文章编号: 1001-0920(2011)01-0054-05

基于强化学习的适应性微粒群算法

邢长明, 刘方爱

(山东师范大学 信息科学与工程学院，济南 250014)

摘 要: 惯性权重是微粒群算法(PSO)的重要参数, 它可以平衡算法的全局和局部搜索能力的关系, 改善算法的性能. 对此, 提出一种基于强化学习的适应性微粒群算法(RPSO). 首先将不同惯性权重调整策略视为粒子的行动集合; 然后通过计算 Q 函数值, 考察粒子多步进化的效果; 进而选择粒子最优进化策略, 动态调整惯性权重, 以增强算法寻找全局最优的能力. 对几种经典函数的测试结果表明, RPSO 能够获得良好的性能, 特别是对多峰函数效果更加明显.

关键词: 微粒群算法；惯性权重；自适应；强化学习

中图分类号: TP18 **文献标识码:** A

An adaptive particle swarm optimization based on reinforcement learning

XING Chang-ming, *LIU Fang-ai*

(School of Information Science and Engineering，Shandong Normal University，Ji'nan 250014，China．Correspondent: XING Chang-ming，E-mail：xingchm@tom.com)

Abstract: The inertia weight is an important parameter of the particle swarm optimization(PSO), which can improve the algorithm's performance through balance the global search ability and the local search ability. Therefore, an algorithm based on PSO based on reinforcement learning(RPSO) is proposed, in which several different adjusting methods on inertia weight are used and each is mapped into an action. For considering multi-step evolutionary performance, reinforcement learning method is introduced. According to the calculated Q value, each particle selects the optimal evolutionary strategy, and then changes the inertia weight dynamically. The experimental results show that RPSO is better than the existing algorithms, especially for multimodal function.

Key words: particle swarm optimization algorithm；inertia weight；self-adaptive；reinforcement learning

1 引 言

微粒群算法(PSO)是Kennedy等人[1]提出的一种全局优化进化算法, 它源于对鸟群觅食行为的模拟. PSO算法作为一种全局优化方法, 其算法的搜索能力较好, 收敛速度较快, 但存在早熟收敛的问题. 在PSO中可以通过调节惯性权重改善算法的性能, 学术界对此已做了大量工作, 如: 文献[2-3]分别提出了随着进化代数的增加, 线性减小和非线性减小惯性权重的方法; [4]提出了用模糊控制器来自适应改变惯性权重的算法, 但该算法需建立模糊规则, 实现比较复杂; [5]通过实验表明, 对于多峰函数, 随机惯性权重策略能在一定程度上避免算法早熟; [6]提出了一种基于粒子相似度动态调整惯性权重的方法; [7]则利用进化过程中粒子适应度的差异来评价种群的早熟收敛程度, 然后动态地改变惯性权重, 取得了较好的效果.

在粒子进化过程中惯性权重影响粒子的速度, 依据不同的惯性权重调整策略可以产生多个可供选择的速度. 上述PSO改进算法采用单一的惯性权重调整方法, 使粒子进化缺乏灵活性, 而且这些调整方法并非在每一步中都最有效. 如果将多种惯性权重调整方法同时应用于粒子进化, 并依据粒子多步进化的效果选择其中一种方法调整粒子的飞行速度(即通过向前考察几步粒子的进化效果, 并将信息向后反馈, 影响粒子速度的选择), 则对粒子进化更加有利. 基于这种思想, 同时借鉴文献[8]的思路, 本文提出一种新的适应性微粒群算法(RPSO), 通过对几种经典函数的测试表明, RPSO能够获得良好的性能, 特别是对多峰函数效果更加明显.

收稿日期: 2009-09-22；**修回日期:** 2009-12-02.

基金项目: 国家自然科学基金项目(60373063, 90612003)；山东省自然科学基金项目(Y2007G11).

作者简介: 邢长明(1983−), 男, 博士生, 从事智能计算、网格计算等研究；刘方爱(1962−), 男, 教授, 博士生导师, 从事并行处理、互联网络等研究.

2　惯性权重调整策略

在微粒群算法每一次迭代中, 每个粒子通过跟踪两个“极值”(个体极值 $p_{\rm best}$ 和全局极值 $g_{\rm best}$) 来更新自己. 即在找到两个极值后, 粒子通过如下两个公式更新自身的速度和位置:

$$v_{k+1} = w_k v_k + c_1 r_1 (p_{{\rm best}_k} - x_k) + c_2 r_2 (g_{{\rm best}_k} - x_k), \tag{1}$$

$$x_{k+1} = x_k + v_k. \tag{2}$$

为了改善PSO算法的性能, 文献 [2] 提出了线性递减的惯性权重调整策略, 即

$$w_k = w_{\rm ini} - \frac{w_{\rm ini} - w_{\rm end}}{k_{\max}} \times k. \tag{3}$$

这种选取策略对某些优化问题会得到很好的效果, 但对于不同的问题, 算法中每一代所需要的比例关系并不相同, 因此, 文献 [3,9] 提出了惯性权重的如下几种调整策略:

1) 惯性权重线性微分递减策略

$$w_k = w_{\rm ini} - \frac{w_{\rm ini} - w_{\rm end}}{k_{\max}^2} \times k^2; \tag{4}$$

2) 惯性权重非线性微分变化策略

$$w_k = w_{\rm ini} + \frac{w_{\rm ini} - w_{\rm end}}{k_{\max}} \times k - \frac{w_{\rm ini} - w_{\rm end}}{k_{\max}^2} \times k^2; \tag{5}$$

3) 步长较小的线性递减策略

$$w_k = w_{\rm ini} - \frac{w_{\rm ini} - w_{\rm end}}{k_{\max}^2} \times k; \tag{6}$$

4) 步长较小的非线性递减策略

$$w_k = w_{\rm end} + \frac{(k_{\max} - k)^m}{k_{\max}^m} \times (w_{\rm ini} - w_{\rm end}). \tag{7}$$

文献 [3,9] 中的大量实验表明, 对于不同的测试函数, 上述调整策略各有优势. 然而, 对于一个未知问题, 应该选择哪种调整策略更为合适? 在一个算法中综合运用这些调整策略是否会产生更好的效果? 为此, 将各种不同的调整策略视为行动, 在粒子进化的每一步选择一个合适的行动控制粒子的速度. 另外, 在做出决策之前, 如果能够多向前考虑几步, 则做出正确决定的可能性会更大. 因此, 在粒子选择行动时, 可通过多步进化的效果评价各种行动的好坏, 增加粒子的寻优能力.

3　适应性微粒群算法

在任何一个粒子进化时, 如何选择一个最合适的调整策略很关键. 强化学习[10]通过选择最大化代理带折扣的累积收益的行动, 可以逐步学习得到最优行动策略. 在PSO中, 若将惯性权重调整策略看成行动, 则粒子选择最优调整策略便转化为代理选择最优行动.

基于带折扣方式计算代理累积收益, 并选择最大化累积收益对应行动的思想, 就是在选择行动前考虑了行动的立即效果及多步滞后效果[11]. 另外, 在粒子选择速度时, 上述调整策略仅考虑了粒子单步进化的效果. 如果通过考察粒子多步进化的效果来选择粒子的速度, 则粒子的寻优能力应该更强.

3.1　Q 学习算法

Q 学习是一种模型无关的强化学习算法, 通过代理多步学习求出最大折扣回报的行动, 得到最优行动策略. 其基本形式为

$$Q(s,a) = r(s,a) + \gamma \max_{a'} Q(s', a').$$

其中: s 和 s' 为代理的两个状态; a 和 a' 为代理的两个行动; $r(s,a)$ 为代理在状态 s 选择行动 a 后的立即收益; $\gamma (0 \leqslant \gamma < 1)$ 为折扣因子; a' 为代理下一个状态 s' 可选择的行动, 显然 $\max\limits_{a'} Q(s', a')$ 表示代理在下一状态 s' 选择不同行动所能得到的最大回报值.

Q 学习通过估计最优 Q 值来得到最优策略 π^*. 记最优 Q 值为 $Q^*(s,a)$, 若 $a^* = \arg\max\limits_{a \in A} Q^*(s,a)$, 则 $\pi^*(s) = a^*$, 其中 A 为代理可选行动策略的集合.

假设代理在任何状态下可选择的行动集合均为 $A = \{a_1, a_2, \cdots, a_n\}$, 即代理在每个状态都有 n 个可供选择的行动. 如果向前看 $m+1$ 步，代理开始选择的行动为 a, 则在忽略状态参数的情况下, 计算代理选择行动 a 时的收益为

$$Q(a) = r(a) + \gamma Q(a^{(1)}) + \gamma^2 Q(a^{(2)}) + \cdots + \gamma^m Q(a^{(m)}). \tag{8}$$

其中: $a, a^{(i)} \in A$, 且 $1 \leqslant i \leqslant m$, 参数 m 控制计算 Q 值向前看的步数.

3.2　RPSO 算法的原理

借助上述 Q 学习算法, 将粒子视为代理, 各种惯性权重调整策略视为代理行动的集合, 则可以实现微粒群算法和 Q 学习算法之间的映射.

对于微粒群中的任一粒子, 向前看 $m+1$ 步, 可以通过式 (8) 计算粒子选择行动 a 时的收益 $Q(a)$. 定义个体立即收益 $r(a) = f_p(a) - f_o(a)$. 其中: $f_p(a)$ 为父代粒子对应的函数值, $f_o(a)$ 为选择行动 a 后生成的子代粒子对应的函数值. 如果粒子可选行动的个数为 n, 则经过 m 次进化后将产生 n^m 个粒子. 要计算粒子最优进化策略, 需要进行指数级的计算, 计算量太大. 为此采用如下方法: 粒子第 1 次进化后产生 n 个新粒子; 这 n 个新粒子再次进化, 每个新粒子仍产生 n 个后代. 首先采用 Boltzmann 分布, 根据下式计算每个后代被保留下来的概率:

$$p(a_i) = {\rm e}^{r(a_i)/T} \Big/ \sum_{i=1}^{n} {\rm e}^{r(a_i)/T}; \tag{9}$$

然后从 n 个后代中选择一个保留下来. 被保留的后代再采用同样的方式产生 n 个后代, 并保留其中的一

个后代. 通过这种简化后, 一个粒子经一次进化后产生n个后代, 每个后代再次进化, 只保留一个新的后代. 这样, 设m为向前看的步数, 经m次进化后, 一个粒子共产生的新粒子个数为$n+(m-1)n^2$, 用于计算Q值需保留的个体数为$m\times n$, 从而使算法的时间复杂度降为多项式级.

文献[8,11]对基于Q学习的进化规划算法(QEP)的收敛性进行了证明. RPSO的证明过程与此类似, 限于篇幅, 本文不再对其收敛性进行证明. RPSO算法的收敛性表明: 在粒子进化的初始阶段, 粒子在进化策略空间进行探索, 随着进化步数的增加, 粒子探索的成分逐步减少, 越来越趋向于选择适应度函数大的进化策略, 此时粒子选择最优变异策略的概率逐步增加.

3.3 RPSO算法设计

RPSO算法利用多步Q学习搜索粒子最优进化策略, 具体算法设计如下:

1) 设定参数. 包括粒子个数N; 惯性权重的取值范围$\omega_{\rm ini},\omega_{\rm end}$; 学习因子$c_1,c_2$; 计算$Q$所需要粒子进化的步数$m$; 折扣因子$\gamma$; 算法迭代次数.

2) 对每个粒子初始化. 随机产生N个初始解和N个初始速度, 设定各粒子的个体极值$p_{\rm best}$为初始解.

3) 根据当前位置和速度产生各个粒子的新位置.

4) 计算每个粒子新位置的适应度值. 对于各个粒子, 若粒子的适应度优于原来的个体极值$p_{\rm best}$, 则设置当前值为个体极值$p_{\rm best}$.

5) 根据各个粒子的个体极值$p_{\rm best}$找出全局极值$g_{\rm best}$.

6) 利用Q学习算法得到下一步进化的ω值, 即选择如下最优进化策略:

①对于每个粒子, 利用给定的n个行动产生n个新后代, 并设定$t=1$;

②Do while $t<m$, 每个后代粒子利用给定的n个行动产生n个后代粒子, 并利用式(9)从中保留一个, 令$t=t+1$;

③计算每个进化策略对应的Q, 选择使Q最大化的行动所对应的ω值为当前ω值, 同时忽略其他$n-1$个ω值.

7) 每个粒子按式(1)更新自己的速度, 并将其限制在$v_{\max}$内.

8) 每个粒子按式(2)更新当前的位置.

9) 判断是否满足终止条件. 若是, 则算法结束; 否则, 转步4)继续执行.

另外, 微粒群算法中, 粒子的速度更新公式(1)中包含$p_{\rm best}$和$g_{\rm best}$两个参数; 在RPSO算法的粒子Q学习过程中, 两个参数分别记为$qp_{\rm best}$和$qg_{\rm best}$. 其更新规则如下:

1) 任一粒子的初始$qp_{\rm best}$和$qg_{\rm best}$都等于第1步Q学习时粒子自身的位置;

2) 根据粒子行动集合的大小n, 可形成n种不同的行动策略, 每个策略的$qp_{\rm best}$为该策略的最优适应度所对应的位置, $qg_{\rm best}$为所有策略的最优适应度所对应的位置.

4 算法有效性验证

4.1 测试函数

为了评价本文提出的RPSO算法的性能, 用表1中5个标准测试函数[12]进行一系列实验. 这5个标准函数具有不同的特点, 可以充分考察算法对不同类型问题的优化性能.

表1 测试函数

函数	x_i
$f_1=\sum_{i=1}^{n}x_i^2$	[-100, 100]
$f_2=\sum_{i=1}^{n-1}(100(x_{i+1}-x_i^2)^2+(x_i-1)^2)$	[-2.048, 2.048]
$f_3=\sum_{i=1}^{n}[x_i^2-10\cos(2\pi x_i)+10]$	[-5.12, 5.12]
$f_4=\frac{1}{4\,000}\sum_{i=1}^{n}x_i^2-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$	[-600, 600]
$f_5=-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right)-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right)+20+{\rm e}$	[-32, 32]

f_1和f_2是连续的单峰函数, 其中f_2是一个经典的复杂优化问题, 取值区间内走势平坦, 为算法提供少量信息, 要收敛到全局最优点的机会微乎其微, 常用于测试算法收敛速度. $f_3\sim f_5$是复杂的非线性多峰函数, 存在大量局部极值, 可以检验算法的全局搜索性能和逃离局部极值、避免早熟收敛的能力.

4.2 参数设置

实验中各算法群体规模size$=20$, 学习因子$c_1=c_2=1.496\,18$, 微粒搜索位置范围为表1中x_i的区间, 函数维数$\dim=10$, 最大迭代次数$\max_{\rm gen}=3\,000$. SPSO的惯性权重$\omega=0.72$, 其他算法的惯性权重$\omega_{\rm ini}=0.9$, $\omega_{\rm end}=0.4$.

本文选择$\omega=0.72$和式(3), (4), (7)四种惯性权重调整策略作为RPSO算法的行动集合, 折扣因子$\gamma=0.5$, $m=3$.

4.3 实验结果及分析

4.3.1 各算法的性能比较

将RPSO与标准PSO(SPSO), 线性递减惯性权重

PSO(LPSO), 线性微分递减惯性权重PSO(DPSO)和步长较小的非线性递减惯性权重PSO(NPSO)进行对比, 所有实验数据均源于算法独立运行30次的平均抽样. 表2详细记录了这些算法30次寻优结果的平均值(MB)及标准方差(SD).

从表2可以看出, 对于单峰函数优化问题, SPSO算法采用经验参数$\omega = 0.72$时的性能较好; 而对于多峰函数优化问题, 各类PSO变种算法的性能均优于SPSO算法. 采用上述参数设置, 基于强化学习的RPSO算法对函数f_1, f_3, f_4, f_5的测试结果好于其他4种PSO算法; 然而对于函数f_2的测试结果, RPSO算法未能优于SPSO.

通过大量模拟实验发现, 对于函数f_2的优化, SPSO算法的惯性重权参数$0.6 < \omega < 0.75$时性能较好, 当改变其惯性权重参数为其他值时性能变差. 改变RPSO算法的参数, 设置$\omega_{\text{ini}} = 0.8$, $\omega_{\text{end}} = 0.6$时, 对于测试函数$f_2$, 其$\text{MB} = 0.51$, $\text{SD} = 0.77$, 此时RPSO算法的性能好于SPSO.

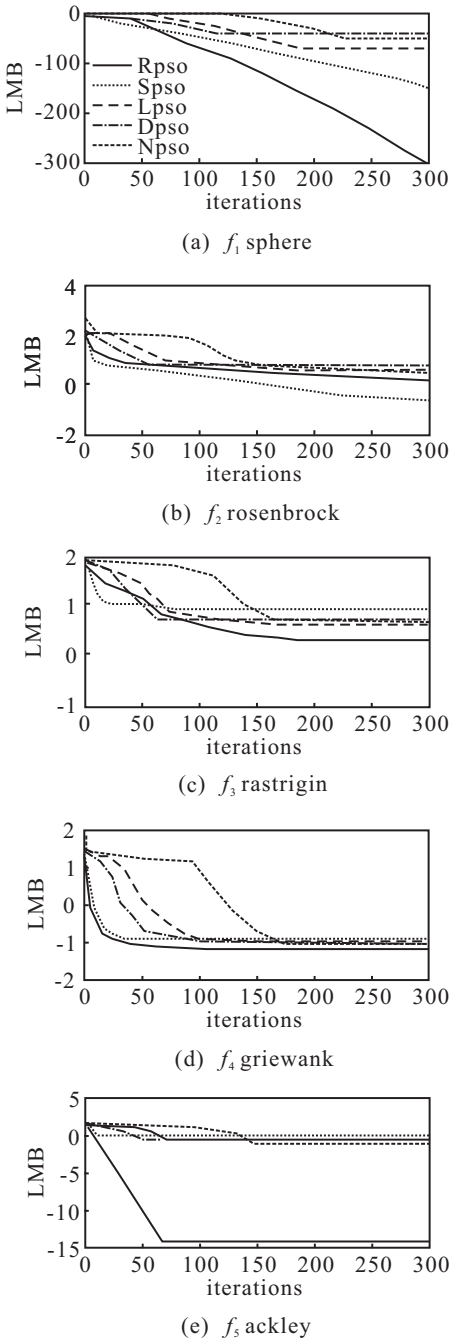
综合实验结果和上述分析可以看出, 基于强化学习的适应性微粒群算法优于其他PSO变种算法, 同时RPSO也是"做决策时向前多看几步, 决策的正确性就会增加"这一经验的一个佐证.

表 2　RPSO 与其他算法的性能比较 (MD, SD)

函数	算法				
	RPSO	SPSO	LPSO	DPSO	NPSO
f_1	1.60E-261	9.16E-128	2.44E-45	1.57E-28	4.08E-33
	0	2.83E-127	1.33E-44	8.57E-28	2.23E-32
f_2	1.59	0.73	3.82	4.11	4.22
	0.54	1.51	3.88	1.06	1.09
f_3	2.75	7.26	4.41	4.28	5.34
	2.02	3.17	4.36	2.29	2.76
f_4	7.58E-02	9.58E-02	9.45E-02	0.09	7.79E-02
	0.04	5.21E-02	8.40E-02	0.05	0.04
f_5	3.76E-15	0.30	5.06E-15	0.05	0.17
	9.01E-16	0.56	4.98E-15	0.30	0.45

仍采用第4.2节的参数设置, 比较不同算法的动态性能. 图1显示了函数$f_1 \sim f_5$采用RPSO, SPSO, LPSO, DPSO, NPSO算法求解运行30次后得到的平均最佳适应度进化曲线. 为了便于比较, 图1的横坐标为算法每迭代10次保留一次最优值, 纵坐标利用适应度的对数值(LMB)表示.

由图1(a)和图1(b)可见, 对于单峰函数, RPSO运行性能优于其他3种PSO变种算法, 不仅具有较快的搜索速度, 而且其最优解的收敛精度较高. 与固定参数的SPSO算法相比, 对于函数f_1优化, RPSO明显优于SPSO; 但对于函数f_2优化, RPSO未能达到SPSO



(a) f_1 sphere

(b) f_2 rosenbrock

(c) f_3 rastrigin

(d) f_4 griewank

(e) f_5 ackley

图 1　RPSO 与其他算法的动态性能比较

的收敛精度, 这可能与其参数设置有关(将在下一步工作中对其进行深入分析).

对于多峰函数优化, 观察图1(c)和图1(d), RPSO算法的性能明显优于其他PSO算法. 所有算法在迭代初期种群微粒运行良好; 但在迭代后期(200代之后), SPSO, LPSO, DPSO, NPSO算法很容易失去种群的多样性而收敛于某一局部极值点, 且没有能力改进群体最佳适应度值. 根据图1(c)和图1(d)的最优适应度变化曲线可以看到, RPSO在函数优化上不会发生类似现象, 该算法在500代之内全局搜索速度较快, 而且能够得到较高的搜索精度. 这主要是由于RPSO算法采用Q学习, 每次粒子进化时通过向前多看几步, 便能以较大的概率选择最好的进化行动. 500代之

后, 即使RPSO也将陷入局部极值, 但经过多次迭代后便能逃离局部极值, 逐渐改进最优解的质量. 例如, 对于函数局部最优点与全局最优点距离较远的Ackley函数[12], RPSO算法能在600代之内收敛于精度较高的局部最优解3.996 8 E-15, 但经过几十步迭代后仍能跳出该局部极值, 收敛于精度更高的局部最优解4.440 86 E-16.

4.3.2 不同γ下RPSO算法性能比较

参数设置直接影响算法的性能, 为此仍采用第4.2节的参数设置, 测试折扣因子γ分别取值为0.1, 0.5, 0.8时, RPSO算法性能的实验结果如表3所示.

表 3 不同γ时RPSO的性能比较(MD, SD)

函数	γ		
	0.1	0.5	0.8
f_1	2.66E-299 0	1.60E-261 0	5.91E-193 0
f_2	1.49 0.67	1.59 0.54	1.67 0.60
f_3	3.35 2.50	2.75 2.02	2.75 2.35
f_4	7.80E-02 0.03	7.58E-02 0.04	6.49E-02 0.03
f_5	3.88E-15 6.49E-16	3.76E-15 9.01E-16	3.76E-15 9.01E-16

折扣因子γ的取值直接影响计算Q值, 随着γ取值的变大, 滞后收益的作用将增强. 由表3可见, 对于单峰函数, 随着折扣因子γ取值的减小, RPSO算法性能不断增强; 然而对于多峰函数, 随着折扣因子γ取值的增大, RPSO算法性能也有所增强. 分析其原因, 针对单峰函数, 可能与γ取值范围的设置有关. 因为单峰函数的经验惯性权重值为0.729 84, 即当γ取其他值时, PSO算法的性能变差, 而本文RPSO算法中的3种粒子行动策略(式(3), (5), (9))都不具有比SPSO($w = 0.72$)更好的性能. 在Q学习过程中, γ值增大, 滞后收益的作用增强, 学习到其他3种粒子行动策略的可能性也增大, 因此算法的性能变差. 针对多峰函数, 3种粒子行动策略(式(3), (4), (7))均具有比SPSO($w = 0.72$)更好的性能, 因此，其原因正好与单峰函数相反.

5 结 论

本文提出了一种适应性的微粒群算法, 该算法将不同惯性权重调整策略看成粒子的行动集合, 考察粒子多步进化的效果, 通过Q学习计算粒子的立即收益和折扣收益, 选择最优行动, 增强算法寻找全局最优的能力. 现有的任何PSO变种算法均可视为RPSO算法中粒子的一种行动, 因此RPSO还可以作为融合现有PSO变种算法的一个框架. 文献[8,11]的相关理论可以证明, 该算法以概率1收敛到全局最优解. 相关的实验结果表明, 与其他PSO改进算法相比, RPSO具有更好的函数优化性能.

需要说明的是, 由于RPSO算法计算Q值时需要产生多代新粒子, 增加了算法的计算量, 这是该算法的不足之处. 为了进一步改善RPSO的性能, 下一步工作将扩大算法的测试范围, 寻找更好的粒子行动策略以及探讨RPSO算法新增参数的设定问题.

参考文献(References)

[1] Kennedy J, Eberhart R C. Particle swarm optimization[C]. Proc of the IEEE Int Conf on Neural Network. Perth: IEEE Inc, 1995: 1942-1948.

[2] Shi Y, Eberhart R C. A modified particle swarm optimizer[C]. IEEE World Conf on Computational Intelligence. Piscataway: IEEE Press, 1998: 69-73.

[3] 胡建秀, 曾建潮. 微粒群算法中惯性权重的调整策略[J]. 计算机工程, 2007, 33(11): 192-195.
(Hu J X, Zeng J C. Selection on inertia weight of particle swarm optimization[J]. Computer Engineering, 2007, 33(11): 192-195.)

[4] Shi Y, Eberhart R C. Fuzzy adaptive particle swarm optimization[C]. Proc of the IEEE Conf on Evolutionary Computation. Piscataway: IEEE Press, 2001: 101-106.

[5] Zhang L P, Yu H J, Hu S X. A new approach to improve particle swarm optimization[C]. Lecture Notesin Computer Science. Chicago: Springer-Verlag, 2003: 134-139.

[6] 刘建华, 樊晓平, 瞿志华. 一种基于相似度的新型粒子群算法[J]. 控制与决策, 2007, 22(10): 1155-1159.
(Liu J H, Fan X P, Qu Z H. A new particle swarm optimization algorithm based on similarity[J]. Control and Decision, 2007, 22(10): 1155-1159.)

[7] 韩江洪, 李正荣, 魏振春. 一种自适应粒子群优化算法及其仿真研究[J]. 系统仿真学报, 2006, 18(10): 2969-2971.
(Han J H, Li Z R, Wei Z C. Adaptive particle swarm optimization algorithm and simulation[J]. J of System Simulation, 2006, 18(10): 2969-2971.)

[8] Zhang H X, Lu J. Adaptive evolutionary programming based on reinforcement learning[J]. Information Sciences, 2008, 178(4): 971-984.

[9] Chatterjee A, Siarry P. Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization[J]. Computers and Operations Research, 2006, 33(3): 859-871.

[10] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. Cambridge: MIT Press, 1998.

(下转第64页)