混合排名映射概率和混沌搜索的ABC算法

张新明 魏 峰 牛丽平 王鲜芳
(河南师范大学计算机与信息工程学院 新乡 453007)

摘 要 针对由于人工蜂群算法(Artificial Bee Colony algorithm,ABC)采用直接映射概率选择食物源而引起收敛速度慢、陷入局部最优等问题,提出一种混合排名映射概率和混沌搜索的人工蜂群算法((Artificial Bee Colony algorithm based on Hybrid rank mapping probability and Chaotic search,ABC-HC))。首先,利用目标函数值的排名来获取选择食物源的排名映射概率,并提出计算排名映射概率的两种方法;然后,在观察蜂阶段,融合这两种计算概率的方法,即不同的搜索阶段采用不同的排名映射方法计算食物源选择概率,构造基于混合排名映射概率的人工蜂群算法,以便能够维持种群的多样性避免陷于局部最优;最后,在侦查蜂阶段,使用混沌搜索替代随机搜索以便进一步提高收敛速度,最终获得较好的全局最优解。对10个标准测试函数进行仿真,结果表明,ABC-HC算法不仅提高了收敛速度,而且更能跳出局部最优,有效地找到全局最优解,优于标准的ABC算法和进化算法。

关键词 人工蜂群算法,排名映射概率,直接映射概率,混沌搜索,随机搜索
中图法分类号 TP181 **文献标识码** A

Artificial Bee Colony Algorithm Based on Hybrid Rank Mapping Probability and Chaotic Search

ZHANG Xin-ming WEI Feng NIU Li-ping WANG Xian-fang
(College of Computer and Information Engineering, Henan Normal University, Xinxiang 453007, China)

Abstract In view of the shortcomings of artificial bee colony algorithms, such as the low convergence rate and being trapped into local optimums owing to choosing the food source based on direct mapping probability, an Artificial Bee Colony optimization algorithm based on Hybrid rank mapping probability and Chaotic search (ABC-HC) was proposed in this paper. First, two computing probability method to choose food sources were created based on rank mapping. Then the ABC algorithm based on combining the two probability methods in a onlooker bee phrase was proposed in order to keep diversities of the solutions and not to be trapped into local optimums. Finally, in a scout bee phrase, random search was replaced with chaotic search to get a higher convergence rate and a global solution effectively. The simulation results on 10 standard test complicated functions indicate that the proposed optimization algorithm is rapid and effective and outperforms the standard ABC algorithm and the evolutionary ones.

Keywords Artificial bee colony algorithm (ABC), Rank mapping probability, Direct mapping probability, Chaotic search, Random search

1 引言

人工蜂群(artificial bee colony,ABC)算法[1]是一种新型群智能算法,它模拟实际蜜蜂采蜜机制处理优化问题:通过不同工种蜜蜂之间的合作,解决扩展新解域与在已知解域进行精密搜索之间的矛盾。大量Benchmark函数测试实验表明,该算法具备比传统优化方法更好的优化性能;而且ABC算法具有操作简单、易于实现及控制参数少等特点,成为现代智能优化领域等的研究热点[2-7]。但由于ABC算法是一种新颖的群集智能优化算法,还有许多问题值得研究,如收敛速度慢、容易出现"早熟"、易陷于局部最优等问题,导致这些问题的一个主要原因是ABC算法直接依据函数值映射的概率来选择食物源。本文针对标准的ABC算法存在的问题,提出了一种混合排名映射概率和混沌搜索的人工蜂群优化算法(Artificial Bee Colony algorithm based on Hybrid rank mapping probability and Chaotic search,ABC-HC)。首先,提出两种不同排名映射概率的计算方法,在观察蜂阶段,不是采用直接映射概率,而是混合使用两种排名映射概率来选择新蜜源,由此构建全局优化算法,克服ABC算法陷入局部最优的问题;然后,在观察蜂阶段,使用新型的混沌搜索算子,即用混沌搜索算子替代标准ABC算法中的随机搜索算子搜索新蜜源,更好地克服传统的ABC算法收敛速度慢的缺点,以便获得更好的优化性能,为进一步改善及应用打下良好的基础。

到稿日期:2013-05-20 返修日期:2013-08-01 本文受国家自然科学基金(61173071),河南省重点科技攻关项目(092102210017)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、数字图像处理等,E-mail:xinmingzhang@126.com;**魏 峰**(1980—),男,讲师,主要研究方向为农业信息化和图像处理;**牛丽平**(1979—),女,副教授,主要研究方向为数字图像处理;**王鲜芳**(1969—),女,教授,硕士生导师,主要研究方向为优化算法和模糊数据处理等。

2 标准的 ABC 算法

ABC 算法模拟自然界蜜蜂采蜜的过程，模拟蜜蜂根据分工不同完成采蜜过程的各阶段任务，通过食物源信息的收集与共享，寻找问题的最优解。算法将蜜蜂分为采蜜蜂、观察蜂和侦查蜂 3 类工种。食物源的数量与采蜜蜂和观察蜂的数量相等，食物源的位置代表优化问题的一个可能解，每个食物源的花蜜量对应每个解的适应度。整个算法分为 4 个阶段：初始化阶段、采蜜蜂阶段、观察蜂阶段和侦查蜂阶段。

初始化阶段。ABC 算法随机产生 N 个初始解，即 N 个采蜜蜂和食物源，计算这些初始解的适应度。每个解 $x_i(i=1,2,\cdots,N)$是一个 D 维向量，D 为优化参数的个数。完成初始化后，蜜蜂开始对所有初始解进行循环搜索。

采蜜蜂阶段。采蜜蜂会以一定概率对记忆中的食物源(原始解)位置产生改变从而找到一个新的食物源(新解)，并确认新的食物源的花蜜量，即计算新解的适应度。如果新解的适应度高于原始解，则采蜜蜂将记忆新解忘记原始解。随后，所有采蜜蜂完成搜索后回到蜂巢，将食物源信息(解的位置和适应度)与观察蜂共享。

观察蜂阶段。观察蜂根据搜集到的信息，按式(1)计算选择食物源概率 p_i，并利用轮盘赌方式取合适的标记蜜源并在其附近按式(2)搜索新蜜源，并与初始标记蜜源进行比较，选取较优的蜜源更改本次循环的初始标记蜜源。

$$p_i=fit_i/\sum_{n=1}^{N}fit_n \tag{1}$$

式中，fit_i 为第 i 个解的适应度，N 为食物源数量。

在以上两个阶段，采蜜蜂和观察蜂对记忆中原始解的邻域进行搜索的操作可定义为：

$$v_{ij}=x_{ij}+r_{ij}(x_{ij}-x_{kj}) \tag{2}$$

式中，v_{ij} 为新的蜜源位置，x_{ij} 为第 i 个蜜源的第 j 维位置，x_{kj} 为随机选择的不等于 i 的蜜源的第 j 维位置，r_{ij} 为$[-1,1]$之间的随机数。

侦查蜂阶段。如果在采蜜过程中，蜜源经若干次搜索不变，相应的采蜜蜂变成侦查蜂，随机搜索新蜜源代替初始标记蜜源中的相应位置，确定最终蜜源。

整个算法按照上述的采蜜蜂阶段、观察蜂阶段和侦查蜂阶段反复循环迭代，直到满足算法的终止条件。

3 改进的 ABC 算法

在以上标准的 ABC 算法中，观察蜂选择蜜源时直接通过目标函数值来获取食物源的概率(简称直接映射概率)，然后依据轮盘赌方式选择食物源，这是一种基于贪婪策略的选择方式，会使种群多样性降低，从而导致算法过早收敛和提前停滞。本文利用排名映射概率来提高 ABC 算法的全局搜索能力，利用混沌搜索加快收敛速度。

3.1 排名映射概率

在标准 ABC 算法的函数优化中，假如求函数最小值，并设第 i 个解的目标函数值为 f_i，则第 i 个解的适应度为：

$$fit_i=1/|f_i| \tag{3}$$

或者采用文献[5]的方法，第 i 个解的适应度为：

$$fit_i=\begin{cases}1/(1+f_i), & \text{if } f_i\geqslant 0\\ 1+|f_i|, & \text{if } f_i<0\end{cases} \tag{4}$$

其中，$|f_i|$是求 f_i 的绝对值，$i=1,2,\cdots,N$。

但这两种方式会导致如下几种情况发生：

(1)当待优化的目标函数值 f_i 无意义时，例如，函数 $f=\sin(x)/x$，当 $x=0$ 时，f 无意义，以上两式无法计算，而 f_i 为 0 时，如 $f=\sin(x)$，当 $x=0$ 时，式(3)无意义，导致 ABC 算法不能正常运行或者无限循环；

(2)当 f_i 特别小，且 $f_i>>f_k$，k 不等于 i，$k\in[1,2,\cdots,N]$时，通过式(3)计算第 i 个解的 p_i 接近 1，而 p_k 接近 0，此时，在每次循环中，观察蜂通过概率 p_i 仅仅选择第 i 个解邻域进行搜索，使种群多样性降到最低，从而引起过早收敛和提前停滞；

(3)对于式(4)，正如文献[3]指出：可以发现当 f_i 大于 0，但是一个非常小的值，如 1e−25 时，适应度 fit_i 为 1/(1+1e−25)，近似等于 1，这将导致在以后的迭代中所有解的适应度值都为 1，换句话说，适应度 1/(1+1e−25)和适应度 1/(1+1e−125)是没有区别的，因此，一个比旧解好的新解将被忽略，致使搜索停滞。

鉴于以上情况，本文提出基于排名映射获取选择食物源概率(求函数的最小值)，其过程描述如下：

(1)处理无意义的函数值 f_i，如果 f_i 无意义，定义 $f_i=inf$；

(2)对所有解的函数值按升序排名，得到各个解的排名 s_i，函数值越大，排序后的名次越大，反之越小，显然无意义的函数值映射的名次较大，名次最小为 1，最大为 N；

$$s_i=Sort(f_i),s_i\in[1,2,\cdots,N] \tag{5}$$

(3)为了缩小各个解的适应度值差距，对排名进行归一化；

$$u_i=s_i/N,u_i\in[1/N,2/N,\cdots,1] \tag{6}$$

(4)依据式(7)或者式(8)计算各个解的适应度值；

$$fit_i=1/u_i+1/(u_i)^3 \tag{7}$$

$$fit_i=1/(\max[u_1,u_2,\cdots,u_i,\cdots]+u_i) \tag{8}$$

(5)依据式(1)得到选择食物源的概率 p_i，由此得到两种计算排名映射概率的方法。

通过以上过程可以看出，①每个解的适应度与其函数值没有直接的关系，而是通过排名映射获取，与排名成反比，并且每个解的适应度一定是有意义的数值；②对于无意义的函数值其排名靠后，适应度较小，选择概率较小，但仍能保证有机会更新旧解，只是这种更新的概率较小；③由于各个解的排名不会相同，即使函数值相同(在函数值相同时，会按照原解的顺序排名)各个解的概率也绝不相同；④由于每个解都有一个排名，因此排名映射概率绝不会等于 0，不会等于 1，也不会为 inf。如此保证新的优化算法不会出现基于直接映射概率的 ABC 算法所出现的情况，保证种群的多样性，避免出现“早熟”现象的发生。

3.2 混沌搜索

在标准的 ABC 算法中，在侦查蜂阶段，当在某个食物源的位置周围搜索的不变次数达到最大解而仍没有找到更优位置时，重新随机初始化该位置，此操作目的是增强种群的多样性，防止种群陷入局部最优。该步骤虽然增加了种群的多样性，但是也降低了算法的收敛速度。混沌系统所具有内在的随机性、遍历性、“规律”等特点，使得混沌搜索能在一定范围内按其自身的“规律”不重复地遍历每一个状态，因此混沌搜索在数字图像处理等诸多领域中得到广泛的应用[8-10]。也正

因为如此，许多学者使用混沌特性对 ABC 算法进行了不同的改进。如文献[11]使用混沌初始化，使得初始种群在定义域内的分布更加均匀；文献[11,12]采用混沌侦察蜂，针对陷入局部极值的食物源，利用混沌序列进行局部搜索，进而产生较优的食物源。文献[13]在文献[11,12]的基础上，在观察蜂阶段结束之后执行混沌局部搜索算子，增强算法的局部搜索能力。本文引入的混沌搜索主要用来提高算法的收敛速度和解的精度。

混沌搜索[8-10]的基本思想就是把混沌变量线性映射到优化变量的取值区间，然后利用混沌变量进行搜索。本文创建新型的混沌搜索采用 Logistic 映射混沌系统，传统的 Logistic 模型如式(9)所示：

$$z_{k+1,j}=4z_{k,j}(1-z_{k,j}) \tag{9}$$

其中，$z_{k,j}$不能为 0.25、0.5 和 0.75 等值。为了降低 Logistic 映射产生混沌序列的不均匀性，引入分段 Logistic 混沌映射：

$$z_{k+1,j}=\begin{cases}4z_{k,j}(0.5-z_{k,j}), & 0\leqslant z_{k,j}<0.5\\ 4(z_{k,j}-0.5)(1-z_{k,j}), & 0.5\leqslant z_{k,j}<1\end{cases} \tag{10}$$

那么，如果在采蜜过程中，蜜源经若干次搜索不变，相应的采蜜蜂变成侦查蜂，不是随机搜索，而是混沌搜索新蜜源代替初始标记蜜源中的相应位置，确定最终蜜源。受文献[14]的启示，创建混沌搜索算子，如式(11)所示：

$$v_{ij}=x_{ij}+2(z_{i,j}-0.5)\ x_{ij} \tag{11}$$

从以上混沌搜索算子可以看出，混沌搜索范围不是固定的，是依据当前解来确定的；而且每次循环中运行混沌优化算子都是基于 x_{ij}，在 x_{ij} 的尺度下自动调整，x_{ij} 越大，搜索空间越大，反之，越小；这样更能提高搜索精度和收敛速度，所以此混沌搜索算子能较大地提高收敛速度和搜索精度。

3.3 混合排名映射概率和混沌搜索的 ABC 算法(ABC-HC)

ABC-HC 算法的基本步骤如下：

步骤 0　初始化参数。先设定 *limit*、*M* 和 *N* 的值，其中，*limit* 是最大解不变搜索次数，*M* 是最大迭代次数，*N* 是采蜜蜂、观察蜂和蜜源的数量；并初始化解不变搜索次数数组 **lim**，即令 $lim_i=0, i=1,\cdots,N$。

步骤 1　随机构造蜜源位置。按式(12)随机产生 $2N$ 个位置。

$$v_{ij}=a_j+rand(0,1)\times(b_j-a_j) \tag{12}$$

式中，v_{ij}为第 i 个蜜蜂第 j 维对应的搜索后的位置，$rand(0,1)$是[0,1]之间的随机数，b_j 和 a_j 代表第 j 维变量的上下界。

步骤 2　计算适应度。对于以上 $2N$ 个蜜源，计算目标函数值，选取适应度值高的 N 个位置作为待循环搜索的蜜源位置。

步骤 3　搜索新蜜源。采蜜蜂在蜜源附近按式(2)搜索新蜜源，并对新蜜源按式(13)作越界处理。

$$v_{ij}=\max(a_j,v_{ij}),v_{ij}=\min(b_j,v_{ij}) \tag{13}$$

步骤 4　计算新蜜源的适应度，比较前后蜜源优劣。若搜索后的蜜源优于搜索前的蜜源，则代替先前蜜源，并令 $lim_i=0$，否则，令 $lim_i=lim_i+1$。

步骤 5　计算排名映射概率。观察蜂根据采蜜蜂释放的花蜜信息计算排名映射概率，如果搜索处于前半阶段，按式(7)计算排名映射概率；如果搜索处于后半阶段，按式(8)计算排名映射概率。即

if $t>M/2$

采用含有式(7)的排列映射概率方法计算概率；

else

采用含有式(8)的排列映射概率方法计算概率；

endif

其中，t 为当前迭代次数。然后依据所求概率按轮盘赌方式选择蜜源，并在其附近按式(2)搜索新蜜源，对新蜜源按式(13)作越界处理。

步骤 6　计算新蜜源的适应度，比较前后蜜源优劣，若搜索后的蜜源优于搜索前的蜜源，则代替先前蜜源，并令 $lim_i=0$，否则，令 $lim_i=lim_i+1$。

步骤 7　判断是否出现侦查蜂，若某些蜜源经 *limit* 次循环不变，放弃该蜜源，即当 max(**lim**)$>$*limit* 时，相应采蜜蜂变成侦查蜂，按式(11)混沌搜索新蜜源。

步骤 8　计算新蜜源的适应度，如果新蜜源优于初始标记蜜源，则用新蜜源代替初始标记蜜源，并令 $lim_i=0$，否则，令 $lim_i=lim_i+1$。

步骤 9　从步骤 4 开始重新搜索，直到满足终止条件(即迭代次数超过了 *M*)停止搜索，获得全局最优解。

4　仿真实验及结果分析

为了验证本文提出算法的有效性，将它与标准的 ABC 算法进行比较。为了叙述方便，将采用式(3)获取直接映射概率的 ABC 算法称为 ABC-1，而采用式(4)获取直接映射概率的 ABC 算法称为 ABC-2，基于混合排名映射概率的 ABC 算法简称为 ABC-H 算法。限于篇幅，选取 10 个常用测试优化算法性能的典型函数进行计算比较，考察它们搜索到全局最优解、成功率及运行时间等情况。在本实验中，算法采用 MATLAB R2010A 语言实现，所有实验在 DELL Intel 酷睿 370 主频为 2.4G 的 CPU 和内存为 2G DDR3 RAM 的笔记本上进行。鉴于公平性原则，设置相同的参数：采蜜蜂、观察蜂和蜜源的数量都为 40，*limit* 为 $N*D$。各算法的最大迭代次数为：对于 f_{01}、f_{02}、f_{03}、f_{04}、f_{05}、f_{07}、f_{08}和 f_{09}，$M=125$，而 f_{06}和 f_{10}函数，$M=250$。这样对应的函数最大评价次数见表 1，其中，这些评价次数都未包含初始化位置后的评价次数 $2N$，各算法均随机运行 50 次。对于实验 1，选取测试函数搜索的最好值 Best、搜索的平均值 Mean、搜索的最差值 Worst、搜索的方差 Std、目标函数的最大评价次数 Max_FFS、成功率(Success Rate)SR 及运行时间 Time 作为评价标准来考察各算法的寻优性能。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。Best 和 Worst 反映了解的质量；Mean 显示了在给定的函数评价次数下算法所能达到的精度，反映了算法的收敛速度；Std 反映了算法的稳定性和鲁棒性。测试函数的情况见下面的说明。为了使比较更具有普适性，这 10 个函数代表着不同的情况，其维数从 2 维到 6 维不等。这 10 个目标函数表达式和全局最优解等情况如下：

$$f_{01}=0.002+\sum_{j=1}^{25}1/\left[j+\sum_{i=1}^{2}(x_i-a_{ij})^6\right],$$
$$-65.536\leqslant x_i\leqslant 65.536$$

其中，

$$a_{ij}=\begin{bmatrix}-32 & -16 & 0 & 16 & 32 & -32 & -16 & \cdots & 0 & 16 & 32\\ -32 & -32 & -32 & -32 & -32 & -16 & -16 & \cdots & 32 & 32 & 32\end{bmatrix}$$

f_{01}函数为 Foxholes 函数，其维数为 2，含有大量的深度不同的“坑”，“坑”的四周都是相当平滑的表面。全局最优点

为：$\min(f_{01})=f_{01}(-32,-32)\approx 0.998$。

$$f_{02}=(4-2.1x_1^2+\frac{1}{3}x_1^4)x_1^2+x_1x_2+(-4+4x_2^2)x_2^2,\quad -10\leqslant x_i\leqslant 10$$

f_{02}函数为 Six-Hump-Camel-Back 函数，其维数为 2，有 6 个局部最优点、2 个全局最优点，全局最优点为：$\min(f_{02})=f_{02}(0.089842,-0.71266)=f_{02}(-0.089842,0.71266)\approx -1.0316$。

$$f_{03}=(x_2-\frac{5.1}{4\pi^2}x_1^2+\frac{5}{\pi}x_1-6)^2+10(1-\frac{1}{8\pi})\cos x_1+10,\quad -5\leqslant x_1\leqslant 10,0\leqslant x_2\leqslant 15$$

f_{03}函数为 Branin 函数，其维数为 2，多峰函数，全局最优点有 3 个，为：$\min(f_{03})=f_{03}(-3.1416,12.275)=f_{03}(3.1416,2.275)=f_{03}(9.425,2.425)\approx 0.39789$。

$$f_{04}=[1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)][30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)],-5\leqslant x_i\leqslant 5$$

f_{04}函数为 Goldstein & Price 函数，其维数为 2，是一个著名的二维测试函数，在定义域内共有 3 个极小点，其中全局最小点为(0，−1)，对应的全局最小值为 3。两个局部极小点分别为(1.8，0.2)和(1.2，0.8)，对应的局部极小值分别为 84 和 940。

$$f_{05}=-\sum_{i=1}^{4}c_i\exp[-\sum_{j=1}^{3}a_{ij}(x_j-p_{ij})^2],0\leqslant x_j\leqslant 1$$

$$f_{06}=-\sum_{i=1}^{4}c_i\exp[-\sum_{j=1}^{6}a_{ij}(x_j-p_{ij})^2],0\leqslant x_j\leqslant 1$$

在 f_{05}和 f_{06}中，c_i、a_{ij}和 p_{ij}为 3 个参数，限于篇幅，它们的取值没有列出，其取值见文献[15]。f_{05}和 f_{06}函数为 Hartman 族函数。f_{05}为 Hartman1 函数，其维数为 3，有 4 个局部最优点，全局最优点为：$\min(f_{05})=f_{05}(0.114,0.556,0.852)\approx -3.8628$。$f_{06}$函数为 Hartman2 函数，其维数为 6，有 6 个局部最优点，全局最优点为：$\min(f_{06})=f_{06}(0.201,0.15,0.477,0.275,0.311,0.657)\approx -3.261$。

$$f_{07}=-\sum_{j=1}^{5}[\sum_{i=1}^{4}(x_i-C_{ij})^2+B_j]^{-1},0\leqslant x_i\leqslant 10$$

$$f_{08}=-\sum_{j=1}^{7}[\sum_{i=1}^{4}(x_i-C_{ij})^2+B_j]^{-1},0\leqslant x_i\leqslant 10$$

$$f_{09}=-\sum_{j=1}^{10}[\sum_{i=1}^{4}(x_i-C_{ij})^2+B_j]^{-1},0\leqslant x_i\leqslant 10$$

在以上 f_{07}、f_{08}和 f_{09}中的 B_j 和 C_{ij}为：

$B=0.1\times[1,2,2,4,4,6,3,7,5,5]$

$$C=\begin{bmatrix}4&1&8&6&3&2&5&8&6&7\\4&1&8&6&7&9&5&1&2&3.6\\4&1&8&6&3&2&3&8&6&7\\4&1&8&6&7&9&3&1&2&3.6\end{bmatrix}$$

f_{07}、f_{08}和 f_{09}称为 Shekel 族函数，f_{07}、f_{08}和 f_{09}在解空间中分别具有 5、7 和 10 个局部最优解。f_{07}函数为 Shekel1 函数，其维数为 4，B_j 取 B 中前 5 个数据，C_{ij}取 C 中前 5 列数据，全局最优点为：$\min(f_{07})=f_{07}(4,4,4,4)\approx -10.1532$。$f_{08}$函数为 Shekel2 函数，其维数为 4，$B_j$ 取 B 中前 7 个数据，C_{ij}取 C 中前 7 列数据，全局最优点为：$\min(f_{08})=f_{08}(4,4,4,4)\approx -10.4029$。$f_{09}$函数为 Shekel3 函数，其维数为 4，$B_j$ 取 B 中全部数据，C_{ij}取 C 中全部数据，全局最优点为：$\min(f_{09})=f_{09}(4,4,4,4)\approx -10.5364$。

$$f_{10}=-\sum_{i=1}^{n}(x_i-1)^2-\sum_{i=2}^{n}x_ix_{i-1},-n^2\leqslant x_i\leqslant n^2$$

f_{10}函数为 Trid 函数[15]，多峰函数，其维数为 6，全局最优点为：$\min f_{10}=f_{10}(6,10,12,12,10,6)=-50$。

实验 1　ABC-1、ABC-2、ABC-H 和 ABC-HC 优化效果对比。表 1 列出了 4 种优化算法对 10 个函数的优化结果、成功率、目标函数最大评价次数(Max_FFS)和运行时间(Time)，其中，表中优者用黑体表示，成功率为获得成功的次数与运行总次数之比，而成功次数为算法搜索的最优值(x)小于或等于参考值(RF)的数目。具体的比较公式为：round($x*10000$)<=RF，round()为取整运算。由表 1 数据对比可以看出，在所有的标准测试函数中，无论是解的质量还是算法的成功率，ABC-H 算法优化性能优于 ABC-2 算法和 ABC-1 算法。从成功率(SR，见表 1 第 9 列)上看，在这 3 种算法中，ABC-H 算法几乎在所有 10 个测试函数上都取得了较好的优化结果，其成功率都不低于 ABC-1 算法和 ABC-2 算法，例如 ABC-1 算法和 ABC-2 算法虽然在 f_{01}、f_{02}、f_{03}、f_{05}和 f_{06}上的成功率为 100%，但余下的函数成功率不能令人满意，尤其是 f_{07}至 f_{10}成功率较低，对于 ABC-1 算法，在 f_{08}和 f_{09}上更低，其成功率分别为 6%和 2%。从标准方差(Std，见表 1 第 6 列)上看，相对于其它两种算法，ABC-H 算法除 f_{01}外，在其它测试函数上都有较好的方差，尤其在 f_{07}至 f_{10}上更是远远胜过 ABC-1 算法和 ABC-2 算法。这说明 ABC-H 算法有更好的稳定性和鲁棒性，这也证明了标准的 ABC 算法由于采用直接映射概率来选择食物源，在其附近按式(2)搜索新蜜源会出现“早熟”和提前停滞现象；而采用排名映射的 ABC-H 算法使得“早熟”和提前停滞现象大幅度减少。ABC-HC 算法与 ABC-H 算法相比，由于采用了混沌搜索提高了收敛速度，例如 ABC-HC 算法的成功率 f_{01}、f_{02}、f_{03}、f_{05}至 f_{09} 8 个函数是 100%，f_{04}和 f_{10}成功率分别为 98%和 94%；而 ABC-H 算法在除了 f_{10}的成功率为 98%高于 ABC-HC 算法，在 f_{07}至 f_{09}上的成功率都低于 ABC-HC 算法。从标准方差看，ABC-HC 算法在除了 f_{02}和 f_{05}外，其它函数的方差都优于 ABC-H 算法。这说明 ABC-HC 算法相比于 ABC-H 算法有更好的精度、更好的稳定性和收敛速度。从运行时间和目标函数的评价次数看，评价次数最多的是 f_{06}和 f_{10}两个函数，因为这两个函数的维数最高是 6。运行时间和目标函数的评价次数分别见表 1 的第 4 列和第 3 列，4 种算法的目标函数的评价次数相同，运行时间几乎相同。反过来说，如果 ABC 算法需要达到 ABC-HC 算法优化性能，需要增加迭代次数(假定增加迭代次数能够提高优化性能)，也就增加了目标函数的评价次数，从而增加了计算复杂度，耗时就会增加。所以，在达到同样的优化效果情况下，ABC-HC 算法目标评价次数少，需要时间少，这说明 ABC-HC 算法收敛速度更快。此实验说明：本文提出的混合排名映射概率和混沌搜索的 ABC 算法是可行和有效的。

实验 2　ABC-HC 算法与 FEP 算法[16]、LEP 算法[17]和 SPMEP 算法[18]优化效果对比。表 2 是将本文提出的 ABC-HC 算法与 FEP 算法、LEP 算法和 SPMEP 算法在 9 个函数上进行优化的结果。其中，FEP 算法、LEP 算法和 SPMEP 算法都是进化算法和其改进版，其优化数据来自相应的文献。从表 2 看出，(1)本文提出的 ABC-HC 算法与 SPMEP 算法相比，除在 f_{04}优化效果差于 SPMEP 算法外，其他 8 个函数都大大优于 SPMEP 算法的优化效果，见表 2 第 5 列。(2)ABC-

HC算法与LEP算法相比，在9个测试函数中，除了在 f_{01} 优化效果差外，其他8个函数都大大优于LEP算法的优化效果，见表2第4列。(3)ABC-HC算法与FEP算法相比，在9个函数上，ABC-HC算法全部优于FEP算法，见表2第3列。这说明本文提出的ABC-HC算法与进化算法相比，其优势明显。

表1 4种优化算法计算结果以及耗时

Functions	Algorithms	Max_FFS	Time/s	Mean	Std	Best	Worst	SR	RF
f_{01}	ABC-HC	10,000	1.0119	0.9980	2.7379e−16	0.9980	0.9980	50/50	0.9980
	ABC-H	10,000	1.0001	0.9980	3.5677e−16	0.9980	0.9980	50/50	
	ABC-2	10,000	0.9936	0.9980	**2.2984e−16**	0.9980	0.9980	50/50	
	ABC-1	10,000	1.0104	0.9980	2.5867e−16	0.9980	0.9980	50/50	
f_{02}	ABC-HC	10,000	0.3387	−1.0316	7.1000e−16	−1.0316	−1.0316	50/50	−1.0316
	ABC-H	10,000	0.3317	−1.0316	**7.0859e−16**	−1.0316	−1.0316	50/50	
	ABC-2	10,000	0.3278	−1.0316	7.6063e−16	−1.0316	−1.0316	50/50	
	ABC-1	10,000	0.3330	−1.0316	7.6196e−16	−1.0316	−1.0316	50/50	
f_{03}	ABC-HC	10,000	0.3299	0.3979	**9.3688e−12**	0.3979	0.3979	50/50	0.3979
	ABC-H	10,000	0.3211	0.3979	5.3591e−11	0.3979	0.3979	50/50	
	ABC-2	10,000	0.3210	0.3979	1.3649e−09	0.3979	0.3979	50/50	
	ABC-1	10,000	0.3235	0.3979	4.1541e−09	0.3979	0.3979	50/50	
f_{04}	ABC-HC	10,000	0.3418	**3.0000**	**4.7701e−05**	**3.0000**	**3.0003**	**49/50**	3.0000
	ABC-H	10,000	0.3309	3.0000	1.9191e−04	3.0000	3.0014	49/50	
	ABC-2	10,000	0.3336	3.0002	7.3404e−04	3.0000	3.0048	44/50	
	ABC-1	10,000	0.3308	3.0000	6.2771e−05	3.0000	3.0003	46/50	
f_{05}	ABC-HC	10,000	0.6667	−3.8628	9.2765e−15	−3.8628	−3.8628	50/50	−3.8628
	ABC-H	10,000	0.6611	−3.8628	**2.9811e−15**	−3.8628	−3.8628	50/50	
	ABC-2	10,000	0.6511	−3.8628	1.2088e−13	−3.8628	−3.8628	50/50	
	ABC-1	10,000	0.6529	−3.8628	1.4639e−12	−3.8628	−3.8628	50/50	
f_{06}	ABC-HC	20,000	1.3417	−3.3261	**8.2955e−15**	−3.3261	−3.3261	50/50	−3.3261
	ABC-H	20,000	1.3313	−3.3261	5.9955e−14	−3.3261	−3.3261	50/50	
	ABC-2	20,000	1.3143	−3.3261	8.6964e−11	−3.3261	−3.3261	50/50	
	ABC-1	20,000	1.3113	−3.3261	1.0290e−11	−3.3261	−3.3261	50/50	
f_{07}	ABC-HC	10,000	0.6592	**−10.1532**	**1.3835e−06**	**−10.1532**	**−10.1532**	**50/50**	−10.1532
	ABC-H	10,000	0.6571	−10.1532	1.6994e−05	−10.1532	−10.1531	49/50	
	ABC-2	10,000	0.6561	−10.1527	0.0024	−10.1532	−10.1365	40/50	
	ABC-1	10,000	0.6613	−10.1485	0.0127	−10.1532	−10.0922	15/50	
f_{08}	ABC-HC	10,000	0.6605	**−10.4029**	**2.7491e−06**	**−10.4029**	**−10.4029**	**50/50**	−10.4029
	ABC-H	10,000	0.6535	−10.4029	9.0773e−05	−10.4029	−10.4023	49/50	
	ABC-2	10,000	0.6633	−10.3936	0.0626	−10.4029	−9.9602	38/50	
	ABC-1	10,000	0.6603	−10.3833	0.0394	−10.4029	−10.1754	3/50	
f_{09}	ABC-HC	10,000	0.6619	**−10.5364**	**6.4481e−06**	**−10.5364**	**−10.5364**	**50/50**	−10.5364
	ABC-H	10,000	0.6575	−10.5364	4.3248e−05	−10.5364	−10.5361	49/50	
	ABC-2	10,000	0.6666	−10.5356	0.0021	−10.5364	−10.5242	28/50	
	ABC-1	10,000	0.6640	−10.5037	0.0785	−10.5364	−10.1007	2/50	
f_{10}	ABC-HC	20,000	0.7348	**−49.9977**	**0.0032**	**−50.0000**	**−49.9844**	**47/50**	−49.9900
	ABC-H	20,000	0.7303	−49.9971	0.0047	−50.0000	−49.9674	49/50	
	ABC-2	20,000	0.7300	−49.9813	0.0175	−49.9996	−49.9166	19/50	
	ABC-1	20,000	0.7300	−49.9687	0.0368	−49.9984	−49.8101	17/50	

表2 4种优化算法计算结果

Functions	Max_FFS	FEP	LEP	SPMEP	ABC-HC
		Mean/Std	Mean/Std	Mean/Std	Mean/Std
f_{01}	10,000	1.22/0.56	**0.998004/0**	1.0/1.6e−15	0.9980/2.7379e−16
f_{02}	10,000	−1.03/4.9e−07	−1.03/2.9e−04	−1.03/4.3e−10	**−1.0316/7.1000e−16**
f_{03}	10,000	0.398/1.5e−07	0.398/4.4e−04	0.398/5.7e−10	**0.3979/9.3688e−12**
f_{04}	10,000	3.02/0.11	3/3.7e−03	**3.0/5.6e−09**	3.0000/4.7701e−05
f_{05}	10,000	−3.86/1.4e−05	−3.85/9.3e−03	−3.86/1.5e−09	**−3.8628/9.2765e−15**
f_{06}	20,000	−3.27/5.9e−02	−2.87/0.16	−3.25/5.4e−02	**−3.3261/8.2955e−15**
f_{07}	10,000	−5.52/1.59	−4.76/0.16	−6.63/3.5	**−10.1532/1.3835e−06**
f_{08}	10,000	−5.52/2.12	−5.53/1.41	−7.4/3.00	**−10.4029/2.7491e−06**
f_{09}	10,000	−6.57/3.14	−8.23/1.16	−6.53/4.00	**−10.5364/6.4481e−06**

总之，不管从实验1还是从实验2来看，本文提出的ABC-HC算法优化性能都是出色的，在大多数函数上优于采用直接映射概率的ABC算法、FEP算法、LEP算法和SPMEP算法。

结束语 1.针对ABC算法存在收敛速度慢和易陷入局部最优的问题，提出一种基于混合排名映射概率和混沌搜索的ABC算法，实验结果证明ABC-HC算法能够跳出局部最优；也说明本方法是普遍有效的，在保留原算法简单易行、搜索能力强的基础上，更具目的性和灵活性，为进一步改善及应用打下良好的基础。

2.与文献[11-13]不同，本文提出的新型的混沌搜索算子不仅得到全局最优解，而且能提高收敛速度。基于ABC-HC算法还可提出改进算法，以便更进一步提高优化效果。

(下转第144页)

的存储空间，具有一定的实用价值。

参 考 文 献

[1] Engler D, Ashcraft K. RacerX: Effective static detection of race conditions and deadlocks [C] // Proceeding of the 19th ACM Symp on Operating Systems Principles (SOSP). New York: ACM, 2003: 237-252

[2] Blanc N, Kroening D. Race analysis for SystemC using model checking [C] // Proceeding of IEEE/ACM Int Conf on Computer Aided Design. New York: ACM, 2008: 356-363

[3] Yu Y, Rodeheffer T, Chen W. Racetrack: Efficient detection of data race conditions via adaptive tracking [C] // Proceeding of the 12th ACM Symp on Operating Systems Principles. New York: ACM, 2005: 221-234

[4] Klein P N, Lu H I, Netzer R H B. Detecting race conditions in parallel programs that use semaphores [J]. Algorithmic, 2003, 35(4): 321-345

[5] Pozniansky E, Schuster A. Efficient on-the-fly data race detection in multithreaded C++ programs [C] // Proceeding of PPoPP' 03. New York: ACM, 2003: 179-190

[6] Tai K C. Race analysis of traces of asynchronous message-passing programs [C] // Proceeding of ICDCS'97. Piscataway, NJ: IEEE, 1997: 261-268

[7] Lamport L. Time, clocks, and the ordering of events in a distributed system [J]. Communications of the ACM, 1978, 21(7): 558-565

[8] Park M Y, Hai N C T, Jun Y K, et al. Visualization of message races in MPI parallel programs [C] // Proceeding of the 7th IEEE Int Conf on Computer and Information Technology. Piscataway, NJ: IEEE, 2007: 316-321

[9] H wang G H, Chang S J, Chu H D. Technology for testing nondeterministic client/server database applications [J]. IEEE Trans on Software Engineering, 2004, 30(1): 59-77

[10] Schaeli B, Gerlach S, Hersch R D. Decomposing partial order execution graphs to improve message race detection [C] // Proceeding of IEEE Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2007: 1-8

[11] Mitchell B. Resolving race conditions in asynchronous partial order scenarios [J]. IEEE Trans on Software Engineering, 2005, 31(9): 767-784

[12] 汤小丹，梁红兵，哲凤屏，等. 计算机操作系统(第三版)[M]. 西安：电子科技大学出版社，2007：59-61

[13] 陈艳，徐晓峰，李晓潮，等. 实时嵌入式系统的竞态条件及其分析方法研究[J]. 计算机研究与发展，2010，47(7)：1201-1210

(上接第 106 页)

简言之，本文提出的 ABC-HC 算法有较快的收敛速度、较好的全局搜索能力，在解决优化问题上优势明显。

参 考 文 献

[1] Karaboga D. An idea based on honey bee swarm for numerical optimization [R]. Technical Report-TR06. Erciyes University, Kayseri, Turkey, 2005

[2] Karaboga D, Basturk B. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm [J]. Journal of Global Optimization, 2007, 39(3): 459-171

[3] Banharnsakun A, Achalakul T, Sirinaovakul B. The best-so-far selection in artificial bee colony algorithm [J]. Applied Soft Computing, 2011, 11: 2888-2901

[4] Li G Q, Niu P F, Xiao X J. Development and investigation of efficient artificial bee colony algorithm for numerical function optimization [J]. Applied Soft Computing, 2012, 12: 320-332

[5] 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(1): 3508-3531

[6] Gao W F, Liu S Y. A modified artificial bee colony algorithm [J]. Computers and Operations Research, 2012, 39(2): 687-697

[7] 毕晓君，王艳娇. 加速收敛的人工蜂群算法[J]. 系统工程与电子技术，2011，33(12)：2755-2761

[8] 张新明，孙印杰. 基于混沌优化的自适应中值滤波[J]. 电子技术应用，2007，33(9)：63-65

[9] 张新明，徐久成. 基于混沌理论和支持向量机的人脸识别方法[J]. 高技术通讯，2009，19(5)：494-500

[10] Zhang X M, Yan L. A fast image thresholding method based on chaos optimization and recursive algorithm for two-dimensional Tsallis entropy [J]. Journal of Computers, 2010, 5(7): 1054-1061

[11] Alatas B. Chaotic bee colony algorithms for global numerical optimization [J]. Expert Systems with Applications, 2010, 37: 5682-5687

[12] 罗钧，李研. 具有混沌搜索策略的蜂群优化算法[J]. 控制与决策，2010，25(12)：1913-1916

[13] 李志勇，李玲玲，王翔，等. 基于 Memetic 框架的混沌人工蜂群算法[J]. 计算机应用研究，2012，29(11)：4045-4049

[14] 张新明，雷冠军，闫林，等. 一种新型快速的直接随机优化算法[J]. 吉林大学学报：理学版，2012，50(4)：750-756

[15] 纪震，廖慧连，吴青华. 粒子群算法及应用[M]. 北京：科学出版社，2009

[16] Yao X, Liu Y, Lin G. Evolutionary programming made faster [J]. IEEE Transactions on Evolutionary Computation, 1999, 3(2): 82-102

[17] Dong H B, He J, Huang H K, et al. Evolutionary programming using a mixed mutation strategy [J]. Information Sciences, 2007, 177(1): 312-327

[18] Ji M, Yang H, Yang Y, et al. A single component mutation evolutionary programming [J]. Applied Mathematics and Computation, 2010, 215(10): 3759-3768