文章编号: 1001-0920(2019)08-1654-09 DOI: 10.13195/j.kzyjc.2018.0006

基于精英混沌搜索策略的交替正余弦算法

郭文艳†, 王 远, 戴 芳, 刘 婷

(西安理工大学 理学院，西安 710054)

摘 要: 正余弦算法是一种新的基于种群的随机寻优方法,利用正余弦函数使解震荡性地趋于全局最优解,其线性调整策略及较弱的局部搜索能力严重地影响了算法的性能. 为了提高正弦余弦算法的计算精度,提出基于精英混沌搜索策略的交替正余弦算法. 新算法采用基于对数曲线的非线性调整策略修改控制参数,利用精英个体的混沌搜索策略增强算法的开发能力,并将基于该策略的正余弦算法与反向学习算法交替执行增强算法的探索能力,降低算法的时间复杂度,提高算法的收敛速度. 对 23 个基准测试函数进行仿真实验,与改进的正余弦算法以及最新的基于启发式的算法进行比较,深入的参数实验分析以及比较结果验证了所提出算法的有效性,统计分析证实了所提出算法的优越性.

关键词: 正余弦算法；混沌搜索；非线性策略；反向学习；粒子群优化；灰狼优化

中图分类号: TP301.6 **文献标志码:** A

Alternating sine cosine algorithm based on elite chaotic search strategy

GUO Wen-yan†, WANG Yuan, DAI Fang, LIU Ting

(School of Science，Xi'an University of Technology，Xi'an 710054，China)

Abstract: The sine cosine algorithm (SCA) is a new population-based stochastic optimization method. It uses sine and cosine functions to fluctuate the solution run to the global optimal solution. Its linear adjustment strategy and weak local search ability seriously affect the performance of the algorithm. In order to improve the calculation accuracy of the sine cosine algorithm, an alternating sine cosine algorithm based on the elite chaotic search strategy is proposed, which uses the nonlinear adjustment strategy based on logarithmic curve to modify the control parameters, uses the elite individuals' chaotic search strategy to enhance the exploitation ability of the algorithm. The SCA based on this strategy and the opposition-based learning algorithm are alternately implemented to enhance the exploration ability, reduce the time complexity and improve the convergence speed of the algorithm. The proposed method has been tested by 23 benchmark test functions, and compared with the improved SCA and the state-of-the-art heuristic algorithm. The comprehensive parameter experiment and results analysis show the effectiveness and superiority of the proposed algorithm.

Keywords: sine cosine algorithm；chaotic search；nonlinear strategy；opposition-based learning；particle swarm optimization；grey wolf optimization

0 引 言

基于种群的随机优化算法是求解优化问题的重要方法之一,也是求解全局优化问题的有效方法,受到了国内外众多学者的关注,在图像处理、数据处理、工程应用等领域得到了广泛应用[1-2]. 通常对于一组随机产生的初始种群,利用不同的数学方法或随机操作在解搜索空间内对其进行演化更新,在优化过程中实现信息互换,能方便地从不同搜索空间获取信息,兼顾了算法的全局搜索能力和局部开发能力. 基于种群的优化算法随机地寻找优化问题的最优解,不能保证一次运行便能找到解,但是,随着种群规模和迭代次数的增加,找到全局最优解的概率会增大. 近年来,有效的群智能优化算法有粒子群优化算法[1](particle swarm optimization, PSO)和灰狼优化算法[2](grey wolf optimization, GWO)等.

正余弦算法[3](sine cosine algorithm, SCA)于 2016 年提出,利用正余弦函数的数学模型使解震荡性地趋于最优解,算法中随机性参数和自适应参数较好地平衡了算法的开发和探索能力. SCA 具有参数少、易实现、结构简单、收敛速度快等优点,但也存

收稿日期: 2018-01-02；**修回日期:** 2018-03-27.
基金项目: 国家自然科学基金项目(61772416, 11601419).
责任编委: 孙秋野.
†通讯作者. E-mail: wyguo@xaut.edu.cn.

在着计算精度低、易早熟收敛等缺点.

为了改进SCA的性能,文献[4]将反向学习策略融入SCA的初始化过程和更新过程,提出了基于反向学习的SCA(OBSCA),提高了SCA的精度和性能. 为了改进SCA的计算精度,本文对SCA作了以下3个方面的改进: 1)采用非线性对数曲线参数调整策略,平衡算法的开发与探索能力; 2)利用混沌序列的随机性、遍历性以及规律性特点,设计一种基于精英个体的混沌搜索策略,增强算法的局部搜索能力; 3)采用基于精英混沌搜索策略的SCA与反向策略交替进行的方法增强解的多样性,降低算法的复杂度,减少算法运行时间. 23个基准测试函数的实验结果表明,改进的SCA能够有效地协调探索和开发能力,并且在收敛精度与收敛速度上优于对比算法.

1　正弦余弦算法

对于n维最小化问题

$$\min f(z)=\min f(z_1,z_2,\cdots,z_n);$$
$$\text{s.t. } a_i\leqslant z_i\leqslant b_i,\ i=1,2,\cdots,n.$$

其中: x_i为优化变量, a_i、b_i为其上下界.

SCA的求解思路是,设种群规模为N,首先在n维解空间中随机产生N个个体$x_1,x_2,\cdots,x_N$,第i个个体的位置表示为$x_i=(x_{i1},x_{i2},\cdots,x_{in}),i=1,2,\cdots,N$. 通过目标函数计算个体的适应度值$f(x_i)$,记录种群中最优个体位置$x_*: x_*=\text{argmin}f(x_i),i=1,2,\cdots,N$. 种群中第$i$个个体第$j$维按下式进行更新:

$$x_{ij}^{t+1}=\begin{cases}x_{ij}^t+r_1\cdot\sin(r_2)|r_3\cdot x_{*j}^t-x_{ij}^t|,\ r_4<0.5;\\ x_{ij}^t+r_1\cdot\cos(r_2)|r_3\cdot x_{*j}^t-x_{ij}^t|,\ r_4\geqslant 0.5.\end{cases}\tag{1}$$

其中: $r_2\in(0,2\pi),r_3\in(0,2),r_4\in(0,1)$为均匀分布的随机数; x_*^t为当前最优个体位置; r_1为控制参数,表示为

$$r_1=a\Big(1-\frac{t}{t_{\max}}\Big).\tag{2}$$

a为常数; t、$t_{\max}$分别为当前迭代次数和最大化迭代次数,利用贪婪搜索更新种群最优个体位置.

2　基于精英混沌搜索策略的交替正余弦算法

2.1　控制参数的非线性调整策略

在SCA中,参数r_1起着平衡全局探索能力和局部搜索能力的重要作用. 当$r_1<1$时使下一代的解位于当前解与目标解之间,强调算法的开发能力; 当$r_1>1$时解位于当前解和目标解之外,强调算法的探索能力. 由式(2)可以看出, r_1随迭代次数的增加线性递减,在算法迭代初期, r_1的值较大,使算法具有较强的全局探索能力,在算法迭代后期, r_1的值较小,增加了算法的开发能力. 但是,线性变化方式不能有效地增加SCA的寻优精度,本文通过非线性函数[5]对r_1进行改进. 利用自然对数在$[1,e]$上非线性递增的特性, r_1的调整策略为

$$r_1(t)=a_{\text{start}}-(a_{\text{start}}-a_{\text{end}})\ln\Big(1+\frac{(e-1)}{\eta}\cdot\frac{t}{t_{\max}}\Big).\tag{3}$$

其中: a_{start}、a_{end}为控制参数a的初始值和最终值, $a_{\text{start}}>a_{\text{end}}\geqslant 0;\eta>0$为调节系数. 由式(3)可以看出, r_1随着迭代次数的增加非线性变化,可以有效地平衡SCA的探索和开发能力,提高算法的寻优能力和收敛速度.

2.2　反向学习策略略

反向学习(OBL)[6]是增强随机优化算法性能的重要方法,通过贪婪选择目标函数在当前解和反向解种群的适应度值,增强了种群的多样性,提高了算法接近全局最优解的能力. 文献[4]提出的OBSCA在基本SCA中加入反向学习策略,提高了SCA的求解精度. 本文旨在将SCA与反向学习策略交替进行,以降低算法的复杂性,从而在改进种群多样性的同时提高算法的收敛速度.

定义1　反向解. 设第t代种群中第i个个体为$x_i^t=(x_{i1}^t,x_{i2}^t,\cdots,x_{in}^t)$,记$a_j$、$b_j$为搜索空间中第$j$维的下界和上界,令

$$\bar{x}_{ij}^t=a_j+b_j-x_{ij}^t.\tag{4}$$

称$\bar{x}_i^t=(\bar{x}_{i1}^t,\bar{x}_{i2}^t,\cdots,\bar{x}_{in}^t)$为其反向解.

由式(4)可知,反向学习算法通过反向策略产生N个反向解,使算法在$2N$个个体中挑选优秀个体进入下一代,增强了种群的多样性和全局探索能力,同时贪婪选择策略又增加了算法的收敛速度.

2.3　精英混沌搜索策略

群智能算法的优劣在于有效地平衡算法的探索与开发能力,为了提高算法的开发能力,利用混沌的随机性、遍历性和规律性的特点,对精英个体[7]进行混沌变异. 对种群中N个个体按适应度值升序排序,选择前$m=\text{pr}\cdot N$个个体作为精英个体, $\text{pr}\in[0,1]$为选择比例,记精英个体为$\{\text{ex}_1^t,\text{ex}_2^t,\cdots,\text{ex}_m^t\}\subset\{x_1^t,x_2^t,\cdots,x_N^t\}$, $\text{ex}_i^t=(\text{ex}_{i1}^t,\text{ex}_{i2}^t,\cdots,\text{ex}_{in}^t)$, $i=1,2,\cdots,m$. 其第$j(j=1,2,\cdots,n)$维的上下界[8]分别为

$$\begin{cases} \mathrm{eb}_j^t = \max(\mathrm{ex}_{1j}^t, \mathrm{ex}_{2j}^t, \cdots, \mathrm{ex}_{mj}^t), \\ \mathrm{ea}_j^t = \min(\mathrm{ex}_{1j}^t, \mathrm{ex}_{2j}^t, \cdots, \mathrm{ex}_{mj}^t). \end{cases} \tag{5}$$

将精英个体ex_i^t由解空间按式(6)映射到$[0,1]$,得到混沌变量$C_i^t=(C_{i1}^t, C_{i2}^t, \cdots, C_{in}^t)$,有

$$C_{ij}^t = \frac{\mathrm{ex}_{ij}^t - \mathrm{ea}_j^t}{\mathrm{eb}_j^t - \mathrm{ea}_j^t}. \tag{6}$$

利用式(7)对C_{ij}^t进行混沌迭代,有

$$C_{ij}^t(k+1) = \mu C_{ij}^t(k)(1 - C_{ij}^t(k)). \tag{7}$$

其中:μ为常数,取值为4;k为混沌迭代次数,$0 \leqslant k \leqslant \mathrm{ceil}(t/10)$,$\mathrm{ceil}(x)$表示对$x$向上取整.

当k达到最大迭代次数$k_{\max}$时,得到第i个精英个体对应的混沌变量$C_i^{k_{\max}}$,利用下式将其映射至$[\mathrm{ea}_j^t, \mathrm{eb}_j^t]$内,得到对应混沌个体$xC_i^t$,有

$$xC_{ij}^t = C_{ij}^{k_{\max}}(\mathrm{eb}_j^t - \mathrm{ea}_j^t) + \mathrm{ea}_j^t. \tag{8}$$

对混沌个体xC_i^t与原个体ex_i^t,由下式生成候选解:

$$x_i^t(\lambda) = \lambda \mathrm{ex}_i^t + (1-\lambda)xC_i^t, \tag{9}$$

其中λ为收缩因子,其值为

$$\lambda = \frac{t_{\max} - t}{t_{\max}}. \tag{10}$$

在得到的候选解$x_i^t(\lambda)$与对应精英个体ex_i^t之间采用贪婪选择策略,选择优秀的个体$\mathrm{ex}_i^{t'}$取代当前个体ex_i^t,即

$$\mathrm{ex}_i^{t'} = \begin{cases} \mathrm{ex}_i^t, \ f(\mathrm{ex}_i^t) \leqslant f(x_i^t(\lambda)); \\ x_i^t(\lambda), \ f(\mathrm{ex}_i^t) > f(x_i^t(\lambda)). \end{cases} \tag{11}$$

由式(10)可以看出,设计的收缩因子λ随着迭代次数的增加逐渐减小. 式(9)表明,λ越小,ex_i^t对候选解$x_i^t(\lambda)$的影响越小. 因此,算法随着迭代次数的增加,精英个体的上下界范围逐渐缩小至目标解附近,候选解$x_i^t(\lambda)$越侧重随机性、多样性较强的混沌个体xC_i^t,局部搜索能力越强. 精英个体的混沌变异策略所控制的混沌搜索范围随算法迭代次数的增加逐渐缩小,实验验证该策略具有较强的局部搜索能力.

2.4 算法描述

反向学习算法能增加种群的多样性,增强算法的探索能力,精英混沌搜索策略可以增强算法的开发能力,将融入精英混沌搜索策略的SCA与融入精英混沌搜索策略的OBL交替执行,可以很好地降低算法的复杂度,减少算法的运行时间. 因此,基于精英混沌搜索策略的交替正余弦算法(COSCA)步骤如下.

Step 1: 算法参数设置. 种群规模N,控制参数a的初始值a_{start}和最终值a_{end},最大化迭代次数$t_{\max}$,调节系数$\eta > 0$,精英选择比例pr.

Step 2: 初始化. 令$t=0$,随机在解搜索空中产生N个个体,组成种群G.

Step 3: 对G中的个体执行式(4)的反向策略,形成种群$\bar{G}$,计算$\bar{G} \bigcup G$中个体的函数值,按从小到大排序,选择前N个个体组成初始种群G^t,并记录最优个体x_*^t.

Step 4: 若t为奇数,则执行Step 4.1;若t为偶数,则执行Step 4.2:

Step 4.1: 对G^t中的个体执行SCA算法,其中控制参数r_1如式(3)所示,对个体按适应度值由小到大排序得到种群G_z^{t+1};

Step 4.2: 对G^t中的个体进行式(4)的OBL策略更新得到$\bar{G}^t$,对$\bar{G} \bigcup G$中的个体按函数值从小到大排序,选择前N个个体组成种群G_z^{t+1}.

Step 5: 对G_z^{t+1}中的前m个精英个体进行式(5)~(11)所示的精英混沌搜索,其余$N-m$个个体不变,形成种群G^{t+1}.

Step 6: 记录G^{t+1}中最优个体x_*^{t+1},若$f(x_*^t) < f(x_*^{t+1})$,则$x_*^{t+1} = x_*^t$.

Step 7: 判断算法是否满足结束条件,若满足,输出最优值$f(x_*^{t+1})$,否则令$t=t+1$,转至Step 4.

2.5 算法收敛性分析

文献[9]利用鞅论给出了由最优值引导的算法收敛性分析,文献[10-11]给出了混沌搜索和反向学习引导的算法的收敛性分析. 由COSCA过程可以看出,个体的演化是基于随机过程的马尔可夫链. 由文献[9-11]的收敛性分析过程可得到,COSCA随迭代次数的增加,依概率收敛到全局最优解.

3 数值实验与分析

为了验证COSCA的性能,本文选取文献[12-15]中由$F_1 \sim F_7$表示的单峰函数,$F_8 \sim F_{13}$表示的多峰函数和$F_{14} \sim F_{23}$表示的固定维函数的23个标准测试函数进行仿真实验.

3.1 与SCA, OBSCA, GWO, PSO算法的比较

为了测试COSCA的性能,将COSCA与SCA、OBSCA、PSO算法及其改进算法OBPSO[16]算法、GWO算法进行实验比较. 此外,对比实验中加入了未修改参数r_1的COSCA (简记为COrSCA)以验证非线性参数调整策略的有效性. 在对比实验中,为了公平起见,所有算法均使用相同的种群规模$N=30$,算法的最大迭代次数$t_{\max}=500$. 对于每个测试函数,各算法均独立运行20次(runs),记录其平均精度Ave,标准差STD和CPU时间,利用显著性水平$\alpha=0.05$的

Wilcoxon秩和检验[17]评价两种算法的优劣. 平均精度、标准差的计算公式为

$$\mathrm{Ave}=\frac{1}{\mathrm{runs}}\sum_{i=1}^{\mathrm{runs}}\mathrm{value}_i, \tag{12}$$

$$\mathrm{STD}=\sqrt{\frac{1}{\mathrm{runs}-1}\sum_{i=1}^{\mathrm{runs}}(\mathrm{value}_i-\mathrm{Ave})^2}, \tag{13}$$

其中value_i表示算法第i次的运行结果.

COSCA中,r_1取值见式(3),$\eta=1$,$a_{\mathrm{start}}=1$,$a_{\mathrm{end}}=0$,精英个体比例$\mathrm{pr}=0.1$,其他算法的参数设置如表1所示.

表1　算法参数设置

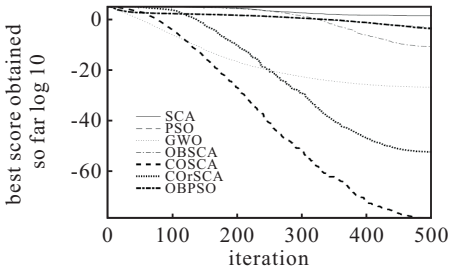
算法	参数名称	数值
SCA	a	2
OBSCA	a	2
PSO	$v_{\max},v_{\min}$	$6,-6$
	$w_{\max},w_{\min}$	$0.9,0.2$
	c_1,c_2	$1.496,1.496$
OBPSO	$v_{\max},v_{\min}$	$6,-6$
	$w_{\max},w_{\min}$	$0.9,0.2$
	c_1,c_2	$1.496,1.496$
	p_0	0.3
GWO	$a_{\max},a_{\min}$	$2,0$

表2给出了7种算法在维数$n=30$时独立运行20次的平均精度和标准差,"+/=/−"分别表示用Wilcoxon秩和检验时对比算法"优于/相当/劣于"COSCA函数的个数,加粗数据表示计算效果好的函数. 表3给出了COSCA与对比算法进行Wilcoxon秩和检测时的概率值P以及运行20次的CPU时间,加粗数据表示对比算法与COSCA算法计算结果差异较小.

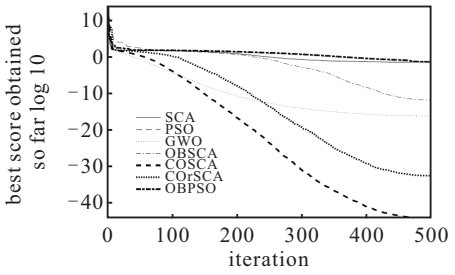
为了更直观地反映各算法的性能,图1给出了7种算法对6个测试函数的收敛曲线. F_1,F_2,F_9,F_{10}的收敛曲线以$\log_{10}(f(x))$作为纵坐标.

由表2和图1可以看出,COSCA的性能明显优于SCA与OBSCA. 通过Wilcoxon秩和检验可知,COSCA对20个函数的结果优于SCA,19个函数的结果优于OBSCA,尤其对函数F_9与F_{11},COSCA可收敛到理论最优值0,且标准差为0.

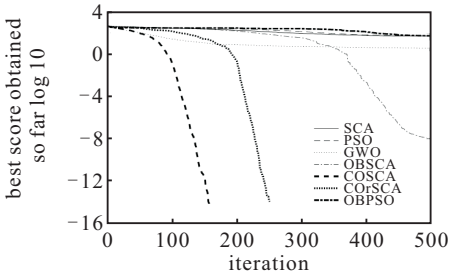
由表2可知,COSCA对14个测试函数的结果优于COrSCA,说明式(3)非线性策略对提高算法精度有显著影响. 对函数$F_1\sim F_7$和$F_{14}\sim F_{23}$,COrSCA标准差数值较大,说明COrSCA稳定性较弱,COSCA标准差较小,说明COSCA鲁棒性较强.



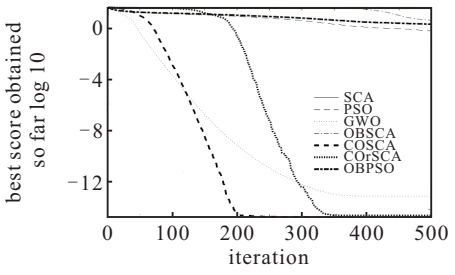
(a) F_1



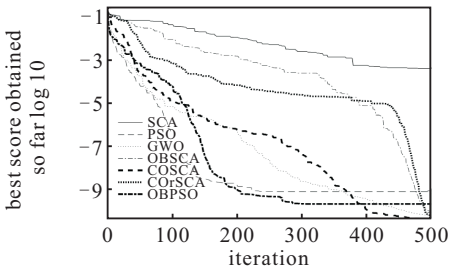
(b) F_2



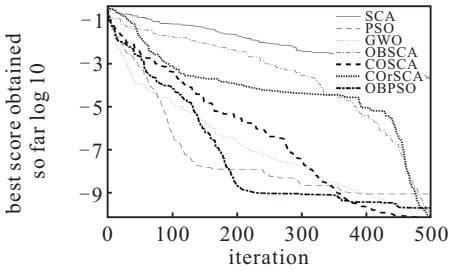
(c) F_9



(d) F_{10}



(e) F_{22}



(f) F_{23}

图1　部分函数适应度值收敛曲线

表 2 COSCA与对比算法在测试函数 $F_1 \sim F_{23}$ 上取得的结果比较($n = 30$)

测试函数		算 法						
		SCA	OBSCA	GWO	PSO	OBPSO	COrSCA	COSCA
F_1	Ave	10.836 2	1.04e-11	2.22e-27	2.75e-04	1.42e-04	7.94e-51	**2.44e-78**
	STD	12.540 9	3.48e-11	4.68e-27	1.09e-19	1.21e-04	3.24e-50	**3.21e-94**
F_2	Ave	0.032 3	1.76e-12	8.55e-17	0.044 8	0.036 6	5.76e-33	**1.52e-44**
	STD	0.044 3	6.57e-12	4.16e-17	6.20e-18	0.033 7	2.19e-32	**1.94e-60**
F_3	Ave	8.28e+03	6.666 1	8.86e-06	86.551 0	78.056 9	2.45e-14	**1.78e-15**
	STD	5.32e+03	18.851 9	1.83e-05	4.76e-15	36.650 1	4.07e-14	**1.45e-30**
F_4	Ave	35.476 4	0.149 4	8.83e-07	1.114 3	1.052 8	5.45e-16	**5.27e-35**
	STD	9.710 9	0.292 0	5.93e-07	5.23e-16	0.198 2	1.49e-15	**1.91e-50**
F_5	Ave	4.71e+04	28.563 9	**27.046 9**	117.010 1	95.361 6	28.389 9	28.373 2
	STD	7.02e+04	0.278 4	**0.886 5**	4.76e-14	23.809 0	0.361 2	7.94e-16
F_6	Ave	15.352 9	4.722 7	0.795 4	4.50e-04	**3.88e-04**	3.512 6	3.823 7
	STD	16.022 7	0.264 1	0.343 0	6.06e-20	**5.23e-04**	0.299 6	7.94e-16
F_7	Ave	0.100 1	0.003 5	0.001 9	0.178 1	0.184 6	0.001 4	**3.21e-04**
	STD	0.085 5	0.002 1	0.001 1	8.68e-17	0.078 6	8.33e-4	**6.06e-21**
F_8	Ave	−3.87e+3	−4.03e+03	−6.03e+03	−5.35e+03	**−8.14e+3**	−3.70e+03	−3.31e+03
	STD	330.602 2	328.513 3	803.094 2	5.02e-13	**958.766 8**	269.718 0	2.64e-12
F_9	Ave	51.578 0	1.24e-08	3.644 0	53.733 9	54.145 3	0	**0**
	STD	37.761 5	3.79e-08	5.189 5	3.81e-14	0.403 7	0	**0**
F_{10}	Ave	16.509 8	2.2835	1.04e-13	0.3418	1.1157	3.55e-15	**2.48e-15**
	STD	7.033 9	3.200 2	1.90e-14	0	0.787 9	1.57e-15	**7.05e-31**
F_{11}	Ave	0.930 0	0.008 3	0.004 0	0.009 8	0.007 3	0	**0**
	STD	0.332 3	0.032 5	0.007 4	3.10e-18	0.005 4	0	**0**
F_{12}	Ave	1.31e+05	0.532 7	0.046 5	**0.010 4**	0.026 0	0.313 0	0.367 9
	STD	5.62e05	0.100 6	0.025 6	**1.55e-18**	0.066 2	0.062 5	1.73e-16
F_{13}	Ave	2.85e+05	2.628 8	0.666 6	**0.006 0**	0.010 0	1.894 4	2.036 1
	STD	8.49e+05	0.164 0	0.201 0	**3.41e-18**	0.009 4	0.134 6	1.20e-15
F_{14}	Ave	**1.495 9**	1.794 4	4.721 4	3.121 0	1.839 6	1.853 6	3.558 7
	STD	**0.858 1**	0.969 7	4.027 5	2.97e-16	1.366 9	0.861 2	5.95e-16
F_{15}	Ave	0.001 0	8.59e-04	0.004 4	8.51e-04	8.58e-04	9.80e-4	**7.87e-04**
	STD	4.30e-04	2.18e-04	0.008 0	1.09e-18	1.540 3e-04	2.69e-4	**7.75e-19**
F_{16}	Ave	−1.031 6	−1.031 6	−1.031 6	−1.031 6	−1.031 6	−1.031 6	**−1.031 6**
	STD	5.27e-05	8.19e-06	2.88e-08	6.95e-16	1.90e-16	4.00e-4	**1.09e-16**
F_{17}	Ave	0.399 5	0.399 0	0.397 9	0.397 9	0.397 9	0.398 2	**0.398 0**
	STD	0.001 7	7.57e-04	8.18e-07	0	0	1.906 2e-005	**0**
F_{18}	Ave	3.000 0	3.000 0	3.000 0	3.000 0	3.000 0	3.000 0	**3.000 0**
	STD	1.08e-04	5.74e-05	6.64e-05	1.58e-15	1.63e-015	1.396 9e-005	**7.94e-16**
F_{19}	Ave	−3.855 0	−3.856 8	−3.861 2	−3.862 8	−3.862 8	−3.858 8	**−3.858 9**
	STD	0.001 7	0.003 4	0.002 4	9.53e-15	2.12e-15	0.002 3	**1.39e-15**
F_{20}	Ave	−2.884 7	−3.108 3	−3.240 5	−3.262 5	**−3.262 6**	−3.166 3	−3.156 1
	STD	0.408 3	0.032 3	0.069 7	9.93e-16	**0.061 0**	0.042 7	9.93e-16
F_{21}	Ave	−2.363 6	−8.967 3	−9.020 0	−7.140 8	−9.019 5	−8.176 0	**−9.583 4**
	STD	1.725 4	1.987 1	1.522 3	3.16e-15	2.374 7	2.421 5	**6.35e-15**
F_{22}	Ave	−3.398 5	−9.936 9	−10.137 4	−7.861 0	−9.609 8	−10.137 0	**−10.320 8**
	STD	2.119 3	0.214 5	1.149 3	8.53e-15	1.937 2	0.349 3	**4.36e-15**
F_{23}	Ave	−4.065 8	−10.208 0	−10.128 5	−9.364 1	−10.000 3	−10.367 7	**−10.482 1**
	STD	1.860 9	0.216 6	1.768 1	1.19e-14	1.650 0	0.173 1	**3.97e-15**
	$+/=/-$	1/2/20	2/2/19	6/2/15	5/2/16	6/0/17	4/5/14	

表 3　COSCA与对比算法的检验值P以及运行时间比较$(n = 30)$

Func	SCA		OBSCA		GWO		PSO		OBPSO		COrSCA	COSCA
	P	CPU/s	P	CPU/s	P	CPU/s	P	CPU/s	P	CPU/s	P	CPU/s
F_1	6.79e-8	3.6635	6.79e-8	4.4503	6.79e-8	4.4151	6.79e-8	2.1978	6.79e-8	2.8623	6.79e-8	3.4796
F_2	6.79e-8	4.1529	6.79e-8	5.2638	6.79e-8	4.8171	6.79e-8	2.6544	6.79e-8	3.2259	6.79e-8	3.9632
F_3	6.79e-8	16.9286	6.79e-8	30.6415	6.79e-8	17.2462	6.79e-8	15.1055	6.79e-8	18.2259	6.61e-5	18.374
F_4	6.79e-8	4.6639	6.79e-8	6.0241	6.79e-8	5.1830	6.79e-8	2.8893	6.79e-8	4.1774	6.79e-8	4.9021
F_5	6.79e-8	5.4872	6.79e-8	7.8935	1.80e-5	6.4877	0.0012	3.9658	1.20e-6	4.7198	**0.3942**	5.6239
F_6	0.0040	3.7184	0.0040	4.5219	6.79e-8	4.1882	6.79e-8	2.2307	6.79e-8	3.2437	0.0010	3.4614
F_7	6.79e-8	5.9879	6.79e-8	9.1176	5.22e-7	6.3740	6.79e-8	4.4161	6.79e-8	4.9008	3.01e-6	6.3252
F_8	**0.2531**	4.0731	**0.1362**	5.1991	7.89e-8	4.4542	2.56e-7	2.5522	6.79e-8	3.3187	1.25e-5	3.9893
F_9	8.00e-9	5.2916	8.00e-9	7.1646	7.90e-9	6.2719	8.00e-9	3.7008	7.89e-8	4.5804	NaN	4.7737
F_{10}	4.14e-8	5.9749	4.14e-8	8.8148	4.03e-8	6.2597	4.14e-8	4.1045	2.86e-8	5.5779	**0.0579**	5.6480
F_{11}	8.00e-9	5.0327	8.00e-9	7.0543	0.0198	5.2623	8.00e-9	3.3910	8.00e-9	4.4193	NaN	5.2364
F_{12}	7.89e-8	10.7797	7.89e-8	18.4836	6.79e-8	10.9661	6.79e-8	8.9245	6.79e-8	11.2894	**0.1264**	10.606
F_{13}	6.79e-8	10.5417	6.79e-8	18.0083	6.79e-8	10.7932	6.79e-8	8.7523	6.79e-8	10.9795	**0.0810**	10.818
F_{14}	0.0056	20.1181	0.0056	38.0248	**0.7150**	19.8476	**0.4084**	18.4022	0.0054	22.3455	0.0021	20.831
F_{15}	**0.0764**	3.4870	**0.0764**	5.4046	0.0071	3.4935	**0.0639**	2.1396	3.74e-4	2.6817	0.0144	3.5136
F_{16}	5.22e-7	2.2384	5.22e-7	3.2040	1.43e-7	2.2292	3.25e-8	1.1462	3.29e-8	2.0479	7.57e-4	2.2698
F_{17}	5.87e-6	1.8421	5.87e-6	2.4388	1.91e-7	1.8942	8.00e-9	0.7406	8.00e-9	1.5542	1.80e-5	1.9703
F_{18}	3.04e-4	1.9158	3.04e-4	2.7977	**0.1264**	1.9097	5.66e-8	0.9363	6.25e-8	1.2140	**0.9031**	0.2085
F_{19}	1.25e-5	3.870	2 1.25e-5	6.0436	3.04e-4	3.8046	2.40e-8	2.4930	3.66e-8	3.7492	0.0167	4.2596
F_{20}	1.57e-6	3.9909	1.57e-6	6.2521	3.74e-4	3.9776	1.17e-5	2.6834	8.57e-7	4.9611	**0.1136**	4.5213
F_{21}	1.06e-7	5.8417	1.06e-7	9.5960	7.57e-6	5.6821	6.79e-8	4.1915	3.34e-4	6.4776	**0.1333**	6.1842
F_{22}	6.79e-8	6.2997	6.79e-8	10.6934	1.20e-6	6.3290	9.5e-04	4.9444	1.05e-4	6.8727	0.0031	7.4280
F_{23}	6.79e-8	7.5541	6.79e-8	13.2510	1.57e-6	7.5459	1.43e-4	6.2286	1.21e-5	8.2253	0.0133	9.0600

由表2可看出: 与GWO、PSO、OBPSO算法相比, COSCA均获得了较好的收敛结果, 函数$F_1 \sim F_4, F_7$中COSCA的精度远高于GWO、PSO、OBPSO算法, 同时COSCA标准差相对较小; COSCA算法对F_9和F_{10}结果较优, 对$F_{11} \sim F_{13}$结果较差, 说明COSCA在求解多峰优化问题时性能上仍需加强. 此外, COSCA对函数$F_{15} \sim F_{23}$的结果较优, 在$F_{21} \sim F_{23}$上获得的结果接近理论最优解. 统计表2中7种算法表现结果, COSCA在16个函数中结果最优, 在18个函数中标准差最小. 在$\alpha = 0.05$的显著性水平下, 由Wilcoxon秩和检验结果可知, GWO、PSO、OBPSO对应结果分别为6/2/15、5/2/16、6/0/17. 由表3可知, COSCA的CPU时间优于OBSCA、GWO、OBPSO算法, 与SCA算法相当, 不及PSO算法.

综上所述, COSCA性能优于6种对比算法, 且COSCA具有收敛速度快、精度高、计算量小、鲁棒性较强等特点, 同时有效地协调了算法探索与开发能力, 能够很好地处理大部分问题.

3.2　精英个体比例分析

精英个体比例pr的选取是精英混沌搜索策略的关键所在. 由第2.3节可知, 精英混沌搜索策略可以增强算法的局部开发能力, 较大的pr会使算法产生早熟, 而pr取值过小则对算法的影响甚微, 因此本节通过仿真实验来检测pr取值对算法性能的影响. 单峰函数可以很好地检验算法的局部搜索能力, 而多峰函数因其拥有大量局部极值被认为是最具挑战性的问题, 故选取函数$F_1 \sim F_{12}$进行实验, pr在$[0, 0.4]$区间内以0.1的间隔取5个值, r_1策略见式(3), 其他参数不变, 用Friedman检验[18]实验结果的差异性. 表4给出了测试函数为30维时的平均精度、标准差、排序(Rank)以及检验的概率值P. 由表4可见, 测试函数对应的Friedman检测值P均小于0.05, 表明5种pr取值对应的算法结果之间存在显著差异. 当参数pr取不同值时, COSCA在测试函数中结果相差较大, 因此精英混沌搜索策略对pr的取值较为敏感. 在本文中, 当$\mathrm{pr} = 0.1$时, COSCA性能最优.

表 4 控制参数pr在测试函数 $F_1 \sim F_{12}$ 上取得的结果比较(均值,标准差,Friedman-P,rank,$n=30$)

pr	$F_1(P=1.227\,7\text{e-}16)$			$F_2(P=2.298\,4\text{e-}16)$			$F_3(P=9.493\,8\text{e-}15)$			$F_4(P=8.146\,5\text{e-}17)$		
	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank
0	1.82e-05	7.57e-22	5	2.52e-07	3.55e-23	5	241.208 0	6.35e-15	5	1.222 0	7.19e-16	5
0.1	7.63e-62	9.12e-62	2	4.87e-36	8.18e-36	2	1.09e-15	1.43e-15	1	1.40e-23	2.45e-23	2
0.2	6.82e-67	9.00e-67	1	6.44e-38	9.26e-38	1	3.80e-13	2.13e-13	2	1.58e-25	2.64e-25	1
0.3	7.91e-31	1.67e-30	3	3.24e-19	6.38e-19	3	9.99e-08	1.78e-07	3	6.77e-11	5.45e-11	4
0.4	5.17e-29	6.11e-29	4	4.11e-14	4.86e-14	4	0.044 3	0.033 8	4	5.11e-11	2.24e-11	3

pr	$F_5(P=4.885\,3\text{e-}11)$			$F_6(P=9.851\,9\text{e-}14)$			$F_7(P=3.403\,5\text{e-}16)$			$F_8(P=1.163\,5\text{e-}09)$		
	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank
0	30.451 9	3.17e-15	5	5.754 7	9.93e-16	5	0.387 3	4.96e-17	5	−2.40e+03	3.559e-13	5
0.1	28.000 8	0.319 9	1	3.352 6	0.183 5	4	8.93e-04	3.43e-04	1	−3.67e+03	254.748 8	4
0.2	28.045 2	0.205 8	3	2.719 4	0.227 0	3	0.001 4	4.80e-04	2	−3.72e+03	278.532 9	3
0.3	28.163 0	0.192 5	4	2.069 5	0.246 2	2	0.003 7	8.76e-04	3	−3.92e+03	261.770 4	1
0.4	28.037 0	0.069 0	2	2.039 7	0.234 9	1	0.007 4	0.001 6	4	−3.85e+03	263.241 6	2

pr	$F_9(P=7.594\,0\text{e-}16)$			$F_{10}(P=1.054\,8\text{e-}16)$			$F_{11}(P=1.552\,6\text{e-}16)$			$F_{12}(P=1.560\,8\text{e-}14)$		
	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank
0	7.53e-05	1.51e-20	5	0.005 2	1.93e-19	5	4.51e-04	2.38e-04	5	0.950 0	7.44e-17	5
0.1	0	0	2	1.95e-15	1.49e-15	1	0	0	3	0.294 7	0.036 7	4
0.2	0	0	2	3.73e-15	1.00e-15	2	0	0	3	0.199 6	0.017 6	3
0.3	0	0	2	4.44e-15	0	3	0	0	3	0.164 6	0.029 7	2
0.4	4.62e-34	2.31e-34	4	1.04e-14	7.94e-15	4	0	0	3	0.152 7	0.047 2	1

3.3 控制参数r_1分析

SCA中,控制参数r_1起着平衡算法开发与探索能力的作用,较大的参数r_1利于探索,较小的参数r_1利于开发. 式(3)中,$r_1 \in [a_{\text{end}}, a_{\text{start}}]$非线性递减,非线性调节系数$\eta$和$a_{\text{start}}$、$a_{\text{end}}$起着重要的作用,本小节通过仿真实验对$\eta$和$a_{\text{start}}$、$a_{\text{end}}$取不同值时的算法性能进行分析. 测试函数与3.2节相同,精英个体比例pr = 0.1,其他参数不变,实验结果通过Friedman进行检验. 不同的η值对COSCA性能的影响结果比较如表5所示,4组不同a_{start}、a_{end}仿真结果如表6所示. 由表5可知,$\eta=1$时,效果最好. 由表6可知,$a_{\text{start}}=1, a_{\text{end}}=0$对应算法性能最优.

表 5 不同的η值对COSCA性能的影响结果比较($n=30, a_{\text{start}}=1, a_{\text{end}}=0$)

η	$F_1(P=2.269\,8\text{e-}14)$			$F_2(P=5.172\,9\text{e-}13)$			$F_3(P=1.431\,9\text{e-}08)$			$F_4(P=5.294\,9\text{e-}16)$		
	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank
1	1.79e-78	5.35e-95	1	8.66e-44	1.31e-58	1	4.55e-16	0	1	5.53e-19	3.44e-34	1
2	2.54e-74	9.30e-74	2	1.14e-45	3.06e-45	2	2.32e-16	3.83e-16	2	9.21e-10	2.38e-09	2
3	4.11e-68	1.27e-67	3	7.26e-43	2.25e-42	3	3.17e-14	5.06e-14	3	1.87e-07	1.90e-07	3
5	5.34e-63	9.65e-79	4	1.57e-40	4.55e-57	4	1.53e-13	1.01e-28	4	9.1511	6.75e-15	4

η	$F_5(P=2.536\,8\text{e-}06)$			$F_6(P=2.381\,0\text{e-}13)$			$F_7(P=8.086\,7\text{e-}12)$			$F_8(P=0.014\,6)$		
	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank
1	28.016 8	7.912e-15	1	3.516 2	2.48e-15	2	8.31e-04	6.54e-19	1	−3.66e+03	2.84e-12	4
2	28.374 5	0.291 9	3	3.423 1	0.317 7	1	0.006 4	0.003 2	2	−3.85e+03	322.249 3	2
3	28.321 8	0.202 5	2	4.209 6	0.512 4	3	0.009 2	0.004 8	3	−3.84e+03	293.672 9	3
5	28.548 4	7.14e-15	4	5.046 4	3.77e-15	4	0.017 6	2.01e-17	4	−3.90e+03	3.35e-12	1

η	$F_9(P=1.131\,9\text{e-}11)$			$F_{10}(P=7.146\,0\text{e-}16)$			$F_{11}(P=6.773\,1\text{e-}14)$			$F_{12}(P=2.038\,7\text{e-}13)$		
	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank
1	0	0	2	2.48e-15	8.81e-031	1	0	0	2	0.307 9	3.72e-17	1
2	0	0	2	4.26e-15	1.628e-15	2	0	0	2	0.374 2	0.116 6	2
3	0	0	2	1.77e-10	7.61e-10	3	0	0	2	0.530 6	0.109 7	3
5	1.25e-13	4.51e-29	4	8.55e-09	0	4	1.15e-14	1.69e-29	4	0.665 4	2.48e-16	4

表 6　不同的$a_{\rm start}, a_{\rm end}$值对COSCA性能的影响结果比较$(n=30)$

$a_{\rm start}$、$a_{\rm end}$	$F_1(P=1.741\,8\text{e-}16)$			$F_2(P=1.741\,8\text{e-}16)$			$F_3(P=3.437\,6\text{e-}15)$			$F_4(P=3.515\,6\text{e-}16)$		
	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank
1, 0	2.27e-79	1.34e-94	1	3.63e-45	5.98e-60	1	6.11e-16	2.20e-31	1	5.60e-35	6.21e-50	1
2, 0	5.66e-60	2.32e-59	2	3.10e-37	8.11e-37	2	1.05e-15	1.60e-15	2	4.78e-21	1.79e-20	2
2, 1	1.47e-29	3.38e-29	3	3.26e-20	7.08e-20	4	4.01e-11	5.72e-11	4	1.31e-07	2.36e-07	3
3, 0	1.17e-43	7.12e-59	4	7.67e-28	5.21e-43	3	6.85e-14	1.12e-29	3	5.03e-11	3.66e-26	4
$a_{\rm start}$、$a_{\rm end}$	$F_5(P=0.308\,4)$			$F_6(P=5.580\,7\text{e-}06)$			$F_7(P=5.883\,3\text{e-}14)$			$F_8(P=8.111\,4\text{e-}8)$		
	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank
1, 0	28.422 9	1.11e-14	3	3.844 7	2.03e-15	4	2.17e-04	2.48e-19	1	−3.30e+03	1.22e-12	4
2, 0	28.241 0	0.425 2	3	3.423 1	0.2621	1	6.11e-04	3.57e-04	2	−3.58e+03	279.969 3	3
2, 1	28.136 4	0.289 9	3	3.467 8	0.337 0	2	0.010 6	0.004 3	4	−3.84e+03	282.626 3	2
3, 0	28.142 5	1.90e-14	3	3.815 3	2.08e-15	3	0.001 3	5.09e-19	3	−3.89e+03	2.54e-12	1
$a_{\rm start}$、$a_{\rm end}$	$F_9(P=4.994\,0\text{e-}04)$			$F_{10}(P=7.146\,0\text{e-}16)$			$F_{11}(P=9.437\,8\text{e-}10)$			$F_{12}(P=1.344\,0\text{e-}05)$		
	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank	Ave	Std	Rank
1, 0	0	0	3	2.48e-15	7.05e-31	1	0	0	3	0.371 8	1.86e-16	3
2, 0	0	0	3	3.19e-15	1.69e-15	2	0	0	3	0.317 1	0.123 5	2
2, 1	0	0	3	1.24e-12	5.05e-12	4	0	0	3	0.445 5	0.130 6	4
3, 0	0	0	3	4.44e-15	0	3	0	0	3	0.306 7	6.82e-17	1

3.4　时间复杂度分析

本节给出COSCA的时间复杂度. 为了简化表达式,在计算时间复杂度时,将精英混沌策略中的混沌迭代部分记为$C^{\rm iter}$,种群规模为N. COSCA的实验阶段使用了快排算法Qs,最优条件下快排N个个体算法的时间复杂度为$O(N\log N)$,最差条件下其时间复杂度为$O(N^2)$.

SCA时间复杂度为

$$O(\text{SCA})=O(N\cdot n\cdot t_{\max}). \tag{14}$$

OBSCA时间复杂度为

$$O(\text{OBSCA})=O((2N\cdot n+O_{QS})\cdot t_{\max}). \tag{15}$$

COSCA时间复杂度为

$$O(\text{COSCA})=O((1.1N\cdot n+O_{QS})\cdot t_{\max}+C^{\rm iter}), \tag{16}$$

$$O(C^{\rm iter})=O\left(\left(0.5+\frac{t_{\max}}{20}\right)\cdot t_{\max}\right). \tag{17}$$

其中:n表示搜索空间维数,$t_{\max}$表示最大迭代次数. COSCA与SCA时间复杂度差值为$O((0.1N\cdot n+O_{QS})\cdot t_{\max}+C^{\rm iter})$,OBSCA与SCA时间复杂度差值为$O((N\cdot n+O_{QS})\cdot t_{\max})$,$C^{\rm iter}$时间复杂度较低,因此COSCA复杂度低于OBSCA.

仿真实验计算机配置: Intel(R) Core(TM) i5-6 200U CPU @2.30 GHz 2.40 GHz, 8 GB 内存. 算法CPU时间可从侧面反映该算法的时间复杂度,将表3中各算法的CPU时间累积,得到PSO算法CPU时间最短(108.79 s), SCA的CPU时间次之(141.23 s), OBPSO算法与COSCA的CPU时间相仿,分别为141.65 s和147.44 s, GWO算法CPU时间为149.43 s, OBSCA的CPU时间最长,为223.71 s.

COSCA时间复杂度低于OBSCA、GWO算法,与SCA、OBPSO算法时间复杂度相仿,复杂度高于PSO算法. OBPSO作为PSO的改进算法,复杂度与COSCA相仿,而COSCA计算精度与稳定性均优于OBPSO,因此COSCA优于PSO算法.

4　结　论

本文提出了一种基于对数非线性参数调整和精英混沌搜索策略的交替正余弦算法,既拓展了算法的全局搜索能力,又增强了算法的局部开发能力. 23个基准测试函数的实验结果表明: COSCA收敛性和稳定性方面优于SCA、OBSCA、GWO以及PSO算法; COSCA时间复杂度优于OBSCA,与SCA、GWO

相仿;PSO算法在时间复杂度上优于COSCA,但收敛精度与稳定性方面劣于COSCA,综合实验结果表明COSCA优于PSO算法.

参考文献(References)

[1] Kenndy J, Eberhart R C. Particle swarm optimization[C]. Proc of IEEE Int Conf on Neural Networks. Perth: IEEE, 1995: 1942−1948.

[2] Mirjalili S, Mirjalili S M, Lewis A. Grey wolf optimizer[J]. Advances in Engineering Software, 2014, 69(7): 46-61.

[3] Mirjalili S. SCA: A sine cosine algorithm for solving optimization problems[J]. Knowledge-Based Systems, 2016, 96: 120-133.

[4] Elaziz M A, Oliva D, Xiong S. An improved opposition-based sine cosine algorithm for global optimization[J]. Expert Systems with Applications, 2017, 90: 484-500.

[5] 龙文, 伍铁斌. 协调探索和开发能力的改进灰狼优化算法[J]. 控制与决策, 2017, 32(10):1749-1758.
(Long W，Wu T B. Improved grey wolf optimization algorithm coordinating the ability of exploration and exploitation[J]. Control and Decision, 2017, 32(10): 1749-1758.)

[6] Tizhoosh H R. Opposition-based learning: A new scheme for machine intelligence[C]. Proc of Int Conf on Computational Intelligence for Modeling Control and Automation. Vienna: IEEE, 2005: 695-701.

[7] 龙文, 蔡绍洪, 焦建军, 等. 求解高维优化问题的混合灰狼优化算法[J]. 控制与决策, 2016, 31(11): 1991-1997.
(Long W, Cai S H, Jiao J J, et al. Hybrid grey wolf optimization algorithm for high-dimensional optimization[J]. Control and Decision, 2016, 31(11): 1991-1997.)

[8] Wang H, Wu Z, Rahnamayan S, et al. Enhancing particle swarm optimization using generalized opposition-based learning[J]. Information Sciences, 2011, 181(20): 4699-4714.

[9] Xu G, Yu G S. On convergence analysis of particle swarm optimization algorithm[J]. J of Computational and Applied Mathematics, 2018, 333: 65-73.

[10] Seif Z, Ahmadi M B. An opposition-based algorithm for function optimization[J]. Engineering Applications of Artificial Intelligence, 2015, 37: 293-306.

[11] Gandomi A H, Yun G J, Yang X S, et al. Chaos-enhanced accelerated particle swarm optimization[J]. Commun Nonlinear Sci Numer Simulat, 2013, 18(2): 327-340.

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

[13] Digalakis J, Margaritis K. On benchmarking functions for genetic algorithms[J]. Int J of Computational Mathematics, 2001, 77(4): 481-506.

[14] Mirjalili S, Lewis A. S-shaped versus V-shaped transfer functions for binary particle swarm optimization[J]. Swarm & Evolutionary Computation, 2013, 9: 1-14.

[15] Mirjalili S, Mirjalili S M, Yang X S. Binary bat algorithm[J]. Neural Computing & Applications, 2014, 25(3/4): 663-681.

[16] Wang H, Liu H, Zeng S Y. Opposition-based particle swarm algorithm with Cauchy mutation[C]. IEEE Congress on Evolutionary Computation. Singapore: IEEE, 2007: 4750-4756.

[17] Zhou J, Yao X. Multi-population parallel self-adaptive differential artificial bee colony algorithm with application in large-scale service composition for cloud manufacturing[J]. Applied Soft Computing, 2017, 56: 179-397.

[18] Pahnehkolaei S M A, Alfi A, Sadollah A, et al. Gradient-based water cycle algorithm with evaporation rate applied to chaos suppression[J]. Applied Soft Computing, 2016, 53: 420-440.

作者简介

郭文艳(1973−), 女, 副教授, 博士, 从事智能算法、图形图像处理等研究, E-mail: wyguo@xaut.edu.cn;

王远(1994−), 男, 硕士生, 从事智能算法的研究, E-mail: 815686940@qq.com;

戴芳(1966−), 女, 教授, 博士, 从事图形图像处理等研究, E-mail: daifang@xaut.edu.cn;

刘婷(1994−), 女, 硕士生, 从事智能算法的研究, E-mail: 983944810@qq.com.

(责任编辑：郑晓蕾)