软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(7):1760−1772 [doi: 10.3724/SP.J.1001.2012.04151]

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

自适应动态控制种群规模的自然计算方法*

王蓉芳[1+], 焦李成[1], 刘 芳[1,2], 杨淑媛[1]

[1](智能感知与图像理解教育部重点实验室(西安电子科技大学),陕西 西安 710071)

[2](西安电子科技大学 计算机学院,陕西 西安 710071)

Nature Computation with Self-Adaptive Dynamic Control Strategy of Population Size

WANG Rong-Fang[1+], JIAO Li-Cheng[1], LIU Fang[1,2], YANG Shu-Yuan[1]

[1](Key Laboratory of Intelligent Perception and Image Understanding of Ministry of Education (Xidian University), Xi'an 710071, China)

[2](School of Computer Science and Technology, Xidian University, Xi'an 710071, China)

+ Corresponding author: E-mail: rongfang.w@gmail.com

Wang RF, Jiao LC, Liu F, Yang SY. Nature computation with self-adaptive dynamic control strategy of population size. *Journal of Software*, 2012,23(7):1760−1772 (in Chinese). http://www.jos.org.cn/1000-9825/4151.htm

Abstract: A new self-adaptive dynamic control strategy of population size is proposed. This strategy can be easily combined with various nature computation methods because its implementation is independent of the evolutionary operation details. The framework of the strategy is first given. Based on the framework, the study proposes a method which can vary the number of increase/decrease on the basis of the logistic model. The study also designs an increase operator giving consideration to the effectiveness and diversity adaptively, as well as a decrease operator with the diversity. The strategy is applied to two different nature computation methods. Experimental evaluation is conducted on both a set of standard test functions and a new set of benchmark functions CEC05. The results show that the new algorithms with proposed strategy outperform the original algorithms on both the precision and convergence rate.

Key words: nature computation; population size; dynamic control; Logistic model; numerical optimization

摘 要: 提出了一种种群规模自适应动态控制策略,实现了种群规模根据进化过程自适应的动态变化.该策略的实现不依赖于算法进化操作的具体步骤,因而适用于各种基于种群优化的自然计算方法.首先给出了动态控制策略的框架;然后,在此框架下,充分利用动态种群规模反馈的有用信息,提出了基于 Logistic 模型的增加/删除数目自适应变化的方法,设计了自适应地兼顾有效性和多样性的增加算子和基于多样性的删除算子.将该策略应用到两种不同的自然计算方法中,采用经典测试函数和新型 CEC05 测试函数验证其性能.实验结果均表明,结合了所提出的种群规模自适应动态控制策略的新算法,比原算法在求解精度和收敛速度上均有明显的提升.

关键词: 自然计算;种群规模;动态控制;Logistic 模型;数值优化

中图法分类号: TP18 **文献标识码**: A

* 基金项目: 国家自然科学基金(60872135, 60803098, 61001202, 61003199); 陕西省"13115"科技创新工程重大科技专项(2008ZDKG-37); 高等学校学科创新引智计划(B07048); 国家教育部博士点基金(200807010003, 20090203120016, 20100203120008); 国家部委科技项目(9140A07011810DZ0107, 9140A07021010DZ0131); 中央高校基本科研业务费专项资金(K50510020001)

收稿时间: 2011-03-10; 定稿时间: 2011-11-02

自然计算(natural computation)通常是指那些受自然现象启发而发展起来的智能算法,如人工神经网络、进化计算、群智能、人工免疫系统、量子计算等[1].自然计算方法因其群体搜索性、本质并行性以及自学习、自组织等特点,解决了很多传统计算方法难以解决的非线性、不可微、多极值和高维复杂函数优化等问题,最近十几年来得到了蓬勃发展,在大规模复杂系统的优化设计、智能控制、计算机网络、无线通信等各个领域均得到了广泛的应用.

虽然各种自然计算方法模拟的生物机理不同,但是它们都有一个共同点,即都是基于种群的优化方法.在这类方法中,种群规模(population size)对算法的全局搜索能力和计算成本有很大的影响.较大的种群规模能够扩大搜索空间,提高找到全局最优解的概率,但是会增加运行时间,减缓收敛速度;而种群规模过小又会导致种群能力不足,无法搜索到更多有效区域.对于多峰问题,还容易陷入局部最优,导致早熟收敛,降低了算法的鲁棒性.所以,对于不同问题,如何选择一个恰当的种群规模来平衡算法的效率和有效性,成为自然计算方法应用的瓶颈问题.

针对该问题,已经出现了很多种群规模动态变化的方法,如变种群的遗传算法[2,3]、变种群的差分进化算法[4,5]以及变种群的粒子群算法[6−8]等.这些方法在一定程度上平衡了算法的效率和有效性,但种群规模的动态变化都是基于某种自然计算方法设计的,且其实现要依赖于该种算法的操作细节.

为了解决这个问题,本文提出了一种普适的种群规模自适应动态控制策略(self-adaptive dynamic control strategy of population size,简称 SaDCPS).该策略对种群规模的动态控制与算法采用何种具体操作无关,因而适用于各种基于种群优化的自然计算方法.其创新点主要体现在以下 4 个方面:

① 给出了动态控制策略的框架;

② 提出了基于 Logistic 模型的增加/删除数目自适应变化的方法;

③ 设计了自适应地兼顾有效性和多样性的增加算子;

④ 设计了基于多样性的删除算子.

并且,将该策略应用到两种不同的自然计算方法中,分别采用经典测试函数和新型 CEC05 测试函数这两组不同的数据集,验证了文中所提出的自适应动态控制策略的普适性和有效性.

1 种群规模自适应动态控制策略

种群规模动态变化总体来说需要解决 3 个问题,即种群规模何时变化、变化多少以及如何变化.

第 1 个问题是指用什么规则来触发增加/删除操作;

第 2 个问题是指增加/删除个体的数目依据什么来确定;

第 3 个问题则是增加/删除算子具体怎么实现.

下面详细介绍所提出的种群规模自适应动态控制策略对这 3 个问题各自的解决办法.

1.1 动态控制策略的框架

文献[7]中提出了一种种群管理(population manager,简称 PM)的方法来改进粒子群算法的性能.PM 的主要思想为:如果全局最优个体连续 K 代都更新,而且当前种群规模值 ps^g 大于 1,则认为现有个体对指导当前种群的搜索有冗余,因此删除一个性能较差的个体,以节约其进化时间来加速搜索过程;反之,如果全局最优个体连续 K 代都不更新,则认为现有个体有可能陷入了局部最优或信息不足,当 ps^g 小于上界时,增加一个新个体,当 ps^g 等于上界时,则删除一个性能较差的个体,为产生新的潜在个体保留空间.其中,K 是 PM 的激活阈值,是一个正常数.为了使 PM 有较高的灵敏度,K 的取值应该较小.可以看出,PM 依据最优个体是否更新这一标准来触发变化算子,简单、易行.而且 PM 加大了种群内部的竞争压力,即在 ps^g 小于上界时,也会删除个体,这样可以最大程度地减少种群的冗余度,节约有限的资源提供给那些更有能力的个体.

然而,PM 方法中存在以下 3 点不足:

首先,增加/删除个体的数目没有随着种群规模的变化而变化,始终为 1;

其次,在两种完全不同的删除情形下,却采用了相同的删除操作;

最后,增加/删除操作采用了相同的激活阈值 K.

因此,在我们的自适应动态控制策略中,只借鉴 PM 的触发规则,而针对上述分析的不足进行了改进,提出了如下动态控制策略框架:

(1) 如果最优个体连续 $2K$ 代都更新,且 $ps^g>PS_{\min}$,则执行删除算子 1,删除 n_{dec} 个个体;

(2) 如果最优个体连续 K 代都不更新,且 $ps^g<PS_{\max}$,则执行增加算子,增加 n_{inc} 个个体;否则,执行删除算子 2,删除 n_{dec} 个个体.

其中,$PS_{\max}$,$PS_{\min}$ 分别表示种群规模的上界和下界.我们将删除算子的激活阈值更改为增加算子激活阈值的两倍,这样既避免了新产生的个体在接下来的进化代里被马上删除,还可以防止种群内部竞争压力过大,而使得种群规模过小,无法探索其他区域.由该框架可以看出,我们的动态控制策略与算法采用何种具体的进化操作(比如交叉、变异、选择)无关,因此适用于各种基于种群优化的自然计算方法,具有很好的普适性.

图 1 是动态控制策略的流程图, fit^g_{best} 表示第 g 代最优个体的适应度值.在此框架下,增加/删除数目的方法不同,以及增加/删除算子设计的不同,就可以生成不同的动态控制策略.本文中,对于增加/删除个体的数目以及增加/删除算子都采用了自适应的方法,提出了自适应的动态控制策略,并且针对两种完全不同的删除情形设计了两种不同的删除算子.

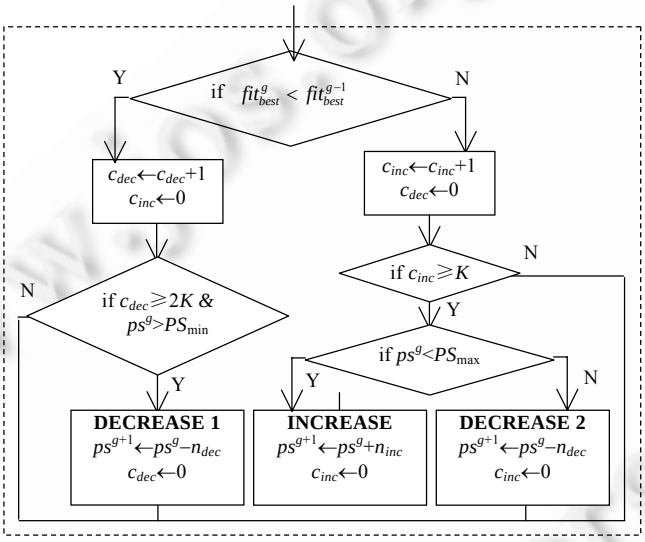


Fig.1 Flowchart of dynamic control strategy

图 1 动态控制策略流程

1.2 基于Logistic模型的增加/删除数目自适应变化的方法

增加/删除个体的数目,决定了种群规模 ps^g 的变化幅度.首先分析一下种群规模变化的内在规律:随着 ps^g 的增大,种群密度会逐渐增加,受有限资源的限制,增加个体的数目 n_{inc} 应该逐渐减小,而删除个体的数目 n_{dec} 应该逐渐增大.而且,当 ps^g 较小时,n_{inc} 应该大于 n_{dec},即种群规模的增长速率应大于收缩速率,以增强种群的全局探索能力(exploitation);相反地,当 ps^g 较大时,n_{dec} 应该大于 n_{inc},以增强深度探测能力(exploration).

基于以上分析,我们在 Logistic 人口模型的基础上设计了自适应的确定增加/删除数目的方法.下面首先简要介绍一下 Logistic 人口模型的主要内容.

Logistic 人口模型[9]是比利时数学家 Verhulst 在 1838 年修正了经典的 Malthus 人口模型后提出来的,他认为,人口的成长不可能超过其地域环境所决定的最大容量 M.曲线方程如下所示:

$$P'(t)=rP(t)=\lambda\left(1-\frac{P(t)}{M}\right)P(t),\lambda>0,0<P(0)<M \tag{1}$$

其中,

- $P(t)$和$P'(t)$分别表示时间 t 时的人口数和增加的人口数;
- r 是成长率;
- λ表示内在成长率(intrinsic growth rate).

$(1-P/M)$项的作用是为了当种群从初始大小 $P(0)$增长到上限 M 时,将成长率 r 从λ减小到 0.该方程的含义是,在人口相对较少时,人口的成长率与总人口数几乎成正比;但当人口相对较多时,人口成长率便会趋缓,而且越靠近人口上限 M 时,成长率越小.当 $P(t)=M$ 时,成长率为 0,人口进入平衡态.

可以看出,Verhulst 的 Logistic 人口模型与我们前面分析的增加个体数目的特点相一致,因此直接套用公式(1),给出了确定增加个体数目的公式(2);由于其模型在上限内,人口只会增加而不会减少,且 Logistic 模型只是以有上限为前提来引导出方程式,不会碰触到人口接近于某上限定值的问题.所以在公式(1)的基础上,结合删除个体数目变化的特点,设计了确定删除个体数目的公式(3).

$$n_{inc}=\left\lceil \lambda_{inc}\cdot\left(1-\frac{ps^g}{PS_{\max}}\right)\cdot ps^g \right\rceil \tag{2}$$

$$n_{dec}=\begin{cases}\left\lceil \lambda_{dec}\cdot\dfrac{ps^g}{PS_{\max}}\cdot(PS_{\max}-ps^g)\right\rceil, & \text{if } ps^g<PS_{\max}\\ random[1,\delta\times ps^g], & \text{if } ps^g=PS_{\max}\end{cases} \tag{3}$$

本文中,定义内在增加率$\lambda_{inc}=(PS_{\max}-ps^g)/PS_{\max}$,内在删除率$\lambda_{dec}=ps^g/PS_{\max}$,分别为当前可增加/删除个体的最大数目与种群规模上界的比值.

图 2、图 3 分别给出了 $PS_{\min}=1$,$PS_{\max}=100$ 时,根据公式(2)、公式(3)得到的增加/删除率,以及增加/删除数目随着种群规模动态变化的曲线.从图中可以直观地看出,基于 Logistic 模型的方法使得增加/删除个体的数目随着种群规模的变化自适应地动态变化,达到了本节开始时分析的预期效果.公式(3)还给出了当 ps^g 达到上界时 n_{dec} 的取值方法,本文中取$\delta=0.5$.

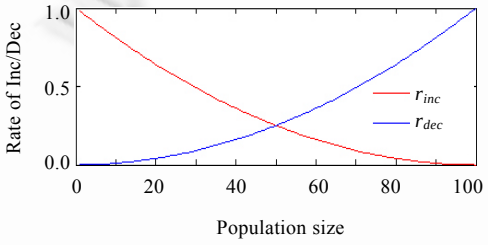


Fig.2 Curves of increase/decrease rate

图 2 增加/删除率曲线

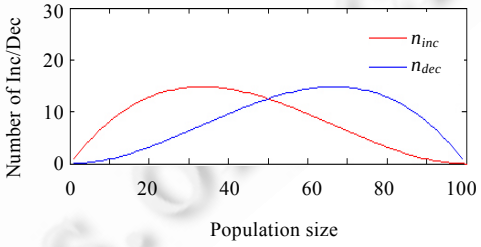


Fig.3 Curves of increase/decrease number

图 3 增加/删除数目曲线

1.3 增加/删除算子设计

种群规模动态变化不仅能够平衡算法的全局搜索能力和计算成本,而且若增加/删除算子设计得当,还能改善种群多样性,加快收敛速度,有效提高算法的性能.下面分别详细介绍本文方法中的增加/删除算子.

1.3.1 增加算子

设计增加算子时,应考虑有效性和多样性.有效性是指生成的新个体应该分享已有个体进化后的信息,使得新个体能够处在比较有利的位置,为种群的下一代进化贡献新的有用信息,加速搜索过程.但是生成新个体的方法又不能过于贪婪,因为仅利用最好解的信息会使得新个体容易陷入一个存在极值的区域,种群中的个体会不

断地向这一区域集中,出现很多相同或相似的个体,使种群的多样性变差,容易导致早熟收敛.因此,我们设计了自适应的兼顾有效性和多样性的增加算子.

首先生成一个大小为 n_{inc} 的父个体集 $\boldsymbol{S}$,然后从 $\boldsymbol{S}$ 中随机选择两个个体 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$,依据公式(4)[10]交叉生成新个体.其中,α是(0,1)之间的随机数.

$$\begin{aligned}\boldsymbol{x}_{new1} &= \alpha^{0.5}\cdot\boldsymbol{x}_1+(1-\alpha^{0.5})\cdot\boldsymbol{x}_2\\ \boldsymbol{x}_{new2} &= \alpha^{0.5}\cdot\boldsymbol{x}_2+(1-\alpha^{0.5})\cdot\boldsymbol{x}_1\end{aligned} \tag{4}$$

我们方法的不同之处,关键在于如何生成父个体集 $\boldsymbol{S}$,具体实现方法如下:

(1) 生成[1,n_{inc}]之间的一个随机数 n_1,将当前种群随机分成 n_1 个组,选择每个组里的最优个体,组成 S_1;

(2) $n_2\leftarrow n_{inc}-n_1$,在个体的初始边界[$XL_{init}$,$XU_{init}$]内,随机生成 n_2 个个体,组成 S_2;

(3) $\boldsymbol{S}\leftarrow\boldsymbol{S}_1\cup\boldsymbol{S}_2$.

可以看出,这样生成 $\boldsymbol{S}$ 后,交叉操作会产生 3 种不同的结果:

① 如果 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 都属于 $\boldsymbol{S}_1$,则新个体侧重有效性;

② 如果 $\boldsymbol{x}_1$ 和 $\boldsymbol{x}_2$ 都属于 $\boldsymbol{S}_2$,则新个体侧重多样性;

③ 如果 $\boldsymbol{x}_1$,$\boldsymbol{x}_2$ 分别属于 $\boldsymbol{S}_1$,$\boldsymbol{S}_2$,则新个体有可能位于还未探索的其他区域.

因此,即使在当前种群陷入局部最优、多样性丧失的情况下,增加算子也可以为种群带来新信息,很好地改善种群的多样性,从而提高了算法有效操作的效率和探索其他极值区域的能力.用分组最优而不是直接选择最优的 n_1 个个体组成 $\boldsymbol{S}_1$,也有利于抑制早熟收敛.

1.3.2 删除算子

根据第 1.1 节的触发规则,有如下两种不同情形的删除操作:

一种是种群连续 $2K$ 代都进化,说明已有个体对于指导当前种群进化可能太多了,为了节约时间,要删除掉冗余的个体.我们采用分组最差的删除方法,设计了删除算子 DECREASE 1.即将当前种群中最优个体以外的其余 ps^g-1 个个体,按照适应度从小到大排序,将整个序列随机分成 n_{dec} 个组,然后删除每个组里最差的个体.这样,可以使得删除的个体比较均匀地分布于种群的当前搜索空间上,删除冗余的同时,不会破坏种群的多样性.

另一种是种群连续 K 代没有进化,此时,种群有可能陷入了存在极值的区域,相似个体增多、多样性变差.本来应该增加个体,但由于种群规模达到上界,为了给产生的新个体保留空间,不得不删除个体.所以,执行采用贪婪方法的删除算子 DECREASE 2,删除当前种群中适应度值最差的 n_{dec} 个个体.

1.4 算法的实现策略

本节给出了算法的具体实现步骤.其中,算法 1 是自适应动态控制种群规模的自然计算方法,算法 2 和算法 3 分别是 INCREASE 算子和 DECREASE 1 算子.由于 DECREASE 2 算子实现较为简单,因此不再给出具体步骤.从算法 1 的描述中可以看出,如果去掉步骤 4~步骤 7,就是种群规模固定不变的自然计算方法.而在步骤 3,如果采用的进化操作方法不同,就可以形成基于不同自然计算方法的自适应动态控制种群规模的新算法.

算法 1. 自适应动态控制种群规模的自然计算方法.

Step 1:设置 $PS_{\min}$,$PS_{\max}$,K,令 $g\leftarrow0$,$ps^g\leftarrow PS_{\min}$,$c_{inc}\leftarrow0$,$c_{dec}\leftarrow0$.初始化种群 $\boldsymbol{X}^0\in[XL_{init},XU_{init}]$,计算初始种群所有个体的适应度值 $\boldsymbol{fit}^0$,令 $fit^0_{best}\leftarrow\min(\boldsymbol{fit}^0)$.

Step 2:若达到算法终止条件,则返回最优解,退出;否则,转步骤 3;

Step 3:对每个个体 $\boldsymbol{x}_i$ 进行一代进化操作(交叉、变异、选择);

Step 4:若 $fit^g_{best}<fit^{g-1}_{best}$,则 $c_{dec}\leftarrow c_{dec}+1$,$c_{inc}\leftarrow0$;否则,$c_{inc}\leftarrow c_{inc}+1$,$c_{dec}\leftarrow0$,转步骤 6;

Step 5:若 $c_{dec}\geqslant2K$ 且 $ps^g>PS_{\min}$,则根据公式(3)计算 n_{dec},执行 DECREASE 1 算子,$ps^{g+1}\leftarrow ps^g-n_{dec}$,$c_{dec}\leftarrow0$;否则,转步骤 8.

Step 6:若 $c_{inc}<K$,则转步骤 8.

Step 7:若 $ps^g<PS_{\max}$,则根据公式(2)计算 n_{inc},执行 INCREASE 算子,$ps^{g+1}\leftarrow ps^g+n_{inc}$,$c_{inc}\leftarrow0$;

否则,根据公式(3)计算 n_{dec},删除最差的 n_{dec} 个个体,$ps^{g+1}\leftarrow ps^g-n_{dec}$,$c_{inc}\leftarrow 0$.

Step 8:$g\leftarrow g+1$,转步骤 2.

算法 2. INCREASE 算子.

Step 1:$n_1\leftarrow random[1,n_{inc}]$,将当前种群随机分成 n_1 个组,选择每个组里适应度值最优的个体,组成 S_1;

Step 2:$n_2\leftarrow n_{inc}-n_1$,在个体的初始边界$[XL_{init},XU_{init}]$内,随机生成 n_2 个个体,组成 S_2;

Step 3:$\boldsymbol{S}\leftarrow \boldsymbol{S}_1\cup \boldsymbol{S}_2$;

Step 4:$i\leftarrow 1$,若 $i\leqslant n_{inc}$,则从 $\boldsymbol{S}$ 里随机选择两个个体,按照公式(4)生成两个新个体,$i\leftarrow i+2$,转步骤 4;

Step 5:计算新个体集里前 n_{inc} 个个体的适应度值;

Step 6:返回 n_{inc} 个新个体及其适应度值.

算法 3. DECREASE1 算子.

Step 1:保留当前种群的最优个体;

Step 2:将其余的 ps^g-1 个个体,按照适应度值从小到大排序;

Step 3:将整个序列随机分成 n_{dec} 个组,把每个组里适应度值最差个体的序列号记录到 ***index*** 中;

Step 4:删除 n_{dec} 个个体及其适应度值:$\boldsymbol{X}^g(\boldsymbol{index},:)\leftarrow[\cdot]$,$\boldsymbol{fit}^g(\boldsymbol{index})\leftarrow[\cdot]$;

Step 5:返回 $\boldsymbol{X}^g,\boldsymbol{fit}^g$.

1.5 算法时间复杂性分析

自适应动态控制种群规模的自然计算方法,每迭代一次所需时间为 $T_{EO}+T_{SaDCPS}$,其中,T_{EO} 是当前种群中所有个体进行一代进化操作的时间,T_{SaDCPS} 是执行 SaDCPS 的时间.由于 T_{EO} 的计算必须依据算法采用的交叉、变异、选择等进化操作的具体步骤,所以我们主要分析文中所提 SaDCPS 的时间复杂性.

从图 1 中可以清楚地看出,每一代执行 SaDCPS,共有 4 种可能的情况:一种是没有达到触发条件,不执行任何变化算子,此时所需时间为 $T_{NON}=0$;其余 3 种情况分别是执行 DECREASE 1,或者 INCREASE,或者 DECREASE 2.根据上文中对这 3 个算子步骤的详细描述和符号 O 的运算规则,推导得出各自所需的时间分别为 $T_{DEC1}=O(PS_{\max}-1)$,$T_{INC}=O(\lceil 4/27\cdot PS_{\max}\rceil)$和 $T_{DEC2}=O(0.05\cdot PS_{\max})$.再根据符号 O 执行条件语句的运算规则

$$T_{SaDCPS}=T_C+\max(T_{NON},T_{DEC1},T_{INC},T_{DEC2}) \tag{5}$$

其中,T_C 是计算触发标准所需时间,共有 3 次比较操作和 2 次赋值操作,因此 $T_C=O(1)$.将前面所有的计算结果代入公式(5),得到 $T_{SaDCPS}=O(1)+O(PS_{\max}-1)$.化简后,得到种群规模自适应动态控制策略每迭代一次的时间复杂度最差为 $O(PS_{\max})$.

2 仿真实验与结果分析

为了验证 SaDCPS 的有效性和普适性,我们将 SaDCPS 应用到两种不同的自然计算方法中:一种是标准差分进化算法(DE/rand/1)[11],另一种是标准粒子群优化算法(PSO)[12].相应得到了两种新算法:自适应动态控制种群规模的差分进化算法(SaDCPS+DE)和自适应动态控制种群规模的粒子群优化算法(SaDCPS+PSO).采用两组数据集来验证 SaDCPS 对不同算法的改进效果:一组是经典测试函数 f_1~f_{10}[13],另一组是近来很常用的新型测试函数 CEC05 f_{CEC1}~f_{CEC10}[14],两组测试函数的维数都取 30 维.

2.1 参数设置

DE 中涉及到的参数有 3 个:种群规模、变异尺度因子 F 和交叉率 CR.依据文献[11]中的设置,令 $F=0.5$,$CR=0.9$.PSO 中涉及的参数有 4 个:种群规模、惯性权重 W、加速常数 C_1 和 C_2.依据文献[12]中的设置,令 $W=0.4$,$C_1=C_2=2$.在我们的 SaDCPS 方法中,涉及的参数也有 3 个:种群规模的下界 $PS_{\min}$、上界 $PS_{\max}$ 和激活阈值 K.按照前面的分析,令 $K=2.PS_{\min}$ 理论上可以取 1,但在 DE 算法中,种群规模至少为 4 算法才能生效,因此本文中令 $PS_{\min}=4$. DE,PSO 的种群规模以及 SaDCPS 中的 $PS_{\max}$ 都设置为 100.所有算法均采用适应度值最大计算次数为终止条件,函数 f_1~f_{10} 的最大适应度计算次数均为 1.5×10^5,函数 f_{CEC1}~f_{CEC10} 的最大适应度计算次数均为 3×10^5.

在优化两组不同的测试函数以及不同算法的进化过程中,均没有任何参数再需调整.

2.2 SaDCPS策略中不同改进点的实验结果与分析

前文中提到,种群规模自适应动态控制策略主要有 3 个改进点,为了验证不同方面对 SaDCPS 的改进效果,我们将原来的 SaDCPS 作一些改变,得到如下 3 种不同的动态种群规模策略(dynamic population size,简称 DPS):

(1) 增加/删除算子不变,增加/删除数目固定为 1,简称为 DPS$^{(1)}$;

(2) 增加/删除数目的方法和删除算子不变,增加算子变为从初始边界内随机生成,简称为 DPS$^{(2)}$;

(3) 增加/删除数目的方法和增加算子不变,删除算子变为删除适应度值最差的个体,简称为 DPS$^{(3)}$.

将 SaDCPS 和这 3 种改动后的策略都与标准 DE 算法相结合,优化经典测试函数 f_2 和 f_9,以验证 SaDCPS 中不同改进点的性能.表 1 中给出了 4 种算法对函数 f_2 和 f_9 独立运行 25 次的平均结果,粗体字表示最好结果.图 4 是 f_2 和 f_9 的收敛比较图,横轴是适应度值计算次数,纵轴是平均适应度值.

Table 1 Averaged results of 30D functions f_2 and f_9

表 1 优化 30 维函数 f_2 和 f_9 的平均结果

	Algorithm	Mean	Std	1st (Min)	13th (Med)	25th (Max)
f_2	SaDCPS+DE	3.96E−12	2.22E−12	9.70E−13	3.65E−12	9.17E−12
	DPS$^{(1)}$+DE	7.01E−02	2.47E−01	3.08E−11	2.03E−10	1.04E+00
	DPS$^{(2)}$+DE	4.08E−08	2.06E−08	1.45E−08	3.62E−08	1.02E−07
	DPS$^{(3)}$+DE	**1.37E−12**	**1.31E−12**	**7.99E−14**	**9.31E−13**	**5.55E−12**
f_9	SaDCPS+DE	**9.47E+00**	**3.02E+00**	**3.98E+00**	**8.95E+00**	**1.79E+01**
	DPS$^{(1)}$+DE	2.52E+01	8.90E+00	1.29E+01	2.39E+01	4.31E+01
	DPS$^{(2)}$+DE	1.76E+02	1.50E+01	1.37E+02	1.79E+02	2.02E+02
	DPS$^{(3)}$+DE	1.18E+01	3.44E+00	6.08E+00	1.18E+01	2.06E+01

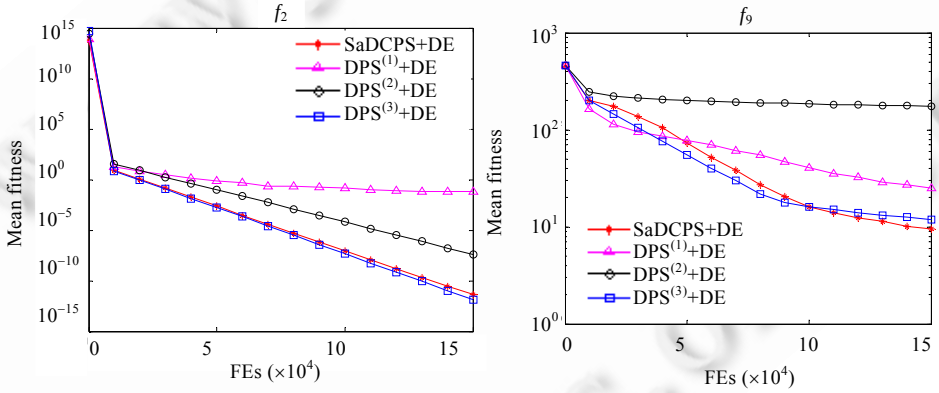


Fig.4 Convergence graph for 30D functions f_2 and f_9

图 4 优化 30 维函数 f_2 和 f_9 的收敛图

综合分析表 1 和图 4 中的结果可以看出:对于连续单峰函数 f_2,算法 DPS$^{(1)}$+DE 的结果最差,说明增加/删除数目的方法对 SaDCPS 的影响最大,其次是增加算子,影响最小的是删除算子.且采用贪婪方法删除最差个体的删除算子,结果还略优于基于多样性的删除算子.这也正符合单峰函数只有一个全局最小值,不会陷入局部极值的特点;而对于非常容易陷入局部极值的多峰函数 f_9 来说,改变增加算子后的算法 DPS$^{(2)}$+DE 的结果最差,收敛曲线也最平缓,说明结合了有效性和多样性的增加算子对 SaDCPS 的提升效果最显著,其次是增加/删除数目的方法,同样,影响最小的是删除算子.但有所不同的是,对于多峰函数而言,基于多样性的删除算子的性能要优于采用贪婪方法的删除算子,因为能否有效改善种群多样性对多峰函数的优化结果影响更大.

2.3 经典测试函数的实验结果与分析

首先,用 10 个经典的测试函数 f_1~f_{10} 来验证 SaDCPS 策略的整体性能.这 10 个函数都是最小化问题,全局最优值都为 f^*=0.其中,f_1~f_4 是连续单峰函数;f_5 在 D=2 和 D=3 时是单峰函数,但在高维有若干极小值 3.18;f_6 是不连续阶梯函数;f_7 是噪声四次函数.f_6~f_{10} 都是多峰函数,而且随着问题维数的增加,局部极小值的数量呈指数级增加.其中,f_5,f_8 和 f_9 非常容易陷入局部极值.这些函数的详细介绍可参考文献[13].每种算法均独立运行 25 次.表 2 给出了标准 DE 算法以及 SaDCPS+DE 的平均结果.表 3 给出了标准 PSO 算法以及 SaDCPS+PSO 的平均结果.

Table 2 Averaged results of DE and SaDCPS+DE for 30D functions f_1~ f_{10}

表 2 DE 和 SaDCPS+DE 优化 30 维函数 f_1~f_{10} 的平均结果

Algo. / Func.	DE					SaDCPS+DE				
	Mean	Std	1st (Min)	13th (Med)	25th (Max)	Mean	Std	1st (Min)	13th (Med)	25th (Max)
f_1	8.98E−14	7.94E−14	1.29E−14	8.89E−14	4.05E−13	**8.14E−23**	**1.42E−22**	**7.83E−25**	**3.64E−23**	**5.39E−22**
f_2	4.89E−07	2.01E−07	1.78E−07	4.65E−07	9.45E−07	**3.96E−12**	**2.22E−12**	**9.70E−13**	**3.65E−12**	**9.17E−12**
f_3	1.38E+00	8.16E−01	3.68E−01	1.21E+00	4.46E+00	**4.30E−03**	**5.78E−03**	**1.61E−04**	**2.34E−03**	**2.71E−02**
f_4	1.91E−01	4.65E−01	1.90E−03	1.39E−02	2.03E+00	**1.74E−01**	**1.70E−01**	**8.95E−04**	**1.27E−01**	**6.11E−01**
f_5	1.65E+01	9.03E−01	1.32E+01	1.67E+01	**1.75E+01**	**1.46E+01**	2.13E+00	**8.92E+00**	**1.46E+01**	1.96E+01
f_6	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
f_7	1.05E−02	3.03E−03	6.12E−03	9.52E−03	1.83E−02	**8.82E−03**	**2.73E−03**	**3.73E−03**	**7.65E−03**	**1.38E−02**
f_8	7.19E+03	3.70E+02	6.22E+03	7.28E+03	7.73E+03	**8.21E+02**	**3.45E+02**	**3.56E+02**	**8.09E+02**	**2.04E+03**
f_9	1.77E+02	1.25E+01	1.46E+02	1.80E+02	1.94E+02	**9.47E+00**	**3.02E+00**	**3.98E+00**	**8.95E+00**	**1.79E+01**
f_{10}	8.33E−08	2.72E−08	4.11E−08	8.60E−08	1.32E−07	**2.71E−12**	**2.50E−12**	**6.83E−13**	**1.73E−12**	**9.80E−12**

Table 3 Averaged results of PSO and SaDCPS+PSO for 30D functions f_1~ f_{10}

表 3 PSO 和 SaDCPS+PSO 优化 30 维函数 f_1~f_{10} 的平均结果

Algo. / Func.	PSO					SaDCPS+PSO				
	Mean	Std	1st(Min)	13th(Med)	25th(Max)	Mean	Std	1st(Min)	13th(Med)	25th(Max)
f_1	3.03E−36	7.88E−36	3.23E−40	1.82E−37	2.97E−35	**6.60E−43**	**2.07E−42**	**4.32E−50**	**5.56E−45**	**9.39E−42**
f_2	2.80E−23	5.49E−23	6.44E−25	9.15E−24	2.32E−22	**4.09E−27**	**1.28E−26**	**4.95E−30**	**1.25E−27**	**6.45E−26**
f_3	3.51E+00	1.69E+00	9.92E−01	3.16E+00	6.86E+00	**8.83E−02**	**8.55E−02**	**1.60E−02**	**6.19E−02**	**3.43E−01**
f_4	2.56E−01	1.39E−01	5.35E−02	2.42E−01	6.00E−01	**4.90E−02**	**3.87E−02**	**1.06E−02**	**3.77E−02**	**1.69E−01**
f_5	4.23E+01	3.02E+01	4.96E+00	2.30E+01	8.12E+01	**2.40E+01**	**2.02E+01**	**9.97E−01**	**1.92E+01**	**7.97E+01**
f_6	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
f_7	8.13E−03	3.19E−03	2.93E−03	7.79E−03	1.61E−02	**5.67E−03**	**1.68E−03**	**3.37E−03**	**5.41E−03**	**9.88E−03**
f_8	2.77E+03	4.70E+02	1.70E+03	2.92E+03	3.49E+03	**2.21E+03**	**4.46E+02**	**1.28E+03**	**2.17E+03**	**2.96E+03**
f_9	3.34E+01	1.02E+01	1.79E+01	3.08E+01	6.57E+01	**1.52E+01**	**3.83E+00**	**6.96E+00**	**1.49E+01**	**2.29E+01**
f_{10}	1.38E−14	2.49E−15	7.99E−15	1.51E−14	1.51E−14	**8.14E−15**	**2.40E−15**	**4.44E−15**	**7.99E−15**	**1.51E−14**

从表 2 的结果中可以看出,DE 和 SaDCPS+DE 对 f_6 都找到了全局最优解;其余 9 个函数在相同的适应度计算次数条件下,SaDCPS+DE 求得的解的精度都要优于原算法 DE.尤其对于函数 f_1~f_3 和 f_8~f_{10},精度有显著的提高.

表 3 的结果中,PSO 和 SaDCPS+PSO 也对 f_6 找到了全局最优解;其余 9 个函数在相同的适应度计算次数条件下,SaDCPS+PSO 求得的解的精度都要优于原算法 PSO.尤其对于单峰函数 f_1~f_4,精度提高得很明显;对多峰函数 f_5 和 f_7~f_{10},有略微提高.

为了更直观地比较 SaDCPS 对算法多样性和收敛速度的改进,在图 5 中画出了 10 个函数的收敛比较图.从图中可以清楚地看到,SaDCPS+DE 和 SaDCPS+PSO 在相同适应度计算次数时,解的精度都比原算法要高,所以收敛速度比原算法要快.尤其对于函数 f_8 和 f_9,原算法的收敛曲线比较平缓,说明陷入了局部最优;但是改进后的算法由于加入了能够改善种群多样性的增加/删除算子,使得算法可以跳出局部最优,收敛曲线明显呈下降趋势.

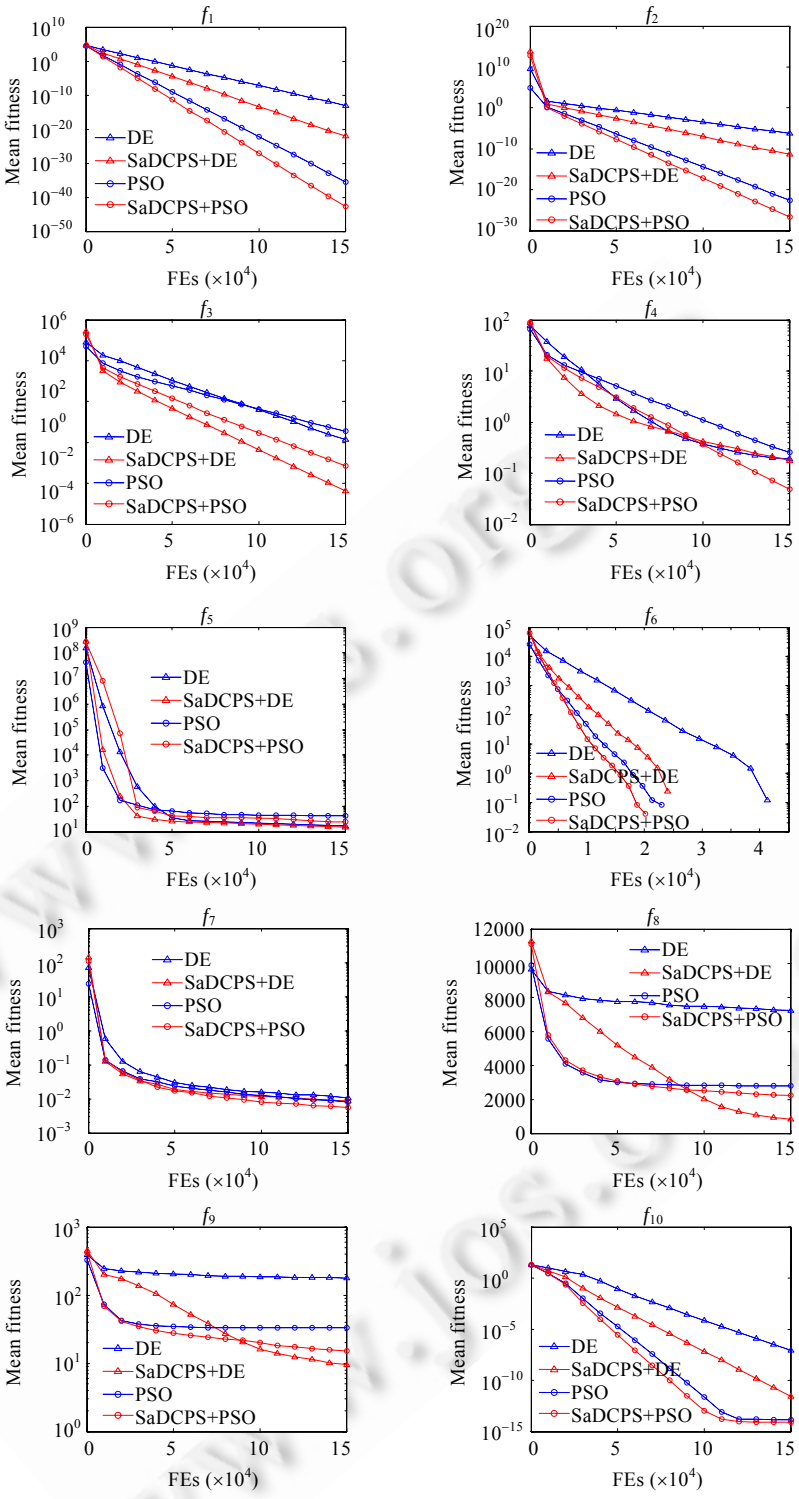


Fig.5 Convergence graph for 30D functions f_1~ f_{10}

图 5 优化 30 维函数 f_1~f_{10} 的收敛图

2.4 CEC05测试函数的实验结果与分析

为进一步验证 SaDCPS 的性能,我们还选择了 10 个 CEC05 函数 f_{CEC1}~f_{CEC10} 作为测试函数,这些函数的复杂度更高.其中,f_{CEC1}~f_{CEC5} 是单峰函数,f_{CEC6}~f_{CEC10} 是多峰函数.对这些函数的详细介绍可参考文献[14].每种算法均独立运行 25 次.CEC05 函数,通常用误差值(当前适应度值与全局最优值之间的差值)来比较算法的性能.表 4 给出了 DE 和 SaDCPS+DE 的平均误差比较结果.表 5 给出了 PSO 以及 SaDCPS+PSO 的平均误差比较结果.

Table 4 Averaged results of DE and SaDCPS+DE for 30D functions f_{CEC1}~f_{CEC10}

表 4 DE 和 SaDCPS+DE 优化 30 维函数 f_{CEC1}~f_{CEC10} 的平均结果

Algo. / Func.	DE					SaDCPS+DE				
	Error	Std	1st (Min)	13th (Med)	25th (Max)	Error	Std	1st (Min)	13th (Med)	25th (Max)
f_{CEC1}	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
f_{CEC2}	1.34E−04	1.43E−04	2.34E−05	7.10E−05	6.26E−04	**5.14E−09**	**6.08E−09**	**9.84E−11**	**2.96E−09**	**2.52E−08**
f_{CEC3}	3.97E+05	1.91E+05	**8.09E+04**	3.84E+05	7.08E+05	**2.28E+05**	**1.03E+05**	1.14E+05	**1.91E+05**	**4.59E+05**
f_{CEC4}	5.32E−02	7.84E−02	4.16E−03	3.23E−02	4.01E−01	**2.07E−04**	**4.55E-04**	**4.12E−06**	**6.66E−05**	**2.02E−03**
f_{CEC5}	5.60E+01	2.99E+01	1.14E+01	5.65E+01	1.28E+02	**4.35E+01**	**2.69E+01**	**9.53E+00**	**3.84E+01**	**1.04E+02**
f_{CEC6}	1.72E+00	1.10E+00	2.63E−03	1.52E+00	**4.45E+00**	**1.04E+00**	1.38E+00	**1.73E−07**	**4.71E−01**	5.55E+00
f_{CEC7}	4.70E+03	1.15E−02	4.70E+03	4.70E+03	4.70E+03	4.70E+03	**5.30E−06**	4.70E+03	4.70E+03	4.70E+03
f_{CEC8}	2.09E+01	5.58E−02	2.08E+01	2.09E+01	2.10E+01	**2.08E+01**	3.13E−01	**2.00E+01**	2.09E+01	2.10E+01
f_{CEC9}	1.41E+02	2.36E+01	9.56E+01	1.43E+02	1.77E+02	**6.15E+00**	**2.85E+00**	**2.98E+00**	**5.01E+00**	**1.33E+01**
f_{CEC10}	1.84E+02	9.20E+00	1.71E+02	1.84E+02	2.01E+02	**3.71E+01**	**1.37E+01**	**1.99E+01**	**3.48E+01**	**6.77E+01**

从表 4 的结果中可以看出,DE 和 SaDCPS+DE 对 f_{CEC1} 和 f_{CEC7} 平均误差精度相同;其余 8 个函数在相同的适应度计算次数条件下,SaDCPS+DE 求得的平均误差的精度都要高于原算法 DE.尤其对于函数 f_{CEC2},f_{CEC4},f_{CEC9} 和 f_{CEC10},精度有显著提高.

从表 5 的结果中可以看出,只有函数 f_{CEC7} 和 f_{CEC10},SaDCPS+PSO 的平均误差略大于 PSO,但对于这两个函数,SaDCPS+PSO 所能找到的最小误差要优于 PSO;其余 8 个函数在相同的适应度计算次数条件下,SaDCPS+PSO 求得的解的精度都要优于原算法 PSO.尤其对于函数 f_{CEC2} 和 f_{CEC8},精度有显著提高.

Table 5 Averaged results of PSO and SaDCPS+PSO for 30D functions f_{CEC1}~f_{CEC10}

表 5 PSO 和 SaDCPS+PSO 优化 30 维函数 f_{CEC1}~f_{CEC10} 的平均结果

Algo. / Func.	PSO					SaDCPS+PSO				
	Error	Std	1st (Min)	13th (Med)	25th (Max)	Error	Std	1st (Min)	13th (Med)	25th (Max)
f_{CEC1}	5.91E−14	1.14E−14	5.68E−14	5.68E−14	1.14E−13	**3.87E−14**	2.71E−14	**0.00E+00**	**5.68E−14**	**5.68E−14**
f_{CEC2}	1.57E−01	3.46E−01	2.16E−03	3.74E−02	1.44E+00	**3.24E−05**	**5.30E−05**	**1.50E−06**	**1.28E−05**	**2.61E−04**
f_{CEC3}	9.81E+06	9.22E+06	1.21E+06	4.08E+06	**2.95E+07**	**4.92E+06**	**8.60E+06**	**4.15E+05**	**1.09E+06**	3.64E+07
f_{CEC4}	1.07E+03	1.09E+03	8.78E+01	8.18E+02	5.13E+03	**5.52E+02**	**9.90E+02**	**5.03E+01**	**2.12E+02**	**5.04E+03**
f_{CEC5}	7.46E+03	1.69E+03	4.87E+03	7.28E+03	1.25E+04	**6.81E+03**	**1.22E+03**	**4.05E+03**	**6.83E+03**	**8.70E+03**
f_{CEC6}	3.14E+01	3.77E+01	1.41E+00	2.04E+01	1.59E+02	**2.38E+01**	**4.03E+01**	**1.52E−02**	**5.73E+00**	**1.28E+02**
f_{CEC7}	**6.21E+03**	**1.08E+02**	5.99E+03	**6.20E+03**	**6.42E+03**	6.25E+03	2.23E+02	**5.85E+03**	6.23E+03	6.56E+03
f_{CEC8}	2.09E+01	4.89E−02	2.08E+01	2.09E+01	2.10E+01	**2.01E+01**	8.19E-02	**2.00E+01**	**2.01E+01**	**2.04E+01**
f_{CEC9}	3.74E+01	9.55E+00	2.29E+01	3.68E+01	5.77E+01	**1.69E+01**	**5.69E+00**	**7.96E+00**	**1.49E+01**	**2.89E+01**
f_{CEC10}	**9.26E+01**	**2.58E+01**	5.87E+01	**8.66E+01**	1.87E+02	1.08E+02	3.35E+01	**4.97E+01**	1.03E+02	**1.80E+02**

图 6 是 10 个函数 f_{CEC1}~f_{CEC10} 的收敛比较图.横轴是适应度计算次数,纵轴是平均误差.从图中可以清楚地看出,SaDCPS+DE 只有对 f_{CEC7} 的收敛速度和 DE 几乎相当,f_{CEC5} 和 f_{CEC6} 的收敛速度略快于 DE,其余 7 个函数的收敛速度与 DE 相比均有显著的提高;对于 SaDCPS+PSO,f_{CEC1} 和 f_{CEC7} 的收敛速度和 PSO 几乎相当,f_{CEC4}~f_{CEC6} 的收敛速度略快于 PSO,f_{CEC10} 的收敛速度略慢于 PSO,其余 4 个函数的收敛速度与 PSO 相比均有显著的提高.而且对于 f_{CEC8}~f_{CEC10},改进后的两种算法都可以跳出局部最优,收敛曲线呈明显的下降趋势.

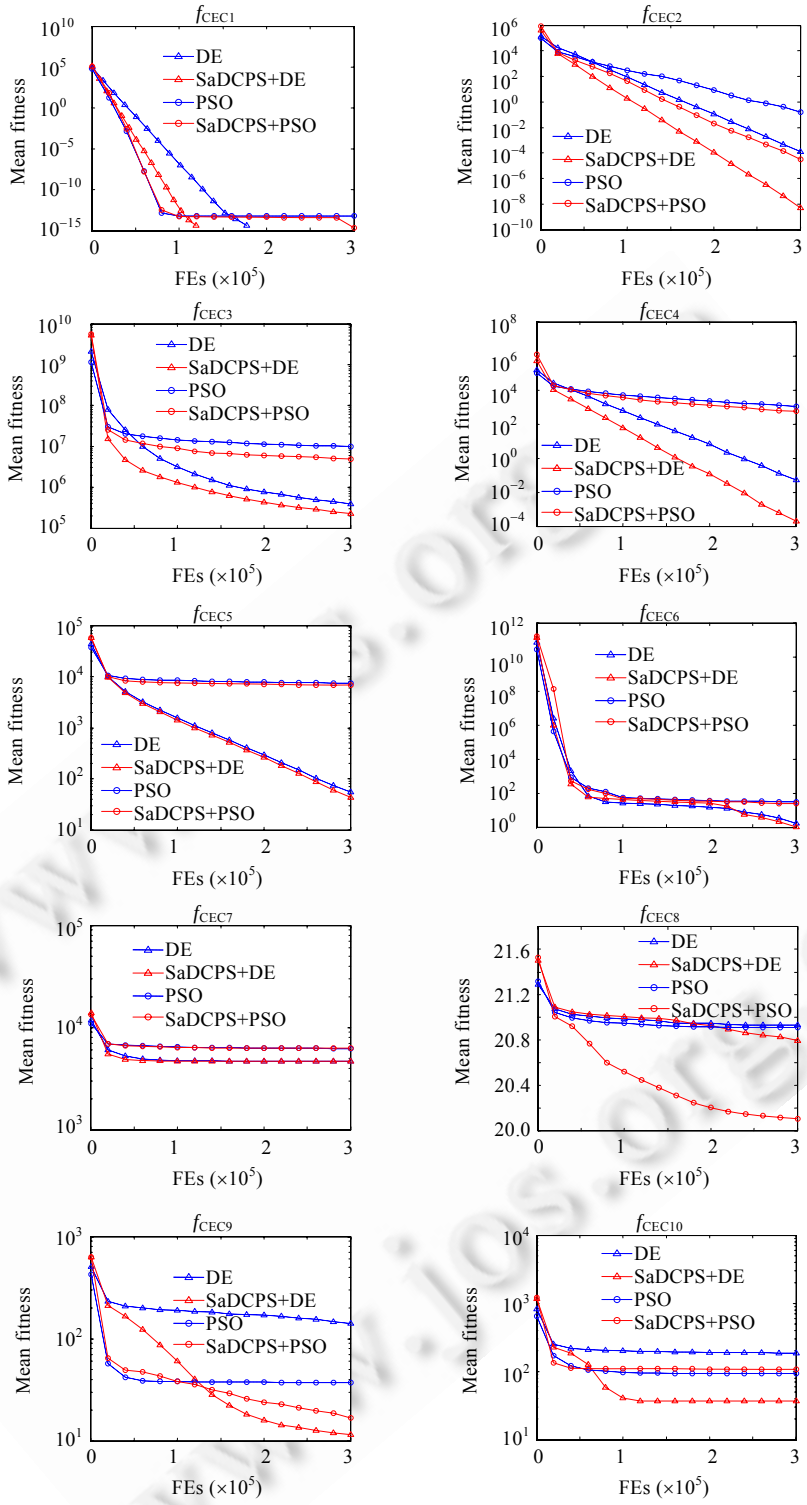


Fig.6 Convergence graph for 30D functions f_{CEC1}~f_{CEC10}

图 6 优化 30 维函数 f_{CEC1}~f_{CEC10} 的收敛图

3 结束语

传统的自然计算方法中,种群规模一般都是固定不变的.事实上,文献[15]中指出,在进化过程中采用始终不变的静态参数,有可能无法得到最好的结果.以往对自然计算方法的各种改进版本中,对交叉率、变异率这些参数的自适应变化研究得比较多,但是对于另一个关键参数——种群规模,还是与传统方法一样,根据经验取固定值.因此,本文提出了一种自适应动态控制种群规模的策略,实现了种群规模在进化过程中自适应地变化.该方法不依赖于进化操作的细节,采用最优个体是否连续更新来触发变化算子,因此可以很容易地与各种自然计算方法相结合,提升原方法的效率与性能.在 DE 和 PSO 上的实验测试结果表明,基于 SaDCPS 的 DE 算法,对两组测试数据均比原算法在求解精度和收敛速度上有所提高;基于 SaDCPS 的 PSO 算法,对经典函数在求解精度和收敛速度上均有所提高;对 CEC05,大部分函数有所提高.而且,兼顾了多样性和有效性的增加算子以及基于多样性的删除算子的使用,还有助于算法跳出局部最优区域,从而验证了 SaDCPS 的普适性和有效性.其中,优化经典函数时,改进的效果更显著.因为标准的 DE 和标准的 PSO 算法在进化过程中涉及的参数都是取固定值,在遇到更复杂的优化问题时,比如 CEC05 测试函数,其本身的性能就不是很好.所以,SaDCPS 对算法的提升程度与原算法的性能有关系.当原算法的性能较优时,结合了 SaDCPS 后,改进效果也会更显著.因此,后续工作尝试将 SaDCPS 应用到种群规模固定但其他参数自适应变化的自然计算方法中,以求解更为复杂的高维优化问题和多目标优化问题.

References:

[1] Castro LN. Fundamentals of natural computing: An overview. Physics of Life Reviews, 2007,4(1):1−36. [doi: 10.1016/j.plrev.2006.10.002]

[2] Arabas J, Michalewicz Z, Mulawka J. GAVaPS—A genetic algorithm with varying population Size. In: Fogel D, ed. Proc. of the 1st IEEE Conf. on Evolutionary Computation. Orlando: IEEE Press, 1994. 73−78. [doi: 10.1109/ICEC.1994.350039]

[3] Shi XH, Wan LM, Lee HP, Yang XW, Wang LM, Liang YC. An improved genetic algorithm with variable population size and a PSO-GA based hybrid evolutionary algorithm. In: Pan ZS, Chen SC, eds. Proc. of the 2nd Int'l Conf. on Machine Learning and Cybernetics. Xi'an: IEEE Press, 2003. 1735−1740. [doi: 10.1109/ICMLC.2003.1259777]

[4] Teo J. Exploring dynamic self-adaptive populations in differential evolution. Soft computing—A fusion of foundations. Methodologies and Applications, 2006,10(8):673−686. [doi: 10.1007/s00500-005-0537-1]

[5] Brest J, Zamuda A, Boskovic B, Maucec MS, Zumer V. High-Dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction. In: Suganthan PN, ed. Proc. of the IEEE Congress on Evolutionary Computation. Hong Kong: IEEE Press, 2008. 2032−2039. [doi: 10.1109/CEC.2008.4631067]

[6] Soudan B, Saad M. An evolutionary dynamic population size PSO implementation. In: Rifai S, Shalati S, eds. Proc. of the 3rd Int'l Conf. on Information and Communication Technologies: From Theory to Applications. Damascus: ICTTA, 2008. 1−5. [doi: 10.1109/ICTTA.2008.4530016]

[7] Hsieh ST, Sun TY, Liu CC, Tsai SJ. Solving large scale global optimization using improved particle swarm optimizer. In: Suganthan PN, ed. Proc. of the IEEE Congress on Evolutionary Computation. Hong Kong: IEEE Press, 2008. 1777−1784. [doi: 10.1109/CEC.2008.4631030]

[8] Leong WF, Yen GG. PSO-Based multiobjective optimization with dynamic population size and adaptive local archives. IEEE Trans. on Systems, Man, and Cybernetics—Part B: Cybernetics, 2008,38(5):1270−1293. [doi: 10.1109/TSMCB.2008.925757]

[9] Tsoularis A, Wallace J. Analysis of logistic growth models. Mathematical Biosciences, 2002,179(1):21−55. [doi: 10.1016/S0025-5564(02)00096-2]

[10] Arumugam MS, Rao MVC. On the improved performances of the particle swarm optimization algorithms with adaptive parameters, cross-over operators and root mean square (RMS) variants for computing optimal control of a class of hybrid systems. Applied Soft Computing, 2008,8(1):324−336. [doi: 10.1016/j.asoc.2007.01.010]

[11] Storn R, Price K. Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization, 1997,11(4):341−359. [doi: 10.1023/A:1008202821328]

[12] Kennedy J, Eberhart R. Particle swarm optimization. In: Proc. of the IEEE Int'l Conf. on Neural Networks. Piscataway: IEEE Service Center, 1995. 1942−1948. http://ieeexplore.ieee.org/search/searchresult.jsp?matchBoolean%3Dtrue%26searchField%3DSearch_All%26queryText%3D%28p_Title%3AParticle+swarm+optimization%29&addRange=1995_1995_Publication_Year&pageNumber=1&resultAction=REFINE

[13] Yao X, Liu Y. Evolutionary programming made faster. IEEE Trans. on Evolutionary Computation, 1999,3(2):82−102. [doi: 10.1109/4235.771163]

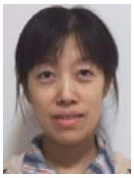
[14] Suganthan PN, Hansen N, Liang JJ, Deb K, Chen YP, Auger A, Tiwari S. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical Report, Report No.2005005, Singapore: Nanyang Technological University, 2005.

[15] Eiben AE, Hinterding R, Michalewicz Z. Parameter control in evolutionary algorithms. IEEE Trans. on Evolutionary Computation, 1999,3(2):124−141. [doi: 10.1109/4235.771166]

王蓉芳(1978－),女,甘肃康县人,博士生,主要研究领域为自然计算,智能信号处理,压缩感知.

刘芳(1963－),女,教授,博士生导师,CCF 高级会员,主要研究领域为人工智能,信号和图像处理,模式识别.

焦李成(1959－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为信号和图像处理,机器学习,计算智能,模式识别.



杨淑媛(1978－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为智能信号处理,多尺度几何分析.