自适应步长萤火虫群多模态函数优化算法

黄正新 周永权
(广西民族大学数学与计算机科学学院 南宁 530006)

摘 要 针对萤火虫群优化(GSO)算法优化多模态函数存在收敛速度慢和求解精度低等缺陷,提出一种自适应步长萤火虫群多模态函数优化算法(SASGSO)。该算法解决了萤火虫群优化(GSO)算法优化多模态函数所存在的不足;同时SASGSO算法也可找到多模态函数的所有极值点。数值实验仿真表明,该算法具有操作简单、易理解、收敛速度快和求解精度高等优点。

关键词 人工萤火虫,多模态函数,GSO,SASGSO
中图法分类号 TP18 **文献标识码** A

Self-adaptive Step Glowworm Swarm Optimization Algorithm for Optimizing Multimodal Functions

HUANG Zheng-xin ZHOU Yong-quan
(College of Mathematics and Computer Science, Guangxi University for Nationalities, Nanning 530006, China)

Abstract Because the GSO algorithm has slow convergence and low precision defects when optimizing the multi-modal function, a self-adaptive step glowworm swarm optimization(SASGSO) algorithms was proposed in this paper. This algorithm can overcome slow convergence and low precision defects of the GSO algorithm simultaneously it can find all peaks of the multi-modal function. Experiments show that, the SASGSO algorithm has the advantages of simple operation, easy to understand, fast convergence rates and high precision.

Keywords Glowworm, Multimodal function, GSO, SASGSO

1 引言

函数优化问题是在工程、控制、决策中普遍存在的一类优化问题,其优化目标函数可能包含多个在一定范围内的连续变量。传统的优化手段对目标函数要求高,如需满足可导或可微等,导致有些函数难以优化,容易陷入局部解而难以得到全局最优解。近年来,人们从仿生学的机理中受到启发,提出了许多用于求解目标函数优化问题的新方法,如模拟退火算法、遗传算法、蚁群算法、鱼群算法等。然而由于函数优化问题的复杂性,每种算法都表现出各自的优势和缺陷。

函数优化问题大多属于多模态函数优化问题(Multimodal Function Optimization Problems),也称多峰函数优化问题。多模态函数是指具有多个峰值点的函数,这些峰值点的函数值可能是相同的,也可能不同。在实际优化问题中,不仅要寻找可行域内的全局最优解,还需要寻找多个全局最优解和有意义的局部最优解,以便给决策者提供多种选择和多方面的参考信息。所以在优化多模态函数时,同时获得多模态函数的全局最优解和局部最优解,是一项具有理论意义和应用价值的工作[4-5,7-11]。

萤火虫群优化(Glowworm Swarm Optimization, GSO)算法是印度学者 Krishnanand 和 Ghose 于 2005 年提出的一种新型仿生群智能优化算法。目前,GSO 算法已在多模态函数优化、多信号源追踪、多信号源定位、集群机器人学和有害气体泄漏定位检测等方面得到成功地应用[1]。特别是 GSO 算法具有在获得多模态函数的所有极值点时,不需要任何记忆、梯度信息和不依赖于任何全局信息等优点。但算法在运行后期仍存在着收敛速度慢和求解精度低等缺陷,大大地限制了 GSO 算法的应用和推广。

本文针对 GSO 优化多模态函数所存在的问题,提出一种自适应步长萤火虫群多模态函数优化算法(self-adaptive step GSO;SASGSO)。数值实验结果表明,该算法可以有效改进 GSO 算法优化多模态函数存在收敛速度慢和求解精度不高的缺点,而且可以同时找到多模态函数的所有极值点,适用于优化任意多模态函数。

2 GSO 算法优化多模态函数

当 GSO 算法优化多模态函数时,每个个体萤火虫 $i(i=1,\cdots,n)$ 被随机分布到目标函数求解空间,然后根据其邻域内其他个体的发出信号(荧光素)强度选择移动。在自然中,萤火虫发光越亮,越能吸引同伴求偶或觅食。GSO 算法是基于这种自然现象机制设计出来的一种新型仿生群智能算法。

GSO 算法优化多模态函数主要有 5 个步骤:1)使用式(1)把萤火虫 i 在 t 迭代的位置 $x_i(t)$ 对应的目标函数值 $J(x_i(t))$ 转化为荧光素值 $l_i(t)$,其中 γ 为荧光素更新率;2)每

到稿日期:2010-08-31 返修日期:2010-12-06 本文受广西自然科学基金项目(0991086),国家民委科研项目基金(08GX01)资助。
黄正新(1987-),男,硕士生,主要研究方向为计算智能及应用;周永权(1962-),男,博士,教授,主要研究方向为计算智能理论与方法、神经网络及应用,E-mail:yongquanzhou@126.com(通信作者)。

只萤火虫在其动态决策域半径 $r_d^i(t)$ 内，选择荧光素值比自己高的个体组成其邻域集 $N_i(t)$，其中 $0<r_d^i(t)\leqslant r_s$，r_s 为萤火虫个体的感知半径；3)用式(2)计算萤火虫 i 移向邻域集内个体 j 的概率 $p_{ij}(t)$；4)选择移动对象，进行移动，根据式(3)更新位置，其中 s 为移动步长；5)根据式(4)更新动态决策域半径的值。

$$l_i(t)=(1-\rho)l_i(t-1)+\gamma J(x_i(t)) \tag{1}$$

$$p_{ij}(t)=\frac{l_j(t)-l_i(t)}{\sum_{k\in N_i(t)}l_k(t)-l_i(t)} \tag{2}$$

$$x_i(t+1)=x_i(t)+s*\left(\frac{x_j(t)-x_i(t)}{\|x_j(t)-x_i(t)\|}\right) \tag{3}$$

$$r_d^i(t+1)=\min\{r_s,\max\{0,r_d^i(t)+\beta(n_t-|N_i(t)|)\}\} \tag{4}$$

2008 年，Krishnanand 和 Ghose 通过仿真实验对 GSO 算法各参数取值做了比较全面的实验分析，从获得测试的目标函数所有极值点因素考虑，最后给出 GSO 算法中各参数参考取值为：$\rho=0.4$，$\gamma=0.6$，$\beta=0.08$，$n_t=5$，$s=0.03$，$l_0=5$。这些参数值为人们解决函数优化及相关目标优化问题提供了方便。

3 SASGSO 算法

3.1 GSO 算法缺点

在 GSO 算法中，我们发现算法执行到后期时，萤火虫个体徘徊在峰值附近，甚至出现振荡现象，这是导致搜索到解的精度不高的主要原因。通过多次实验分析，我们认为出现徘徊或振荡现象的主要原因是萤火虫移动步长选取不够恰当。在 2008 年，Krishnanand 和 Ghose 主要从获得目标函数峰值个数考虑，使用相当小的固定移动步长：$s=0.03$，主要原因是一方面不至于算法初期跳过函数峰值，另一方面考虑到算法后期收敛速度的问题。但对于 GSO 算法来说，使用固定步长不能同时满足算法初期和后期需要不同步长的要求，因此，固定移动步长是导致算法收敛速度慢和求解精度不高的主要原因。

3.2 GSO 算法改进及 SASGSO 算法描述

文献[2]已注意到使用递减的步长选择方式，可以减少执行算法的循环次数，所使用的步长计算公式为：$s(t)=s(0)*q^t$，其中，$s(0)$ 表示初始步长，$s(t)$ 表示第 t 次迭代步长，$q=0.96$。这种改进在基本上满足了算法初期和后期需要不同步长的要求。但是，在同一次循环中不同的萤火虫个体其邻域分布密度通常是不同的。随着循环次数的增加，距峰值中心较近的个体，其邻域分布相对密集，相反，距峰值中心较远的个体其邻域分布相对稀疏。在这种分布密度不同的情况下使用相同的步长，既不利于离峰值中心较远的萤火虫快速收敛到峰值上，也不利于在峰值附近的萤火虫进入较小的调整，从而不能全局搜索到高精度的解。因此，在 GSO 算法中，需要一种自适应步长选择机制来提高其优化性能显得尤为重要。

通过对 GSO 算法存在问题的分析，本文提出一种自适应步长的萤火虫群优化算法，简称 SASGSO。在 SASGSO 算法中，萤火虫个体选择移动步长大小与其邻域内萤火虫个体分布密度有关。在同一次循环中，邻域内萤火虫分布相对稀疏的个体使用较大的步长，加快到峰值上的收敛，相反，分布相对密集的个体选择较小的步长，避免产生振荡现象，这样有利于提高求解的精度。这种自适应步长选择方式，既可以考虑到算法整体的收敛速度，又利于提高求解精度。我们使用式(5)计算每个个体 i 在第 t 次循环内分布密度：

$$H(i,t)=\frac{\min(XN)+\text{mean}(XN)}{\max(XN)+\text{mean}(XN)} \tag{5}$$

式中，$XN=\{\|x_j(t)\||j\in N_i(t)\}$，$\|x_j(t)\|$ 表示向量 $x_j(t)$ 的模；$\min(XN)$，$\text{mean}(XN)$ 和 $\max(XN)$ 分别表示 XN 集合元素的最小值，平均值和最大值，最小值和最大值可以较好地描述邻域集分布最大距离，平均值可以很好地描述其邻域集分布集中趋势；$H\in(0,1)$，当分布距离很小很密集时，H 的值很接近 1。注意，为方便计算，当邻域内只包含一个个体时，令 $H=0.5$。在计算邻域分布密集程度的基础之上，萤火虫 i 在第 t 次循环中移动步长 $s(i,t)$ 为：

$$s(i,t)=s(0)*q^{(t*H(i,t))}+rand() \tag{6}$$

式中，$rand()$ 是一个随机产生的常数，有利于算法后期保持继续搜索，其范围可根据需要的精度设置。基于以上对 GSO 改进的分析，SASGSO 算法描述如下：

1. 算法中各参数表示意义如下：t 为迭代次数；$iter\_max$ 为最大迭代次数；i 为第 i 个萤火虫；$l_i(t)$ 为荧光素值；$x_i(t)$ 为第 i 个萤火虫 i 在 t 代的位置；$J(x_i(t))$ 为目标函数值；γ 为荧光素更新率；$\rho\in[0,1]$ 为任意一参数；$r_d^i(t)$ 为第 i 个萤火虫决策域半径；$N_i(t)$ 为比第 i 个萤火虫荧光素值高的个体组成其邻域集；$p_{ij}(t)$ 为萤火虫 i 移向邻域集内个体 j 的概率；$XN=\{\|x_j(t)\||j\in N_i(t)\}$，$\|x_j(t)\|$ 表示向量 $x_j(t)$ 的模；$\min(XN)$，$\text{mean}(XN)$ 和 $\max(XN)$ 分别表示 XN 集合元素的最小值，平均值和最大值；$H(i,t)$ 为个体 i 在第 t 次循环内的分布密度；$s(i,t)$ 为第 i 个个体 t 次迭代步长。萤火虫个数 n，搜索空间维数 m，初始萤火虫素值 $l_i(0)$，初始决策域半径 $r_d^i(0)$，萤火虫感知最大半径 r_s，最大迭代次数 $iter\_max$，初始步长 $s(0)$，随机产生萤火虫初始位置 $x_i(0)$。

2. 算法描述：

置 $t=1$；$q=0.96$；$\beta=0.08$；

While($t<iter\_max$) do：

{

for $i=1$ to n do

$l_i(t)=(1-\rho)l_i(t-1)+\gamma J(x_i(t))$；

for each glowworm i do：% Movement-phase

{

$N_i(t)=\{j:\|x_j-x_i\|<r_d^i(t);l_i(t)<l_j(t)\}$；

其中，$\|\vec{x}\|$ 表示 $\vec{x}$ 的范数。

for each glowworm $j\in N_i(t)$ do：

$p_{ij}(t)=\frac{l_j(t)-l_i(t)}{\sum_{k\in N_i(t)}l_k(t)-l_i(t)}$

$j=$select_glowworm $(\vec{p})$

其中，$\vec{p}$ 表示 p 中概率值最大萤火虫。

$H(i,t)=\frac{\min(XN)+\text{mean}(XN)}{\max(XN)+\text{mean}(XN)}$

$s(i,t)=s(0)*q^{(t*H(i,t))}+rand()$

$x_i(t+1)=x_i(t)+s(i,t)*\left(\frac{x_j(t)-x_i(t)}{\|x_j(t)-x_i(t)\|}\right)$

$r_d^i(t+1)=\min\{r_s,\max\{0,r_d^i(t)+\beta(n_t-|N_i(t)|)\}\}$

}

$t\leftarrow t+1$

}

3.3 SASGSO 算法荧光素值收敛性分析

本文提出的改进并没有改变 GSO 算法荧光素值的收敛

性。文献[2,3]已给出了GSO算法中萤火虫个体荧光素值收敛证明。证明指出，存在点列 $X=\{x_i(1),\cdots,x_i(t)\}$，其中，

$$x_i(t+1)=x_i(t)+s*\left(\frac{x_j(t)-x_i(t)}{\|x_j(t)-x_i(t)\|}\right) \tag{7}$$

使得 $\lim\limits_{t\to\infty} l_i(t)=\lim\limits_{t\to\infty}(1-\rho)*l_i(t-1)+\gamma*J(x_i(t))=\frac{\gamma}{\rho}*J_i^*$ 成立。

式中，J_i^* 表示萤火虫 i 所在峰的最大值，s 为固定的移动步长。在SASGSO算法中，用式(6)计算结果 $s(i,t)$ 代替式(7)中固定的 s 值计算。需要指出的是 $\lim\limits_{t\to\infty} s(i,t)=\lim\limits_{t\to\infty}(s(0)*q^{*H(i,t)}+rand())=rand()$，即 $t\to\infty$ 时，步长是一个大于零的随机常数，可以保证存在点列 $X=\{x_i(1),\cdots,x_i(t')\}$，$t'\to\infty$，使得 $\lim\limits_{t'\to\infty} l_i(t')=\lim\limits_{t\to\infty}(1-\rho)*l_i(t'-1)+\gamma*J(x_i(t'))=\frac{\gamma}{\rho}*J_i^*$ 成立，所以，SASGSO算法中萤火虫个体荧光素值最终收敛到 $\frac{\gamma}{\rho}*J_i^*$。如果式(6)没有随机项时，当 $t\to\infty$，步长 $s(i,t)\to 0$ 时，导致算法不能继续搜索，不能保证所有萤火虫的荧光素值最收敛到 $\frac{\gamma}{\rho}*J_i^*$。

4 实验结果与分析

4.1 实验环境

本实验在Matlab 2008a，电脑CPU T4400 2.20GHZ，内存2048MB，Windows XP操作系统上进行。

4.2 算法比较

4.2.1 SASGSO算法与GSO算法比较

为了验证本文提出的算法可以有效改进GSO算法优化多模态函数存在的不足，提高GSO算法优化多模函数的收敛速度和求解精度，我们选用文献[2]中的 J_1 函数进行测试。

J_1 函数的解析式为：

$$J_1(x,y)=3(1-x^2)e^{-[x^2+(y+1)^2]}-10\left(\frac{x}{5}-x^3-y^5\right)e^{-(x^2+y^2)}-\frac{1}{3}e^{-[y^2+(x+1)^2]},x,y\in(-3,3)$$

函数 J_1 在搜索区域内有3个极大值点。

实验对比两种算法在相同迭代次数下的函数平均值，计算公式是：$J(t)=\frac{1}{n_c}\sum_{i=1}^{n_c}J(x_i(t))$，其中 n_c 表示最终收敛到同一个峰值的萤火虫数目。两种算法各运行20次。为了确保两种算法每次测试都能搜索到3个极值点，本实验设置萤火虫个数 $n=70$，SASGSO算法其他参数设置如下：$r_s=2.5$，$r_d(0)=1.8$，$s(0)=0.12$，$q=0.98$，$\rho=0.4$，$\gamma=0.6$，$\beta=0.08$，$n_t=5$，$l_i(0)=5$。GSO算法其他参数设置与文献[2]相同。表1是两种算法分别在100，200，300，400和500次迭代时的平均值，图1是两种算法优化函数平均值随迭代次数变化图。

表1 SASGSO与GSO算法优化函数 J_1 的实验平均值

迭代次数 算法	峰序号	100	200	300	400	500
GSO	N1	3.10304	3.56051	3.58507	3.58930	3.58991
	N2	3.41829	3.77103	3.77250	3.77274	3.77290
	N3	7.69486	8.09709	8.10283	8.10296	8.10273
SASGSO	N1	3.58949	3.59212	3.59230	3.59240	3.59242
	N2	3.76885	3.77578	3.77638	3.77652	3.77655
	N3	8.10384	8.10576	8.10608	8.10620	8.10621

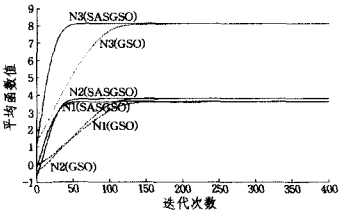


图1 SASGSO与GSO优化函数 J_1 的平均值迭代次数变化图

观察表1数据和图1收敛曲线可知，本文算法在收敛速度和求解精度上都比GSO算法要好。在相同的迭代次数下，SASGSO算法优化结果都比GSO算法大。值得指出的是，当SASGSO算法迭代到200次时，计算得3个极大值点的函数值分别是：3.59212，3.77578和8.10576；而GSO算法迭代到500次时，才计算出3个极大值点的函数值分别为：3.58991，3.77290和8.10273；3个极值都比SASGSO算法计算结果小，也就是说，SASGSO算法比GSO算法少用300次迭代，却计算出比GSO算法更好的结果，少用的迭代次数是SASGSO算法使用迭代次数的1.5倍。

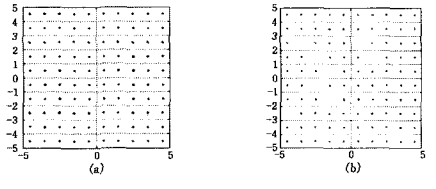
此外，我们还测试了文献[2]中的 J_2 函数，它是改进的Rastrigin函数，是一多模态函数优化常用的基准测试函数之一。

$J_2=20+(x^2-10\cos(2\pi x)+y^2-10\cos(2\pi x))$，

$x,y\in[-5,5]$

在优化区域内 J_2 函数有100个峰值。

本实验中，SASGSO算法设置萤火虫数目 n 与文献[2]相同，即 $n=1500$。



(a) SASGSO算法迭代200次时，所有萤火虫的位置分布图　(b) 文献[2]用GSO算法迭代500次时，所有萤火虫的位置分布图

图2

由图2可知，SASGSO算法迭代到200次时，所有萤火虫个体已全部收敛到峰值上，且函数的100个极值点全部被找到。相比之下，GSO算法迭代到500次时，虽然全部萤火虫也都收敛到峰值上，但捕获到极值点的个数只有92个，比SASGSO算法少了8个。实验说明，使用相同的萤火虫数目时，SASGSO算法捕获函数极值点个数的能力比GSO算法强。

综合以上实验对比分析可知，本文提出的改进算法不仅可以有效提高GSO算法的收敛速度和求解精度，还可以有效提高GSO算法捕获目标函数极值点个数的能力。

4.2.2 SASGSO与其它文献算法比较

本节实验选择文献[3,6]同时使用的两个典型多模态函数进行测试，每个函数用SASGSO迭代20次，最后随机选择一次运行结果的数值作为本文提出算法的求解结果与文献算法的结果进行比较分析。表2－表3中"－－"表示该文献没有提供该项数据。

函数 1　改进的 Himmelblau 函数

$\max f(x,y)=660-(x^2+y-11)^2-(x+y^2-7)^2$，

$x,y\in[-6,6]$

函数 2　Shekel's Foxholes 函数

$$\min f(X)=\left[\frac{1}{500}+\sum_{j=1}^{25}\frac{1}{j+\sum_{i=1}^{2}(x_i-a_{ij})}\right]^{-1}$$

式中，$a_{ij}^k=$

$$\begin{pmatrix} -32 & -16 & 0 & 16 & 32 \\ -32+k\times16 & -32+k\times16 & -32+k\times16 & -32+k\times16 & -32+k\times16 \end{pmatrix},$$

$k=0,1,2,3,4$。改进的 Himmelblau 函数有 4 个函数值相等的极大值点，且极大值为 660。实验参数设置如下：萤火虫数目 $n=200$，$q=0.98$，迭代次数为 650，$Rs=r_d^i(0)=3$，$s(0)=0.4$，$\rho=0.4$，$\gamma=0.6$，$\beta=0.08$，$n_t=5$，$l_i(0)=5$。Shekel's Foxholes 函数有 25 个函数值不等的极小值点，其实验参数设置为：萤火虫数目 $n=1200$，$q=0.98$，迭代次数为 1200，$Rs=r_d^i(0)=8.5$，$s(0)=2.5$，其余参数设置与改进的 Himmelblau 函数相同。两个函数实验测试结果如表 2 和表 3 所列。

表 2　SASGSO 算法与文献算法优化 Himmelblau 函数结果

算法	最优解集	函数值	发现率	迭代次数
MPES	N1(2.999950 64, 2.000018 94)	659.9999999 6	——	——
	N2(3.584426 82, −1.848071 22)	659.9999999 6	——	
	N3(−2.805116 02, 3.131279 53)	659.9999999 3	——	
	N4(−3.779329 09, −3.283158 95)	659.9999999 2	——	
CMPGA	N1(2.999962 806666, 2.000022 888205)	659.9999999 56937	100%	900
	N2(3.584436 020313, −1.848171 089336)	659.9999999 70170	100%	
	N3(−2.805115 513912, 3.131283 885273)	659.9999999 66907	100%	
	N4(−3.779306 201273, −3.283158 572348)	659.9999999 69017	100%	
SASGSO	N1(2.999997 197455, 2.000009 061717)	659.9999999 98821	100%	650
	N2(3.584433 602037, −1.848119 788248)	659.9999999 97637	100%	
	N3(−2.805131 794556, 3.131298 515552)	659.9999999 85761	100%	
	N4(−3.779309 455141, −3.283198 161332)	659.9999999 93154	100%	

表 3　本文算法与文献算法优化 Shekel's Foxholes 函数实验结果

算法		MPES	CMPGA	SASGSO
最优解函数值（发现率）	N1	0.998003 84(——)	0.999999 845161(100%)	0.998003 837794(100%)
	N2	1.992030 90(——)	2.005344 519708(100%)	1.992030 900225(100%)
	N3	2.982105 16(——)	3.005343 303859(100%)	2.982105 156817(100%)
	N4	3.968250 11(——)	3.999996 075272(100%)	3.968250 105641(100%)
	N5	4.950491 23(——)	4.999996 105545(100%)	4.950491 231787(100%)
	N6	5.928845 13(——)	6.003201 196214(100%)	5.928845 125607(100%)
	N7	6.903335 69(——)	6.999982 147455(100%)	6.903335 694363(100%)
	N8	7.873992 98(——)	8.003186 484800(100%)	7.873992 977146(100%)
	N9	8.840835 96(——)	8.999970 488267(100%)	8.840835 962756(100%)
	N10	9.803897 94(——)	9.999975 419727(100%)	9.803897 942948(100%)
	N11	10.763180 67(——)	10.999969 99556(100%)	10.763180 669155(100%)
	N12	11.718699 56(——)	12.003156 99023(100%)	11.718699 562088(100%)
	N13	12.670505 81(——)	12.999937 37525(100%)	12.670505 811136(100%)
	N14	13.618608 92(——)	13.999928 01300(100%)	13.618608 923889(100%)
	N15	14.563054 16(——)	5.005294 06783(100%)	14.563054 157597(100%)
	N16	15.503816 80(——)	15.999936 98644(100%)	15.503816 803951(100%)
	N17	16.440907 31(——)	17.003104 37648(100%)	16.440907 314755(100%)
	N18	17.374406 50(——)	18.003090 63420(100%)	17.374406 500768(100%)
	N19	18.304309 52(——)	18.999868 46531(100%)	18.304309 520096(100%)
	N20	19.230678 13(——)	19.999901 46587(100%)	19.230678 130659(100%)
	N21	20.153486 96(——)	20.999930 61615(100%)	20.153486 960472(100%)
	N22	21.072687 51(——)	22.003090 14807(100%)	21.072687 508733(100%)
	N23	21.988407 55(——)	23.005225 58529(100%)	21.988407 546116(100%)
	N24	22.900634 08(——)	23.999858 01438(100%)	22.900634 082795(100%)
	N25	23.809434 47(——)	24.999901 50463(100%)	23.809434 471297(100%)
迭代次数		(——)	18000	1200

由于极值点个数比较多，表 2 没有一一列出 MPES 和 CMPGA 算法求得的每个极值点的坐标，只列出它们的函数值。以下列出本文算法优化 Shekel's Foxholes 函数得到的 25 个极值点的坐标：

N1(−31.978420258154,−31.978293020449)，

N2(−15.984937628006,−31.969801686451)，

N3(−0.013517709739,−31.965167084296)，

N4(15.990571894877,−31.961389510662)，

N5(31.958816811058,−31.958640394408)，

N6(−31.952755659804,−15.985838988654)，

N7(−15.981280485006,−15.973415080225)，

N8(0.019421602548,−15.974002174229)，

N9(15.9723105231159,−15.972851330668)，

N10(31.943453527174,−15.976583267674)，

N11(−31.944970384007,−0.030249229480)，

N12(−15.967529746507,0.025020411769)，

N13(0.0275175121689,0.027708629217)，

N14(15.966067162602,−0.026827523730)，

N15(31.932832269898,−0.025028053417)，

N16(−31.930254330333,15.973157774619)，

N17(−15.958802765224,15.962358902117)，

N18(0.027476755008,15.961415497543)，

N19(15.968014629285,15.965002969922)，

N20(31.924866545302,15.959438277004)，

N21(−31.926497710410,31.926460580692)，

N22(−15.961945762849,31.921544859095)，

N23(0.037722925146,31.921424051558)，

N24(15.956974978078,31.918210982726)，

N25(31.921114801689,31.921119624668)。

其中，极值点坐标序号与表中的函数值序号相对应。

由表中数据可知，在测试改进的 Himmelblau 函数实验中，SASGSO 算法与 CMPGA 算法一样，在每次运行中都能 100%地发现函数的 4 个极值点。从函数值计算结果相同的有效值来看，CMPGA 算法除了 N1 极值点外，都明显优化于 MPES 算法。而 SASGSO 算法计算出的 4 个极值点的函数值都明显大于 CMPGA 算法的计算结果，CMPGA 算法计算出 4 个峰值的最大值为 659.999999970170，比 SASGSO 算法计算出 4 个峰值的最小值 659.999999985761 小 0.000000015591。而且 SASGSO 算法使用的迭代次数为 650，比 CMPGA 算法少了 250 次。在测试 Shekel's Foxholes 函数实验中，SASGSO 算法也和 CMPGA 算法一样每次运行都能 100%发现函数的 25 个极值点，但是 SASGSO 算法只使用了 1200 次迭代，是 CMPGA 算法迭代次数的 1/15。

在相同的有效值下相比，SASGSO 算法计算出的函数值并不比 MPES 算法差。Shekel's Foxholes 函数有 25 个极值点，是改进的 Himmelblau 函数极值个数的 6 倍；而 SASGSO 算法优化 Shekel's Foxholes 函数所用的迭代次数是 1200，不到优化改进的 Himmelblau 函数的两倍，随着函数峰数目成倍增加而不需要明显增加迭代次数。总体上，SASGSO 算法优化两个测试函数的性能都优于文献中的两种算法。

结束语 针对 GSO 算法优化多模态函数存在的问题，提出了一种自适应步长萤火虫群优化算法。算法中的萤火虫个体根据其邻域分布密集程度自适应调整其移动步长，以达到使用合理的移动步长进行移动。通过实验对比分析表明，SASGSO 算法不仅提高了 GSO 算法的收敛速度和求解精度，还提高了其捕获峰值个数的能力。与其它同类文献算法相比，ASAGSO 算法使用的迭代次数更少，却能找到更精确的解，随着函数峰数目成倍增加而不需要明显增加迭代次数，所以，SASGSO 算法非常适用于解决多模态函数优化及相关工程、控制、决策中普遍存在的优化问题。

参 考 文 献

[1] Krishnand K N, Ghose D. Glowworm swarm optimisation: a new method for optimising multi-modal functions[J]. Int. J. Computational Intellingence Studies, 2009, 1(1): 93-119

[2] Krishnand K N, Ghose D. Glowworm swarm optimisation for simultaneous Capture of mutiple local optima of multimodal functions[J]. Swarm Intell, 2009(3): 87-124

[3] Krishnand K N, Ghose D. Glowworm swarm based optimization algorithm for multimodal functions with collective robotics applications[J]. Multiagent and Grid Systemss, 2006, 2(3): 209-222

[4] 李敏强，寇纪淞. 多模态函数优化的协同多群体遗传算法[J]. 自动化学报，2002，28(4)：497-504

[5] 覃俊，康立山，陈毓屏. 一种新的求解多峰函数优化问题的动态演化算法[J]. 计算机学，2004，31(3)：134-136

[6] 王湘中，喻寿益. 多模态函数优化的多种群进化策略[J]. 控制与决策，2006，21(3)：285-288

[7] 郑士芹，王秀峰. 基于多模态函数优化的改进克隆选择算法[J]. 计算机工程与应用，2006(3)：15-18

[8] 腾泓虬，李春华. 小生境人工免疫算法用于多峰函数优化[J]. 计算机仿真，2009，26(12)：148-150

[9] 陆青，梁昌勇，杨善林，等. 面向多模态函数优化的自适应小生境遗传算法[J]. 模式识别与人工智能，2009，22(1)：91-100

[10] 郑高飞，王秀峰. 带子群自组织蠕虫算法及其在多模态问题中的应用[J]. 计算机工程，2006，32(7)：182-184

[11] 杨诗琴，须文波，孙俊. 用于多峰函数优化的改进小生境微粒群算法[J]. 计算机应用，2007，27(5)：1191-1193

[12] 李莉，李洪奇，谢绍龙. 一种有效的多峰函数优化算法[J]. 计算机应用研究，2008，25(10)：2973-2976

(上接第 211 页)

次得到的输出值都具有不确定性，但所有输出值都在合理范围内上下波动。

结束语 建立在定性规则基础上的不确定推理的核心问题是规则中的定性概念的定量描述。基于概率理论的不确定性推理需要给出规则中定性概念的概率密度函数，这些函数的获取需要很大工作量，影响了概率方法的实际应用。同样模糊推理方法要给出精确的隶属函数，且对于相同输入，都会得到完全相同的输出[10]。云模型用期望、熵和超熵描述自然语言中的定性概念，简单有效地解决了不确定的定性概念的定量转换问题。在云模型基础上构建的规则发生器描述定性规则同样简单易于理解，且对于相同输入，每次得到的输出值都具有不确定性，但整体上又保持了与模糊推理相同的变化趋势。近年来，不确定性推理方法在智能控制、知识发现与预测等研究领域具有广泛应用。李德毅院士提出的定性、定量转换方法云模型已被国内外同行公认，基于云模型的不确定推理方法为不确定推理研究提供了一条新途径，必将有广泛的应用前景。

参 考 文 献

[1] 李德毅，刘常昱，杜鹢，等. 不确定性人工智能[J]. 软件学报，2004，15(11)：1583-1594

[2] Bouchon-Neunier B, Valverde L, Yager R R. Uncertain in Intelligent Systems[M]. Amsterdam, New York: North Holland, 1993

[3] 张博锋，谭建荣，蔡青. 通用不确定性推理模型[J]. 模式识别与人工智能，1999，12(3)：292-299

[4] 张文修，梁怡. 不确定性推理原理[M]. 西安：西安交通大学出版社，1996

[5] Pearl J. Fusion, propagation, and structuring in belief networks [J]. Artificial Intelligence, 1986, 29: 241-288

[6] Kleite G D. Bayesian diagnosis in expert systems[J]. Artificial Intelligence, 1992, 54: 1-32

[7] Kim H, Swain P H. Evidential reasoning approach to multisource data classification in remote sensing[J]. IEEE Transactions on Systems, Man, and Cybernetics, 1995, 25(8): 1257-1265

[8] Pawlak Z. Rough sets[J]. International Journal of Information and Computer Science, 1982, 11: 344-356

[9] Zadeh L A. Fuzzy sets[J]. Information and Control, 1965, 8(3): 338-353

[10] 王立新. 模糊系统与模糊控制[M]. 王迎军，译. 北京：清华大学出版社，2003

[11] Karnik N N, Mendel J M, Liang Q. Type-2 fuzzy logic systems [J]. IEEE Transactions on Fuzzy Systems, 1999, 7(6): 643-658

[12] Mendel J M, John R I, Liu F L. Interval type-2 fuzzy logic systems made simple [J]. IEEE Transactions on Fuzzy Systems, 2006, 14(6): 808-821

[13] 李德毅，杜鹢. 不确定性人工智能[M]. 北京：国防工业出版社，2005

[14] 李德毅，孟海军，史学梅. 隶属云和隶属云发生器[J]. 计算机研究与发展，1995，32(6)：15-20

[15] 李德毅，刘常昱. 论正态云模型的普适性[J]. 中国工程科学，2004，6(8)：28-33

[16] 宋远骏，杨孝宗，李德毅，等. 考虑环境因素的计算机可靠性云模型评价[J]. 计算机研究与发展，2001，38(5)：631-636

[17] 宋远骏，李德毅，杨孝宗，等. 电子产品可靠性的云模型评价方法[J]. 电子学报，2000，28(12)：74-77

[18] 刘禹，李德毅，张光卫，等. 云模型雾化特性及在进化算法中的应用 [J]. 电子学报，2009，37(8)：1651-1658