软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2013,24(11):2687−2698 [doi: 10.3724/SP.J.1001.2013.04476]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

逐维改进的布谷鸟搜索算法*

王李进[1,2], 尹义龙[2], 钟一文[1]

[1](福建农林大学 计算机与信息学院,福建 福州 350002)

[2](山东大学 计算机科学与技术学院,山东 济南 250101)

通讯作者: 尹义龙, E-mail: ylyin@sdu.edu.cn, http://mla.sdu.edu.cn/ylyin.html

摘 要: 布谷鸟搜索(cuckoo search,简称 CS)算法是一种新兴的仿生智能算法,对解采用整体更新评价策略.在求解多维函数优化问题时,由于各维之间相互干扰,采用整体更新评价策略将恶化算法的收敛速度和解的质量.为了弥补此缺陷,提出了基于逐维改进的布谷鸟搜索算法.在改进算法的迭代过程中,针对解采用逐维更新评价策略.该策略将各维的更新值与其他维的值组合成新的解,并采用贪婪方式接受能够改善解质量的更新值.实验结果说明,改进策略能够有效地提高CS算法的收敛速度并改善解的质量.与相关的改进布谷鸟搜索算法以及其他演化算法的比较结果表明,改进算法在求解连续函数优化问题上是具有竞争力的.

关键词: 布谷鸟搜索算法;逐维改进;函数优化;多维函数;干扰现象

中图法分类号: TP183 **文献标识码**: A

中文引用格式: 王李进,尹义龙,钟一文.逐维改进的布谷鸟搜索算法.软件学报,2013,24(11):2687−2698. http://www.jos.org.cn/1000-9825/4476.htm

英文引用格式: Wang LJ, Yin YL, Zhong YW. Cuckoo search algorithm with dimension by dimension improvement. Ruan Jian Xue Bao/Journal of Software, 2013,24(11):2687−2698 (in Chinese). http://www.jos.org.cn/1000-9825/4476.htm

Cuckoo Search Algorithm with Dimension by Dimension Improvement

WANG Li-Jin[1,2], YIN Yi-Long[2], ZHONG Yi-Wen[1]

[1](School of Computer and Information Science, Fujian Agriculture and Forestry University, Fuzhou 350002, China)

[2](School of Computer Science and Technology, Shandong University, Ji'nan 250101, China)

Corresponding author: YIN Yi-Long, E-mail: ylyin@sdu.edu.cn, http://mla.sdu.edu.cn/ylyin.html

Abstract: Cuckoo search (CS) is a new nature-inspired intelligent algorithm which uses the whole update and evaluation strategy on solutions. For solving multi-dimension function optimization problems, this strategy may deteriorate the convergence speed and the quality of solution of algorithm due to interference phenomena among dimensions. To overcome this shortage, a dimension by dimension improvement based cuckoo search algorithm is proposed. In the progress of iteration of improved algorithm, a dimension by dimension based update and evaluation strategy on solutions is used. The proposed strategy combines an updated value of one dimension with values of other dimensions into a new solution, and greedily accepts any updated values that can improve the solution. The simulation experiments show that the proposed strategy can improve the convergence speed and the quality of the solutions effectively. Meanwhile, the results also reveal the proposed algorithm is competitive for continuous function optimization problems compared with other improved cuckoo search algorithms and other evolution algorithms.

Key words: cuckoo search algorithm; dimension by dimension improvement; function optimization; multi-dimension function; interference phenomena

自 20 世纪以来,人们利用蚂蚁、鸟类等群居生物的自组织行为,提出了粒子群算法(particle swarm

* 基金项目: 新世纪优秀人才支持计划(NCET-11-0315); NSFC-广东联合基金重点支持项目(U1201258); 福建省自然科学基金(2011J05044, 2013J01216); 山东省自然科学杰出青年基金(2013JQE27038)

收稿时间: 2013-04-27; 修改时间: 2013-07-17; 定稿时间: 2013-08-27

optimization,简称 PSO)[1,2]和蚁群算法(ant colony optimization,简称 ACO)[3]等群智能算法,并成功用于求解大量的实际问题[4].近几年来,源于生物系统的灵感,涌现了诸多新颖的仿生优化算法,如蜂群优化算法[5]、族群优化算法[6]、布谷鸟搜索(cuckoo search,简称 CS)算法[7,8]等.

CS 算法是一种新颖的群智能算法,其思想源于对布谷鸟寄生育雏行为以及鸟类或果蝇 Lévy flights 行为的模拟.CS 算法不仅具有简单、参数少、易于实现等优点,而且其性能接近于标准的粒子群优化算法和差分演化算法(differential evolution,简称 DE)[9,10].

CS 算法具有显著的高效性,主要源于两个非常关键的组件:Lévy flights 随机游动和偏好随机游动,二者平衡算法的局部搜索和全局搜索.鉴于简单和高效,众多学者对 CS 算法进行研究,并提出相关改进版本,主要涉及步长改进、自适应以及与其他算法混合等方面.Walton 等人[11]针对 Lévy flights 随机游动中的 Lévy 随机步长大小提出一种改进版本以加强局部搜索,实验结果说明,对大多数测试函数而言,改进算法具有较好的性能,特别是在高维上能够较快地收敛于全局最优.Tuba 等人[12]针对偏好随机游动中的步长提出一种基于种群排序的改进版本,实验结果显示,改进算法的性能略优于标准的 CS 算法.Valian 等人[13]引入自适应机制,并提出了一种自适应步长和自适应发现概率的 CS 算法,实验结果说明,在大多数函数上解的质量优于标准 CS 算法的解.Layeb 等人[14]引入量子比特、量子纠缠以及量子变异等量子计算概念,以提高 CS 算法种群的多样性,并成功地应用于求解装箱问题.Ghodrati 等人[15]借鉴 PSO 算法中全局最优和个体最优的概念,在 CS 算法 Lévy flights 随机游动和偏好随机游动之间引入 PSO 组件,其性能与相关算法相比具有一定的竞争力.Wang 等人[16]将 PSO 与 CS 串行,在每次迭代过程中首先用 PSO 算法优化种群,并记录全局最优和个体最优,其次采用 CS 算法对种群个体最优继续寻优,仿真实验验证,改进算法比 CS 算法有更好的寻优能力.Srivastava 等人[17]在 Lévy flights 进行局部搜索时引入禁忌搜索思想,将当前最优解存入禁忌搜索队列中,以避免陷入局部最优,并成功地应用于求解软件测试数据自动生成问题.Li 等人[18,19]在偏好随机游动之后引入正交学习机制,以提高算法的搜索能力,在求解混沌系统的参数估算问题和连续函数优化问题上都获得了较好的性能.

上述改进策略都较好地改善了 CS 算法的收敛速度或解的质量,然而无论算法采用 Lévy flights 随机游动还是偏好随机游动生成新的解,都是在整体更新后才对这些解进行评价,即对每个新解修改其所有维的值,再对其评价.对于多维的目标问题而言,由于存在着维间互扰现象,整体更新评价策略将影响算法的收敛速度和解的质量[20].另一方面,前期的研究结果[20]表明,迭代改进策略能够较好地避免维间互扰.迭代改进策略为了降低算法对目标函数评价的依赖,重新构建新的评价算子对解的更新值进行评价.然而,对于非线性目标函数或者更复杂的优化问题,重新构建新的评价算子是困难的,甚至是不可行性的,这将降低算法的实用性.本文借鉴迭代改进策略的思想,提出逐维改进的布谷鸟搜索算法(cuckoo search algorithm with dimension by dimension improvement,简称 DDICS).DDICS 算法通过在偏好随机游动中引入逐维改进策略来避免维间互扰,同时改写偏好随机游动的步长更新方式,增加搜索方向,以便有效利用单维的信息加强算法的局部搜索能力,从而提高解的质量.需要指出的是,DDICS 算法在逐维改进过程中直接采用目标函数,而不是重新构建新的评价算子对解的更新值进行评价,这有利于算法的实际应用和推广.实验结果说明,改进策略能够有效地提高 CS 算法的收敛速度并改善解的质量,而且在求解连续函数优化问题上是具有竞争力的.

1 布谷鸟搜索算法

布谷鸟采用寄生育雏的特殊繁殖后代策略,总是将自己的蛋存放到其他鸟类的巢里,让其他鸟类为其孵化下一代.为了降低被发现的概率,一些布谷鸟将自己的蛋模仿成所选鸟类的蛋.当其他鸟类发现其巢里有外来的蛋,则会将外来的蛋丢弃或者放弃自己的巢,在其他地方构建新的巢.Yang 和 Deb 基于上述的布谷鸟繁殖策略,抽象出布谷鸟搜索算法,且该算法基于 3 条理想规则[7,8]:

规则 1. 每只布谷鸟 1 次只产 1 颗蛋,并随机选择 1 个鸟巢存放.

规则 2. 蛋最好的鸟巢将会被保留至下一代.

规则 3. 可用鸟巢的数量是固定的,而且鸟巢中外来蛋被发现的概率是 $p_a \in [0,1]$.

基于上述的 3 条规则,CS 算法的基本流程如算法 1 所示.

算法 1. CuckooSearch.

Begin

初始化种群:n host nests X_i(i=1,2,…,n);

计算适应值:F_i(i=1,2,…,n).

While (不满足停止条件)

采用 Lévy flight 生成新的解 X_i;

计算新解 X_i 的适应值 F_i;

选择候选解 X_j;

If (F_i>F_j)

用新的解替代候选解;

End

按发现概率 p_a 丢弃差的解;

用偏好随机游动产生新的解替代丢弃的解;

保留最好的解.

End

End

在 CS 算法中,1 个鸟巢表示 1 个候选解.CS 算法首先在当前解的基础上以 Lévy flights 随机游动方式生成新的解,评价并保留较好的解;其次,按发现概率 p_a 丢弃部分解;最后,按偏好随机游动方式重新生成与丢弃解相同数量的新解,在评价并保留较好的解之后,完成一次迭代.

当采用 Lévy flights 随机游动生成新的解 X_i 以后,执行公式(1)的操作:

$$X_{g+1,i}=X_{g,i}+\alpha\oplus Lévy(\beta) \tag{1}$$

其中,$X_{g,i}$ 表示第 g 代第 i 个解;α 是步长信息,用于控制随机搜索的范围.为了能够从当前最优解中获得更多有用的步长信息,文献[8]采用公式(2)计算步长信息:

$$\alpha=\alpha_0(X_{g,i}-X_{best}) \tag{2}$$

其中,α_0 是常数(α_0=0.01),X_{best} 表示当前最优解.

公式(1)中,⊕是点乘积(entry-wise multiplications),$Lévy(\lambda)$服从 Lévy 概率分布:

$$Lévy(\beta)\sim u=t^{-1-\beta},\ 0<\beta\leqslant 2 \tag{3}$$

为了便于计算,文献[8]采用公式(4)计算 Lévy 随机数:

$$Lévy(\beta)\sim\frac{\phi\times u}{|v|^{1/\beta}} \tag{4}$$

其中,u,v 服从标准正态分布,β=1.5.

$$\phi=\left(\frac{\Gamma(1+\beta)\times\sin(\pi\times\beta/2)}{\Gamma\left(\frac{1+\beta}{2}\right)\times\beta\times 2^{(\beta-1)/2}}\right)^{1/\beta} \tag{5}$$

显然,综合公式(1)~公式(5),在 Lévy flights 随机游动组件中,CS 算法采用公式(6)生成新的解 X_i:

$$X_{g+1,i}=X_{g,i}+\alpha_0\frac{\phi\times u}{|v|^{1/\beta}}(X_{g,i}-X_{g,best}) \tag{6}$$

在按一定的概率(发现概率)丢弃部分解之后,算法采用偏好随机游动重新生成相同数量的新解,见公式(7):

$$X_{g+1,i}=X_{g,i}+r(X_{g,j}-X_{g,k}) \tag{7}$$

其中,r 是缩放因子,是(0,1)区间的均匀分布随机数;$X_{g,j}$ 和 $X_{g,k}$ 是表示第 g 代的两个随机解.

2 逐维改进的 CS 算法

2.1 基于贪婪的逐维更新评价策略

在CS算法中,虽然偏好随机游动组件有利于提高种群的多样性以及加强算法的全局搜索能力,但对于多维目标函数而言,对解采用整体更新评价策略将影响算法收敛速度和解的质量.假设目标函数 $f(X)=x_1^2+x_2^2+x_3^2$,以及在第 g 代的第 i 个解 $X_{g,i}$=(0.5,0.5,0.5),此时,目标函数值 $f(X_{g,i})$=0.75.在算法迭代过程中,假设首先采用公式(7)将 $X_{g,i}$ 的整体更新为 $X_{g+1,i}$=(0,1,−1),其次根据目标函数评价该解,则目标函数值 $f(X_{g+1,i})$=2.由于 $f(X_{g+1,i})$=2>$f(X_{g,i})$未能改进当前代的解,算法将放弃新的解,保留当前代的解.然而,对比更新前后解的各维的值以及目标函数的全局最优解 X_{opt}=(0,0,0)可知,第 1 维的值由 0.5 进化为 0,第 2 维和第 3 维的值却分别退化为 1 和−1.正因为第 2 维和第 3 维的退化,使整体的函数值变差,导致此解的质量不高而被抛弃,最终忽略第 1 维的进化,浪费函数评价次数,恶化收敛速度.

逐维更新评价策略能够较好地避免上述问题.该策略分别考虑各维值更新,当某一维的值通过公式更新后,将与其他维的值组合成一个新的解,继而评价该新解.而基于贪婪的逐维更新评价策略,只有能够改善当前解的更新值才被接受.例如,当 $X_{g,i}$ 第 1 维的值由 0.5 更新为 0 时,便与其他维的值组合成一个新解 $X_{g+1,i}$=(0,0.5,0.5),因为函数值 $f(X_{g+1,i})$=0.5<$f(X_{g,i})$,所以基于贪婪的策略将接受该值的更新,并进入下一维的更新操作.若 $X_{g,i}$ 第 1 维值由 0.5 更新为 1 时,便与其他维的值组合成一个新解 $X_{g+1,i}$=(1,0.5,0.5),由于函数值 $f(X_{g+1,i})$=1.5>$f(X_{g,i})$,更新的值未能改善当前的解,将放弃当前维的更新值,并进入下一维的更新操作.基于贪婪的逐维更新评价策略使得解的进化维不会因其他维的退化而被忽略,保障算法能够利用进化的单维信息引导当前解进行局部搜索,从而获得更高质量的解并提高算法的收敛速度.

2.2 改进算法的流程

DDICS 算法通过在偏好随机游动中引入逐维更新评价策略,以避免维间互扰.然而,偏好随机游动中的步长更新方式是基于双随机解,并没有充分利用解的本身信息,不利于逐维改进策略进行局部搜索.与此同时,公式(7)中的缩放因子 r 取(0,1)区间的均匀分布随机数,在一定程度上限制了偏好随机游动的搜索方向.然而,r 可取(−1,1)区间的均匀分布随机数以增加反向搜索,达到双向搜索,提高局部搜索能力.因此,DDICS 算法将公式(7)修改为公式(8):

$$X_{g+1,i}=X_{g,i}+r(X_{g,j}-X_{g,i}) \tag{8}$$

其中,r 是(−1,1)区间的随机数,$X_{g,j}$ 是表示第 g 代的随机解.

整个改进算法的流程可抽象为算法 2.

算法 2. DDICS.

Begin

初始化种群:n host nests X_i(i=1,2,…,n);

评价 X_i(i=1,2,…,n).

While (不满足停止条件)

For i=1 to n

采用公式(6)生成新的解 $X_{g+1,i}$;

选择候选解 $X_{g,i}$;

If ($X_{g+1,i}$ 优于 $X_{g,i}$)

$X_{g+1,i}$ 替代 $X_{g,i}$.

End

End

For i=1 to n

```
            If (r>p_a)
                For j=1 to d
                    X_c=X_i;
                        X_c[j]=X_c[j]+r(X[k,j]–X[i,j]);
                    If (X_c 优于 X_i)
                        接受更新:X_i=X_c;
                    End
                End
            End
        End
        保留最好的解.
    End
End
```

3 实验与结果

为了观察改进算法求解多维函数优化问题的收敛速度和解的质量,实验选择 3 类测试函数,见表 1.第 1 类是单峰函数,其中f_1是简单的单峰函数,f_2在二、三维时是简单的单峰函数,但在更多维时可看作多峰函数[21−23];第 2 类是具有众多局部极值点的多峰函数[24];第 3 类是具有变换或旋转的单峰或多峰函数[25].

Table 1 Test functions

表 1 测试函数

	测试函数	搜索空间	误差阈值	函数名称
单峰	$f_1(x)=\sum_{i=1}^{D}x_i^2$	[−100 100]	10^{-6}	Sphere[24]
	$f_2(x)=\sum_{i=1}^{D-1}(100(x_{i+1}-x_i^2)^2+(1-x_i)^2)$	[−100 100]	10^{-6}	Rosenbrock[24]
多峰	$f_3(x)=20+\exp(1)-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)$	[−32 32]	10^{-6}	Ackley[24]
	$f_4(x)=\sum_{i=1}^{D}\frac{x_i^2}{4000}-\prod_{i=1}^{D}\cos\frac{x_i}{\sqrt{i}}+1$	[−600 600]	10^{-6}	Griewank[24]
	$f_5(x)=\sum_{i=1}^{D}(x_i^2-10\cos(2\pi x_i)+10)$	[−5.12 5.12]	10^{-6}	Rastrigin[24]
	$f_6(x)=418.9829n+\sum_{i=1}^{n}\left(x_i\sin\left(\sqrt{\lvert x_i\rvert}\right)\right)$	[−500 500]	10^{-6}	Schwefel[24]
	$f_7(x)=\frac{\pi}{n}\left\{10(\sin(\pi y_1))^2+\sum_{i=1}^{n-1}((y_i-1)^2[1+10(\sin(\pi y_{i+1}))^2])+(y_n-1)^2\right\}+\sum_{i=1}^{n}u(x_i,10,100,4)$ where $y_i=1+\frac{1}{4}(x_i+1)$, $u(x_i,a,k,m)=\begin{cases}k(x_i-a)^m, & x_i>a\\ 0, & -a\leqslant x_i\leqslant a\\ k(-x_i-a)^m, & x_i<-a\end{cases}$	[−50 50]	10^{-6}	Generalized Penalized 1[24]
	$f_8(x)=0.1\left\{(\sin(\pi 3x_1))^2+\sum_{i=1}^{n-1}((x_i-1)^2[1+(\sin(\pi 3x_{i+1}))^2])+(x_n-1)^2[1+(\sin(2\pi x_n))2]\right\}+\sum_{i=1}^{n}u(x_i,5,100,4)$	[−50 50]	10^{-6}	Generalized Penalized 2[24]

Table 1 Test functions (Continued)

表 1 测试函数(续)

	测试函数	搜索空间	误差阈值	函数名称
变换旋转	$f_9(x)=\sum_{i=1}^{D}z_i^2+f\_bias_1,\ z=x-o,\ o=[o_1,o_2,...,o_D]$	[−100 100]	10^{-6}	Shifted Sphere[25]
	$f_{10}(x)=\sum_{i=1}^{D-1}(100(z_i^2-z_{i+1})^2+(z_i-1)^2)+f\_bias_6,\ z=x-o+1,\ o=[o_1,o_2,...,o_D]$	[−100 100]	10^{-2}	Shifted Rosenbrock[25]
	$f_{11}(x)=\sum_{i=1}^{D}\frac{z_i^2}{4000}-\prod_{i=1}^{D}\cos\frac{z_i}{\sqrt{i}}+1+f\_bias_7,$ $z=(x-o)\cdot M,\ o=[o_1,o_2,...,o_D],\ M=M'(1+0.3\lvert N(0,1)\rvert)$ M': Linear transformation matrix	[0 600]	10^{-2}	Shifted Rotated Griewank[25]
	$f_{12}(x)=20+\exp(1)-20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D}z_i^2}\right)-\exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi z_i)\right)+f\_bias_8,$ $z=(x-o)\cdot M,\ o=[o_1,o_2,...,o_D]$ M: Linear transformation matrix, condition number=100	[−32 32]	10^{-2}	Shifted Rotated Ackley[25]
	$f_{13}(x)=\sum_{i=1}^{D}(z_i^2-10\cos(2\pi z_i)+10)+f\_bias_9,\ z=x-o,\ o=[o_1,o_2,...,o_D]$	[−5 5]	10^{-2}	Shifted Rastrigin[25]

同时,表 1 给出了各函数的收敛误差阈值[25].函数误差计算见公式(9):

$$Error=f(X)-f(X^*) \tag{1}$$

其中,X 表示算法得到的解,X^*是函数全局最优解.从公式(9)可知:函数值误差越小,解的质量越好.

在实验中,DDICS 算法与 ICS[13],OLCS[18,19]和 CSPSO[16]等改进的 CS 算法,以及 DEahcSPX[24],jDE[26],CLPSO[27]和 CoDE[28]等其他演化算法作性能比较,以便分析改进算法的竞争力.同时,每个算法解的质量以"平均误差±标准差"形式表示,并给出 0.05 显著水平下的双侧 t-检验.

- "≈"表示算法的平均误差与 DDICS 算法的平均误差在 0.05 显著水平下的双侧 t-检验是不显著的,解的质量相似;
- "‡"表示算法的平均误差与 DDICS 算法的平均误差在 0.05 显著水平下的双侧 t-检验是显著的,解的质量比 DDICS 算法要差;
- "†"表示算法的平均误差与 DDICS 算法的平均误差在 0.05 显著水平下的双侧 t-检验是显著的,解的质量比 DDICS 算法要好.

3.1 解的质量分析

表 2 给出了 CS 算法以及 DDICS 算法在 D=30 维空间上的函数值平均误差及标准差.其中,种群的规模 $NP=D$,发现概率 p_a=0.25,最大函数评价次数 FES=10000×D,每个算法独立运行 50 次.

Table 2 Mean function value errors of CS and DDICS (D=30)

表 2 CS 与 DDICS 的平均函数值误差(D=30)

函数	CS	DDICS	函数	CS	DDICS
f_1	8.46E−31±1.19E−30‡	**3.15E−79±7.62E−79**	f_8	4.64E−29±1.37E−28‡	**1.35E−32±1.11E−47**
f_2	1.07E+01±9.44E+00‡	**7.20E−01±1.25E+00**	f_9	4.21E−30±9.91E−30‡	**0.00E+00±0.00E+00**
f_3	3.73E −02±1.84E −01‡	**3.41E−14±3.93e−15**	f_{10}	2.76E+01±2.34E+01‡	**4.18E+00±6.67E+00**
f_4	3.29E−16±1.75E−15‡	**0.00E+00±0.00E+00**	f_{11}	8.30E−04±2.24E−03†	1.97E−02±5.60E−03
f_5	2.54E+01±4.39E+00‡	**0.00E+00±0.00E+00**	f_{12}	2.09E+01±4.67E−02‡	**2.07E+01±5.95E−02**
f_6	1.25E+03±3.04E+02‡	**0.00E+00±0.00E+00**	f_{13}	2.66E+01±5.29E+00‡	**0.00E+00±0.00E+00**
f_7	2.05E−20±7.28E −20‡	**1.57E−32±5.53E−48**			

从表 2 可看出,对单峰函数而言,DDICS 算法与 CS 算法相比,显著提高了解的质量,且具有较高的精度.在有众多局部极值点的多峰函数中,DDICS 算法在 f_3,f_7 和 f_8 函数上显著提高了解的质量,并且在 f_4,f_5 和 f_6 函数上取得全局最优解.在第 3 类函数中,无论是变换的单峰函数 f_9,还是变换的多峰函数 f_{10} 和 f_{13},DDICS 算法都能较好地提高解的质量,特别是在 f_9 和 f_{13} 上获得全局最优解;对旋转且变换的多峰函数 f_{12} 而言,DDI 策略能够有效

提高 CS 算法解的质量.然而对于旋转且变换的多峰函数 f_{11},DDICS 算法虽然获得了较好的解,却未能提高解的质量.DDICS 算法之所以能在大多数的函数上改善解的质量,是因为采用 DDI 策略使算法避免了维间互扰,充分利用单维有用的信息加强算法局部搜索,从而改善算法的性能.而在 f_{11} 上未能改善解的质量,则归因于该函数没有边界,且最优解在初始化搜索空间之外,导致单维的信息无法引导算法展开局部搜索,最终影响解的质量.

3.2 收敛速度分析

表 3 给出了算法收敛于指定误差阈值所需的平均函数评价次数和标准差及成功次数.“–”表示未能收敛于指定误差阈值.由于 CS 算法采用整体更新评价策略存在维间互扰,在一定程度上浪费了函数评价次数,未能获得较好的解,使得收敛较慢.反之,虽然 DDICS 算法逐维评价消耗一定的函数评价次数,但逐维更新能够加强局部求精能力,有利于获得较好的解,从而加快算法的收敛.从表 3 的实验结果也可以看出:对单峰函数而言,CS 算法和 DDICS 算法都只在 f_1 函数上收敛于指定误差阈值,但借助于平均函数评价次数,DDICS 算法明显具有较快的收敛速度.在复杂的多峰函数中,针对 f_4,f_7 和 f_8 函数,两算法都以相同的成功次数收敛于指定误差阈值,但根据平均函数评价次数,DDICS 算法以较快的速度收敛.对于 f_3,f_5 和 f_6 函数,借助于平均函数评价次数,DDICS 算法快速收敛于指定误差阈值,而且借助于成功次数,DDICS 算法的收敛具有较好的稳定性.针对 f_{12} 函数,由于 DDICS 算法未能获得较好的解,从而几乎未能收敛于指定误差阈值,但在 f_9 和 f_{11} 函数上,DDICS 算法快速且稳定地收敛于指定误差阈值.

Table 3 Mean FES of CS and DDICS to reach specified error threshold (D=30)

表 3 CS 与 DDICS 收敛于指定误差阈值的平均函数评价次数(D=30)

函数	CS	DDICS	函数	CS	DDICS
f_1	87890.9±2706.3(50)‡	**38764.2±1452.8(50)**	f_7	148538.2±20898.2(50)‡	**33410.0±1718.5(50)**
f_3	162313.3±34687.8(48)‡	**64676.3±1265.8(50)**	f_8	99216.0±4868.5(50)‡	**37549.4±1535.1(50)**
f_4	124996.5±22839.0(50)‡	**49461.9±6207.9(50)**	f_9	91621.1±3116.0(50)‡	**40536.1±1155.5(50)**
f_5	–	**78996.3±6353.0(50)**	f_{11}	155926.6±25420.5(50)†	283974.0±0.0(1)
f_6	–	**115460.4±11417.7(50)**	f_{13}	–	**88962.7±8390.8(50)**

为了进一步说明 DDICS 算法的收敛速度优于 CS 算法,图 1~图 4 图形化展示两算法收敛于指定误差阈值的部分函数的收敛过程.从图 3~图 6 可以看出:DDICS 算法的收敛过程明显有较好的收敛曲线,而且在前期具有较快的收敛速度,在后期具有较好的求精能力,从而验证了表 2 的结果.

对 f_2,f_{10} 和 f_{12} 函数而言,虽然 DDICS 算法与 CS 算法都未能收敛于指定误差阈值,但是图 5~图 7 说明 DDICS 算法的收敛过程同样有较优的收敛曲线.例如,DDICS 算法优化 f_2 和 f_{10} 函数在早期具有较快的收敛速度,在后期虽然收敛速度放缓,但与 CS 算法相比仍然具有较好的求精能力.在求解 f_{12} 函数过程中,DDICS 算法在早期收敛速度虽显著优于 CS 算法,但随着迭代的进行,表现出较好的收敛速度和求精能力.

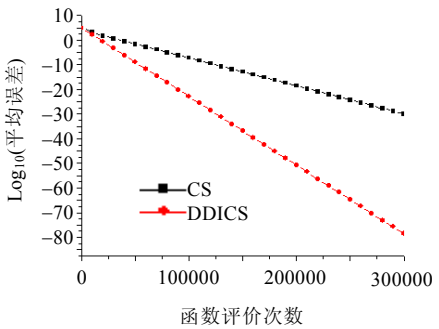

Fig.1 Convergence curves of DDICS and CS for f_1 function

图 1 DDICS 和 CS 求解 f_1 函数的收敛曲线

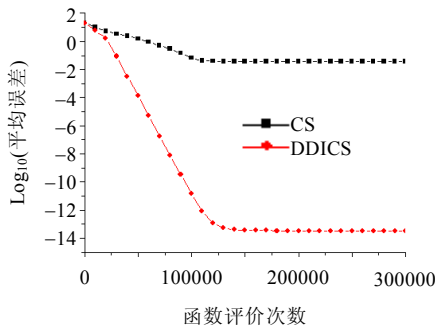

Fig.2 Convergence curves of DDICS and CS for f_3 function

图 2 DDICS 和 CS 求解 f_3 函数的收敛曲线

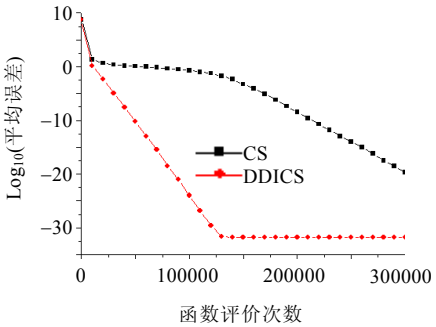

Fig.3 Convergence curves of DDICS and CS for f_7 function

图 3 DDICS 和 CS 求解 f_7 函数的收敛曲线

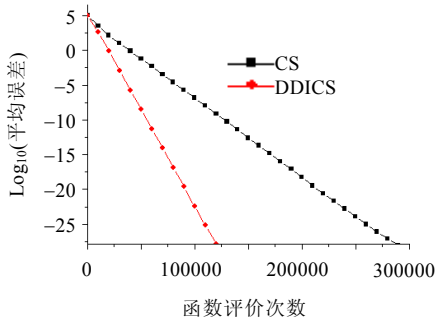

Fig.4 Convergence curves of DDICS and CS for f_9 function

图 4 DDICS 和 CS 求解 f_9 函数的收敛曲线

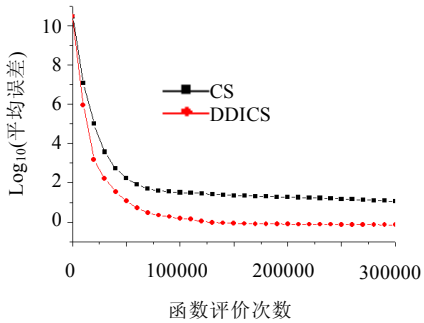

Fig.5 Convergence curves of DDICS and CS for f_2 function

图 5 DDICS 和 CS 求解 f_2 函数的收敛曲线

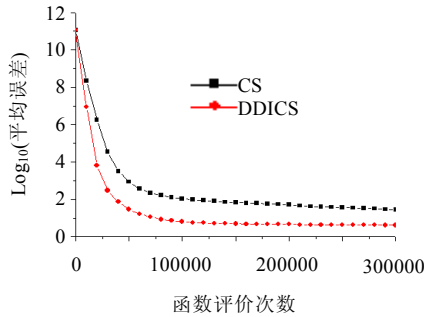

Fig.6 Convergence curves of DDICS and CS for f_{10} function

图 6 DDICS 和 CS 求解 f_{10} 函数的收敛曲线

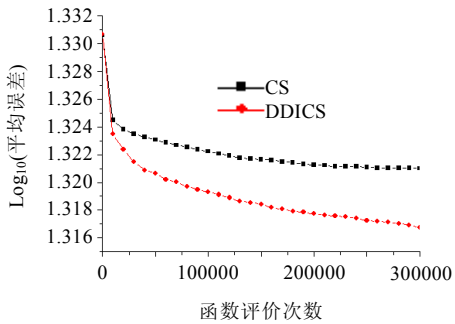

Fig.7 Convergence curves of DDICS and CS for f_{12} function

图 7 DDICS 和 CS 求解 f_{12} 函数的收敛曲线

3.3 维数变化分析

为了观察并分析 DDICS 算法随维数变化的性能,表 4 列出了 DDICS 算法在不同维数上的平均函数值误差和标准差及成功次数.由于第 3 类函数最高维是 50 维,表 4 给出 50 维的实验结果,而其他函数分别完成 50 维、100 维和 200 维的实验结果.其中,规模 $NP=D$,发现概率 $p_a=0.25$,$FES=10000\times D$,每个算法独立运行 50 次.

从表 4 可知,在 50 维上,DDICS 算法性能可以得到与第 3.1 节相同的结论.在 100 维和 200 维上,CS 算法收

敛于指定误差阈值的成功次数越来越少,特别是在 200 维的空间上已无法成功地收敛于指定的误差阈值.反之,DDICS 算法在 100 维和 200 维的空间上仍然有部分函数稳定收敛于指定误差阈值,体现出算法在求解该类函数上具有较强的稳定性.随着维数的增加,借助于平均函值数误差,DDICS 算法与 CS 算法相比仍有更高质量的解.

Table 4 Mean function value errors with different dimensions

表 4 不同维数的平均函数值误差

D=50					
函数	CS	DDICS	函数	CS	DDICS
f_1	3.61E−17±2.76E−17(50)‡	**9.68E−46±1.1E−45**(50)	f_8	6.00E−15±1.04E−14(50)‡	**1.35E−32±1.11E−47**(50)
f_2	4.51E+01±1.99E+01‡	**5.22E−01±1.13E+00**	f_9	1.20E−16±9.69E−17(50)‡	**0.00E+00±0.00E+00**(50)
f_3	1.91E−02±1.20E−01‡	**6.19E−14±5.52E−15**(50)	f_{10}	6.93E+01±3.18E+01‡	**1.82E+00±2.60E+00**
f_4	1.48E−04±1.05E−03‡	**0.00E+00±0.00E+00**(50)	f_{11}	6.31E−04±8.62E−04(50)†	6.58E−02±1.81E−02
f_5	8.55E+01±1.10E+01‡	**0.00E+00±0.00E+00**(50)	f_{12}	2.09E+01±4.67E−02≈	2.09E+01±4.81E−02
f_6	4.40E+03±3.78E+02‡	**1.84E−11±1.14E−12**(50)	f_{13}	1.25E+02±1.16E+01‡	**0.00E+00±0.00E+00**(50)
f_7	3.88E−04±2.60E−03(16)‡	**9.42E−33±1.38E−48**(50)	–	–	–
D=100			*D*=200		
函数	CS	DDICS	函数	CS	DDICS
f_1	7.01E−07±2.41E−07(46)‡	**4.56E−21±3.23E−21**(50)	f_1	9.34E−02±1.28E−02‡	**1.22E−07±5.92E−08**(50)
f_2	1.13E+02±3.18E+01‡	**7.71E−01±7.74E−01**	f_2	8.05E+02±1.52E+02‡	**1.01E+02±2.45E+01**
f_3	1.83E+00±3.42E−01‡	**1.50E−10±3.88E−11**(50)	f_3	2.29E+00±1.33E−01‡	**5.88E−04±1.29E−04**
f_4	1.18E−05±3.69E−05‡	**0.00E+00±0.00E+00**(50)	f_4	3.71E−02±1.60E−02‡	**1.17E−07±6.39E−08**(50)
f_5	2.86E+02±2.22E+01‡	**9.53E−02±2.46E−01**(13)	f_5	8.68E+02±5.22E+01‡	**2.57E+01±2.16E+00**
f_6	1.40E+04±7.42E+02‡	**4.47E+02±1.14E+02**	f_6	3.35E+04±1.39E+03‡	**4.21E+03±2.84E+02**
f_7	3.10E−01±1.83E−01‡	**4.67E−23±3.06E−23**(50)	f_7	7.87E−01±1.60E−01‡	**4.43E−10±2.12E−10**(50)
f_8	6.65E−04±1.23E−03(20)‡	**2.12E−21±1.32E−21**(50)	f_8	1.36E+01±4.33E+00‡	**3.29E−08±1.55E−08**(50)

3.4 与其他CS算法比较

表 5 给出了 DDICS 算法与其他改进的 CS 算法的性能比较.其中,维数 D=30,规模 NP=D,FES=10000×D,每个算法独立运行 50 次,其他改进算法的参数根据其文献设置.从表中可知,各改进算法针对不同的函数,呈现的性能不一样.针对 f_1 函数,DDICS 算法的性能弱于 OLCS 算法,但优于其他算法;而在 f_2 函数上,DDICS 算法的性能最优.在复杂的多峰函数中,虽然 DDICS 算法优化 f_3 函数的性能逊色于 ICS 和 OLCS 算法,但在其他的函数上,DDICS 算法获得的性能是最优的,特别是在 f_4,f_5 和 f_6 函数上,DDICS 算法稳定收敛于全局最优解.针对变换或旋转的函数,DDICS 算法优化 f_{11} 函数的性能最不理想,但在 f_9,f_{10},f_{12} 和 f_{13} 函数上的性能是最优的,而且在 f_9 和 f_{13} 函数上,DDICS 算法稳定收敛于全局最优解.借助于表中“‡”的统计结果,DDICS 算法总体上优于其他改进算法,体现出较强的竞争力.

Table 5 Comparison results of the DDICS algorithm and other CS algorithms (*D*=30)

表 5 DDICS 算法与其他 CS 算法的比较结果(*D*=30)

函数	ICS	OLCS	CSPSO	DDICS
f_1	6.13E−49±8.19E−49(50)‡	3.20E−128±2.13E−127(50)†	1.03E−44±3.43E−44(50)‡	3.15E−79±7.62E−79(50)
f_2	9.51E+00±3.21E+00‡	2.49E+00±8.45E−01‡	7.21E−01±1.55E+00(4)≈	7.20E−01±1.25E+00
f_3	9.81E−15±3.25E−15(50)†	2.66E−15±0.00E+00(50)†	1.86E−02±1.32E−01(49)‡	3.41E−14±3.93E−15(50)
f_4	0.00E+00±0.00E+00(50)≈	0.00E+00±0.00E+00(50)≈	4.29E−03±5.00E−03(27)‡	0.00E+00±0.00E+00(50)
f_5	1.71E+01±3.76E+00‡	0.00E+00±0.00E+00(50)≈	3.08E+01±1.07E+01‡	0.00E+00±0.00E+00(50)
f_6	7.53E+02±3.12E+02‡	2.13E+03±2.87E+02‡	4.22E+03±7.48E+02‡	0.00E+00±0.00E+00(50)
f_7	1.57E−32±5.53E−48(50)≈	5.01E−30±7.48E−30(50)‡	7.88E−02±1.37E−01(31)‡	1.57E−32±5.53E−48(50)
f_8	1.35E−32±1.11E−47(50)≈	4.46E−29±5.83E−29(50)‡	6.59E−04±2.64E−03(47)‡	1.35E−32±1.11E−47(50)
f_9	0.00E+00±0.00E+00(50)≈	2.41E−26±6.21E−26(50)‡	3.28E−28±3.52E−28(50)‡	0.00E+00±0.00E+00(50)
f_{10}	1.28E+01±4.72E+00‡	2.45E+01±1.99E+01‡	4.50E+00±1.15E+01(23)≈	4.18E+00±6.67E+00
f_{11}	2.03E−03±2.94E−03 (48)†	4.72E−04±1.13E−03(50)†	7.40E−03±1.02E−15(50)†	1.97E−02±5.60E−03(1)
f_{12}	2.09E+01±4.87E−02‡	2.09E+01±5.31E−02‡	2.09E+01±4.96E−02‡	2.07E+01±5.95E−02
f_{13}	1.67E+01±4.62E+00‡	3.54E+01±6.64E+00‡	1.55E+02±2.49E+01‡	0.00E+00±0.00E+00(50)
‡	7	8	10	–
≈	4	2	2	–
†	2	3	1	–

3.5 与其他演化算法比较

DDICS 算法与其他演化算法的比较结果见表 6.其中,维数 D=30,规模 NP=D,FES=10000×D,每种算法独立运行 50 次,其他演化算法的详细参数见文献[24,26−28].针对单峰函数,DDICS 算法稍逊色于 jDE 算法,但优于其他算法.在多峰函数中,DDICS 算法优化 f_3 的性能最差,但在优化其他 5 个函数时的性能最优.对于变换与旋转的函数,DDICS 算法性能优于 CLPSO,稍优于 jDE 算法和 DEahcSPX 算法,略逊色于 CoDE 算法.表 6 中的"‡"统计结果说明 DDICS 算法总体上具有一定的竞争力.

Table 6 Comparison results of the DDICS algorithm and other evolution algorithms (D=30)

表 6 DDICS 算法与其他演化算法的比较结果(D=30)

函数	CLPSO	DEahcSPX	jDE	CoDE	DDICS
f_1	2.14E−40±1.54E−40‡	1.75E−31±4.99E−31‡	0.00E+00±0.00E+00†	3.04E−67±4.41E−67‡	3.15E−79±7.62E−79
f_2	4.61E+00±6.32E+00‡	4.52E+00±1.55E+01≈	8.77E−01±1.67E+00≈	3.19E−01±1.09E+00≈	7.20E−01±1.25E+00
f_3	7.60E−15±1.44E−15†	2.66E−15±0.00E+00†	6.68E−15±2.84E−15†	3.55E−15±0.00E+00†	3.41E−14±3.93E−15
f_4	0.00E+00±0.00E+00≈	2.07E−03±5.89E−03‡	2.80E−03±8.85E−03‡	1.48E−04±1.05E−03‡	0.00E+00±0.00E+00
f_5	0.00E+00±0.00E+00≈	2.14E+01±1.23E+01‡	2.19E−01±4.62E−01‡	0.00E+00±0.00E+00≈	0.00E+00±0.00E+00
f_6	4.74E+00±2.34E+01‡	4.70E+02±2.96E+02‡	7.82E+01±8.83E+01‡	2.37E+00±1.67E+01‡	0.00E+00±0.00E+00
f_7	1.57E−32±5.53E−48≈	2.07E−02±8.46E−02‡	2.07E−03±1.47E−02‡	2.07E−03±1.47E−02‡	1.57E−32±5.53E−48
f_8	1.35E−32±1.11E−47≈	1.71E−31±5.35E−31‡	5.68E−02±2.82E−01‡	1.35E−32±1.11E−47≈	1.35E−32±1.11E−47
f_9	0.00E+00±0.00E+00≈	0.00E+00±0.00E+00≈	8.05E−29±2.14E−28‡	0.00E+00±0.00E+00≈	0.00E+00±0.00E+00
f_{10}	8.95E+00±1.64E+01≈	3.84E+00±3.75E+00≈	7.97E−01±1.61E+00†	7.97E−02±5.64E−01†	4.18E+00±6.67E+00
f_{11}	4.51E−01±8.47E−02‡	7.39E−03±6.32E−03†	1.75E−02±1.22E−02≈	1.08E−02±1.00E−02†	1.97E−02±5.60E−03
f_{12}	2.09E+01±5.72E−02‡	2.09E+01±1.12E−01‡	2.09E+01±5.89E−02‡	2.02E+01±1.34E−01†	2.07E+01±5.95E−02
f_{13}	0.00E+00±0.00E+00≈	2.04E+01±8.19E+00‡	3.38E−01±8.67E−01‡	3.98E−02±2.81E−01‡	0.00E+00±0.00E+00
‡	5	8	8	5	–
≈	7	3	2	4	–
†	1	2	3	4	–

4 结束语

布谷鸟搜索算法是一种新颖的仿生优化算法,具有简单和高效的特点,并被成功地应用于经典理论研究和工程应用领域.算法求解连续函数优化问题,采用整体更新评价解,由于存在维间互扰现象,算法的局部求精能力和收敛速度将受到一定的影响.本文引入逐维更新评价策略,并改写偏好随机游动组件的步长公式,使得算法在逐维更新过程中充分利用单维的进化信息,加强局部搜索,改善算法的收敛速度和解的质量.实验结果说明:对于单峰函数、多峰函数以及变换旋转的函数,改进算法都能显著提高其收敛速度和解的质量.不同维数的实验结果说明:随着维数的增加,逐维改进策略稳定改善算法的收敛速度和解的质量.与相关的改进 CS 算法及其他演化算法相比,结果显示,改进算法在求解多维函数优化问题上是具有竞争力的.

所提出的改进策略虽然能够有效改善 CS 算法的性能,但针对无边界的函数却未能有效改善收敛速度和解的质量;而且随着维数的增加,迭代将消耗大量的函数评价次数.因此,后续工作将考虑分块更新并评价的策略.此外,还需要在更多的实际优化问题中应用改进算法进行分析并验证.

致谢 在此,我们感谢本文的评审专家,同时也感谢向我们提供相关算法源代码以及实验数据的学者.

References:

[1] Kennedy J, Eberhart RC. Particle swarm optimization. In: Proc. of the IEEE Int'l Conf. on Neural Networks. Piscataway: IEEE Inc., 1995. 1942−1948. [doi: 10.1109/ICNN.1995.488968]

[2] Eberhart RC, Kennedy J. A new optimizer using particle swarm theory. In: Proc. of the 6th Int'l Symp. on Micro Machine and Human Science. Piscataway: IEEE Inc., 1995. 39−43. [doi: 10.1109/MHS.1995.494215]

[3] Dorigo M, Maniezzo V, Colorni A. The ant system: Optimization by a colony of cooperating agents. IEEE Trans. on Systems, Man, and Cybernetics, Part B, 1996,26(1):29−41. [doi: 10.1109/3477.484436]

[4] Clerc M. Particle Swarm Optimization. Landon: Wiley-ISTE, 2006.

[5] Karaboga D. An idea based on honey bee swarm for numerical optimization. Technical Report, TR-06, Computer Engineering Department, Engineering Faculty, Erciyes University, 2005.

[6] Chen H, Cui DW, Cui YA, Tao YQ, Liang K. Ethnic group evolution algorithm. Ruan Jian Xue Bao/Journal of Software, 2010, 21(5):978−990 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3484.htm [doi: 10.3724/SP.J.1001.2010.03484]

[7] Yang XS, Deb S. Cuckoo search via Lévy flights. In: Abraham A, Carvalho A, Herrera F, *et al*., eds. Proc. of the World Congress on Nature and Biologically Inspired Computing (NaBIC 2009). Piscataway: IEEE Publications, 2009. 210−214. [doi: 10.1109/NABIC.2009.5393690]

[8] Yang XS, Deb S. Engineering optimisation by cuckoo search. Int'l Journal of Mathematical Modeling and Numerical Optimisation, 2010,1(4):330−343. [doi: 10.1504/IJMMNO.2010.03543]

[9] Yang XS, Deb S. Multi-Objective cuckoo search for design optimization. Computers & Operations Research, 2013,40(6): 1616−1624. [doi: 10.1016/j.cor.2011.09.026]

[10] Civicioglu P, Besdok E. A conceptual comparison of the cuckoo search, particle swarm optimization, differential evolution and artificial bee colony algorithms. Artificial Intelligence Review, 2013,39(4):315−346. [doi: 10.1007/s10462-011-9276-0]

[11] Walton S, Hassan O, Morgan K, Brown MR. Modified cuckoo search: A new gradient free optimization algorithm. Chaos, Solitons & Fractals, 2011,44(9):710−718. [doi: 10.1016/j.chaos.2011.06.004]

[12] Tuba M, Subotic M, Stanarevic N. Modified cuckoo search algorithm for unconstrained optimization problems. In: Leandre R, Demiralp M, Tuba M, *et al*., eds. Proc. of the European Computing Conf. (ECC 2011). Athens: WSEAS Press, 2011. 263−268.

[13] Valian E, Mohanna S, Tavakoli S. Improved cuckoo search algorithm for global optimization. Int'l Journal of Communications and Information Technology, 2011,1(1):31−44.

[14] Layeb A, Boussalia SR. A novel quantum inspired cuckoo search algorithm for bin packing problem. Int'l Journal of Information Technology and Computer Science, 2012,4(5):58−67. [doi: 10.5815/ijitcs.2012.05.08]

[15] Ghodrati A, Lotfi S. A hybrid CS/PSO algorithm for global optimization. In: Pan JS, Chen SM, Nguyen NT, eds. Proc. of the ACIIDS 2012, Part III. LNAI 7198, Berlin, Heidelberg: Springer-Verlag, 2012. 89−98. [doi: 10.1007/978-3-642-28493-9_11]

[16] Wang F, He XS, Luo LG, Wang Y. Hybrid optimization algorithm of PSO and cuckoo search. In: Proc. of the 2nd Int'l Conf. on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC). Piscataway: IEEE Inc., 2011. 1172−1175. [doi: 10.1109/AIMSEC.2011.6010750]

[17] Srivastava PR, Khandelwal R, Khandelwal S, Kumar S, Ranganatha SS. Automated test data generation using cuckoo search and tabu search (CSTS) algorithm. Journal of Intelligent Systems, 2012,21(2):195−224. [doi: 10.1515/jisys-2012-0009]

[18] Li XT, Yin MH. Parameter estimation for chaotic systems using the cuckoo search algorithm with an orthogonal learning method. Chinese Physics B, 2012,21(5):050507-1−050507-6. [doi: 10.1088/1674-1056/21/5/050507]

[19] Li XT, Wang JN, Yin MH. Enhancing the performance of cuckoo search algorithm using orthogonal learning method. Neural Computing & Applications. Publised online: 09 February 2013. [doi: 10.1007/s00521-013-1354-6]

[20] Zhong YW, Liu X, Wang LJ, Wang CY. Particle swarm optimization algorithm with iterative improvement strategy for multi-dimensional function optimization problems. Int'l Journal of Innovative Computing and Application, 2012,4(3-4):223−232. [doi: 10.1504/IJICA.2012.050051]

[21] Hansen N, Ostermeier A. Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation, 2001,9(2): 159−195. [doi: 10.1162/106365601750190398]

[22] Deb K, Anand A, Joshi D. A computationally efficient evolutionary algorithm for real-parameter optimization. Evolutionary Computation, 2002,10(4):371−395. [doi: 10.1162/106365602760972767]

[23] Shang YW, Qiu YH. A note on the extended rosenbrock function. Evolutionary Computation, 2006,14(1):119−126. [doi: 10.1162/106365606776022733]

[24] Noman N, Iba H. Accelerating differential evolution using an adaptive local search. IEEE Trans. on Evolutionary Computation, 2008,12(1):107−125. [doi: 10.1109/TEVC.2007.895272]

[25] Suganthan PN, Hansen N, Liang JJ, Deb K, Chen YP, Auger A, Tiwari S. Problem definitions and evaluation criteria for the CEC2005 special session on real-parameter optimization. Technical Report, KanGAL Report #2005005, Singapore: Kanpur Genetic Algorithms Laboratory, Nanyang Technological University, 2005.

[26] Brest J, Greiner S, Boskovic B, Mernik M, Zumer V. Self-Adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Trans. on Evolutionary Computation, 2006,10(6):646−657. [doi: 10.1109/TEVC.2006.872133]

[27] Liang JJ, Qin AK, Suganthan PN, Baskar S. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Trans. on Evolutionary Computation, 2006,10(3):281−295. [doi: 10.1109/TEVC.2005.857610]

[28] Wang Y, Cai ZX, Zhang QF. Differential evolution with composite trial vector generation strategies and control parameters. IEEE Trans. on Evolutionary Computation, 2011,15(1):55−66. [doi: 10.1109/TEVC.2010.2087271]

附中文参考文献:

[6] 陈皓,崔杜武,崔颖安,陶永芹,梁琨.族群进化算法.软件学报,2010,21(5):978−990. http://www.jos.org.cn/1000-9825/3484.htm [doi: 10.3724/SP.J.1001.2010.03484]

王李进(1977－),男,福建泉州人,博士,副教授,主要研究领域为计算智能及应用.
E-mail: lijinwang@fafu.edu.cn

钟一文(1968－),男,博士,教授,CCF 会员,主要研究领域为计算智能,生物信息学.
E-mail: yiwzhong@fafu.edu.cn

尹义龙(1972－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为机器学习,数据挖掘,图像处理.
E-mail: ylyin@sdu.edu.cn