一种改进的自适应逃逸微粒群算法及实验分析*

赫 然[1+], 王永吉[1,2], 王 青[1], 周津慧[1], 胡陈勇[1]

[1](中国科学院 软件研究所 互联网软件技术实验室,北京 100080)

[2](中国科学院 软件研究所 计算机科学重点实验室,北京 100080)

An Improved Particle Swarm Optimization Based on Self-Adaptive Escape Velocity

HE Ran[1+], WANG Yong-Ji[1,2], WANG Qing[1], ZHOU Jin-Hui[1], HU Chen-Yong[1]

[1](Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

[2](Laboratory of Computer Science, Institute of Software, The Chinese Academy of Sciences, Beijing 100080, China)

+ Corresponding author: Phn: +86-10-62632367 ext 805, E-mail: heran@itechs.iscas.ac.cn, http://itechs.iscas.ac.cn

Received 2004-10-25; Accepted 2005-04-10

He R, Wang YJ, Wang Q, Zhou JH, Hu CY. An improved particle swarm optimization based on self-adaptive escape velocity. *Journal of Software*, 2005,16(12):2036–2044. DOI: 10.1360/jos162036

Abstract: To deal with the problem of premature convergence and slow search speed, this paper proposes a novel particle swarm optimization (PSO) called self-adaptive escape PSO, which is guaranteed to converge to the global optimization solution with probability one. Considering that the organisms have the phenomena of escaping from the original cradle when they find the survival density is too high to live, this paper uses a special mutation –escape operator to make particles explore the search space more efficiently. The novel strategy produces a large speed value dynamically according to the variation of the speed, which makes the algorithm explore the local and global minima thoroughly at the same time. Experimental simulations show that the proposed method can not only significantly speed up the convergence, but also effectively solve the premature convergence problem .

Key words: particle swarm optimization; escape velocity; self-adaptive; mutation; swarm intelligence

摘 要: 分析了变异操作对微粒群算法(particle swarm optimization,简称 PSO)的影响,针对收敛速度慢、容易陷入局部极小等缺点,结合生物界中物种发现生存密度过大时会自动分家迁移的习性,给出了一种自适应逃逸微粒群算法,并证明了它依概率收敛到全局最优解.算法中的逃逸行为是一种简化的确定变异操作.当微粒飞行速度过小时,通过逃逸运动使微粒能够有效地进行全局和局部搜索,减弱了随机变异操作带来的不稳定性.典型复杂函数优化的

* Supported by the National Natural Science Foundation of China under Grant Nos.60373053, 60473060 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2004AA112080 (国家高技术研究发展计划(863)); the Hundred Talents of the Chinese Academy of Sciences (中国科学院"百人计划")

作者简介: 赫然(1979－),男,辽宁大连人,助理软件工程师,主要研究领域为演化计算,数据库系统;王永吉(1962－),男,博士,研究员,博士生导师,主要研究领域为人工智能,实时系统,网络优化;王青(1964－),女,博士,研究员,主要研究领域为软件过程技术与质量管理;周津慧(1957－),女,副研究员,主要研究领域为软件过程技术与质量管理;胡陈勇(1979－),男,助理软件工程师,主要研究领域为信息检索.

仿真结果表明,该算法不仅具有更快的收敛速度,而且能更有效地进行全局搜索.

关键词: 微粒群算法;逃逸速度;自适应;变异操作;群体智能
中图法分类号: TP18 文献标识码: A

微粒群算法(particle swarm optimization,简称 PSO)是由 Kennedy 和 Eberhart 等人[1]于 1995 年提出的一种基于种群搜索的自适应进化计算技术.算法最初受到飞鸟和鱼类集群活动的规律性启发,用组织社会行为代替了进化算法的自然选择机制,通过种群间个体协作来实现对问题最优解的搜索.PSO 算法随机产生一个初始种群并赋予每个微粒一个随机速度,在飞行过程中,微粒的速度通过自身以及同伴的飞行经验来进行动态调整,整个群体有飞向更好搜索区域的能力.由于 PSO 算法概念简单、实现容易、参数较少、能有效解决复杂优化任务[2],所以在过去几年中获得了飞速发展,并在图像处理、模式识别、多目标优化和游戏设计等很多领域得到广泛应用.

目前,PSO 算法作为一种崭新的随机搜索算法,仍旧存在着早熟收敛和收敛较慢这两个难题,并且具有种群多样性随代数增加下降过快、有可能不收敛到全局最优解等缺点.为了避免早熟收敛,一些研究者提出了通过控制种群多样性来提高算法性能的方法.一方面,通过解决微粒间的冲突和聚集[3]、分析微粒和种群最优位置的距离[4]、增加环境检测和响应、种群随机多代初始化[5]等思想,给出了不同策略来增强种群多样性,使算法不至于过早陷入局部极小.另一方面,通过引入遗传算法的“变异”操作来增强全局搜索性能也是一种十分有效的方式.文献[6]通过对微粒速度或位置引入了一个小概率随机变异操作来增强种群多样性(dissipative PSO,简称 DPSO),使算法能够有效地进行全局搜索,但是过大的变异率在增加种群多样性的同时也将导致群体发生混乱,使种群不能进行精确的局部搜索,延缓了算法的收敛速度.文献[7–9]深入分析了不同均值和方差的高斯变异操作对增强算法性能的影响.文献[10]提出了一种改善微粒多样性的途径——自组织临界点控制.算法对每个微粒增加了当前临界值属性,如果两个微粒间的距离小于预定阈值,则增加彼此的临界值,重新分配其在搜索空间中的位置,以达到控制种群多样性的目的.文献[11]在自组织算法的基础上给出了一种变异操作随时间变化的自适应层次 PSO 算法(self-organizing hierarchical PSO,简称 HPSO)以进一步提高搜索性能,并给出了适合变异操作的自适应参数选择方式,但是 HPSO 算法消除了速度公式中的惯性部分,其发生变异的条件是微粒速度为 0,使微粒不能快速、有效地逃出局部极小点.虽然这些研究工作已经给出了提高 PSO 算法全局搜索能力的方法,但是它们很难在提高搜索速度和保持种群多样性之间达到平衡.

本文分析了 PSO 算法种群多样性与微粒速度的关系,指出可以通过控制微粒速度的方式来协调算法的种群多样性.同时,我们注意到在现实生物界中,当物种生存密度过大时,群体有自动分家并找到新的生存空间的特性,结合当前改进 PSO 算法的特性,在变异操作基础上对 PSO 算法模型进行了改进,给出了一种自适应逃逸微粒群算法,并证明它能依概率收敛到全局最优解.新算法通过限制微粒在搜索空间内的飞行速度和速度的自适应策略,使微粒能够在大空间内广泛搜索、在局部区域内快速搜索,有效加快了收敛速度并增强了全局搜索能力.为了深入对比分析逃逸运动对 PSO 算法的影响,本文引入了 6 个著名的 Benchmark 函数问题(包括简单、复杂函数和单模态、多模态函数)进行数字实验,并与基本 PSO,HPSO,DPSO 算法进行了对比分析,结果显示,逃逸运动是一种比随机变异操作更有效、更适合 PSO 算法的操作.

1 微粒群算法

考虑全局优化问题(P)

$$\min\{f(x): x\in\Omega\subset R^n\},\ f:\Omega\subset R^n\to R^1,$$

则问题(P)的多个可行解的一个集合称为一个种群(swarm).种群中每个元素(可行解)称为一个微粒(particle).微粒的个数称为种群规模(size).我们用 n 维向量 $X_i=(x_{i1},x_{i2},...,x_{in})^T\in\Omega$ 来表示第 i 个微粒的位置,用 $V_i=(v_{i1},v_{i2},...,v_{in})^T\in\Omega$ 来表示第 i 个微粒的速度.微粒在搜索空间飞行过程中,它自身所经历的最佳位置记为 $P_i=(p_{i1},p_{i2},...,p_{in})^T$,也称为 pbest.在群体中,所有微粒经历过的最佳位置用索引符号 g 表示,即 P_g,也称为 gbest.

因此,微粒在每一代中的速度和计算评价函数的位置通过如下两个公式计算:

$$v_{id}(t+1)=\omega\times v_{id}(t)+c_1\times r_1\times(p_{id}(t)-x_{id}(t))+c_2\times r_2\times(p_{gd}(t)-x_{id}(t)) \quad (1)$$

$$x_{id}(t+1)=x_{id}(t)+v_{id}(t+1) \quad (2)$$

其中,w 为惯性权重(inertia weight),c_1 和 c_2 为加速常数(acceleration constants),r_1 和 r_2 为两个在[0,1]范围内服从均匀分布的随机变量.公式(1)中的第 1 部分称为动量部分,表示微粒对当前自身运动状态的信任,为微粒提供了一个必要动量,使其依据自身速度进行惯性运动.第 2 部分称为认知部分,代表了微粒自身的思考行为,鼓励微粒飞向自身曾经发现的最佳位置.第 3 部分称为社会部分,表示微粒间的信息共享与相互合作,它引导微粒飞向微粒群中的最佳位置.这 3 个部分之间的相互平衡和制约决定了算法的主要性能.

为了解决 PSO 算法早熟收敛,搜索能力随代数增加而逐渐下降的特点,DPSO 算法引入了随机变异操作,使微粒有更大几率逃出局部极小点,公式描述如下:

$$\text{If } (r_3<C_v) \text{ then } v_{id}=r_4\times V_{\max}/C_m \quad (3)$$

其中 r_3,r_4 为相互独立的均匀分布在[0,1]的随机变量,C_v 用来控制速度的变异率,C_m 是一个常量,用来控制变异幅度的大小.而 HPSO 在引入变异操作的同时,消除了速度的惯性部分,给出了新的参数自适应方法、速度公式和变异条件:

$$v_{id}(t+1)=c_1\times r_1\times(p_{id}(t)-x_{id}(t))+c_2\times r_2\times(p_{gd}(t)-x_{id}(t)) \quad (4)$$

$$\text{If } (v_{id}=0) \text{ then } v_{id}=r_6\times r_5\times v \quad (5)$$

其中 v 是重新初始化的速度;r_5 为均匀分布在[0,1]的随机变量;r_6 为随机变量,当随机数小于 0.5 时为 1,大于 0.5 时为−1.

2 自适应逃逸微粒群算法(AEPSO)

文献[12]的理论结果显示,PSO 算法的微粒在搜索空间中以不同正弦曲线波飞行.由式(1)可知,速度 v 将会逐渐变小,假如速度的社会部分和认知部分都变小,微粒的速度将不会再变大,并丧失了空间探索的能力.当社会部分和认知部分趋近于 0 时,由于 w 小于 1,速度将会迅速下降到 0.换句话说,当初始速度不为 0 时,微粒会通过惯性运动远离 gbest;假如微粒的速度逐渐接近 0 时,所有的微粒将会逐渐接近 gbest 并且停止运动.实际上,PSO 算法并不能保证收敛到全局最优点,而仅仅是收敛到种群最好点 gbest.由公式(1)我们还可以看出,速度的大小还代表了微粒相对于 gbest 的距离.当微粒离 gbest 远时,速度就大;当微粒离 gbest 近时,速度就会逐渐变小.所以,可以通过控制微粒速度的方式来避免微粒快速聚集到 gbest,从而达到有效控制种群多样性的目的.

对于 DPSO 算法的随机变异操作,我们发现在算法后期随机变异使微粒群不能进行精确的局部搜索,并且对于单模态 Benchmark 问题并不能有效提高算法的收敛速度.而 HPSO 算法在引入变异操作和新的参数策略的同时,消除了速度的惯性部分.它在微粒速度为 0 时给予微粒一个变异操作,使微粒能在全局范围内搜索,但是微粒速度的快速下降,使其不能在搜索空间内进行详细的局部搜索.所以,本文结合两者的优点给出了一种定向、定时的变异操作,即当速度小于一定阈值时,给予速度一个变异操作——逃逸运动,描述为

$$\text{If } (v_{id}<T_d) \text{ then } v_{id}=rand\times V_{\max} \quad (6)$$

其中,阈值 T_d 用来保存种群中第 d 维的当前阈值速度且其值恒大于零;$rand$ 为一个在[0,1]范围内服从均匀分布的随机变量.

在自然界生物群中,如蚂蚁、蜜蜂等,当物种生存密度过小时,生物有自动分家并寻找新集聚地繁衍后代的特性.对于 PSO 中的种群,当微粒发现物种生存密度过小时,将会通过逃逸运动寻找新的生存地,这时它们将会忘记自身的历史最好位置而仅记住种群的最佳位置.PSO 算法用如下公式描述微粒的这种行为:

$$\text{If } (v_{id}<T_d) \text{ then } p_{id}=x_{id} \quad (7)$$

其中的判定条件与式(6)的判定条件相对应,即当微粒发生逃逸时,对 pbest 初始化一个新的值,其大小为微粒发生逃逸后所产生的新位置.实验结果显示,式(7)的引入将会显著提高算法的收敛速度.

式(5)中阈值 threshold 的大小将会影响算法的执行效果,并且很难给出一个固定的阈值适合解决所有问题.

过大的阈值将会导致种群发生混乱,使算法不能进行有效的局部搜索;过小的阈值将会导致微粒速度需要一个相对长的时间下降到逃逸条件,使算法需要一个相对长的时间逃出局部极小,不能有效提高算法的搜索速度.所以我们假设微粒的各维速度之间相互独立,给予每维速度一个阈值,当有过多微粒速度达到这个值时,就自动使其下降,通过自动调整每一维速度的阈值来动态调整微粒速度.公式描述如下:

$$F_d(t)=F_d(t-1)+\sum_{i=1}^{Size}b_{id}(t) \tag{8}$$

其中 $b_{id}(t)=\begin{cases}0, & v_{id}(t)>T_d\\ 1, & v_{id}(t)<T_d\end{cases}$.

$$\text{If } F_d(t)>k_1 \text{ then } F_d(t)=0;\ T_d=T_d/k_2 \tag{9}$$

其中,频率 $F_d(t)$用来标记种群第 d 维速度曾经发生逃逸的次数;常数 k_1 用来标记调整阈值 T_d 和频率 $F_d(t)$的条件值;常数 k_2 用来控制阈值速度下降的幅度.算法通过限制微粒飞行速度的方式,有效地协调了全局搜索与局部搜索之间的关系,在加快算法收敛速度的同时,提高了算法的全局寻优能力.

3 收敛性分析

假设 **1**. (1) 问题(P)的可行域Ω为 R^n 中的有界闭区域.(2) 目标函数 $f(x)$是区域 Ω 上的连续函数.

定义 **1**. 设$\{X(k)\}$是由算法 M 产生的种群序列,其中 $x^*(k)\in X(k)$为第 k 代种群中的最优个体,若存在 $f(x^*(k))\le f(x^*(k-1))$,且序列中的某一点 $x^*(N)$本身或者序列有一个极限 x^*是问题 P 的一个极小点,则称算法 M 是局部收敛的.

引理 **1**. 若问题 P 存在局部极小点 x^*,则 $\lim_{t\to\infty}p_g(t)=x^*$.

证明:因为 $p_g(t)$为迭代过程中 t 时刻的微粒群最优个体,则有 $f(p_g(t))\le f(p_g(t-1))$成立.把式(1)代入式(2)中得到

$$x_{id}(t+1)=x_{id}(t)+\omega\times v_{id}(t)+c_1\times r_1\times(p_{id}(t)-x_{id}(t))+c_2\times r_2\times(p_{gd}(t)-x_{id}(t)),$$

由于 $v_{id}(t)=x_{id}(t)-x_{id}(t-1)$,则

$$x_{id}(t+1)=x_{id}(t)+\omega\times(x_{id}(t)-x_{id}(t-1))+c_1\times r_1\times(p_{id}(t)-x_{id}(t))+c_2\times r_2\times(p_{gd}(t)-x_{id}(t)),$$

对于种群最优个体有 $p_{gd}(t)=p_{id}(t)$,则式(6)变为

$$x_{id}(t+1)=x_{id}(t)+\omega\times(x_{id}(t)-x_{id}(t-1))+(c_1\times r_1+c_2\times r_2)\times(p_{gd}(t)-x_{id}(t)) \tag{10}$$

由文献[13,14],$x(t)$将会逐渐逼近种群最好点 $p_g(t)$,由式(5)可知,$|v_{id}|$恒大于 0.

假设 $p_g(t)$不是局部极小点,则存在 $r>0$,使得当$||x-p_g(t)||\le r$ 时,有 $f(x)<f(p_g(t))$.由于 $f(x)$连续,由式(10)在 $x(t)$向 $p_g(t)$接近过程中,必然$\exists t'<t$,使得$||x(t')-p_g(t)||\le r$,即有 $f(x(t'))<f(p_g(t))$成立.这与已知相矛盾,所以假设不成立.

所以 $p_g(t)$为局部极小点,命题成立. □

定理 **1**. AEPSO 算法是局部收敛的.

证明:对于种群最优微粒有 $p_g(t)=p_i(t)$,由于 $\lim_{t\to\infty}x(t)=p_g(t)$[13,14], $\lim_{t\to\infty}p_g(t)=x^*$($x^*$是局部极小点),并且存在序列 $f(p_g(t))\le f(p_g(t-1))\le\ldots\le f(p_g(0))$,由定义 1 可知,AEPSO 算法是局部收敛的.

引理 **2**. 设 $\Delta x_{id}=v_{id}$, $r_1,r_2\sim N(0,1)$,则 $\Delta x_{id}\sim N(\mu_d,\sigma_d)$.

证明:由式(1)可得,

$$\Delta x_{id}=\omega\times v_{id}+c_1\times(p_{id}-x_{id})\times r_1+c_2\times(p_{gd}-x_{id})\times r_2.$$

设

$$\phi_1=\omega\times v_{id},\phi_2=c_1\times(p_{id}-x_{id}),\phi_3=c_2\times(p_{gd}-x_{id}),$$

则

$$\Delta x_{id}=\phi_1+\phi_2\times r_1+\phi_3\times r_2.$$

对于 AEPSO,在粒子运行过程中,式(5)定期给粒子一个较大的速度冲量,使 ϕ_1,ϕ_2,ϕ_3 在搜索过程为非 0 变量.由于 $r_1,r_2\sim N(0,1)$,显然有 $\Delta x_{id}\sim N(\mu_d,\sigma_d)$ (在 PSO 算法中 r_1,r_2 取正数部分). □

定理 2. 设$\{X(k)\}$是由 AEPSO 算法产生的种群序列,其中 $x^*(k)\in X(K)$为第 k 代种群中的最优个体,即 $x^*(k)=\arg\min_{1\le i\le \mu} f(x_i(k))$,如果问题(P)中的目标函数和可行域满足假设 1,则有 $P\{\lim_{k\to\infty} f(x^*(k))=f^*\}=1$,即种群序列以概率为 1 收敛于全局最优解.

证明:由引理 2 可知,$\Delta x_{id}\sim N(\mu_d,\sigma_d)$,由式(2)可知,AEPSO 算法是一个仅带有变异操作的进化策略算法,由文献[15,16]可知,定理 2 成立(详细证明参见文献[15,16]). □

4 对比实验

4.1 参数设置及Benchmark函数问题

为了分析 AEPSO 算法的收敛速度、全局搜索性能和惯性权重对算法执行性能的影响,本文选择 PSO 及其改进算法进行对比实验,引入 6 个 Benchmark 优化问题来进行分析.所有实验的种群规模均为 20,函数维度设置为 30.每个函数独立运行 50 次,每次运行 6 000 代(函数评价次数为 1.2×10^5).表 1 显示算法的实验参数设计.对于基本 PSO 算法,传统理论分析显示,线性下降的惯性权重有利于种群搜索,所以,选择 w 在[0.95,0.4]之间随代数线性递减;对于 DPSO 和 HPSO 算法选用文献[6,11]中的默认参数,HPSO 的加速变量 c_1,c_2 均采用线性策略;在 AEPSO 算法中, AEPSO1 算法使用线性惯性权重,AEPSO2 算法使用固定惯性权重,以便分析速度惯性权重对逃逸运动的影响,对于表 2 中的函数 f_5,f_6,设置 K_1=5,对于其他函数 K_1=10.

Table 1 Parameter selection for different PSOs

表 1 微粒群算法参数设置

Algorithm	w	c_1	c_2	C_v	K_1	K_2	Swarm size
PSO	[0.95,0.4]	1.4	1.4	–	–	–	20
DPSO	0.7	1.4	1.4	0.001	–	–	20
HPSO	0	[2.5,0.5]	[0.5,2.5]	–	–	–	20
AEPSO1	[0.95,0.4]	1.4	1.4	–	5/10	10	20
AEPSO2	0.7	1.4	1.4	–	5/10	10	20

本文选择了 PSO 算法和遗传算法(genetic algorithm,简称 GA)经常使用的 6 个 Benchmark 函数问题进行数值实验,并根据函数性质分为具有单一极小点(单模态)和多个局部极小点(多模态)两大类,每类 3 个函数.表 2 显示了这些 Benchmark 函数的定义、取值范围和全局最优解.算法实验结果中的数值"0"表示 Matlab 精度值 1e−308(Matlab 当小于这个值时显示 0).

Table 2 Benchmark functions used in this study. f_1~f_3 are unimodal functions and f_4~f_6 are multimodal functions

表 2 Benchmark 函数.f_1~f_3 为单模态函数.f_4~f_6 为多模态函数

Function	Name	Dim.	Search space	Min/Best position
$f_1=10^6x_1^2+\sum_{i=2}^{n}x_i^2$	Tablet	30	$(-100,100)^N$	0 (0,…,0)
$f_2=\sum_{i=1}^{n}\left(\sum_{j=1}^{i}x_j\right)^2$	Quadric	30	$(-100,100)^N$	0 (0,…,0)
$f_3=\sum_{i=1}^{n}(100(x_{i+1}-x_i^2)+(x_i-1)^2)$	Rosenbrock	30	$(-50,50)^N$	0 (1,…,1)
$f_4=\frac{1}{4000}\sum_{i=1}^{n}(x_i)^2-\prod_{i=1}^{n}\cos\left((x_i)/\sqrt{i}\right)+1$	Griewank	30	$(-300, 300)^N$	0 (0,…,0)
$f_5=\sum_{i=1}^{n}(x_i^2-A\cos(2\pi x_i)+A)$, A=10	Rastrigrin	30	$(-5.12,5.12)^N$	0 (0,…,0)
$f_6=\sum_{i=1}^{n-1}(x_i^2+x_{i+1}^2)^{0.25}\times\left[\sin(50\times(x_i^2+x_{i+1}^2)^{0.1})+1.0\right]$	Schaffer's f_7	30	$(-100, 100)^N$	0 (0,…,0)

表 2 中,函数 f_1~f_3 是单模态函数.Tablet 和 Quadric 函数是 Spherical 函数的变形,增加了函数各维之间的相互作用(Spherical 函数是一个连续的、简单单模态函数,通常用来分析算法的执行性能).Rosenbrock 函数是一个经典复杂优化问题,它的全局最优点位于一个平滑、狭长的抛物线形山谷内.由于函数仅仅为优化算法提供了

少量信息,使算法很难辨别搜索方向,找到全局最小点的机会微乎其微,因此 Rosenbrock 函数通常用来评价优化算法的执行效率[11].函数 f_4~f_6 是典型的非线性多模态函数.它们具有广泛的搜索空间、大量的局部极小点和高大的障碍物,通常被认为是遗传算法很难处理的复杂多模态问题.

4.2 单模态Benchmark问题对比实验

图 1 显示了 30 维 Quadric 函数的优化实验结果.AEPSO2 算法能够始终保持较快的收敛速度,持续、有效地搜索全局最小点,而使用线形下降惯性权重的 AEPSO1 算法在搜索的开始和后期阶段与 PSO 算法一样,都具有一个比较缓慢的下降过程.这是因为在开始时的权值过大,使微粒大幅度震动不能详细搜索;而后期阶段惯性权重过小、速度下降过快,使 PSO 算法的速度逐渐趋近于 0,使 AEPSO1 算法不能以一个相对大的步长进行局部搜索.虽然 DPSO 和 HPSO 算法也在不同程度上提高了搜索能力,但是由于后期的变异操作带有随机性,反而减缓了算法的收敛速度,而 AEPSO2 算法则能根据微粒速度的变化,给出相应的调整策略,显著提高了算法的收敛速度.

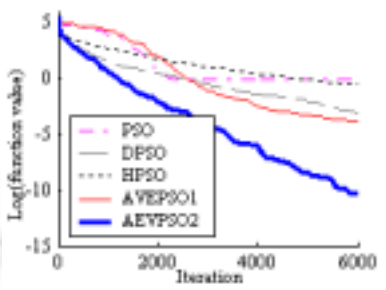

Fig.1 Log minima value for 30-D Quadric

图 1 30 维 Quadric 函数数值对比图

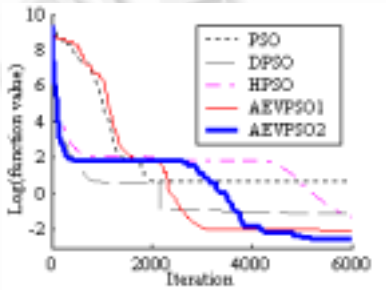

Fig.2 Log minima value for 30-D Rosenbrock

图 2 30 维 Rosenbrock 函数数值对比图

对于复杂单模态 30 维 Rosenbrock 函数问题,AEPSO 算法仍具有较快的收敛速度.在算法初始阶段,使用线性下降惯性权重的 AEPSO1 和 PSO 算法均具有较慢的收敛速度,但是在搜索中期阶段,AEPSO1 借助逃逸运动提高了收敛速度.由于 Rosenbrock 山谷仅仅给算法提供了很少的信息,使算法不能有效地辨别搜索方向,所以图 2 中的所有算法都具有一个相对稳定的阶段,用来寻找搜索方向.对于 AEPSO 算法,都能在一个相对较短的时间内找到正确的搜索方向,并具有较快的下降速度.表 3 中 Rosenbrock 函数的中值、平均值和最差值的数据显示,AEPSO2 算法具有更好的稳定性,单次执行结果之间的相差明显小于其他算法.

表 3 显示了单模态 Benchmark 问题的对比实验结果.从各个算法的最优值和最差值的实验结果中我们可以看出,DPSO 算法在函数 Quadric 和 Rosenbrock 函数上有较快的收敛速度,HPSO 在函数 Tablet 和 Rosenbrock 函数上能取得较好的结果,而 AEPSO 算法在所有函数上均能取得较快的收敛速度,具有更快的局部搜索能力,特别是在 Tablet 和 Rosenbrock 函数上明显优于其他算法.Tablet 函数和 Quadric 函数的测试结果显示,在某些情况下,DPSO 和 HPSO 算法具有比 PSO 算法更差的搜索性能.这是因为通过简单变异操作和消除速度惯性部分的做法增大了 PSO 算法的随机性,使算法不能在搜索后期得到一个较快的搜索速度.而对于单模态 Benchmark 问题,AEPSO 算法的逃逸运动不是使微粒逃离全局最小点,而是在微粒速度过小时,给微粒提供一个较大的惯性速度,使其在搜索空间内保持快速飞行.

由于 PSO 算法在种群初始化和微粒飞行过程中均具有随机性,使得算法的执行结果也具有随机性.表 3 中的实验结果显示,PSO 及其改进算法在 Quadric 和 Tablet 函数优化问题上具有较好的稳定性,所有算法各项数据间的差异都不大.但是对于复杂 Rosenbrock 函数优化问题,由于很难辨别搜索方向,各个算法单次实验结果存在较大差异.表 3 中的各项数据显示,AEPSO2 算法能够快速找到 Rosenbrock 函数的搜索方向,显著提高了 PSO 算法的搜索性能和收敛速度,数据差异相对较小,具有更好的稳定性和健壮性.

Table 3 Performance comparison between AEPSO and other PSOs for unimodal Benchmark problem

表 3 AEPSO 和其他 PSO 算法在单模态 Benchmark 问题上的数值对比

Function	Algorithm	Min	Median	Mean	Deviation	Max
Tablet	PSO	8.2e−22	1.3e−20	1.5e−19	7.8e−19	5.6e−18
	DPSO	1.9e−11	1.0e−10	1.3e−10	9.7e−11	3.7e−10
	HPSO	2.4e−26	2.0e−24	3.9e−24	4.9e−24	1.9e−23
	AEPSO1	4.1e−115	9.7e−109	1.4e−101	9.1e−101	6.4e−100
	AEPSO2	1.5e−128	6.6e−124	2.0e−122	8.8e−122	5.9e−121
Quadric	PSO	0.772	70.45	1.0e+2	1.0e+2	4.8e+02
	DPSO	7.8e−04	3.5e−03	3.8e−03	2.0e−3	1.1e−02
	HPSO	0.255	1.198	1.412	0.802	3.834
	AEPSO1	1.4e−04	1.3e−03	2.1e−03	2.3e−03	9.0e−03
	AEPSO2	4.4e−11	6.5e−10	1.2e−09	1.4e−09	7.5e−09
Rosenbrock	PSO	4.873	77.24	93.71	1.5e+02	1.0e+03
	DPSO	8.0e−02	21.04	34.02	35.10	1.4e+02
	HPSO	5.8e−02	73.53	1.1e+02	1.5e+02	7.2e+02
	AEPSO1	9.4e−03	22.17	40.13	41.78	1.5e+02
	AEPSO2	2.8e−03	8.247	14.00	19.91	76.71

传统的理论分析和实验数据显示,使用线形下降惯性权重 w 更有利于微粒进行局部搜索.但是表 3 的实验数据显示,对所选的测试函数,AEPSO2 算法都显著优于 AEPSO1 算法.这是因为逃逸速度的引入改变了微粒的搜索行为,使其具有一个较快的搜索速度.而 AEPSO1 算法与线形下降惯性权重 PSO 算法一样,由于 w 在开始和结束阶段的差别,使其在这两个阶段都具有相对较慢的搜索速度.所以,本文推荐 w 应该选择一个适中的值,如本文中的 0.7.

4.3 多模态Benchmark问题对比实验

对于 30 维多模态 Griewank 函数,AEPSO1 和 AEPSO2 算法在 50 次实验中分别有 20%和 40%的机会找到近似全局最优值 0.其全局搜索能力显著优于 PSO 及其改进算法,能够有效逃出局部极小点找到全局最小点.AEPSO 的中值、平均值和标准方差等各项数据均显著优于其他算法,显示了新算法的稳定性与健壮性.但是 AEPSO 算法并不能保证在有限代内找到全局极小点,表 4 的中值、平均值两项数据显示 AEPSO 算法也有陷入局部极小的可能,这是因为 Griewank 函数各维之间显著相关,某一维位置的变化并不能提高微粒的适应值,只有当各维同时发生变化时才有可能提高微粒的适应值,这使得微粒逃出极小点的概率非常小.

Table 4 Performance comparison between AEPSO and other PSOs for multi-model Benchmark problem

表 4 AEPSO 和其他 PSO 算法在多模态 Benchmark 问题上的数值对比

Function	Algorithm	Min	Median	Mean	Deviation	Max
Griewank	PSO	6.7e−11	1.96e−02	2.6e−02	2.8e−02	0.108
	DPSO	4.4e−12	1.7e−02	2.5e−02	3.0e−02	0.154
	HPSO	1.1e−16	9.9e−03	1.5e−02	2.1e−02	8.5e−02
	AEPSO1	**0(20%)**	7.4e−03	1.2e−02	1.3e−02	6.1e−02
	AEPSO2	**0(40%)**	8.6e−03	1.2e−02	1.6e−02	6.6e−02
Rastrigin	PSO	30.84	54.72	55.18	12.31	81.59
	DPSO	7.2e−09	7.0e−06	0.201	0.498	2.058
	HPSO	12.93	26.86	29.07	8.963	55.83
	AEPSO1	1.2e−13	0.995	1.234	1.577	7.960
	AEPSO2	**0(20%)**	4.0e−12	0.577	1.084	3.980
Schaffer	PSO	2.4e+02	2.9e+02	2.9e+02	30.91	3.5e+02
	DPSO	82.35	1.5e+02	1.5e+02	36.67	2.4e+02
	HPSO	1.9e+02	2.4e+02	2.4e+02	22.42	2.9e+02
	AEPSO1	54.83	1.5e+02	1.4e+02	41.63	2.4e+02
	AEPSO2	44.57	1.2e+02	1.2e+02	39.39	2.3e+02

多模态 Rastrigrin 函数是一个典型的用来测试算法全局搜索性能的函数.从图 3 和表 4 中我们可以看出,AEPSO 算法具有良好的全局搜索性能和较快的搜索速度.同时,图 3 也显示 PSO 和 HPSO 算法在搜索后期容易陷入局部极小点,不能进一步找到全局极小点.而 DPSO 和 AEPSO 算法都能持续寻找全局最小点,并且 AEPSO2 算法找到近似全局最优值 0.AEPSO1 算法和 AEPSO2 算法虽然没有在表 4 中每次运行都能找到近似全局最优解 0,但是只要给予足够长的迭代次数,AEPSO 算法总能逃出局部极小点找到全局最小点.

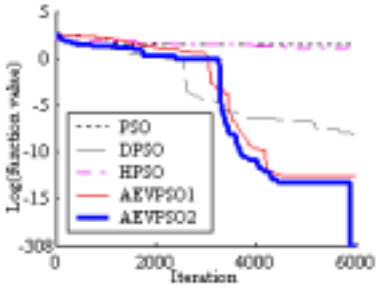

Fig.3 Log minima value for 30-D Rastrigrin

图 3 30 维 Rastrigrin 函数数值对比图

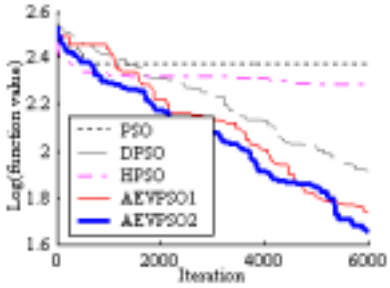

Fig.4 Log minima value for 30-D Schaffer

图 4 30 维 Schaffer 函数数值对比图

图 4 显示了 30 维 Schaffer 函数的运行结果,AEPSO 算法不论是在搜索的初期阶段还是在后期阶段都具有较快的收敛速度,能保持正确的搜索方向,持续地寻找全局最优点,显著优于其他 PSO 算法.而 DPSO 和 HPSO 算法可能是由于随机变异操作的引入,反而降低了搜索性能,陷入了局部极小点.与 Griewank 函数一样,由于微粒没有充分利用各维之间的信息,使算法需要一个缓慢的过程才能找到全局最优点.

从表 4 中我们可以看出,对于多模态 Benchmark 问题,HPSO 和 DPSO 算法均能有效避免 PSO 算法的早熟收敛问题,但是 AEPSO 算法在各个方面进一步提高了全局搜索能力,AEPSO2 算法在有限代数内几乎找到了 Griewank 和 Rastrigrin 函数的近似理论最优值 0,其 50 次实验中找到最好解的机率分别为 40%和 20%.对于 Rastrigrin 和 Schaffer 函数,只要给以足够长的迭代次数,AEPSO2 算法就有能力不断下降找到近似全局最优解.本文中 Rastrigrin 函数问题是因为迭代次数限制了算法的进一步搜索,只要给予足够长的迭代次数,AEPSO 均能找到近似最优值 0;而对于 Griewank 函数问题,虽然有很大几率找到近似最优解 0,但是算法仍旧有陷入局部极小点的可能,特别是在维度较低的情况下.

传统理论分析显示,速度的惯性部分有协调算法局部搜索和全局搜索的能力,但是多模态 Benchmark 问题的实验结果显示,传统参数选择方式对于逃逸运动和变异操作并没有显著帮助.由于逃逸运动的引入,使种群的全局搜索能力伴随着搜索的全部过程,并不会因为代数的增加而消逝.同时,HPSO 的实验数据显示,速度惯性部分的消除反而会在某种程度上降低算法的收敛速度,所以本文并不赞成通过消除微粒的惯性速度部分的方法来增强全局搜索能力.从表 4 中“最优值”、“中值”和“最差值”实验结果之间的差异,我们还可以看出,不论是 PSO 改进算法还是 AEPSO 算法都具有不稳定性,这是由于算法在初始化阶段微粒分布不均匀造成的.

5 结 论

本文借鉴生物界中物种生存密度过大时会自动分家迁移的习性,分析了微粒速度和种群密度之间的关系,结合遗传算法的变异操作,给出了一种解决数值优化问题的改进 PSO 算法——自适应逃逸微粒群算法,并从理论上证明了它具有全局收敛性.AEPSO 算法给出了一种定向变异操作——逃逸运动,一方面通过控制微粒在飞行中速度变化的快慢,有效地提高了 PSO 算法收敛速度;另一方面通过逃逸运动使微粒陷入局部极小点时能有效寻找全局最优解,对于 Griewank 和 Rastrigrin 函数问题在高维情况下能找到近似最优解 0.实验结果显示:新算法不论在单模态和多模态 Benchmark 问题上均好于现存 PSO 及其改进算法;逃逸运动和速度的惯性部分均有加快算法收敛速度的功能;固定惯性权重比线性变化惯性权重更适合 AEPSO 算法.但是在实验中我们发现,AEPSO 算法仅仅显著增加了个体微粒的搜索性能,在迭代后期并没有利用种群间的相互信息;与其他一些进化算法相比,对于 Schaffer 问题仍存在收敛速度比较慢、不能快速逃出局部极小点等特点.所以,进一步的工作主要有以下两个方面:

(1) 提高算法搜索后期的搜索性能和种群最优个体的寻优能力,进一步提高算法收敛速度.

(2) 结合演化算法,利用种群信息找出更有效的逃逸行为,增强算法全局搜索能力.

References:

[1] Kennedy J, Eberhart RC. Particle swarm optimization. In: Proc. of the IEEE Int'l Conf. on Neural Networks. Perth Australia: IEEE Press, 1995. 1942−1948. http://www.engr.iupui.edu/~shi/Coference/psopap4.html

[2] Parsopoulos KE, Vrahatis MN. On the computation of all global minimizers through particle swarm optimization. IEEE Trans. on Evolutionary Computation, 2004,8(3):211−224.

[3] Krink T, Vesterstrom JS, Riget J. Particle swarm optimization with spatial particle extension. In: Proc. of the IEEE Int'l Conf. on Evolutionary Computation. Honolulu: IEEE Inc., 2002. 1474−1497.

[4] Kazemi BAL, Mohan CK. Multi-Phase generalization of the particle swarm optimization algorithm. In: Proc. of the IEEE Int'l Conf. on Evolutionary Computation. Honolulu: IEEE Inc., 2002. 489−494.

[5] Hu XH, Eberhart RC. Adaptive particle swarm optimization: Detection and response to dynamic system. In: Proc. of the IEEE Int'l Conf. on Evolutionary Computation. Honolulu: IEEE Inc., 2002. 1666−1670.

[6] Xie XF, Zhang WJ, Yang ZL. A dissipative particle swarm optimization. In: Proc. of the IEEE Int'l Conf. on Evolutionary Computation. Honolulu: IEEE Inc., 2002. 1456−1461.

[7] Higashi N, Iba H. Particle swarm optimization with Gaussian mutation. In: Proc. of the IEEE Swarm Intelligence Symp. Indianapolis: IEEE Inc., 2003. 72−79.

[8] Kennedy J. Bare bones particle swarms. In: Proc. of the IEEE Swarm Intelligence Symp. Indianapolis: IEEE Press, 2003. 53−57.

[9] Zhang WJ, Xie XF. DEPSO: Hybrid particle swarm with differential evolution operator. In: Proc. of the IEEE Int'l Int'l Conf. on Systems, Man and Cybernetics. Washington: IEEE Inc., 2003. 3816−3821.

[10] Lovbjerg M, Krink T. Extending particle swarm optimizers with self-organized critically. In: Proc. of the IEEE Int'l Conf. on Evolutionary Computation. Honolulu: IEEE Inc., 2002. 1588−1593.

[11] Ratnaweera A, Halgamuge SK, Watson HC. Self-Organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. IEEE Trans. on Evolutionary Computation, 2004,8(3):240−255.

[12] Ozcan E, Mohan CK. Particle swarm optimzation: Surfing the waves. In: Proc. of the IEEE Int'l Conf. on Evolutionary Computation. Washington: IEEE Inc., 1999. 1939−1944.

[13] Clerc M, Kennedy J. The particle swarm——Explosion, stability, and convergence in a multidimensional complex space. IEEE Trans. on Evolutionary Computation, 2002,6(1):58−73.

[14] Cristian TI. The particle swarm optimization algorithm: Convergence analysis and parameter selection. Information Processing Letters, 2003,85(6):317−325.

[15] Li H, Tang HW, Guo CH. The convergence analysis of a class of evolution strategies. OR Trans., 1999,3(4):79−83 (in Chinese with English abstract).

[16] Guo CH, Tang HW. Global convergence properties of evolution strategies. Mathematica Numerica Sinica, 2001,23(1):105−110 (in Chinese with English abstract).

附中文参考文献:

[15] 李宏,唐焕文,郭崇慧.一类进化策略的收敛性分析.运筹学学报,1999,3(4):79−83.

[16] 郭崇慧,唐焕文.演化策略的全局收敛性.计算数学,2001,23(1):105−110.