息监测工程技术研究中心，广州 510642)

摘 要: 针对基本蝙蝠算法存在容易过早陷入局部最优以及求解精度低的问题,提出一种改进的蝙蝠算法(SABA),加入自适应的步长控制机制和变异机制. 通过对12个单峰/多峰函数的测试表明,与粒子群算法、蝙蝠算法相比,SABA算法能够有效解决算法陷入局部最优的问题,从而具有较高的求解精度.

关键词: 蝙蝠算法；自适应；步长控制机制；变异机制

中图分类号: TP301.6 **文献标志码:** A

Improved bat algorithm using self-adaptive step

LV Shi-lei[1,2], *HUANG Yong-lin*[1], *CHEN Hai-qiang*[1], *LI Zhen*[1,2†], *WANG Wei-xing*[1,2]

(1. College of Electronic Engineering，South China Agricultural University，Guangzhou 510642，China；2. Guangdong Engineering Research Center for Monitoring Agricultural Information，Guangzhou 510642，China)

Abstract: For the problems of low solution precision by the initial bat algorithm and falling into local optimum easily, an improved self-adaptive bat algorithm(SABA) is proposed, which combines the mechanisms of step-control and variation. Experiments are conducted on a set of 12 benchmark functions, and the results show that the proposed SABA has better performance than the particle swarm optimization(PSO) algorithm and initial bat algorithm(BA) in terms of accuracy and convergence speed.

Keywords: bat algorithm；self-adaptive；step-controlled mechanism；variation mechanism

0 引 言

群体智能优化算法(SIOA)是近年来迅速发展的人工智能学科领域的重要组成部分. 学者们通过研究动物群体的行为,利用群体中个体之间的信息交流与合作机制,在去中心化和自组织模型的前提下,提出了许多区别于传统优化策略的智能仿生算法,包括粒子群优化算法(PSO)[1]、人工蜂群算法(ABC)[2]、细菌觅食算法(BFO)[3]、蝙蝠算法(BA)[4]、布谷鸟搜索算法(CS)[5]、鸡群算法(CSO)等[6],并将其应用于解决实际工程的数据处理与优化问题[7-10].

BA算法是受蝙蝠回声定位行为的启发而提出的一种新型搜索优化算法[4]. 近年来,BA算法的研究涉及到数学理论、智能优化、工程等领域. 李枝勇等[11]用数学方法证明了基本BA算法的速度和位置更新方式具有较大的局限性. Fister等[12]通过引入差分演化策略,提出一种混合BA算法,仿真测试表明算法性能有较明显的提升. 肖辉辉等[13]将差分算法中的变异、交叉、选择机制融入基本BA算法中,提出一种基于差分进化算法的改进BA算法. Chen等[14]采用融合策略对基本BA算法作出了改进. 刘长平等[15]利用混沌运动的随机性和初值敏感性提高基本BA算法的效率,提出一种基于混沌序列的改进BA算法. Yang等[16]将多目标的BA算法和帕累托前沿应用于电力系统的无源滤波器设计,以减少谐波效应对电力系统的影响. Wang等[17]将差分进化策略引入基本BA算法,并将其应用于无人机航路规划领域.

针对基本BA算法在求解高维度问题时容易陷入局部最优并且存在求解精度低的问题[12],本文通过引入自适应的步长控制机制和变异机制,提出一种改进的蝙蝠算法(SABA).

1 基本蝙蝠算法

BA算法利用蝙蝠在觅食时所发出脉冲的频率f、响度A、脉冲发射率R的变化建立模型. 在算法迭

收稿日期: 2016-12-21；**修回日期:** 2017-03-08.

基金项目: 国家自然科学基金项目(61601189)；现代农业产业技术体系建设专项资金(CARS-26)；广东省科技计划项目(2015A020209161, 2016A020210088, 2016A020210093)；广州市科技计划项目(201605030013).

作者简介: 吕石磊(1984−),男,讲师,博士,从事智能计算、优化算法应用等研究；李震(1981−),男,教授,博士,从事机电一体化技术应用等研究.

†通讯作者. E-mail: lizhen@scau.edu.cn

代过程中,蝙蝠个体i的参数更新可描述为

$$f_i = f_{\min} + (f_{\max} - f_{\min})\beta, \tag{1}$$

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - x_*)f_i, \tag{2}$$

$$x_i^t = x_i^{t-1} + v_i^t. \tag{3}$$

其中:t为当前迭代次数;$\beta \in [0,1]$为一个随机向量,服从均匀分布;x_*为当前的全局最优解.

2) 在当前全局最优解附近进行局部搜索,即

$$x_i^t = x_* + \varepsilon A^t. \tag{4}$$

其中:$\varepsilon \in [-1,1]$为一个随机数,A^t为当前所有蝙蝠的平均响度.

3) 更新脉冲响度A和脉冲发射频率R,即

$$A_i^{t+1} = \alpha A_i^t, \tag{5}$$

$$R_i^{t+1} = R_{\max}[1 - \exp(-\gamma t)]. \tag{6}$$

其中:α和γ为常量,一般取值为0.9,随着迭代次数的增加,可得到

$$A_i^t \to 0, R_i^t \to R_{\max}, t \to \infty. \tag{7}$$

2 自适应的改进蝙蝠算法

通过对蝙蝠行为特性的分析,可得知蝙蝠在靠近猎物的过程中,需要不断提升脉冲发射频率R并减小脉冲响度A,以便在随时掌握猎物动静的同时防止被察觉,因此脉冲响度A和脉冲发射频率R的变化对BA算法的寻优过程有重要影响. 在算法前期,脉冲响度A数值较大,而此时应当进行更多的全局搜索,所以将其与算法的全局搜索能力相结合,使全局搜索概率得到调整;在算法后期,脉冲发射频率R数值较大,而此时应进行更多的局部搜索,故将R变化与局部搜索调整相结合. 基于此,本文在BA算法中引入自适应步长控制机制和变异机制.

2.1 步长控制机制

1) 针对式(2)中蝙蝠个体i速度更新方式,本文在速度项添加惯性权重因子,有

$$v_i^t = \omega v_i^{t-1} + f_1 r_1(h_* - x_i^{t-1}) + f_2 r_2(x_* - x_i^{t-1}). \tag{8}$$

其中:惯性权重ω可以影响蝙蝠的局部和全局寻优能力, 本文采用线性递减权重的方法,前期ω较大使算法具有较强的全局搜索能力,后期ω较小以保证算法具有较强的局部搜索能力; h_*为当前蝙蝠个体i最优解; x_*为当前蝙蝠群体的最优解; f_1和f_2为脉冲频率; r_1和r_2为(0,1)范围内的均匀随机数.

2) 式(8)中脉冲频率f_1和f_2分别代表蝙蝠个体和蝙蝠群体对速度项的影响. 在算法前期,f_1较大能够增加蝙蝠群体多样性,从而增强算法全局搜索能力;在算法后期,f_2较大能够保证算法的收敛性. 本文采用自适应异步变化的脉冲频率,提出f_1和f_2随蝙蝠个体适应值和迭代次数变化的自适应学习方法. f_1和f_2的更新公式为

$$f_1 = \alpha(1 - \mathrm{e}^{-|F_{\text{avg}} - F_{\text{best}}|}) + \gamma\Big(1 - \frac{t}{t_{\max}}\Big) + f_{\min}, \tag{9}$$

$$C_w = f_1 + f_2. \tag{10}$$

其中:F_{avg}为蝙蝠种群所有个体的平均适应值;F_{best}为当前蝙蝠种群中最优个体;f_1的更新取决于当前蝙蝠种群的平均适应值和当前的迭代次数,两者所占的权重分别为常量α和γ;$f_{\min}$为常数,用于设定和控制f_1最小值;f_2的更新过程与f_1相反,为简化运算,本文通过设定二者之和C_w来计算;考虑到式(8)中蝙蝠个体和蝙蝠群体对速度项的影响(f_1、f_2)与式(2)中蝙蝠群体对速度项的影响(f_i)应具有一致性,并且r_1、r_2均为(0,1)范围内的均匀随机数,故C_w的取值范围设为[0,4]. 测试结果表明,$C_w = 3$会取得较好效果. 因此,$f_1 \in [f_{\min}, 3]$, $f_2 \in [0, 3 - f_{\min}]$.

3) 针对式(3)中蝙蝠个体i位置更新方式,本文增加权重系数μ,用于约束蝙蝠个体的迭代步长,即

$$x_i^t = x_i^{t-1} + \mu v_i^t, \tag{11}$$

其中μ的取值范围为(0,1].

4) 针对式(4)中蝙蝠个体i的局部搜索操作,本文提出一种与蝙蝠个体脉冲发射频率R、脉冲响度A和当前迭代次数相结合的局部搜索策略,具体流程描述如下.

首先,设置随机数$\beta \in [0,1]$,若β小于脉冲发射频率R,则该蝙蝠个体将进行局部搜索. 因为R将随着迭代次数增加而增大,所以在算法后期,蝙蝠个体的局部搜索概率也随之增大.

然后,局部搜索策略描述为:当评价指数$k < 0.4$时,蝙蝠个体i位置更新为

$$x_i^t = x_* + Asg(k)\delta; \tag{12}$$

当$k \geqslant 0.4$时,蝙蝠个体i位置更新为

$$x_i^t = x_* + As0.1^{g(k)}\delta. \tag{13}$$

其中:评价指数$k = t/t_{\max}$, $t_{\max}$为最大迭代次数,则$k \in (0,1]$; x_*为当前蝙蝠种群最优解; A为当前蝙蝠种群的平均响度; s为求解问题的可行解域上下边界距离与蝙蝠种群数量的比值,参数s能够将局部搜索步长与求解问题的规模自适应; $\delta \in [-1,1]$为随机向量.

最后,为提高蝙蝠个体的局部搜索效率,本文在

式(12)和(13)中引入分段函数$g(k)$,如下所示:

$$g(k)=\begin{cases}2,\ k\leqslant 0.1;\\ 1.5,\ 0.1<k\leqslant 0.2;\\ 1,\ 0.2<k\leqslant 0.3;\\ 0.5,\ 0.3<k\leqslant 0.4;\\ 1,\ 0.4<k\leqslant 0.6;\\ 3,\ 0.6<k\leqslant 0.7;\\ 5,\ 0.7<k\leqslant 0.8;\\ 7,\ 0.8<k\leqslant 0.9;\\ 9,\ 0.9<k.\end{cases}\tag{14}$$

本文使用$g(k)$的主要目的是在算法前期,较大的局部搜索步长有利于扩展算法搜索域;在算法后期,较小的局部搜索步长有利于提高算法搜索精度;应注意$g(k)$的参数设置受评价指数k和局部搜索策略的影响,也可使用其他参数组合来提高算法性能.

2.2　变异机制

上述步长控制机制能够有效提高算法的全局搜索能力,但不能确保寻找到全局最优解. 而且,在算法后期,f_1减小,全局搜索能力降低,算法仍有概率陷入局部最优. 在基本BA算法中,如式(7)所示,算法末期响度A将降为0,发射频率R将增大至1,从而不能进行变异操作. 本文对响度A和发射频率R作出限制,使得SABA算法在后期也能够有效进行变异操作,从而降低算法陷入局部最优的概率. 脉冲响度A和发射频率R的更新公式为

$$A_i^{t+1}=\frac{f_1}{f_{\max}},\tag{15}$$

$$R_i^{t+1}=\frac{f_2}{f_{\max}},\tag{16}$$

其中$f_{\max}$为脉冲频率的上限值. 因为脉冲频率f_1和f_2的更新取决于当前迭代次数和种群平均适应值,所以响度A和发射频率R能够根据算法寻优过程自适应变化.

通过对蝙蝠行为特性的分析,本文提出与蝙蝠脉冲响度A结合的变异机制,具体流程概述为: 生成随机数$\beta_1\in[0,1]$,若β_1小于A且蝙蝠个体未进行局部搜索,则生成随机数$\beta_2\in[0,1]$;若β_2大于设定常数$\rho\in(0,1)$,则对该蝙蝠个体随机重置.

综上所述,本文所提出的步长控制机制和变异机制是在深入分析蝙蝠行为特性的基础上,充分与脉冲响度A、脉冲发射频率R及算法寻优过程相适应,从而使SABA算法能够在前期有效避免过早陷入局部最优,并且在后期提高算法求解精度.

2.3　算法流程

SABA算法流程如图1所示,具体描述如下.

Step 1: 初始化蝙蝠种群参数.

Step 2: 根据式(9)和(10)更新脉冲频率f,根据式(15)和(16)更新响度A和发射频率R.

Step 3: 根据式(8)更新蝙蝠个体速度,根据式(11)更新蝙蝠个体位置,计算所有个体的适应值.

Step 4: 判断蝙蝠个体是否进行局部搜索操作,若满足条件,则根据式(12)～(14)进行局部搜索,计算适应值,并转至Step6.

Step 5: 判断蝙蝠个体是否进行变异操作,若满足条件,则进行变异,并计算适应值.

Step 6: 更新蝙蝠个体位置参数和全局最优解.

Step 7: 判断是否达到终止条件,如不满足终止条件,则转至Step2.

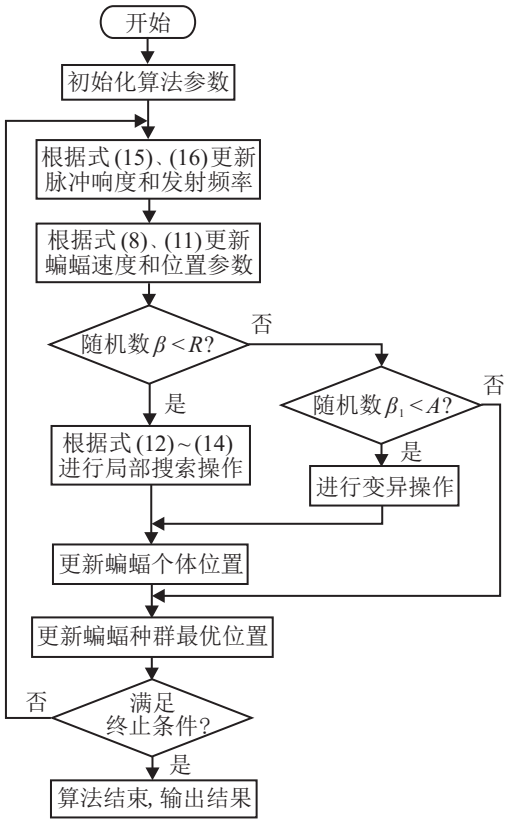

图 1　SABA算法流程

2.4　算法性能分析

根据式(8)和(11),即SABA算法的速度更新公式和位置更新公式,可得出蝙蝠个体i的位置递推公式

$$x_i^t+(\mu f_1r_1+\mu f_2r_2-1-\omega)x_i^{t-1}+\omega x_i^{t-2}=\mu f_1r_1h_*+\mu f_2r_2x_*.\tag{17}$$

其中:t为当前迭代次数,μ为位置更新权重系数,f_1和f_2为蝙蝠脉冲频率,r_1和r_2为(0,1)范围内的均匀随机数,ω为速度惯性权重系数,h_*为蝙蝠个体i最优

解;x_*为当前蝙蝠群体的最优解.

根据式(17),可得出当迭代次数趋于无穷时($t\to\infty$),SABA算法中蝙蝠个体i位置x_i^t的极限

$$\lim_{t\to\infty} x_i^t=\frac{f_1r_1h_*+f_2r_2x_*}{f_1r_1+f_2r_2}. \tag{18}$$

根据式(2)和(3),可得出基本BA算法中蝙蝠个体i位置x_i^t的极限

$$\lim_{t\to\infty} x_i^t=x_*. \tag{19}$$

基于对式(18)和(19)的分析可知,SABA算法中蝙蝠个体i位置的极限值收敛于其个体最优解和蝙蝠群体最优解之间,而基本BA算法中蝙蝠个体i位置的极限值始终收敛于蝙蝠群体最优解. 在蝙蝠群体的寻优过程中,对于所得到的可行解域中某一局部最优解x'_*,若SABA算法中存在蝙蝠个体的极限值不属于该局部最优解x'_*的最佳邻域时,则SABA算法存在一定概率跳出该局部最优,从而具有较好的防止局部收敛能力;与之相比,基本BA算法则易于陷入局部最优.

此外,随着迭代次数的增加,脉冲频率f_1逐渐减小至$f_{\min}$,f_2逐渐逼近C_w. 该设定使得SABA算法前期,蝙蝠个体最优解h_*的权重较大,蝙蝠群体侧重于全局搜索;在算法后期,蝙蝠群体最优解x_*的权重较大,蝙蝠群体侧重于局部搜索,从而有助于提高算法的求解精度.

设蝙蝠群体数量为N,SABA算法的时间复杂度可分析如下:算法初始化阶段,时间复杂度为$O(N\times 1)=O(N)$;算法迭代运行阶段,蝙蝠个体单次位置更新及计算适应度值的时间复杂度为$O(1)$,记变异的蝙蝠个体数量为λN,则蝙蝠群体单次迭代的时间复杂度为$O((1+\lambda)\times N)$. 若设迭代次数为M,则SABA算法的时间复杂度为$O((1+\lambda)\times M\times N)+O(N)$,可简化为$O((1+\lambda)\times M\times N)$.

3 测试结果与分析

3.1 测试函数

本文通过使用12个基准测试函数来分析算法性能,如表1所示. 其中$f_1\sim f_5$为单峰函数,主要用于测试算法的收敛速度和执行效率;$f_6\sim f_{12}$为多峰函数,可用于测试算法的全局搜索能力、跳出局部极值及避免早熟的收敛能力.

3.2 测试环境

操作系统: Microsoft Windows 7 Pro 64-bit, CPU:

表1 测试函数

函数	表达式	可行解域
Sphere	$f_1(x_i)=\sum_{i=1}^{D}x_i^2$	$[-100,100]$
Shifted Sphere	$f_2(x_i)=\sum_{i=1}^{D}(x_i-10)^2-450$	$[-100,100]$
Zakharov	$f_3(x_i)=\sum_{i=1}^{D}x_i^2+\Big(\frac{1}{2}\sum_{i=1}^{D}ix_i\Big)^2+\Big(\frac{1}{2}\sum_{i=1}^{D}ix_i\Big)^4$	$[-10,10]$
Schwefel	$f_4(x_i)=\sum_{i=1}^{D}\lvert x_i\rvert+\prod_{i=1}^{D\lvert x_i\rvert}$	$[-10,10]$
Shifted Schwefel	$f_5(x_i)=\sum_{i=1}^{D}\Big(\sum_{j=1}^{i}(x_j-20)\Big)^2-450$	$[-100,100]$
Shifted Rosenbrock	$f_6(x_i)=\sum_{i=1}^{D-1}(100(x_i^2-x_(i+1))^2+(x_i-1)^2)+390$	$[-100,100]$
Griewank	$f_7(x_i)=\frac{1}{4\,000}\sum_{i=1}^{D}x_i^2-\prod_{i=1}^{D}\cos\Big(\frac{x_i}{\sqrt{i}}\Big)+1$	$[-600,600]$
Ackley	$f_8(x_i)=-20\exp\Big[-\frac{1}{5}\sqrt{\frac{1}{D}x_i^2}\Big]-\exp\Big[\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\Big]+20+e$	$[-32,32]$
Rastrigin	$f_9(x_i)=10D+\sum_{i=1}^{D}[x_i^2-10\cos(2\pi x_i)]$	$[-5.12,5.12]$
Shifted Rastrigin	$f_{10}(x_i)=10D+\sum_{i=1}^{D}[(x_i-1)^2-10\cos(2\pi(x_i-1)]-330$	$[-5,5]$
Penalized1	$f_{11}(x_i)=\frac{\pi}{D}\Big\{10\sin^2(\pi y_1)+\sum_{i=1}^{D-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})]+(y_n-1)^2\Big\}\sum_{i=1}^{D}u(x_i,10,100,4),$ $y_i=1+\frac{x_i+1}{4}$	$[-50,50]$
Penalized2	$f_{12}(x_i)=0.1\Big\{\sin^2(3\pi x_1)+\sum_{i=1}^{D-1}(x_i-1)^2[1+\sin^2(3\pi x_{i+1})](x_D-1)^2\Big\}+\sum_{i=1}^{D}u(x_i,5,100,4)$	$[-50,50]$

Intel®Core™ i5-4200M @ 2.50 GHz, RAM: 4.0 GB, 编程工具: Matlab R2014a.

3.3 测试结果与分析

本文使用的对照算法包括PSO算法[1]、基本BA算法[4]和FSABA算法[14],各算法参数设置与原文献一致, 具体设置如下.

改进蝙蝠算法(SABA): $\alpha = 1, \gamma = 1.5$,最大脉冲频率 $f_{\max} = 2.5$,最小脉冲频率 $f_{\min} = 0.5, C_w = 3$,最大脉冲发射频率为0.7,最小脉冲响度为0.3,惯性权重 $w_{\max} = 0.9, w_{\min} = 0.4, \mu = 0.7, \rho = 0.5$. PSO算法:学习因子 $c_1 = c_2 = 2$,惯性权重 $w = 0.729$. 基本BA算法:脉冲频率范围为 $[-2, 0]$,响度 A 初始值范围为 $(1, 2)$,脉冲发射频率 R 初始值为 $(0, 0.5)$,最大发射频率 $R_{\max} = 0.9$. FSABA算法:脉冲频率范围为 $[0, 2]$,响度初始值 $A_0 = 0.5$,脉冲发射频率初始值 $R_0 = 0.5$.

上述算法的种群规模均为40,最大迭代次数均为500次,独立运行50次. 各算法基于30D、50D和100D测试函数的计算结果,包括求解测试函数的适应值最优解(best)、最差解(worst)、均值(mean)、标准方差(std)和平均运行时间(time/s),如表2∼表4所示.

由表2∼表4的数据分析可知:在寻优精度方面,基本BA算法对单峰测试函数($f_1 \sim f_5$)的求解精度稍差于PSO算法;在多峰测试函数中,基本BA算法的整体性能优于PSO算法;随着测试函数维度的增加,对照算法的求解精度均有较大幅度下降. 与上述多种对照算法相比,本文所提出的SABA算法在单峰函数和多峰函数中均具有较好的求解精度;并且,由不同测试函数在不同维度测试结果的标准方差可知,SABA算法具有较好的鲁棒性.

在算法运行时间方面,随着不同测试函数的维度变化,PSO算法和基本BA算法互有优劣,但总体运行时间均优于SABA算法和FSABA算法. FSABA算法改进了基本BA算法中脉冲发射频率 R 的更新过程,减缓其增大速率,使得算法前期可搜索到更多可行解,同时也增加了算法的运行时间. 与基本BA算法相比,SABA算法在迭代过程中加入了变异机制,当满足随机数 β 大于脉冲发射频率 R,并且随机数 β_1 小于脉冲响度 A 时,SABA算法进行变异操作; 由式(9)和(10)可知,在算法前期,为增加蝙蝠群体多样性,脉冲频率 f_1 较大, f_2 较小;由式(15)和(16)可知,此时脉冲响度 A 较大,脉冲发射频率 R 较小. 因此,SABA算法在前期具有较大概率进行变异操作,一方面,变异机制能够使得算法有效避免过早陷入局部最优;另一方面,该机制也将抑制算法收敛速度,增加算法运行时间.

表 2　各算法求解30D测试函数的结果

测试函数		PSO	BA	FSABA	SABA
f_1	best	4.943 7	0.544 0	2.452 5	1.700 5e-19
	worst	52.775 8	96.519 6	4.335 5	5.169 3e-18
	mean	19.419 1	16.952 6	3.457 7	1.306 1e-18
	std	11.309 1	16.372 0	0.395 6	8.663 5e-19
	time/s	0.475 3	0.526 5	3.935 0	0.624 6
f_2	best	−445.505 1	−444.781 5	−447.845 3	−450.000 0
	worst	−404.620 7	−191.037 3	−445.369 5	−450.000 0
	mean	−426.154 1	−396.948 9	−446.474 4	−450.000 0
	std	9.146 5	44.718 4	0.498 6	1.199 0e-13
	time/s	0.385 3	0.443 1	2.583 6	0.953 8
f_3	best	0.405 8	4.007 1e-05	1.471 8e+05	3.849 2e-21
	worst	1.218 1e+03	4.241 3e+04	1.528 3e+06	3.5516e-20
	mean	79.989 8	849.1908	5.804 7e+05	1.440 4e-20
	std	210.685 1	5.998 0e+03	3.389 7e+05	7.942 1e-21
	time/s	0.312 3	0.449 3	3.062 5	0.493 7
f_4	best	1.744 9	0.118 4	7.030 4	5.398 2e-10
	worst	8.715 9	52.853 7	2.409 5e+05	2.914 5e-04
	mean	4.462 2	9.555 3	4.902 1e+03	1.695 8e-05
	std	1.663 4	11.627 8	3.406 5e+04	5.520 6e-05
	time/s	0.160 4	0.495 8	3.705 2	0.565 3
f_5	best	−318.211 6	3.477 9e+03	−441.782 0	−450.000 0
	worst	6.635 5e+03	8.515 6e+04	−423.744 0	−449.506 8
	mean	326.209 4	3.359 5e+04	−434.138 4	−449.960 9
	std	1.189 0e+03	2.109 6e+04	3.949 3	0.079 1
	time/s	0.700 3	0.445 7	3.305 8	0.946 8
f_6	best	2.381 3e+03	442.522 9	708.016 9	390.000 6
	worst	5.267 0e+04	1.404 2e+05	1.097 8e+04	467.013 4
	mean	1.707 8e+04	1.532 6e+04	2.491 9e+03	408.470 9
	std	1.083 7e+04	2.736 1e+04	2.939 7e+03	15.659 9
	time/s	0.215 3	0.432 9	2.773 2	0.761 2
f_7	best	1.069 5	0.440 5	0.129 1	4.440 9e-16
	worst	1.364 1	1.746 4	0.241 6	0.029 5
	mean	1.169 7	1.166 0	0.191 3	0.003 7
	std	0.076 0	0.191 6	0.029 5	0.007 0
	time/s	0.242 7	0.436 0	2.435 8	0.971 9
f_8	best	3.680 7	0.096 7	18.757 7	1.296 7e-10
	worst	8.941 2	3.117 1	20.567 0	5.085 8e-10
	mean	5.661 3	1.361 7	19.884 7	2.866 2e-10
	std	0.990 5	0.801 2	0.379 1	7.316 6e-11
	time/s	0.239 5	0.258 3	4.522 2	1.126 8
f_9	best	26.353 1	1.068 0	215.859 2	8.881 8e-15
	worst	67.593 3	170.047 9	310.733 3	0.995 0
	mean	42.731 7	59.518 8	257.519 3	0.040 4
	std	10.459 9	48.103 9	23.237 3	0.196 9
	time/s	0.171 5	0.332 2	4.111 7	0.554 4
f_{10}	best	−300.878 7	−329.955 1	−133.938 4	−330.000 0
	worst	−235.899 8	−144.277 7	2.899 4	−329.005 0
	mean	−268.908 9	−287.837 9	−73.286 2	−329.980 1
	std	13.851 9	36.653 4	30.507 1	0.140 7
	time/s	0.189 3	0.349 0	2.762 2	0.393 5
f_{11}	best	1.256 5	8.258 5e-04	13.825 5	2.755 4e-21
	worst	11.226 2	2.519 4	37.085 4	2.767 1e-20
	mean	4.722 6	0.367 8	21.556 0	1.181 0e-20
	std	2.330 4	0.533 9	5.627 7	5.454 9e-21
	time/s	1.452 7	1.804 9	4.507 2	2.508 9
f_{12}	best	4.753 1	0.014 2	0.377 8	6.450 8e-20
	worst	72.367 3	4.670 3	0.784 3	5.231 4e-19
	mean	28.714 4	0.806 7	0.564 6	2.017 1e-19
	std	17.809 4	0.934 9	0.085 8	1.044 5e-19
	time/s	1.231 9	1.404 9	3.378 9	1.449 6

表3 各算法求解50D测试函数的结果

测试函数		PSO	BA	FSABA	SABA
f_1	best	102.299 4	10.240 2	8.379 9	1.383 7e-16
	worst	417.541 4	492.031 9	14.320 5	3.448 8e-12
	mean	204.122 2	74.752 5	11.266 7	1.178 8e-13
	std	60.838 7	88.884 0	1.259 9	5.169 7e-13
	time/s	0.204 6	0.489 7	4.995 9	0.814 7
f_2	best	−345.564 6	−407.638 1	−442.248 4	−450.000 0
	worst	−8.617 9	120.348 0	−435.935 0	−450.000 0
	mean	−231.217 8	−239.868 5	−438.676 3	−450.000 0
	std	77.815 4	143.861 8	1.277 6	1.197 7e-12
	time/s	0.286 7	0.280 3	4.570 3	0.487 1
f_3	best	1.521 6e+04	1.535 5e-04	1.521 1e+08	1.485 5e-18
	worst	1.236 4e+07	1.973 4e+07	1.935 6e+09	3.001 7e-15
	mean	1.106 3e+06	5.311 1e+05	5.211 0e+08	3.890 7e-16
	std	1.953 7e+06	2.931 6e+06	3.561 6e+08	8.313 1e-16
	time/s	0.170 4	0.307 1	5.426 5	0.522 8
f_4	best	8.121 2	0.314 9	18.648 0	1.251 5e-05
	worst	20.832 0	139.461 3	1.388 0e+10	0.154 8
	mean	14.436 9	23.836 9	3.236 3e+08	0.012 4
	std	3.226 1	30.084 6	1.977 5e+09	0.030 1
	time/s	0.166 0	0.205 5	5.897 1	0.620 6
f_5	best	3.185 8e+03	1.855 2e+04	−327.523 6	−448.368 3
	worst	2.082 4e+04	3.932 4e+05	−182.177 5	−344.256 6
	mean	9.616 3e+03	1.573 6e+05	−258.750 9	−425.812 2
	std	3.983 8e+03	1.062 0e+05	39.802 1	20.438 3
	time/s	0.772 5	0.732 3	3.873 3	1.696 0
f_6	best	1.464 1e+05	540.8244	1.901 6e+03	390.000 0
	worst	3.034 3e+06	1.9753e+05	1.139 3e+04	541.107 7
	mean	7.323 2e+05	3.266 4e+04	4.010 6e+03	431.308 7
	std	5.729 5e+05	4.401 3e+04	2.460 2e+03	29.089 7
	time/s	0.196 2	0.229 7	3.246 1	0.613 3
f_7	best	1.789 5	0.833 9	0.300 5	9.370 3e-14
	worst	3.879 3	18.411 9	0.561 6	0.034 4
	mean	2.680 9	2.546 4	0.407 9	0.003 0
	std	0.527 9	2.801 6	0.047 2	0.007 1
	time/s	0.280 7	0.308 6	4.799 7	0.799 4
f_8	best	5.784 3	0.026 1	19.964 0	2.281 9e-09
	worst	10.121 0	5.480 4	20.701 6	1.894 6e-07
	mean	7.824 9	1.846 1	20.367 1	3.777 0e-08
	std	0.875 0	1.184 2	0.162 9	4.556 4e-08
	time/s	0.272 9	0.664	3.423 0	0.769 4
f_9	best	76.528 7	1.079 1	427.827 2	2.060 6 e-13
	worst	157.405 9	272.897 1	571.140 9	2.000 8
	mean	113.815 7	84.094 2	504.503 6	0.242 0
	std	22.188 2	59.207 6	34.551 4	0.621 5
	time/s	0.201 7	0.376 2	3.384 2	0.813 3
f_{10}	best	−248.998 5	−329.387 0	98.217 7	−330.000 0
	worst	−135.687 0	−81.473 4	291.281 4	−328.009 8
	mean	−190.424 2	−263.598 6	192.495 3	−329.838 6
	std	25.324 7	56.899 8	47.855 9	0.464 7
	time/s	0.202 3	0.442 8	3.266 3	0.611 2
f_{11}	best	4.497 1	6.714 2e-04	14.092 2	8.765 8e-16
	worst	30.036 4	7.072 8	38.595 3	5.940 3e-12
	mean	10.627 0	0.464 6	22.662 1	4.046 1e-13
	std	5.008 8	1.040 8	5.093 2	1.062 0e-12
	time/s	1.082 7	1.833 1	7.971 4	4.986 8
f_{12}	best	49.199 5	0.012 0	1.246 3	1.639 1e-14
	worst	751.596 5	18.678 8	2.485 2	2.796 1e-11
	mean	137.573 7	2.802 7	1.855 7	2.688 6e-12
	std	114.499 2	3.570 0	0.236 3	5.062 6e-12
	time/s	0.790 8	2.052 6	5.829 7	2.464 7

表4 各算法求解100D测试函数的结果

测试函数		PSO	BA	FSABA	SABA
f_1	best	1.335 4e+03	37.648 5	41.847 8	3.882 0e-06
	worst	2.856 2e+03	5.228 7e+04	59.664 8	1.375 3e-04
	mean	1.940 7e+03	2.510 1e+03	52.897 1	2.248 8e-05
	std	366.679 3	8.367 4e+03	4.304 2	2.162 8e-05
	time/s	0.535 7	0.527 3	7.999 1	1.147 4
f_2	best	893.429 7	−444.022 8	−404.973 4	−450.000 0
	worst	3.017 2e+03	2.519 8e+04	−388.958 9	−449.999 9
	mean	1.778 2e+03	1.468 7e+03	−397.054 6	−450.000 0
	std	456.633 4	3.696 2e+03	3.816 3	1.878 6e-05
	time/s	0.381 8	0.430 2	6.858 3	0.973 2
f_3	best	9.432 2e+09	0.019 6	1.161 2e+12	3.173 6e-08
	worst	2.407 0e+11	7.242 0e+10	7.813 6e+12	2.190 5e-05
	mean	6.984 8e+10	4.411 7e+09	3.726 7e+12	7.793 4e-07
	std	5.254 2e+10	1.571 0e+10	1.453 5e+12	3.072 0e-06
	time/s	0.533 2	0.460 2	9.199 5	0.899 4
f_4	best	32.773 5	0.771 7	241.674 6	0.004 8
	worst	60.951 4	363.243 8	4.121 3e+26	0.265 8
	mean	18.584 7	44.215 6	1.002 3e+25	0.079 9
	std	6.378 6	57.426 9	5.933 7e+25	0.064 4
	time/s	0.623 9	0.310 1	8.037 5	1.057 9
f_5	best	3.813 8e+04	2.806 9e+05	6.112 3e+03	−191.898 3
	worst	3.477 0e+05	2.376 7e+06	1.173 7e+04	2.723 6e+03
	mean	1.210 9e+05	7.921 4e+05	8.516 7e+03	882.482 8
	std	5.342 4e+04	4.171 6e+05	1.359 1e+03	878.974 9
	time/s	2.298 7	2.173 6	7.388 0	3.564 8
f_6	best	7.844 0e+06	2.596 5e+03	1.427 4e+04	390.126 1
	worst	4.568 2e+07	5.042 4e+05	6.894 8e+04	776.674 3
	mean	2.351 1e+07	1.401 0e+05	2.709 3e+04	509.547 9
	std	7.885 5e+06	1.195 6e+05	1.031 1e+04	96.722 4
	time/s	0.418 6	0.302 2	7.732 2	1.380 8
f_7	best	13.501 8	1.097 2	0.749 0	2.231 5e-05
	worst	28.546 0	312.309 7	0.921 8	0.049 2
	mean	19.504 4	27.737 8	0.852 0	0.002 8
	std	3.232 8	65.496 4	0.039 4	0.010 0
	time/s	0.980 7	0.577 4	9.240 2	2.271 2
f_8	best	8.249 3	0.041 6	20.363 9	4.304 9e-04
	worst	12.025 0	13.723 6	20.836 7	0.010 1
	mean	10.015 7	3.630 3	20.619 9	0.002 9
	std	0.836 2	3.064 5	0.120 2	0.002 3
	time/s	0.983 8	0.613 8	7.415 9	1.615 1
f_9	best	302.891 2	17.747 1	1.040 1e+03	3.547 5e-05
	worst	511.303 5	483.420 9	1.274 7e+03	1.066 0
	mean	379.999 5	164.309 6	1.160 5e+03	0.154 7
	std	44.512 3	98.511 0	52.332 8	0.351 9
	time/s	0.811 5	0.314 3	6.215 6	1.638 7
f_{10}	best	8.201 8	−329.704 1	702.745 2	−329.999 9
	worst	291.936 0	52.099 0	1.047 1e+03	−327.985 0
	mean	129.767 7	−194.362 3	882.209 5	−329.839 3
	std	50.993 4	90.238 5	70.051 9	0.396 5
	time/s	0.384 8	0.375 1	9.073 0	1.210 5
f_{11}	best	11.625 7	0.012 0	20.521 1	2.199 1e-08
	worst	847.521 2	3.857 1	41.156 6	1.476 2e-06
	mean	40.430 1	0.846 0	29.735 2	1.975 5e-07
	std	116.978 4	0.905 0	4.896 0	2.548 4e-07
	time/s	4.240 8	2.088 6	13.465 9	8.623 0
f_{12}	best	670.936 9	0.339 9	8.764 8	2.944 4e-07
	worst	9.105 9e+04	36.594 6	160.778 5	0.011 0
	mean	2.097 9e+04	9.765 4	17.892 7	2.239 8e-04
	std	2.128 9e+04	8.022 2	22.437 6	0.001 6
	time/s	3.022 1	2.011 0	9.828 7	8.661 8

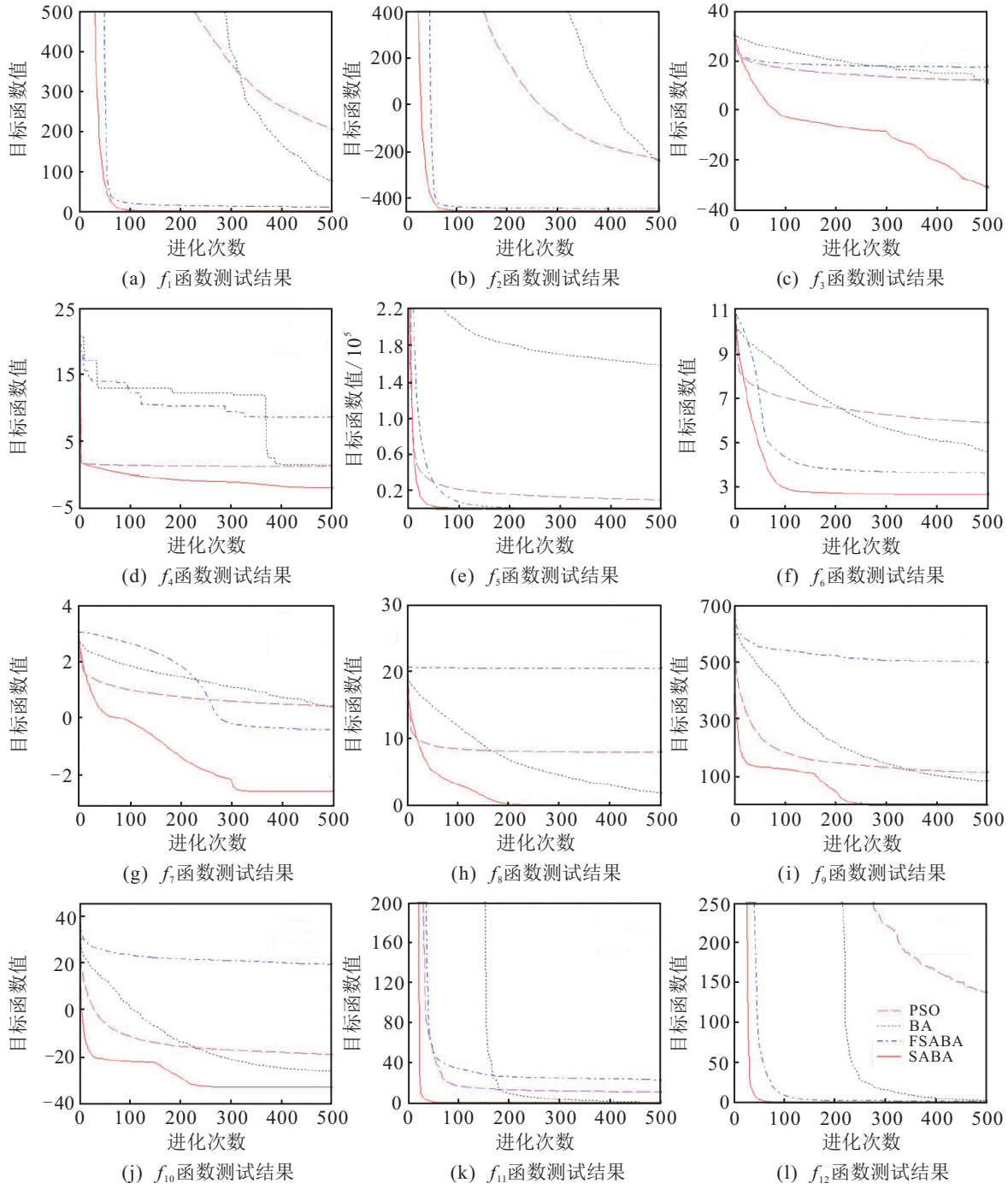

图 2　测试结果(50D)

各算法求解50D测试函数的迭代曲线如图2所示,各分图图注同图2(l)一致. 由于不同算法的求解结果存在较大差异,图中部分结果进行了对数化处理. 分析图2可知,SABA算法的收敛速度优于对照算法,在测试函数f_1和f_2,f_4和f_5,f_{11}和f_{12}中,其结果能够在50次迭代内趋于稳定,其余测试函数的计算结果能够在300次迭代内趋于稳定. 另外,图2中PSO算法的迭代曲线整体较为平滑,而其余算法的迭代曲线均会出现不同程度陡降,这主要是因为基本BA算法在寻优过程中使用了蝙蝠个体局部概率搜索操作,从而有助于提高求解精度. 另外,SABA算法求解测试函数f_9和f_{10}的迭代曲线在150 ～ 250次之间出现了较大降幅,这主要是因为SABA算法设定蝙蝠个体的局部搜索概率随迭代次数增加而增大,因此在算法后期,蝙蝠个体将进行更多局部搜索操作;同时,本文设定算法评价指数k为0.4,即算法在进行200次迭代后,将使用较小的局部搜索步长来提高算法求解精度.

以上结果表明,SABA算法在求解不同维度测试函数的过程中,能够通过自适应步长控制机制和变异机制有效解决基本BA算法易陷入局部最优的问题,从而具有良好的算法性能.

4 结 论

本文通过分析蝙蝠寻找猎物的行为特性,提出了与脉冲响度、脉冲发射频率和寻优过程自适应的步长控制机制和变异机制,SABA算法特点在于能够根据求解问题的可行解域和当前迭代进度调整蝙蝠移动步长,从而有利于提高算法的求解精度;此外,变异机制能够提高全局探索能力,避免算法过早陷入局部最优. 测试函数结果表明,SABA算法具有较好的算法性能. 在下一步工作中,作者考虑将SABA算法应用于工程问题中,以进一步验证算法的有效性.

参考文献(References)

[1] Poli R, Kennedy J, Blackwell T. Particle swarm optimization[J]. Swarm intelligence, 2007, 1(1): 33-57.

[2] Karaboga D, Basturk B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony(ABC) algorithm[J]. J of Global Optimization, 2007, 39(3): 459-471.

[3] Singh S, Ghose T, Goswami S K. Optimal feeder routing based on the bacterial foraging technique[J]. IEEE Trans on Power Delivery, 2012, 27(1): 70-78.

[4] Yang X S. A new metaheuristic bat-inspired algorithm[C]. Nature Inspired Cooperative Strategies for Optimization(NICSO 2010). Berlin Heidelberg: Springer Publications, 2010: 65-74.

[5] Yang X, Deb S. Cuckoo search via Levy flights[C]. World Congress on Nature & Biologically Inspired Computing. Piscataway: IEEE Publications, 2009: 210-214.

[6] Meng X, Liu Y, Gao X, et al. A new bio-inspired algorithm: Chicken swarm optimization[M]. Advances in Swarm Intelligence, Cham: Springer Publications, 2014: 86-94.

[7] Cheng S, Zhang Q, Qin Q. Big data analytics with swarm intelligence[J]. Industrial Management & Data Systems, 2016, 116(4): 646-666.

[8] Ali F. Swarm intelligent application in networks routing problem[J]. Int J of Computer Applications, 2016, 133(1): 25-28.

[9] 杨文强, 邓丽, 牛群, 等. 改进型细菌觅食算法及多货叉仓库调度应用[J]. 控制与决策, 2015, 30(2): 321-327.
(Yang W Q, Deng L, Niu Q, et al. Improved bacterial foraging algorithm and application of warehousing scheduling with multi-shuttles[J]. Control and Decision, 2015, 30(2): 321-327.)

[10] 刘韵婷,张嗣瀛,井元伟,等.基于粒子群优化的无线传感器网络非视距节点定位算法[J]. 控制与决策, 2015, 30(6): 1106-1110.
(Liu Y T, Zhang S Y, Jing Y W, et al. Non-line of sight node localization algorithm based on particle swarm optimization for wireless sensor networks[J]. Control and Decision, 2015, 30(6): 1106-1110.)

[11] 李枝勇, 马良, 张惠珍.蝙蝠算法收敛性分析[J]. 数学的实践与认识, 2013, 43(12): 182-190.
(Li Z Y, Ma L, Zhang H Z. Convergence analysis of bat algorithm[J]. J of Mathematics in Practice and Theory, 2013, 43(12): 182-190.)

[12] Fister I, Fister D, Yang X S. A hybrid bat algorithm[J]. Elektrotehniski Vestnik, 2013, 1(80): 1-7.

[13] 肖辉辉, 段艳明. 基于DE算法改进的蝙蝠算法的研究及应用[J]. 计算机仿真, 2014, 31(1): 272-277.
(Xiao H H, Duan Y M. Research and application of improved Bat algorithm based on DE algorithm[J]. Computer Simulation, 2014, 31(1): 272-277.)

[14] Chen Z, Zhou Y, Lu M. A simplified adaptive bat algorithm based on frequency[J]. J of Computational Information Systems, 2013, 9(16): 6451-6458.

[15] 刘长平, 叶春明. 具有混沌搜索策略的蝙蝠优化算法及性能仿真[J]. 系统仿真学报, 2013, 25(6): 1183-1188.
(Liu C P, Ye C M. Bat algorithm with chaotic search strategy and analysis of its property [J]. J of System Simulation, 2013, 25(6): 1183-1188.)

[16] Yang N C, Le M D. Optimal design of passive power filters based on multi-objective bat algorithm and pareto front[J]. Applied Soft Computing, 2015, 35(C): 257-266.

[17] Wang G G, Chu H C E, Mirjalili S. Three-dimensional path planning for UCAV using an improved bat algorithm[J]. Aerospace Science & Technology, 2016, 49: 231-238.

（责任编辑：孙艺红）