文章编号: 1001-0920(2009)06-0864-05

一种基于栅格的动态粒子数微粒群算法

刘 勇, 梁 彦, 潘 泉, 程咏梅
(西北工业大学 自动化学院, 西安 710072)

摘 要: 微粒群算法的全局搜索性能容易受到局部极值点的影响. 对此, 提出一种基于栅格的动态粒子数微粒群算法 (GB-DPPPSO). 通过设计栅格信息更新策略、粒子产生策略和粒子消灭策略, 可以根据种群搜索情况动态控制粒子数变化, 以保持种群多样性, 提高全局搜索性能. 通过对 4 个典型数学验证函数的仿真实验, 表明了该算法相对于 DPPPSO 在全局搜索成功率和搜索效率两方面均有明显改进.

关键词: 微粒群算法; 栅格; 动态粒子数
中图分类号: TP391 **文献标识码**: A

Grid based dynamic particle population particle swarm optimization

LIU Yong, LIANG Yan, PAN Quan, CHENG Yong-mei
(College of Automation, Northwestern Polytechnical University, Xi'an 710072, China. Correspondent: LIU Yong, E-mail: dudu2077@mail.nwpu.edu.cn)

Abstract: Particle swarm optimization (PSO) is easy to be trapped by the local optimum. Therefore, the grid based dynamic particle population PSO (GB-DPPPSO) is proposed. GB-DPPPSO has three strategies, grid information update strategy, particle generalization strategy and particle vanishing strategy, which keep the diversity of swarm through convergence and enhance the global searching ability. Simulation tests on four benchmark functions prove that the method performs better than DPPPSO on global successful searching probability and searching efficiency.

Key words: Particle swarm optimization; Grid; Dynamic particle population

1 引 言

微粒群优化算法 (PSO)[1] 是由 Kennedy 和 Eberhart 于 1995 年提出的. 该算法收敛速度快、计算量低、易于实现、需要调整的参数少, 因此在众多领域得到了广泛应用. PSO 除了具有以上诸多优点之外, 也面临着"早熟"的问题, 即在所优化问题存在局部极值解的情况下, 微粒种群搜索容易陷入局部极值, 使得算法的优化能力降低. 针对这一问题, 国内外学者进行了很多研究, 他们大多关注于在固定粒子数的基础上对其他优化策略和参数的改进[2-5], 而对于动态粒子数的研究则主要出现在遗传算法中[6-9].

最近, 一种基于动态粒子数的 PSO 改进方法被提出. 这种方法通过设计特定的粒子生灭机制来控制种群的粒子数量和多样性, 与以往基于固定粒子数的改进算法相比, 能更有效地提高种群收敛效率和全局搜索能力, 从而有效解决"早熟"问题. 耶刚强等[10] 提出了基于动态粒子数的 PSO 算法 (DPPPSO), 收敛过程中粒子数按照特定变化函数增加或减少. 算法每一拍根据粒子数变化大小, 在搜索空间中随机地产生或消灭粒子, 使得种群能够不断改变其搜索范围和搜索能力, 较好地改善了"早熟"现象. 但由于它使用了固定的粒子数变化函数, 算法不能根据搜索情况作出自适应调整. 在种群未出现"早熟"的情况下, 粒子数的硬性增加会为算法增加无谓的计算量; 而在种群已经出现"早熟"的情况下, 粒子数的硬性减少可能使得情况更加恶化.

本文针对 PSO 受局部极值影响的问题, 提出一种基于栅格策略的动态粒子数微粒群算法 (GB-DPPPSO). 通过建立栅格信息更新策略、粒子产生策略和粒子消亡策略, 自适应地控制种群粒子数量, 调整种群多样性, 从而在种群的全局搜索能力和搜

收稿日期: 2008-06-05; **修回日期**: 2008-10-20.
基金项目: 国家自然科学基金重点项目(60634030); 高等学校博士学科点专项科研基金项目(20060699032); 教育部新世纪优秀人才支持计划(NCET-06-878); 西北工业大学科技创新基金项目(W016143).
作者简介: 刘勇(1983—), 男, 河北衡水人, 博士生, 从事无线多传感器网络的研究; 潘泉(1961—), 男, 重庆人, 教授, 博士生导师, 从事多目标跟踪与识别、信息融合等研究.

索效率之间建立平衡,达到以较小的计算负荷快速获得全局最优的目的. 仿真结果表明了该算法的有效性.

2　基于栅格的动态粒子数微粒群算法

为了防止粒子种群陷入局部极值点,就要使种群保持较好的多样性. 随着种群的不断搜索,粒子受本身和群体经验的共同影响朝向某最优位置聚集,种群搜索范围逐渐减少. 而在种群认为的最优位置区域内,局部粒子密度逐渐增大. 此时,为了保持种群多样性,应在未搜索的区域内产生新的粒子进行搜索,同时,使粒子密度过大区域内的粒子消灭以降低计算量. 为此本算法设计了以下策略:

1) 栅格信息更新策略. 对搜索区域进行栅格划分,在种群搜索过程中更新每个栅格的粒子密度和搜索密度,并以此决定栅格的粒子产生概率.

2) 粒子产生策略. 每个栅格根据本身粒子产生概率在其空间内随机产生新的粒子,以保证种群多样性和全局搜索能力.

3) 粒子消灭策略. 在粒子密度超过阈值的栅格中,将多余粒子消灭,以减少冗余的计算量.

算法每一拍在经过以上 3 个策略确定种群之后,粒子的速度和位置的更新与 PSO 相同,即

$$v_{id}(t+1) = wv_{id}(t) + C_1\,\mathrm{rand}_1(p_{id}(t) - x_{id}(t)) + C_2\,\mathrm{rand}_2(p_{gd}(t) - x_{id}(t)), \quad (1)$$

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

其中: $x_{id}(t)$ 和 $v_{id}(t)$ 为 t 时刻第 d 个粒子在第 i 维上的位置和速度, $p_{id}(t)$ 和 $p_{pg}(t)$ 分别为第 d 个粒子在第 i 维上的最佳搜索位置和整个种群最佳搜索位置, w 为速度保持因子, C_1 和 C_2 为加速度因子, rand_1 和 rand_2 为均匀分布 $[0\ 1]$ 的随机数.

算法程序伪代码如下:

```
Begin
  Parameter Initialization
  Grid Initialization
  Set Maximun Iteration(iter_max)
  Set Iteration iter = 0
  Set Iteration End Marker finish = FALSE
  While !finish{
    iter = iter + 1
    Particle Generation Strategy
    For each particle{
      Calculate fitness value
      If the fitness value is better than the best
      fitness value(p_Best) in history{
        Set current value as the new p_Best}}
    Choose the particle with the best fitness value
    of all the particles as g_Best
    For each particle{
      Update particle velocity according to p_Best
      and g_Best
      Update particle position}
    Update Grid Information
    Particle Vanishing Strategy
    If iter ≻ iter_max{
      finish = TURE}}}
```

2.1　栅格信息更新策略

为了计算得到粒子在搜索空间中的收敛程度,从而决定粒子的产生和消灭,将 m 维搜索空间划分为 N 个栅格, $N = K_1 \times K_2 \times \cdots \times K_m$. 则在第 i 维上的栅格宽度为

$$W_i = \frac{\max f_i - \min f_i}{K_i},\ i = 1, 2, \cdots, m. \quad (3)$$

其中: f_i 表示优化问题的第 i 个未知参数, K_i 表示搜索空间第 i 维上划分的栅格数.

同时建立 3 个与搜索空间维数相同的矩阵: 第 1 个矩阵用来保存栅格粒子产生概率; 第 2 个矩阵保存栅格的粒子密度; 第 3 个矩阵保存栅格的搜索密度.

栅格的粒子产生概率为

$$P_c = \begin{cases} \lambda, & S_c \le \xi; \\ 0, & \text{other}. \end{cases} \quad (4)$$

其中: S_c 为搜索密度,即栅格的累计粒子密度

$$S_c = \sum_{j}^{t} d_c^j; \quad (5)$$

λ 为粒子产生概率的初始值; ξ 为搜索密度阈值,用来控制粒子对空间的搜索次数(当搜索密度 S_c 超过该阈值时, P_c 为零); d_c^j 为第 j 拍第 c 个栅格的粒子密度,即其中的粒子数目; t 为当前时刻.

3 个矩阵进行更新的过程为: 粒子在每一拍移动之后寻找对应栅格,并对该栅格的粒子密度加 1,搜索密度加 1,判断搜索密度是否超过阈值,若是,则将粒子产生概率置零. 图 1 给出了该过程在 2 维搜索空间中的一个例子,其中 $\xi = 2$.

2.2　粒子产生策略

若当前粒子数小于设置的最大粒子数 mp,则启动粒子产生策略. 粒子的产生是在每次循环开始时由栅格的粒子产生概率决定的. 栅格的粒子产生概率矩阵反映出粒子种群的多样性和全局搜索能力,搜索次数超过阈值的栅格的粒子产生概率为零,即在该区域内不再产生新的粒子,而对搜索次数少的栅格,则根据粒子产生概率产生新的粒子. 这样,

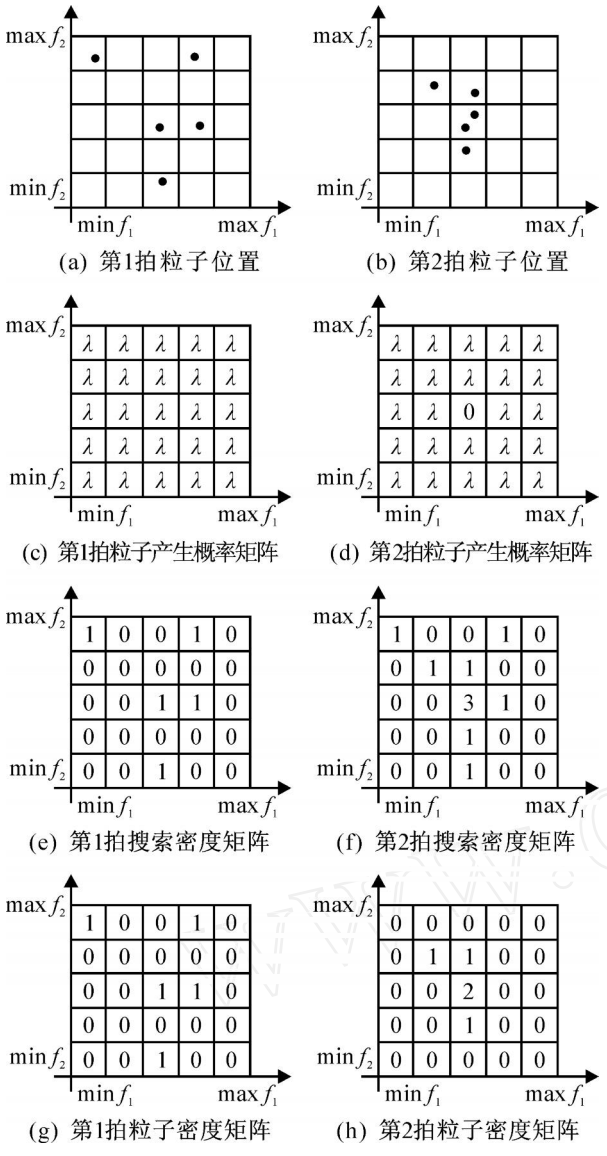


图 1 栅格信息更新策略过程示意

随着算法的不断循环,种群的搜索可以覆盖到整个搜索空间,从而保证了种群多样性和全局搜索能力.

由粒子产生概率决定有新粒子产生之后,栅格将在本身区域内产生新粒子

$$p_i = \mathrm{low}_c^i + \mathrm{rand}(\mathrm{up}_c^i - \mathrm{low}_c^i). \tag{6}$$

其中:p_i 为粒子第 i 维上的坐标,low_c^i 和 up_c^i 分别为第 c 个栅格在第 i 维上的下界和上界,rand 产生(0,1) 之间的均匀分布随机数. 粒子产生之后,将所处栅格在粒子密度矩阵和搜索密度矩阵中的对应元素上分别加 1.

2.3 粒子消灭策略

随着种群的不断收敛,粒子会向极值参数位置所在的栅格聚集,造成种群的过分密集搜索,因此在每次循环粒子位置更新后,对多余粒子采用消灭策略以减少冗余的计算量. 设置栅格中的粒子密度阈值 k,若每个栅格对应的粒子密度矩阵数值 $d_c^j > k$,则按照粒子适应性值的优劣进行排序,然后仅保留前 k 个粒子,其余粒子消灭. 即

$$\mathrm{marker}_p = \begin{cases} 0, & p \text{ is vanished;} \\ 1, & p \text{ is alive.} \end{cases} \tag{7}$$

其中 marker_p 为第 p 个粒子的消灭标志位:当粒子被消灭时置 0,当粒子处于激活状态时为 1. 最后,将所在栅格对应的粒子密度矩阵值减去 $d_c^j - k$.

3 数学函数验证

本节通过验证函数,从算法搜索成功率和搜索效率两方面,将本文算法与 DPPPSO 以及标准 PSO 进行对比,以验证本文算法的有效性. 仿真程序采用 Matlab 语言编写,运行环境为 Intel Core2 Duo 8300 @2.4 GHz,4G RAM.

3.1 搜索成功率分析

验证函数 1

$$f_1 = \frac{\sin^2\sqrt{x_1^2 + x_2^2} - 0.5}{[1 + 0.001(x_1^2 + x_2^2)]^2} + 0.5. \tag{8}$$

该函数的全局最小点位于点(0,0),且最小值为 0. 它在距全局极小值点大约 3.14 范围内存在无穷多个局部极小值将其包围,并且函数强烈振荡,一般很难得到最优解.

在验证函数 1 的仿真中,本文算法的参数设置为 $w = 0.6$, $C_1 = C_2 = 2$, $\lambda = 0.1$, $\xi = 5$, $k = 30$,最大粒子数为 60,最大速度为(10,10),搜索范围为 $[-1000,1000] \times [-1000,1000]$,栅格数为 10 × 10,循环次数为 500 次. DPPPSO 算法参数设置参见文献[10]中效果最好的 DPPPSO-3 设置,其衰减项中 $N_{\max} = 100$, $N_{\min} = 5$,波动项中 $A = 25$. 标准 PSO 算法的粒子数为 50. 三个算法其他基本设置相同,循环次数均为 500 次.

验证函数 2

$$f_2(x) = \sum_i^n (-x_i) \cdot \sin(\sqrt{|x_i|}). \tag{9}$$

该函数为 Schwefel 函数,其全局最小点在(420.9687, 420.9687) 处,且最小值为 $-n \times 418982.9$. 该函数在 $-500 \le x_i \le 500 (i = 1,2,\cdots,n)$ 范围内有多个局部极值,且都与全局极值接近,因此也很难得到最优解.

在仿真中,验证函数 2 的参数 $n = 2$. 本文算法的参数设置为 $w = 0.6$, $C_1 = C_2 = 2$, $\lambda = 0.1$, $\xi = 5$, $k = 6$,最大粒子数为 30,最大速度为(50,50),搜索范围为 $[-500,500] \times [-500,500]$,栅格数为 10 ×10,循环次数为 500 次. DPPPSO 算法同样参照 DPPPSO-3 设置,其衰减项中 $N_{\max} = 80$, $N_{\min} = 5$,波动项中 $A = 25$. 标准 PSO 算法粒子数为 50. 三个算法其他基本设置相同,循环次数均为 500 次.

下面通过多次仿真来验证算法对全局极值的搜

索成功率. 表 1 给出了 100 次仿真中 3 个算法最佳适应性值在 (0,1e-10), (1e-10,0.01) 和 (0.01, + ∞) 3 个区间上出现次数的统计结果. 从中可以看出,本文算法搜索到更多的全局极值,证明它拥有更高的全局搜索成功率. 本文算法有效保证了搜索过程中的种群多样性,从而减少了种群陷入局部极值的可能. 相对于标准 PSO,改进后的两个算法虽然取得了更好的全局搜索成功率,但也都在一定程度上增加了更多的计算量. 通过表 1 可知,本文算法所增加的计算负担小于 DPPPSO,这主要是由于本文算法通过粒子自适应的生灭机制,更有效地控制了种群粒子数量与种群多样性之间的平衡,且使用了更少的粒子.

表 1　函数 f_1 下多次仿真结果统计

		本文算法	DPPPSO	PSO
循环次数		500	500	500
运行次数		100	100	100
平均单次运行时间/s		0.69	0.88	0.38
最佳适应性值分布	(0,1e-10)	52	34	25
	(1e-10,0.01)	48	66	65
	(0.01, + ∞)	0	0	10

对于 Schwefel 函数,表 2 统计了多次仿真中 3 个算法所得到的最佳适应性值与其最小值 (- 837.9667) 差的绝对值的分布情况. 可以看到,本文算法在 Schwefel 函数下,同样以较小的计算耗时得到了更高的搜索成功率.

表 2　函数 f_2 下多次仿真结果统计

		本文算法	DPPPSO	PSO
循环次数		500	500	500
运行次数		100	100	100
平均单次运行时间/s		0.41	0.60	0.36
最佳适应性值分布	(0,1e-3)	77	65	53
	(1e-3,150)	23	35	45
	(150, + ∞)	0	0	2

3.2　搜索效率分析

验证函数 3

$$f_3 = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\frac{x_i}{\sqrt{i}} + 1. \qquad (10)$$

该函数为 Griewank 函数,其全局最优点位于点 (0,0) ,且最小值为 0.

在验证函数 3 的仿真中,本文算法的参数设置为 $w = 0.6$, $C_1 = C_2 = 2$, $\lambda = 0.2$, $\xi = 5$, $k = 5$,最大粒子数为 30,最大速度为 (5,5),搜索范围为 $[-1000,1000] \times [-1000,1000]$,栅格数为 10 × 10. DPPPSO 算法同样参照 DPPPSO-3 设置,其衰减项中 $N_{max} = 40$, $N_{min} = 5$,波动项中 $A = 25$. 标准 PSO 算法粒子数为 20. 三个算法的其他基本设置相同,循环次数均为 500 次.

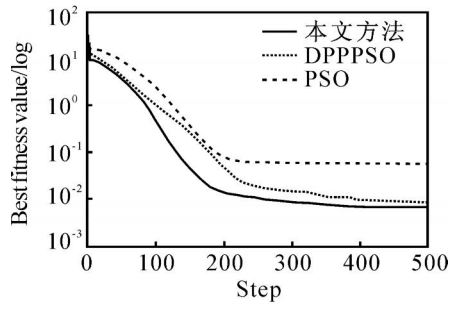
验证函数 4

$$f_4 = \sum_{i}^{n}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]. \qquad (11)$$

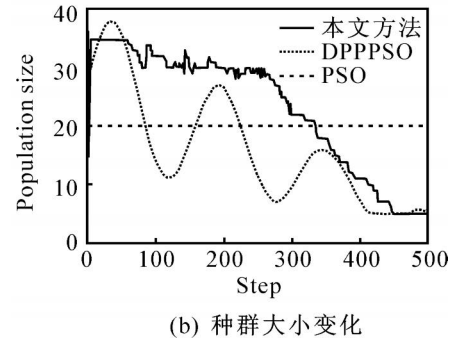
该函数为 Rosenbrock 函数,在 $n = 2$ 时,其全局最优点位于点 (1,1) ,且最优值为 0. 其最小值位于非常狭窄的一个条带上,沿着这一条带,函数变化非常缓慢. 因此尽管 Rosenbrock 函数是一个单峰值的函数,但它常被广泛地用于测试全局优化算法的搜索效率.

在验证函数 4 的仿真中,本文算法的参数设置为 $w = 0.6$, $C_1 = C_2 = 2$, $\lambda = 0.1$, $\xi = 5$, $k = 5$,最大粒子数为 30,最大速度为 (100,100) ,搜索范围为 $[-10000,10000] \times [-10000,10000]$,栅格数为 10 ×10. DPPPSO 算法同样参照 DPPPSO-3 设置,其衰减项中 $N_{max} = 40$, $N_{min} = 5$,波动项中 $A = 25$. 标准 PSO 算法粒子数为 20. 三个算法的其他基本设置相同,循环次数均为 500 次.

图 2 和图 3 给出了 3 个算法 100 次仿真后在每一拍的最佳适应性值和种群大小的平均值. 从图 2(a) 和图 3(a) 中可以得到,本文算法更快地找到了更好的极值位置,这主要是因为本文算法的基于栅格的粒子产生和消灭策略会动态地根据粒子种群的收敛情况改变种群大小,而不像 DPPPSO 中使用固定的粒子函数控制粒子数变化. 以上结果证明了本

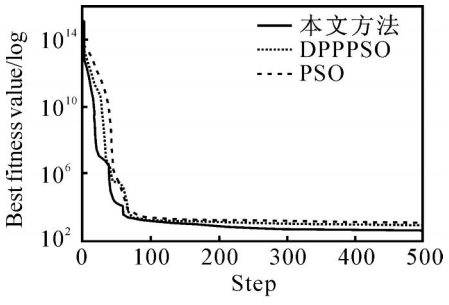


(a) 最佳适应性值变化

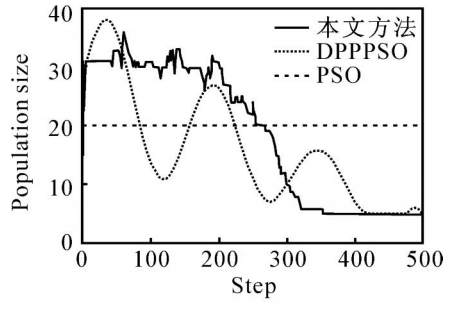


(b) 种群大小变化

图 2　函数 f_3 下 100 次仿真平均结果



(a) 最佳适应性值变化



(b) 种群大小变化

图 3 函数 f_4 下 100 次仿真平均结果

文算法的粒子生灭机制的有效性,它可以改进算法的搜索效率.

4 结 论

针对标准微粒群算法容易陷入局部最优的缺陷,本文提出了基于栅格的动态粒子数微粒群算法.在算法中,设计了栅格信息更新策略、粒子产生策略和粒子消灭策略.栅格信息更新策略用来维护和更新每个栅格的粒子密度和搜索密度,并以此决定栅格的粒子产生概率.粒子产生策略是指每个栅格根据本身粒子产生概率在其空间内随机产生粒子,以保证种群多样性和全局搜索能力.粒子消灭策略是将粒子密度较大的栅格中的粒子进行消灭,以消除冗余计算量.通过数学函数验证仿真,从搜索成功概率和搜索效率两方面与 DPPPSO 和标准 PSO 进行了对比.结果表明,本文算法能够较好地维持粒子种群多样性,同时有效控制粒子数量,提高了全局搜索成功率和效率.

参考文献(References)

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

[2] Shi Y, Eberhart R C. Empirical study of particle swarm optimization [C]. Proc of the 1999 Congress on Evolutionary Computation. Piscataway: IEEE, 1999: 1945-1950.

[3] 赫然, 王永吉, 王青, 等. 一种改进的自适应逃逸微粒群算法及实验分析[J]. 软件学报, 2005, 16(12): 2036-2044.
(He R, Wang Y J, Wang Q, et al. An improved particle swarm optimization based on self-adaptive escape velocity[J]. J of Software, 2005, 16(12): 2036-2044.)

[4] Ratnaweera A, Halgamuge S K, Watson H C. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients [J]. IEEE Trans on Evolutionary Computation, 2004, 8(3): 240-255.

[5] Fan H Y. A modification to particle swarm optimization algorithm [J]. Engineering Computations, 2002, 19(7/8): 970-989.

[6] Kennedy J, Mendes R. Neighborhood topologies in full-informed and best-of-neighborhood particle swarms[C]. Proc of the 2003 IEEE Int Workshop on Soft Computing in Industrial Application. Piscataway, 2003: 45-50.

[7] Arabas J, Michalewicz Z, Mulawka J. GAVaPS — A genetic algorithm with varying population size[C]. Proc of Congress on Evolutionary Computation. Orlando, 1994: 73-74.

[8] Zhuang N, Benten M, Cheung P. Improved variable ordering of BBDS with novel genetic algorithm[C]. Proc of IEEE Int Symposium on Circuits and Systems. Atlanta, 1996: 414-417.

[9] Alander J. On optimal population size of genetic algorithms [C]. Proc of IEEE Int Conf on Computer Systems and Software Engineering. Silverspring, 1992: 65-70.

[10] 耶刚强, 孙世宇, 梁彦, 等. 基于动态粒子数的微粒群优化算法[J]. 信息与控制, 2008, 37(1): 18-27.
(Ye G Q, Sun S Y, Liang Y, et al. Particle swarm optimization based on dynamic population size [J]. Information and Control, 2008, 37(1): 18-27.)