DOI: 10.11992/tis.201612030
网络出版地址: http://kns.cnki.net/kcms/detail/23.1538.TP.20170704.1702.006.html

一种求解多模态复杂问题的混合和声差分算法

黎延海, 拓守恒
（陕西理工大学 数学与计算机科学学院, 陕西 汉中 723001）

摘　要: 针对多模态复杂优化问题, 提出了一种基于和声搜索和差分进化的混合优化算法: HHSDE 算法。在不同的进化阶段, HHSDE 算法依据累积加权更新成功率来自适应地选择和声算法或差分算法作为更新下一代种群的方式, 并改进了差分算法的变异策略来平衡差分算法的全局与局部搜索能力。通过对 10 个多模态 Benchmark 函数进行测试, 利用 Wilcoxon 秩和检验对不同算法的计算结果进行比较, 结果表明 HHSDE 算法具有收敛速度快, 求解精度高, 稳定性好等优势。

关键词: 和声搜索; 差分进化; 混合机制; 更新成功率; 变异策略; 多模态优化问题
中图分类号: TP391　　**文献标志码:** A　　**文章编号:** 1673−4785(2018)02−0281−09

中文引用格式: 黎延海, 拓守恒. 一种求解多模态复杂问题的混合和声差分算法[J]. 智能系统学报, 2018, 13(2): 281–289.
英文引用格式: LI Yanhai, TUO Shouheng. Hybrid algorithm based on harmony search and differential evolution for solving multi-modal complex problems[J]. CAAI transactions on intelligent systems, 2018, 13(2): 281–289.

Hybrid algorithm based on harmony search and differential evolution for solving multi-modal complex problems

LI Yanhai, TUO Shouheng
(School of Mathematics and Computer Science, Shaanxi University of Technology, Hanzhong 723001, China)

Abstract: This paper presents a hybrid algorithm (HHSDE) based on harmony search and differential evolution for solving multi-modal complex optimization. In different evolution stages, HHSDE algorithm self-adaptively selects harmony search (HS) or differential evolution (DE) algorithm as the means of updating the next generation of population on basis of the cumulative success rate of weighted update, in addition, it changes the mutation strategy of differential evolution (DE) algorithm for balancing the global and local search ability of the differential evolution (DE) algorithm. To investigate the performance of HHSDE, ten multi-modal Benchmark functions were tested. The experimental results, compared with other algorithms by Wilcoxon rank sum test, indicate that HHSDE algorithm has the advantages such as fast convergence speed, high solution precision and excellent stability.

Keywords: harmony search; differential evolution; hybrid mechanism; success rate; mutation strategy; multimodal optimization problem

随着社会的进步和科技的快速发展, 大量的复杂优化问题相继出现, 为此, 许多群体智能优化算法[1-5]被提出并成功应用于解决科学计算和工程技术中的大规模复杂问题。差分进化算法 (differential evolution, DE)[2]与和声搜索算法 (harmony search, HS)[5]是两种优秀的群体智能优化算法, 已经吸引了众多研究者的关注。虽然两种算法具有较好的搜索能力, 但仍然存在一些固有的缺陷: HS 算法局部开发能力较弱, 求解精度低; DE 算法容易陷入局部最优而导致停滞。为克服两种算法的不足, 近年来涌现了许多改进算法。一方面, 许多 HS 和 DE 算法的变体被提出, 如改进和声搜索算法 (improved harmony search algorithm, IHS)[5]、全局和声搜索算法 (novel global harmony search algorithm,

收稿日期: 2016−12−26.　**网络出版日期:** 2017−07−04.
基金项目: 国家自然科学基金项目 (11401357); 陕西省教育厅科研项目 (14JK1130); 陕西理工大学校级科研项目 (SLGKY 2017-05).
通信作者: 黎延海. E-mail: Chenxi81991@sina.com.

NGHS)[6]、多子群混合和声搜索算法 (multiple-sub-groups hybrid harmony search algorithm, MHHS)[7]、动态降维和声算法 (dyna-mic adjustment atrategy in IHS, DIHS)[8]、全局竞争和声搜索算法 (global competitive harmony search algorithm, GCHS)[9]、复合实验向量生成策略的差分进化算法 (DE with composite trial vector Generation Strategies, CoDE) [10]、全局自适应差分优化算法 (DE with strategy adaptation, SaDE)[11]、双变异差分进化算法 (DE with double mutation strategies, DaDE)[12]等；另一方面，也出现了一些 HS 和 DE 的融合算法，主要有：采用双种群进化的协同差分和声算法 (coevolutionary DE with HS, CDEHS)[13]、使用各种差分变异算子来代替 HS 算法中原有的音调调节方法的混沌自适应差分和声搜索算法 (chaotic self-adaptive differential harmony search algorithm, CSADHS)[14]、基于差分算子的和声搜索算法[15-16]、改进的和声差分算法 (improved harmony search with differential operator, IHSDE)[17]等。这些改进算法从一定程度上提高了算法的性能，但是在解决部分多模态复杂优化问题时，收敛速度慢，求解精度和鲁棒性仍不够理想。

HS 算法在搜索过程中能很好地维持种群的多样性，具有很强的全局搜索能力，以致它能快速发现最优解所在的区域，但其步长调整策略在进化后期盲目搜索，不能有效调整解的结构，使得和声记忆库的多样性逐渐消失，无法获得高精度的全局最优解；DE 算法由于采用“贪婪”的选择机制，具有很强的收敛能力，可以获得较高精度的解，但在处理多模态复杂优化问题时，由于极值点个数随着维度的增加而急剧增多，种群容易快速聚集，从而导致最优解丢失。针对 HS 算法和 DE 算法在处理多模态复杂优化问题时的特点，本文提出了一种混合和声差分算法 (hybrid algorithm based on HS and DE, HHSDE)。不同于已有的两种算法的融合方式，HHSDE 算法设计了一种混合自适应调整机制，在同一种群中采用累积加权更新成功率来自适应地选择用 HS 算法或 DE 算法作为下一代种群的更新方式。为验证 HHSDE 算法的性能，通过求解 10 个多模态 Benchmark 测试函数[18-19]，并与 6 种优化算法 (SaDE、CoDE、DE、IHS、DIHS、NGHS) 进行了对比，验证了所提算法的有效性和可靠性。

1　和声搜索算法

1.1　标准和声搜索算法

标准和声算法的基本步骤[5]如下：

1) 设置参数

D表示问题的维数，HMS (harmony memory size) 为和声记忆库的大小，$T_{\max}$为算法迭代的最大次数，HMCR (harmony memory considering rate) 为和声记忆库选择概率，PAR (pitch-adjusting rate) 为局部微调概率，bw 为局部微调步长 。

2) 随机初始化和声记忆库 HM (harmony memory)

$$x_i^j = xL_i + \text{rand}\cdot(xU_i - xL_i) \tag{1}$$

式中：$i = 1,2,\cdots,D$；$j = 1,2,\cdots,$HMS；rand 为(0,1)的随机数。

$$\mathbf{HMS} = \begin{bmatrix} x^1 \\ x^2 \\ \vdots \\ x^{\text{HMS}} \end{bmatrix} = \begin{bmatrix} x_1^1 & x_2^1 & \cdots & x_1^1 \\ x_1^2 & x_2^2 & \cdots & x_D^2 \\ \vdots & \vdots & & \vdots \\ x_1^{\text{HMS}} & x_2^{\text{HMS}} & \cdots & x_D^{\text{HMS}} \end{bmatrix}$$

3) 使用 3 种调节规则创作新的和声

每次迭代可通过如下 3 种规则产生新和声：

$$\boldsymbol{x}^{\text{new}} = [x_1^{\text{new}}\ x_2^{\text{new}}\ \cdots\ x_D^{\text{new}}]$$

①从和声记忆库 HM 中选择。新和声x^{new}的第i维变量$x_i^{\text{new}}(i = 1,2,\cdots,D)$以概率 HMCR 从和声记忆库的第 i 维$\{x_i^1, x_i^2, \cdots, x_i^{\text{HMS}}\}$中选择。

②局部微调。将②中产生的$x_i^{\text{new}}(i = 1,2,\cdots,D)$以概率 PAR 进行微调，如式 (2)：

$$x_i^{\text{new}} = x_i^{\text{new}} \pm \text{rand}\cdot\text{bw}(i) \tag{2}$$

③在搜索空间内随机生成。新和声x^{new}的第i维变量$x_i^{\text{new}}(i = 1,2,\cdots,D)$以概率 1 – HMCR 在搜索空间内随机生成。具体方法如下：

$$x_i^{\text{new}} = xL_i + \text{rand}\cdot(xU_i - xL_i) \tag{3}$$

4) 更新和声记忆库

对 3) 中产生的新和声x^{new}进行评估，如果优于记忆库中的最差和声x^{worst}，则用x^{new}替换x^{worst}，否则转 3)。重复 3)、4)，直到满足终止条件为止。

1.2　改进的和声算法

为改善 HS 算法的性能，文献[5]提出了改进的和声算法 (IHS)，算法动态地对参数 PAR 和 bw 进行调整。参数 PAR 随迭代的增加而线性增大，bw 呈指数地递减，迭代初期用较大的值来增加多样性，迭代后期使用较小的值来提高解的精度。

$$\text{PAR}(t) = \text{PAR}_{\min} + \frac{\text{PAR}_{\max} - \text{PAR}_{\min}}{T_{\max}} \times t \tag{4}$$

$$\text{bw}(t) = \text{bw}_{\max}\exp(t \times \ln(\frac{\text{bw}_{\min}}{\text{bw}_{\max}})/T_{\max}) \tag{5}$$

式中：$T_{\max}$为最大迭代次数，t为当前迭代次数，$\text{PAR}_{\max}$为最大微调概率，$\text{PAR}_{\min}$为最小微调概率，$\text{bw}_{\max}$为最大微调步长，$\text{bw}_{\min}$是最小微调步长。

2　差分进化算法

2.1　标准的差分进化算法

标准差分算法包括变异、交叉和选择 3 种基本

操作, 其基本步骤[2]如下:

1) 参数设置及初始化

D表示问题的维数, NP 表示种群的规模, CR 为交叉概率, F为缩放因子, $T_{\max}$为最大迭代次数。随机初始化种群$\boldsymbol{X}^0=[x_1^0\ x_2^0\ \cdots\ x_{\mathrm{NP}}^0]$, x_i^t表示第t代种群的第i个个体。

2) 变异操作

对t代种群中的每个个体x_i^t按式 (6) 进行变异操作, 得到个体:

$$v_i^t=x_{r1}^t+F\cdot(x_{r2}^t-x_{r3}^t) \tag{6}$$

式中 $r_1,r_2,r_3\in\{1,2,\cdots,\mathrm{NP}\}$互不相同且不同于$i$。

3) 交叉操作

对个体x_i^t和v_i^t进行交叉操作, 生成实验个体:

$$u_{i,j}^{t+1}=\begin{cases}v_{i,j}^t, & (\mathrm{rand}(j)\leqslant \mathrm{CR})\,\mathrm{or}\,(j=j_{\mathrm{rand}})\\ x_{i,j}^t, & \text{其他}\end{cases} \tag{7}$$

4) 选择操作

对个体x_i^t和u_i^{t+1}的适应值进行比较, 选择更优个体作为新种群的个体:

$$x_i^{t+1}=\begin{cases}u_i^{t+1}, & f(u_i^{t+1})<f(x_i^t)\\ x_i^t, & \text{其他}\end{cases} \tag{8}$$

式中$f(\cdot)$为适应值函数。

通过不断进化, 直到满足终止条件停止。

2.2　改进的差分进化算法

差分算法区别于其他优化算法之处在于差分变异算子的引用, 具有代表性的变异策略主要包括 5 种: DE/rand/1、DE/best/1、DE/c–t–b/1、DE/best/2、DE/rand/2。以上常用变异策略中, DE/rand/1 的全局搜索能力强, 但收敛速度慢, DE/best/1 的局部搜索能力强, 收敛速度快, 但容易陷入局部最优。综合考虑两种变异方式的特点, 为平衡算子的全局探索和局部开能力, 提出了改进的变异策略, 具体操如式 (9):

$$v_i^t=x_{r1}^t+F\cdot[\lambda x_{\mathrm{best}}^t+(1-\lambda)x_{r2}^t-x_{r3}^t] \tag{9}$$

当$t\leqslant T_{\max}/2$, $\lambda=0$, 式 (9) 即为 DE/rand/1, 表示在迭代的前半阶段, 算法主要进行全局搜索; 当$T_{\max}/2\leqslant t\leqslant T_{\max}$, $\lambda=1$, 即在迭代的后半阶段, 算法主要进行局部开发, 提高收敛速度和求解精度。

3　混合和声差分算法

对于不同的优化问题甚至同一问题的不同进化阶段, 最适合的进化策略都不同。针对多模态复杂优化问题, 为充分发挥 HS 算法和 DE 算法的各自优势, 本文设计了一种混合机制, 自适应地选择使用 HS 算法或 DE 算法来更新新一代种群。

3.1　算法混合机制

算法使用自适应选择因子 (selected factor, SF) 来决定在一个选择周期 (T) 内选择 HS 算法或 DE 算法作为种群更新方式的比例, 而自适应选择因子是依据一个选择周期内两种算法的加权累积成功率 (success rate, SR) 动态得到的。在第k个选择周期, 首先生成一个随机数$r^{(k)}\in(0,1)$, 如果$r^{(k)}<\mathrm{SF}^{(k)}$, 选择使用 HS 算法来更新种群; 否则, 使用 DE 算法来更新种群。

具体如下: 令$\mathrm{SR}_H^{(0)}=1$, $\mathrm{SR}_D^{(0)}=1$, $\mathrm{SF}^{(0)}=0.5$。对$k=1,2,\cdots,[T_{\max}/T]$,

$$\mathrm{SR}_H^{(k)}=\frac{c_1(k)-c_1(k-1)}{T\times\mathrm{SF}^{(K-1)}}+\rho\cdot\mathrm{SR}_H^{(k-1)}=\mathrm{SP}_H^{(k)}+\rho\cdot\mathrm{SP}_H^{(k-1)}+\cdots+\rho^{k-1}\cdot\mathrm{SP}_H^{(1)}=\sum_{i=0}^{k}\rho^i\cdot\mathrm{SP}_H^{(k-i)} \tag{10}$$

$$\mathrm{SR}_D^{(k)}=\frac{c_2(k)-c_2(k-1)}{T\times\mathrm{SF}^{(K-1)}}+\mu\cdot\mathrm{SR}_H^{(k-1)}=\mathrm{SP}_D^{(k)}+\mu\cdot\mathrm{SP}_D^{(k-1)}+\cdots+\mu^{k-1}\cdot\mathrm{SP}_D^{(1)}=\sum_{i=0}^{k}\mu^i\cdot\mathrm{SP}_D^{(k-i)} \tag{11}$$

$$\mathrm{SF}^{(k)}=\frac{\mathrm{SR}_H^{(k)}}{\mathrm{SR}_H^{(k)}+\mathrm{SR}_D^{(k)}} \tag{12}$$

式中: $T_{\max}$为最大迭代次数; T为选择周期, 即经过T次迭代后重新计算选择因子, 本文中$T=120$; ρ和μ是加权系数, 表示考虑进化过程的历史信息, 本文中$\rho=1.02$, $\mu=1$; $c_1(k)$、$c_2(k)$分别表示前k个选择周期内使用 HS 算法和 DE 算法累计更新成功的个体数目; $\mathrm{SP}_D^{(k)}$、$\mathrm{SP}_H^{(k)}$分别表示 DE 算法和 HS 算法在第k个选择周期的实际更新成功率, 即更新成功的个体数与实际产生的新个体数的比值; $\mathrm{SR}_D^{(k)}$、$\mathrm{SR}_H^{(k)}$分别表示 DE 算法和 HS 算法在第k个选择周期的实际更新成功率的加权和, 即累积加权更新成功率; $\mathrm{SF}^{(k)}$表示第k个选择周期的选择因子。

分析上述过程, 在第 1 个选择周期内, 两种算法被选择的概率相同。以后的每个周期, 兼顾考虑进化过程的当前信息和历史信息, 根据累积加权更新成功率来动态选择更新种群的方式, 哪种算法在进化过程中越活跃, 被选择的概率就越大。使用累积成功率和设置选择周期也可以防止两种更新策略退化为单一策略现象的发生。迭代初期, HS 算法因为具有优越的全局搜索能力而会获得较多的机会, 有利于快速定位最优解所在的区域; 迭代中后期, DE 算法的更新成功率增大, 主要进行精细搜索, 获得精度较高的解。两种算法在同一种群中共享信息, 相互协作, 进化早期的 DE 算法可以加快收敛速度, 后期的 HS 算法能够维持种群的多样性。

3.2 算法流程

HHSDE 算法的具体流程如图 1 所示：

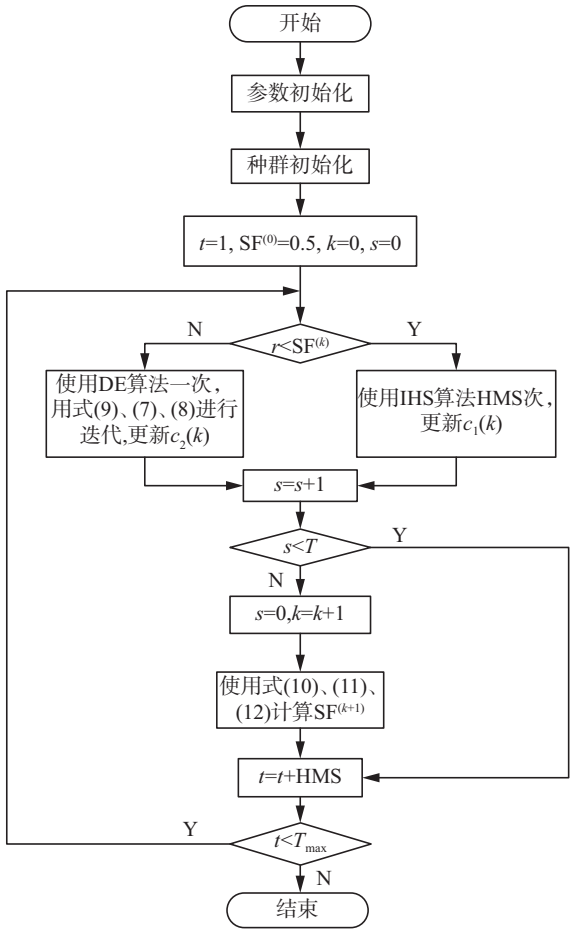

图 1　HHSDE 算法流程图

Fig. 1　The flow chart of HHSDE algorithm

4 实验仿真测试

为评估本文所提算法的性能，将其与另外 6 种优秀算法 (SaDE、CoDE、DE、HIS、DIHS、NGHS) 在 10 个多模态的 benchmark 函数[19]上比较测试。

F_1: Ackley Function;

F_2: Griewank Function;

F_3: Levy Function;

F_4: Schwefel 2.22 Function;

F_5: Schwefel 2.26 Function;

F_6: Rastrigin Function;

F_7: FastFractal‘DoubleDip’ Function;

F_8: Ackley Shift Function;

F_9: Griewank Shift Function;

F_{10}: Rastrigin Shift Function。

其中 F_1~F_7 是非常复杂的多模态函数，当维数大于 50 时，很多优秀的群智能优化算法都不能得到满意的解；F_8~F_{10} 是对 3 个经典的测试函数进行了 Shift 操作，使其计算难度大幅增加，能够更好地测试算法的通用性 (许多优秀的智能优化算法往往对这类问题存在偏好型，比如当最优解在 (0,0, …, 0) 时，求解性能非常好，反之性能变得很差)。

4.1 运行环境与参数设置

本文进行的所有测试，硬件环境为戴尔 PC 机：Intel Xeon(TM)i7-4790 3.6 GHz CPU,8 GB 内存；软件环境 Window7 操作系统，MATLAB 2014b 软件。为公平起见，本文采用相同的最大适应度函数评价次数 (MaxFEs=5 000×D)，D 为维数。6 种比较算法的参数设置参照于原文献，本文算法参数具体设置为：C_r=0.4, F=0.5, PARmax=0.99, PARmax=0.1, $\mathrm{bw}_{\max}=((x^U-x^L)/100$, $\mathrm{bw}_{\min}=(x^U-x^L)/10^{10}$, T=120 HMCR=0.98, ρ=1.02, μ=1。

4.2 实验结果与分析

表 1 ~ 2 分别展示了 D=30 和 D=100 时，10 个测试函数的 30 次独立实验统计结果，对每一个函数都记录了最优解和最差解，并统计了 30 次实验的最优平均值和运行时间。从表 1 ~ 2 可以看出，这 10 个测试函数中，HHSDE 算法除了在函数 F_4(Schwefel 2.22 Function) 上仅弱于 SaDE 外，对其他问题，HHSDE 算法的评价指标均优于或不弱于其他 6 种算法。在运行时间方面，当 D=30 时，本文算法仅比 DE 算法用时稍长，在某些问题上与 NGHS 算法相当，但远少于其他几种算法；当 D=100 时，算法用时就仅少于 DE 算法。总体来说，对这 10 个多模态复杂优化问题，用时短的算法在解的精度方面弱于本文算法，而相较于获得相同最优解的算法，本文算法的用时却最短。

为检验本文算法与比较算法之间的差异，表 3 列出了 30 次独立运算下本文算法与其他 6 种算法之间置信度为 0.05 的 Wilcoxon 符号秩检验而得到的 P-value 值，“–”, “+”, “≈”分别表示相应算法的成绩与本文算法相比是差、好、相当。(NaN 表示算法成绩类似)

从表 3 可以看出，其他 6 种算法在 10 个问题上的成绩没有优于本文算法的，DIHS, IHS 和 CoDE 分别在 6 个、4 个和 2 个问题上与本文算法的成绩相当。由此可见，本文算法相较于其他算法在统计意义上有着更好的表现。

为进一步分析实验结果，图 2~5 给出了 7 种算法在 D=80 维时 30 次独立运行的平均收敛曲线图和最优解分布盒图。从收敛曲线图不难看出，本文算法对两个复杂优化函数 (fastfractal“doubledip” function, rastrigin shift function) 的收敛曲线呈下降

趋势, 收敛精度高于其他算法, 说明本文算法的全局搜索能力较强, 不易陷入局部最优。从统计盒图可以看出, 本文算法的 30 次最优解的分布很集中, 说明了本文算法具有很强的稳定性。

表 1　算法 30 次独立运行结果比较 (D=30)

Table 1　Thirty times optimization results of the algorithm (D=30)

函数	算法	最优解	最优平均值	最差解	运行时间/s	函数	算法	最优解	最优平均值	最差解	运行时间/s
	SaDE	8.44×10^{-15}	9.57×10^{-1}	2.49×10^{0}	10.253 3		SaDE	0.00	7.46×10^{-1}	1.99×10^{0}	9.595 1
	CoDE	2.61×10^{-7}	5.16×10^{-7}	9.86×10^{-7}	3.335 9		CoDE	1.63×10^{1}	2.06×10^{1}	2.45×10^{1}	3.643 2
	DE	7.99×10^{-15}	1.07×10^{-14}	1.15×10^{-14}	2.214 8		DE	1.02×10^{2}	1.14×10^{2}	1.25×10^{2}	2.204 3
F_1	IHS	1.22×10^{-13}	1.55×10^{-13}	1.64×10^{-13}	3.268 3	F_6	IHS	0.00	2.37×10^{-2}	1.59×10^{-1}	3.286 1
	DIHS	2.22×10^{-10}	2.91×10^{-10}	3.33×10^{-10}	2.885 4		DIHS	0.00	1.89×10^{-3}	3.71×10^{-2}	3.708 8
	NGHS	1.12×10^{-12}	4.67×10^{-12}	1.21×10^{-11}	2.377 5		NGHS	5.68×10^{-14}	1.76×10^{-13}	3.41×10^{-13}	2.366 3
	HHSDE	7.99×10^{-15}	8.35×10^{-15}	1.15×10^{-14}	2.345 4		HHSDE	0.00	0.00	0.00	2.430 6
	SaDE	0.00	1.87×10^{-2}	6.37×10^{-2}	10.345		SaDE	0.00	0.00	0.00	9.726 7
	CoDE	2.12×10^{-12}	5.71×10^{-8}	1.12×10^{-6}	3.495 0		CoDE	3.20×10^{-7}	6.36×10^{-6}	1.84×10^{-5}	4.032 6
	DE	0.00	0.00	0.00	2.358 6		DE	7.77×10^{0}	1.27×10^{1}	1.77×10^{1}	2.144 9
F_2	IHS	0.00	1.25×10^{-2}	4.19×10^{-2}	3.371 3	F_7	IHS	0.00	0.00	0.00	3.195 8
	DIHS	0.00	0.00	0.00	3.110 0		DIHS	0.00	0.00	0.00	3.727 3
	NGHS	0.00	4.13×10^{-2}	2.47×10^{-1}	2.442 2		NGHS	5.68×10^{-14}	1.76×10^{-13}	4.55×10^{-13}	2.390 2
	HHSDE	0.00	0.00	0.00	2.508 5		HHSDE	0.00	0.00	0.00	2.442 8
	SaDE	1.50×10^{-32}	1.49×10^{-1}	9.98×10^{-1}	12.061 7		SaDE	0.00	8.87×10^{-1}	2.01×10^{0}	12.464
	CoDE	2.01×10^{-14}	2.25×10^{-13}	8.91×10^{-13}	5.119 1		CoDE	1.75×10^{-7}	4.23×10^{-7}	7.90×10^{-7}	5.784 0
	DE	5.64×10^{-30}	3.22×10^{-29}	1.53×10^{-28}	4.048 6		DE	7.11×10^{-15}	1.01×10^{-14}	1.07×10^{-14}	4.627 5
F_3	IHS	1.92×10^{-27}	2.36×10^{-27}	2.83×10^{-27}	5.196 3	F_8	IHS	1.42×10^{-13}	1.34×10^{-2}	1.76×10^{-1}	6.167 7
	DIHS	5.28×10^{-21}	8.10×10^{-21}	1.14×10^{-20}	4.777 6		DIHS	2.54×10^{-10}	2.89×10^{-10}	3.26×10^{-10}	6.795 8
	NGHS	1.54×10^{-25}	4.80×10^{-24}	3.11×10^{-23}	4.243 0		NGHS	2.94×10^{-12}	7.38×10^{-12}	2.48×10^{-11}	5.334 5
	HHSDE	1.50×10^{-32}	4.55×10^{-30}	5.64×10^{-29}	4.280 2		HHSDE	3.55×10^{-15}	6.57×10^{-15}	1.07×10^{-14}	5.380 9
	SaDE	6.38×10^{-58}	2.19×10^{-53}	2.52×10^{-52}	9.8259		SaDE	0.00	2.39×10^{-2}	1.27×10^{-1}	12.659
	CoDE	8.75×10^{-8}	1.77×10^{-7}	3.03×10^{-7}	3.322 7		CoDE	9.75×10^{-13}	2.70×10^{-11}	3.34×10^{-10}	5.972 8
	DE	6.90×10^{-18}	9.76×10^{-18}	1.55×10^{-17}	2.132 9		DE	0.00	0.00	0.00	4.767 1
F_4	IHS	1.14×10^{-13}	1.43×10^{-13}	1.55×10^{-13}	3.089 7	F_9	IHS	0.00	1.48×10^{-2}	5.66×10^{-2}	6.166 6
	DIHS	1.83×10^{-10}	2.35×10^{-10}	2.99×10^{-10}	2.796 6		DIHS	0.00	0.00	0.00	6.781 0
	NGHS	6.64×10^{-13}	2.16×10^{-12}	8.73×10^{-12}	2.223 4		NGHS	0.00	4.57×10^{-2}	1.25×10^{-1}	5.441 0
	HHSDE	3.68×10^{-19}	1.20×10^{-20}	2.92×10^{-18}	2.346 3		HHSDE	0.00	0.00	0.00	5.313 0
	SaDE	7.28×10^{-12}	2.96×10^{1}	1.18×10^{2}	9.531 7		SaDE	9.95×10^{-1}	3.33×10^{0}	8.95×10^{0}	11.552 6
	CoDE	1.43×10^{-6}	1.00×10^{-5}	3.35×10^{-5}	3.533 6		CoDE	7.98×10^{0}	1.16×10^{1}	1.49×10^{1}	5.677 8
	DE	1.13×10^{3}	1.90×10^{3}	2.35×10^{3}	1.784 4		DE	9.19×10^{1}	1.03×10^{2}	1.16×10^{2}	3.776 2
F_5	IHS	7.28×10^{-12}	7.28×10^{-12}	7.28×10^{-12}	2.713 7	F_{10}	IHS	0.00	1.46×10^{-2}	9.78×10^{-2}	5.161 7
	DIHS	7.28×10^{-12}	7.28×10^{-12}	7.28×10^{-12}	3.284 4		DIHS	0.00	1.03×10^{-4}	2.07×10^{-3}	5.652 5
	NGHS	9.09×10^{-12}	1.17×10^{-11}	1.64×10^{-11}	1.935 4		NGHS	0.00	0.00	0.00	4.308 0
	HHSDE	7.28×10^{-12}	7.28×10^{-12}	7.28×10^{-12}	2.037 3		HHSDE	0.00	0.00	0.00	4.281 2

表 2　算法 30 次独立运行结果比较 (*D*=100)

Table 2　Thirty times optimization results of the algorithm (*D*=100)

函数	算法	最优解	最优平均值	最差解	运行时间/s	函数	算法	最优解	最优平均值	最差解	运行时间/s
	SaDE	3.03×10^{0}	4.44×10^{0}	5.54×10^{0}	37.056 1		SaDE	1.79×10^{1}	2.99×10^{1}	4.18×10^{1}	32.654 4
	CoDE	5.16×10^{-12}	1.17×10^{-11}	2.48×10^{-11}	13.466 9		CoDE	2.20×10^{2}	2.52×10^{2}	2.92×10^{2}	16.801 5
	DE	6.71×10^{-6}	1.30×10^{-5}	2.98×10^{-5}	10.465 6		DE	7.28×10^{2}	8.09×10^{2}	8.41×10^{2}	10.513 8
F_1	IHS	2.35×10^{-13}	2.55×10^{-13}	2.71×10^{-13}	17.536 8	F_6	IHS	0.00	7.46×10^{-3}	4.01×10^{-2}	17.518 2
	DIHS_	4.16×10^{-10}	4.41×10^{-10}	4.63×10^{-10}	18.549 2		DIHS_	0.00	0.00	0.00	20.704 4
	NGHS	2.14×10^{-4}	2.94×10^{-4}	3.90×10^{-4}	13.357 9		NGHS	3.05×10^{-6}	5.93×10^{-6}	1.69×10^{-5}	13.939 9
	HHSDE	2.58×10^{-14}	2.98×10^{-14}	3.29×10^{-14}	11.866 8		HHSDE	0.00	0.00	0.00	13.128 3
	SaDE	0.00	1.32×10^{-1}	1.59×10^{0}	37.617 9		SaDE	0.00	4.97×10^{-2}	9.95×10^{-1}	32.856 8
	CoDE	0.00	0.00	0.00	14.236 7		CoDE	5.16×10^{1}	5.92×10^{1}	6.90×10^{1}	18.707 2
	DE	4.51×10^{-10}	1.03×10^{-9}	1.61×10^{-9}	11.268 4		DE	1.05×10^{2}	1.11×10^{2}	1.20×10^{2}	10.281 5
F_2	IHS	0.00	0.00	0.00	18.301 1	F_7	IHS	0.00	0.00	0.00	17.344 7
	DIHS_	0.00	0.00	0.00	19.328 5		DIHS_	0.00	0.00	0.00	20.660 0
	NGHS	1.69×10^{-6}	7.74×10^{-3}	5.14×10^{-2}	13.702 8		NGHS	6.86×10^{-8}	1.23×10^{-7}	1.93×10^{-7}	12.861 9
	HHSDE	0.00	0.00	0.00	12.811 3		HHSDE	0.00	0.00	0.00	12.260 2
	SaDE	2.51×10^{0}	5.08×10^{0}	1.07×10^{1}	54.6495		SaDE	1.60×10^{0}	3.71×10^{0}	5.24×10^{0}	39.9545
	CoDE	1.03×10^{-22}	4.96×10^{-2}	4.54×10^{-1}	30.571 1		CoDE	8.07×10^{-12}	1.72×10^{-11}	3.09×10^{-11}	20.173 3
	DE	3.24×10^{-8}	1.16×10^{-7}	5.10×10^{-7}	27.482 4		DE	2.31×10^{-6}	3.78×10^{-6}	5.12×10^{-6}	17.050 1
F_3	IHS	1.70×10^{-26}	1.98×10^{-26}	2.29×10^{-26}	35.436 0	F_8	IHS	2.20×10^{-13}	2.57×10^{-13}	2.74×10^{-13}	26.371 9
	DIHS_	5.48×10^{-20}	6.09×10^{-20}	7.60×10^{-20}	35.581 1		DIHS_	4.14×10^{-10}	4.41×10^{-10}	4.65×10^{-10}	29.941 3
	NGHS	1.29×10^{-8}	2.62×10^{-8}	3.57×10^{-8}	30.334 4		NGHS	2.26×10^{-4}	3.08×10^{-4}	3.97×10^{-4}	21.913 2
	HHSDE	1.80×10^{-29}	3.66×10^{-28}	2.87×10^{-27}	29.274 4		HHSDE	2.49×10^{-14}	2.84×10^{-14}	3.20×10^{-14}	20.390 9
	SaDE	7.35×10^{-27}	1.71×10^{-20}	2.80×10^{-19}	35.167 0		SaDE	0.00	7.52×10^{-2}	4.59×10^{-1}	47.156 3
	CoDE	3.40×10^{-12}	6.58×10^{-12}	1.12×10^{-11}	12.560 7		CoDE	0.00	6.16×10^{-4}	1.23×10^{-2}	26.651 2
	DE	8.77×10^{-7}	1.57×10^{-6}	2.27×10^{-6}	9.048 9		DE	8.27×10^{-11}	1.81×10^{-10}	2.88×10^{-10}	23.562 7
F_4	IHS	7.38×10^{-13}	7.75×10^{-13}	8.16×10^{-13}	16.308 7	F_9	IHS	0.00	0.00	0.00	31.454 6
	DIHS_	1.04×10^{-9}	1.14×10^{-9}	1.20×10^{-9}	16.903 2		DIHS_	0.00	0.00	0.00	35.264 8
	NGHS	4.34×10^{-4}	6.75×10^{-4}	9.80×10^{-4}	11.764 2		NGHS	1.63×10^{-6}	4.81×10^{-3}	1.72×10^{-2}	27.501 6
	HHSDE	5.96×10^{-19}	2.23×10^{-18}	5.24×10^{-18}	10.961 5		HHSDE	0.00	0.00	0.00	25.705 0
	SaDE	5.72×10^{2}	1.09×10^{3}	1.76×10^{3}	31.838 6		SaDE	6.47×10^{1}	9.06×10^{1}	1.37×10^{2}	38.640 0
	CoDE	3.85×10^{3}	5.94×10^{3}	7.71×10^{3}	16.501 1		CoDE	1.40×10^{2}	1.89×10^{2}	2.22×10^{2}	22.272 6
	DE	9.16×10^{3}	9.80×10^{3}	1.02×10^{4}	8.175 1		DE	6.77×10^{2}	7.02×10^{2}	7.31×10^{2}	14.920 9
F_5	IHS	1.31×10^{-10}	1.31×10^{-10}	1.31×10^{-10}	15.275 5	F_{10}	IHS	0.00	5.22×10^{-3}	7.66×10^{-2}	23.020 5
	DIHS_	1.31×10^{-10}	1.31×10^{-10}	1.31×10^{-10}	18.713 5		DIHS_	0.00	2.01×10^{-4}	2.38×10^{-3}	26.475 0
	NGHS	1.07×10^{-5}	2.42×10^{-5}	4.82×10^{-5}	10.892 7		NGHS	2.66×10^{-6}	5.47×10^{-6}	1.09×10^{-5}	18.421 1
	HHSDE	1.31×10^{-10}	1.31×10^{-10}	1.31×10^{-10}	10.269 7		HHSDE	0.00	0.00	0.00	18.096 3

表 3　标准测试函数 30 次 Wilcoxon 符号秩检验的 P-value(D=100)

Table 3　Thirty times P-values of Wilcoxon signed rank test (D=100)

函数	DIHS	NGHS	IHS	DE	CoDE	SaDE
F_1	1.04×10^{-8}	1.04×10^{-8}	1.04×10^{-8}	1.04×10^{-8}	1.04×10^{-8}	1.04×10^{-8}
F_2	NaN	1.12×10^{-4}	NaN	1.22×10^{-9}	NaN	1.22×10^{-9}
F_3	1.21×10^{-8}	1.21×10^{-8}	1.21×10^{-8}	1.21×10^{-8}	2.92×10^{-4}	1.21×10^{-8}
F_4	1.21×10^{-8}	1.21×10^{-8}	1.21×10^{-8}	1.21×10^{-8}	1.21×10^{-8}	1.21×10^{-8}
F_5	NaN	1.22×10^{-9}	NaN	1.22×10^{-9}	1.22×10^{-9}	1.22×10^{-9}
F_6	NaN	1.22×10^{-9}	1.12×10^{-4}	1.22×10^{-9}	1.22×10^{-9}	1.22×10^{-9}
F_7	NaN	1.22×10^{-9}	NaN	1.22×10^{-9}	1.22×10^{-9}	NaN
F_8	1.22×10^{-9}	1.22×10^{-9}	1.22×10^{-9}	1.22×10^{-9}	1.22×10^{-9}	1.22×10^{-9}
F_9	NaN	1.12×10^{-4}	NaN	1.22×10^{-9}	NaN	3.82×10^{-3}
F_{10}	NaN	1.22×10^{-9}	3.82×10^{-3}	1.22×10^{-9}	1.22×10^{-9}	1.22×10^{-9}
–	4	10	6	10	8	9
+	0	0	0	0	0	0
≈	6	0	4	0	2	1

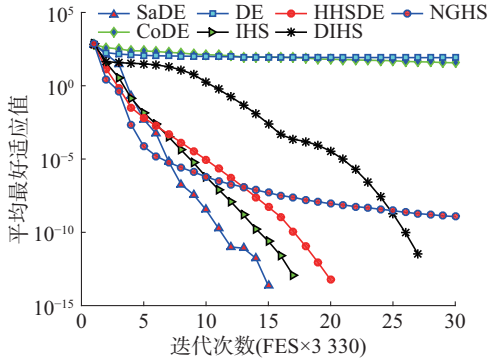

图 2　F_7 函数的收敛曲线图

Fig. 2　Convergence curves' comparison of F_7

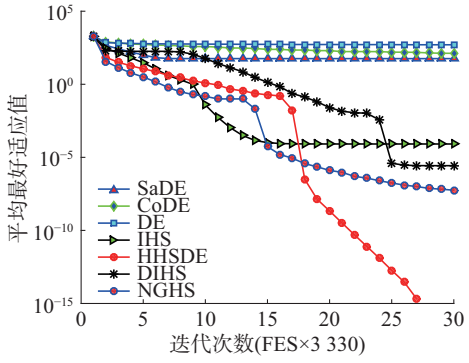

图 3　F_{10} 函数的收敛曲线图

Fig. 3　Convergence curves' comparison of F_{10}

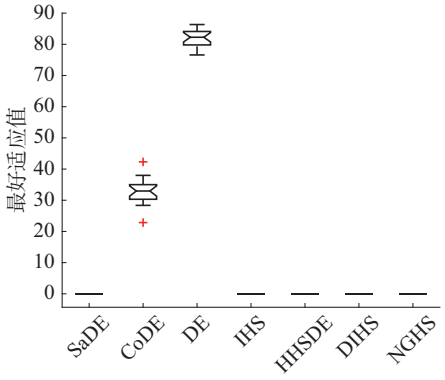

图 4　F_7 函数的统计盒图

Fig. 4　Boxplot of F_7

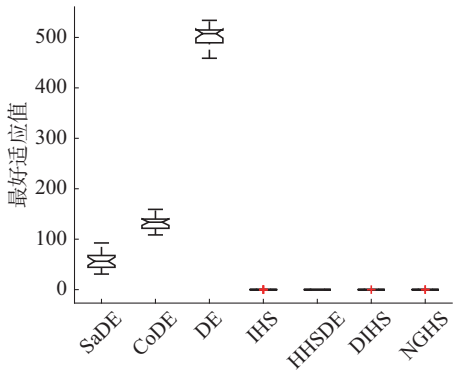

图 5　F_{10} 函数的统计盒图

Fig. 5　Boxplot of F_{10}

4.3　HHSDE 算法的混合机制分析

为分析本文算法的自适应混合机制, 跟踪记录了 HHSDE 算法中的 IHS 和 DE 两种策略的更新成功率。图 6 和图 7 分别绘制了 D=100 维时两种策略的累积更新个体数目图和更新成功率曲线。从图 6 可以看出, 在迭代初期, IHS 算法累积成功更新

的个体数量较多，可以快速定位最优解的区域，但随着迭代的进行，DE 算法累积更新的个体数目迅速增多，主要进行深度开发，提高解的精度。从图 7 以看出，IHS 算法的概率随着进化的进行逐渐减小，而 DE 算法被选择的概率逐渐增大。

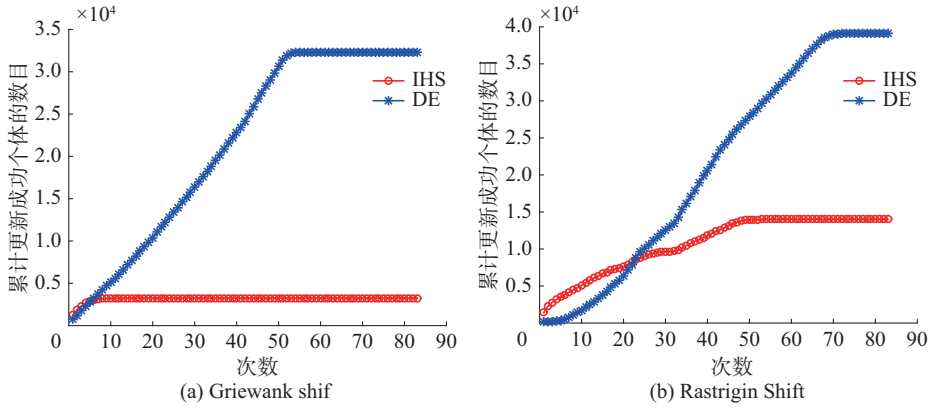

图 6　两种更新策略的累积更新个体数目变化曲线

Fig. 6　The change curves of cumulative update individuals for the two kinds of update strategy

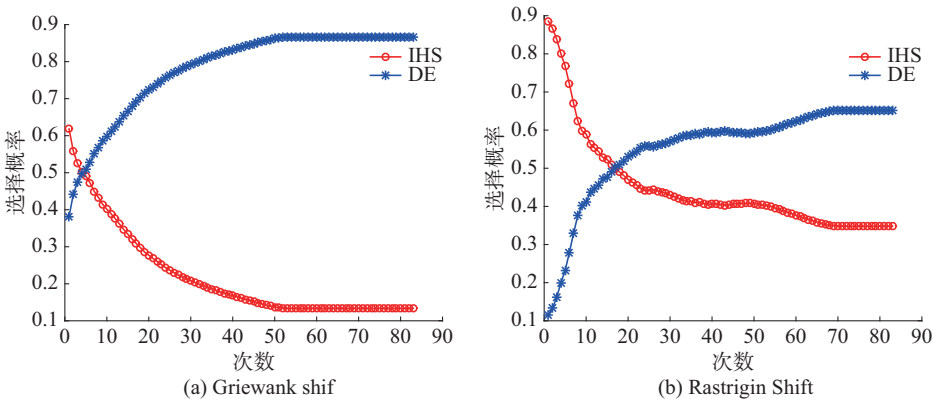

图 7　两种更新策略选择概率变化曲线

Fig. 7　The change curves of select probability for the two kinds of update strategy

所提算法能根据寻优问题的特点，针对不同难易程度的优化对象，依据进化过程的历史经验自适应地选择合适的更新策略来满足不同进化阶段的要求，动态调节两种进化策略的选择比例。对 Griewank shif，IHS 算法能快速定位最优解所在区域，然后主要使用 DE 算法进行局部搜索，所以两种算法在整个进化过程中的选择概率差异较大；Rastrigin Shift 比 Griewank shif 更复杂，存在更多的局部极小值，进化过程中需要不断地跳出局部极值，从而 IHS 算法的选择概率下降得较慢，整个进化过程中两种算法的选择概率差异较小。

5　结束语

针对多模态复杂优化问题，提出了一种混合和声差分算法——HHSDE 算法。算法通过在不同进化阶段依据累积加权更新成功率来自适应地选择 IHS 算法和 DE 算法作为更新种群的方式，能够有效地平衡进化过程的全局搜索与局部搜索。利用 10 个复杂多模态 Benchmark 函数对 HHSDE 算法和其他 6 种优秀算法进行仿真比较，实验结果和统计分析表明，在 100 维以内，HHSDE 算法收敛速度快，求解精度高，算法稳定性好，能有效求解多模态复杂优化问题。但由于混合算法采用了两种进化机制，参数较多，同时对超过 200 维的复杂问题，优化效果也不尽理想，在后续的研究过程中，可以设计更好的混合机制来解决更高维的复杂优化问题。

参考文献：

[1] TRELEA I C. The particle swarm optimization algorithm: convergence analysis and parameter selection[J]. Information processing letters, 2003, 85(6): 317–325.

[2] DAS S, SUGANTHAN P N. Differential evolution: a survey of the state-of-the-art[J]. IEEE transactions on evolutionary computation, 2011, 15(1): 4–31.

[3] DASGUPTA K, MANDAL B, DUTTA P, et al. A genetic algorithm (GA) based load balancing strategy for cloud computing[J]. Procedia technology, 2013, 10: 340–347.

[4] DEEPA O, SENTHILKUMAR A. Swarm intelligence from natural to artificial systems: ant colony optimization[J]. International journal on applications of graph theory in wireless Ad hoc networks and sensor networks, 2016, 8(1): 9–17.

[5] MAHDAVI M, FESANGHARY M, DAMANGIR E. An improved harmony search algorithm for solving optimization problems[J]. Applied mathematics and computation, 2007, 188(2): 1567–1579.

[6] ZOU Dexuan, GAO Liqun, WU Jianhua, et al. Novel global harmony search algorithm for unconstrained problems[J]. Neurocomputing, 2010, 73.

[7] 夏红刚, 欧阳海滨, 高立群. 多子群混合和声搜索算法[J]. 东北大学学报: 自然科学版, 2015, 36(2): 171–175, 187.
XIA Honggang, OUYANG Haibin, GAO Liqun. Multiple-sub-groups hybrid harmony search algorithm[J]. Journal of Northeastern university: natural science, 2015, 36(2): 171–175, 187.

[8] 拓守恒, 雍龙泉, 邓方安. 动态调整策略改进的和声搜索算法[J]. 智能系统学报, 2015, 10(2): 307–315.
TUO Shouheng, YONG Longquan, DENG Fang'an. Dynamic adjustment strategy for improving the harmony search algorithm[J]. CAAI transactions on intelligent systems, 2015, 10(2): 307–315.

[9] 夏红刚, 欧阳海滨, 高立群, 等. 全局竞争和声搜索算法[J]. 控制与决策, 2016, 31(2): 310–316.
XIA Honggang, OUYANG Haibin, GAO Liqun, et al. Global competitive harmony search algorithm[J]. Control and decision, 2016, 31(2): 310–316.

[10] WANG Yong, CAI Zixing, ZHANG Qingfu. Differential evolution with composite trial vector generation strategies and control parameters[J]. IEEE transactions on evolutionary computation, 2011, 15(1): 55–66.

[11] QIN A K, HUANG V L, SUGANTHAN P N. Differential evolution algorithm with strategy adaptation for global numerical optimization[J]. IEEE transactions on evolutionary computation, 2009, 13(2): 398–417.

[12] 李荣雨, 陈庆倩, 陈菲尔. 改进种群多样性的双变异差分进化算法[J]. 运筹学学报, 2017, 21(1): 44–54.
LI Rongyu, CHEN Qingqian, CHEN Feier. Differential evolution algorithm with double mutation strategies for improving population diversity[J]. Operations research transactions, 2017, 21(1): 44–54.

[13] WANG Ling, LI Lingpo. A coevolutionary differential evolution with harmony search for reliability-redundancy optimization[J]. Expert systems with applications, 2012, 39(5): 5271–5278.

[14] ARUL R, RAVI G, VELUSAMI S. Chaotic self-adaptive differential harmony search algorithm based dynamic economic dispatch[J]. International journal of electrical power and energy systems, 2013, 50: 85–96.

[15] 雍龙泉, 刘三阳, 张建科, 等. 基于差分算子的和声搜索算法求解非线性 l1 模极小化问题[J]. 兰州大学学报: 自然科学版, 2013, 49(4): 541–546.
YONG Longquan, LIU Sanyang, ZHANG Jianke, et al. Improved harmony search algorithm with differential operator for nonlinear l1 norm minimization problems[J]. Journal of Lanzhou university: natural sciences, 2013, 49(4): 541–546.

[16] YONG Longquan, LIU Sanyang. An improved harmony search algorithm with differential operator for absolute value equation[J]. ICIC express letters, 2014, 8(4): 1151–1157.

[17] ABEDINPOURSHOTORBAN H, HASAN S, SHAMSUDDIN S M, et al. A differential-based harmony search algorithm for the optimization of continuous problems[J]. Expert systems with applications, 2016, 62: 317–332.

[18] TANG K, YAO X, SUGANTHAN P N, et al. Benchmark functions for the CEC'2008 special session and competition on large scale global optimization[R]. Technical Report. China: Nature Inspired Computation and Applications Laboratory, USTC, 2007.

[19] TANG Ke, LI Xiaohong, SUGANTHAN P N, et al. Benchmark functions for the CEC'2010 special session and competition on large-scale global optimization[R]. Technical Report. Nanyang: Nature Inspired Computation and Applications Laboratory, USTC, China and Nanyang Technological University, 2009.

作者简介:

黎延海, 男, 1981 年生, 讲师, 硕士, 主要研究方向为智能优化算法及应用。

拓守恒, 男, 1978 年生, 副教授, 博士研究生, CCF 会员, 主要研究方向为智能优化算法、生物信息分析与处理, 发表学术论文多篇。