文章编号: 1001-0920(2011)08-1158-05

一种高效粒子群优化算法

高卫峰, 刘三阳

(西安电子科技大学 数学科学系, 西安 710071)

摘 要: 针对标准粒子群算法收敛速度慢和易出现早熟收敛等问题, 提出一种高效粒子群优化算法. 首先利用局部搜索算法的局部快速收敛性, 对整个粒子群目前找到的最优位置进行局部搜索; 然后, 为了跳出局部最优, 保持粒子的多样性, 给出一个学习算子. 该算法能增强算法的全局探索和局部开发能力. 通过对 10 个标准测试函数的仿真实验并与其他算法相比较, 结果表明了所提出的算法具有较快的收敛速度和很强的跳出局部最优的能力, 优化性能得到显著提高.

关键词: 粒子群优化; 局部搜索; 学习算子; 差分进化

中图分类号: TP18 **文献标识码:** A

An efficient particle swarm optimization

GAO Wei-feng, LIU San-yang

(Department of Applied Mathematics, Xidian University, Xi'an 710071, China. Correspondent: GAO Wei-feng, E-mail: gaoweifeng2004@126.com)

Abstract: To the problems of low searching speed and premature convergence frequently appeared in standard particle swarm optimization(PSO) algorithm, an efficient particle swarm optimization(AEPSO) is proposed in this paper. The method makes full use of the local convergent performance of the local random search algorithm to optimize the global best position of the swarm found so far. Then to go out of the local optimum in PSO and maintain the population diversity in the process of evolution, a learning operator is presented. This algorithm can enhance the exploration and exploitation ability of the algorithm. Through testing the performance of the proposed approach on a suite of 10 benchmark functions and comparing with other meta-heuristics, the result of simulation shows that the proposed approach has better convergence rate, great capability of preventing premature convergence and superior performance.

Key words: particle swarm optimization; local search; learning operator; differential evolution

1 引 言

粒子群优化 (PSO) 算法是 Kennedy 和 Eberhart[1] 于 1995 年提出的一种智能优化算法. PSO 算法来源于对聚集生物行为, 例如鸟群和鱼群的觅食行为的研究. 由于算法结构简单、易于实现、无需梯度信息、参数较少等特点, 一经提出便受到众多学者的关注和研究, 并在函数优化、多目标优化、模式识别、系统辨识等诸多领域得到广泛应用. 与其他智能算法类似, 标准 PSO 算法也存在早熟收敛和局部寻优能力差等缺点. 这些缺点限制了 PSO 在实际中的应用.

对此, 人们提出了多种改进策略[2-6]来优化 PSO 的性能. 文献 [2] 引入动态惯性权重来提高算法的收敛速度; [3] 将混沌系统嵌入 PSO 的机制中以避免早熟收敛; [4] 利用单纯形法的快速收敛性, 提出了混合单纯形粒子群算法; [5] 指出早熟是由于粒子速度降低而失去继续搜索可行解的能力, 进而提出一种基于种群速度动态改变惯性权重的粒子群算法. 然而, 到目前为止, 很难实现在提高算法局部寻优能力的同时避免早熟收敛. 例如, 基于广泛学习的 PSO 算法 (LPSO)[6] 侧重于避免早熟收敛, 但却以牺牲算法收敛速度为代价.

为了提高标准 PSO 算法的收敛速度和寻优精度, 本文提出一种克服早熟的高速收敛的改进粒子群算法. 首先, 对整个粒子群当前找到的最优位置进行局部搜索, 以加快算法的收敛速度; 然后, 在标准粒子群算法中设计一个学习算子, 一旦检测到早熟迹象, 便

收稿日期: 2010-05-10; **修回日期:** 2010-09-22.

基金项目: 国家自然科学基金项目(60974082); 中央高校基本科研业务费专项资金项目(K50510700004).

作者简介: 高卫峰(1985–), 男, 博士生, 从事进化计算、最优化理论与方法的研究; 刘三阳(1959–), 男, 教授, 博士生导师, 从事智能信息处理、最优化方法等研究.

通过学习算子使其尽快跳出局部最优. 该方法能提高算法的探索和开发能力, 避免算法因陷入局部最优而导致早熟收敛. 大量的仿真实验表明, 该算法能显著提高优化效率和优化性能.

2　标准 PSO 算法

标准PSO算法是模拟鸟群飞行觅食的行为, 通过个体之间的协作来寻找最优解的进化计算技术. 设在 D 维空间中, 有 m 个粒子组成一个群落, 其中第 i 个粒子的位置 $\boldsymbol{x}_i=[x_{i1},x_{i2},\cdots,x_{iD}]^{\mathrm{T}}$, 速度 $\boldsymbol{v}_i=[v_{i1},v_{i2},\cdots,v_{iD}]^{\mathrm{T}}$, $i=1,2,\cdots,m$. 第 i 个粒子搜索到的历史最优位置为 $\boldsymbol{p}_i$, 整个粒子群搜索到的最优位置为 $\boldsymbol{p}_g$.

将 $\boldsymbol{x}_i$ 代入目标函数, 计算其适应值. 对于第 $k+1$ 次迭代, 每个粒子根据下式更新自己的速度和位置:

$$v_{id}^{k+1}=\omega v_{id}^{k}+c_1r_1(p_{id}-x_{id}^{k})+c_2r_2(p_{gd}-x_{id}^{k}), \quad (1)$$

$$x_{id}^{k+1}=x_{id}^{k}+v_{id}^{k+1}. \quad (2)$$

其中: $d=1,2,\cdots,D$; ω 为惯性权重; c_1 和 c_2 分别为认知参数和社会参数; r_1 和 r_2 为 [0, 1] 之间的随机数.

3　高效粒子群优化算法

3.1　局部搜索算子

最近 Hamzacebi 和 Kutay[7-8]提出一种新的启发式算法——动态随机搜索技术. 它由一般搜索和局部搜索两部分构成. 其局部搜索过程具有很强的搜索能力, 具体步骤如下:

Step 1: 初始化. 设定局部搜索代数 E, 搜索步长上限 α_0, $X_{\mathrm{current}}=X_{\mathrm{best}}$, epoch $=0$, $k=0$.

Step 2: 重置迭代计数器, $n=0$.

Step 3: 产生随机向量 dx, 且 $-\alpha_k\leqslant \mathrm{dx}\leqslant\alpha_k$.

Step 4: 更新 epoch, 即 epoch $=$ epoch $+1$.

Step 5: $f_{\mathrm{new}}=f(X_{\mathrm{current}}+\mathrm{dx})$.

if $f_{\mathrm{new}}<f_{\mathrm{best}}$, then $f_{\mathrm{best}}=f_{\mathrm{new}}$,

$X_{\mathrm{best}}=X_{\mathrm{current}}+\mathrm{dx}$.

$n=n+1$, 转 Step 7.

if $f_{\mathrm{new}}<f_{\mathrm{current}}$, then $f_{\mathrm{current}}=f_{\mathrm{new}}$,

$X_{\mathrm{current}}=X_{\mathrm{current}}+\mathrm{dx}$.

$n=n+1$, 转 Step 7.

Step 6: $f_{\mathrm{new}}=f(X_{\mathrm{current}}-\mathrm{dx})$.

if $f_{\mathrm{new}}<f_{\mathrm{best}}$, then $f_{\mathrm{best}}=f_{\mathrm{new}}$,

$X_{\mathrm{best}}=X_{\mathrm{current}}-\mathrm{dx}$.

$n=n+1$, 转 Step 7.

if $f_{\mathrm{new}}<f_{\mathrm{current}}$, then $f_{\mathrm{current}}=f_{\mathrm{new}}$,

$X_{\mathrm{current}}=X_{\mathrm{current}}-\mathrm{dx}$.

$n=n+1$, 转 Step 7.

Step 7: if $n<N$, 则转 Step 3.

Step 8: $k=k+1$.

Step 9: $\alpha_k=\alpha_{k-1}\times 0.5$.

Step 10: if epoch $=E$, 则算法终止; 否则, 转 Step 2.

其中: epoch为局部搜索迭代计数器; X_{best} 为算法目前找到的最优解; $f(\cdot)$ 为计算适应值的函数; $f_{\mathrm{best}},f_{\mathrm{new}},f_{\mathrm{current}}$ 为对应解处的适应值.

3.2　学习算子

在标准粒子群算法中, $\boldsymbol{p}_g$ 对种群起着非常重要的引导作用. 如果 $\boldsymbol{p}_g$ 陷入局部最优而跳不出去, 则它会把所有的粒子引向局部最优区域, 导致算法出现早熟现象. 如果对陷入局部最优的 $\boldsymbol{p}_g$ 给出一个随机的大范围的变异, 虽然能在一定程度上使算法跳出局部最优位置, 但同时增加了算法无效迭代的次数, 从而降低了算法的收敛速度.

针对以上问题, 本文设计了一个学习算子. 当检测到早熟迹象时, 分别对 $\boldsymbol{p}_g$ 进行倒位操作和简单操作, 然后以一定的概率对种群进行非一致性变异.

倒位操作是指: 个体 $\boldsymbol{p}_g=[x_{g1},\cdots,x_{gi},\cdots,x_{gj},\cdots,x_{gD}]^{\mathrm{T}}$, 倒位后为 $\boldsymbol{p}_g^1=[x_{g1},\cdots,x_{gj},x_{gj-1},\cdots,x_{gi-1},x_{gi},\cdots,x_{gD}]^{\mathrm{T}}$, 其中 i 和 j 为倒位点.

简单操作是指: 随机选择一个个体 $\boldsymbol{x}_i$, 产生新个体 $\boldsymbol{p}_g^2$. 其中: $p_{gd}^2=p_{gd}+\lambda(p_{gd}-x_{id})(d\in 1,2,\cdots,D)$ 为随机选取的一个数, λ 为 [−1, 1] 之间的随机数.

非一致性变异是指: 设 $\boldsymbol{x}_i=[x_{i1},x_{i2},\cdots,x_{iD}]^{\mathrm{T}}$ 为一个体, 分量 x_{ik} 被选中进行变异, 其定义区间为 $[a_k,b_k]$, 则变异后的解为 $\boldsymbol{x}_i'=[x_{i1},x_{i2},\cdots,x_{ik}',\cdots,x_{iD}]^{\mathrm{T}}$. 其中 x_{ik}' 从下面两种可能的方案中随机选择:

$$x_{ik}'=x_{ik}+\Delta(t,b_k-x_{ik}), \quad (3)$$

$$x_{ik}'=x_{ik}-\Delta(t,x_{ik}-b_k). \quad (4)$$

具体细节可参见文献 [9].

用 $\boldsymbol{p}_g^1$ 和 $\boldsymbol{p}_g^2$ 中较优个体更新 $\boldsymbol{p}_g$, 这样既可更新 $\boldsymbol{p}_g$, 又不使其漫无目的地跳出局部最优位置; 同时, 以一定的概率对种群进行非一致性变异, 可使算法快速跳出局部最优区域.

3.3　高效粒子群优化算法的基本思想和步骤

标准粒子群算法在寻优过程中存在收敛速度慢、容易出现早熟的缺点, 严重影响了算法的性能. 针对这一问题, 本文提出了一种改进的粒子群算法 (AEPSO). 其基本思想是: 在PSO算法的迭代中引入局部搜索算子, 对整个粒子群目前找到的最优位置进行局部搜索, 使其快速收敛到搜索区域的最优位置. 为了防止算法出现早熟收敛现象, 设计了一个学习算子. 当检测到出现早熟迹象时, 对粒子群当前找到的最优位置通过倒位操作和简单操作来更新, 赋予其打

破僵局的能力, 使其有目的性地跳出局部最优位置; 然后, 对种群以一定变异概率进行非一致性变异操作, 以加快算法跳出局部最优区域, 从而有效地减少无效迭代的次数. 这样即可在提高算法收敛速度和保持种群多样性两方面取得平衡, 使算法的性能得到大幅度的提高.

因为本文引入的局部搜索算子搜索能力很强, 所以当粒子群目前找到的最优位置进行局部搜索后适应值没有变化时, 很有可能是 $\boldsymbol{p}_g$ 陷入局部最优所致. 大量实验表明, 若本次迭代找到的最优值与上次找的最优值之差不小于误差限 ε, 则算法不易陷入局部最优位置. 这样便可不用进行学习算子操作, 从而在一定程度上降低了算法的计算量; 否则, 算法很有可能陷入局部最优. 根据这两种情况, 对于判断 $\boldsymbol{p}_g$ 是否陷入局部最优, 算法在 Step 4 中给出了实现这部分的计算与判断.

算法的具体步骤如下:

Step 1: 初始化. 设定影响因子 c_1, c_2 和 r_1, r_2, 惯性权重 ω, 最大函数评价次数 M. 在定义空间中, 随机产生 m 个粒子组成初始种群 $\boldsymbol{x}_i$, 产生各粒子的初始速度 $\boldsymbol{v}_i\,(i=1,2,\cdots,m)$, 变异概率 p, 以及误差限 ε.

Step 2: 计算各粒子的适应度函数值, 更新各粒子的 $\boldsymbol{p}_i$ 和 $\boldsymbol{p}_g$.

Step 3: 依据式 (1) 和 (2) 更新各粒子速度和空间位置.

Step 4: 运用局部搜索算子对 $\boldsymbol{p}_g$ 进行进一步的搜索, 并更新 $\boldsymbol{p}_g$.

Step 5: 若粒子群当前找到的最优位置进行局部搜索后适应值没有变化, 且本次迭代找到的最优值与上次找的最优值之差小于误差限 ε, 则可判断 $\boldsymbol{p}_g$ 陷入局部最优, 通过学习算子对种群进行操作; 否则, 转 Step 6.

Step 6: 检验是否满足停止条件. 若满足, 则停止迭代, 并输出最优粒子的 $\boldsymbol{p}_g$ 及其对应的适应度值; 否则, 返回 Step 2.

4 仿真实验

4.1 AEPSO 与其他算法性能比较

为了测试本文提出的改进粒子群算法的效果, 对文献 [10] 中的 10 个标准测试函数进行仿真实验, 测试函数如表 1 所示. 其中: $f_1 \sim f_4$ 为单峰函数, $f_5 \sim f_{10}$ 为多峰函数.

种群规模为 30. ω, c_1, c_2 采用文献 [11] 的参数设置方案, 即 $\omega = 0.9$, $c_1 = 0.5$, $c_2 = 0.3$, r_1 =rand, r_2 =rand; 局部搜索算子中, $E = 100$, $N = 10$; 学习算子中, $\varepsilon = 10^{-3}$, $p = 0.3$. 将本文算法与标准的粒子群和差分进化 (DE) 算法进行比较, 它们的参数设置参见文献 [11-12]. 采用 Matlab 7.0 编程工具进行仿真实验. MFV, SD 和 T 分别为算法独立实验 30 次的平均最优适应值、标准差和平均运行时间. MFV 显示了在给定的函数评价次数下算法所能达到的精度, 反映了算法的收敛速度; SD 反映了算法的稳定性和鲁棒性; T 反映了算法的时间效率. 结果如表 2 所示.

表 1 测试函数的维数、搜索空间和最优值

函数	名称	维数	最大评价次数	搜索空间	最优值
f_1	Sphere	30	100, 000	$[-100, 100]$	0
f_2	Schwefel 2.22	30	100, 000	$[-10, 10]$	0
f_3	Schwefel 1.2	30	100, 000	$[-100, 100]$	0
f_4	Rosebrock	30	200, 000	$[-30, 30]$	0
f_5	Schwefel	30	200, 000	$[-500, 500]$	$-12\,569.5$
f_6	Rastrigin	30	200, 000	$[-5.12, 5.12]$	0
f_7	Griewank	30	200, 000	$[-600, 600]$	0
f_8	Ackley	30	200, 000	$[-32, 32]$	0
f_9	Penalized	30	200, 000	$[-50, 50]$	0
f_{10}	Penalized2	30	200, 000	$[-50, 50]$	0

表 2 3 种算法对 10 个函数的计算结果比较

函数	性能指标	DE	PSO	AEPSO
f_1	MFV	7.01e-011	1.21e-010	1.70e-023
	SD	9.71e-011	2.07e-010	1.63e-023
	T/s	1.90	1.88	1.89
f_2	MFV	5.49e-007	2.580 159	3.23e-011
	SD	9.93e-008	1.393 700	1.66e-011
	T/s	1.71	1.73	1.72
f_3	MFV	8.470 035	14.065 476	0.491 113
	SD	12.476 960	8.275 569	0.380 573
	T/s	6.01	5.74	5.86
f_4	MFV	47.602 178	17.586 022	4.318 630
	SD	33.067 438	25.192 361	5.014 865
	T/s	4.21	4.43	4.15
f_5	MFV	-8 467.186	-6 670.543	-10 050.368
	SD	9.49e+002	4.48e+002	3.52e+002
	T/s	7.62	7.63	7.93
f_6	MFV	46.762 984	50.543 843	0
	SD	21.695 685	7.158 155	0
	T/s	4.42	4.21	4.63
f_7	MFV	1.001 415	0.021 638	0
	SD	1.623 554	0.026 344	0
	T/s	7.63	7.75	7.87
f_8	MFV	10.038 785	4.658 622	2.70e-011
	SD	1.974 086	0.762 242	6.12e-012
	T/s	6.58	6.62	5.90
f_9	MFV	9.355 329	7.703 553	5.13e-032
	SD	4.497 648	3.778 742	2.61e-032
	T/s	70.3	65.8	68.3
f_{10}	MFV	34.696 755	21.044 640	2.71e-028
	SD	33.500 469	21.509 867	1.33e-028
	T/s	72.1	70.05	71.3

由表 2 数据对比可以看出, 在 10 个标准测试函数中, 无论是算法的收敛精度, 还是算法的稳定性, AEPSO 算法都比标准 PSO 算法有了很大的提高, 其性能也远远优于 DE 算法. 相比于其他算法, AEPSO 没有明显增加时间复杂度. 然而正如"没有免费的午餐定理[13]"所指出的, 任何一种算法不可能在所有的性能方面占尽优势. 虽然本文算法在收敛精度和稳定性方面具有一定的优势, 但时间效率不是最好的. 其中对 f_5, f_6, f_7 的时间效率较差, 这是引入了较多次学习算子所致. 这 3 个函数是高维多峰函数, 已有的算法都很难处理. AEPSO 算法在这几个函数的优化上取得了比较满意的结果, 这与本文设计学习算子和淘汰机制的初衷是吻合的.

为了更直观地反映算法的寻优效果, 将 AEPSO 算法与标准 PSO 算法和 DE 算法进行比较. 3 种算法对相关测试函数的收敛曲线如图 1～图 4 所示. 由图可知, 本文提出的粒子群算法由于引入了局部搜索算子, 增加了评价次数, 在早期并没有优势; 但在进化中后期, 收敛曲线的下降速度比标准 PSO 和 DE 要快得多, 且目标函数值也下降到更低的水平. 因此可以得

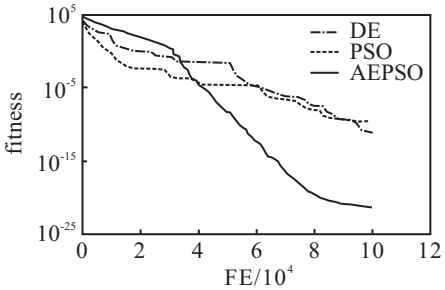

图 1　f_1 函数的收敛性能比较

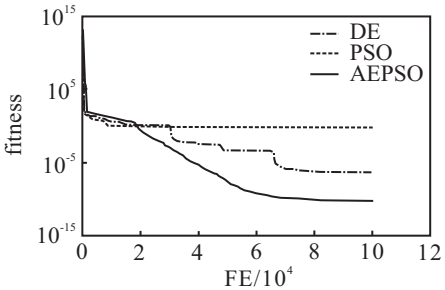

图 2　f_2 函数的收敛性能比较

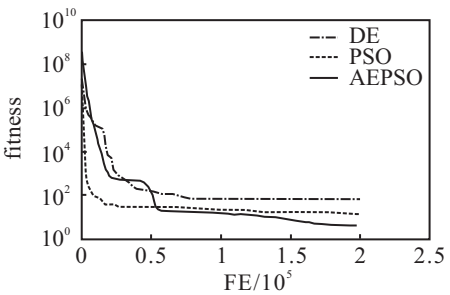

图 3　f_4 函数的收敛性能比较

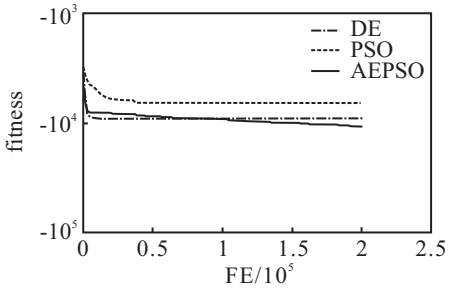

图 4　f_5 函数的收敛性能比较

出结论: 本文算法的优化能力强于标准 PSO 和 DE.

为进一步展示 AEPSO 算法的先进性, 将 AEPSO 算法与文献 [10] 和文献 [14-16] 算法的结果进行比较, 如表 3 所示. 对于上述文献中没有计算的函数, 本文没有给出比较结果. 从表 3 中可以看出, 对于除 f_1 外的 4 个函数, AEPSO 算法的性能均有较大幅度的改善, 其优化结果和理论值的误差最小.

表 3　各种算法计算结果对比

算法	f_1	f_4	f_6	f_7	f_8
HP1CN	6.614e-010	34.581 9	29.396 4	0.094 8	0.024 8
HP2CN	3.780e-010	44.535 4	33.589 1	0.138 4	0.024 1
HP1DN	1.161e-009	28.977 6	31.529 7	0.395 2	0.021 5
HP2DN	1.254e-009	22.549 2	34.473 4	1.157 8	0.023 1
SPSO	2.26e-010	289.593	37.279 6	0.012 6	N
QPSO	1.87e-028	59.029 1	22.959 4	0.116 1	N
AQPSO	5.04e-006	61.622 8	35.236 6	0.014 23	N
LEP	6.32e-004	43.40	5.85	2.4e-002	1.9e-002
FEP	6.32e-004	5.06	4.6e-002	1.6e-002	1.8e-002
AEPSO	1.70e-023	4.318 6	0	0	2.70e-011

4.2　AEPSO 算法性能分析

为了分析算法中局部搜索和学习算子对算法性能的影响, 将标准的 PSO, 带有局部搜索的 PSO (PSOL) 和 AEPSO 进行比较, 所得结果如表 4 所示.

从表 4 中可以看出, 对于 $f_1 \sim f_4$ 这 4 个单峰函数而言, PSOL 和 AEPSO 比标准 PSO 的性能有很大的提高; PSOL 和 AEPSO 的性能则相差无几. 这充分说明局部搜索对单峰函数的寻优起到了很好的作用. 对于 $f_5 \sim f_{10}$ 这 6 个多峰函数而言, AEPSO 比标准 PSO 和 PSOL 的性能有了很大的提高; 而在标准的 PSO 和 PSOL 中, 对于 f_5, f_6 和 f_8 而言, 标准的 PSO 较好一些; 对于 f_7, f_9 和 f_{10}, PSOL 则较好一些. 这说明引入局部搜索虽然在一定程度上提高了算法的搜索能力, 但也使算法更容易陷入局部最优位置, 难以搜索到全局最优解.

从上面的分析可知, AEPSO 算法由于引入了局部搜索和学习算子, 在提高算法收敛速度和维持种群多样性两方面取得了平衡, 避免了陷入局部最优, 得到了较好的优化结果.

表 4 改进措施及其综合实验结果

函数	性能指标	PSO	PSOL	AEPSO
f_1	MFV	1.21e-010	6.50e-023	1.70e-023
	SD	2.07e-010	6.50e-023	1.63e-023
f_2	MFV	2.580 159	6.54e-011	3.23e-011
	SD	1.393 700	2.35e-011	1.66e-011
f_3	MFV	14.065 476	0.554 254	0.491 113
	SD	8.275 569	0.528 909	0.380 573
f_4	MFV	17.586 022	5.101 500	4.318 630
	SD	25.192 361	12.671 068	10.525 348
f_5	MFV	−6 770.543	−6 676.436	−10 050.368
	SD	4.48e+002	5.02e+002	3.52e+002
f_6	MFV	50.543 843	98.500 632	0
	SD	7.158 155	22.269 936	0
f_7	MFV	0.021 638	0.012 554	0
	SD	0.026 344	0.008 415	0
f_8	MFV	4.658 622	5.866 236	2.70e-011
	SD	0.762 242	3.292 844	6.12e-012
f_9	MFV	7.703 553	0.269 6 58	5.13e-032
	SD	3.778 742	0.275 277	2.61e-032
f_{10}	MFV	21.044 640	0.086 081	2.71e-028
	SD	21.509 867	0.155 739	1.33e-028

5 结 论

为解决标准粒子群算法收敛速度慢和早熟收敛等问题, 本文提出了一种高效粒子群算法. 为了平衡算法的全局探索和局部开发能力, 改进的算法引入了局部搜索和学习算子. 从而, 既提高了算法的收敛速度和精度, 又保持了种群的多样性. 对 10 个函数寻优的实验结果表明, 本文算法在优化效率、优化性能和鲁棒性方面比标准 PSO, DE 及其他 4 种算法均有较大的改善.

新算法也有自身的局限性. 由于在标准的 PSO 算法中引入局部搜索算子, 增加了函数的评价次数, 在进化初期没有优势. 但在进化的中后期, 通过学习算子与局部搜索算子的相互作用, 使得算法的性能得到改善, 取得了令人满意的结果.

参考文献(References)

[1] Kennedy J, Eberhartr C. Particle swarm optimization[C]. Proc of IEEE Int Conf on Neural Networks. Perth: IEEE Piscataway, 1995: 1942-1948.

[2] Jiao B, Lian Z G, Gu X S. A dynamic inertia weight particle swarm optimization algorithm[J]. Chaos Solitons & Fractals, 2008, 37(3): 698-705.

[3] Jiang C W, Etorre B. A hybrid method of chaotic particle swarm optimization and linear interior for reactive power optimization[J]. Mathematics and Computers in Simulation, 2005, 68(1): 57-65.

[4] Fan S F, Zahara E. Hybrid simplex search and particle swarm optimization for unconstratined optimization problems[J]. European J of Operational Research, 2007, 181(2): 527-548.

[5] 张顶学, 廖锐全. 一种基于种群速度的自适应粒子群算法[J]. 控制与决策, 2009, 23(7): 756-761.
(Zhang D X, Liao R Q. Adaptive particle swarm optimization algorithm based on population velocity[J]. Control and Decision, 2009, 23(7): 756-761.)

[6] Liang J J, Qin A K, Suganthan P N, et al. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions[J]. IEEE Trans on Evolutionary Computation, 2006, 10(3): 281-295.

[7] Hamzacebi C, Kutay F. Continuous functions minimization by dynamic random search technique[J]. Applied Mathematical Modelling, 2007, 31(10): 2189-198.

[8] Hamzacebi C. Improving genetic algorithms' performance by local search for continuous function optimization[J]. Applied Mathematics and Computation, 2008, 196(1): 309-317.

[9] 潘正君, 康立山, 陈毓屏. 演化计算[M]. 北京: 清华大学出版社, 1998.
(Pan Z J, Kang L S, Chen Y P. Evolutionary computation[M]. Beijing: Tsinghua University Press, 1998.)

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

[11] Zhao X C. A perturbed particle swarm algorithm for numerical optimization[J]. Applied Soft Computing, 2010, 10(1): 119-124.

[12] Noman N, Iba H. Accelerating differential evolution using an adaptive local search[J]. IEEE Trans on Evolutionary Computation, 2008, 12(1): 107-125.

[13] Wolpert D H, Macready W G. No free lunch theorems for optimization[J]. IEEE Trans on Evolutionary Computation, 1997, 1(1): 67-82.

[14] Lee C, Yao X. Evolutionary programming using mutations based on the levy probability distribution[J]. IEEE Trans on Evolutionary Computation, 2004, 8(1): 1-13.

[15] Gimmler J, Stutzle T, Exner T. Hybrid particle swarm optimization: An examination of the influence of iterative improvement algorithms on performance[C]. Proc of the 5th Int Workshop on Ant Colony Optimization and Swarm Intelligence. Berlin: Springer-Verlag, 2006: 436-443.

[16] Xu W B, Sun J. Adaptive parameter selection of quantum-behaved particle swarm optimization on global level[C]. Proc of Int Conf of Intelligent Computing. Berlin: Springer-Verlag, 2005: 420-428.