ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.21, No.11, November 2010, pp.2790−2801
doi: 10.3724/SP.J.1001.2010.03673

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

一种高效的复杂系统遗传算法*

庄 健[1+], 杨清宇[2], 杜海峰[3], 于德弘[1]

[1](西安交通大学 机械工程学院,陕西 西安 710049)

[2](西安交通大学 电子与信息工程学院,陕西 西安 710049)

[3](西安交通大学 公共管理学院,陕西 西安 710049)

High Efficient Complex System Genetic Algorithm

ZHUANG Jian[1+], YANG Qing-Yu[2], DU Hai-Feng[3], YU De-Hong[1]

[1](School of Mechanical Engineering, Xi'an Jiaotong University, Xi'an 710049, China)

[2](School of Electronic and Information Engineering, Xi'an Jiaotong University, Xi'an 710049, China)

[3](School of Public Policy and Administration, Xi'an Jiaotong University, Xi'an 710049, China)

+ Corresponding author: E-mail: zhuangjian@mail.xjtu.edu.cn, http://www.xjtu.edu.cn

Zhuang J, Yang QY, Du HF, Yu DH. High efficient complex system genetic algorithm. *Journal of Software*, 2010,21(11):2790−2801. http://www.jos.org.cn/1000-9825/3673.htm

Abstract: In order to overcome the problems of genetic algorithm, such as the low efficiency, genetic algorithm is redesigned by the complex system theory in this paper. First, the selecting operator is rebuilt by the power law, which is considered to be the self-organized criticality of complex system and sound distribution system of energy. Second, the crossover operator is redesigned by the characteristic of a self-learning complex system. Third, the generation strategy is improved by the mechanism of feedback. Finally, the gene floating operator is added to the algorithm. Because all operators are balanced with each other and restrict each other, the newly designed algorithm, complex system genetic algorithm (CSGA), improves efficiency and premature markedly. At last, experiments show that the CSGA is capable of dealing with high dimensional global optimization problems.

Key words: genetic algorithm; complex system; power law; gene floating

摘 要: 针对遗传算法效率低等问题,基于复杂系统理论对其作了以下改进:首先,用反映复杂系统能量分布的幂律法则改造了选择算子;其次,引入复杂系统自学习特性重新设计了交叉算子;再次,采用反馈机理改进了更新策略;最后,在算法中增加了基因漂流算子.通过上述改造,复杂系统遗传算法各个算子相互平衡、相互制约,有效地抑制了遗传算法的"早熟",并在很大程度上提高了算法的效率.进一步通过实验结果表明,该算法在高维优化中具有较好的性能.

关键词: 遗传算法;复杂系统;幂律法则;基因漂流

中图法分类号: TP18 **文献标识码**: A

* Supported by the National Natural Science Foundation of China under Grant No.50705073 (国家自然科学基金); the Shaanxi Provincial Natural Science Foundation of China under Grant No.2006E224 (陕西省自然科学基金)

Received 2008-12-08; Revised 2009-04-10; Accepted 2009-07-07

遗传算法(genetic algorithm)由美国密执安大学(University of Michigan)的 Holland 教授于 1975 年提出,是当今影响最为广泛的进化计算(evolutionary computation)方法之一,其模拟生物进化过程成功地解决了工业、工程等领域一些复杂的优化问题.遗传算法具有许多优点,但存在着搜索效率较低、早熟等缺点.其一般由交叉、变异、选择等基本算子构成,相对于复杂的生物生命活动及其进化规律而言,目前的遗传算法只是对其过程与功能的简单模拟而已.复杂系统理论是系统科学中的一个前沿方向,其强调用整体论和还原论相结合的方法去分析系统.就生命系统本身而言,其具有非均匀性、非线性、自适应性和网络性等特性,无疑是一类巨型的复杂系统.因此,本文考虑如何以复杂系统的观点对遗传算法进行改进,以获得更好的计算性能[1−6].

为了提高遗传算法的效率,许多学者做了非常有益的研究工作,大致可以分为两大类:其一,主要集中在研究遗传算法内部算子机理改善和参数合理的选择上;其二,主要与其他算法结合来提高遗传算法的效率.目前,研究工作主要集中在遗传算法局部的算子或机制上.实际上,遗传算法效率不是单一算子或参数的反映,而是算子、参数之间相互制约、相互平衡的结果.例如,变异算子的效率不仅取决于变异参数或是变异机制,还会受到种群分布情况的影响.如果种群分布情况为相对某一较小区域比较集中时,那么变异算子对该区域的搜索就比较全面;而如果种群分布相对分散,那么变异算子的搜索也就相对随机.但是,种群分布的情况又取决于其他的算子或参数(种群规模、交叉算子、更新策略等).

在真实的自然界中,生物种群数目庞大、种群分布多样,充分发挥了生物进化的效率,并且生物进化过程不仅仅是同一物种内部相互作用,还包括了不同物种之间的关系(共生、寄生)以及物种与环境之间的互动.文献[7]系统地总结了生物在进化过程中合作的 5 种法则.可见,自然界中的生物系统是一类巨型的复杂系统,而模拟其进化活动的遗传算法也应属于复杂系统的范畴.所以,从复杂系统理论来重新考虑遗传算法的结构和参数之间的平衡关系,将复杂系统理论中自组织、自适应等特性应用到遗传算法的设计中,对其性能改善与提高将会有巨大的作用.本文基于上述的理念,将采用反映复杂系统能量分布的幂律法则来改造选择算子,通过合理选择遗传算法中参与基因传递的个体以提高算法的效率;基于生物进化中环境与生物体的作用与反作用机制和“趋同进化”现象设计了环境-基因双演化的交叉算子,使得交叉算子不仅是两个父代个体之间的信息交互,还引入了一个全新的环境变量,该变量包括了个体在进化过程中的信息,从而使遗传算法具备了学习能力,提高了算法的搜索速度;利用反馈机理设计了自动调节更新策略,保证了小规模群体遗传算法种群的多样性,在种群整体趋于进化时,更多个体采用“优胜劣汰”的更新策略,保证了种群分布的集中性,提高了算法局部搜索效率,而在种群整体进化趋于停顿时,更多的个体采用了“子代直接取代父代”的更新策略,引入更多的新生基因,提高了算法的全局搜索能力;通过生物体之间基因漂流现象启发,在遗传算法中新增了基因漂流算子,该算子通过在种群中扩散最优个体的等基因位,不仅能够加速算法的搜索速度,而且对算法的收敛性有显著影响.我们将通过上述改造的遗传算法称为复杂系统遗传算法(complex system genetic algorithm,简称 CSGA).在该算法中,所有的算子与机制相互平衡、相互制约.实验结果表明,本文设计的 CSGA 在高维优化中具有较好的性能.

1 复杂系统遗传算法的定义

复杂系统遗传算法所要解决的无约束的全局优化问题可以定义为

$$\left.\begin{array}{ll}\text{Minimize} & f(\boldsymbol{X})\\ \text{Subject to} & X\in\Omega\end{array}\right\} \tag{1}$$

式中,f 是实值函数,$\boldsymbol{X}=\{x_1,x_2,\ldots,x_m\}^T$ 为在 $\boldsymbol{R}^N$ 空间内变量集,Ω为所有可行解集合.

不失一般性,对复杂系统遗传算法中的符号作如下定义:$\boldsymbol{X}$ 为变量集,其包含 $\boldsymbol{X}=\{x_1,x_2,...,x_m\}_m^T$ 个变量,其中,每个变量 $x_i=c_{i1}c_{i2}\ldots c_{il}$,由长度为 l 的符号组成;$\boldsymbol{C}$ 为限长度字符集合,可定义为 $\boldsymbol{C}=\{a_1,a_2,\ldots,a_k\}$的集合.因此,变量 $\boldsymbol{X}$ 可以写成

$$\boldsymbol{X}=\begin{bmatrix} c_{11}c_{12}...c_{1l} \\ c_{21}c_{22}...c_{2l} \\ ... \\ c_{m1}c_{m2}...c_{ml} \end{bmatrix} \tag{2}$$

将 $c_{11},\ldots,c_{ml}$ 称为遗传基因,c_{ij} 称为基因位,c_{ik} 与 c_{jk} 称为 x_i 变量与 x_j 变量的等基因位(equi-locus).本文采用十进制编码,所有变量归一化在[0,1]区间,所以 $\boldsymbol{C}$={0,1,3,4,5,6,7,8,9},变量 x_i 译码可以写为

$$x_i=c_{i1}10^{-1}+c_{i2}10^{-2}+...+c_{il}10^{-l} \tag{3}$$

算法群体空间记作 $\vec{\boldsymbol{X}}=\{\boldsymbol{X}_1,\boldsymbol{X}_2,\ldots,\boldsymbol{X}_N\}$,$N$ 称为种群规模.

2 复杂系统遗传算法的设计

2.1 幂律法则选择算子

幂律法则(power law)的现象在 100 多年前即被发现,诸如都市人口、网站规模、英文字符出现频率、国民生产总值等都符合幂律法则.该法则被认为是复杂系统自组织能力的表现,是系统能量合理分配的一种体现.在生物进化中,总是少数的高适应度个体占有更多的资源,这与幂律法则现象基本一致,所以我们在遗传算法的选择算子设计中通过借鉴互联网的网站规模分布模型引入幂律法则[8].由于数值模拟的进化计算中种群规模有限(本文中种群规模为 N=50),为了充分利用种群内部模式,本文算法对父代的选择采用幂律法则,而对母代的选择则采用顺序选择.因此,在 t 代时 $\boldsymbol{X}_i$ 被选为父代个体的概率为

$$\left.\begin{aligned} P\{T_s(\vec{\boldsymbol{X}}^t)=\boldsymbol{X}_i^t\}=\frac{10^{-2.03rank(f(\boldsymbol{X}_i^t))}}{\sum_{j=1}^{n}10^{-2.03rank(f(\boldsymbol{X}_j^t))}} \\ rank(f(\boldsymbol{X}_{\max}^t))=0, rank(f(\boldsymbol{X}_{\min}^t))=N-1 \end{aligned}\right\} \tag{4}$$

式中,T_s 为选择算子,$rank()$为排序函数(在 N 规模种群中,适应度值最大的排序函数值为 0,最小的排序函数值为 N–1,无并列排名).

基于幂律法则的选择算子使得群体中的最优个体被选为父代个体的概率远远大于群体中其他的个体.当群体规模 N=50 时,最优个体被选为交配父代的概率接近于 1.其优点在于能够充分搜索每代种群中最优个体内所含的模式,可以加速算法的收敛;缺点则是易“早熟”.因此,基于幂律法则的选择算子必须与算法中其他算子配合使用,才能保证算法不出现“早熟”.

2.2 环境——基因双演化交叉算子

一般普通遗传算法的交叉算子只考虑父代的基因编码形式,而往往忽略了个体的学习能力以及环境对个体的影响.自然界中的不同种类生物如果生活在条件相同的环境中,在同样选择压力的作用下,有可能产生功能相同或十分相似的形态结构以适应环境,该现象称为趋同进化(convergent evolution)[9],如鲸、海豚等和鱼类的亲缘关系很远,前者是哺乳类,后者是鱼类,但都具有相似的外形.显然,这就是环境对生物进化的影响;而生物的学习能力对生物进化历程也具有非常大的影响力,虽然“魏斯曼障碍(Weismann barrier)”[10]从分子结构上论证了任何后天学习得到的特性无法对遗传基因产生影响,但是生物学习能力却可以通过修复自身周围的环境等方法达到加速或改变自身的进化.生物修复环境的过程称为生态位构建(niche construction)[11],比如,人类对工具的使用大大加速了人类的文明进化历程.因此,环境-基因双演化交叉算子引入了一个环境变量,每个个体能够修复自身的环境,再通过环境变量反作用于自身基因编码,进而体现群体在进化过程中的学习能力,并采用“趋同进化”作为个体修改自身环境变量的规则.环境-基因双演化交叉算子具体描述如下:

$$T_c(\boldsymbol{X}_k^{t+1}\mid \boldsymbol{X}_i^t,\boldsymbol{X}_j^t)=\begin{Bmatrix}\boldsymbol{X}_i^t+r_1\boldsymbol{E}^t,\ p_e\leqslant 0.5\\ \boldsymbol{X}_j^t+r_1\boldsymbol{E}^t,\ p_e>0.5\end{Bmatrix}$$
$$\boldsymbol{E}^t=\begin{Bmatrix}r_2\boldsymbol{E}^{t-1}+r_3(\boldsymbol{X}_i^t-\boldsymbol{X}_j^t),\ f(\boldsymbol{X}_i^t)>f(\boldsymbol{X}_j^t)\\ r_2\boldsymbol{E}^{t-1}+r_3(\boldsymbol{X}_j^t-\boldsymbol{X}_i^t),\ f(\boldsymbol{X}_j^t)>f(\boldsymbol{X}_i^t)\end{Bmatrix} \tag{5}$$

其中,$\boldsymbol{E}$ 为环境变量集,r_1 为学习系数,r_2 为遗忘系数,r_3 为修改系数,t 为进化代数,p_e 为规则选择概率:

$$p_e=random(0,1).$$

环境-基因双演化交叉算子其目的有二:其一,通过幂律选择算子与交叉算子共同作用在群体中较优个体周围探索前进方向,从而改善交叉算子效率;其二,通过环境变量获取进化过程中的历史经验,使算法在一定程度上具有学习能力.公式(5)中,$\boldsymbol{E}$ 为环境变量集,能够积累个体进化过程的经验.因此,通过环境——基因双演化交叉算子从某种程度上使得个体具备了学习能力.

2.3 自适应更新策略

一般情况下,遗传算法的种群规模 N 有限,在如此小规模的种群情况下,既要保证种群的多样性,又要防止种群的“早熟”是一项非常困难的任务.为此,在本文的遗传算法中引入了自适应更新策略,采用两种群体更新策略,具体描述如下:

$$T_r(\boldsymbol{X}_i^{t+1},\boldsymbol{X}_i^t,\hat{\boldsymbol{X}}_i^t)=\begin{cases}\boldsymbol{X}_i^{t+1}=\hat{\boldsymbol{X}}_i^t,\ f(\hat{\boldsymbol{X}}_i^t)>f(\boldsymbol{X}_i^t)\\ \boldsymbol{X}_i^{t+1}=\hat{\boldsymbol{X}}_i^t,\ f(\hat{\boldsymbol{X}}_i^t)<f(\boldsymbol{X}_i^t),\ Np_r<i\leqslant N\\ \boldsymbol{X}_i^{t+1}=\boldsymbol{X}_i^t,\ \text{otherwise}\end{cases} \tag{6}$$

式中,T_r 为更新策略算子,p_r 为更新规模变量,N 为遗传算法种群规模,$\hat{\boldsymbol{X}}_i^t$ 为交叉、变异后所产生的新个体.

公式(6)中,更新规模变量 p_r 亦采用反馈修改方式.当种群逐代进化时,则较多的个体采用“优胜劣汰”的更新策略,促使群体迅速收敛;但当种群陷于局部最优解,进化趋势不明显时,则采用“子代直接取代父代”更新策略,为群体引入更多的基因模式.p_r 的修改方式如下:

$$p_r^{t+1}=\begin{cases}p_r^t+\alpha,\ f(\vec{\boldsymbol{X}}_{\max}^{t+1})>f(\vec{\boldsymbol{X}}_{\max}^t)\\ p_r^t-\beta,\ f(\vec{\boldsymbol{X}}_{\max}^{t+1})\leqslant f(\vec{\boldsymbol{X}}_{\max}^t)\end{cases},\ p_r\in\left[\frac{1}{Np_r},1.0\right] \tag{7}$$

式中,α 为递增量常数,β 为递减量常数.

由于在算法中对种群进行了排序,其 0 号个体始终是种群适应度最大的个体.所以,当 Np_r 的最小值为 1 时,保证了最优个体始终采用“优胜劣汰”更新策略,其实质作用类似于最优个体保存,保证了算法的收敛.

2.4 基因漂流算子

现代人类研究发现,遗传物质 DNA 的复杂性远远超过了人们的预期,DNA 链不仅能够感知其结构的微小变化,还拥有自我修复的能力,人们对其机理与工作模式至今仍知之甚少.现代的 DNA 技术揭示,全球近 30 亿人在 Y 染色体上都拥有一种被科学家称为 M168 的特殊变异,这就意味着他们拥有同一男性祖先[12].由此可见,优势基因位能够在生物进化的过程中得到传播与扩散,并进而再影响生物的进化过程.但在自然进化中,优势基因位的传播过程相对比较缓慢,需要几万年,甚至上百万年的时间.为了加速遗传算法的进化速度,本文借鉴转基因技术中的基因漂流(gene floating)[13]概念,让优势基因位能够在个体基因组中得到迅速传播.

$$T_f(\boldsymbol{X})=\begin{bmatrix}c_{11},\ldots,c_{1k-1},c_{1k}^*,c_{1k+1},\ldots,c_{1l}\\ c_{21},\ldots,c_{2k-1},c_{2k}^*,c_{mk+1},\ldots,c_{2l}\\ \cdots\\ c_{m1},\ldots,c_{mk-1},c_{mk}^*,c_{mk+1},\ldots,c_{ml}\end{bmatrix},\ c_{ik}^*\in\boldsymbol{X}_{best},\ i\subseteq[1,l] \tag{8}$$

式中,T_f 为基因漂流算子,$\boldsymbol{X}_{best}$ 为最优个体.

基因漂流算子具有发现解空间内相似的待优化参数的能力,但是如果解空间内待优化的参数完全相异时,则可能会增加算法的运算成本.因此,对基因漂流事件发生概率也采用了反馈控制技术,其具体描述如下:

$$p_f^{t+1}=\begin{cases}p_f^t+k_1p_f, f(\vec{\boldsymbol{X}}_{\max}^{t+1})>f(\vec{\boldsymbol{X}}_{\max}^{t})\\ p_f^t-k_2p_f, f(\vec{\boldsymbol{X}}_{\max}^{t+1})<f(\vec{\boldsymbol{X}}_{\max}^{t})\end{cases}, p_f\subseteq[0,1] \tag{9}$$

式中,p_f为基因漂流概率,k_1为概率递增系数,k_2为概率递减系数.

2.5 复杂系统遗传算法的流程

复杂系统遗传算法流程描述如下:

1. 设置算法参数.
2. 群体初始化.
3. 群体排序并统计,获取群体中最优适应度值.
4. 判断是否满足停止条件,如果满足,则跳转到步骤 5;否则,执行步骤 4.1.
   - 4.1. 采用贝努利实验以概率 p_f判定基因漂流事件是否成立,如果成立,则执行步骤 4.1.1;否则,跳转至步骤 4.2.
     - 4.1.1. 对 t 代、规模为 N 的种群 $\vec{\boldsymbol{X}}^t$ 中所有个体依次按步骤 4.1.1.1 操作,如果所有个体均操作完毕则跳转至步骤 4.2.
       - 4.1.1.1. 随机选择个体某一参数优势基因位,并以该基因位替换所有个体基因中其等基因位.
     - 4.1.2. 以公式(8)更新基因漂流概率 p_f.
   - 4.2. 对群体中所有个体依次按以下步骤进化,若进化完毕,则跳转至步骤 4.3.
     - 4.2.1. 采用上文设计的幂律选择算子获取交配父代个体号 n,而交配的母代为依次进化的个体.
     - 4.2.2. 采用贝努利实验判断以概率为 p_c 交叉事件为真,如果是,则执行步骤 4.2.2.1;否则,直接将父母作为子代个体 $\boldsymbol{X}^f$,并跳转到步骤 4.2.3.
       - 4.2.2.1. 将父母按照上文描述的环境-基因交叉算子杂交获得一个子代 $\boldsymbol{X}_u^f$,并按公式(5)修改基因变量组.
     - 4.2.3. 对子代个体 $\boldsymbol{X}^f$ 基因中的每个基因位采用贝努利实验以概率为 p_s 判断变异事件是否发生,如果是,则该基因位变异;否则,直接遗传.
     - 4.2.4. 按照本文设计的更新策略更新群体获得新种群 $\vec{\boldsymbol{X}}^{t+1}$.
   - 4.3. 按照公式(7)调节更新规模变量 p_r.
   - 4.4. 排序并获取群体中最优适应度值,并跳转至步骤 4.
5. 输出结果,结束程序.

3 数值实验及结果

为了检验本文所设计的复杂系统遗传算法的性能,选择了 14 个高维全局优化测试函数[14],并设计了 5 组实验.第 1 组实验是为了对比 CSGA 与现有高效遗传算法的搜索效率和搜索能力.而第 2 组实验则是在上述选取的 14 个高维全局优化测试函数中选取了 9 个作了偏移改造,进一步验证了 CSGA 的高维空间搜索能力.在第 3 组实验中,对 9 个偏移改造后的测试函数从 10 维测试到 1 000 维,分析了搜索空间维数对 CSGA 性能的影响.第 4 组实验着重测试了幂律选择算子、环境-基因双演化交叉算子、自适应更新策略以及基因漂流算子对 CSGA 的影响.最后一组实验则对比了 CSGA 中环境-基因双演化交叉算子的固定参数与变化参数对算法性能的影响.

3.1 算法性能对比测试

根据文献[14],本文选择了常用的 14 个非约束的测试函数,其维数、可行解空间范围以及函数可见 http://www.74tiger.com.cn/Code/14 个非约束的测试函数.

复杂系统遗传算法参数为:群体规模 N=50,交叉概率 p_c=0.7,变异概率 p_m=0.01,环境变量学习参数 r_1=*random*(0.0,1.2),环境变量遗忘参数 r_2=*random*(0.6,0.7),环境变量修改参数 r_3=*random*(0.0,1.4),α=0.1,β=0.01,

k_1=0.2,k_2=0.1.采用连续 50 代不能提高群体的最优解时停止为算法终止条件.算法对各个测试函数进行 50 次独立优化实验,对实验结果进行了统计,统计参数包括平均调用测试函数次数 FES_{avr}、最优适应度平均值 F_{avr}、最优适应度的极好值 F_b、最优适应度极差值 F_w 以及最优适应度的方差 F_α.复杂系统遗传算法的测试结果见表 1,与 OGA/Q 的对比结果见表 2.

由表 2 数据发现,除了 f_{12} 以外,本文所设计的 CSGA 搜索速度比 QGA/Q 快几十倍,虽然 f_{12} 的 CSGA 搜索速度要慢于 QGA/Q,但是前者最优适应度平均值–98.88.根据数据表 1,其最优适应度的极大值达到–99.36,与全局最小值–99.6 非常接近.而后者的最优适应度平均值仅仅为–92.83.对比所有的测试函数最优适应度平均值与方差,CSGA 算法基本上要好于 QGA/Q 算法,其中较为明显的除了 f_{12} 以外,还有测试函数 f_1(QGA/Q 的最优适应度平均最优值为–12569.45,而 CSGA 则为–12569.49)、测试函数 f_{10}(QGA/Q 的最优适应度平均最优值为–78.3000,而 CSGA 则为–73.33233).显然,本文设计的 CSGA 性能在各个方面都要强于 QGA/Q.

Table 1 Test results of CSGA

表 1 CSGA 算法测试结果

Function No.	Average function estimation times FES_{avr}	Best unit				Global minimal fitness
		Average fitness F_{avr}	Best fitness F_b	Worst fitness F_w	Variance of fitness F_α	
f_1	7 742	–12 569.49	–12 569.49	–12 569.49	6.00e–8	–12 569.5
f_2	6 773	0.0	0.0	0.0	0.0	0
f_3	12 271	5.00e–8	0.0	7.30e–7	1.20e–7	0
f_4	7 389.	0.0	0.0	0.0	0.0	0
f_5	13 738	6.00e–8	0.0	2.10e–7	5.00e–8	0
f_6	15 172	2.90e–7	0.0	2.76e–7	5.30e–8	0
f_7	7 367	0.0	0.0	0.0	0.0	0
f_8	6 206	1.76e–3	2.166e–5	5.399e–3	1.49e–3	0
f_9	7 733	2.20e–7	0.0	2.690e–5	7.30e–7	0
f_{10}	5 510	–78.332 33	–78.332 33	–78.332 32	0.0	–78.332 36
f_{11}	8 022	5.83e–5	0.0	3.175e–4	9.28e–5	0
f_{12}	957 144	–98.88	–99.36	–97.98	0.36	–99.6
f_{13}	5 912	0.0	0.0	0.0	0.0	0
f_{14}	5 824	0.0	0.0	0.0	0.0	0

Table 2 Test results: CSGA vs. OGA/Q

表 2 CSGA 与 OGA/Q 算法结果对比表

Function No.	Average function estimation times FES_{avr}		Average of best fitness F_{avr}		Variance of best fitness F_α	
	OGA/Q	CSGA	OGA/Q	CSGA	OGA/Q	CSGA
f_1	302 166	7 742	–12 569.45	–12 569.49	6.5e–4	6.000e–8
f_2	224 710	6 773	0.0	0.0	0.0	0.0
f_3	112 421	12 271	4.44e–16	5.00e–8	3.898e–17	1.20e–7
f_4	13 400	7 389	0.0	0.0	0.0	0.0
f_5	112 612	13 738	0.0	6.00e–8	0.0	5.00e–8
f_6	112 893	15 172	0.0	2.90e–7	0.0	5.30e–7
f_7	112 599	7 367	0.0	0.0	0.0	0.0
f_8	112 652	6 206	6.301e–3	1.76e–3	4.069e–4	1.49e–3
f_9	167 863	7 733	7.520e–1	2.20e–7	1.140e–1	7.30e–7
f_{10}	245 930	5 510	–78.300 0	–78.332 33	6.288e–3	0.0
f_{11}	112 893	8 022	0.0	5.83e–5	0.0	9.28e–5
f_{12}	302 773	957 144	–92.83	–98.88	2.626e–2	0.36
f_{13}	134 558	5 912	6.019e–6	0.0	6.619e–6	0.0
f_{14}	134 143	5 824	1.869e–4	0.0	2.615e–5	0.0

3.2 CSGA算法高维空间搜索性能测试

由于本文所设计的 CSGA 算法采用了基因漂流算子,该算子能够非常有效地处理测试函数中在解空间待优化参数相同的情况,其实际上是对待搜索函数进行了降维处理.在 14 个测试函数中,除了 f_{12} 以外,其最优解集各个维上的数值均相同,所以 CSGA 算法性能表现得很好.但这种情况并不能真实地反映 CSGA 算法在高维空间的搜索性能.因此,我们对测试函数 f_1~f_7,f_9,f_{10} 作了相应的改造,对函数自变量 $\boldsymbol{X}$ 增加了随机扰动生成新的函数

自变量 $\boldsymbol{Y}$,具体如下:

$$y_i=x_i+random(0,1) \tag{10}$$

通过上述改造后,f_1~f_7,f_9,f_{10} 函数的全局最小值不变,只改变了全局最小空间点的位置.例如:测试函数 f_9 全局最小值为 0,全局最小解集为 $[1]^m$;加入一个随机扰动后,f_9 全局最小值仍为 0,全局最小解集则变为 $[1+random(0,1)]^m$.此时,全局最小解集中每个解的值均不同,其他测试条件不变,测试结果见表 3.

测试数据表 3 与测试数据表 1 相比,CSGA 算法对改造前测试函数与改造后的测试函数在最优适应度上无论均值、极大值、极小值以及方差上变化均不大;但是在测试函数平均调用次数上,改造后的要明显大于改造前的.对比结果表明,虽然改造后的测试函数最优解集的每维都不再相同,但是 CSGA 算法在付出一定搜索代价后均可以发现比较理想的最优适应度.可见,本文所设计的 CSGA 算法具备较好的复杂高维空间搜索能力.

Table 3 Test results of rebuilding functions by using CSGA

表 3 CSGA 算法对改造后函数的测试结果

Function No.	Average function estimation times FES_{avr}	Best unit			
		Average fitness F_{avr}	Best fitness F_b	Worst fitness F_w	Variance of fitness F_α
f_1	25 892	−12 569.49	−12 569.49	−12 569.49	3.600e−7
f_2	3 520	0.0	0.0	0.0	0.0
f_3	47 592	2.270e−6	4.300e−7	6.300e−5	1.520e−6
f_4	24 558	4.500e−7	4.000e−8	1.670e−6	4.100e−7
f_5	49 074	1.960e−6	3.400e−7	7.450e−6	1.520e−6
f_6	74 049	1.819e−5	6.250e−6	3.642e−5	6.820e−6
f_7	27 400	4.700e−7	6.000e−8	2.650e−6	5.000e−7
f_9	1 064 598	1.003e−3	1.858e−5	3.631e−3	1.051e−3
f_{10}	71 744	−78.332 33	−78.332 33	−78.332 32	1.130e−6

3.3 搜索空间维数对CSGA性能影响的测试

为了进一步研究搜索空间维数对算法的影响,对改造后的 f_1~f_7,f_9,f_{10} 测试函数(即最优解集的每维值均不相同的情况)分别进行了 10 维~1000 维的测试.此时,算法停止条件为:在最优适应度精度满足小于等于 10^{-3} 的情况下,连续 50 代不能提高群体的最优解就终止算法;或者在最优适应度值精度不满足上述条件下,每维最多进化 3 000 代.测试数据见表 4(每个表格单元中的 3 行数据由上到下分别为:平均调用测试函数次数 FES_{avr}、最优适应度极差值 F_w 以及最优适应度的方差 F_α),测试结果见表 4.

由表 4 中的数据可以看出,CSGA 算法对 10 维~1000 维空间搜索都满足精度小于等于 10^{-3} 的条件,并且在搜索空间维数变化时,最优解的最差值与最优解的方差变化很小(测试函数 f_1 的最优解值与维数相关,$f_{\min}$=−418.9829*m).由此可见,CSGA 算法发现最优解的质量并不受搜索空间维数影响.但显然,随着搜索空间维数增加,CSGA 算法发现各个测试函数最优解的搜索代价也增加了.为了发现平均调用测试函数次数与搜索空间维数之间的变化规律,我们给出上述各个测试函数平均调用测试函数次数与搜索空间维数之间的变化关系图,如图 1 所示.

图 1 中对平均调用测试函数次数作了归一化处理(每个测试函数最大平均调用测试函数次数为 1).由图中我们发现,除了 f_6,f_9 以外,其他 7 个测试函数的平均调用测试函数次数与搜索空间维数之间的关系为线性规律,而 f_6 和 f_9 则更符合多项式规律.所以可以认为,在 CSGA 算法中,优化问题所花费的代价与待优化问题的复杂度相关,如果待优化问题在维度空间复杂度是线性的,那么 CSGA 所表现出的搜索代价也呈现出线性变化规律.

为了进一步对比本文所设计 CSGA 算法在高维空间的性能特性,我们引入 CEC2008-special session on large scale numerical optimization 测试函数集进行测试,并对比了 CEC2008 中 3 种较好的算法(DEwSAcc[15], EPUS-PSO[16],jDEdynNP-F[17])在 1 000 维时的测试结果,对比结果见表 5(注:CEC2008 测试函数的下载地址为 http://nical.ustc.edu.cn/conferences/cec2008/lsgo/LSGO.CEC08.Benchmark.zip; 测试方法与 CEC2008-special session on large scale numerical optimization 相同,代价函数调用次数 FES=5000×1000,每个问题独立运行 25 次).

Table 4 Relationship between the search space dimension and the algorithm capability

表 4 搜索空间维数与算法性能的关系

M	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_9	f_{10}
10	10 834 −4 189.8 4.0e−8	16 730 6.0e−7 2.0e−8	18 358 3.6e−7 1.0e−7	14 527 7.4e−3 1.0e−2	16 991 3.5e−7 1.0e−7	14 068 1.8e−6 6.7e−7	10 067 1.0e-7 4.0e−8	21 209 4.6e−8 1.4e−6	9 972 −78.332 1.0e−8
20	16 752 −8 379.7 2.1e−7	30 943 3.5e−7 1.0e−7	30 888 4.3e−6 1.1e−6	17 124 8.9e−7 2.8e−7	33 053 3.0e−6 6.9e−7	35 813 1.0e−5 2.0e−6	19 245 4.8e−7 1.2e−7	61 942 6. 8e−4 1.0e−4	16 976 −78.332 1.8e−7
30	24 278 −12 569.5 4.9e−7	51 848 1.0e−6 2.2e−7	45 243 4.3e−6 9.1e−7	23 632 1.2e−6 3.1e−7	46 918 5.2e−6 1.3e−6	61 697 4.0e−5 7.8e−6	26 682 1.0e−6 2.5e−7	113 735 6.4e−4 1.3e−4	25 064 −78.332 1.4e−7
50	37 581 −20 949.1 7.2e−7	84 602 3.2e−6 6.3e−7	70 921 1.8e−6 3.4e−6	36 698 3.3e−6 6.7e−7	75 165 1.19e−5 2.2e−6	125 979 9.8e−5 1.3e−5	42 537 4.4e−6 1.0e−6	284 773 9.2e−4 2.5e−4	38 774 −78.332 4.2e−7
100	75 500 −41 898.3 1.6e−6	166 753 7.7e−6 1.2e−6	134 042 2.5e−5 4.5e−6	64 473 5.7e−6 1.2e−6	147 957 3.8e−5 5.9e−6	334 056 3.9e−4 5.1e−6	82 657 8.6e−6 1.7e−6	1 075 565 1.0e−3 2.5e−4	71 991 −78.332 8.8e−7
200	169 298 −83 796.6 4.0e−7	426 095 1.73e−6 3.30e−7	315 136 7.1e−6 1.4e−6	144 004 3.7e−6 6.0e−7	359 034 2.1e−5 3.1e−6	2 569 392 4.6e−4 5.5e−5	188 874 2.2e−6 4.6e−7	4 109 027 1.9e−3 4.7e−4	152 619 −78.332 3.0e−7
300	251 344 −125 694.9 7.1e−7	612 130 3.9e−6 7.4e−7	442 441 1.3e−5 1.9e−6	205 549 4.3e−6 8.7e−7	525 960 2.4e−5 3.6e−6	4 982 998 8.3e−4 8. 1e−5	275 282 3.3e−6 5.9e−7	9 484 418 1.0e−3 3.8e−4	217 689 −78.332 5.4e−7
500	410 267 −209 491.4 1.2e−6	1 098 662 6.50e−6 1.25e−6	698 230 2.1e−5 3.0e−6	358 581 6.9e−6 1.1e−6	865 039 2.9e−5 5.3e−6	17 319 676 1.0e−3 3.6e−6	451 475 7.7e−6 1.3e−6	23 670 752 1.0e−3 3.8e−4	339 858 −78.332 7.0e−7
700	566 391 −293 288.0 2.0e−6	1 616 886 9.8e−6 1.7e−6	927 870 2.5e−5 4.0e−6	429 781 9.8e−6 1.8e−6	1 211 441 5. 4e−5 8.7e−6	45 968 095 1.0e−3 5.5e−7	620 531 8.3e−6 1.3e−6	44 803 267 1.0e−3 3.3e−4	456 861 −78.332 1.0e−6
800	646 821 −335 186.3 2.4e−6	1 874 201 1.2e−5 1.8e−6	1 042 006 3.0e−5 4.3e−6	486 798 1.2e−5 2.0e−6	1 373 488 4.0e−5 5.8e−6	66 090 828 1.0e−3 1.9e−7	706 721 1.6e−5 2.2e−6	57 391 206 1.0e−3 3.4e−4	517 607 −78.332 7.4e−7
1000	789 690 −418 982.9 2.4e−6	2 375 196 1.6e−5 1.5e−6	1 266 073 3.7e−5 5.6e−6	594 458 1.5e−5 2.3e−6	1 706 923 5.2e−5 9.1e−6	121 635 622 1.0e−3 6.1e−7	872 230 1.3e−5 2.1e−6	84 566 082 1.0e−3 2.1e−4	623 521 −78.332 1.4e−6

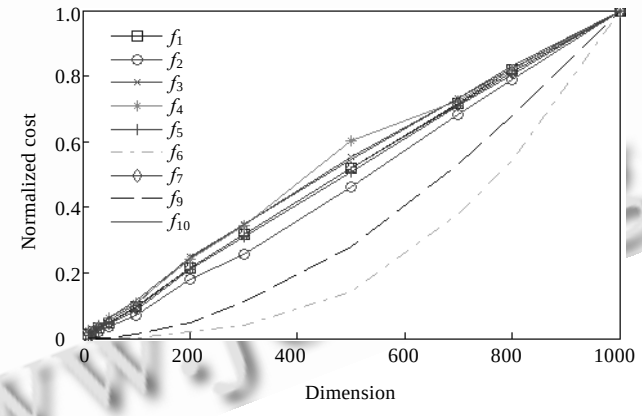

Fig.1 Relationship between the normalized cost of the average function estimation times and the dimension of test functions

图 1 平均调用次数泛化代价与测试函数维数的关系图

Table 5 Test results of CEC2008 function sets at 1 000 dimensions

表 5 CEC2008 测试函数集在 1 000 维时算法测试结果

Test function	Errors	DEwSAcc	EPUS-PSO	jDEdynNP-F	CSGA
$F^1_{CEC2008}$	Worst value	2.6490e–02	6.14e+02	1.1368e–13	1.0061e–11
	Best value	3.1715e–03	5.07e+02	1.1368e–13	8.0150e–12
	Mean value	8.7874e–03	5.53e+02	1.1368e–13	9.2745e–12
	Standard	5.2705e–03	2.86e+01	0.0	5.1390e–13
$F^2_{CEC2008}$	Worst value	9.9238e+01	4.73e+01	2.3161e+01	4.9723e+01
	Best value	9.2531e+01	4.59e+01	1.6031e+01	4.8942e+01
	Mean value	9.6058e+01	4.66e+01	1.9529e+01	4.9490e+01
	Standard	1.8194	4.00e–01	2.2525	1.8510e–01
$F^3_{CEC2008}$	Worst value	1.1629e+04	1.30e+06	1.5714e+03	1.2238e+03
	Best value	6.9294e+03	5.35e+05	1.1359e+03	6.2792e+02
	Mean value	9.1498e+03	8.37e+05	1.3136e+03	9.8128e+02
	Standard	1.2605e+03	1.52e+05	1.3635e+02	1.8564e+02
$F^4_{CEC2008}$	Worst value	2.0998e+03	8.01e+03	1.4753e–03	1.0687e–11
	Best value	1.5824e+03	7.27e+03	9.2726e–06	8.7540e–12
	Mean value	1.8239e+03	7.58e+03	2.1668e–04	9.5362e–12
	Standard	1.3773e+02	1.51e+02	4.0563e–04	4.3763e–13
$F^5_{CEC2008}$	Worst value	2.0272e–02	6.81e+00	5.6843e–14	6.3560e–01
	Best value	2.3156e–04	5.15e+00	2.8421e–14	4.2350e–12
	Mean value	3.5826e–03	5.89e+00	3.9790e–14	8.0500e–02
	Standard	5.7398e–03	3.91e–01	1.4211e–14	1.7330e–01
$F^6_{CEC2008}$	Worst value	2.7450	2.09e+01	1.1962e–10	1.8430e–01
	Best value	1.5605	1.33e+01	1.3073e–12	6.1005e–07
	Mean value	2.2956	1.89e+01	1.4687e–11	1.2500e–02
	Standard	2.9834e–01	2.49e+00	2.4310e–11	4.1500e–02
$F^7_{CEC2008}$	Worst value	–1.0268e+04	–6.58e+03	–1.3424e+04	–1.3651e+04
	Best value	–1.1508e+04	–6.72e+03	–1.3593e+04	–1.3888e+04
	Mean value	–1.0585e+04	–6.62e+03	–1.3491e+04	–1.3815e+04
	Standard	4.1846e+02	3.18e+01	4.6038e+01	5.59028e+01

通过测试结果对比表 5,CSGA 的效果要明显好于 DEwSAcc,只有测试函数 $F^5_{CEC2008}$ 的最差值要差于 DEwSAcc 算法;而 EPUS-PSO 算法只有测试函数 $F^5_{CEC2008}$ 要稍稍好于本文的 CSGA 算法,其他测试结果都明显差于 CSGA;CSGA 算法在测试函数集中 $F^3_{CEC2008}$,$F^4_{CEC2008}$,$F^7_{CEC2008}$ 要好于 jDEdynNP-F 算法,其中,$F^3_{CEC2008}$,$F^7_{CEC2008}$ 在测试函数集中属于搜索难度相对较高的测试函数.而对于其他测试函数,虽然 CSGA 不如 jDEdynNP-F 算法,但相差的绝对差值并不大,表明 CSGA 的搜索能力要强于 jDEdynNP-F 算法,但是在精度和稳定性上还有改进的空间.

3.4 CSGA算子影响分析

为了检验各个算子对 CSGA 算法的作用,我们设计了 3 组实验来分别检测幂律选择算子、环境-基因双演化交叉算子、自适应更新策略以及基因漂流算子对算法影响.

首先,将 CSGA 算法中的幂律选择算子换成普通的赌轮算子;其次,将 CSGA 算法中的环境-基因双演化交叉算子的环境变量集 $\boldsymbol{E}$ 去除,改用基于方向的交叉算子(direction-based crossover),算法其他部分保持不变.以测试函数 f_9 为测试对象,分别检测维数 m=10,20,30,50,100 时的计算结果,算法连续 50 代不能提高群体最优解时停止,其他测试条件同上,测试结果见表 6.

由数据表 6 可以看出:(1) 在采用赌轮选择算子替换 CSGA 幂律法则选择算子情况下,算法无法找到满意的适应度解;(2) 在采用方向交叉算子替换 CSGA 环境-基因双演化交叉算子情况下,在 10~50 低维时算法还能找到满意的适应度值,但是其测试函数平均调用次数是正常算法的 5 倍以上;而在 100 维时,使用方向交叉算子算法就无法每次搜索到满意的适应度值.可见,幂律法则选择算子保证了 CSGA 算法的收敛性,而环境-基因双演化交叉算子则提高了 CSGA 算法的效率以及稳定性.

Table 6 Effects analysis of the power law selector and the environment-gene coevolution crossover operator

表 6 幂律法则选择算子与环境-基因双演化交叉算子影响分析

m	Average function estimation times FES_{avr}			Average fitness of best units F_{avr}			Variance of fitness of best units F_{α}		
	Normal operator	Roulette wheel selection	Directional drossover	Normal operator	Roulette wheel selection	Directional drossover	Normal operator	Roulette wheel selection	Directional drossover
10	21 673	5 204	110 444	7.2e–6	109.3	1.1e–3	1.0e–5	2.8	7.9e–5
20	64 667	6 111	359 121	5.7e–5	166.6	3.3e–4	9.6e–5	3.1	2.2e–4
30	124 404	6 084	651 035	1.2e–4	228.6	6.2e–4	2.0e–4	0.8	3.3e–4
50	290 852	5 537	1 450 662	2.9e–4	486.3	1.4e–2	4.1e–4	0.8	6.8e–4
100	1 060 818	7 376	3 983 841	1.3e–3	1146.9	2.695	1.4e–4	0.7	13.799

再次,为了检测自适应更新策略对 CSGA 算法的影响,将本文设计的更新策略分别采用两种常用的群体更新方法取代:其一,子代直接取代父代个体策略;其二,只有适应度值大于父代的子代取代父代策略.依旧以测试函数 f_9 为测试对象,检测维数 m=30,其他测试条件不变,测试结果见表 7.

Table 7 Effects analysis of self-adaptive renew strategy

表 7 自适应更新策略影响分析

Test method	Average function estimation times FES_{avr}	Best unit			Variance of fitness F_{α}
		Average fitness F_{avr}	Best fitness F_b	Worst fitness F_{α}	
Self-Adaptive renew	124 404	1.159e–4	1.660e–6	1.072e–3	1.963e–4
Direct replacing	412 260	–26.991	25.704	27.950	0.552
Better fitness replacing	114 307	7.974e–2	9.000e–7	3.987	0.558

由表 7 可以看出:在采用子代直接取代父代的更新策略时,算法根本无法找到满意的适应度解;在完全采用适应度值大于父代的子代取代父代策略时,虽然测试函数平均调用次数会略小于自适应更新策略,但是此时不能保证算法每次都发现到满意的最优解,其最优适应度的最差值为 3.987.由此可见,本文设计的自适应更新策略保持了 CSGA 算法收敛性与收敛速度之间的平衡.

最后,为了检测基因漂流算子对 CSGA 算法的影响,仍然以测试函数 f_9 为测试对象,当维数 m=30 时,分别在全局最小解集在$[1]^m$ 和$[1+random(0,1)]^m$ 两种状态下,测试有基因漂流算子和无基因漂流算子两种情况,其他测试条件不变,测试结果见表 8.

Table 8 Effects analysis of gene floating

表 8 基因漂流算子影响分析

Test method		Average function estimation times FES_{avr}	Best Unit			Variance of fitness F_{α}
			Average fitness F_{avr}	Best fitness F_b	Worst fitness F_w	
Solution set at $[1]^m$	Use gene floating	6 383	0	0	0	0
	No gene floating	102 769	0.48	5.30e–7	3.987	1.295
Solution set at $[1+random(0,1)]^m$	Use gene floating	124 404	1.16e–4	1.66e–6	1.07e–3	1.963e–4
	No gene floating	100 299	1.12	7.80e–7	3.987	1.827

由表 8 中数据可知:当解集在$[1]^m$ 时,基因漂流算子不仅能够对相同值的解集空间进行降维处理,提高算法的收敛速度,而且保证了算法的收敛性,因为此时无基因漂流算子最优适应度的极差值为 3.987;当解集在$[1+random(0,1)]^m$ 时,基因漂流算子虽然多花费了测试函数平均调用次数的代价,但是保证了算法的收敛性.可见,基因漂流算子对 CSGA 算法的收敛性和搜索效率有显著影响.

3.5 CSGA重要参数影响分析

CSGA 算法除了一般遗传算法群体规模 N、交叉概率 p_c、变异概率 p_m 以外,还有环境变量学习参数 r_1、环境变量遗忘参数 r_2、环境变量修改参数 r_3 以及 4 个系数α,β,k_1,k_2.其中,群体规模 N、交叉概率 p_c、变异概率 p_m 对遗传算法的影响已有许多研究,而 CSGA 与一般遗传算法相似,在此不再赘述.而 4 个设定系数(α,β,k_1,k_2)只要

保证递增量大于递减量即可,对 CSGA 算法性能影响不大.而环境-基因双演化交叉算子中 3 个参数则对算法性能有很大的影响,在实际操作中发现,选择合适的 3 个固定参数值是一件十分困难的事情.因此,对 3 个参数设计成随机动态选取,从而减少了操作难度.为了对比环境-基因双演化交叉算子中固定参数与随机动态选取两种策略对算法性能的影响,以测试函数 f_9 为测试对象,维数 m=1000,全局最小解集在[1+$random$(0,1)]^m 状态下(全局最小解集在 3 次测试中不变),终止条件为 5 000 000 测试函数调用次数,分别测试了 3 种情况:固定值下 2 组(r_1=1.2,r_2=0.65,r_3=1.4 与 r_1=0.6,r_2=0.65,r_3=0.7)以及随机动态值 1 组(r_1,r_2,r_3 变化规律与上文中 CSGA 参数设置相同),测试数据见表 9.

Table 9 Effects analysis of environment variables

表 9 环境变量参数影响分析

Test method	Best unit			Variance of best fitness F_α
	Average fitness F_{avr}	Best fitness F_b	Worst fitness F_w	
r_1=1.2, r_2=0.65, r_3=1.4	987.54	986.11	989.61	3.4e−6
r_1=0.6, r_2=0.65, r_3=0.7	987.62	986.02	988.78	3.5e−6
Dynamical value r_1,r_2,r_3	821.76	821.30	822.54	2.7e−6

最好个体的最优适应度值与测试函数调用次数的变化关系如图 2 所示.

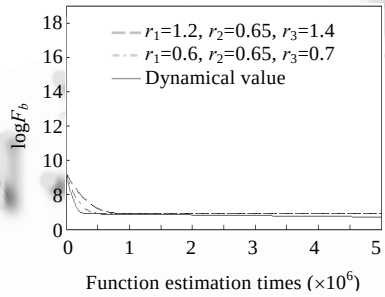

Fig.2 Relationship between the best unit optimal fitness and the function estimation times

图 2 最好个体的最优适应度值与函数调用次数的关系图

由数据表 9 可知,动态选取策略的最优适应度的平均值、最好值与最差值都要小于固定选取策略,更接近理论最优适应度值.由图 2 可以看出,动态选取策略的最优适应度下降速率也要好于固定选取策略,可见,环境-基因双演化交叉算子的 3 个参数随机动态选取策略要优于固定选取策略.

4 结束语

本文基于复杂系统的理念来重新考虑遗传算法的结构和参数之间的平衡关系,将复杂系统理论中自组织、自适应等特性应用到遗传算法的设计中.用合理反映系统能量分布的幂律法则来改造选择算子,并基于生物进化中环境与生物体的作用与反作用机制以及“趋同进化”理论设计了环境-基因的双演化交叉算子,还利用反馈机理自动调节更新策略,最后由生物体基因漂流现象的启发给遗传算法增加了基因漂流算子,上述的算子与机制相互平衡、相互制约,缺一不可.实验结果表明,本文所提出的复杂系统理论遗传算法在高维优化中具有较好的性能.

复杂系统遗传算法的程序和部分测试结果可以在 http://www.74tiger.com.cn//Program-ch.htm 下载.

致谢 在此,我们向对本文的工作给予建议的同行表示感谢.

References:

[1] Koumousis VK, Katsaras CP. A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance. IEEE Trans. on Evolutionary Computation, 2006,10(1):19−28. [doi: 10.1109/TEVC.2005.860765]

[2] Lee CY. Entropy-Boltzmann selection in the genetic algorithms. IEEE Trans. on Systems, Man, and Cybernetics—Part B: Cybernetics, 2003,33(1):138−142.

[3] Vitousek PM, Mooney HA, Lubchenco J, Melillo JM. Human domination of earth's ecosystems. Science, 1997,277(25):494−499.

[4] Kuo T, Hwang SY. A genetic algorithm with disruptive selection. IEEE Trans. on Systems, Man, and Cybernetics—Part B: Cybernetics, 1996,26(2):299−307.

[5] Meng W, Han XD, Hong BR. Bee evolutionary genetic algorithm. Chinese Journal of Electronics, 2006,34(7):1294−1300 (in Chinese with English abstract).

[6] Peng XB, Gui WH, Huang ZW, Hu ZK, Li YG. GAPSO: Effective genetic particle swarm algorithm and its application. Journal of System Simulation, 2008,20(18):5025−5031 (in Chinese with English abstract).

[7] Nowak MA. Five rules for the evolution of cooperation. Science, 2006,314(12):1560−1563. [doi: 10.1126/science.1133755]

[8] Albert R, Jeoung H, Barabasi AL. Diameter of the world wide Web. Nature, 1999,401(9):130−131. [doi: 10.1038/43601]

[9] Dauplais M, Lecoq A, Song JX, Cotton J, Jamin N, Gilquin B, Roumestand C, Vita C, de Medeiros CLC, Rowan EG, Harvey AL, Ménez A. On the convergent evolution of animal toxins—Conservation of a DIAD of functional residues in potassium channel-blocking toxins with unrelated structures. Journal of Biological Chemistry, 1997,272(7):4302−4309. [doi: 10.1074/jbc.272.7.4302]

[10] Ainsworth C. Cell biology: The story of *i*. Nature, 2006,440(6):730−733. [doi: 10.1038/440730a]

[11] Laland KN, Odling-Smee FJ, Feldman MW. Evolutionary consequences of niche construction and their implications foreecology. PNAS, 1999,96(18):10242−10247. [doi: 10.1073/pnas.96.18.10242]

[12] Gibbons A. Human anthropology: Modern men trace ancestry to African migrants. Science, 2001,292(11):1051−1052.

[13] Melby AE, Warga RM, Kimmel CB. Specification of cell fates at the dorsal margin of the zebrafish gastrula. Developmental Dynamics, 1997,122(7):2225−2237.

[14] Leung YW, Wang YP. An orthogonal genetic algorithm with quantization for global numerical optimization. IEEE Trans. on Evolutionary Computation, 2001,5(1):41−52. [doi: 10.1109/4235.910464]

[15] Zamuda A, Boškovc B, Zumer V. Large scale global optimization using differential evolution with self-adaptation and cooperative co-evolution. In: Tang K, ed. Proc. of the IEEE Congress on Evolutionary Computation (CEC 2008). Hong Kong: Springer-Verlag, 2008. 3718−3725.

[16] Hsieh ST, Sun TY, Liu CC, Tsai SJ. Solving large scale global optimization using improved particle swarm optimizer. In: Tang K, ed. Proc. of the IEEE Congress on Evolutionary Computation (CEC 2008). Hong Kong: Springer-Verlag, 2008. 1777−1784.

[17] Brest J, Zamuda A, Bõskovc B, Maucec MS, Zumer V. High-Dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction. In: Tang K, ed. Proc. of the IEEE Congress on Evolutionary Computation (CEC 2008). Hong Kong: Springer-Verlag, 2008. 2032−2039.

附中文参考文献:

[5] 孟伟,韩学东,洪炳镕.蜜蜂进化型遗传算法.电子学报,2006,34(7):1294−1300.

[6] 彭晓波,桂卫华,黄志武,李勇刚.GAPSO:一种高效的遗传粒子混合算法及其应用.系统仿真学报,2008,20(18):5025−5031.

庄健(1974－),男,江苏南通人,博士,副教授,主要研究领域为复杂系统理论,进化进算,故障诊断.

杜海峰(1972－),男,博士,教授,博士生导师,主要研究领域为进化计算.

杨清宇(1974－),男,博士,副教授,主要研究领域为复杂系统理论,进化进算,故障诊断.

于德弘(1949－),男,教授,博士生导师,主要研究领域为复杂系统理论,进化进算,成形技术.