基于 DBSCAN 聚类的细菌自适应步长觅食算法

汪　洋,谢　芬,刘　清

(南京师范大学计算机科学与技术学院,江苏 南京 210023)

[**摘要**]　自适应细菌觅食算法(adaptive bacterial foraging algorithm,ABFA)在一定程度上解决了经典觅食算法步长选择的问题,加快了算法的收敛速度. 但随着细菌代价函数值的减小,自适应细菌觅食算法原有的趋化步长调整函数易使步长快速进入极小,造成算法早熟. 本文提出了一种基于 DBSCAN 聚类的细菌自适应步长觅食算法(DBSCAN-based adaptive bacterial foraging algorithm,DBSCAN-ABFA),算法利用 DBSCAN 聚类对核心点区域的细菌进行标记,通过对被标记细菌采用改进的趋化步长调整函数,降低自适应步长的缩小速率来解决步长快速进入极小的问题,最终避免算法早熟,并通过实验验证了算法的有效性.

[**关键词**]　细菌觅食算法,自适应步长,算法早熟,DBSAN

[**中图分类号**]TP18　[**文献标志码**]A　[**文章编号**]1672-1292(2014)03-0063-07

DBSCAN-Based Adaptive Bacterial Foraging Algorithm

Wang Yang,Xie Fen,Liu Qing

(School of Computer Science and Technology,Nanjing Normal University,Nanjing 210023,China)

Abstract:The adaptive bacterial foraging algorithm(ABFA),to some extent,solves the problem of chemotactic step size choice in bacterial foraging algorithm(BFA)and subsequently accelerates the convergence rate. However,along with the decrease of bacterial cost function value,the original chemotactic step size adjust function is liable to make chemotactic step size minimum,leading to the algorithm premature. Adaptive bacterial foraging algorithm based on density-based spatial clustering of applications with noise (DBSCAN-based adaptive bacterial foraging algorithm,DBSCAN-ABFA) is designed,with the purpose of avoiding the algorithm premature by changing the chemotactic step size adjust function of the labeled core points bacterial according to DBSCAN,and the improved chemotactic step size adjust function can reduce the shrink rate of step size,so the algorithm premature is ultimately avoided. To verify the feasibility of the algorithm,trials are also designed.

Key words:BFA,adaptive chemotactic step size,algorithm premature,DBSAN

细菌觅食算法(bacterial foraging algorithm,BFA)是 Kevin M Passino 在 2002 年提出的一种随机仿生优化算法[1-3],是一种模仿大肠杆菌(*E. coli*)行为的全局启发搜索式的仿生算法. BFA 算法在提出之后即成为研究热点,并广泛应用,例如运用在优化分布式电源配置[4]、不平衡电压下的感应电动机磁场效率评估[5]、多目标优化[6]、黑白图像边缘检测[7]、优化感应加热器的电容计算[8]等等.

传统的细菌觅食算法和其他全局优化算法(如微粒子群算法[9]、进化算法[10])以及混合优化算法(如细菌群优化算法[11])相比具有良好的全局搜索能力,但是收敛速度较慢. 通过加大步长,可以使算法的收敛速度加快,但在接近最优解释时,易产生震荡,使算法找不到最优解;而不适当的减小步长,又会使算法陷入局部最小,导致算法早熟[2,12]. 针对上述问题提出了一些自适应步长的细菌觅食算法[12-14]. 这些算法都是通过一个线性的自适应步长缩小函数使趋化步长在细菌觅食过程中随着代价函数值的减少逐渐缩小,来解决传统算法步长选择的问题,以提高算法的收敛速度. 但当算法接近最优解且代价函数值减小到一定程度时,线性的缩小步长会使步长趋近于零,造成细菌位置基本不变化,导致算法早熟. 综上所述,如

收稿日期:2014-05-20.

基金项目:国家自然科学基金(61103185)、江苏省高校自然科学基金项目(10KJD520004).

通讯联系人:汪洋,硕士研究生,研究方向:人工智能、图像处理. E-mail:leo_tony163@163.com

何在细菌种群接近最优解时,解决自适步长进入极小的问题,是避免算法出现早熟的关键.

针对上述问题,本文采用 DBSCAN 聚类算法[15,16]对细菌觅食算法进行阶段划分. 开始阶段细菌种群采用线性的自适应步长缩小函数,当种群中部分细菌接近最优解时,最优解附近区域细菌密度增大. 通过 DBSCAN 聚类算法对密度较大的核心点区域细菌进行标记,被标记的细菌已接近最优解. 此时为了避免线性的缩小步长导致被标记细菌步长趋近于零,对被标记细菌采用改进的自适应步长函数,减小了步长的缩小速率,最终达到提高最优解精度、避免算法陷入早熟的目的.

1 传统的细菌觅食算法

细菌觅食算法 BFA 是一种仿生类优化算法. 该算法包含趋化、繁殖和驱散 3 个步骤.

(1)趋化操作:在趋化过程中细菌向任意方向移动单位步长叫做翻转;细菌沿同一方向移动若干步称为前进. 设 $\boldsymbol{\theta}^i(j,k,l)$ 表示第 i 个细菌在第 j 次趋化,第 k 次复制,第 l 次驱散后的位置. $\boldsymbol{C}(i)$ 代表了第 i 个细菌的步长,$\boldsymbol{\Delta}(i)$ 代表随机方向向量.

$$\boldsymbol{\theta}^i(j+1,k,l)=\boldsymbol{\theta}^i(j,k,l)+\boldsymbol{C}(i)\frac{\boldsymbol{\Delta}(i)}{\sqrt{\boldsymbol{\Delta}(i)^{\mathrm{T}}\boldsymbol{\Delta}(i)}}. \tag{1}$$

(2)趋化过程完毕后,细菌进行繁殖. 细菌种群中代价函数值较差的一半细菌死亡,较好的一半细菌分列成了两个子细菌. 子细菌具有母细菌相同的位置和步长.

(3)完成繁殖操作之后,细菌将以一定概率被驱散到搜索空间中的任意位置. 这个过程称为驱散,驱散过程加强算法的全局搜索能力.

经典的细菌觅食算法,具有良好的全局搜索能力. 但是其收敛速度慢,并且单一步长难以满足多样性需求. 之后提出了自适应步长的细菌觅食算法.

2 自适应步长的细菌觅食算法

Sambarta Dsgpta 等在 2009 年提出了一种自适应步长的细菌觅食算法[12].

$$\boldsymbol{C}(i)=\frac{|J(\boldsymbol{\theta}^i)|}{|J(\boldsymbol{\theta}^i)|+\lambda}. \tag{2}$$

该算法采用线性的策略通过代价函数值 $J(\boldsymbol{\theta}^i)$ 的减小而逐步缩小步长. 其中 $\boldsymbol{C}(i)$ 代表第 i 个细菌的自适应趋化步长,λ 为恒定的参数,针对不同的代价函数 λ 的值也不同. 初始时 $J(\boldsymbol{\theta}^i)$ 偏大 $\boldsymbol{C}(i)\approx 1$,随着趋化过程的进行 $J(\boldsymbol{\theta}^i)$ 逐渐缩小,$\boldsymbol{C}(i)$ 也随之减小. 实验证明[13],自适应步长的细菌觅食算法具有更快的收敛速度并且在一定程度上解决了传统觅食算法步长选择的问题. 但是自适应步长的细菌觅食算法也有它的局限性. 当算法接近最优解时,$J(\boldsymbol{\theta}^i)$ 远远小于 λ,使 $\boldsymbol{C}(i)$ 趋近于 0. 由公式(1)可知,此时 $\boldsymbol{\theta}^i(j+1,k,l)=\boldsymbol{\theta}^i(j,k,l)$,细菌位置不变造成 $J(\boldsymbol{\theta}^i)$ 不再改变,导致算法早熟.

取单峰值函数 Sphere 进行实验,通过对公式(2)设定不同的 λ 值,来得到不同的 $\boldsymbol{C}(i)$ 缩小曲线. 之后通过观察不同 $\boldsymbol{C}(i)$ 缩小曲线对应的算法优化情况,来分析步长对于自适应细菌觅食算法的影响.

$$f(\boldsymbol{X})=\sum_{i=1}^{P}x_i^2,x_i\in[-100,100]^P,P=30.$$

曲线 1 由 $\lambda=100$ 得到,由图 1 可知曲线 1 不仅完成了由大步长逐渐缩小到小步长的目的,而且步长最晚进入极小. 所以图 2 中曲线 1 所对应的一组最优解无论在收敛速度还是精度上都要好于其他四组解.

曲线 2 由 $\lambda=1\ 000$ 得到,曲线 2 的坡度不如曲线 1 平滑但要优于曲线 3 和曲线 4,并且步长也要比曲线 3 和曲线 4 更晚进入极小. 所以曲线 2 所对应的一组最优解在收敛精度和速度上不如曲线 1,但要优于其他三组解.

曲线 3 由 $\lambda=10\ 000$ 得到,曲线 3 的步长在曲线 4 和曲线 5 之后进入极小,所以图 2 中曲线 3 所对应的一组最优解优于曲线 4 和曲线 5.

曲线 4 由 $\lambda=100\ 000$ 得到,曲线 4 步长过早就进入极小,导致算法出现了 $\boldsymbol{\theta}^i(j+1,k,l)=\boldsymbol{\theta}^i(j,k,l)$ 的情况,造成算法早熟.

曲线 5 由 $\lambda=10$ 得到,曲线 5 并没有完成由大步长逐渐缩小到小步长的目的,由于步长过大最终算法

的最优解在 100 附近出现了震荡.

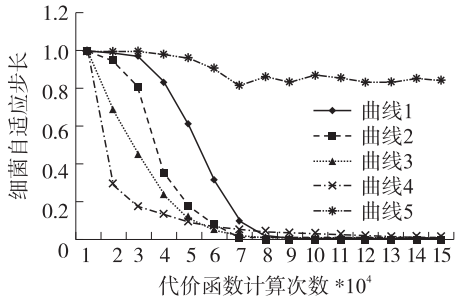

图 1　不同 λ 时步长的变化

Fig. 1　The changes of step length regarding to different λ

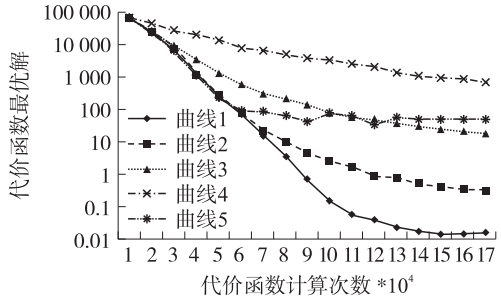

图 2　不同 λ 时最优解的变化

Fig. 2　The changes of optimal solution regarding to different λ

综上所述,虽然自适应觅食算法中 λ 在一定程度上影响了步长的变化,但是改变 λ 的值不能解决步长过早进入极小的问题. 实验中 λ = 100 时结果最优,但最优解优化至 0.01 时,此时步长由公式(2)计算可得 $\boldsymbol{C}(i) = 0.0001$,步长过小导致算法早熟. 由此,我们根据细菌在最优解附近的聚集程度作为阈值,采用 DBSCAN 聚类算法对核心点区域细菌进行标记,再通过新的自适应步长调整函数,降低被标记细菌的步长下降速度,达到优化算法的目的.

3　基于 DBSCAN 聚类的自适应步长细菌觅食算法

3.1　利用 DBSCAN 算法对细菌进行聚类

DBSCAN 聚类算法是一种经典的基于密度的聚类算法. 它通过计算欧氏距离,依据密度大小把一离散空间内的点划分为核心点、边界点和噪声点. 细菌之间的欧式距离计算公式如(3)所示.

$$d(\boldsymbol{\theta}^i,\boldsymbol{\theta}^j) = \| \boldsymbol{\theta}^i - \boldsymbol{\theta}^j \| = \sqrt{\sum_{m=1}^{P} (\boldsymbol{\theta}^i_m - \boldsymbol{\theta}^j_m)^2}\ , \tag{3}$$

其中 $d(\boldsymbol{\theta}^i,\boldsymbol{\theta}^j)$ 代表第 i 个细菌与第 j 个细菌的欧氏距离,P 代表维数,$\boldsymbol{\theta}^i_m$ 代表第 i 个细菌位置的第 m 维的值,$\boldsymbol{\theta}^j_m$ 代表第 j 个细菌位置的第 m 维的值. 核心点为在半径 *Eps* 内含有超过 *MinPts* 数目的点;边界点定义为在该点半径 *Eps* 内点的数目小于 *MinPts*,但与核心点的距离小于 *Eps* 的点;噪声点为任何不是核心点和噪声点的点. 每次迭代过程中在算法完成驱散过程之后对细菌种群进行 DBSCAN 聚类,具体过程如下所示:

(1)取细菌种群中一个未处理的细菌.

(2)如果该点为核心点细菌,找出核心点区域所有边界点细菌同时进行标记. 标记细菌采用函数 4 逐渐缩小细菌步长.

(3)如果该细菌为噪声点,依旧采用函数 2 逐渐缩小细菌步长.

(4)寻找下一个细菌,直到所有细菌点都被处理.

3.2　改进的自适应步长缩小函数

在 DBSCAN 对细菌种群进行聚类之后,我们标记出了核心点区域的细菌. 设改进的自适应步长调整函数如公式(4)所示.

$$\boldsymbol{C}(i) = \frac{e^{|J(\boldsymbol{\theta}^i)|}}{e^{|J(\boldsymbol{\theta}^i)|} + \lambda'}. \tag{4}$$

新的自适应趋化步长调整函数具有更慢的减小速率,并且当代价函数值过小时,减小速度趋近于 0,避免了步长进入极小. 下面通过对步长调整函数进行微分,分析步长变化的速度. 由两种自适应趋化步长调整函数(2)、(4),分别对 $|J(\theta)|$ 求导得到公式(5)和(6):

$$F_1'(|J(\boldsymbol{\theta})|) = \frac{\lambda}{(|J(\boldsymbol{\theta})| + \lambda)^2}, \tag{5}$$

$$F_2'(|J(\boldsymbol{\theta})|) = \frac{\lambda' \times e^{|J(\boldsymbol{\theta})|}}{(e^{|J(\boldsymbol{\theta})|} + \lambda')^2}, \tag{6}$$

对公式(5)和(6)进行比例运算得到公式(7)

$$\frac{F_1'}{F_2'}=\frac{1}{e^{|J(\boldsymbol{\theta})|}}\times\frac{\lambda}{\lambda'}\times\frac{(e^{|J(\boldsymbol{\theta})|}+\lambda')^2}{(|J(\boldsymbol{\theta})|+\lambda)^2},\tag{7}$$

由于算法在进入寻优阶段时,

$$\lambda>>|J(\boldsymbol{\theta})|,\lambda'>>e^{|J(\boldsymbol{\theta})|},$$

故可由公式(7)得到公式(8)

$$\frac{F_1'}{F_2'}=\frac{1}{e^{|J(\boldsymbol{\theta})|}}\times\frac{\lambda'}{\lambda}.\tag{8}$$

算法在由全局搜索阶段进入寻优阶段时,两种步长调整函数所计算出的步长相同,可推出:

$$\frac{|J(\boldsymbol{\theta})|}{|J(\boldsymbol{\theta})|+\lambda}=\frac{e^{|J(\boldsymbol{\theta})|}}{e^{|J(\boldsymbol{\theta})|}+\lambda'},$$

简化上述方程得到公式(9)

$$\frac{\lambda'}{\lambda}=\frac{e^{|J(\boldsymbol{\theta})|}}{|J(\boldsymbol{\theta})|},\tag{9}$$

由方程(9)带入(8)中得到公式(10)

$$\frac{F_1'}{F_2'}=\frac{1}{|J(\boldsymbol{\theta})|}.\tag{10}$$

当$|J(\theta)|$接近最优值时,$|J(\theta)|<1$,

$$\frac{F_1'}{F_2'}>1.$$

由推理可知,在算法进行到寻优阶段之后,新的自适应趋化步长调整函数的减小速率要小于原有的调整策略,并且随着$|J(\boldsymbol{\theta})|$的减小,步长的减小速率也会随之减小. 取单峰值函数 Sphere 进行举例说明,本次对比实验 λ 设为 100:

$$f(\boldsymbol{X})=\sum_{i=1}^{P}x_i^2,x_i\in[-100,100]^P,P=30.$$

从图 3 可以看出原有的步长调整策略下自适应趋化步长持续减小,最后接近 0.000 1. 而新的调整策略下自适应趋化步长的减小速率明显慢于原有策略下的步长,并且当步长减小至 0.01 时,减小速率趋近于 0,停止减小. 由图 4 可知 DBSCAN 聚类的细菌自适应算法解决了步长进入极小的问题,避免了算法出现早熟.

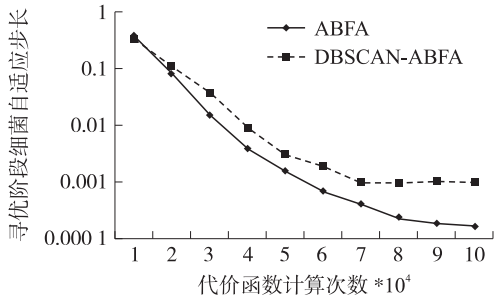

图 3　不同自适应步长函数步长的变化

Fig. 3　The changes of step length regarding to different adaptive functions

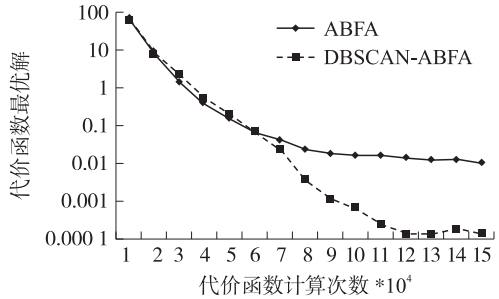

图 4　不同自适应步长函数最优解的变化

Fig. 4　The changes of optimal solution regarding to different adaptive functions

4　DBSCAN-ABFA 算法性能实验分析

本文选取 6 组包含单峰、多峰的经典测试函数,分别对 BFA 算法、ABFA 算法、DBSCAN-ABFA 算法进行 50 组测试,记录最优解,并计算出平均值和标准差. 通过对比,来说明 DBSCAN-ABFA 算法的优越性.

4.1　检测函数

表 1　测试函数信息

Table 1　Information of test functions

函数名	函数表达式	测试范围	最优解
Sphere(f_1)	$f_1(\boldsymbol{X})=x_i^2$	$(-100,100)^P$	$f_1(\mathbf{0})=0$
Axis(f_2)	$f_2(\boldsymbol{X})=\sum_{i=1}^{P} i\times x_i^2$	$(-100,100)^P$	$f_2(\mathbf{0})=0$
Rosenbrock(f_3)	$f_3(\boldsymbol{X})=\sum_{i=1}^{P-1}[100(x_{i+1}-x_i^2)^2+(x_i-1)^2]$	$(-100,100)^P$	$f_3(\mathbf{0})=0$
Griewank(f_4)	$f_4(\boldsymbol{X})=\frac{1}{4\,000}\times\sum_{i=1}^{P}x_i^2-\prod_{i=1}^{P}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	$(-600,600)^P$	$f_4(\mathbf{0})=0$
Six-Hump Camel-Back(f_5)	$f_5(\boldsymbol{X})=4x_1^2-2.1x_1^4+\frac{1}{3}x_1^6+x_2x_1-4x_2^2+4x_2^6+1.031\,628$	$(-5,5)^2$	$f_5(0.089\,83,-0.712\,6)=$ $f_5(-0.089\,83,0.712\,6)=0$
Goldstein-Price(f_6)	$f_6(\boldsymbol{X})=\{1+(x_1+x_2+1)^2(19-14x_2+13x_1^2-14x_2)\}\{30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)\}-3$	$(-2,2)^2$	$f_6(0,-1)=0$

4.2　实验环境设置

算法仿真平台为 Windows 7 Ultimate,处理器为 i3-2130,主频为 3.40 GHz,内存为 3.81G 可用,软件版本为 Microsoft Visual Studio 2010. 实验参数见表 2 所示. ABFA 算法,DBSCAN-ABFA 算法中的基本参数与 BFA 的相同,参数 λ 见表 3.

表 2　实验参数信息

Table 2　Information of experiment parameters

BFA						DBSCAN-ABFA	
S	*Nc*	*Nre*	*Ned*	*Ns*	*Ped*	*Eps*	*MinPts*
100	100	16	4	12	0.25	0.5	60% S

4.3　实验仿真结果

实验结果如表 3 所示. DBSCAN-ABFA 算法在参数相同的情况下对于这 6 组测试函数的不同维数条件下所求得的解都要优于 BFA 算法以及 ABFA 算法.

表 3　实验结果

Table 3　Experimental results

函数名	维数	参数 λ	平均代价函数最优解(标准方差)			函数名	维数	参数 λ	平均代价函数最优解(标准方差)		
			BFA	ABFA	DBSCAN-ABFA				BFA	ABFA	DBSCAN-ABFA
f_1	15	4 000	0.279 7 (0.061 0)	0.012 9 (0.003 1)	$2.243e^{-6}$ ($1.284e^{-6}$)	f_2	15	4 000	1.545 8 (0.348 6)	0.121 1 (0.030 1)	$1.679e^{-4}$ ($3.846e^{-5}$)
	30	4 000	0.820 2 (0.099 0)	0.018 8 (0.003 5)	$7.200e^{-6}$ ($3.390e^{-6}$)		30	4 000	9.881 4 (1.381 2)	0.234 2 (0.072 3)	$2.223e^{-3}$ ($8.010e^{-3}$)
	45	4 000	1.499 3 (0.184 1)	0.030 5 (0.004 2)	$2.833e^{-5}$ ($2.054e^{-5}$)		45	4 000	28.154 9 (3.213 7)	0.454 8 (0.102 3)	0.009 75 (0.057 0)
	60	4 000	2.220 6 (0.285 9)	0.046 8 (0.005 8)	$2.261e^{-4}$ ($2.605e^{-5}$)		60	4 000	50.207 8 (4.267 0)	1.128 5 (0.231 8)	0.013 2 (0.235 1)
f_3	15	4 000	38.626 4 (5.619 8)	11.106 0 (2.373 6)	0.463 7 (0.089 7)	f_4	15	10	0.073 5 (0.011 6)	0.006 48 (0.013 7)	$1.256e^{-5}$ ($1.94e^{-6}$)
	30	4 000	105.992 3 (13.192 5)	26.438 5 (3.205 8)	1.799 3 (0.109 5)		30	10	0.052 9 (0.150 5)	0.036 1 (0.019 5)	$2.35e^{-4}$ ($7.73e^{-5}$)
	45	4 000	187.838 7 (16.838 4)	43.447 8 (4.714 9)	2.923 5 (0.293 6)		45	10	0.269 0 (0.248 5)	0.062 5 (0.041 3)	$7.46e^{-4}$ ($5.11e^{-4}$)
	60	40 000	263.490 1 (21.681 9)	56.624 8 (7.846 2)	3.223 1 (0.353 3)		60	10	25.349 3 (5.311 3)	0.821 0 (0.092 6)	0.001 2 (0.000 64)
f_5	2	10	0.008 4 (0.009 4)	$9.35e^{-4}$ ($2.69e^{-5}$)	$6.30e^{-6}$ ($3.55e^{-6}$)	f_6	2	10	1.763 9 (0.521 7)	$2.32e^{-8}$ ($2.42e^{-8}$)	$1.19e^{-14}$ ($2.13e^{-14}$)

4.4　收敛特性

图 5 给出了 BFA 算法、ABFA 算法、DBSCAN-BFA 算法在优化 6 组测试函数时的收敛曲线图. 其中 f_1、f_2、f_3、f_4 取 60 维的收敛过程,f_5、f_6 取 2 维的收敛过程. 我们可以看出在整个过程中 AFBA 算法和 DBSCAN-

BFA 算法的收敛速度和最优解精度都要优于 BFA 算法. 但当算法进入寻优阶段后 DBSCAN-BFA 算法的收敛速度明显快于 ABFA 算法,并且所找到的最优解也具有更高的精度. 同时,ABFA 算法由于步长进入极小已经出现早熟.

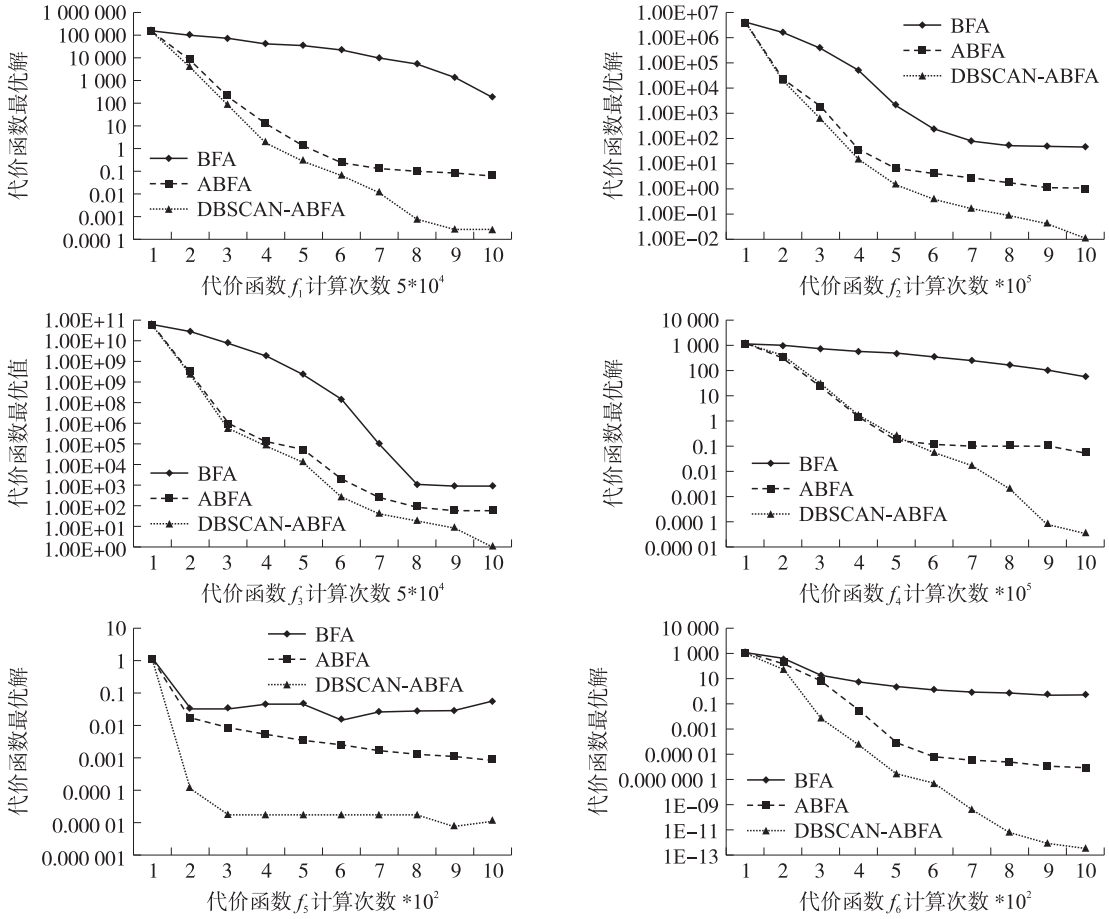

图 5　测试函数收敛曲线

Fig. 5　Convergence curves of test functions

5　结束语

本文首先简单介绍了经典觅食算法,以及自适应步长的觅食算法. 并指出了自适应步长觅食算法在细菌接近最优解时由于步长陷入极小而出现早熟的问题. 在此问题上对其进行改进. 利用 DBSCAN 聚类的方法对在核心点区域的细菌进行标记. 之后通过改变被标记细菌的自适应步长的调整策略,达到减小步长缩小速率的目的,避免了算法陷入早熟. 最后选取 6 组测试函数,通过对比实验验证了 DBSCAN-BFA 算法的有效性和可行性.

[参考文献](References)

[1] Kevin M Passino. Biomimicry of bacterial foraging for distributed optimization and control[J]. IEEE Control Systems Magazine,2002,22:52-67.

[2] Das S,Biswas A,Dasgupta S,et al. Bacterial foraging optimization algorithm:theoretical foundations,analysis,and applications[M]//Foundations of Computational Intelligence,Volume 3. Heidelberg:Springer Berlin,2009,203:919-941.

[3] Liu Y,Kevin M Passion. Biomimicry of social foraging bacteria for distributed optimization:modles,principle,and emergent behaviours[J]. Journal of Optimization Theory and Application,2002,115(3):603-628.

[4] Devi S,Geethanjali M. Application of modified bacterial foraging optimization algorithm for optimal placement and sizing of Distributed Generation[J]. Expert Systems with Applications,2014,41:2 772-2 781.

[5] Santos V S, Felipe P V, Sarduy J G. Bacterial foraging algorithm application for induction motor field estimation under unbalanced voltages[J]. Measurement,2013,46:2 232-2 237.

[6] Niu B,Wang H,Wang J,et al. Multi-objective bacterial foraging optimization[J]. Neurocomputing,2013,116:336-345.

[7] Verma O P,Sharma R,Kumar D. Binarization based image edge detection using bacterial foraging algorithm[J]. Procedia Techology,2012(6):315-323.

[8] Daryabeigi E,Zafari A,Shamshirband S,et al. Calculation of optimal induction heater capacitance based on the smart bacterial foraging algorithm[J]. Electrical Power and Energy Systems,2014,61:326-334.

[9] Shi Y,Eberhart R C. Monitoring of particle swarm optimization[J]. Front Comput Sci China,2009,3(1):31-37.

[10] El-Abd M. Performance assessment of foraging algorithm vs. evolutionary algorithms[J]. Information Sciences,2012,182(1):243-263.

[11] Biswas A, Dasgupa S, Das S, et al. Synergy of PSO and bacterial foraging optimization: a comparative study on numberical benchmarks[C]//Proc 2nd Int Symp Hybrid Aritifical Intell Syst(HAIS). Berlin, Germany: Springer-Verlag, 2007:255-263.

[12] Das S, Biswas A, Dasgupta S, et al. Adaptive computational chemotaxis in bacterial foraging optimization: an analusis[J]. IEEE Transactions on Evolutionary Computation,2009,13(4):919-941.

[13] Sanyal N,Chatterjee A, Munshi S. An adaptive bacterial foraging algorithm for fuzzy entropy based image segmention[J]. Expert Systems with Application,2011,38:15 489-15 498.

[14] Mezura-Montes E, Elyar A Lopez-Davila. Adaptation and local search in the modified bacterial foraging algorithm for constrained optimization[C]//WCCI 2012 IEEE World Congress on Computational Intelligence. Brisbane, Australia, 2012.

[15] Xu R. Survey of clustering algorithm[J]. IEEE Transations on Neural Network,2005,16(3):165-678

[16] Duan L,Xu L,Guo F,et al. A local-density based spatial clustering algorithm with noise[J]. Information Systems,2007,32:978-986.

[责任编辑:顾晓天]