第三章 改进的微粒群算法

3.1 对基本微粒群算法进化方程的改进

3.1.1 基本微粒群算法分析

为了方便起见，我们将基本微粒群算法的形式表述如下：

$$V_i(t+1) = V_i(t) + c_1 r_1 (P_i - X_i(t)) + c_2 r_2 (P_g - X_i(t)) \tag{3.1}$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{3.2}$$

其中 P_i 表示第i个微粒所经历过的最好位置，P_g 表示所有微粒所经历过的最好位置，c_1、c_2为常数，$r_1, r_2 \in [0,1]$ 为均匀分布的随机数。

为了更好地分析基本微粒群算法，我们将（3.1）式改写为：

$$V_i(t+1) = G_1 + G_2 + G_3 \tag{3.3}$$

其中

$$G_1 = V_i(t)$$

$$G_2 = c_1 r_1 (P_i - X_i(t))$$

$$G_3 = c_2 r_2 (P_g - X_i(t))$$

从（3.3）可以看出，其右边可以分成三部分：第一部分为原先的速度项；第二、三部分分别表示对原先速度的修正。其中，第二部分考虑该微粒历史最好位置对当前位置的影响，而第三部分考虑微粒群体历史最好位置对当前位置的影响。为了考虑 G_1、G_2、G_3 对微粒群搜索能力的影响，我们首先将（3.3）修改为：

$$V_i(t+1) = G_1 \tag{3.4}$$

（3.4）式表示微粒速度的进化方程仅保留第一部分，此时，微粒将会保持速度不变，沿该方向一直“飞”下去直至到达边界。因而，在这种情形下，微粒很难搜索到较优解。

接着我们将（3.3）修改为：

$$V_i(t+1) = G_2 + G_3 \tag{3.5}$$

（3.5）式表示微粒速度的进化方程保留第二、三部分，由于微粒的速度将取决于其历史最优位置与群体的历史最优位置，从而导致速度的无记忆性。假设在开始时，微粒 j 处于整体的最优位置，则按照（3.5）式它将停止进化直到群体发现更好的位置。此时，对于其它微粒而言，$\lim_{t\to\infty} X_i(t) = P_g$。这表明，整个微粒群的搜索区域将会收缩到当前最优位置附近，即如果没有第一项，则整个进化方程具有很强的局部搜索能力。

对于基本微粒群算法而言，其具有全局搜索能力，这表明，微粒速度的进化方程的第一项是

用于保证算法具有一定的全局搜索能力。通过以上的分析，我们发现，G_2、G_3使得微粒群算法具有局部收敛能力，而G_1则是用于保证算法的全局收敛性能。

3.1.2 带有惯性因子的改进微粒群算法

1、一般的惯性因子设计。

对于不同的问题，如何确定局部搜索能力与全局搜索能力的比例关系，对于其求解过程非常重要。甚至对于同一个问题而言，进化过程中也要求不同的比例。为此，Yuhui Shi[Yuhui Shi1998]提出了带有惯性权重的改进微粒群算法。其进化方程为：

$$V_i(t+1) = wV_i(t) + c_1 r_1(P_i - X_i(t)) + c_2 r_2(P_g - X_i(t)) \tag{3.6}$$

$$X_i(t+1) = X_i(t) + V_i(t+1) \tag{3.7}$$

当惯性权重$w=1$时，式（3.6）与（3.1）相同，从而表明带惯性权重的微粒群算法是基本微粒群算法的扩展。文献[Yuhui Shi1998]建议w的取值范围为$[0,1.4]$，但实验结果表明当w取$[0.8,1.2]$时，算法收敛速度更快，而当$w>1.2$时，算法则较多地陷入局部极值。

惯性权重w表明微粒原先的速度能在多大程度上得到保留。假设微粒 j 的初始速度非零，当$c_1=c_2=0$且$w>0$时，则微粒将会加速直至$v_{\max}$；当$w<0$，则微粒将会减速直至 0。当$c_1, c_2 \neq 0$时，情况比较复杂，但[Yuhui Shi1998]的实验结果表明，$w=1$要好一些。

惯性权重w类似模拟退火中的温度，较大的w有较好的全局收敛能力，而较小的w则有较强的局部收敛能力。因此，随着迭代次数的增加，惯性权重w应不断减少，从而使得微粒群算法在初期具有较强的全局收敛能力，而晚期具有较强的局部收敛能力。在[Yuhui Shi1999]中，惯性权重w满足

$$w(t) = 0.9 - \frac{t}{MaxNumber} \times 0.5 \tag{3.8}$$

其中，MaxNumber 为最大截止代数，这样，将惯性权重w看作迭代次数的函数，可从 0.9 到 0.4 线性减少，从对四个测试函数的测试结果来看，效果很好。

2、基于模糊系统的惯性因子的动态调整。

对于惯性权重w来说，对于不同的问题，其每一代所需要的比例关系并不相同，这样，线性递减关系只对某些问题有效，对于其它问题而言显然不是最佳的。为此，[Yuhui Shi2001]提出了基于模糊系统的惯性权重的动态调整。该模糊系统需要两个输入参数：当前种群最优性能指标（the current best performance evaluation, CBPE）和当前的惯性权重，输出为惯性权重的调节量。具体操作时，首先假设所优化的问题为求解最小值的问题，其次对性能指标规范化操作，即

$$NCBPE = \frac{CBPE - CBPE_{\min}}{CBPE_{\max} - CBPE_{\min}} \tag{3.9}$$

其中，全局最小值（或估计值）为$CBPE_{\min}$，所得到的某个上界为$CBPE_{\max}$。三个变量（两个输入变量，一个输出变量）分为低、中、高三种状态。对于每种状态，其隶属度函数分别称为：

$f_{left\_triangle}$， $f_{triangle}$， $f_{right\_triangle}$，具体定义为：

$$f_{left\_triangle}=\begin{cases}1, if\,(x<x_1)\\ \dfrac{x_2-x}{x_2-x_1}, if\,(x_1\le x\le x_2)\\ 0, if\,(x>x_2)\end{cases} \tag{3.10}$$

$$f_{triangle}=\begin{cases}0, if\,(x<x_1)\\ 2\dfrac{x-x_1}{x_2-x_1}, if\,(x_1\le x\le \dfrac{x_1+x_2}{2})\\ 2\dfrac{x_2-x}{x_2-x_1}, if\,(\dfrac{x_1+x_2}{2}\le x\le x_2)\\ 0, if\,(x>x_2)\end{cases} \tag{3.11}$$

$$f_{right\_triangle}=\begin{cases}0, if\,(x<x_1)\\ \dfrac{x-x_1}{x_2-x_1}, if\,(x_1\le x\le x_2)\\ 1, if\,(x>x_2)\end{cases} \tag{3.12}$$

其中， x_1、 x_2 为两个参数用以确定函数的形状和位置。

整个模糊系统可以如下表示：

9
2 1
NCBPE 3 0 1
    LeftTriangle 0 0.06
    Triangle 0.05 0.4
    RightTriangle 0.3 1

Weight 3 0.2 1.1
    LeftTriangle 0.2 0.6
    Triangle 0.4 0.9
    RightTriangle 0.6 1.1

W_change 3 -0.12 0.05
    LeftTriangle –0.12 –0.02
    Triangle –0.04 –0.04
    RightTriangle 0.0 0.05

1 1 2
1 2 1
1 3 1
2 1 3
2 2 2
2 3 1
3 1 3

3 2 2

3 3 1

其中，第一行的“ 9 ”表示共有九条规则。第二行的“ 2 1 ”表示有两个输入变量和一个输出变量。第三行的“ NCBPE 3 0 1 ”表示第一个输入变量是个 NCBPE 变量（规范化操作），有三个模糊集，取值范围为 $(0,1)$ 。当使用隶属度函数 $f_{left\_triangle}$ 时，参数 x_1， x_2 取 0.0 和 0.06；使用隶属度函数 $f_{triangle}$ 时，参数 x_1， x_2 取 0.05 和 0.4；使用隶属度函数 $f_{right\_triangle}$ 时，参数 x_1， x_2 取 0.3 和 1.0。这四行完整的定义了第一个输入变量。第二个输入变量为 weight，输出变量为 w_change，分别进行了定义。定义完之后，又给出了具体的九条模糊规则。在规则中“ 1 ”代表低状态，“ 2 ”表示中状态，“ 3 ”表示高状态。因此，第一条规则“ 1 1 2 ”就表示输入变量 NCBPE 为低，weight 为低，输出变量 w_change 为中的情形。

3.1.3 带有收缩因子的微粒群算法

在Clerc[M.Clerc1999,D.Corne1999]的研究中，提出了收缩因子的概念。该方法描述了一种选择 w，c_1和 c_2 的值的方法，以确保算法收敛。通过正确地选择这些控制参数，就没有必要将$v_{i,j}$的值限制在[$-v_{max}$, v_{max}]之中。接下来首先讨论一个与带有收缩因子的微粒群算法相关的收敛模式特例。

一个与某个收敛模式相符合的改进了的速率方程式以以下形式提出：

$$v_{i,j}(t+1)=\chi\left(v_{i,j}(t)+c_1r_{1,j}(t)(p_{i,j}(t)-x_{i,j}(t))+c_2r_{2,j}(t)(p_{g,j}(t)-x_{i,j}(t))\right), \tag{3.13}$$

这里，

$$\chi=\frac{2}{\left|2-\ell-\sqrt{\ell^2-4\ell}\right|} \tag{3.14}$$

且 $\ell=c_1+c_2,\ell>4$

设 $c_1=c_2=2.05$，将 $\ell=c_1+c_2=4.1$ 带入（3.14），得出 χ=0.7298 并带入方程式（3.13），同时省略参数 t，结果为：

$$v_{i,j}(t+1)=0.7298\left(v_{i,j}+2.05\times r_{1,j}(p_{i,j}-x_{i,j})+2.05\times r_{2,j}(p_{g,j}-x_{i,j})\right) \tag{3.15}$$

因为 $20.5\times0.7298=1.4962$，所以这个方程式与在改进的 PSO 速率更新方程式使用 $c_1=c_2=1.4962$ 和 w=0.7298 所得到的方程式是等价的。

Eberhart 和 Shi 将分别利用 $v_{\max}$ 和收缩因子来控制微粒速度的两种算法性能作了比较，结果表明，后者比前者通常具有更好的收敛率。然而在有些测试函数的求解过程中，使用收缩因子的 PSO 在给定的迭代次数内无法达到全局极值点。按照 Eberhart 和 Shi 的观点，这是由于微粒偏离所期望的搜索空间太远而造成的。为了降低这种影响，他们建议在使用收缩因子时首先对算法进行限定，比如设参数 $v_{\max}=x_{\max}$，或者预先设置搜索空间的大小。这样几乎可以改进算法对所有测试函数的求解性能——不管是在收敛率方面还是在搜索能力方面。

3.2 基于遗传思想改进 PSO

3.2.1 利用选择的方法

在一般微粒群算法中，每个微粒的最优位置的确定相当于隐含的选择机制，因此，[Perter J.Angeline1998]引入了具有明显选择机制的改进微粒群算法，仿真结果表明算法对某些测试函数具有优越性。

改进算法所使用的选择算子为锦标赛选择算子（Tournament Selection Method），将每个个体的适应度，基于其当前位置，与 k 个其他个体进行比较，并记下最差的一个点。群体再用这个记录排序，最高的得分出现在群体的头部。

算法的流程为：

1) 从种群选择一个个体。将该个体的适应度与种群中的其他个体的适应度逐一进行比较，如果当前个体的适应度优于某个个体的适应度，则每次授予该个体一分。对每一个个体重复这一过程。
2) 根据前一步所计算的分数对种群中的个体进行由大到小的排列。
3) 选择种群中顶部的一半个体，并对他们进行复制，取代种群底部的一半个体，在此过程中最佳个体的适应度并未改变。

为了测试算法的性能，[Perter J.Angeline1998]使用了四个测试函数，它们分别是

$$f_0(x)=\sum_{j=1}^{n}x_j^2$$

$$f_1(x)=\sum_{j=1}^{n}(100(x_{j+1}-x_j^2)+(x_j-1)^2)$$

$$f_2(x)=\sum_{j=1}^{n}(x_j^2-10\cos(2\pi x_j)+10)$$

$$f_3(x)=\frac{1}{4000}\sum_{j=1}^{n}x_j^2-\prod_{j=1}^{n}\cos(\frac{x_j}{\sqrt{j}})+1$$

在仿真试验中，初始种群使用了两种不同的选择方式[D.Gehlhaar 1996]，用以比较两种算法的性能。一种使用典型的对称区域，具体的数据见表 3.1。另一种使用不对称的定义区域，见表 3.2。

表 3.1、对称区域的范围

Function	Initialization Range
f_0	$(-15,15)^n$
f_1	$(-15,15)^n$
f_2	$(-15,15)^n$
f_3	$(-600,600)^n$

表 3.2、不对称区域的范围

Function	Initialization Range
f_0	$(7.5,15)^n$
f_1	$(7.5,15)^n$
f_2	$(7.5,15)^n$
f_3	$(300,600)^n$

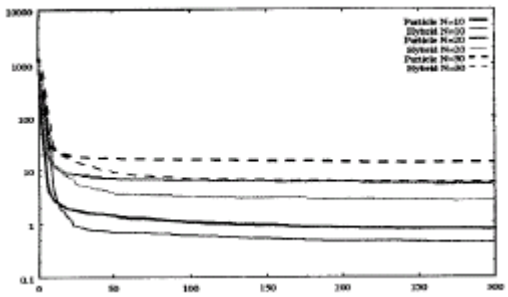

图 3.1、定义域为对称区域在 f_0 的性能比较

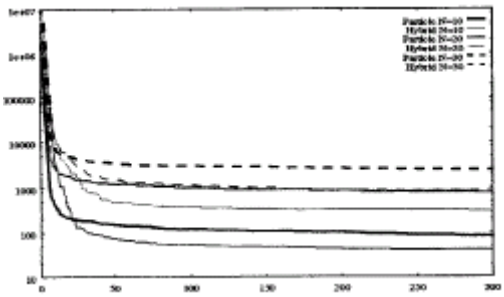

图 3.2、定义域为对称区域在 f_1 的性能比较

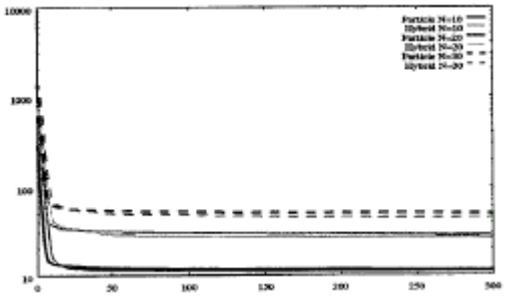

图 3.3、定义域为对称区域在 f_2 的性能比较

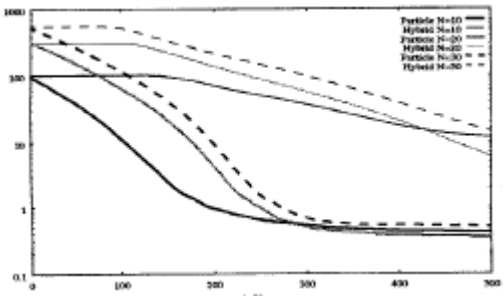

图 3.4、定义域为对称区域在 f_3 的性能比较

表 3.3、定义域为对称区域仿真结果

函数	维数	最大截止代数	方差	标准差
F0	10	500	0.7022	0.3823
F0	20	750	4.8467	2.2405
F0	30	1000	11.949	5.2181
F1	10	500	75.482	38.049
F1	20	750	640.16	253.65
F1	30	1000	1993.68	711.04
F2	10	500	11.214	10.965
F2	20	750	27.730	25.747
F2	30	1000	47.275	42.844
F3	10	500	0.4115	11.985
F3	20	750	0.3060	1.5058
F3	30	1000	0.4527	2.5145

图 3.1-3.4 表示的是初始种群为对称区域的实验结果。从图中可以看出，对于函数 $f_0 - f_2$ 而言，虽然开始基本微粒群算法的速度较快，但在 25 代以内，带有选择的改进微粒群算法效率提高很快，并超过基本微粒群算法的结果。但对于函数 f_3 而言，性能并没有提高很多。

对于初始种群为不对称区域，其实验结果可参考图 3.5-3.8，其结果与图 3.1-3.4 比较相似，这里就不再赘述。

表 3.4、定义域为不对称区域仿真结果

函数	维数	最大截止代数	方差	标准差
F0	10	500	0.7120	0.4016
F0	20	750	4.8669	2.2714
F0	30	1000	11.674	5.1870
F1	10	500	76.186	39.506
F1	20	750	674.64	269.22
F1	30	1000	1988.3	669.98
F2	10	500	11.305	10.972
F2	20	750	27.739	26.101
F2	30	1000	47.897	42.950
F3	10	500	1.2666	60.433
F3	20	750	0.3463	39.267
F3	30	1000	0.4425	50.704

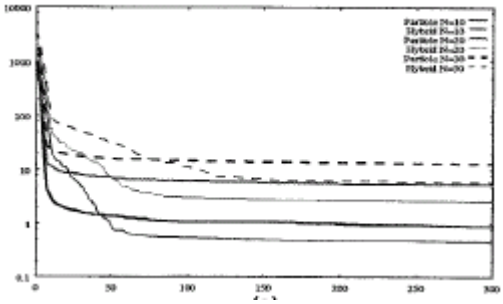

图 3.5、定义域为不对称区域在 f_0 的性能比较

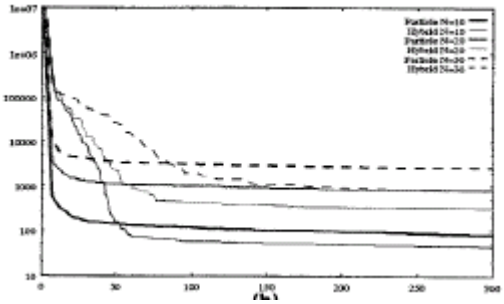

图 3.6、定义域为不对称区域在 f_1 的性能比较

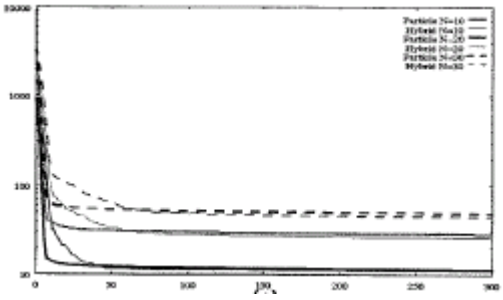

图 3.7、定义域为不对称区域在 f_2 的性能比较

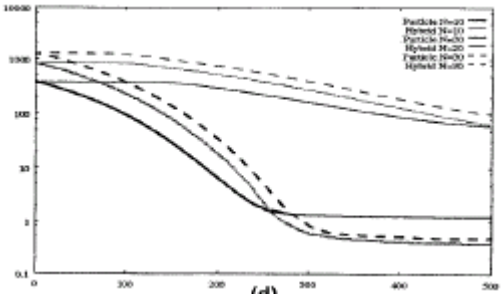

图 3.8、定义域为不对称区域在 f_3 的性能比较

3.2.2 借鉴杂交的方法

Angeline[Angeling 1998]提出了杂交微粒群算法，微粒群中的微粒被赋予一个杂交概率，这个杂交概率是用户确定的，与微粒的适应值无关。在每次迭代中，依据杂交概率选取指定数量的微粒放入一个池中。池中的微粒随机地两两杂交，产生相同数目的子代，并用子代微粒取代父代微粒，以保持种群的微粒数目不变。

让 a 和 b 表示被选择的两个亲代个体的指针，那么杂交算法的计算公式表示如下：

$$X_a(t+1) = r_1 X_a(t) + (1.0 - r_1) X_b(t) \tag{3.16}$$

$$X_b(t+1) = r_1 X_b(t) + (1.0 - r_1) X_a(t) \tag{3.17}$$

$$V_a(t+1) = \frac{V_a(t) + V_b(t)}{\left\| V_a(t) + V_b(t) \right\|} \left\| V_a a(t) \right\| \tag{3.18}$$

$$V_b(t+1) = \frac{V_a(t) + V_b(t)}{\left\| V_a(t) + V_b(t) \right\|} \left\| V_b a(t) \right\| \tag{3.19}$$

这里r_1~U(0,1)。经过杂交操作，在由亲代个体形成超立方体中随机产生了两个新的位置。速率的交叉处将两个亲代个体的速率之和的长度规格化，因此只有方向受到影响，数量却没有被影响。

Lovbjerg 等人[M.Lovbjerg2001]的研究结果表明繁殖操作降低了单峰值函数的收敛率，因此，应用了繁殖算子的 PSO 比原始的 PSO 效率更低。但是在拥有多个局部最小值的函数中情况恰恰相反。所以应用了繁殖算子的 PSO 比较先进。拥有全局最佳模式的无比较的方案被提出，因此它不清楚在拥有多个局部最小值的函数中繁殖算子是否比原始的全局最佳运算规则具有更好的执行效率。

3.3 利用小生境思想所作的改进

3.3.1 基于动态邻域的改进微粒群算法

基本微粒群中的LBEST模型，根据微粒的下标将微粒群体分割成若干个相邻的区域，也就是说，微粒x_1和x_2在一个半径为 1 的邻近区域内可以看成是相邻的，而不管他们的空间位置如何，根据微粒的空间位置，在每次运算规则迭代中，种群中一个微粒到其他微粒之间的距离都被计算出来。并且用变量$d_{\max}$ 来标记任何两个微粒之间的距离的最大值。对于每一个微粒来说，$\left\| x_a - x_b \right\| / d_{\max}$ 的比值也被计算出来，这里$\left\| x_a - x_b \right\|$是当前微粒$a$到另一个微粒b的距离。这个比值可用来作为选择相邻的微粒的依据，利用较小比值或较大的比值作为选择依据。基于该想法，P.N.Suganthan[P.N.Suganthan1999]于 1999 年提出一种基于邻域思想的微粒群算法，其基本思想是在算法开始阶段，每个个体的邻域为其自身，随着进化代数的增长，其邻域范围也在不断增大直至整个种群。

该想法的基本框架为

step1、初始化各微粒的位置及速度；

step2、计算各微粒适应值；

step3、比较并替换全局最优位置 PBEST；

step4、对每个微粒计算其邻域并确定局部最优 LBEST；

step5、利用速度进化方程进行进化；

$$v(t+1) = wv(t) + C_1(t) r_1 (PBEST - x(t)) + C_2(t) r_2 (LBEST - x(t))$$

step6、修改各微粒位置；

step7、按下式修改参数：

$$w(t) = w^{\infty} + (w^{0} - w^{\infty})[1 - \frac{t}{K}]$$

$$C_1(t) = C_1^{\infty} + (C_1^{0} - C_1^{\infty})[1 - \frac{t}{K}]$$

$$C_2(t) = C_2^{\infty} + (C_2^{0} - C_2^{\infty})[1 - \frac{t}{K}]$$

step8、重复 step2-step7，直至满足结束条件。

其中，K 为最大进化代数，t 为当前进化代数，且∞和 0 分别表示参数的起始和最终的值。

为了验证算法有效性，使用了与 3.2.1 相同的四个测试函数，并利用基本微粒群算法与该算法进行比较。在测试中，每个测试函数分别考虑 20、30 和 50 维的情形，种群所含微粒数 40，最大进化代数为 1000，每个测试函数作 50 次试验，且惯性因子 w 满足下面的形式：

$$w_{initial} = 0.95$$
$$w_{final} = 0.2$$
$$w(t) = (w_{initial} - 0.2) \times (MAXITER - ITER) + 0.2$$

参数 C_1, C_2 均为常数 2.0。实验结果见表 3.6。

表 3.5、实验结果

函数	维数	基本 PSO 算法结果	改进 PSO 算法结果
F1	20	0.0	0.0
		0.0	0.0
F1	30	0.000006	0.000005
		0.000043	0.000034
F1	50	0.8929	0.953
		9.162	8.80
F2	20	0.0108	0.0244
		0.02891	0.0318
F2	30	0.189051	0.00552
		0.2125	0.0194
F2	50	0.1669	0.1611
		0.5635	0.4104
F3	20	51.74	49.26
		1516.1	1443.7
F3	30	106.59	109.45
		3506.85	3447.99
F3	50	383.0	431.4
		11745.3	11557.5
F4	20	0.1104	0.1399
		1.705	1.48
F4	30	0.0615	0.0659
		0.662	0.5778
F4	50	0.0528	0.0615
		0.783	0.7614

显然，通过表 3.5，我们可以发现，对于函数 f_1, f_3, f_4 而言，不论其维数多少，基于动态邻域的改进微粒群算法的收敛精度均比基本微粒群算法要好，当然，对于不同的测试函数，其效果不太相同。函数 f_2 比较特殊，当维数为 20 时，改进微粒群算法不如基本微粒群算法好，但对于维数 30 和 50 时，改进微粒群算法的计算结果均优于基本微粒群算法。当然，我们可以看到，其改进效果比较弱，这是由该算法的邻域结构有关，因此，针对不同的问题提出适合该问题的邻域

结构就非常重要了。下面我们具体给出一些基本的邻域结构。

3.3.2 基本的邻域结构

事实上，我们可以看到，微粒群算法中的 LBEST 模型实际上就是一种简单的邻域结构，其每个邻域均由微粒的下标决定，而与微粒的具体位置无关。为了进一步改善算法性能，避免过早收敛的现象，J.Kennedy 在 1999 年[J.Kennedy1999]提出几种基本的邻域结构，即环形结构（ring topology）和轮形结构（wheel topology）及它们的推广。

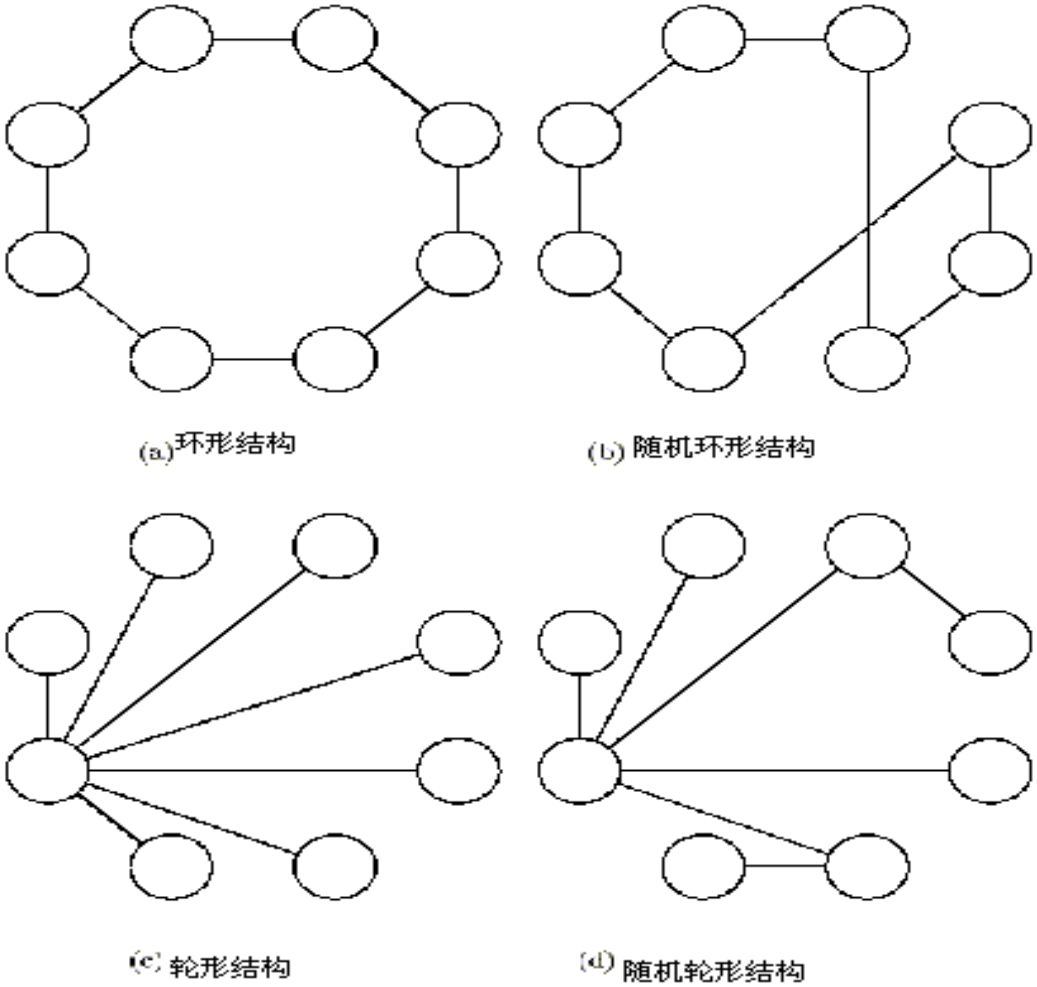

图 3.9 几种常见的邻域结构

环形结构是一种最基本的邻域结构，在该结构中，每个微粒仅与其附近的两个微粒相互交流信息，从而能有效的保证种群的多样性。比如，对于微粒 j 而言，其邻域为微粒 j+1 和 j-1，从而其信息只能在这两个微粒中交流。这时，对于微粒 k 与微粒 2k 而言，其信息的交流至少需要经过 k 步才能完成，从而能保证算法充分对每个区域进行搜索，而不至于陷入过早收敛的情形。此外，这里的邻域仅与微粒的下标有关，而与微粒的真实位置无关，从而可以有效的对搜索空间进行搜索，但有时会影响算法效率。作为该结构的推广，随机环形结构强调了信息交流的迟滞性，但对环形结构的邻域仅由微粒下标决定的缺点作了一定调整，其邻域虽然主体仍由微粒下标决定，但存在两个微粒的邻域可以随机选择，从而在一定程度上提高了算法效率，对算法速度与精度的调整起到了平衡作用。

但环形结构的信息交流非常慢，因而即使是随机环形结构，有时其问题的求解也比较慢。此时，可以考虑使用轮形结构。在该结构中，每个邻域的不再仅仅是 3 个微粒了，而是若干个。其中一个为中心，其余的微粒都与它相邻。换句话说，在每个邻域中，只有中心微粒与该邻域中的所有微粒可以交换信息，而其他微粒彼此间不能互相交换信息，这样提高了信息的传播速度，避

免了信息的丢失，从而能更好的提高算法效率。

3.3.3 一种保证种群多样性的微粒群算法

为了避免微粒群算法所存在的过早收敛问题，J.Riget[J.Riget2002]提出了一种保证种群多样性的微粒群算法（Attractive and Repulsive Particle Swarm Optimizer，简称 ARPSO）。该算法引入“吸引”（attractive）和“扩散”（repulsive）两个算子，动态地调整“勘探”与“开发”比例，从而能更好的提高算法效率。

ARPSO 算法的基本流程图如下：

```
Program PSO
    Init();
    While not done do
      SetDirection();       //新算子
      UpdateVelocity();
      NewPosition();
      AssignFitness();
      CalculateDiversity();     //新算子
```

该算法的速度进化方程为

$$V_i(t+1) = V_i(t) + dir(c_1 r_1(P_i - X_i(t)) + c_2 r_2(P_g - X_i(t))) \tag{3.20}$$

其中

$$dir = \begin{cases} -1, if\ (dir > 0).and.(diversity < d_{low}) \\ 1, if\ (dir < 0).and.(diversity > d_{high}) \end{cases} \tag{3.21}$$

并且提出了种群多样性函数

$$diversity(S) = \frac{1}{|S| \cdot |L|} \cdot \sum_{i=1}^{|S|} \sqrt{\sum_{j=1}^{N} (p_{ij} - \bar{p}_j)^2} \tag{3.22}$$

其中，S 为种群，|S|为种群所含微粒的个数，|L|为搜索空间的最长半径，N 为问题的维数，p_{ij} 为第 i 个微粒的第 j 个分量。在算法运行过程中，如果种群多样性函数满足 $diversity(S) < d_{low}$，则 $dir = -1$，从而种群不再向整体最优位置靠近，而是纷纷远离该最优位置，从而执行了“扩散”操作，而当种群多样性逐步增大，直至超出上限 d_{high} 时，$dir = 1$，从而种群又开始向整体最优位置靠拢，即执行了“吸引”操作。

为了验证算法有效性，利用 Griewank 函数、AckeyF1 函数、Rosenbrock 函数和 Rastrigin 函数进行了测试，并与基本微粒群算法和遗传算法进行了比较。其中，种群的邻域定义为上节所使用的。惯性因子 w 定义为 $w = 1 - t/t_{\max}$，t 表示当前进化代数，而 $t_{\max}$ 表示最大截止代数。此外，d_{high} 为 0.25，d_{low} 为 5.0×10^{-6}。

在遗传算法中使用了锦标赛选择算子（tournament-selection, tournamentsize 为 2），单点杂交算子及高斯分布 $N(0,\sigma^2)$ 的变异算子，其中 $\sigma^2(t) = 1 - t/t_{\max}$ 或 $\sigma^2(t) = 1/(1+\sqrt{t})$

每个测试函数分别考虑维数为 20、50、100 的情形，且相应的最大进化代数分别为 40000、100000、200000，此外，还考虑了一种特殊情形（表 3.3.1 中的 ARPSO*），即不论维数多少，其最大进化代数均为 200000。误差为1.0×10^{-10}，结果如表 3.6 与 3.7 所示。

表 3.6、运行三种算法所得到的平均适应值

Dim	20	50	100
Performance on Griewank			
GA	1.71E-2	8.51E-2	2.39E-1
BPSO	1.74E-2	1.35E-2	1.25E-2
ARPSO	2.50E-2	3.05E-2	9.84E-2
ARPSO*	2.40E-2	2.99E-2	3.74E-2
Performance on AckeyF1			
GA	0.012	0.376	0.389
BPSO	0.018	0.668	0.830
ARPSO	0.33E-7	0.027	0.218
ARPSO*	0.30E-7	0.96E-2	0.015
Performance on Rosenbrock			
GA	107.1	199.41	254.0
BPSO	11.16	30.08	122.14
ARPSO	2.34	10.43	103.46
ARPSO*	1.67E-3	0.116	88.71
Performance on Rastrigin			
GA	14.43	45.36	63.07
BPSO	9.71	47.14	96.59
ARPSO	0	0.2E-1	0.438
ARPSO*	0	0	0

表 3.7、算法 ARPSO 中“吸引”与“扩散”算子对平均时间及效率的贡献度

Problem	Time spent		Improvements	
	Attraction	Repulsion	Attraction	Repulsion
Ackey20	76.9	23.1	98.5	1.5
Ackey50	78.4	21.6	98.9	1.1
Ackey100	81.1	18.9	98.9	1.1
Griewank20	76.8	23.2	98.2	1.8
Griewank50	78.6	21.4	97.1	2.9
Griewank100	80.1	19.9	96.4	3.6
Rastrigin20	83.6	16.4	98.9	1.1
Rastrigin50	85.6	14.4	98.8	1.2
Rastrigin100	88.6	11.4	98.9	1.1
Rosenbrock20	69.2	30.8	88.8	11.2
Rosenbrock50	71.3	28.7	89.3	10.7
Rosenbrock100	79.2	20.8	95.4	4.6

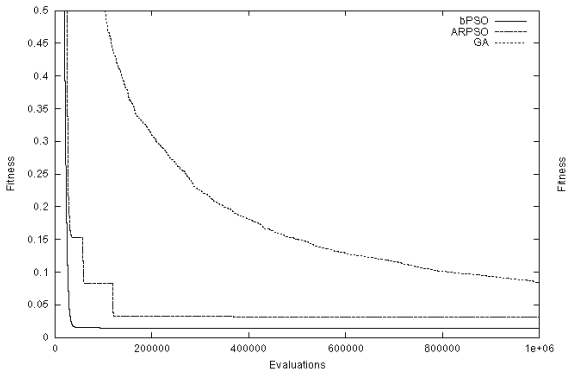

图 3.10、Griewank 函数维数为 50 的最优适应值

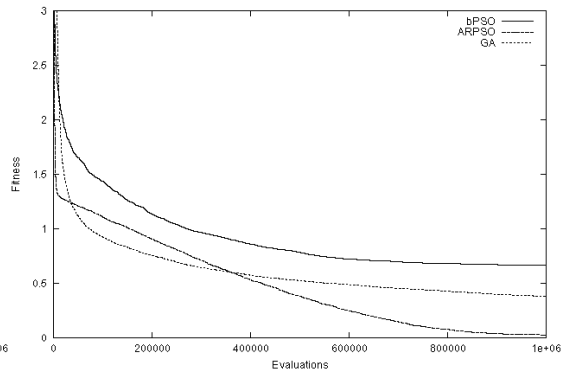

图 3.11、AckeyF1 函数维数为 50 的最优适应值

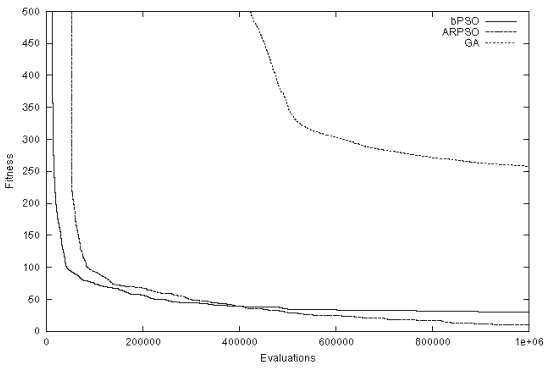

图 3.12、Rosenbrock 函数维数为 50 的最优适应值

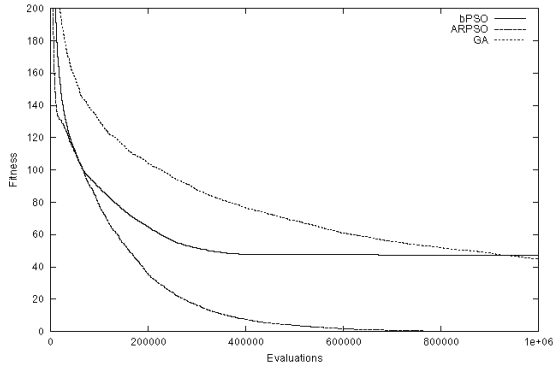

图 3.13、Rastrigin 函数维数为 50 的最优适应值

为了更好的比较基本微粒群算法与 ARPSO 算法的种群多样性的差别，对 50 维的 Rastrigin 函数进行了比较，结果如图 3.14 所示。

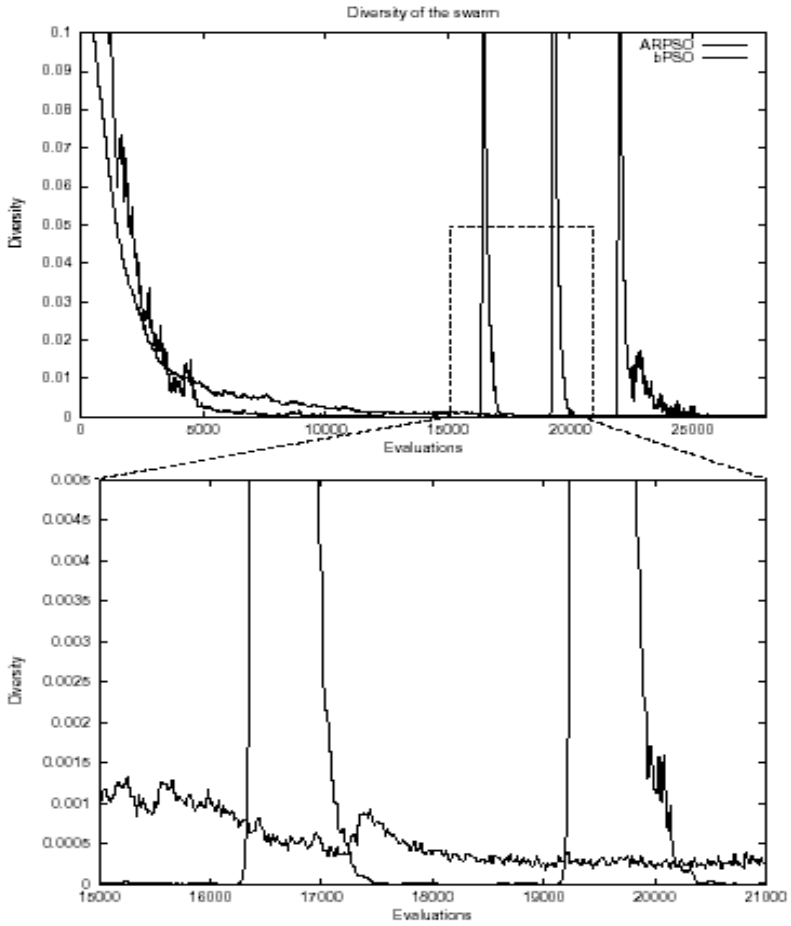

图 3.14、基本 PSO 与 ARPSO 的多样性比较

从表 3.6 中可以看出，对于 4 个测试函数的 12 种不同的测试结果来看，ARPSO 的收敛精度都远远超过了基本 PSO 算法，尤其对于高维情形，效果更佳。表 3.7 表明随着维数的增加，ARPSO 的“扩散”效果逐步减弱，而“吸收”效果逐步增强，从而证明了算法非常适用于高维测试函数。从图 3.10-3.13 可以看出，虽然开始时 ARPSO 的效果不一定比基本 PSO 好，但当算法运行到 40000 代附近时，ARPSO 的效果明显超过基本 PSO 算法，并迅速收敛到全局最优解附近。

3.4 利用收敛性分析所作的改进

3.4.1 保证收敛的改进微粒群算法

由于基本 PSO 算法有过早收敛的可能，F.van den Bergh 提出了具有局部收敛性能的改进微粒群算法 GCPSO(Guaranteed Convergence Particle Swarm Optimizer)[F.van den Bergh 2002a]，并给出了仿真结果。同年，在[F.van den Bergh2002b]中讨论了 GCPSO 的局部收敛性能，并指出该算法不具有全局收敛性能。为了进一步讨论 GCPSO 的性能，在[E.S.Peer 2003]中考虑了 GCPSO 在不同邻域结构中的表现。

为了方便起见，这里将基本微粒群算法的进化方程复述一下。

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,j}(p_{i,j} - x_{i,j}(t)) + c_2 r_{2,j}(p_{g,j} - x_{i,j}(t)) \tag{3.23}$$

其中，w 为惯性权重，用以确定该微粒的先前速度的保留情况。c_1, c_2 是两个正的加速因子，$r_{1,j}, r_{2,j}$ 是两个介于（0,1）的随机数分布序列。P_i 是微粒 i 所得到的最优位置，$p_{g,j}$ 是所有微粒得到的历史最优位置。

当微粒 i 现在的位置是当前种群的最优位置时，即 $x_{i,j}(t) = p_{i,j} = p_{g,j}$，此时（3.23）中的第二部分和第三部分将为 0，从而可简化为

$$v_{i,j}(t+1) = wv_{i,j}(t)) \tag{3.24}$$

对于（3.24）式，由于仅有一项，即原有速度的调整，从而微粒 i 将一直沿着该方向搜索，直到发现更好的解或者到达边界。但是，对于整个区域而言，沿直线进行搜索，其发现更好解的概率几乎为 0，因而其效率极低。此外，如果 w<1，即 $v_{i,j}(t+1) < v_{i,j}(t)$，则存在某一固定的数 N，使得当 $t > N$ 时，若群体的当前最优解没发生变化，则微粒 i 的速度各分量 $v_{i,j}(t) < \varepsilon$（ε 为一给定的误差），从而使得该微粒停止进化。这样，即使该方向上存在更好解，也可能在搜索到该解之前停止进化，从而导致过早收敛。

为了解决这个问题，我们设 τ 是当前最优位置所在的下标，则有 $y_\tau = P$

为了保证该微粒能正常移动，我们引入下面的速度进化方程：

$$v_{\tau,j}(t+1) = -x_{\tau,j}(t) + P_{g,j}(t) + wv_{\tau,j}(t) + \rho(t)(1 - 2r_{2,j}(t)) \tag{3.25}$$

此时，微粒 τ 的进化方程为

$$x_{\tau,j}(t+1) = P_{g,j}(t) + wv_{\tau,j}(t) + \rho(t)(1 - 2r_{2,j}(t)) \tag{3.26}$$

其中，参数 $\rho(t)$ 按下式进行更新：

$$\rho(t+1)=\begin{cases}2\rho(t), if\ (\#success > s_c)\\ 0.5\rho(t), if\ (\#\ failure >\ f_c)\\ \rho(t), otherwise\end{cases} \tag{3.27}$$

其中，初始值 $\rho(0)=0$，而#success 和#failure 分别表示成功和失败的次数。若

$$f(\hat{y}(t))=\ f(\hat{y}(t-1)) \tag{3.28}$$

则表示一次失败。这样，我们有

$$\#success(t+1) >\#success(t) \Rightarrow\#\ failure(t+1)=0 \tag{3.29}$$

$$\#\ failure(t+1) >\#\ failure(t) \Rightarrow\#success(t+1)=0 \tag{3.30}$$

此外，f_c 和 s_c 为两个阈值。

3.4.2 保证全局收敛的随机微粒群算法

为了描述方便，这里重新描述一下微粒群算法（PSO）的进化方程为：

$$V_i(t+1)=wV_i(t)+c_1r_1(P_i-X_i(t))+c_2r_2(P_g-X_i(t)) \tag{3.31}$$

$$X_i(t+1)=X_i(t)+V_i(t+1) \tag{3.32}$$

其中 p_i 表示第i个微粒所经历过的最好位置，p_g 表示所有微粒所经历过的最好位置，w、c_1、c_2为常数，$r_1,r_2\in[0,1]$ 均匀分布的随机数。

在（3.31）、（3.32）所描述的基本 PSO 算法中，当 w=0 时，微粒的飞行速度只取决于微粒的当前位置 $x_i(t)$、历史最好位置 p_i 和微粒群的历史最好位置 p_g，速度本身无记忆性。这样，对于位于全局最好位置的微粒将保持静止，而其它微粒则趋向它本身最好位置 p_i 和全局最好位置 p_g 的加权中心。也就是说，微粒群将收缩到当前的全局最好位置，更像一个局部算法；当 w≠0 时，使得微粒具有了扩展搜索空间的趋势，即具有一定的全局搜索能力。w 越大，全局搜索能力越强。

根据上述分析，当 w=0 时，（3.31）、（3.32）描述的进化方程为

$$X_i(t+1)=X_i(t)+c_1r_1(P_i-X_i(t))+c_2r_2(P_g-X_i(t)) \tag{3.33}$$

与基本 PSO 算法相比，（3.33）式描述的进化方程使得全局搜索能力减弱，而局部搜索能力加强。同时，当 $x_j^{(t)}=P_j=P_g$ 时，第 j 个微粒将停止进化。为了改善（3.33）式的全局搜索能力，可保留 P_g 作为微粒群的历史最好位置，而在搜索空间 S 重新随机产生微粒 j 的位置 $x_j(t+1)$，其它微

粒 i 以（3.33）式进化产生 $x_i(t+1)$（i≠j）,则

$$P_j = X_j(t+1)\,,$$

$$P_i = \begin{cases} P_i, f(P_i) < f(X_i(t+1)) \\ X_i(t+1), f(P_i) \geq f(X_i(t+1)) \end{cases} \tag{3.34}$$

$$p_g' = \arg\min\{f(P_i) \mid i = \overline{1,S}\}$$

$$p_g = \arg\min\{f(P_g'), f(P_g)\} \tag{3.35}$$

若 $p_g = p_j$，则随机产生的微粒 j 处于历史最好位置，无法按（3.33）式进化，继续在搜索空间 S 随机产生，其它微粒在更新 p_g、 p_i 后按（3.33）式进化；若 $p_g \neq p_j$，且 p_g 未更新，则所有微粒均按（3.33）式进化；若 $p_g \neq p_j$，且 p_g 已更新，即存在 k≠j,使得 $x_k(t+1) = p_k = p_g$，则微粒 k 停止进化，在搜索空间 S 重新随机产生，其余微粒在更新 p_g、 p_i 后按（3.33）式进化。这样在进化的某些代，至少有一个微粒 j 满足 $x_j(t) = p_j = p_g$，也就是说，至少有一个微粒需在 S 中重新随机产生，这样就势必增强了全局搜索能力。为了与基本 PSO 算法相区别，上述算法称之为随机 PSO 算法（SPSO）（源程序见附录 2）。

定义 $\varphi_1 = c_1 r_1$， $\varphi_2 = c_2 r_2$， $\varphi = \varphi_1 + \varphi_2$，由（3.35）式可得：

$$X_i(t+1) = (1-\varphi)X_i(t) + \varphi_1 P_i + \varphi_2 P_g \tag{3.36}$$

当 p_g、 p_i 固定时，上式为一简单的线性差分方程，当 $x_i(0) = x_{i0}$ 时，其解为：

$$X_i(t) = k + (x_{i0} - k)(1-\varphi)^t \tag{3.37}$$

其中，

$$k = \frac{\varphi_1 P_i + \varphi_2 P_g}{\varphi} \tag{3.38}$$

（3.37）式是在假设随着t的变化而 p_g、 p_i 固定不变的情况下得到的。但在SPSO算法的进化过程中， p_g、 p_i 则随时可能更新，因此，（3.37）、（3.38）式仅在新的更好位置产生之前有效。一旦产生新的更好位置（ p_g 或者 p_i），微粒的运动轨迹方程将按照新的 p_g、 p_i，并将当前位置作为初始点重新计算，也就是说（3.37）式中k,x_{i0}的值重新设置。

从（3.37）式可以看出，当 $|1-\varphi| < 1$ 时，（3.36）式所描述的进化方程线性收敛，即当 $t \to \infty$

时，$X_i(t) \to \dfrac{\varphi_1 P_i + \varphi_2 P_g}{\varphi}$。根据$|1-\varphi|<1$，可得：$0 < c_1 + c_2 < 2$。也就是说，当$0 < c_1 + c_2 < 2$时，SPSO 算法的进化方程线性渐近收敛。

定理 1：当$|1-\varphi|<1$时，$\lim_{t\to\infty} X_i(t) = P_g$

证明：由（3.31）式知，当$|1-\varphi|<1$时，$\lim_{t\to\infty} x_i(t) = k = \dfrac{\varphi_1 P_i + \varphi_2 P_g}{\varphi}$，而

$$X_i(t+1) = X_i(t) - (\varphi_1 + \varphi_2) X_i(t) + \varphi_1 P_i + \varphi_2 P_g$$

当$t \to \infty$时，$X_i(t+1) = X_i(t)$，则

$$-(\varphi_1 + \varphi_2) X_i(t) + \varphi_1 p_i + \varphi_2 p_g = 0$$

由于φ_1, φ_2为随机变量，显然只有当$x_i(t) = p_i = p_g$时，上式满足，也就是说，$\lim_{t\to\infty} x_i(t) = p_i = p_g$。

为了使微粒 j 以较大概率位于最优点附近，可采用其它一些非群体随机优化方法进行生成，如模拟退火方法。以当前代历史最好位置p_g为初始状态，即$x_j(t) = p_g$，并选择初始温度$T = T_0$，采用下式产生下一状态：

$$X_j'(t+1) = X_j(t) + \eta\xi \tag{3.39}$$

其中η为扰动幅值参数，ξ为随机变量，一般可服从柯西分布、正态分布或均匀分布。计算$f_j' = f(x_j'(t+1)), f_j = f(x_j(t))$，$\Delta = f_j' - f_j$，

$$X_j(t+1) = \begin{cases} X_j'(t+1), \min\{1, e^{-\Delta/T_k}\} \geq \gamma_j \\ X_j(t), otherwise \end{cases} \tag{3.40}$$

其中$\gamma_j \in [0,1]$均匀分布的随机变量。

$$T_{k+1} = \lambda T_k, 0 < \lambda < 1 \tag{3.41}$$

由于模拟退火方法本身具有很好的全局收敛性，因而采用该方法生成微粒 j 并依（3.39）、（3.40）式进行状态更新对于 SPSO 算法的全局收敛性不会产生负面影响。

为了验证本节算法的有效性，分别采用基本 PSO 算法和 SPSO 算法对下列函数的最小化问题进行实例计算。

（1） Goldstein-Price 函数

$$f_1(x) = [1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$$

$$\times [30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$$

$$-2 \le x_1, x_2 \le 2$$

（2） J.D.Schaffer 函数

$$f_2(X) = \frac{\sin^2\sqrt{x_1^2 + x_2^2} - 0.5}{[1 + 0.001(x_1^2 + x_2^2)]^2} - 0.5，\ -100 \le x_1, x_2 \le 100$$

在基本 PSO 算法中，w 从 1.0 到 0.4 随进化而线性减少，$c_1 = c_2 = 1.8$。在 SPSO 算法中，取$c_1 = c_2 = 0.5$，对停止进化微粒分别采用随机产生和模拟退火等两种方法分别对上述问题进行了 50 次仿真计算，群体规模为 20，最大进化代数为 500，其计算结果如表 1 所示。为了比较三个算法的性能，我们在相同的初始条件下，对两个测试函数进行计算，每隔 25 代纪录一次当前最优解，结果如图 3.15、3.16 所示。其中，在图 3.15 中，由于初始值太大（测试数据为 108.3457），为了更好的比较，我们将第一个测试数据略去，仅在图 3.15 中描绘了 24 个点。

表 3.9、实例计算结果

函数	算法	误差	平均收敛率	平均收敛代数
F1	基本 PSO	0.0001	100	156.8
F1	随机 PSO	0.0001	100	15.34
F1	模拟退火 PSO	0.0001	100	13.56
F2	基本 PSO	0.01	30	67.066667
F2	随机 PSO	0.01	62	233.806452
F2	模拟退火 PSO	0.01	98	56.7832

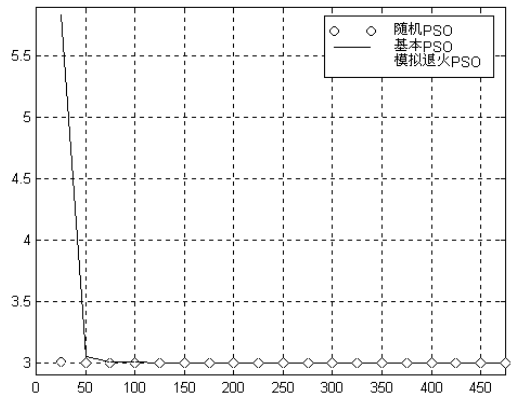

图 3.15、三种算法在函数 f_1 中的表现

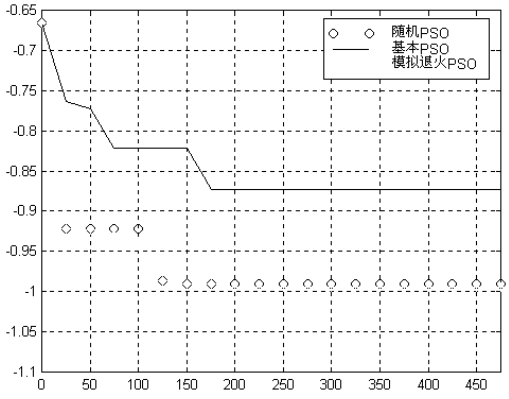

图 3.16、三种算法在函数 f_2 中的表现

从表 3.9 及图 3.15、3.16 中可以看出，本节所提出的 SPSO 算法在全局收敛性方面具有明显的优势，特别是模拟退火与 SPSO 算法的结合取得了几乎每次均收敛于全局最优解的理想结果，而且与初始群体的选取关系不大，从而验证了理论分析结果。

3.5 离散变量的微粒群算法

3.5.1 二进制编码的微粒群算法

二进制编码作为一种比较重要的编码形式，首先由 J.Kennedy 和 Eberhart[J.Kennedy 1997]在 1997 年将基本微粒群算法应用于二进制编码，并作了大量的数值研究[J.Kennedy 1998]。

显然，在二进制编码中，$x_j(t)$ 应取 0 或 1，但由（3.25）可以看出，$v_j(t)$ 计算结果可能不是整数，且由（3.26）可以看到，迭代后 $x_{j+1}(t)$ 可能会取 0、1 以外的其它数值。因此，如何调整（3.25）及（3.26），使得算法迭代后，结果 $x_{j+1}(t)$ 仍能取 0 或 1?

为此，Kennedy 引入了模糊函数 Sig(x)，其定义为

$$Sig(x) = \frac{1}{1+\exp(-x)} \tag{3.42}$$

这样，迭代公式（3.26）就变为

$$x_{i,j}(t+1) = \begin{cases} 0, if\ (r_3 \geq Sig(v_{i,j}(t+1))) \\ 1, otherwise \end{cases} \tag{3.43}$$

其中，$r_3 \in [0,1]$ 为均匀分布的随机数。通过这种方式，我们将 $x_{j+1}(t)$ 限制在集合 $\{0,1\}$ 中。

需要说明的是，在基本微粒群算法中，$v_j(t)$ 表示速度，能够对当前位置 $x_j(t)$ 的方向和位置随机产生一定的影响，使得算法在给定区域上进行搜索。而在二进制编码的微粒群算法中，$v_j(t)$ 仅表示一个概率，即微粒的每一维分量的取值以 $Sig(v_j(t))$ 的概率取 1，而以 $1-Sig(v_j(t))$ 的概率取 0。换句话说，如果微粒的某维分量是 0，则其变为 1 的概率为 $Sig(v_j(t))$ ；反之，如果该分量为 1，则其变为 0 的概率为 $1-Sig(v_j(t))$。这样，我们可以定义每一位的改变概率

$$\rho(\Delta) = Sig(v_{id})(1 - Sig(v_{id})) \tag{3.44}$$

即

$$\rho(\Delta) = Sig(v_{id}) - Sig^2(v_{id}) \tag{3.45}$$

在基本微粒群算法中，$v_j(t)$ 有个上限 $v_{\max}$，它表示微粒所允许的最大飞行速度。在二进制编码中，$v_{\max}$ 也存在，但它表示算法所允许的概率范围。但是，通过数值计算，可以发现，如果 $v_{\max} > 10$，则 Sig 函数将趋近于 0。因此，[J.Kennedy1997]中取 $v_{\max}$ 为 6.0，此时 Sig 函数的范围为 0.9975-0.0025，这表明随着 $v_j(t)$ 的增加，Sig 函数将逐渐减少，但最低将达到 0.0025，从而保证算法仍能够有能力发生变化。而[R.C.Eberhart2001]中 $v_{\max}$ 则为 4，此时 Sig 函数的范围为 0.982-0.018，从而使得算法能以较大的概率发生变化。

在[J.Kennedy1997]中，利用了 De Jong 的五个函数 $f_1 - f_5$ 测试了一下二进制编码的微粒群算法的性能，结果如图 3.17-3.21 所示。在测试中，每个测试函数运行 20 次，种群数为 20，$v_{\max}$ 为

6.0。图像 3.1-3.5 的横坐标为进化代数，纵坐标为最优解的均值，算法每隔 10 代测量一次。

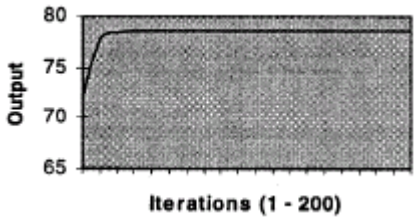

图 3. 17 二进制编码的 PSO 算法在 f_1 的性能测试

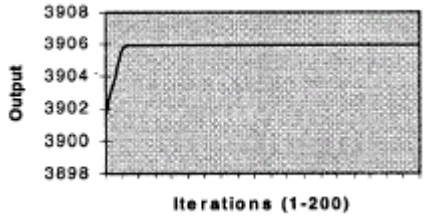

图 3.18 二进制编码的 PSO 算法在 f_2 的性能测试

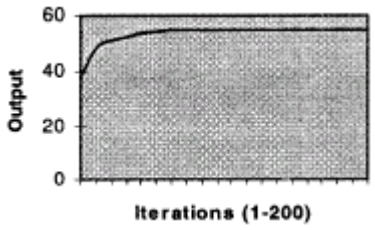

图 3.19、二进制编码的 PSO 算法在 f_3 的性能测试

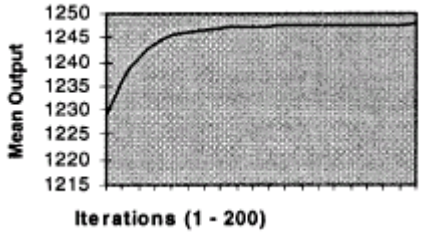

图 3.20、二进制编码的 PSO 算法在 f_4 的性能测试

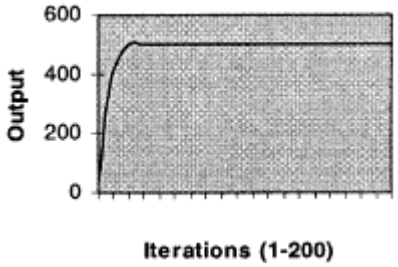

图 3.21、二进制编码的 PSO 算法在 f_5 的性能测试

在图 3.17 中，二进制编码的微粒群算法迅速收敛到函数 f_1 的最优解附近，输出的最优目标值是 78.60，在 20 次运行中有 10 次得到了 78.599998 的最优结果。函数 f_2 是对于二进制编码的微粒群算法而言效果最差的，只有 4 次收敛到最优解。对于函数 f_3 而言，每次均可收敛到全局最优解。由于 De Jong 的函数 f_4 中引入了高斯噪声，因而测量的是整个种群的平均适应值，而不是个体的最优适应值。对于函数 f_5，最优解为 500.0，算法得到的最优解为 499.056335，从图 3.21 中可以看出， 算法的收敛速度很快。

为了更好的体现二进制微粒群算法的优越性，[J.Kennedy1998]针对不同的数值优化问题，考虑了不同情形的遗传算法和二进制微粒群算法进行比较。他主要考虑了以下几种算法比较。

A. 只含有杂交、选择算子的遗传算法，简记作 GA_c;

B. 只含有变异、选择算子的遗传算法，简记作 GA_m;

C. 含有杂交、变异和选择算子的简单遗传算法，简记作 GA;

D. 二进制编码的微粒群算法，简记作 PS.

在实验中，使用了多峰问题生成器，它是[De Jong1990]中的 P-峰问题生成器的改进。设生成的问题有 P 个极值点，则个体 c 的适应值定义为与其最近的极值点的相同位的个数除以串长 L，即

$$f(c)=\frac{1}{L}\max_{j=1}^{P}\{L-Ham\min g(c,Peak_j)\} \tag{3.46}$$

其中，Hamming(x,y)表示串 x 与 y 的 Hamming 距离， $Peak_j$ 表示第 j 个极值点。

实验中，对每种算法，设计了 20 和 100 个极值点的两种类型，位长为 20 和 100 位。总的进化代数为 20000 代。其中，第 20 代进行一次抽样，从 1000 代开始每隔 1000 代抽样一次，从而得到 21 次抽样结果。在算法 GA 和 GA_m 中的变异概率为 0.001，算法 GA 和 GA_c 中的杂交概率为 0.6，对于二进制编码的微粒群算法，$v_{\max}$ 取为 2.0，群体数为 100。下面的图 3.22 和表 3.10 均取自[J.Kennedy1998]。

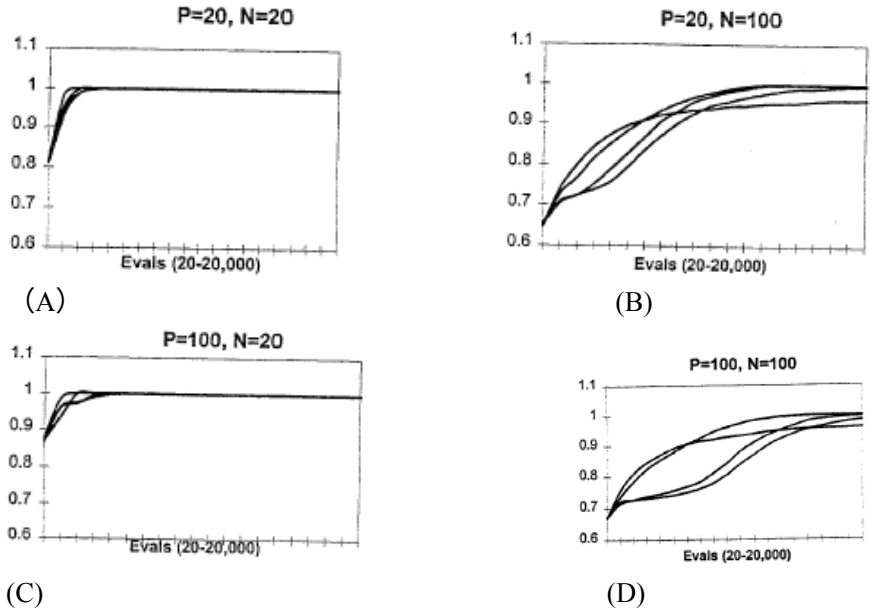

图 3.22 四种算法对于各种情形的效率比较

表 3.10、[J.Kennedy1998]给出的实验结果

Eval	Alg	P	L	Alg*L	Alg*P	P*L	Apg*P*L
20	0.18	165.07	3536.16	0.46	0.18	33.25	0.58
1000	67.24	25.20	10112.56	5.00	1.04	0.22	1.40
2000	165.02	0.18	14679.93	54.87	3.83	13.51	0.17
3000	261.10	3.53	14515.03	210.61	3.52	0.06	1.43
4000	282.39	23.31	10593.87	274.78	7.87	20.16	7.58
5000	217.89	60.90	6090.26	217.89	16.77	60.90	16.77
6000	181.59	121.67	4239.01	181.59	32.99	121.67	32.99
7000	145.04	190.55	2905.38	145.04	49.22	190.55	49.22
8000	101.20	173.05	1808.74	101.20	45.05	173.05	45.05
9000	72.49	155.65	1189.71	72.49	41.57	155.65	41.57
10000	47.11	103.29	739.47	47.11	29.18	103.29	29.18
11000	41.81	80.21	529.33	41.81	21.10	80.21	21.10
12000	45.00	72.92	484.22	45.00	49.73	72.92	19.73
13000	50.44	64.51	410.93	50.44	18.64	64.51	18.64
14000	53.14	43.03	540.84	53.14	14.30	43.03	14.30

15000	77.61	30.55	334.90	77.61	13.64	30.53	13.64
16000	105.81	27.97	377.71	105.81	10.80	27.97	10.80
17000	142.12	27.94	400.91	142.12	9.11	27.94	9.11
18000	188.60	22.25	417.82	188.60	6.88	22.25	6.88
19000	164.57	10.20	301.36	164.57	3.80	10.20	3.80
20000	152.23	6.50	253.34	152.23	2.07	6.50	2.07

从图 3.22 中可以看出，GA_m 一般在进化开始效果最好，但难于收敛，而 GA 和 GA_c 虽然开始较差，且收敛速度较慢，但最终都能由于 GA_m 的效果。对于 PSO 而言，大多数情形开始时慢于 GA_m，但最终结果要优于 GA 和 GA_c。

3.5.2 混合编码的微粒群算法

H.Yoshida 在 1999 年 [H.Yoshida1999]、Y.Fukuyama 在 2001 年[Y.Fukuyama2001] 分别提出了混合编码的微粒群算法。这里将介绍 H.Yoshida 在 1999 年提出的用于求解电压控制（Volt/Var Control,简称 VCC）的混合编码的微粒群算法。

下面首先给出 VCC 的数学模型：

$$\min f_c = \sum_{i=1}^{n} Loss_i \tag{3.47}$$

其中，n 表示所有的电流分路，而 $Loss_i$ 表示在第 i 条分路中的能量损失。并且应该满足下面的条件：

1）每个节点处的电压限制；
2）每条分路的能量限制；
3）电压的稳定性限制。

至于所求的适应值函数可以通过测量电压负荷后求和计算得到。

在 VCC 问题的求解过程中，需要应用到下述信息：

1) AVR 的操作值（连续变量）；
2) OLTC 位置（离散变量）；
3) 补充能量装置（离散变量）。

算法实现时，设每个微粒的位置变量为 $(AVR_1,...,AVR_T,OLTC_1,...,OLTC_H,E_1,...,E_M)$，其中 AVR_j 表示 AVR 的操作值的第 j 维分量，$OLTC_j$ 表示 OLTC 的位置的第 j 维分量，E_j 表示补充能量装置的第 j 维分量。由于 AVR 的操作值为连续变量，因此，在进化时使用（3.1）、（3.2）式。而 OLTC 位置与补充能量位置为离散变量，因而在进化时使用（3.42）、（3.43）式。

在仿真中，该文分别给出具有 5、14、12 个节点的系统，由于比较专业，我们就不一一介绍了。这里仅给出在 14 个节点时的收敛曲线，用以表明该算法的有效性。

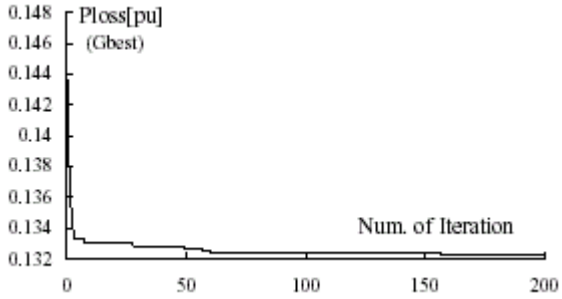

图 3.23、一个简单的 14 节点系统的收敛性能

3.5.3 整数空间的微粒群算法

整数规划问题可描述为：

$$\min\ f(\underline{x}),\quad \underline{x}\in S\subseteq Z^n \tag{3.48}$$

其中 Z^n 为 n 维整数空间，S 为一整数集，可以为无界集。

当采用 PSO 算法求解整数规划问题时，由于 $\omega, c_1, r_1, c_2, \gamma_2$ 的存在，即使微粒的当前位置和速度均为整数，下一位置则可能为实数，这样就使得搜索仍在包含 S 的实数空间中进行。下面针对整数取值问题，对（3.25）式进行分析。

在 $x_{id}(t), v_{id}(t+1)$ 为整数时，式（3.26）保证了 $x_{id}(t+1)$ 为整数，这样只需保证 $v_{id}(t+1)$ 在进化过程中为整数，就可以保证进化搜索在整数空间中进行。对于（3.25）式中的 $c_1 r_1(p_{id}-x_{id}(t))$ 项，由于 r_1 为在[0，1]内均匀分布的随机数，定义 $\phi_1 = c_1 r_1(p_{id}-x_{id}(t))$，则当 $p_{id} > x_{id}(t)$ 时，ϕ_1 为在[0，$c_1(p_{id}-x_{id}(t))$]内均匀分布的随机数。当 $p_{id} < x_{id}(t)$ 时，ϕ_1 为在[$c_1(p_{id}-x_{id}(t)),0$]内均 分布的随机数。为了讨论方便，定义 ϕ_1 均匀分布的区间为[a_1, b_1]，设[a_1, b_1]区间内的整数点数为 m_1，则若 ϕ_1 按等概率分布在 m_1 个整数点中进行选择，即：

$$P\{z_i \,|\, z_i \in [a_1,b_1]\text{的整数}\} = P\{z_j \,|\, z_j \in [a_1,b_1]\text{的整数}\} = \frac{1}{m_1} \tag{3.49}$$

就可以保证 ϕ_1 为整数，对 $\phi_2 = c_2 r_2(p_{gd}-x_{id}(t))$ 可作同样分析。

对于 $wv_{id}(t)$ 项，由于 $v_{id}(t)$ 本身为整数，一种方法是固定惯性加权系数 ω 为 1，同时引入飞行最大速度 $v_{\max}$，当 $v_{id}(t) > v_{\max}$ 时，$v_{id}(t) = v_{\max}$；另一种方法是对 $\omega v_{id}(t)$ 项进行取整，即 $int(\omega v_{id}(t))$，这样就可以保证 $v_{id}(t+1)$ 为整数。

有了上述讨论，整数规划问题的 PSO 算法可形式化描述为：

$$v_{id}(t+1) = \text{int}(\omega v_{id}(t)) + \phi_1 + \phi_2 \tag{3.50}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1) \tag{3.51}$$

其中：

$$\phi_1 \in [a_1,b_1]\text{等概率分布的整数，即 } p\{\phi_1\} = 1/m_1 \tag{3.52}$$

$$\phi_2 \in [a_2,b_2]\text{等概率分布的整数，即} p\{\phi_2\} = 1/m_2 \tag{3.53}$$

$$a_1=\begin{cases}0 & p_{id}>x_{id}(t)\\ c_1(p_{id}-x_{id}(t)) & p_{id}\le x_{id}(t)\end{cases} \tag{3.54}$$

$$b_1=\begin{cases}c_1(p_{id}-x_{id}(t)) & p_{id}>x_{id}(t)\\ 0 & p_{id}\le x_{id}(t)\end{cases} \tag{3.55}$$

$$a_2=\begin{cases}0 & p_{gd}>x_{id}(t)\\ c_2(p_{gd}-x_{id}(t)) & p_{gd}\le x_{id}(t)\end{cases} \tag{3.56}$$

$$b_2=\begin{cases}c_2(p_{gd}-x_{id}(t)) & p_{gd}>x_{id}(t)\\ 0 & p_{gd}\le x_{id}(t)\end{cases} \tag{3.57}$$

上式反映了标准 PSO 算法进化计算的基本思想，同时将进化计算有效地控制在整数空间。下面给出求解整数规划的 PSO 算法的步骤：

step1：初始化一群体规模为 D 的微粒群，包括随机位置和速度；

step2：计算每个微粒的适应值（即目标函数）；

step3：对每个微粒 i，将其适应值于其经历过的最好位置 P_i 做比较，如果较好，则将其作为当前的最好位置 P_i ；

step4：对每个微粒 i，将其适应值与全局所经历过的最好位置 P_g 做比较，如果较好，则将其作为当前的全局最好位置 P_g ；

step5：根据方程（3.50）—（3.57）进化每个微粒的速度和位置

step6：如未达到结束条件（通常为足够好的适应值或达到一个予设的最大迭代次数），则返回 step2。

为了验证上述求解整数规划 PSO 算法的有效性，对下述几个问题用该方法进行了实例计算。

$$F_1(\underline{x})=\|\underline{x}\|_1=|x_1|+|x_2|+\ldots+|x_D|,\quad \underline{x}=(x_1,x_2,\ldots,x_D)^T\in[-100\quad 100]^D$$

$$F_2(\underline{x})=\underline{x}^T\underline{x}=x_1^2+x_2^2+..x_D^2,\quad \underline{x}=(x_1,x_2,..,x_D)^T\in[-100\quad 100]^D$$

$$F_3(\underline{x})=-(15\quad 27\quad 36\quad 18\quad 12)\underline{x}+\underline{x}^T\begin{pmatrix}35&-20&-10&32&-10\\-20&40&-6&-31&32\\-10&-6&11&-31&32\\32&-31&-6&38&-20\\-10&32&-10&-20&31\end{pmatrix}x$$

$$F_4(\underline{x})=(x_1+10x_2)^2+5(x_3-x_4)^2+(x_2-2x_3)^4+10(x_1-10x_4)^4$$

对 $F_1(\underline{x})$，$F_2(\underline{x})$，D 分别取 5，10，20，30，其计算结果如表 3.11。对 $F_3(\underline{x})$，$F_4(\underline{x})$，取值区间分别取 $[-100\quad 100]^D$，$[-50\quad 50]^D$，$[-25\quad 25]^D$，其计算结果如表 3.12 所示。在所有实例计算中，$c_1=c_2=0.5$，ω从 1.2 到 0.4 逐渐减少。

表 3.10 $F_1(\underline{x})$、$F_2(\underline{x})$ 的计算结果

函　数	维数 D	群体规模	收敛率	平均进化代数
F1	5	20	100%	50
	10	50	100%	126
	20	200	98%	348

	30	300	95%	520
F2	5	20	100%	48
	10	50	100%	135
	20	200	100%	315
	30	300	95%	610

表 3.11 $F_3(\underline{x})$、$F_4(\underline{x})$ 的计算结果

函　数	取值范围	群体规模	收敛率	平均进化代数
F1	[-100, 100]	30	95%	450
	[-50, 50]	30	99%	412
	[-25, 25]	30	100%	350
F2	[-100, 100]	30	98%	432
	[-50, 50]	30	100%	410
	[-25, 25]	30	100%	365

从表 3.10、表 3.11 的计算结果可以看出，本节所提出的在整数域进行直接进化搜索的 PSO 算法从收敛的最优性和收敛速度方面均具有很好的性能。与应用实数域方法再取整相比，各方面性能明显提高。特别是对于一些最优点远离实数域最优点的问题，本节方法显示出了更加明显的优越性。

从理论分析及实验结果可以看出，本节方法至少在以下方面优于实数域方法：

1）将进化限制在整数空间，减少了许多不必要的非可行解域的搜索，提高了计算效率；

2）充分利用了整数规划问题的特点，使得进化向着目标函数改进的方向进行，而不是像实数域方法那样，向着目标函数“伪”改进方向进行；

3）进化的每一代的计算量基本没有变化，群体的多样性仍然保证与基本 PSO 算法一致。

3.5.4 求解旅行商问题的微粒群算法

为了将微粒群算法应用于组合优化问题，我们需要提出有序编码的微粒群算法。此时，关键在于给出微粒的位置与速度的类似定义[Clerc M.2000][Kang-Ping Wang2003]。

旅行商问题可用图论的语言描述为：

给定有向图 $G=\{E_G,V_G\}$，其中 E_G 为点（城市）的集合，V_G 为赋权边（表示两个城市间的距离），目的是寻找一个 Hamilton 圈（即起点与终点相同，而中间的任何两个点均不相同）。如果设 $E_G=\{1,2,...,N\}$，$V_G=\{(i,j,w_{ij}),i,j\in E_G,w_{ij}\in R^+\}$，则 Hamilton 圈为路径 $v_{k_1k_2}v_{k_2k_3}...v_{k_{N-1}k_N}v_{k_Nk_1}$，其中，$k_j\in E_G$，且 $k_i\neq k_j \Leftrightarrow i\neq j$。

在下面，我们首先给出一些必要的概念，然后给出相应的进化公式，最后是具体的实验结果。

1） 状态空间与位置

由于 TSP 问题的结果是要求出具有最短路径的 Hamilton 圈，因此，我们可以认为位置是一个具有所有的节点的 Hamilton 圈，从而状态空间即为所有的位置的集合。

2） 目标函数

设共有 N 个节点，不妨设某个位置 $x=(n_1,n_2,...,n_N,n_{N+1})$，其中 $x_1=x_{N+1}$，

$n_j \in E_G, j = 1,2,...,N$，若弧$(n_j, n_{j+1}) \in V_G$，则存在正数$w_{n_j n_{j+1}} \in R^+$，使得其为$n_j$到$n_{j+1}$的距离。若弧$(n_j, n_{j+1}) \notin V_G$，则称为虚弧，定义相应的距离$w_{n_j n_{j+1}} \in R^+$为

$$\begin{cases} w_{n_j n_{j+1}} = L_{\max} + (N-1)(L_{\max} - L_{\min}) \\ L_{\max} = \max\{w_{i,j}\}_{i,j=1}^{N} \\ L_{\min} = \min\{w_{i,j}\}_{i,j=1}^{N} \end{cases} \tag{3.58}$$

从而位置 x 的适应值函数为

$$f(x) = \sum_{j=1}^{N} w_{n_j n_{j+1}} \tag{3.59}$$

3） 速度

速度定义为点的位置的变换集。比如，如果用 v 表示速度的话，$\| v \|$表示该速度所含交换的数目，则该速度 v 可以表示为

$$v = \{(i_k, j_k), i_k, j_k \in \{1,2,...,N\}, k \in \{1,2,...,\| v \|\}\} \tag{3.60}$$

它表示首先交换n_{i_1}, n_{j_1}的位置，然后交换n_{i_2}, n_{j_2}的位置，以此类推。

如果速度v_1, v_2作用于任意位置，得到相同的结果，则称速度v_1, v_2等价，记作$v_1 \cong v_2$。比如$\{(1,3),(2,4)\} \cong \{(2,4),(1,3)\}$。

定义空速度为一个空表，记作ϕ。

定义逆速度为速度所含元素顺序的逆，记作$\neg v$，即

$$\neg v = \{(i_k, j_k), i_k, j_k \in \{1,2,...,N\}, k \in \{\| v \|,...,2,1\}\} \tag{3.61}$$

显然，$\neg(\neg v) = v$。

4） 速度与位置的加法

设 x 为某个位置，v 为速度，则定义它们的加法$x + v$为依次作用 v 中的变换于 x。例如$x = \{1,2,3,4,5,1\}$，$v = \{(1,2),(2,3)\}$，则$x + v$的结果为$\{3,1,2,4,5,3\}$。

定义位置 x 与位置 y 的减法为速度 v，即$x - y = v \Leftrightarrow x = y + v$，从而有$x - y = \neg(y - x)$。

5） 速度与速度的加法

定义速度v_1与速度v_2的加法为两个变换集的并。从而有

$$v \oplus \neg v = \phi$$

$$v_1 \oplus v_2 = v_2 \oplus v_1$$

6） 速度的倍数

设$v=\{(i_k,j_k),i_k,j_k\in\{1,2,...,N\},k\in\{1,2,...,\|v\|\}\}$，c 为任意实数，则 cv 可定义为

$$cv=\begin{cases}0,if\,(c=0)\\(i_k,j_k),i_k,j_k\in\{1,2,...,N\},k\in\{1,2,...,\|cv\|\},if\,(0<c\le 1)\\v\oplus v\oplus...\oplus v\oplus c'v,if\,(c>1)\\(-c)\neg v,if\,(c<0)\end{cases} \tag{3.62}$$

其中，$\|cv\|=\mathrm{int}eger(c\|v\|)$，函数 integer（）为向下取整函数。而当 c>1 时，如 $c=k+c'$，其中$c'\in(0,1)$，则 cv 表示 k 个 v 相加，再加上 c’v。

7） 两个位置的距离

设 x，y 为两个位置，则定义其距离为

$$d(x,y)=\|x-y\|$$

从而有

$$\begin{aligned}&d(x,y)=d(y,x)\\&d(x,y)=0\Leftrightarrow x=y\\&d(x,y)+d(y,z)\ge d(x,z)\end{aligned} \tag{3.63}$$

利用上面的概念，我们可以给出应用于 TSP 问题的离散微粒群算法的进化方程：

$$V_i(t+1)=V_i(t)\oplus c_1r_1(P_i-X_i(t))\oplus c_2r_2(P_g-X_i(t)) \tag{3.64}$$

$$X_i(t+1)=X_i(t)+V_i(t+1) \tag{3.65}$$

如果$c_1r_1=c_2r_2$，则令

$$P_i=P_g+\frac{1}{2}(P_i-P_g)$$

则上述方程可简化为

$$V_i(t+1)=V_i(t)\oplus c_3(P_i-X_i(t)) \tag{3.66}$$

$$X_i(t+1)=X_i(t)+V_i(t+1) \tag{3.67}$$

在[Clerc M.2000]中利用（3.64）、（3.65）求解 TSP 问题，而[Kang-Ping Wang2003]则利用（3.62）、（3.63）求解 TSP 问题。

在[Kang-Ping Wang2003]中，考虑了一个具有 14 个节点的经典 TSP 问题，其数据见下表：

表 3.12、具有 14 个节点的经典 TSP 问题

Node	1	2	3	4	5	6	7
Coord X	16.47	16.47	20.09	22.39	25.23	22.00	20.47
Coord Y	96.10	94.44	92.54	93.37	97.24	96.05	97.02
Node	8	9	10	11	12	13	14
Coord X	17.20	16.30	14.05	16.53	21.52	19.41	20.09
Coord Y	96.29	97.38	98.12	97.38	95.59	97.13	94.55

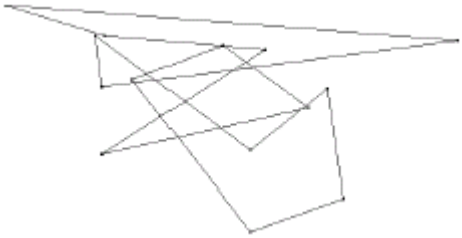

图 3.24、随机产生的初始解（距离为 59.8462）

图 3.25、得到的最优结果（距离为 30.8785）

在该问题中，解空间共有 $\frac{14!}{14*2}=3113510400$ 组解。微粒种群内有 100 个微粒，算法最大迭代次数为 20000，得到的最优解为

1->10->9->11->8->13->7->12->6->5->4->3->14->2