动态高斯变异和随机变异融合的自适应细菌觅食优化算法

张新明 尹欣欣 冯梦清
(河南师范大学计算机与信息工程学院 新乡 453007)

摘 要 针对细菌觅食优化(Bacterial Foraging Optimization,BFO)算法在高维函数优化上性能较差和普适性不强的问题,提出一种动态高斯变异和随机变异融合的自适应细菌觅食优化算法。首先,将原随机迁徙方案修改为动态高斯变异与随机变异融合的迁徙方法,即搜索前期利用随机迁徙有利于增加解的多样性,获得全局最优解,搜索后期改用动态的高斯变异来提高算法的收敛速度;然后,对趋化操作中的步长参数使用动态调整和自适应调整来增强算法的普适性;最后,构建全局极值感应机制使优化更有效,从而获得了一种高性能的自适应BFO算法,以便能够高效解决高维函数的优化问题。14个高维函数优化的仿真结果表明,提出的算法不仅优化效果好、普适性强,而且能以更快的速度找到全局最优解,性能优于SBFO、POLBBO、BFAVP和RABC算法。

关键词 优化方法,细菌觅食优化算法,高斯变异,高维函数优化,动态调整
中图法分类号 TP181 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.6.023

Adaptive Bacterial Foraging Optimization Algorithm Based on Dynamic Gaussian Mutation and Random One for High Dimensional Functions

ZHANG Xin-ming YIN Xin-xin FENG Meng-qing
(College of Computer and Information Engineering, Henan Normal University, Xinxiang 453007, China)

Abstract In view of the shortcomings of facterial foraging optimization (BFO), such as the bad optimization performance and generalization in its application of high dimensional function optimization, an adaptive bacterial foraging optimization algorithm based on combing dynamic Gaussian mutation and random one was proposed in this paper. First, the original elimination-dispersal operator is replaced with a new one based on combining random mutation to add population diversity and dynamical Gaussian mutation to raise convergence rate. Then a chemotactic step mechanism is adopted with dynamical adjusting and self-adapting adjusting. Finally, a new communication mechanism is added to the improved BFO. The simulation results on 14 high-dimensional functions indicate that the proposed optimization algorithm is rapid and has good performance and generalization, and outperforms the current global optimization algorithms such as SBFO, POLBBO, BFAVP and RABC.

Keywords Optimization method, Bacterial foraging optimization (BFO), Gaussian mutation, High dimensional function optimization, Dynamical adjusting

1 引言

群体智能优化算法是一类模拟自然界中生物群体的随机优化算法,该类优化算法通过群体中个体之间的相互协作与竞争实现对问题最优解的搜索。函数优化问题几乎普遍存在于科学和工程等领域的各个分支中,而在这些问题中高维函数优化占有比较大的比例;由于高维函数的搜索空间大,变量耦合强,会使优化算法的全局搜索能力下降,因此高维函数优化问题是群体智能优化算法研究中一个极为重要的问题[1]。细菌觅食优化算法[2]是Passino K M于2002年基于Ecoli大肠杆菌在人体肠道内搜寻食物行为过程中表现出来的群体竞争协作机制而提出的一种新型仿生类群体智能算法。该算法因具有群体智能算法并行搜索、易跳出局部极值等优点,被广泛用于函数优化[3]、预测控制[4]、电气工程与控制[5]及图像处理[6,7]等方面,已经成为生物启发式计算研究领域的又一热点。为克服BFO算法在多维复杂优化问题中其觅食机制容易引起算法早熟收敛,较难获得全局最优解的缺陷[3],一些学者从算法参数上进行了研究:针对趋化步长对算法性能的影响,文献[8]提出用TS模糊方案获取最优步长的模糊BFO算法,文献[9]提出一种随适应度值动态变化的趋化步长控制策略,文献[10]提出用自适应增量调制来控制趋化步长,文献[11]将PSO进化机理引入到BFO趋化操作中,在不增加算

到稿日期:2014-04-06 返修日期:2014-05-23 本文受河南省重点科技攻关项目(132102110209),河南省基础与前沿技术研究计划项目(142300410295)资助。

张新明(1963—),男,教授,硕士生导师,CCF会员,主要研究方向为智能优化算法、模式识别和数字图像处理等,E-mail:xinmingzhang@126.com;尹欣欣(1990—),女,硕士生,主要研究方向为数字图像处理、智能优化算法;冯梦清(1990—),女,硕士生,主要研究方向为数字图像处理、智能优化算法。

法复杂性的前提下，上述方法有效地提高了算法收敛速度；而另外一些学者从算法融合方面进行了研究：根据无免费午餐定理，文献[12-14]分别结合差分进化、分布估计和量子空间下的概率分布模型对 BFO 进行改进，这些混合算法结合各自算法的优点，取得了不错的效果。但是对于多峰高维函数的优化问题，以上改进算法仍仅对部分高维函数有效，表现出普适性不强等缺陷。因此，本文针对 BFO 算法在高维函数优化中存在的问题，提出了一种动态高斯变异和随机变异融合的自适应 BFO 算法。将原随机迁徙机制改为融合动态高斯变异与随机变异的迁徙方法：搜索前期使用随机迁徙有利于增加解的多样性，搜索后期改用动态的高斯变异有利于提高算法的收敛速度；构建全局极值感应方法替换原感应机制以提高优化性能和优化速度；采用动态和自适应混合调整趋化步长的策略更有利于增强普适性，最终获得一种动态高斯变异和随机变异融合的自适应 BFO 算法，以提高高维函数优化性能。

2 标准的细菌觅食优化算法

BFO 算法模拟大量细菌在觅食过程中的一系列活动，通过趋化操作、繁殖操作、迁徙操作及群体感应机制的迭代计算来寻找优化问题的最优解。

1)趋化操作。细菌在向富养区域聚集或躲避有害区域的趋化过程中，首先向任意方向移动单位步长到达新位置，如果新位置具有较好的适应值，则沿着同一方向继续移动若干步，直到适应度值不再改善或达到规定的最大移动步数 N_s。

设细菌数为 N，一个细菌个体所处的位置表示问题的一个候选解，细菌 i 的位置用 D 维向量表示：$x_i=[x_{i1},x_{i2},\cdots,x_{iD}]$，$i=1,2,\cdots,N$。$x_i(j,k,l)$ 表示细菌 i 在第 j 次趋化操作、第 k 次繁殖操作和第 l 次迁徙操作之后的位置。细菌 i 的每一步趋化操作表示如下：

$$x(j+1,k,l)=x(j,k,l)+step(i)\theta(j) \tag{1}$$

其中，$step(i)>0$ 表示向前移动的步长，$\theta(j)$ 表示移动后选择的一个随机前进方向。

2)群体感应机制。细菌个体在移动过程中还会释放信号，以便于周围细菌个体决定是否向自己靠拢；同时，也会收到附近细菌个体发出的排斥力信号，以保持细菌个体与个体之间的安全距离。因此，BFO 算法中的每一个细菌个体寻找食物的决策行为受两个因素的影响：(1)自身的信息，即细菌个体觅食目的，其目的是使个体在单位时间内获取的能量最大；(2)其他细菌个体的信息，即种群中其他细菌传递的觅食信息，这种现象称为感应机制。设 $X(j,k,l)=\{x_i(j,k,l)\mid i=1,2,\cdots,N\}$ 表示种群中个体的位置，$J(i,j,k,l)$ 表示细菌 i 在第 j 次趋化操作、第 k 次繁殖操作和第 l 迁徙操作之后的适应度值，种群细菌之间传递信号的影响值是：

$$\begin{aligned}J_{cc}(x,X(j,k,l))&=\sum_{i=1}^{N}J_{cc}(x,x^i(j,k,l))\\&=\sum_{i=1}^{N}[-d_{attractant}\exp(-w_{attractant}\sum_{n=1}^{D}(x_n-x_i)^2)]+\sum_{i=1}^{N}[h_{repellant}\exp(-w_{repellant}\sum_{n=1}^{D}(x_n-x_i)^2)]\end{aligned} \tag{2}$$

考虑上述两个因素对细菌行为的影响，执行一次趋化操作后细菌 i 的新适应度函数值为：

$$J(i,j+1,k,l)=J(i,j,k,l)+J_{cc}(x(j+1,k,l),X(j+1,k,l)) \tag{3}$$

其中，$d_{attractant}$ 为吸引力的深度，$w_{attractant}$ 为吸引力的宽度，$h_{repellant}$ 为排斥力的高度，$w_{repellant}$ 为排斥力的宽度。

3)繁殖操作。当细菌完成趋化操作后，将细菌个体依照在迭代过程中按式(4)的 J_{sum} 优劣排序，让排在后面的 $N/2$ 个细菌死亡，剩余的 $N/2$ 个细菌进行繁殖，每个细菌分裂成两个子细菌，这个过程称为繁殖操作。在式(4)中，N_c 为趋化次数。

$$J_{sum}(i)=\sum_{j=1}^{N_c}J(i,j,k,l) \tag{4}$$

4)迁徙操作。在算法执行过程中，细菌个体会依据一定的迁徙概率 Ped 突然消亡，并随机产生一个新的细菌个体，使细菌个体的总量不变，这就是迁徙操作。这种操作有利于细菌个体跳出局部最优点，寻找全局最优解。

标准的细菌觅食优化算法就是通过这 4 种基本操作对问题求解，在达到一定的迭代次数或者精度时，算法结束。

3 动态高斯迁徙的自适应细菌觅食优化算法

3.1 全局极值感应机制

从标准的 BFO 算法中可以看出，其群体感应机制虽然在理论上有利于提高优化性能，但也增加了计算复杂度；而且 Tang 等人[15]的仿真实验指出，细菌移动且不进行群体内通信时在收敛速度和精度上反而优于存在群体感应机制的 BFO，这说明标准 BFO 的感应机制不够有效。所以，本文提出一种新的群体感应方案，见式(5)，采用这种新方案替换标准 BFO 算法中的群体感应机制。

$$v_i=x_i+rand(0,1)\times(x_{Best}-x_i) \tag{5}$$

其中，x_{Best} 为当前细菌群体中的最优个体，$rand(0,1)$ 是 0 到 1 之间的随机数。这种新的群体感应机制借助于粒子群算法中的全局极值算子，通过细菌个体所经历的全局最好位置来更新自身位置，使得更新后的个体向着全局最优解的方向逼近，具有向其他个体学习的优点，从而能在较少的迭代次数内找到最优解。这样不仅使优化性能得到提高，而且加快了运行速度。

3.2 动态高斯变异和随机变异融合的迁徙操作

迁徙操作是 BFO 算法中的一个重要部分，类似于遗传算法中的变异操作，但不像遗传算法的变异算子仅仅是对某一位基因以一定概率变异，在当前解的邻域范围内变异，BFO 算法在给定 Ped 下进行完全变异，若种群中某个细菌个体满足迁徙概率，则此细菌个体灭亡，并随机在搜索空间的任意位置生成一个新个体，新个体的基因有可能全面改变。通过迁徙，陷入局部极值的细菌个体可以重新选择位置，从而能够跳出局部极值点，避免早熟收敛。迁徙概率越大，个体重新选择新位置的概率越大，跳出局部极值点的概率就越大。一般来说，迁徙操作使得 BFO 具有随机搜索的能力，有助于 BFO 保持种群的多样性，减少早熟收敛现象的发生，能很好地解决低维优化问题。但对于高维优化问题，由于维数增加，问题的复杂性也随之增加，采用这种随机迁徙操作虽然提高了全局搜索能力，但也使原来已经获得的较优解被破坏，从而得不到最

优解，特别是在搜索后期。针对此问题，本文提出动态高斯变异和随机变异融合的迁徙方法。

高斯分布的概率密度函数如下：

$$f(x)=\frac{1}{\sigma\sqrt{2\pi}}\exp(\frac{(x-\mu)^2}{2\sigma^2}) \tag{6}$$

其中，μ 为均值，σ^2 是方差。那么，高斯变异描述如下：

$$x_{i,j}=x_{i,j}+N_j(0,1) \tag{7}$$

其中，$N_j(0,1)$表示均值 $\mu=0$ 和方差 $\sigma=1$ 的高斯函数。为了进一步提高高斯变异的效果，在搜索前期，高斯变异空间较大，有利于提高解的多样性；而在搜索后期，高斯变异空间小，有利于加快收敛速度和增加解的精度。由此得到动态的高斯变异，其公式如下：

$$x_{i,j}=x_{i,j}+C(k)N_j(0,1) \tag{8}$$

$$C(k)=C_{\max}-(C_{\max}-C_{\min})\times l/N_{ed} \tag{9}$$

为了不降低随机种群的多样性，本文将随机变异与上文叙述的动态高斯变异进行融合，得到混合的动态高斯迁徙方法，即在迁徙前期采用随机迁徙，在迁徙后期采用动态高斯迁徙；而且本文的随机迁徙不是随机地在解空间任意位置生成一个新个体，而是在原个体的基础上对某些维的变量进行随机变异。另外，与标准的 BFO 算法接受所有的变异点不同，在迁徙操作中，仅仅接受迁徙到更优的点，而丢弃迁徙与原解较差的点，以此避免丢失已经找到的最优解。具体迁徙算子的伪代码如下：

```
//动态高斯变异和随机变异融合的迁徙算子
1. for i=1 to N do
2. Set v_i = x_i + rand(0,1)×(x_Best − x_i)//新感应机制
3.   for j=1 to D do
4.     Select a variable v_ij with a probability Ped
5.     if rand (0,1)< Ped then
6.       if g< g_max/2 then
7.         Replace v_ij with a randomly generated variable from its
           range
8.       else
9.         Replace v_ij with dynamical Gauss mutation
10.      end if
11.    end if
12.    end for
13.    Compute the fitness of v_i and decide if x_i is replaced with v_i
       according to the fitness
14. end for
```

其中，g 为当前的迭代次数，$g_{\max}$为最大迭代次数。以上迁徙算子融合了群体全局极值感应机制和新的迁徙操作。

3.3 动态缩进与自适应的趋化操作

在 BFO 算法中，趋向算子使得 BFO 具有局部开采能力，它决定算法的前进方向以及在某一区域搜索的细致程度等，是 BFO 的核心操作，也是设计 BFO 算法时需要重点考虑的部分。细菌趋化操作对应细菌觅食过程中的游动和方向调整策略，决定了 BFO 的收敛性。为了提高细菌趋化行为的效率和种群多样性，针对标准 BFO 趋化操作中固定步长的缺陷，本文提出一种结合动态缩进步长和自适应调整步长的控制策略。将趋化初始步长取变量大的寻优范围，随着趋向迭代运行动态缩进，逐步对趋化步长进行缩小，在保证细菌收敛性的同时拓宽了细菌个体的寻优空间，增强了细菌个体寻优能力。

首先，采用细菌群体中最优解与最差解之差作为步长，因为最优解与最差解在进化的过程中是不断变化的，所以可以达到自适应的目的。搜索初始阶段，最优解与最差解之差一般很大，步长较大，这有利于提高全局搜索能力，避免限于局部最优；搜索结束阶段，最优解 $x_{Best,x}$ 与最差解 x_{Worst} 之差一般很小，即步长减小，这可以加速收敛和提高局部开采能力。自适应调整步长公式为：

$$step(i)=[x_{Best,x}(i)-x_{Worst}(i)] \tag{10}$$

然后，为了进一步增强趋化步长的动态性，在以上自适应的基础上增加动态缩进控制，即：

$$step(i)=[x_{Best,x}(i)-x_{Worst}(i)]\times step1 \tag{11}$$

$$step1=step_{\max}-(step_{\max}-step_{\min})\times l/N_{ed} \tag{12}$$

其中，$step_{\max}$为最大步长，$step_{\min}$为最小步长，N_{ed}为迁徙次数。以上改进的趋化操作更能够在自身邻域中做局部深度搜索，在解的邻域内寻找更优秀的解，增强 BFO 算法的局部搜索能力，提高解的精度，自适应调整也增强了算法的普适性。

3.4 动态高斯迁徙的自适应细菌觅食优化算法的基本流程

动态高斯迁徙的自适应 BFO 算法的基本流程描述如图 1 所示。

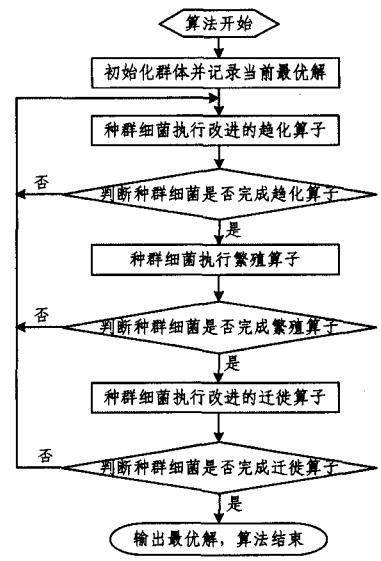

图 1 本文所提算法的流程

①初始化参数，包括繁殖次数 N_{re}，以及 N，N_c，N_s，N_{ed}，$step_{\max}$，$step_{\min}$和 P_{ed}。

②直接使用目标函数值作为适应度值，并依据适应度值记录当前最优解。

③种群进化分为 3 层循环：第一层，外层循环，混合动态高斯变异的迁徙算子；第二层，中层循环，繁殖算子；第三层，动态缩进和自适应调整步长的趋向算子。

④算法结束，输出群体最优解。

4 仿真实验及结果分析

为了验证本文提出的 BFO 算法的有效性，用其进行高维函数优化计算。为了考虑其普适性，特选取 14 个高维函数进行优化比较。比较它们搜索到的全局最优解、优化精度及运行效率等情况。所有实验在 AMD Athlon 64 X2、主频为 2.7G的 CPU 和内存为 1GB DDR RAM 的机器上进行，算法采用 MATLAB R2012A 语言实现，操作系统使用 Windows

XP。为了方便叙述，将标准的细菌觅食算法简称为 SBFO，而本文提出的自适应 BFO 算法简称为 ABFO，所设置的 SBFO 参数和 ABFO 参数如表 1 所列，这些参数是依据大量实验获得最优效果时选取的。

表 1　两种优化算法的参数设置

参数	SBFO	ABFO
N	20	20
N_c	5	1
N_s	4	4
N_{re}	5	1
N_{ed}	100	2500
P_{ed}	0.25	0.01
FES	200500	200500
$d_{attractant}$	0.1	×
$w_{attractant}$	0.2	×
$h_{repellant}$	0.1	×
$w_{repellant}$	10	×
$step_{max}$	×	12
$step_{min}$	×	0.5
C_{max}	×	1.5
C_{min}	×	0.0001
step	见文献[16]	×

标准的 BFO 中的步长 *step* 采用文献[16]的方法：

$$step=1/(1+4000/J(j,k,l)) \tag{13}$$

两种 BFO 算法均随机独立运行 30 次。对于实验 1，选取测试函数运行 30 次中的最好值 *Best*、平均值 *Mean*、最差值 *Worst*、方差 *Std*、实测目标函数的评价次数(*AVG_FES*)及运行时间(*Time*)为评价标准来考察两种算法的寻优性能。对于实验 2，选取测试函数的平均值和方差来考察各算法的寻优性能。为了使比较更具有普适性，这 14 个函数的维数都为 50，代表着不同的情况，如有单峰函数和多峰函数、可分离和不可分离、连续和不连续等，这 14 个目标函数表达式和全局最优解等情况叙述如下：

$$f_{01}=\sum_{i=1}^{n}x_i^2,\ -100\leqslant x_i\leqslant 100 \tag{14}$$

f_{01} 函数为 Sphere 函数，为非线性的对称单峰函数，全局最优点为：$\min(f_{01})=f_{01}(0,0,\cdots,0)=0$。

$$f_{02}=\sum_{i=1}^{n}\left(\sum_{j=1}^{i}x_j\right)^2,\ -100\leqslant x_j\leqslant 100 \tag{15}$$

f_{02} 函数为 Schwefel's Prolem 1.2 函数，为不可分离单峰函数，全局最优点为：$\min(f_{01})=f_{01}(0,0,\cdots,0)=0$。

$$f_{03}=\max_i\{|x_i|,1\leqslant i\leqslant 50\},\ -100\leqslant x_i\leqslant 100 \tag{16}$$

f_{03} 函数为 Schwefel's Prolem 2.21 函数，为不可分离单峰函数，全局最优点为：$\min(f_{03})=f_{03}(0,0,\cdots,0)=0$。

$$f_{04}=\sum_{i=1}^{n}|x_i|+\prod_{i=1}^{n}|x_i|,\ -10\leqslant x_i\leqslant 10 \tag{17}$$

f_{04} 函数为 Schwefel's Prolem 2.22 函数，为不可分离单峰函数，全局最优点为：$\min(f_{04})=f_{04}(0,0,\cdots,0)=0$。

$$f_{05}=-\sum_{i=1}^{n}(x_i\sin\sqrt{|x_i|})+418.982887n,\ -500\leqslant x_i\leqslant 500 \tag{18}$$

f_{05} 函数为 Schwefel's Prolem 2.26 函数，为不可分离的多峰函数，且带有一定的欺骗性，全局最优点为：$\min(f_{05})=f_{05}(420.968746,\cdots)=0$。

$$f_{06}=\sum_{i=1}^{n}(\lfloor x_i+0.5\rfloor)^2,\ -100\leqslant x_i\leqslant 100 \tag{19}$$

f_{06} 函数为 Step 函数，是不连续函数，主要由很多平滑的高地和陡脊组成，是可分离单峰函数，全局最优点为：$\min(f_{06})=f_{06}(0,0,\cdots,0)=0$。

$$f_{07}=\sum_{i=1}^{n}(x_i^2-10\cos(2\pi x_i)+10),\ -100\leqslant x_i\leqslant 100 \tag{20}$$

f_{07} 函数为 Rastrigin 函数，此函数是一个典型的具有大量局部最优点的复杂可分离多峰函数，全局最优点为：$\min(f_{07})=f_{07}(0,0,\cdots,0)=0$。

$$f_{08}=20+e-20\exp\left[-\frac{1}{5}\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right]-\exp\left[\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right] \tag{21}$$

f_{08} 函数为 Ackley 函数，$-32\leqslant x_i\leqslant 32$，是连续、旋转、不可分离的复杂多峰函数，全局最优点为：$\min(f_{08})=f_{08}(0,0,\cdots,0)=0$。

$$f_{09}=1+\sum_{i=1}^{n}\frac{x_i^2}{4000}-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right),\ -600\leqslant x_i\leqslant 600 \tag{22}$$

f_{09} 函数为 Griewank 函数，是不可分离的多峰函数，全局最优点为：$\min(f_{09})=f_{09}(0,0,\cdots,0)=0$。

$$f_{10}=\sum_{i=1}^{n}ix_i^2+random[0,1),\ -1.28\leqslant x_i\leqslant 1.28 \tag{23}$$

f_{10} 函数为 Quartic 函数，是不可分离单峰函数，全局最优点为：$\min(f_{10})=f_{10}(0,0,,0)=0$。

$$f_{11}=\sum_{i=1}^{n}(n+i(1-\cos x_i)-\sin x_i-\sum_{i=1}^{n}\cos x_i)^2 \tag{24}$$

f_{11} 函数为 Trigonometric 函数，$-50\leqslant x_i\leqslant 50$，是不可分离多峰函数，全局最优点为：$\min(f_{11})=f_{11}(0,0,\cdots,0)=0$。

$$f_{12}=\frac{1}{n}\sum_{i=1}^{n}(x_i^4-16x_i^2+5x_i),\ -10\leqslant x_i\leqslant 10 \tag{25}$$

f_{12} 函数为 Himmelblau 函数，为多峰函数，全局最优点为：$\min f_{12}=f_{12}(-2.9043,\cdots,-2.9043)=-78.3323$。

$$f_{13}=\sum_{i=1}^{n}|x_i\sin(x_i)+0.1x_i|,\ -10\leqslant x_i\leqslant 10 \tag{26}$$

f_{13} 函数为 Alpine 函数，为多峰函数，全局最优点为：$\min(f_{13})=f_{13}(0,0,\cdots,0)=0$。

$$f_{14}=10^6x_1^2+\sum_{i=2}^{n}x_i^2,\ -100\leqslant x_i\leqslant 100 \tag{27}$$

f_{14} 函数为 Tablet 函数，为单峰函数，全局最优点为：$\min(f_{14})=f_{14}(0,0,\cdots,0)=0$。

实验 1　SBFO 和 ABFO 优化效果对比。

表 2 列出了两种优化算法对 14 个函数的优化结果、实测目标函数评价次数平均值和运行时间，表中优者用黑体表示。由表 2 数据对比可以看出，在所有的标准测试函数中，ABFO 算法在 f_{01} 至 f_{04}、f_{06}、f_{07}、f_{09} 以及 f_{13} 和 f_{14} 9 个函数上都取得了最好的优化结果，搜索的最好值 *Best*、搜索的平均值 *Mean*、搜索的最差值 *Worst* 和方差 *Std* 都为 0；f_{05}、f_{10}、f_{11} 和 f_{12} 4 个函数稍差，但相比 SBFO 算法，ABFO 算法也具有竞争力的优化性能，其 *Std* 都在 2e−5 以下，*Mean* 都在 5e−4 以下，*Best* 和 *Worst* 反映解的质量，*Mean* 显示在给定的函数评价次数下算法所能达到的精度和反映算法的收敛速度，*Std* 反映算法的稳定性和鲁棒性。因此，这说明无论是解的质量，还是算法的收敛精度和稳定性，ABFO 算法的优化性能大幅度优于 SBFO 算法，14 个函数的优化结果也说明 ABFO 算法具有很好的普适性。对于 f_{05} 函数而言，由于其高维、多峰、复杂，极易发生早熟收敛，一般优化算法优化结果不理想，甚至有些算法干脆不针对其进行比较，而有些算法大都通过增加

迭代次数来提高算法获取全局最优的概率。在 ABFO 算法中，在目标函数评价次数不太多（为 100035）的情况下，获得均值为 6.4800e−04、方差为 3.6982e−06 的好结果，这也说明 ABFO 算法中的动态高斯变异和随机变异融合的迁徙算子有效，能够避免算法限于局部最优找到全局最优解。从实测的目标评价次数的平均值（见表 2 第三列）看，虽然两种算法设置的迭代次数（$Nre\times Ned\times Nc$）都为 2500，但 ABFO 实测的平均值要比 SBFO 多，例如 f_{03} 和 f_{04}，ABFO 实测的 *AVG_FES* 几乎是 SBFO 两倍，其原因是趋向算子的作用。在趋化过程中，细菌先向任意方向移动单位步长到一个新位置，如果新位置有较好的适应度，则沿着同一方向继续移动若干步，直到适应度不再改善或达到规定的最大移动步数。移动的步数越多，目标函数评价次数越多，搜索的细致程度越高，从另一个角度说明 ABFO 算法的开采能力比 SBFO 算法强。理论上 BFO 目标函数的最大评价次数为 $Nre\times Ned\times Nc\times Ns\times N=2500\times 4\times 20=200000$，但实际上不可能达到如此多的次数。由于细菌在趋化过程中不可能每次向任意方向移到的新位置都具有较好的适应度值，即使选择到一个很好的新位置，也不可能每次达到规定的最大移动步数 Ns，这也说明本文提出的动态缩进与自适应的趋向算子是有效的。从运行时间看，在所有的测试函数中，ABFO 算法的耗时都低于 SBFO 算法的耗时，从以上分析可知，ABFO 算法的目标函数评价次数比 SBFO 算法要多。一般来说，目标函数评价次数越多，耗时越多，但 ABFO 算法的耗时少，其原因为 ABFO 算法采用新型的感知方案——全局极值感应方案，这种方案不仅使 ABFO 算法优化效果更好，而且减少了运行时间。而 SBFO 算法虽然 *AVG_FES* 少，但其感知机制计算复杂度高，导致耗时多。

表 2　两种优化算法计算结果以及耗时

Functions	Algorithms	AVG_FES	Time/s	Mean	Std	Best	Worst
f_{01}	SBFO	101819	13.8732	44.0813	2.6400	38.1869	49.7525
	ABFO	106639	**5.7802**	0	0	0	0
f_{02}	SBFO	81155	11.3907	1.3871e+03	3.0916e+02	9.3718e+02	2.2031e+03
	ABFO	105064	**5.8990**	0	0	0	0
f_{03}	SBFO	53286	7.5754	85.8182	3.1493	77.2871	89.8311
	ABFO	113158	**6.0643**	0	0	0	0
f_{04}	SBFO	54864	8.0915	1.2004e+09	6.1325e+09	97.6974	3.3593e+10
	ABFO	111974	**6.3527**	0	0	0	0
f_{05}	SBFO	76282	11.1296	9.3257e+03	7.7839e+02	8.3113e+03	1.0946e+04
	ABFO	100035	**5.9537**	**6.4800e−04**	**3.6982e−06**	**6.4141e−04**	**6.5584e−04**
f_{06}	SBFO	62928	8.8387	1.2015e+04	1.9827e+03	8046	15666
	ABFO	100108	**5.3724**	0	0	0	0
f_{07}	SBFO	66423	9.6773	5.0581e+03	4.2377e+02	4.2680e+03	6.1946e+03
	ABFO	100463	**5.5960**	0	0	0	0
f_{08}	SBFO	53121	8.3634	20.1556	0.3543	19.1696	20.6116
	ABFO	100731	**6.5044**	**−7.6975e−16**	**6.4863e−16**	**−8.8818e−16**	**2.6645e−15**
f_{09}	SBFO	78537	14.2023	8.7427e+02	43.5002	8.0173e+02	9.4291e+02
	ABFO	100430	**8.8928**	0	0	0	0
f_{10}	SBFO	57446	10.6040	1.1512	21.1769	19.6089	24.2640
	ABFO	100063	**9.4942**	**2.2506e−05**	**1.8917e−05**	**1.7113e−06**	**9.8623e−05**
f_{11}	SBFO	76873	17.1469	5.4088	4.7021	0.5614	23.8739
	ABFO	100071	**13.2365**	**4.3332e−05**	**1.1574e−05**	**2.4002e−05**	**7.4469e−05**
f_{12}	SBFO	66298	10.3166	−53.5934	5.3095	−67.1087	−39.8676
	ABFO	100031	**6.7455**	**−78.3323**	**1.5361e−05**	**−78.3323**	**−78.3323**
f_{13}	SBFO	66060	9.4663	35.6309	7.2096	23.1422	52.5227
	ABFO	111902	**6.5896**	0	0	0	0
f_{14}	SBFO	53355	7.6006	1.0286e+05	1.5437e+04	7.2240e+04	1.3599e+05
	ABFO	106691	**5.7759**	0	0	0	0

此实验说明：本文对 SBFO 算法提出的 3 点改进是可行的，提出的 ABFO 算法优化效果显著。

实验 2　ABFO 算法与 RABC 算法[17]、BFAVP 算法[18]和 POLBBO 算法[19]优化效果对比。

表 3 是将本文提出的 ABFO 算法与 RABC 算法、BFAVP 算法和 POLBBO 算法在 10 个函数上进行优化计算的结果。其中，RABC 算法是人工蜂群算法的改进版，POLBBO 算法是生物地理学算法的改进版，BFAVP 算法是细菌觅食算法的改进版。表 3 中这 3 种算法的优化数据分别来自文献[17-19]。相比于实验 1，鉴于比较的可比性，ABFO 算法在函数维数做了调整，即将 D 从 50 调整到 30，N_{ed} 取 2000，其它参数保持不变。另外，仅选择 10 个函数作为测试函数，是因为对比算法的文献中没有 f_{11}、f_{12}、f_{13} 和 f_{14} 的优化数据。从表 3 看出，第一，优化效果（均值 *Mean* 和方差 *Std*）对比：本文提出的 ABFO 算法，在 f_{05} 优化效果稍逊于 RABC 算法和 POLBBO 算法，而其他 9 个函数不亚于 RABC 算法和 POLBBO 算法的优化效果，其中 6 个函数即 f_{01}、f_{02}、f_{03}、f_{04}、f_{08} 和 f_{10} 的 ABFO 算法优化效果优于 RABC 算法，有 7 个函数 f_{01}、f_{02}、f_{03}、f_{04}、f_{07}、f_{08} 和 f_{10} 的 ABFO 算法优化效果优于 POLBBO 算法。与 BFAVP 算法相比，ABFO 算法在 7 个可比较的函数中，除了在 f_{06} 优化效果一样外，而在其他 6 个函数上大幅度优于 BFAVP 算法。第二，从 BFAVP 算法、POLBBO 算法和 RABC 算法的目标函数最大评价次数（见表 3 第二列和第五列）及 ABFO 算法目标函数评价次数（见表 3 倒数第二列）看出，ABFO 算法评价次数少，RABC 算法、BFAVP 算法和 POLBBO 算法的目标函数评价次数是 ABFO 算法的 2 到 10 倍，这说明本文提出的 ABFO 算法收敛速度快，效果明显。

表 3　4 种优化算法的计算结果

Functions	Max_FES	POLBBO	RABC		BFAVP		ABFO
		Mean/Std	Mean/Std	FES	Mean/Std	AVG_FES	Mean/Std
f_{01}	150000	3.67e−70/5.44e−70	9.1e−61/2.1e−60	150000	9.9908e−09/8.7307e−0 9	85941	0/0
f_{02}	500000	3.68e−05/2.42e−05	2.9e−24/1.5e−23	×	×	84760	0/0
f_{03}	500000	1.07e−18/9.41e−19	2.8e−02/1.7e−02	150000	0.1504/0.1221	92159	0/0
f_{04}	200000	1.17e−57/1.16e−57	3.2e−74/2.0e−73	150000	5.4701e−05/3.6527e−05	91041	0/0
f_{05}	500000	0/0	0/0	×	×	80034	1.0492e−06/4.9029e−07
f_{06}	150000	0/0	0/0	150000	0/0	80061	0/0
f_{07}	500000	7.15e+04/9.47e+03	0/0	150000	0.3319/0.3368	80314	0/0
f_{08}	200000	2.66e−15/0	3.8e−14/ 4.4e−15	150000	1.6325e−05/ 2.9589e−05	80563	−8.8818e−16/0
f_{09}	300000	0/0	0/0	150000	2.4649e−02/4.2953e−02	80299	0/0
f_{10}	300000	1.44e−03/2.65e−04	3.6e−02/6.8e−03	×	×	80041	2.3332e−05/1.8331e−05

总之，不管从实验 1 还是从实验 2 来看，本文提出的 ABFO 算法的优化性能都是出色的，在大多数函数上优于 SBFO、RABC 算法、BFAVP 算法和 POLBBO 算法。

结束语　本文针对细菌觅食优化算法在高维函数优化上性能较差和普适性不强的问题，提出一种动态高斯变异和随机变异融合的自适应细菌觅食优化算法。实验结果表明，本文提出的方法是有效的，不仅全局搜索能力和普适性强，而且优化精度高，能很好地应用于高维函数的优化问题中。

参 考 文 献

[1] 张新明，李晓安，何文涛，等. 基于排名映射概率的混沌人工蜂群算法[J]. 计算机科学，2013，40(12)：98-103
Zhang Xin-ming，Li Xiao-an，He Wen-tao，et al. Chaotic artificial bee colony algorithm based on rank mapping probability [J]. Computer Science，2013，40(12)：98-103

[2] Passino K M. Biomimicry of bacterial foraging for distributed optimization and control [J]. IEEE Control Systems Magazine，2002，22(3)：52-67

[3] Chatzis S P，Koukas S. Numerical optimization using synergetic swarms of foraging bacterial populations [J]. Expert Systems with Applications，2011，38(12)：15332- 15343

[4] 王雪松，程玉虎，郝名林. 基于细菌觅食行为的分布估计算法在预测控制中的应用[J]. 电子学报，2010，38(2)：333-339
Wang Xue-song，Cheng Yu-hu，Hao Ming-lin. Estimation of distribution algorithm based on bacterial foraging and its application in predictive control [J]. Acta Electronic Sinica，2010，38(2)：333-339

[5] Saber A Y. Economic dispatch using particle swarm optimization with bacterial foraging effect [J]. Electrical Power and Energy Systems，2012，34(1)：38-46

[6] Verma P O，Hanmandlu M，Kumar P，et al. A novel bacterial foraging technique for edge detection [J]. Pattern Recognition Letters，2011，32(8)：1187-1196

[7] Sathya P D，Kayalvizhi R. Optimal segmentation of brain MRI based on adaptive bacterial foraging algorithm [J]. Neurocomputing，2011，74(3)：2299-2313

[8] Mishra S. A hybrid least square-fuzzy bacteria foraging strategy for harmonic estimation [J]. IEEE Transactions on Evolutionary Computation，2005，9(1)：61-73

[9] Das S，Biswas A，Dasgupta S，et al. Bacterial foraging optimization algorithm：theoretical foundations，analysis，and applications [J]. Foundations of Computational Intelligence，2009，203：23-55

[10] Chen H N，Zhu Y L，Hu K Y. Adaptive bacterial foraging optimization [J]. Abstract and Applied Analysis，2011，2011(1)：1-27

[11] Tang W J，Wu Q H. A bacterial swarming algorithm for global optimization [C]// Proceedings of IEEE Conference on Evolutionary Computation. Singapore，2007：1207-1212

[12] Biswas A，Dasgupta S，Das S，et al. A synergy of differential evolution and bacterial foraging algorithm for global optimization [J]. Neural Network World，2007，17(6)：607-626

[13] 刘小龙，李荣钧，杨萍. 基于高斯分布估计的细菌觅食优化算法[J]. 控制与决策，2011，26(8)：1233-1238
Liu Xiao-long，Li Rong-jun，Yang Ping. Bacterial foraging optimization algorithm based on estimation of distribution [J]. Control and Decision，2011，26(8)：1233-1238

[14] 章国勇，伍永刚，谭宇翔. 一种具有量子行为的细菌觅食优化算法[J]. 电子与信息学报，2013，35(3)：614-621
Zhang Guo-yong，Wu Yong-gang，Tan Yu-xiang. Bacterial foraging optimization algorithm with quantum behavior [J]. Journal of Electronics & Information Technology，2013，35(3)：614-621

[15] Tang W J，Wu Q H，Saunders J R. Bacterial foraging algorithm for dynamic environment[C]// Proceedings of IEEE Conference on Evolutionary Computation. Canada，2006：4467-4473

[16] Dasgupta S，Biswas A，Abraham A，et al. Adaptive computational chemotaxis in bacterial foraging algorithm[C]// Proceedings of International Conference on Complex，Intelligent and Software Intensive Systems. 2008：64-71

[17] Kang F，Li J J，Ma Z Y. Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions [J]. Information Sciences，2011，181(16)：3508-3531

[18] Li M S，Ji T Y，Tang W J，et al. Bacterial foraging algorithm with varying population [J]. BioSystems，2010，100(3)：185-197

[19] Xiong G J，Shi D Y，Duan X Z. Enhancing the performance of biogeography-based optimization using polyphyletic migration operator and orthogonal learning [J]. Computers & Operations Research，2014，41(1)：125-139