第32卷 第15期 计 算 机 工 程 2006年8月
Vol.32 № 15 Computer Engineering August 2006
·博士论文· 文章编号：1000—3428(2006)15—0025—03 文献标识码：A 中图分类号：TP18

一种改进的微粒群优化算法

郑小霞，钱 锋
(华东理工大学自动化研究所，上海 200237)

摘 要：提出了一种基于差分进化算子变异的改进微粒群优化算法，为减小陷入局优的可能性，在群体最优信息陷入停滞时引入差分进化算子变异，使算法摆脱局部极优点的束缚，同时又保持前期搜索速度快的特性，提高全局搜索能力。仿真实验表明：与标准微粒群优化算法相比，该文算法的全局收敛性能得到了显著提高，能有效避免微粒群优化算法中的早熟收敛问题。

关键词：微粒群；优化；差分进化；变异

A Modified Particle Swarm Optimization Algorithm

ZHENG Xiaoxia, QIAN Feng
(Institute of Automation, East China University of Science and Technology, Shanghai 200237)

【Abstract】 This paper proposes a modified particle swarm optimization (MPSO) with differential evolution operator mutation. When the optimum information of the warm is stagnant, differential evolution operator mutation is introduced to reduce the possibility of trapping at the local optimum. By adding the mutation operator to the PSO algorithm, the advantaged algorithm can maintain the characteristic of fast speed in the early convergence phase and improve the global search ability. The experimental results indicate that MPSO not only has great advantage of convergence property over PSO, but also can avoid the premature convergence problem effectively.

【Key words】 Particle swarm; Optimization; Differential evolution; Mutation

微粒群优化(Particle Swarm Optimization, PSO)[1]是由Kennedy和Eberhart等人于1995年提出的，是群智能的代表性方法之一。虽然PSO算法发展迅速并取得了可观的研究成果，但在复杂的多峰问题中易陷入局部最优，增大微粒数目对算法性能有一定改善，但不能从根本上解决问题。如果能为PSO算法提供一种新机制，使其在陷入局部最优时，以更大概率跳出局部最优位置，进入解空间的其它区域进行搜索，就可大大增强PSO算法的全局搜索能力。

1 基本微粒群优化算法

1.1 算法原理

PSO算法将每个个体看作D维搜索空间中的一个没有体积的微粒，在搜索空间中以一定的速度飞行，并根据它本身和同伴的飞行经验进行动态调整。设第i个微粒表示为$x_i=(x_{i1},\dots,x_{id},\dots,x_{iD})$，它经历过的最好位置记为$p_i=(p_{i1},\dots,p_{id},\dots,p_{iD})$，也称$pbest$。在群体所有微粒经历的最好位置的索引号用符号$g$表示，即$P_g$，也称为$g_{best}$。微粒$i$的速度用$v_i=(v_{i1},\dots,v_{id},\dots,v_{iD})$。对每一代，其第$d$维根据如下方程变化：

$$v_{id}=wv_{id}+c_1rand_1()(p_{id}-x_{id})+c_2rand_2()(p_{gd}-x_{id}) \quad (1a)$$

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

其中w为惯性权重，c_1和c_2为加速常数，$rand_1()$和$rand_2()$为[0,1]之间的随机数。微粒通过不断学习更新，最终飞至解空间中最优解所在的位置，最后输出的$gbest$就是算法找到的全局最优解。

1.2 算法解析

式(1a)右边的的第1部分为微粒先前的速度，它维持算法拓展搜索空间的能力；第2部分为“认知”部分，表示微粒自身的思考，防止算法陷入局部最优；第3部分为“社会”部分，表示微粒间的信息共享和互相合作。若没有第3部分，就意味着个体之间没有交互，一个规模为m的群体等价于m个单个微粒的运行，此时得到最优解的概率就很小。正如Kennedy所指出的，PSO算法的寻优能力主要来自于微粒之间的相互作用和相互影响，如果从算法中去除相互作用和影响，则PSO算法的寻优能力就变得非常有限[2]。对一些函数的测试结果也验证了这一点。

如果微粒群的历史最优微粒位置P_g在较长时间内未发生变化，则微粒群很接近P_g时，其速度更新将主要由$wv_{i\,d}$来决定，$w<1$时速度将越来越小，此时微粒群表现出强烈的“趋同性”。$v_{i\,d}$接近于0时，微粒将慢慢停止前进，最后收敛到搜索空间的某一位置。微粒数越少，这种停滞越明显。实际上，此时的收敛并不能保证找到全局最优，甚至不能保证找到局部最优，这仅仅意味着微粒收敛于目前为止微粒所能找到的最好位置。Van den Bergh对于PSO的收敛性和稳定性作了初步分析，并给出了一些保证PSO算法收敛的参数条件[3]。

2 改进的微粒群优化算法

从前面分析知，PSO算法在运行过程中，很容易发生早熟收敛。如果当前最优解为一局部最优位置，那么一旦所有微粒都收敛于该位置之后，这些微粒将很难跳出局部最优，虽然遗传算法也存在这样的问题，但是变异及交叉等机制能

基金项目：国家“973”计划基金资助项目(2002CB3122000)；国家自然科学基金资助项目(60074027)；国家“863”计划基金资助项目(2003AA412010)

作者简介：郑小霞(1978—)，女，博士生，主研方向：工业过程智能故障诊断与优化等；钱 锋，教授、博导

收稿日期：2005-11-09 **E-mail：**zxxkx@163.com

够增强遗传算法跳出局部最优的能力。因此，为了克服 PSO 算法的早熟收敛问题，也必须提供一种机制，在算法发生停滞时使微粒能够及时跳出局部最优。Van den Bergh 提出了 Multistart PSO，每次迭代若干次后，保留微粒群的历史最优位置，微粒全部重新初始化，以提高微粒的多样性，扩大搜索空间，摆脱局部最优点的吸引，保证收敛到全局最优[3]。Zhang, W 等将差分进化算子变异引入 PSO 中，并采用 PSO 算法和变异操作交替迭代的方式[4]。但微粒群的全部初始化将会完全破坏当前微粒的结构，而 PSO 算法和变异操作交替迭代也可能会破坏 PSO 在正常寻优的情况下微粒的结构，使得收敛速度减缓，而搜索精度也可能降低，体现不出 PSO 算法本身的优势。

鉴于此，本文提出一种基于差分进化算子变异的改进微粒群算法。与文献[4]不同的是，本文在微粒群发现的全局最大适应值 P_g 连续 G 代没有改进时，引入变异操作改变微粒的位置，进而改变微粒的前进方向，让微粒进入其它区域进行搜索。在其后的搜索过程中算法就可能发现新的个体极值 *pbest* 和全局极值 *gbest*，如此循环，算法就可找到全局最优解。这就是本文提出变异的基本思想，下面对差分进化算法做简单介绍。

2.1 差分进化的基本原理

差分进化(Differential Evolution, DE)算法是由 Storn 和 Price 提出的一种较新的全局并行搜索算法[5]。差分进化也有类似遗传算法的变异、交叉和选择等操作，其中变异操作定义如下：

$$v_{i,G+1} = x_{r_1,G} + F \cdot (x_{r_2,G} - x_{r_3,G}) \tag{2}$$

式中：r_1，r_2，$r_3 \in \{1,2,\ldots,NP\}$，$F$ 是位于[0,2]的一个常数，式(2)表示从群体中随机取出两个矢量 $x_{r2,G}$ 和 $x_{r3,G}$，将二者相减，二者的差经 F 放大后加到第 3 个矢量 $x_{r1,G}$ 上，得到下一代新的矢量 $v_{i,G+1}$。差别进化的交叉操作定义如下：

$$u_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \ldots, u_{Di,G+1}) \tag{3}$$

其中各分量由下式决定：

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1} & rand(j) \le CR \ or \ \ j = rn(i) \\ x_{ji,G+1} & rand(j) > CR \ and \ \ j \ne rn(i) \end{cases} \quad j = 1,2,\ldots,D \tag{4}$$

式中 *rand*(*j*)为位于[0,1]的随机数，*CR* 为位于[0,1]的交叉常数，*rn*(*i*)为位于[1,*D*]的随机整数。

2.2 差分进化算子变异

本文提出的变异算子是针对微粒的位置 x_i 的，其第 d 维变异定义为下式：

IF (*rand*()<*CR* or d = = k)　THEN

$$Trail_{id} = x_{id} + F \cdot (p_{1,d} + p_{2,d} - p_{3,d} - p_{4,d}) \tag{5}$$

其中 k 是[1,*D*]之间的一个随机数，以保证 x_i 至少有一维产生变异；*rand*()是[0,1]之间的一个随机数；*CR* 为自定义的变异概率，F 是位于[0,2]的一个常数。$P_{1,d}P_{2,d}P_{3,d}P_{4,d}$ 是从 p_i 中随机选取的值。

根据微粒位置变异向量 *Trail* 可计算相应的适应值，只有当由 $Trail_i$ 得到的适应值优于变异前 x_i 的适应值时，x_i 才被 $Trail_i$ 取代，否则仍保留 x_i。此种情况可视为变异不成功，下一步将继续对 x_i 产生新的变异，直到变异成功，使微粒跳出目前的局部值。

一般的微粒群算法在运行前期收敛较快，因此，在算法运行初期，差分变异不起作用，只有在微粒群发现的全局最大适应值 P_g 连续 G 代没有改善时，才对微粒的位置引入差分变异操作。为了避免采用的变异操作影响算法的稳定性，必须合理地选择代数 G。若 G 太小，就有可能影响有潜力微粒的飞行，使得算法变为类似于随机搜索的算法；若 G 太大，将使微粒群处于停滞状态过久，消弱引入差分算子变异的作用。根据作者对 Rosenbrock、Rastrigrin 和 Griewank 等高维、多峰测试函数的实验经验，G 取值在 10～20 比较合适，本实验均采用 G＝15。另外，本文实验中的变异率 CR 取为 0.1，F 取为 0.6。

2.3 改进算法的流程

本文提出的改进微粒群算法的基本流程如下：

Step1：随机初始化微粒的位置与速度。

Step2：将微粒的 *pbest* 设置为当前位置，*gbest* 设置为初始群体中最佳微粒的位置。

Step3 :判断算法收敛准则(群体最大适应度满足条件或超过最大迭代次数)是否满足，如果满足，转向 Step9；否则，执行 Step4。

Step4：对于所有微粒，执行如下操作：

(1) 根据式(1)更新微粒的位置与速度；

(2) 如果微粒适应度优于 *pbest* 的适应度，*pbest* 设置为新位置；

(3) 如果微粒适应度优于 *gbest* 的适应度，*gbest* 设置为新位置；

Step5：根据连续不变化次数 G 判断微粒是否处于停滞状态，是则执行 Step6，否则，执行 Step8

Step6：按变异概率，根据式(5)进行变异得 $Trail_i$。

Step7：若 $Trail_i$ 的适应度优于 x_i，则将 x_i 取代，否则返回 Step6 继续变异，直到成功。

Step8：判断算法收敛准则是否满足，如果满足，执行 Step9，否则，转向 Step4。

Step9：输出 *gbest*，算法运行结束。

从上述流程可以看出，差分进化算子变异的微粒群优化算法实际上是在微粒群优化算法的基本框架中增加了差分变异算子，通过对 x_i 的变异操作来提高微粒群优化算法跳出局部最优解的能力。

3 性能测试

本文拟用 4 个常用的测试函数对改进 PSO 算法(MPSO)的寻优性能进行测试，并与标准的 PSO 算法(SPSO)作比较。DeJong 函数是一个单峰二次函数，它的全局极小点在 x^*=(0,0,…,0)处，全局极小值 $f(x^*)$=0。

$$f_1(x) = \sum_{i=1}^{10} x_i^2, \qquad -10 \le x_i \le 10$$

Schaffer F6 函数是一个具有强烈振荡的多峰函数，具有一个全局极大点 x^*=(0,0,…,0)处，其函数值 $f(x^*)$=1。

$$f_2(x) = 0.5 - \frac{\left(\sin\sqrt{x_1^2 + x_2^2}\right)^2 - 0.5}{\left(1 + 0.001 \cdot (x_1^2 + x_2^2)\right)^2}, \quad -10 \le x_1, x_2 \le 10$$

Rastrigin 函数是一个多峰函数，有很多正弦凸起的局部极小点，它的全局极小点在 x^*=(0,0,…,0)处，全局极小值 $f(x^*)$=0。

$$f_3(x) = \sum_{i=1}^{10} \left(\left(x_i^2 - 10\cos(2\pi x_i) \right) + 10 \right), \quad -10 \le x_i \le 10$$

Griewank 函数也是一个多峰函数，它的全局极小点在 x^*=(0,0,…,0)处，全局极小值 $f(x^*)$=0。

$$f_4(x) = \frac{1}{4000}\sum_{i=1}^{10} x_i^2 - \prod_{i=1}^{10} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad -600 \le x_i \le 600$$

由于在群体规模较小时，PSO 算法比较容易出现停滞现象，为了更好地验证改进算法的有效性，选取种群微粒数为 20；速度上限 v_{max} 取各函数初始范围的上限，c_1=c_2=2；惯性权重采用时变权重 0.9~0.2，递减率为：变化范围/1000。为了全面比较优化效果，测试以两种方式进行：(1)迭代 3 000 次，

比较寻优率和平均适应函数值；(2)给定寻优应达到的精度和最大迭代次数(20 000 次)，比较平均进化迭代次数和寻优率。由于 SPSO 和 MPSO 都是随机搜索算法，单凭一两次搜索效果很难说明算法的有效性，本文从概率的角度来对比算法的有效性。优化时每种算法对每一个测试函数分别进行 200 次独立实验，统计实验结果汇总如表 1 所示。

表 1 在给定进化代数下的寻优率和平均适应函数对比

试函数	最优值	精度	迭代 3 000 次				迭代 20 000 次			
			MPSO		SPSO		MPSO		SPSO	
			寻优率	平均值	寻优率	平均值	平均步数	寻优率	平均步数	寻优率
DeJong	0	10^{-5}	100%	0	100%	0	2 870.5	100%	2 959.1	100%
Schaffer f6	1	10^{-5}	100%	1	85%	0.998 32	3 706.3	100%	4 140.8	100%
Rastrigrin	0	1	59%	3.971 0	6%	5.015 14	6 538.5	91.5%	16 062.3	29%
Griewank	0	10^{-1}	93.5%	0.025 37	76%	0.082 62	4 257.2	100%	7 618.4	89%

从表 1 可以看出，对于 $f_1(x)$函数，标准 PSO 和改进 PSO 算法均能取得很好的优化效果，而对于函数 $f_2(x)$，$f_3(x)$和 $f_4(x)$，SPSO 算法在优化过程中陷入了局部最优，给定迭代次数 3 000 次，SPSO 的寻优率较低。对于 Rastrigin 函数，寻优率只有 6%，而改进算法的寻优率则明显提高，达到 59%，这源于差分进化算子的变异使得陷入停滞的微粒能够有效的跳出局优点，具有强大的全局搜索能力，能有效地避免早熟收敛问题。在给定寻优精度下，对于设定的最大进化次数 20 000 次，SPSO 算法所需的平均进化代数也很多，对于 Rastrigin 函数寻优率仍然很低，而改进 PSO 算法寻优所需的平均进化步数明显减少，寻优率也得到很大改进。

由以上结果可以看出，改进的 PSO 算法在提高微粒的多样性的同时保证搜索精度，另外由于改进算法首先判断最优值是否连续 G 代不变化，对运算量的增加很少。相反由于能及时判断是否已经收敛于局部最优点，并迅速摆脱它的束缚，提高了搜索成功率。

4 结束语

作为群智能的代表性方法之一，PSO 算法实现简易，效果良好，是一种新颖的智能计算优化方法。

本文针对微粒群算法的早熟停滞问题，提出了一种基于差分进化算子变异的微粒群优化算法，其运算量比标准 PSO 算法略有增加，但寻优性能大大提高，能有效避免早熟收敛问题，是一种非常实用的优化算法。

参考文献

1 Kennedy J, Eberhart R C. Particle Swarm Optimization [C]. Proc. of IEEE Int. Conf. on Neural Networks, Perth, WA, Australia, 1995: 1942-1948.

2 Kennedy J. The Particle Swarm: Social Adaptation of Knowledge[C]. Proc. of Int. Conf. Evolutionary Computation, Indianapolis, IN, 1997-04: 303-308.

3 Frans V D B. An Analysis of Particle Swarm Optimizers[D]. South Africa: Department of Computer Science, University of Pretoria, 2002.

4 Zhang W, Xie X. DEPSO: Hybrid Particle Swarm with Differen tial Evolution Operator[C]. Proceedings of IEEE International Conference on Systems, Man and Cybernetics, 2003: 3816-3821.

5 Storn R, Price K. Differential Evolution—A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces[R]. International Computer Science Institute, Berkley, 1995.

（上接第 9 页）

另一种想法是分别投票方案，即选民为每个候选人都写张选票，共写 M 张选票，选票总数为 N 票。

经过对这两个简单方案进行改进，文献[4]提出了两个比较高效的可分选举方案。

(1)提出了可分多集的概念，即对任意正整数 M（≥3）和 N，我们可以构建集合 X 为（M, N）可分多集，对所有满足 $\sum_{i=1}^{M} x_i = N$ 的非负整数(x1,x2,…,xM)，都存在多集 $X_1,...,X_M \subseteq X$，使得 $(\bigcup_{1\le i\le M} X_i = X) \wedge (\forall i \in \{1,...,M\} : \sum_{a\in X_i} a = x_i)$。可分多集的概念保证了选民可以将选票集合任意拆分成多个子集后投给相应的候选人，因此重复选举方案中的重复次数就可从 N 次减少为子集的个数，选民的投票成本下降为 O(M2log(N))。

(2)分别投票方案的改进。选民投给候选人 j 的选票表示为 $V_j = \sum_{l=0}^{t} b_{j,l} L^l$，其中 $L \le N+1, b_{j,l} \in \{0,1,...,L-1\}, t = \lfloor \log_L N \rfloor$。T 和 L 是公开的，$b_{j,l}$是保密的。投票时对每张选票 V_j 的加密就变成了对 $b_{j,l}$ 的加密，虽然增加了加密成本，但由于分别投票方案中对每张选票都必须证明 V_j 在 0 到 N 之间，而改进后只需要证明 $b_{j,l}$ 在 0 到 L-1 之间，因此减少了零知识证明成本，该方案的投票成本由此下降为 O(Mlog(N))。

5 结论

电子选举有着明确的安全需求，成熟的协议模型和日趋多样的选票形式。国内外政府和研究机构对其进行了广泛的探讨与研究，并提出了一系列的解决方案。但目前大多数的方案仍然存在如下问题：

(1)依赖完全可信第 3 方：第 3 方可能串通泄漏中间结果；
(2)选民有收据：收据可能被用于买卖选票和胁迫投票；
(3)选票形式单一：无法在一个界面下实现多种选票形式的统一；
(4)效率较低：不适用于大规模选举。

针对这些问题，对安全电子选举的研究已经从初期对选举系统基本安全性质的考虑，转向研究如何实现高效率、安全性和灵活性相结合的电子选举协议，特别是在如何解决无收据性和公开可校验性的矛盾，实现高效自计票方案，融合主流方案的不同技术和拓展选票形式的深度上挖掘题材。虽然目前研究人员已经提出了不少电子选举方案，如视觉密码方案、自计票选举方案、矢量投票方案和可分选举方案，但这些方案在实用性和安全性上仍有待进一步地发展和完善。

参考文献

1 Chaum D. Secret-ballot Receipts: True Voter-verifiable Elections[J]. IEEE Security & Privacy, 2004, 2(1): 38.

2 Kiayias A, Yung M. Self-tallying Elections and Perfect Ballot Secrecy[C]. Proceedings of International Workshop on Practice and Theory in Public-key Cryptography, 2002: 141.

3 Kiayias A, Yung M. The Vector-ballot E-voting Approach[C]. Proc. of Financial Cryptography, 2004: 72.

4 Ishida N, Matsuo S, Ogata W. Divisible Voting Scheme[C]. Proc. of the 6th Information Security Conference, 2004: 137.