Journal of System Simulation

Volume 32 | Issue 9 Article 11

9-18-2020

Bacterial Foraging Algorithm with Gravitational Mechanism

Yitao He
College of Electron and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China;

Li Jun
College of Electron and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China;

Liyan Hao
College of Electron and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China;

Bacterial Foraging Algorithm with Gravitational Mechanism

Abstract

Abstract: Aiming at the weak quorum sensing ability and slow convergence in bacterial foraging algorithm, an algorithm with gravitational mechanism is proposed. *The algorithm provides an optimization direction for each bacterium by introducing a gravitational mechanism in the gravitational search algorithm. The original swimming operation of bacterial foraging algorithm is used to realize the local optimization strategy, and the local dimension update is added after the swimming to widen the bacterial search scope in chemotaxis operation. In the migration operation of the bacterial foraging algorithm, the bimodal Gaussian function is introduced to re-initialize the position of the bacteria to avoid falling into the local extremum for the algorithm and improves the optimization ability.* The experiments prove that the improved bacterial foraging algorithm has better searching ability than the basic algorithm.

Keywords

bacterial foraging algorithm, gravitational search algorithm, local dimension update, Gaussian function, function optimization

Recommended Citation

He Yitao, Li Jun, Hao Liyan. Bacterial Foraging Algorithm with Gravitational Mechanism[J]. Journal of System Simulation, 2020, 32(9): 1724-1735.

第 32 卷第 9 期 系统仿真学报© Vol. 32 No. 9
2020 年 9 月 Journal of System Simulation Sep., 2020

具有引力机制的细菌觅食算法

何奕涛，李珺，郝丽艳
(兰州交通大学电子与信息工程学院，甘肃 兰州 730070)

摘要：针对细菌觅食算法中群体感应能力较弱和算法的收敛速度较慢的问题，提出一种具有引力机制的细菌觅食算法。*该算法通过引入引力搜索算法中的引力机制来为每个细菌提供寻优的方向；采用细菌觅食算法原有的游动操作来实现局部寻优策略，并在游动之后增加局部维度更新，使得细菌在趋化操作中搜索范围更广；在细菌觅食算法的迁徙操作中引入双高斯函数来重新初始化细菌的位置，从而更好得避免算法陷入局部极值，提高算法的寻优能力。*通过实验证明改进后的细菌觅食算法比基本细菌觅食算法拥有更好的寻优能力。

关键词：细菌觅食算法；引力搜索算法；局部维度更新；高斯函数；函数优化
中图分类号：TP301 文献标识码：A 文章编号：1004-731X (2020) 09-1724-12
DOI: 10.16182/j.issn1004731x.joss.19-0096

Bacterial Foraging Algorithm with Gravitational Mechanism

He Yitao, *Li Jun*, *Hao Liyan*
(College of Electron and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730070, China)

Abstract: Aiming at the weak quorum sensing ability and slow convergence in bacterial foraging algorithm, an algorithm with gravitational mechanism is proposed. *The algorithm provides an optimization direction for each bacterium by introducing a gravitational mechanism in the gravitational search algorithm. The original swimming operation of bacterial foraging algorithm is used to realize the local optimization strategy, and the local dimension update is added after the swimming to widen the bacterial search scope in chemotaxis operation. In the migration operation of the bacterial foraging algorithm, the bimodal Gaussian function is introduced to re-initialize the position of the bacteria to avoid falling into the local extremum for the algorithm and improves the optimization ability.* The experiments prove that the improved bacterial foraging algorithm has better searching ability than the basic algorithm.

Keywords: bacterial foraging algorithm; gravitational search algorithm; local dimension update; Gaussian function; function optimization

引言

细菌觅食算法(Bacterial foraging optimization algorithm, BFO)是模仿大肠杆菌趋药性运动的仿生类群算法[1]，目前已广泛应用于比例-积分-微分控制器（Proportion integration differentiation，PID）参数整定、模式识别、图像处理等领域。该算法构造简单，易于实现，在搜索上具有并行性，容易跳出局部极值的优点，因而引起了许多研究者的兴趣。在细菌觅食算法的趋化操作中，由于每个细菌游动的方向都是随机产生的，个体在整个寻优空间盲目搜

收稿日期：2019-03-07 修回日期：2019-05-23；
基金项目：甘肃省自然科学基金(1606RJZA033)，甘肃省教育厅科研项目(1204-13)，甘肃省科技计划项目(1506RJZA084)，甘肃省教育科学‘十二五’规划课题(GS[2015]GHB0907)；
作者简介：何奕涛(1995-)，男，湖南永兴，硕士，研究方向为智能计算。

索，无任何的导向性。Tang 等[2]的仿真实验指出 BFO 的群体感应机制会干扰细菌觅食算法的收敛性，文献[3]通过大量的实验测试发现，当群体中有较多的个体分布在局部极值附近时，已经找到全局最优解的个体，也会因为受到群体感应机制的作用而陷入到局部最优中，这也造成细菌觅食算法在求解高维问题或者复杂函数时，无法快速收敛获取函数的全局最优解。在迁徙操作中只是通过重新初始化细菌个体的位置来帮助算法跳出全局最优，这种方式可能会导致重新初始化的细菌仍然停留在局部最优解的附近，无法达到应有的效果。

针对上述问题，很多学者都提出了解决方案。Mishra 等[4]在一种模糊细菌觅食算法(Fuzzy bacterial foraging，FBF)中提出对细菌觅食算法步长的改进，之后，又有许多学者对细菌觅食算法中的步长进行研究。Niu 等[5]通过验证步长对全局收敛性的影响，提出了非线性递减和指数非线性递减步长，刘珍等[6]在一种改进的细菌觅食优化算法中提出基于非线性递减的余弦自适应步长，Tan 等[7]根据细菌当前的最优位置、历史的最优位置和种群中所有细菌的平均位置来计算当前细菌趋化的步长。为进一步提高算法的性能，学者们将基本细菌觅食算法开始和别的元启发式算法进行结合，提出混合细菌觅食算法。周文宏等[8]将细菌觅食算法与粒子群算法进行结合，提出变概率混合细菌觅食优化算法。刘小龙等[9]将免疫算法中的克隆选择思想引入到细菌觅食算法中，杜鹏桢等[10]根据人工蜂群算法设计出一种雇佣蜂式趋化方式。为了提高算法的全局寻优能力，章国勇等[11]将量子力学中的量子势能阱引入到细菌觅食算法中，在一定程度上提高细菌觅食算法寻优能力；刘璐等[12]设计出非线性动态自适应旋转角继续对量子细菌觅食算法中趋化操作进行改进，进一步提高算法的收敛速度。从现有的细菌觅食算法相关文献可以看出，大多数改进方案都是停留在对算法步长的改进，没有综合的考虑整个种群中细菌个体之间的关系对寻优过程产生的影响。量子细菌觅食算法虽然相对于基本细菌觅食算法而言，它的求解性能得到一定的改善，但是随着维数的增加，算法的求解能力会变得较弱。

本文提出具有引力机制的细菌觅食算法(Bacterial foraging optimization algorithm with gravitational search algorithm，GSABFO)，借助于引力搜索算法中的引力机制，为每个细菌都赋予惯性质量，通过细菌的惯性质量，计算细菌之间的引力，根据细菌之间引力关系给出细菌在趋化操作中确定的游动方向。在细菌结束游动之后，设计局部维度更新操作，对游动之后的细菌进行更新，使得细菌在维度增加后，也能够保持较好的寻优能力。在迁徙操作中引入双高斯函数来重新初始化个体位置，进一步加强种群的多样性。最后，通过基准函数测试 GSABFO 的性能，并将测试结果与粒子群算法(Particle swarm optimization，PSO)、引力搜索算法(Gravitational search algorithm，GSA)、GSABFO 派生算法及其他改进算法进行了对比，实验结果证明 GSABFO 可以获得较好的最优解并且具有较快的收敛速度。

1 基本细菌觅食算法

细菌觅食算法主要由趋化操作、复制、迁徙 3 个部分组成。算法流程如图 1 所示。

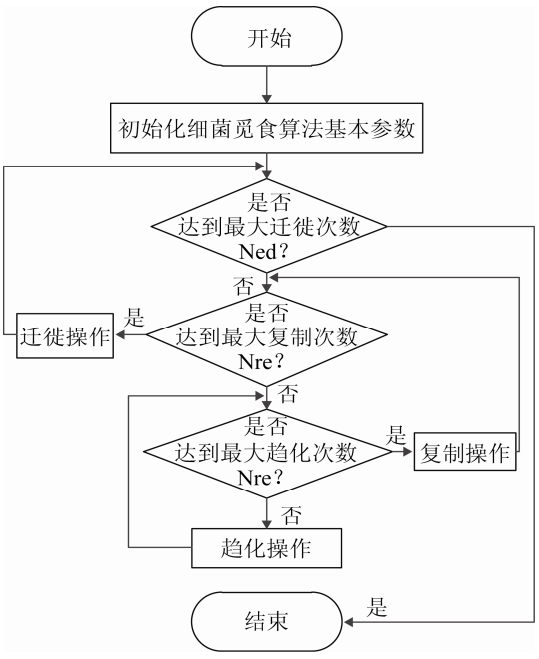

图 1 基本细菌觅食算法流程图

Fig. 1 Flow chart of bacterial foraging algorithm

DOI: 10.16182/j.issn1004731x.joss.19-0096

第 32 卷第 9 期 系统仿真学报 Vol. 32 No. 9
2020 年 9 月 Journal of System Simulation Sep., 2020

该算法主要通过趋化操作更新细菌个体的位置进行寻优，通过复制操作加速算法的收敛速度，用迁徙操作防止算法陷入局部最优。在趋化操作中，其主要步骤为游动和旋转。在进行趋化操作时，细菌会随机产生一个方向，然后进行移动，如果该方向上的适应度更好，则细菌发生游动，否则细菌继续停留在原地。该过程的具体数学描述如公式(1)~(2)所示。

$$\Phi(i)=\frac{\Delta}{\sqrt{\Delta\times\Delta^T}} \tag{1}$$

$$\theta_i(j+1,k,l)=\theta_i(j,k,l)+C(i)\times\Phi(i) \tag{2}$$

式中：Δ为随机向量，其每一个元素取值范围在(–1,1)之间；Φ(*i*)为细菌趋化过程中的方向；$\theta_i(j,k,l)$为细菌 *i* 在 *j* 次趋化 *k* 次复制 *l* 次迁徙后的位置；*C*(*i*)为游动的步长。在复制操作中，设有 *N* 个细菌，发生趋化操作的次数为 *Nc* 次，将细菌 *i* 在发生 *Nc* 次趋化操作中的所有适应度值进行叠加，然后再将这 N 个细菌根据前面叠加的 *Nc* 次趋化操作的适应度进行排序，将排在前面的 *N*/2 个细菌的位置复制给后面的 *N*/2 个细菌。在迁徙操作中，为每个细菌个体生成一个随机数 *rand*，若该细菌的 *rand* 值小于迁徙阈值 *Ped*，则该细菌发生迁徙，在整个解空间中重新随机初始化位置。

2 引力搜索算法

引力搜索算法是伊朗克曼大学教授 Esmat Rashedi 等[13]在 2009 年提出来的，该算法实现简单，控制参数较少，收敛性较快。目前已经有学者证明算法的收敛性优于粒子群算法。引力搜索算法主要是通过模拟自然界的万有引力作用，算法中，所有粒子都被赋予一种惯性质量，粒子的适应度越好，惯性质量越大，粒子作用在其他粒子上的引力越大，产生在该方向上的加速度越大，因而所有的粒子都向着惯性质量越大的粒子运动。引力搜索算法通过粒子之间的引力来实现优良信息的共享[14]。

引力搜索算法的主要流程为：

step 1：初始化种群中粒子的位置 X_i，$i\in\{1,2,\ldots,N\}$，粒子的初始速度 V_i，引力参数 *G*，参数 α，算法最大迭代次数 max_*it*。

step 2：计算当前种群的适应度值 fit_i，获取当前种群中的最好适应度 best 和最差适应度 worst，并且通过公式 $m_i=(fit_i-\text{worst})/(\text{best}-\text{worst})$，$M_i=m_i\Big/\sum_{i=1}^{N}m_i$ 计算出粒子惯性质量 M_i。

step 3：通过公式 $F_{ij}^d=G\times\frac{M_i\times M_j(x_i^d-x_j^d)}{R+\varepsilon}$，$F_i^d=\sum_{j=1,i\neq j}^{N}rand\times F_{ij}^d$ 计算出每个粒子在 *d* 维所受其他粒子的合力，*rand* 取值为(0, 1)。(文中出现的 *rand* 的取值与此处相同)

step 4：通过公式 $a_i^d=F_i^d/M_i$ 计算粒子的加速度。

step 5：更新粒子的速度 V_i，$V_i^d(t)=rand\times V_i^d(t)+a_i^d$，更新粒子的位置 P_i，$P_i^d(t+1)=P_i^d(t)+V_i^d$，更新引力参数 *G*，$G=G\times\exp(\frac{-\alpha\times t}{\max\_it})$。

step 6：重复步骤 step 2~5 直到满足最大迭代次数 max_*it*。

3 具有引力机制的细菌觅食算法

3.1 趋化操作改进

基本的细菌觅食算法是通过细菌的游动来更新位置，随机初始化细菌的游动方向来搜索整个解空间，若游动的方向上适应度更好，则细菌继续向该方向上游动，直到游动到最大步数；否则细菌在原地旋转。通过这种搜索策略使得细菌觅食算法拥有较强的局部搜索能力。但是，随机生成细菌游动的方向必定会导致整个算法需要较多的游动次数才能使得算法获得较优的解，进而减缓了算法的收敛速度。

为了加速细菌觅食算法的收敛速度，本文引入了引力搜索算法中的引力机制。为 BFO 中的每个

细菌赋予惯性质量，然后计算细菌之间的引力和它们之间产生的加速度，用产生的加速度来代替细菌个体在游动过程中的随机方向。在引力的作用下，细菌会向惯性质量较大的个体移动，而惯性质量大的细菌占据着适应度较优的位置。通过个体之间的引力来共享细菌适应度较优位置，根据细菌觅食算法原有的游动操作来进行局部的寻优。这种操作加速了算法的收敛速度，同时又不破坏原有算法的局部开采能力。

通过细菌觅食算法和引力细菌觅食算法对函数 $f(x_1,x_2)=x_1^2+x_2^2$ 寻优的过程，说明在具有引力机制的细菌觅食算法和基本细菌觅食算法中，细菌个体在寻优空间中的移动情况。从图 2 可以看出，在没有引力机制的情况下，细菌 B2 处于一种随机游走的状态。图 2 中，由于可以通过个体之间的引力传递适应度较优的方向，细菌 B2 在受到种群中细菌 B1，B3，B4 的引力影响之后(图 3 中虚线表示细菌 B2 受到其他细菌引力的合力方向)，移动方向有向着当前种群最优个体 B4 的位置移动的趋势。

给出趋化操作的具体改进过程。细菌种群规模为 N，每个细菌拥有 D 维的寻优空间。细菌的初始位置为 $\theta_i^d(j,k,l)$，其含义为第 i 个细菌的第 d 维在第 j 次趋向操作第 k 次复制操作第 l 次迁徙操作后的位置，可以计算当前细菌的适应度为：

$$J_i(j,k,l)=fitness(\theta_i^1(j,k,l),\theta_i^2(j,k,l),\ldots\theta_i^D(j,k,l))$$

其中，d 的取值为 $d\in\{1,2,\ldots D\}$，有了细菌的适应度，可以定义当前种群中细菌的惯性质量 $M_i(j,k,l)$为：

$$m_i(j,k,l)=\frac{J_i(j,k,l)-\text{worst}(j,k,l)}{\text{best}(j,k,l)-\text{worst}(j,k,l)} \tag{3}$$

$$M_i(j,k,l)=\frac{m_i(j,k,l)}{\sum_{i=1}^{N}m_i(j,k,l)} \tag{4}$$

式中：best(j, k, l)和 worst(j, k, l)为当前种群中适应度最好的个体和当前种群中适应度最差的个体。

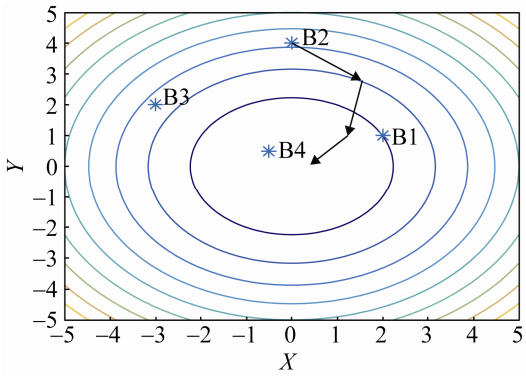

图 2 基本细菌觅食算法细菌游动情况
Fig. 2 BFO algorithm bacterial swimming

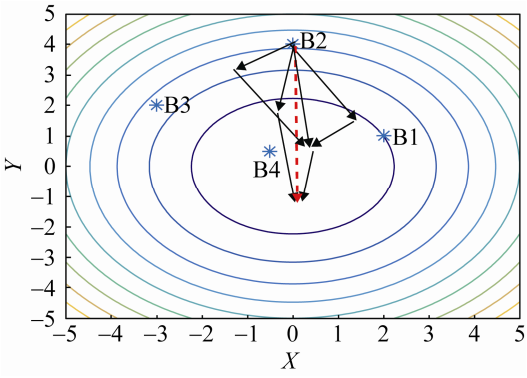

图 3 GSABFO 算法细菌游动情况
Fig.3 GSABFO algorithm bacterial swimming

在搜索目标函数最小值问题时，当前种群中适应度最好的细菌个体和最差个体分别为：

$$\text{best}(j,k,l)=\min\{J_1(j,k,l),J_2(j,k,l),\ldots J_N(j,k,l)\}$$

$$\text{worst}(j,k,l)=\max\{J_1(j,k,l),J_2(j,k,l),\ldots J_N(j,k,l)\}$$

在定义惯性质量之后，细菌游动的方向 $\Phi(i)$ 将不再由公式(1)随机生成。可以计算 2 个细菌之间在第 d 维上的引力 $F_{if}^d(j,k,l)$。在标准的引力搜索算法中计算 2 个粒子之间引力的时候，需要更新参数 G，但是文献[15]中通过大量实验对引力搜索算法中的参数 G 进行研究，在其实验结果中指出，参数 G 在整个算法中充当的角色为粒子移动的步长，由于细菌觅食算法已存在步长，因而在 GSABFO 中我们不引入参数 G。这样细菌之间的引力公式定义如式(5)所示：

$$F_{if}^d=\frac{M_i(j,k,l)\times M_f(j,k,l)\times(\theta_i^d(j,k,l)-\theta_f^d(j,k,l))}{R_{if}(j,k,l)+\varepsilon} \tag{5}$$

https://dc-china-simulation.researchcommons.org/journal/vol32/iss9/11
DOI: 10.16182/j.issn1004731x.joss.19-0096

第 32 卷第 9 期 系统仿真学报 Vol. 32 No. 9
2020 年 9 月 Journal of System Simulation Sep., 2020

式中：$R_{if}(j, k, l)$为细菌 i 和细菌 f 之间的距离，一般取细菌 i 和细菌 f 之间的欧式距离；ε 为指非常小的常数。有了 2 个细菌之间的引力的计算公式，可以计算出细菌 i 的第 d 维在当前种群中受到其他细菌的引力之和 $F_i^d(j,k,l)$，具体公式见式(6)：

$$F_i^d(j,k,l)=\sum_{f\neq i,f=1}^{d} rand\times F_{if}^d(j,k,l) \tag{6}$$

进一步可以得到细菌 i 在第 d 维的加速度公式：

$$a_i^d=\frac{F_i^d(j,k,l)}{M_i(j,k,l)} \tag{7}$$

综合公式(5)~(7)可以得到一个更简单的加速度计算公式。

$$a_i^d(j,k,l)=\sum_{f\neq i,f=1}^{N}\left(\frac{M_f(j,k,l)}{R_{if}}\times[rand\times(\theta_i^d(j,k,l)-\theta_f^d(j,k,l))]\right) \tag{8}$$

这时用计算得到的 $a_i(j,k,l)=(a_i^1(j,k,l), a^2(j,k,l),\ldots a_i^D(j,k,l))$ 来代替原来的随机方向 $\Phi(i)$。引入引力机制后细菌的位置更新公式如式(9)所示。

$$\theta_i^d(j+1,k,l)=\theta_i^d(j,k,l)+C(i)\times a_i^d \tag{9}$$

3.2 局部维度更新

通过在细菌觅食算法中引入引力机制，细菌种群的整体游动趋势开始朝着最优解的方向运动。但是随着迭代次数的增加，细菌种群的多样性不断的下降，到了算法后期，细菌之间的引力作用将只会受到种群中最优细菌个体的影响，因而算法容易陷入到局部最优中。求解高维多峰函数时，在种群向着最优解方向移动的过程中，有些维度会向着最优解方向移动，但是有些维度却会朝着相反的方向移动，这种现象被称为“前进两步，退一步”[16]，这也造成了一些高维多峰函数很难优化。为了不完全破坏引力机制搜索得到的最优解，同时考虑高维多峰函数优化问题中的“前进两步，退一步”的现象，我们在细菌游动之后，对当前细菌进行局部维度更新。更新过程如下：

(1) 在当前细菌个体上随机选取第 P 维，作为局部维度更新的起始位置，为了确保发生局部更新的维度长度为 L，起始点 P 应满足.$P\leqslant D-L+1$.(D 为维度长度，L 为待更新维度长度)。

(2) 将细菌 i 原来的位置向量 $\theta_i(j,k,l)$从第 P 维到 $P+L-1$ 维用向量 $\theta_{best}(j,k,l)-\theta_{wrst}(j,k,l)$相应位置的维度值进行替换，其中，$\theta_{best}(j,k,l)$.为当前种群中适应度最好细菌的位置，$\theta_{wrst}(j,k,l)$为当前种群中适应度最差细菌的位置，替换得到新位置向量为 $\theta_{new}(j,k,l)$。

(3) 如果新产生的 $\theta_{new}(j,k,l)$比 $\theta_i(j,k,l)$的适应度值更好，则用 $\theta_{new}(j,k,l)$去替换 $\theta_i(j,k,l)$,否则，维持细菌位置 $\theta_i(j,k,l)$不变。

3.3 迁徙操作改进

为了维护种群的多样性并且帮助个体跳出局部最优值，细菌觅食算法随机的为每一个细菌赋予一个迁徙概率 $rand$，设定迁徙的阈值为 Ped，当种群中的某一个细菌的迁徙概率 $rand$ 小于迁徙阈值时，就认为这个细菌已经死亡了，需要在寻优空间中为当前死亡的细菌重新初始化位置。然而，这种迁徙方式，不一定能使处于局部最优解附近的个体跳出局部极值，随机的初始化也有一定概率导致初始化的解又在其原来的位置附近，不利于细菌种群多样性的增加。为了避免这种现象的发生，本文引入双高斯函数，使得新生成的细菌位置远离原位置[17-18]。

以要发生迁徙操作的细菌个体 P 为例，说明双高斯函数初始化与随机初始化细菌位置的不同。P 点原有位置为 P(1,1)，迁徙范围为[–8,8]，观察其采用随机方式迁徙 100 次和采用双高斯函数迁徙 100 次后，位置的不同。图 4 中红色圆圈表示要迁徙的点 P，蓝色三角形表示随机迁徙后的细菌个体位置，绿色星号表示双高斯函数迁徙后的细菌个体位置。可以看出，采用双高斯函数迁徙的细菌初始化在原来细菌 P 位置附近的个体

十分少，大部分细菌位置都远离细菌 P；而采用随机初始化方式，细菌在整个解空间中都有分布，并且还是有许多的个体初始化在距离细菌 P 位置不远处，不利于种群多样性的增加。采用双高斯函数方式完成迁徙操作，种群具有更高的多样性，能够增强算法的全局寻优能力。

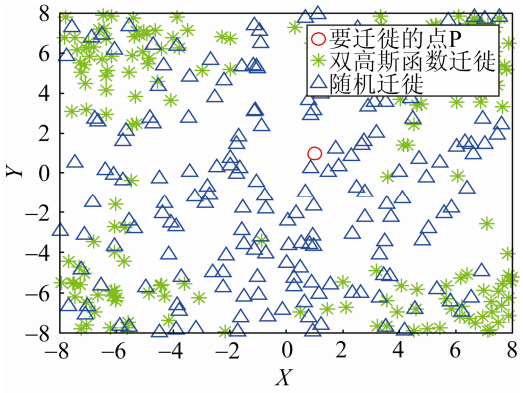

图 4 随机迁徙和双高斯函数迁徙

Fig. 4 Random migration and double Gaussian migration

新生成的细菌位置公式如公式(10)所示。

$$\theta_i^d(j,k,l+1)=(low^d+\sigma_1\times e)\times H(r_1-r)+(up^d+\sigma_2\times e)\times H(r-r_1) \quad (10)$$

式中：$r=\dfrac{\theta_i^d(j,k,l)-\text{low}^d}{\text{up}^d-\text{low}^d}$；$\sigma_1$,$\sigma_2$ 为高斯函数的方差；up^d, low^d 为求解空间的第 d 维的上下界；e 为标准正态分布的随机数；r_1 为[0,1]之间的随机数；$H(x)$为一个阶跃函数，其定义如公式(11)所示。

$$H(x)=\begin{cases}0 & x<0\\ 0.5 & x=0\\ 1 & x>0\end{cases} \quad (11)$$

3.4 GSABFO 算法流程

GSABFO 算法流程如下，GSABFO 伪代码为：

初始化，细菌的个数 N，趋化次数 Nc，复制次数 Nre，

迁徙次数 Ned，游动最大步长 Ns，迁徙概率 Ped，细菌

游动的步长 C，局部维度更新长度 L，细菌位置。

for l=1：Ned //迁徙循环开始

for k=1：Nre //复制循环开始

for j=1：Nc //趋化循环开始

计算当前种群中细菌的适应度

根据 3.1 节中的公式(9)计算出每个细菌的加速度

for i=1：N

保持细菌的适应度值 $Jlast$

采用 3.1 中的公式(10)更新细菌 i 的位置

//游动操作开始

m=0;

while m<Ns

m=m+1;

if $J(i,j,k,l)$<$Jlast$

$Jlast$=$J(i,j,k,l)$;

采用 3.1 节中的公式(10)继续更新细菌的位置

else

m=Ns;

end

end //游动操作结束

根据 3.2 节的描述进行局部维度更新操作

end

end //趋化循环结束

//复制操作

根据目标函数值升序排列细菌的位置

将排序在前面的 N/2 个细菌的位置复制给后面的 N/2 细菌

end //复制操作结束

//迁徙操作

for i=1：N

if $rand$<Ped

细菌 i 按照 3.3 节中的公式(10)重新初始化其位置

else

保持原细菌的位置不变

end

DOI: 10.16182/j.issn1004731x.joss.19-0096

第 32 卷第 9 期 系统仿真学报 Vol. 32 No. 9
2020 年 9 月 Journal of System Simulation Sep., 2020

end

end //迁徙操作结束

4 实验结果及分析

4.1 实验环境及参数设置

本文选取 10 个标准函数对 GSABFO 算法进行测试，并将测试结果与标准细菌觅食算法(BFO)、标准粒子群算法(PSO)、引力搜索算法(GSA)以及近期提出的一些改进智能算法 LAQBFO, QBFO, NAQBFO, APSO-IV 进行对比。LAQBFO, QBFO 和 NAQBFO 实验数据来自文献[12]，APSO-IV 实验数据来自文献[8]，这 10 个测试函数的维度均设定为 30 维，最优值均为 0，如表 1 所示。所有的仿真实验都是在 windows7，Intel Core i5 CPU，主频 1.6 GHZ 的机器上进行，实验采用的软件为 Matlab2016a。对于标准的细菌觅食算法和 GSABFO 算法主要参数设置为：细菌个数 N=50，迁徙次数 Ned=4，复制次数 Nre=3，趋化次数 Nc=100，游动次数 m=5，迁徙概率 Ped=0.25，游动步长 C=0.01 (ub–lb)，其中 ub 表示优化函数维度的上界，lb 表示优化函数维度的下界，对于 GSABFO 算法还需要单独设置局部维度更新参数 L，其值设置为：L=3。对于粒子群算法其参数设置为：粒子个数 N=50，学习因子 c_1=2，c_2=2，权重因子 w_1=0.9，w_2=0.1，迭代次数 $iter$_max=1 200。引力搜索算法的参数设置为：种群粒子个数 N=50，参数 G=100，α=20，迭代次数 $iter$_max=1 200。

表 1 测试函数
Tab. 1 Test function

函数表达式	维数	取值范围	最优解
$f_1(x)=\sum_{i=1}^{n}x_i^2$	30	[–100,100]	0
$f_2(x)=\sum_{i=1}^{n}\lvert x_i\rvert+\prod_{i=1}^{n}\lvert x_i\rvert$	30	[–10,10]	0
$f_3(x)=\sum_{i=1}^{n}(\sum_{j=1}^{i}x_j)^2$	30	[–100,100]	0
$f_4(x)=\max_i\{\lvert x_i\rvert,1\leqslant i\leqslant n\}$	30	[–100,100]	0
$f_5(x)=\sum_{i=1}^{n-1}[100(x_{i+1}-x_i^2)^2+(x_i-1)^2]$	30	[–30,30]	0
$f_6(x)=\sum_{i=1}^{n}ix_i^4+random[0,1)$	30	[–1.28,1.28]	0
$f_7(x)=1+\sum_{i=1}^{n}(\frac{x_i^2}{4000})-\prod_{i=1}^{n}(\cos(\frac{x_i}{\sqrt{i}}))$	30	[–600,600]	0
$f_8(x)=10n+\sum_{i=1}^{n}(x_i^2-10\cos(2\pi x_i))$	30	[–5.12,5.12]	0
$f_9(x)=-20\exp(0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2})-\exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i))+20+e$	30	[–32,32]	0
$f_{10}(x)=418.9829n+\sum_{i=1}^{n}-x_i\sin(\sqrt{\lvert x_i\rvert})$	30	[–500,500]	0

4.2 实验结果

表 2~4 中结果为算法运行 20 次后所得的最好结果，括号内的值为每个基准函数 20 次实验所得结果的平均值。

表 2 GSABFO 算法与基本算法以及 LAQBFO、QBFO、NAQBFO、APSO-IV 算法对比

Tab. 2 Comprision of GSABFO algorithm, basic algorithm and LAQBFO, QBFO, NAQBFO, APSO-IV algorithm

函数	BFO	GSA	PSO	LAQBFO	QBFO	NAQBFO	APSO-IV	GSABFO
F1	2.37E+01 (2.82E+0)	1.25E-16 (2.33E-16)	3.19E-13 (8.80E-12)	--	--	--	1.3116E-02	2.36E-06 (1.98E-05)
F2	4.51E+00 (9.61E+00)	3.53E-08 (4.37E-08)	5.21E-07 (1.16E-05)	--	--	--	2.1745E-02	5.80E-04 (1.09E-03)
F3	2.46E+02 (2.84E+02)	3.01E+02 (4.47E+02)	4.78E+00 (8.32E+00)	--	--	--	9.8422E-02	6.59E-02 (1.11E-01)
F4	5.32E+00 (6.00E+00)	7.87E-02 (6.18E-01)	2.96E-01 (3.95E-01)	--	--	--	9.0489E-03	1.12E-01 (6.10E-01)
F5	4.27E+02 (4.37E+02)	2.73E+01 (2.75E+01)	1.28E+01 (5.46E+01)	2.64E+01	1.21E+02	2.61E+01	--	1.14E+00 (6.15E+00)
F6	4.47E-01 (5.61E-01)	2.32E-03 (8.10E-03)	3.44E-02 (4.48E-02)	--	--	--	8.03E-01	2.14E-03 (2.81E-03)
F7	1.27E+00 (1.29E+00)	1.62E+01 (1.99E+01)	3.21E-14 (1.23E-02)	1.17E-01	1.13E+00	1.03E-01	9.30E+00	2.48E-06 (2.59E-02)
F8	1.21E+02 (1.35E+02)	5.97E+00 (1.05E+01)	3.58E+01 (4.69E+01)	3.54E+01	1.02E+02	2.10E+01	--	4.56E-02 (2.47E-01)
F9	1.66E+01 (1.73E+01)	7.05E-09 (8.05E-09)	9.55E-07 (2.72E-06)	1.60E+00	1.11E+01	1.59E-01	6.61E+00	7.34E-04 (4.55E-01)
F10	4.06E+03 (4.25E+03)	8.96E+03 (9.93E+03)	5.73E+03 (6.25E+03)	9.27E+03	1.33E+04	1.74E+03	--	1.10E-03 (1.01E-02)

表 3 测试函数在 100 维时的优化结果

Tab. 3 Optimization results of test function in 100 dimensions

函数	BFO	ABC	CSO	HJCSO	BAABC	SGHS	GSABFO
F1	1.60E+02	--	8.34E+02	2.03E+00	2.90E-04	1.30E+00	3.30E-05
F2	2.92E+02	--	--	--	--	6.33E+00	3.30E-03
F3	1.64E+04	--	--	--	--	2.84E+04	4.79E+00
F5	2.59E+03	6.80E+09	3.47E+00	9.42E-01	2.20E+00	--	3.20E-01
F6	3.68E+00	--	--	--	4.20E-02	--	2.23E-02
F7	2.52E+00	3.40E+01	4.21E-01	8.17E-05	1.30E-05	5.14E-01	5.20E-02
F8	5.60E+02	2.24E+02	4.23E+02	5.93E-01		4.84E+01	2.30E-03
F9	1.95E+01	1.63E+01	6.56E+00	8.82E-02	6.10E-03	1.71E-01	4.90E-03
F10	1.60E+04	2.55E+00	--	--	--	--	1.30E-03

从表 2 中的实验结果可以看出 GSABFO 算法在 10 个基准函数上的寻优结果基本上可以接近于目标函数的全局最优解。在收敛精度上，对所有的基准函数，GSABFO 算法的收敛精度明显高于基本 BFO；对比 PSO 算法，虽然在函数 F1，F2，F7，F9 上，收敛精度弱于 PSO，但是 GSABFO 算法在求解其他函数，特别是在 F3，F8，F10 函数上，其精度明显优于 PSO。与引力搜索算法(GSA)相比，GSA 只在 F1，F2，F4，F9 函数的寻优结果上优于 GSABFO，在求解其他函数时，GSABFO 算

第 32 卷第 9 期 系统仿真学报 Vol. 32 No. 9
2020 年 9 月 Journal of System Simulation Sep., 2020

法领先于 GSA；并且 GSA 算法并不能优化 F3 和 F10 函数，但 GSABFO 算法却在优化以上两个函数时获得了较好的结果。GSABFO 与 LAQBFO、QBFO、NAQBFO、APSO-IV 算法相比较，只有在 F4 函数的求解精度 GSABFO 低于 APSO-IV，在其他函数上 GSABFO 算法比这 4 个改进的智能算法的寻优精度都要高。

表 4 GSABFO 算法与派生算法对比
Tab. 4 Comparison of GSABFO and derivative algorithm

函数	BFO	GSABFO_1	GSABFO_2
F1	2.37E+01	7.60E-05	1.56E+01
	(2.82E+01)	(1.15E-04)	(1.80E+01)
F2	4.51E+00	3.45E-02	1.80E+00
	(9.61E+00)	(9.51E+04)	(1.97E+00)
F3	2.46E+02	8.95E+02	1.03E+02
	(2.84E+02)	(2.09E+03)	(1.32E+02)
F4	5.32E+00	2.92E+00	3.68E+00
	(6.00E+00)	(3.36E+00)	(3.86E+00)
F5	4.27E+02	2.33E-01	2.24E+02
	(4.37E+02)	(2.52E+01)	(2.62E+02)
F6	4.47E-01	1.72E-02	2.42E-01
	(5.61E-01)	(2.02E-02)	(3.33E-01)
F7	1.27E+00	2.70E+02	1.14E+00
	(1.29E+00)	(3.30E+02)	(1.15E+00)
F8	1.21E+02	4.98E+01	2.00E+01
	(1.35E+02)	(1.00E+02)	(2.17E+01)
F9	1.66E+01	1.87E+01	2.63E+00
	(1.73E+01)	(1.90E+01)	(2.82E+00)
F10	4.06E+03	4.57E+03	5.45E+01
	(4.25E+03)	(5.41E+03)	(5.82E+01)

为了进一步验证改进算法的性能，将表 1 中测试函数的维度提高到 100 维，用来测试 GSABFO 算法在高维函数上的求解性能，并将实验结果与基本 BFO、ABC、CSO、HJCSO、BAABC、SGHS 六个算法进行对比。ABC 算法的实验结果来自文献[19]，CSO 和 HJCSO 算法实验结果来自文献[20]，BAABC 算法实验结果来自文献[21]，SGHS 算法实验结果来自文献[22]。基本 BFO 算法与 GSABFO 算法参数设置与本文求解 30 维函数时的设置相同，仅将迁徙操作次数提高为 *Ned*=10。

实验结果如表 3 所示。从实验结果可以看出，GSABFO 求解 100 维函数时，对 9 个函数测试，仅在 F7 函数的测试结果上，弱于 HJCSO 和 BAABC，在其他函数上的求解结果均优于其余 6 个对比算法。

为了检验不同的改进策略与基本 BFO 结合的效果，本文将 BFO 和其派生出来的 2 个算法 GSABFO_1 和 GSABFO_2 进行对比。GSABFO_1 是基本的 BFO 中只引入 3.1 节的改进策略，GSABFO_2 中仅引入 3.2 节的改进策略。两个算法的参数设置同上。

将基本 BFO 和 GSABFO_1 进行对比，在 F1、F2 和 F4 这 3 个函数测试中，GSABFO_1 算法求解函数的适应度值领先于 BFO 的求解结果，这是由于 GSABFO_1 通过引力机制传递种群中细菌最优位置的信息，使得整个种群都有着向当前最优解移动的趋势。而 BFO 中不存在引力机制，在其趋化操作中，进行随机的游动，使得很多次游动都是无效的。在对函数 F5~F10 的测试中，BFO 求得的最优解在 F7，F9，F10 的表现反而优于 GSABFO_1，这是由于高维多峰函数中的局部极值点较多，造成解的分散，以引力方式求得的合力方向偏离全局最优解的方向过多，使得引力方向反而干扰细菌向全局最优解的方向移动。将标准 BFO 的数据与 GSABFO_2 进行对比，在高维单峰函数 F1~F4 的求解中，可以看出只进行局部维度更新的细菌觅食算法，对寻找函数的最优解的能力得到了提升，但效果不是很明显。但是在高维多峰函数中，从表 3 可以看出求解函数 F8~F10，提升的效果非常明显，在求解精度上都至少有一个数量级的提升。综合改进策略 3.1 节和 3.2 节得到 GSABFO 算法，从表 2 可以看出，GSABFO 得到的结果明显优于 BFO。

4.3 收敛速度分析

为了证明 GSABFO 算法收敛速度比经典的细菌觅食算法的收敛速度快，本文作出 BFO 和 GSABFO 在所有函数上的收敛图，如图 5 所示。

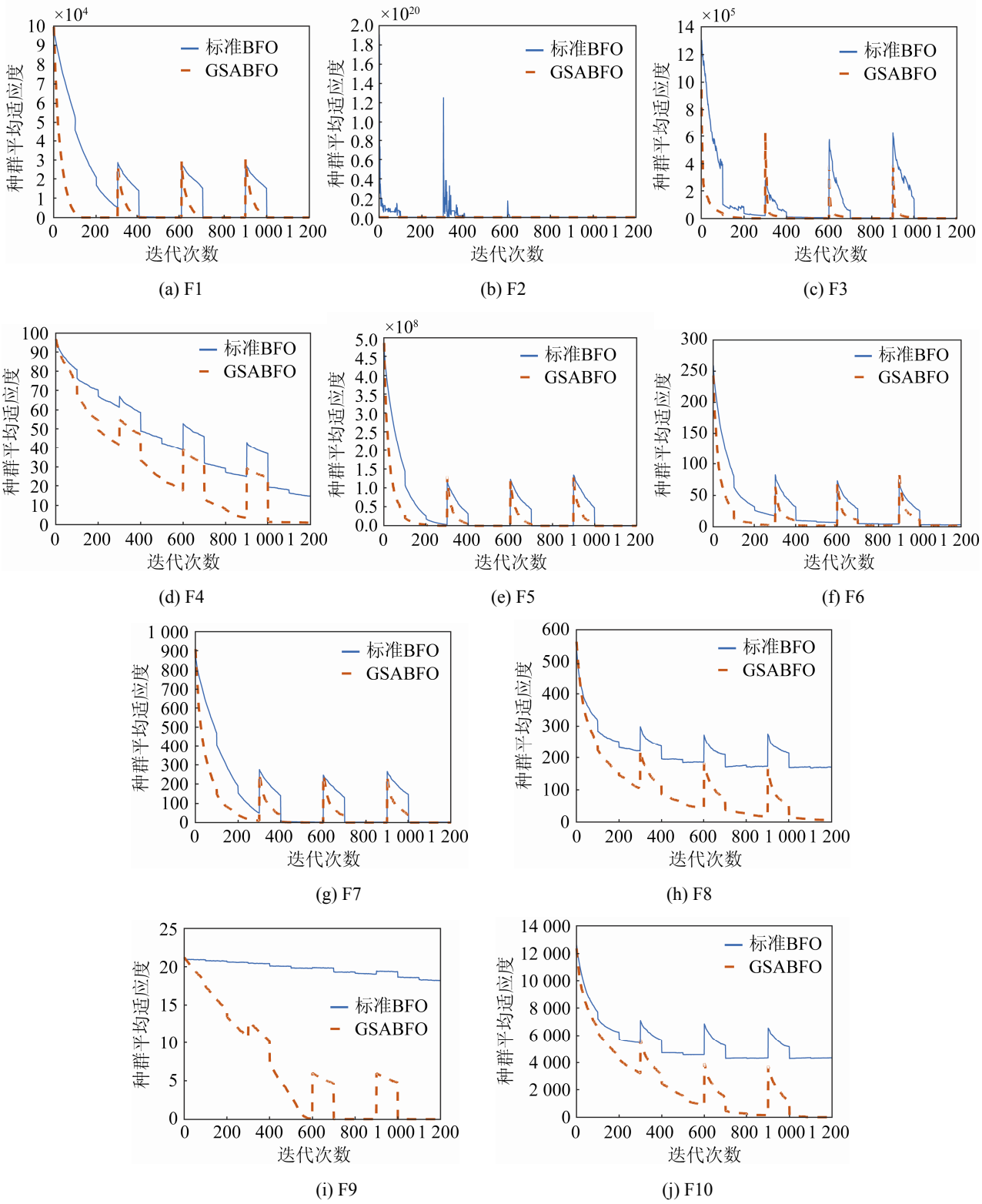

图 5　GSABFO 和 BFO 在所有测试函数收敛图

Fig. 5　GSABFO and BFO convergence graph in test function

https://dc-china-simulation.researchcommons.org/journal/vol32/iss9/11
DOI: 10.16182/j.issn1004731x.joss.19-0096

第 32 卷第 9 期 系统仿真学报 Vol. 32 No. 9
2020 年 9 月 Journal of System Simulation Sep., 2020

对比 2 个算法求解基准函数的收敛效果图，可以明显的发现，GSABFO 的收敛速度明显快于标准 BFO 的收敛速度，这主要得力于 GSABFO 算法中的引力机制在传递着种群中最优解的信息。在函数 F2，F4，F8，F9，F10 的收敛图中可以看出，GSABFO 的收敛精度远高于基本的 BFO，能够在 BFO 已经收敛的情况下，继续探寻全局最优解，算法拥有极强的开采能力。在所有的收敛图中，都有尖峰出现，这是由于在迁徙时，部分个体因迁徙而导致种群平均适应度值下降而产生的。采用双高斯函数完成迁徙操作，在产生尖峰后，种群的平均适应度能够持续下降并优于迁徙前，说明双高斯函数迁徙方式有效的增加了种群多样性，使得群体能够更好的进行全局搜索。

5 结论

本文通过对基本 BFO 算法的改进，提出了 GSABFO 算法。在标准 BFO 中引入引力机制，为每一个细菌赋予惯性质量，利用引力搜索算法计算细菌之间的引力，最终得到细菌个体趋化操作的更新方向，同时在趋化操作中引入局部维度更新策略，这些改进方式在一定程度上加速了细菌觅食算法的收敛速度并提高了收敛精度，在一些优化比较困难的多峰函数中可以获得较好的寻优结果。GSABFO 算法具有更好的普适性，但是在一些特殊的基准函数的求解精度上仍有待提高。未来主要研究点是更进一步提高算法在基准函数上面的求解精度以及将 GSABFO 算法应用到更多的场景中。

参考文献:

[1] Passino K M. Biomimicry of bacterial foraging for distributed optimization and control[J]. IEEE Control Systems Magazine (S1066-033X), 2002, 22(3): 52-67.

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

[3] 储颖, 糜华, 纪震, 等. 基于粒子群优化的快速细菌群游算法[J]. 数据采集与处理, 2010, 25(4): 442-448.
Chu ying, Mi Hua, Ji Zhen, et al. Fast Bacterial Swarming Algorithm Based on Particle Swarm Optimization[J]. Journal of Data Acquisition & Processing, 2010, 25(4): 442-448.

[4] Mishra S. A hybrid least square-fuzzy bacteria foraging strategy for harmonic estimation[J]. IEEE Transactions on Evolutionary Computation (S1089-778X), 2005, 9(1): 61-73.

[5] Niu B, Wang J, Wang H. Bacterial-inspired algorithms for solving constrained optimization problems[J]. Neurocomputing (S0925-2312), 2015, 148: 54-62.

[6] 刘珍, 孙京诰. 一种改进的细菌觅食优化算法[J].华东理工大学学报(自然科学版), 2016, 42(2): 225-232.
Liu Zhen, Sun Jinggao. An Improved Bacterial Foraging Optimization Algorithm[J]. Journal of East China University of Science and Technology (Natural Science Edition), 2016, 42(2): 225-232.

[7] Tan L J, Yi W J, Yang C, et al. Adaptive Structure-Redesigned-Based Bacterial Foraging Optimization[M]. Lanzhou China: Intelligent Computing Theories and Application, 2016.

[8] 周文宏, 雷欣, 姜建国, 等. 变概率混合细菌觅食优化算法[J]. 系统工程与电子技术, 2016, 38(4): 960-964.
Zhou Wenhong, Lei Xin, Jiang Weiguo, et al. Variable probability And hybrid bacterial foraging Optimization algorithm[J]. Systems Engineering and Electronics, 2016, 38(4): 960-964.

[9] 刘小龙, 赵奎领. 基于免疫算法的细菌觅食优化算法[J]. 计算机应用, 2012, 32(3): 634-637.
Liu Xiaolong, Zhao Kuiying. Bacteria foraging optimization algorithm based on immune algorithm[J]. Journal of Computer Applications, 2012, 32(3): 634-637.

[10] 杜鹏桢, 唐振民, 孙研. 一种混合蜂群算法的自适应细菌觅食优化算法[J]. 计算机工程, 2014, 40(7): 138-142.
Du Pengzhen, Tang Zhenmin, Sun Yan. An Adaptive Bacterial Foraging Optimization Algorithm Mixed with Bee Colony Algorithm[J]. Computer Engineering, 2014, 40(7): 138-142.

[11] 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 Guoyong, Wu Yonggang, Tan Yuxiang. Bacterial Foraging Optimization Algorithm with Quantum Behavior[J]. Journal of Electronics & Information Technology, 2013, 35(3): 614-621.

[12] 刘璐, 单梁, 戴跃伟, 等. 非线性动态自适应旋转角的量子菌群算法[J]. 控制与决策, 2017, 32(12): 2137-2144.
Liu Lu, Shan Liang, Dai Yuewei, et al. Nonlinear notation angle for dynamic adaptation in quantum bacterial foraging optimization algorithm[J]. Control and Decision, 2017, 32(12): 2137-2144.

[13] Rashedi E, Nezamabadi-Pour H, Saryazdi S. GSA: a gravitational search algorithm[J]. Information Sciences (S0020-0255), 2009, 179(13): 2232-2248.

[14] 肖辉辉, 万常选, 段艳明, 等. 基于引力搜索机制的花朵授粉算法[J]. 自动化学报, 2017, 43(4): 576-594.
Xiao Huihui, Wan Changxuan, Duan Yanming, Tan Qian-Lin. Flower Pollination Algorithm Based on Gravity Search Mechanism[J]. Acta Automatica Sinica, 2017, 43(4): 576-594.

[15] 范炜锋. 万有引力搜索算法的分析与改进[D]. 广州: 广东工业大学, 2014.
Fan Weifeng. Analysis and Improvement of Gravitational Search Algorithm[D]. Guangzhou: Guangdong University of Technology, 2014.

[16] 姜建国, 周佳薇, 郑迎春, 等. 一种自适应细菌觅食优化算法[J]. 西安电子科技大学学报, 2015, 42(1): 75-81.
Jiang Jianguo, Zhou Jiawei, Zheng Yingchun, et al. Adaptive bacterial foraging Optimization algorithm[J]. Journal of Xidian University, 2015, 42(1): 75-81.

[17] 彭君君, 刘勇进. 基于双高斯函数的一种高效鸟群优化算法[J]. 现代电子技术, 2018, 41(23): 106-112.
Peng Junjun, Liu Yongjin. An efficient bird swarm optim-ization algorithm based on double Gaussian function[J]. Modern Electronics Technique, 2018, 41(23): 106-112.

[18] 薛俊杰, 王瑛, 李浩, 等. 一种狼群智能算法及收敛性分析[J]. 控制与决策, 2016, 31(12): 2131-2139.
Xue Junjie, Wang Ying, Li Hao, et al. A smart wolf pack algorithm and its convergence analysis[J]. Control and Decision, 2016, 31(12): 2131-2139.

[19] 史旭栋, 高岳林. 基于鸡群优化和人工蜂群优化的混合算法[J]. 合肥工业大学学报(自然科学版), 2018, 41(5): 589-594.
Shi Xudong, Gao Yuelin. Hybrid algorithm based on chicken swarm optimization and artificial bee colony[J]. Journal of Hefei University of Technology (Natural Science), 2018, 41(5): 589-594.

[20] 张慕雪, 张达敏, 何锐亮. 基于模式搜索法的鸡群优化算法[J]. 微电子学与计算机, 2018, 35(4): 46-52.
Zhang Muxue, Zhang Damin, He Ruiliang. A Chicken Swarm Algorithm Based on Hooke-Jeeves[J]. Microelectronics & Computer, 2018, 35(4): 46-52.

[21] 贺桂娇, 周树亮, 冯冬青. 求解高维复杂优化问题的改进人工蜂群算法[J]. 计算机工程与应用, 2018, 54(12): 131-137.
He Guijiao, Zhou Shuliang, Feng Dongqing. Improved artificial bee algorithm for high dimensional complex optimization problems[J]. Computer Engineering and Applications, 2018, 54(12): 131-137.

[22] Pan Q K, Suganthan P N, Tasgetiren M F, et al. A self-adaptive global best harmony search algorithm for continuous optimization problems[J]. Applied Mathematics and Computation (S0096-3003), 2010, 216(3): 830-848.

DOI: 10.16182/j.issn1004731x.joss.19-0096