第八章 微粒群算法在函数优化中的应用

8.1 使用函数“stretching”技术的 PSO 算法

寻求性能良好的全局优化算法并使之能可靠收敛于问题的全局最优解，这一直是优化领域孜孜以求的研究目标及热点。目前已有的各种全局优化算法，如 GA、ES、EP 等，尽管已被成功地应用于各种优化问题及实际工程领域，但是当面对复杂的优化问题，尤其是多峰、多极值的多模态函数优化问题时，由于目标问题存在着众多的局部极值，因此不可避免地存在着早熟、收敛速度慢等缺陷。PSO 算法虽然已被证明是一种高效、简单的全局优化算法，但是随着目标问题的复杂化，同样也面临着上述挑战。

为提高 PSO 算法对复杂问题全局最优解的探测能力，文献[Parsopoulos KE 2001a]在 PSO 算法中引入了一种函数扩展技术（Function “Stretching” Technique）[Vrahatis 1996]，这一技术能够有效地提高 PSO 算法的全局收敛性能。本节内容将基于复杂函数优化问题对此项技术做详细介绍。

8.1.1 函数扩展技术（Function “Stretching” Technique）

一般的函数优化问题均可以用下式进行描述：

$$\begin{aligned} &\min f(X) \\ &X \in S \subseteq R^n; S = [a_i, b_i]_{i=1}^n \end{aligned} \tag{8.1}$$

其中，$S \subseteq R^n$ 称为搜索空间，$f(X)$ 称为目标函数。具有上述形式的优化问题均可称为极小化问题。

定义 8.1 对极小化问题（8.1 式），设 $x' \in S$，若存在 $\delta > 0$，使得当 $x \in S \cap \{x; \| x - x' \| < \delta\}$ 时，总有下式成立

$$f(x') \le f(x) \tag{8.2}$$

则称 x' 是 $f(x)$ 在 S 上的一个局部极小点。若上式中的严格不等式成立，则 x' 与 $f(x')$ 分别称为严格局部极小点(局部最优解)和严格局部极小值（局部最优解）。

定义 8.2 对极小化问题（8.1 式），若存在 $x^* \in S$，使得对任意 $x \in S$ 都有下式成立

$$f(x^*) \le f(x) \tag{8.3}$$

则称 x^* 是 $f(x)$ 在 S 上的一个全局极小点。若上式中的严格不等式成立，则 x^* 与 $f(x^*)$ 分别称为严格全局极小点(全局最优解)和严格全局极小值（全局最优值）。

8.1 式所描述的优化问题中，有极大一部分问题其目标函数往往存在着符合 8.2 与 8.3 定义式的多个局部极值或多个全局极值，即多峰、多极值，并且具有很复杂的函数形态，其最优解或者被众多的局部极值所环绕，或者是多个最优解与局部极值交互错杂地分布在解空间中。对于这类问题的求解，即便是全局优化算法，也不可避免得会以较大的概率收敛于某些局部极值，出现所谓的“早熟”现象。要想有效地解决多模态函数的全局优化问题，必须使得优化问题在搜索过程中能够及时从各个局部极值点逃逸出来，从而保证算法以较大的机率收敛于问题的全局最优解。

基于这种考虑，Vrahatis MN 提出了一种函数“ Stretching ”技术，用于帮助提高优化算法的搜索效率及全局收敛能力。

函数“ Stretching ”技术的实质是借助于问题的局部极值点信息，对原始目标函数进行一种“ 拉伸 ” 变换，其目的是在优化算法的实施过程中，不断缩小目标函数的极值范围，从而降低优化问题的搜索难度。具体的变换定义如下：

$$G(x) = f(x) + \gamma_1 \| x - x' \| (sign(f(x) - f(x')) + 1) \tag{8.4}$$

$$H(x) = G(x) + \gamma_2 \frac{sign(f(x) - f(x')) + 1}{\tanh(\mu(G(x) - G(x')))} \tag{8.5}$$

上述两种变换式中，x' 是目标函数的局部极值点，符合定义式（8.2）；γ_1、γ_2 和 μ 是三个任意的正数常量；$sign(\bullet)$ 为常见的符号函数：

$$sign(x) = \begin{cases} 1, & \text{当 } x > 0 \\ 0, & \text{当 } x = 0 \\ -1, & \text{当 } x < 0 \end{cases} \tag{8.6}$$

也可采用人工神经网络中常用的线性激励函数 sigmoid 函数来近似计算：

$$sign(x) \approx \log sig(x) = \frac{2}{1 + \exp(-\lambda x)} - 1 \approx \tanh(\frac{\lambda x}{2}) \tag{8.7}$$

通常在使用函数“ Stretching ”技术之前，首先要通过优化算法按照常规的方法对其局部极值点进行搜索。当算法探测到某一局部极值点 x' 之后，再通过 8.4 与 8.5 式，对目标函数进行两次变换。在整个变换过程中，函数的解空间根据已搜索到的局部极值 $f(x')$ 信息，被划分为两部分来考虑。一部分为区域 $S_1 = \{x \mid f(x) < f(x')\}$，另一部分为区域 $S_2 = \{x \mid f(x) > f(x')\}$。分析 8.4 与 8.5 式可知，区域 S_1 在整个变换过程中，始终保持原始目标函数的形态特征不变，即对于任意 $x \in S_1$，均有 $G(x) = f(x) = H(x)$，同样原始目标函数在区域 S_1 中的极值点（包括全局极小点在内）也始终保留不变。区域 S_2 则不同，在两次变换中，目标函数经历了不同程度的拉伸。实施 8.4 式变换后，目标函数由 $f(x)$ 变换为 $G(x)$，此时 $G(x) = f(x) + 2\gamma_1 \| x - x' \|$，相当于原始目标函数在区域 S_2 中的每一函数值均向上进行了拉伸，并且点 x 越远离局部极值点 x'，则其函数值被拉伸的幅度越大。此次拉伸的结果，使得区域 S_2 中目标函数的形态变得平缓，并且其中所包含的部分极值也由此转变为非极值，这意味着从搜索空间剔除掉了函数值高于 $f(x')$ 的部分极值，从而降低了后续搜索的难度。区域 S_2 中的目标函数经 8.5 式实施第二次变换后，由 $G(x)$ 变换为

$H(x)$，此时 $H(x)=G(x)+\dfrac{2\gamma_2}{\tanh(\mu(G(x)-G(x')))}$。很显然，$H(x)$ 是将目标函数 $f(x)$ 在 $G(x)$ 的基础上进一步向上拉伸，其中距离极值点 x' 越近且函数值越逼近 $f(x')$ 的点，其拉伸程度越大，因此第二次变换的实质是将局部极值点 x' 及其邻域范围内的点整体向上拉伸。既然点 x' 被探测为目标函数的局部极值，则其邻域范围内的点往往与其具有接近的特性，因此第二次变换的实施是有一定意义的，可以进一步缩小后续的搜索空间。

下面将函数“拉伸”技术作用于第六章所介绍的多模态优化函数 F6，Levy No.5 函数。此函数是一个著名的两维测试函数，具有 760 个局部极值点。众多的局部极值点使得各种优化算法很难搜索到全局最优解。为方便起见，现重新表述如下：

$$f_6(X)=\sum_{i=1}^{5}[i\cos((i-1)x_1+i)]\sum_{j=1}^{5}[j\cos((j+1)x_2+j)]+(x_1+1.42513)^2+(x_2+0.80032)^2 \tag{8.8}$$

图 8.1~8.3 分别描述了 Levy No.5 函数在拉伸变换前后函数的具体形态。

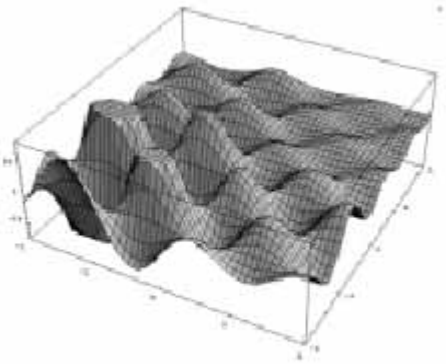

图8.1 Levy No.5函数原始空间曲面图

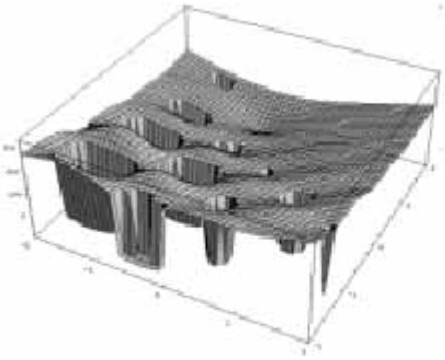

图8.2 Levy No.5函数第一次变换后的空间曲面图

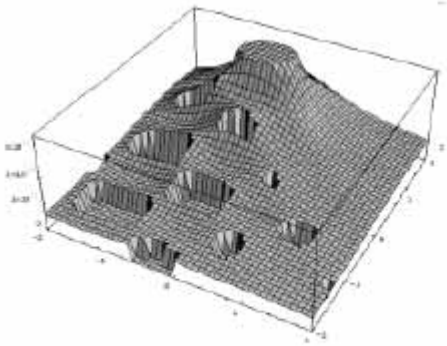

图8.3 Levy No.5函数第二次变换后的空间曲面图

经过上述分析可知，Vrahatis 所提出的函数“Stretching”技术，可以有效地降低目标函数的复杂性。把此项技术与全局优化算法相结合，利用优化算法不断去搜索问题的局部极值信息，每当探测到问题的一个局部极小点，即对函数进行两次拉伸变换，将局部极小点以及劣于局部极小点的函数极值从搜索空间剔除掉，而优于局部极小点的解以及全局最优点却始终保持不变。因此，

函数拉伸技术的使用，并没有改变搜索目标，只是有效地降低后续搜索过程的难度，提高了优化算法的搜索效率。

为了有效地求解多模态复杂函数优化问题，K.E.Parsopoulos 等人将函数“ Stretching ”技术引入到 PSO 算法中，形成了一种高效的全局优化算法——“ Stretched PSO ”(简称 SPSO)。下面所介绍的内容，即是 K.E.PARSOPOULOS 等人利用 SPSO 求解复杂函数优化问题取得的相关成果。

8.1.2 Stretched PSO 算法(SPSO)

从上述的分析可知，函数扩展技术的有效实施，必须借助于目标函数的局部极值信息。在利用 SPSO 优化问题之前，首先要利用基本 PSO 模型对原始目标函数进行优化，探测其局部极小点。当算法收敛于某一局部极值点后，再根据 8.4 与 8.5 式对目标函数进行两次变换，随后，对扩展以后的函数继续利用 PSO 算法进行优化。重复上述操作，直至满足问题全局收敛的终止条件。SPSO 算法的伪码描述如下：

PROCEDURE SPSO
begin
t=0
随机初始化微粒群
while (终止条件未满足)
begin
对所有微粒进行一次 PSO 操作
利用 $f(x)$ 对所有微粒进行评价
if 找到一局部极值点 x'，then
令 $f(x) = H(x)$
t=t+1
end while
end begin

K.E.PARSOPOULOS 利用 SPSO 对下述的复杂优化函数进行了求解，发现 SPSO 的全局收敛性能比起基本的 PSO 算法有了显著的提高。

表 8.1 实验中不同优化函数的变量取值及微粒群规模

优化问题	维数	微粒群规模	变量取值范围
Levy No.3	2	20	[-10,10]
Levy No.5	2	20	[-2,2]
Levy No.8	3	20	[-5,5]
Freud-Roth	2	20	[-5,5]
Goldst-Price	2	20	[-2,2]
Corana 4D	4	80	[-1,1]
XOR	9	80	[-1,1]

表中前六个优化函数具体见第六章，XOR 优化问题函数[Vrahatis 2000]如下：

$$
\begin{aligned}
f(x) = [1+\exp(-\frac{x_7}{1+\exp(-x_1-x_2-x_5)} - \frac{x_8}{1+\exp(-x_3-x_4-x_6)} - x_9)]^{-2} \\
+[1+\exp(-\frac{x_7}{1+\exp(-x_5)} - \frac{x_8}{1+\exp(-x_6)} - x_9)]^{-2} \\
+[1-\{1+\exp(-\frac{x_7}{1+\exp(-x_1-x_5)} - \frac{x_8}{1+\exp(-x_3-x_6)} - x_9)\}^{-1}]^2 \\
+[1-\{1+\exp(-\frac{x_7}{1+\exp(-x_2-x_5)} - \frac{x_8}{1+\exp(-x_4-x_6)} - x_9)\}^{-1}]^2
\end{aligned}
\tag{8.9}
$$

实验中采用带有惯性权重的 PSO 计算模型，主要的实验参数如下：$\gamma_1 = 5000$，$\gamma_2 = 0.5$，$\mu = 10^{-10}$；$c_1 = c_2 = 0.5$，惯性权重系数W 随时间递减，从 1.0 减小至 0.4。

下表 8.2 是相关的实验数据及结果[Parsopoulos KE 2000c]。

表 8.2 不同算法对各类优化问题的求解结果

Test Problem	“Stretching”applied			PSO			SPSO		
	Mean.	St.D	Succ.	Mean	St.D.	Succ.	Mean	St.D.	Succ.
Levy No.3	15246.6	6027.3	15%	5530.5	6758.0	85%	6988.0	7405.5	100%
Levy No.5	3854.2	1630.1	7 %	1049.4	235.1	93%	1245.8	854.2	100%
Levy No.8	0	0	0%	509.6	253.3	100%	509.6	253.2	100%
Freud.-Roth	3615.0	156.1	40%	1543.3	268.1	60%	2372.0	1092.1	100%
Goldst.-Price	17420.0	3236.56	5%	1080.0	225.6	95%	1897.0	3660.3	100%
Corana 2D	0	0	0%	1409.6	392.8	100%	1409.6	395.8	100%
Corana 4D	13704.6	7433.5	26%	2563.2	677.5	74%	5460.0	6183.8	100%
XOR	29328.6	15504.2	23%	1459.7	1143.1	77%	7869.6	13905.4	100%

从上表可以看出，利用单纯的函数扩展技术去求解复杂优化函数，其成功率很低；但将此技术引入 PSO 算法中，却能大大提高算法的搜索性能，对于各类复杂优化问题，均能可靠的收敛到其全局最优解。

8.2 基于 PSO 算法求解多目标优化问题

8.2.1 多目标优化问题的基本概念

在实际的经济、生产、工程应用领域中普遍存在着对多个目标的方案、计划以及设计的决策问题。在解决这类问题时，决策者往往要综合考虑现实世界中各种因素的制约，寻求满足多个目标的最佳设计方案，这就是所谓的多目标优化问题（Multi-Objective Optimization 简称 MO）。

假定某一决策过程中，需要同时考察 k 个目标，且要求所有目标函数在满足约束的条件下越小越好，则这样的优化问题可以表述如下：

$$\begin{gathered}\min_{X\in R} F(X)\\ F(X)=(f_1(X),\ f_2(X),\ \cdots,\ f_k(X))^T\\ R=\{X\mid g(X)\le 0\},\quad g(X)=(g_1(X),g_2(X),\cdots,g_m(X))^T\\ X=(x_1,x_2,\cdots,x_n)^T,\quad X\in R\subset E^n\end{gathered}\tag{8.10}$$

其中 $F(X)$ 为优化目标向量，$g(X)$ 为约束向量，X 为决策变量。

对于 8.10 式所描述多目标优化问题来说，其所包含的不同目标函数之间往往存在着一定的矛盾冲突，因此在求解过程中，很难在问题的约束集合 R 中找到一个解向量，能够使 k 个目标函数同时达到最小。也就是说，多目标优化问题中的多个目标几乎不可能在同一解上同时达到最优。因此多目标优化问题的解往往不是单一的，而是一组解的集合。

所有的优化问题在求解过程中，必须对解的质量进行评价。对于单目标优化问题，只要比较任意两个解对应的目标函数值就可以区分它们的优劣；而对于多目标优化问题，则很难通过简单的比较来确定谁优谁劣。法国经济学家 V.Pareto 最早研究经济领域内的多目标优化问题，他的理论被称为 Pareto 最优性理论。Pareto 最优性理论提出了一种 Pareto 支配（Pareto Dominance）原则，作为判断多目标优化问题解优劣的依据，在此基础上定义了 Pareto 最优集（Pareto-optimal set）的概念。

定义一：对于最小化 MO 问题，存在两个目标向量 $u=(u_1,\cdots,u_k)^T$ 与 $v=(v_1,\cdots,v_k)^T$，若目标向量 u 支配 v，当且仅当存在着决策变量使得下式成立：

$$u_i\le v_i,\quad i=1,\cdots,k\tag{8.11}$$

其中至少有一个严格不等式成立。

定义二：对于最小化 MO 问题，存在目标向量 $u=(f_1(X_u),f_2(X_u),\cdots,f_k(X_u))^T$，其中 $X_u\in R$ 为决策变量，X_u 若为 Pareto 最优解则：

当且仅当，问题的可行域内不存在决策变量 X_v，使得目标向量 $v=(f_1(X_v),f_2(X_v),\cdots,f_k(X_v))^T$ 支配目标向量 $u=(f_1(X_u),f_2(X_u),\cdots,f_k(X_u))^T$。

所有满足上述定义二的解的集合，构成了多目标优化问题的 Pareto 最优解集，又称为非劣解集（Noninferior Set）；集合中的每一个解都是一个 Pareto 最优解，又称为非劣解（Noninferior Solution）或有效解。

假若记 MO 问题的最优解集为 P^*，则所有 Pareto 最优解所对应的目标向量构成了问题的 Pareto 前沿（矩阵），具体形式如下：

$$PF^*=\{(f_1(x),\ldots,f_k(x))\mid x\in P^*\}\tag{8.12}$$

定义三：一个 Pareto 前沿被称为凸集，当且仅当

$\forall u,v\in PF^*,\quad \lambda\in(0,1),\quad \exists w\in PF^*$ 满足 $\lambda\|u\|+(1-\lambda)\|v\|\ge\|w\|$。

定义四：一个 Pareto 前沿被称为凹集，当且仅当

$\forall u,v\in PF^*,\quad \lambda\in(0,1),\quad \exists w\in PF^*$ 满足 $\lambda\|u\|+(1-\lambda)\|v\|\le\|w\|$。

多目标优化问题的 Pareto 前沿或为凸集，或为凹集，在某些复杂情况下，还可能是半凸、半凹或

者不连续。Pareto 前沿的复杂性，也增加了多目标优化问题的求解难度。

对于多目标问题的求解，传统的方法是将多个目标根据某种效用函数合成单一目标，利用单目标函数的优化方法去求解，如目标加权法、功效系数法、层次优化法、目标规划法等等。但大多数情况下，在优化之前这种效用函数是难以确知的，并且每次优化过程中只能找到一个最优解，如果没有足够的先验知识，决策者将难以判断所求解的质量。因此，对于多目标问题来说，往往需要给决策者提供符合条件的多种方案以便其作出合理的最终选择。与传统的方法相比，进化计算在求解多目标优化问题中具有很大的优越性[Zitzler,1999]。作为一种随机搜索算法，进化计算通过代表整个解集的种群进化，以内在并行的方式搜索多个非劣解，能够给决策者提供问题的 Pareto 最优解集以便其做出合理的选择。

PSO 算法是一种简单有效的仿生算法，其基于多点并行搜索的特性，使它同样适用于求解多目标优化问题。下面的内容将围绕 K.E.Parsopoulos 与 M.N.Vrahatis 的研究成果[Parsopoulos and Vrahatis,2002]，介绍 PSO 在求解 MO 问题中的应用。

8.2.2 求解多目标优化问题的 PSO 算法

1、基于目标加权法的 PSO 算法（PSO With Weighted Aggregation，WAPSO）

目标加权法是求解多目标优化问题最常用的一种方法，其基本思想是给予问题中的每一目标一个权重，将所有目标分量乘上各自相应的权重系数然后再加合起来构成一个新的目标函数，采用单目标优化方法去求解。常规的目标加权法如下：

$$F(X)=\sum_{i=1}^{k} w_i f_i(X) \tag{8.13}$$

其中 w_i $(i=1,\cdots,k)$，是目标 $f_i(X)$ 的非负权重系数，并且满足 $\sum_{i=1}^{k} w_i=1$。$F(X)$ 通常被称为新的效用函数。

传统目标加权法（Conventional Weighted Aggregation，简称 CWA）在形成单一的效用函数时[Jin et al., 2001]，往往采用一组固定的权重系数。这种方法具有两种缺陷：其一，为了找到合理的 Pareto 最优解，需要足够的先验知识为不同目标确定最佳的权重系数。事实上，在问题未被优化之前，这种先验知识往往是缺乏的，而权重系数是否能够反映各目标在实际问题中的作用大小，直接影响着优化结果的好坏；其二，采用单目标优化方法去优化新的效用函数，每次优化只能获得一个 Pareto 最优解，如果不具备足够的关于问题的先验知识，即使优化过程很成功，也难以判断优化结果的可靠性及最优性。因此为了获取问题的一个 Pareto 最优解集，以便于决策者从中进行更合理的选择，必须花费大量的时间及计算代价。但对于决大多数 MO 优化问题来说，在实际应用中这一点是不可取的。

为了减小传统目标加权法中固定权重系数对优化结果的不利影响，许多算法采用在优化过程中动态调整权重系数的方法，例如有名的“ Bang-Bang ”加权算法（Bang-Bang Weighted Aggregation，简称 BWA）[Jin et al. 2001]。BWA 是针对两目标 MO 问题而设计的，其中在构造新的效用函数时，目标的权重系数按如下方式进行调整：

$$\begin{aligned} w_1(t) &= sign(\sin(s\pi t/R)) \\ w_2(t) &= 1-w_1 \end{aligned} \tag{8.14}$$

其中 t 为优化操作代数，R 是权重系数的变化频率，通常取一常数。

由 8.14 式中可知，权重系数 w_1 的变化由符号函数 $sign(\bullet)$ 来确定，其值在 ± 1 之间跳变；而 w_2 的值在 0、2 之间跳变。为了减缓权重系数在两值间的剧变，8.14 式可以调整如下：

$$w_1(t) = |\sin(s\pi t / R)|$$
$$w_2(t) = 1 - w_1 \tag{8.15}$$

可以看出，8.15 式中的权重系数以正弦函数形态缓慢波动，这种动态调整方法使得算法更适于求解 Pareto 矩阵为凸集的 MO 问题，此方法被称为动态加权法（Dynamic Weighted Aggregation，简称 DWA）[Jin et al. 2001]。

K.E.Parsopoulos 与 M.N.Vrahatis 将 PSO 算法用于求解 MO 问题,在优化过程中采用上述三种方法构造新的效用函数对微粒进行评价，形成求解 MO 问题的不同 WAPSO 算法，并将算法用于求解多个 MO 测试问题（具体见第六章）[Parsopoulos and Vrahatis,2002]。结果表明，PSO 算法同样是一种有效的求解 MO 问题的方法，能够以较低的计算代价和较高的收敛率搜索到 MO 问题的多个 Pareto 最优解，其中采用 DWA 的 PSO 算法搜索性能最佳。

2、基于向量求值的 PSO 算法（Vector Evaluated PSO，VEPSO）

VEPSO 的算法思想源于 Schaffer 提出的用于解决 MO 问题的一种基于向量求值的遗传算法（Vector Evaluated Genetic Algorithm，简称 VEGA）[Schaffer1985]。VEGA 算法采用多子群体的进化方式，并对简单遗传算法的选择方法进行了调整。在算法执行的每一进化代中，所有目标函数被视为单目标优化问题，分别对所有个体进行评价，独立计算个体的适应度，然后利用适应度比例法针对每一目标函数选择一定数目的个体构成一个子群体。假设问题包含 k 个目标函数，则需要形成 k 个子种群，每一子群体的个体均代表某一目标函数的一组较优解。由于每个子群体只着重考虑对某一目标函数分量的优化，而不是按照 Pareto 原则进行个体的选择，因此 VEGA 算法被视为一种非 Pareto 方法。当生成新一代群体时，需要将所有子群体中的个体混合起来，再从中选取个体进行交叉、变异等遗传操作。

K.E.Parsopoulos 与 M.N.Vrahati 借鉴了 VEGA 算法的思想，将基于目标向量的个体评价方法与 PSO 算法相结合，形成了 VEPSO 算法[[Parsopoulos and Vrahatis 2002]]。

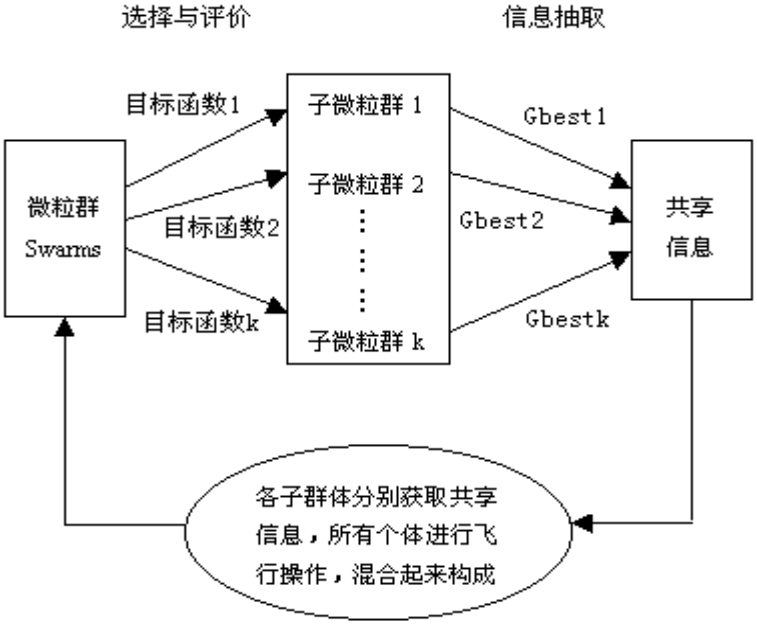

图 8.4　VEPSO 算法框架

下面具体以两目标优化问题的求解为例，详细介绍 VEPSO 算法的实现。假设有一两目标优

化向量 $F(X)=(f_1(X),f_2(X))^T$，要求目标向量越小越好。则 VEPSO 算法的具体步骤如下：

(1) 初始化群体。
在解空间随机产生 N 个微粒（N 为微粒群规模），每个微粒由一向量来表示，每一向量的分量均按均匀分布随机产生。

(2) 形成子微粒群。
假设 Nsg 代表子微粒群规模，则 $Nsg=N/k$，对于两目标函数来说，需要产生两个子微粒群，每一子微粒群中含有 $\frac{N}{2}$ 个微粒。根据单目标优化函数的个体评价方法，利用目标函数 $f_1(X)$ 与 $f_2(X)$ 分别对所有微粒进行评价。基于 $f_1(X)$ 计算所有微粒的函数值，从中选择 Nsg 个较优微粒构成子微粒群 1；同理，基于 $f_2(X)$ 计算所有微粒的函数值，从中选择 Nsg 个较优微粒构成子微粒群 2。

(3) 抽取共享信息。
对于每一微粒群，确定其中的最佳微粒，并将其作为微粒群的社会共享信息。

(4) 所有微粒进行飞行操作。
不同子微粒群中的微粒，均利用来自其它子微粒群的社会共享信息来调整自己的飞行速度。当计算子微粒群 1 中所有微粒的飞行速度时，将子微粒群 2 的最佳微粒信息作为其每一微粒所获得的全局最佳位置 Gbest1；而计算子微粒群 2 中所有微粒的飞行速度时，则将子微粒群 1 中的最佳微粒信息作为其每一微粒所获得的全局最佳位置 Gbest2。之后，按照微粒群算法飞行模型，计算每一微粒的飞行速度，并更新其位置。更新后的所有微粒重新组合成新的微粒群。

(5) 重复上述步骤 2~4，直至满足终止条件。最后，从两个子微粒群中选取一组 Pareto 最优解构成 Pareto 最优解集，作为优化结果输出。

从上述的实现过程可知，VEPSO 算法首先利用单目标优化函数的个体评价方法对微粒群中的所有微粒进行评价，在此基础上形成不同的子微粒群，每一子微粒群中的微粒均是某一目标函数的较优解，同时微粒在飞行过程中接受来自其他子微粒群的社会影响，从而使微粒又朝着满足其它目标函数分量的方向飞行，逐渐满足更多的目标函数分量，不断朝着问题的 Pareto 最优解方向飞行。

K.E.Parsopoulos 与 M.N.Vrahati 利用 VEPSO 算法求解了多个两目标优化函数，实验结果表明：VEPSO 算法是一种更加灵活更加有效的求解 MO 问题的算法，能很好的捕获 MO 问题的 Pareto 前沿的形态特征，并以较小的计算代价搜索到问题的 Pareto 最优解集，从而也说明了微粒群算法同样是一种性能良好的求解多目标优化问题的随机算法。

8.3 用 PSO 算法求解约束优化问题

8.3.1 约束优化问题(Constrained Optimization，CO)

在科学与工程领域中，许多极值问题的求解往往受到各种现实因素的制约，这些制约通常由一系列的约束条件来描述。求解带有约束条件的极值问题则被称为约束优化问题，具体可由下述一般形式的非线性规划来表示：

$$\begin{aligned} &\min f(x) && x \in E^n \\ s.t. \quad &g_i(x) \geq 0 && i = 1, 2, \cdots, m \end{aligned} \tag{8.16}$$

其中 $g_i(x)$ 表示问题的约束条件。

由于约束条件的存在，使得约束极值问题的求解要比无约束优化问题的求解复杂、困难得多。对于约束极小化问题来说，不仅要使目标函数值在迭代过程中不断减小，而且还要注意解的可行性。为了简化 CO 问题的寻优过程，通常可采用如下的思路去构造算法：将约束优化问题转化为无约束优化问题、将非线性规划问题转化为线性规划问题、将复杂问题转化为简单问题。

求解约束极值问题的传统方法有可行方向法（Feasible Direction）、梯度投影法(Gradient Projection Method)、约束集法(Active Set Method)、罚函数法 Penalty Function Method)等等。这些方法各有不同的适用范围及局限性，其中大多数方法需要借助问题的梯度信息，要求目标函数或约束条件连续可微，并且常常为满足严格的可行性要求而付出很大的计算代价，最终也往往只能求解到问题的局部极值点而以失败告终。

为了有效地求解约束优化问题，人们逐渐将目光转向随机搜索算法，其中以进化计算（EA）为代表的仿生随机算法，以其较强的求解力、通用性日益受到广大学者的青睐，并日渐成为求解约束优化问题的重要工具。与传统的优化算法相比，进化计算具有许多不可比拟的优越性，其中最主要的一点是求解过程不依赖目标函数的解析性质，同时能通过对问题解空间的多点并行搜索，以较大概率收敛于全局最优解。另外进化计算在处理约束条件时比传统的搜索算法要灵活得多，不仅可借助进化操作对传统的罚函数法进行改进使之更加有效的处理约束条件，也可以通过特定的编码或遗传算子保证后代个体落在可行点集内，或者在每一演化代通过特定的修正算子对后代个体进行修正以满足约束条件，而不需要借助于问题的梯度信息。

微粒群算法是一种简单有效的随机算法，与其它进化计算方法比较起来，它的求解过程更加简单易行，所付出的计算代价更小，因此也被视为求解 CO 问题的可行方法。

8.3.2 非固定多段映射罚函数法

罚函数法是应用最为广泛的一种处理约束的方法，其基本思想是通过序列无约束最小化技术（Sequential Unconstrained Minimization Technique），将约束优化问题转化为一系列无约束优化问题进行求解，应用起来比较方便。具体的方法是在目标函数中加上一个能够反映点是否满足约束的惩罚项，从而构成一个无约束的广义目标函数，使得算法在惩罚项的作用下找到问题的最优解。

通常，所构造的广义目标函数具有如下形式：

$$F(x) = f(x) + h(k)H(x), \quad x \in S \subset R^n \tag{8.17}$$

其中 $f(x)$ 代表原目标函数；$h(k)H(x)$ 称为惩罚项，$h(k)$ 表示惩罚力度，$H(x)$ 为惩罚因子。在上式中，如果 $h(k)$ 在求解 CO 问题的整个过程中固定不变，则称之为固定罚函数法（Stationary PFM，SPFM）；相反则称之为非固定罚函数法（Non-Stationary PFM，NSPFM）。通常 NSPFM 对 CO 问题的求解结果总是要优于 SPFM。

Parsopoulos 与 Vrahatis[ ]较早尝试用 PSO 算法去求解约束优化问题。在求解过程中，同样采用上述的罚函数法去处理问题的约束条件，其中 $h(k)$ 可以被动态调整，$H(x)$ 具体定义如下：

$$H(x) = \sum_{i=1}^{m} \theta(q_i(x)) q_i(x)^{\gamma(q_i(x))} \tag{8.18}$$

上式中，$q_i(x)=\max\{0, g_i(x)\}, i,\ldots,m.$表示解对约束的违背程度；$\theta(q_i(x))$是一个多段映射函数[]；$\gamma(q_i(x))$表示惩罚函数的强度。上述方法被称为非固定多段映射罚函数法(Non-stationary multi-stage assignment Penalty Function Method)。

8.3.3 求解 CO 的 PSO 算法

Parsopoulos 与 Vrahatis 将 PSO 算法用于求解 CO 问题，具体的 PSO 计算模型如下：

$$\begin{aligned} v_{id}(t+1) &= \chi(\omega v_{id}(t)+c_1 r_1\left(P_{id}(t)-x_{id}(t)\right)+c_2 r_2\left(P_{gd}(t)-x_{id}(t)\right)\right) \\ x_{id}(t+1) &= x(t)+v_{id}(t+1) \end{aligned} \tag{8.19}$$

其中，χ为收缩因子，ω为权重系数。

为了研究 PSO 算法求解 CO 问题的性能，Parsopoulos 与 Vrahatis 用上述 PSO 模型求解了大量的 Benchmarks 测试函数，其中对约束的处理方法采用上述的非固定多段映射罚函数法。

测试问题一：
$$f_1(x)=(x_1-2)^2+(x_2-1)^2$$
$$\textit{subject to}: \quad \begin{aligned} x_1 &= 2x_2-1 \\ x_1^2/4+x_2^2-1 &\le 0 \end{aligned}$$

其最优解 $f^*=1.3934651$。

测试问题二：
$$f_2(x)=(x_1-10)^3+(x_2-20)^3$$
$$\textit{subject to}: \quad \begin{aligned} 100-(x_1-5)^2-(x_2-5)^2 &\le 0 \\ (x_1-6)^2+(x_2-5)^2-82.81 &\le 0 \\ 13\le x_1\le 100,\ \ 0 &\le x_2 \le 100 \end{aligned}$$

其最优解 $f^*=-6961.81381$。

测试问题三：
$$\begin{aligned} f_3(x)=&(x_1-10)^2+5(x_2-12)^2+x_3^4+3(x_4-11)^2 \\ &+10x_5^6+7x_6^2+x_7^4-4x_6x_7-10x_6-8x_7 \end{aligned}$$
$$\textit{subject to}: \quad \begin{aligned} -127+2x_1^2+3x_2^4+x_3+4x_4^2+5x_5 &\le 0 \\ -282+7x_1+3x_2+10x_3^2+x_4-x_5 &\le 0 \\ -196+23x_1+x_2^2+6x_6^2-8x_7 &\le 0 \\ 4x_1^2+x_2^2-3x_1x_2+2x_3^2+5x_6-11x_7 &\le 0 \\ -10\le x_i\le 10,\ \ i=1,\cdots,7 & \end{aligned}$$

其最优解 $f^*=680.630057$

测试问题四：$f_4(x)=5.3578547x_3^2+0.8356891x_1x_5+37.293239x_1-40792.141$

$$
\begin{aligned}
& 0 \le 85.334407 + 0.0056858T_1 + T_2 x_1 x_4 - 0.0022053 x_3 x_5 \le 92 \\
\textit{subject to}: \quad & 90 \le 80.51249 + 0.0071317 x_2 x_5 + 0.0029955 x_1 x_2 + 0.0021813 x_3^2 \le 110 \\
& 20 \le 9.300961 + 0.0047026 x_3 x_5 + 0.0012547 x_1 x_3 + 0.0019085 x_3 x_4 \le 25 \\
& 78 \le x_1 \le 102, \quad 33 \le x_2 \le 45, \quad 27 \le x_i \le 45, i = 3,4,5
\end{aligned}
$$

其中，$T_1 = x_2 x_5$， $T_2 = 0.0006262$ ；最优解 $f^* = -30665.538$。

测试问题五：形式与测试问题四相同，只是$T_1 = x_2 x_3$， $T_2 = 0.00026$，最优解未知。

测试问题六： $f_6(x) = -10.5x_1 - 7.5x_2 - 3.5x_3 - 2.5x_4 - 1.5x_5 - 10y - 0.5\sum_{i=1}^{5} x_i^2$

$$
\begin{aligned}
\textit{subject to}: \quad & 6x_1 + 3x_2 + 3x_3 + 2x_4 + x_5 - 6.5 \le 0 \\
& 10x_1 + 10x_3 + y \le 20 \\
& 0 \le x_i \le 1, \quad i = 1,\ldots,5; \quad y \ge 0
\end{aligned}
$$

最优解 $f^* = -213.0$。

结合非固定多段映射罚函数法，Parsopoulos 与 Vrahatis 将 PSO 算法用于求解上述的六个 CO 问题。其中罚函数中的$\gamma(.), \theta(.), h(.)$ 函数具体如下：

$$
\gamma(q_i(x)) = \begin{cases} 1, & q_i(x) < 1 \\ 2, & q_i(x) \ge 1 \end{cases} \tag{8.20}
$$

$$
\theta(q_i(x)) = \begin{cases} 10 & q_i(x) < 0.001 \\ 20 & q_i(x) \le 0.1 \\ 100 & q_i(x) \le 1 \\ 300 & else \end{cases} \tag{8.21}
$$

对于测试问题一， $h(k) = \sqrt{k}$ ；其它测试问题中， $h(k) = k\sqrt{k}$ 。

实验中采用的控制参数如下：$c_1 = c_2 = 2$ ， $\chi = 0.73$ ，$V_{\max} = 4$ ，ω 从 1.2 逐渐减小至 0.1，微粒群规模为 100。

一系列的实验结果表明，PSO 算法同样适用于求解约束优化问题，并且在某些问题的求解上，性能要优于 EA。

8.4 在最大最小优化问题中的应用

首先给出最大最小问题的一般情形：

$$
\min_x f(x)
$$

$$
f(x) = \max_{i=1,2,\ldots,m} f_i(x) \text{，且 } f_i(x): S \subset R^n \to R \tag{8.22}
$$

一般而言，求解最大最小问题使用光滑化技术，本文所使用的是 Xu S 于 2001 年[Xu S2001]

提出了一种新的光滑化技术，即提出下面形式的光滑函数

$$f(x,\mu)=\mu\ln\sum_{i=1}^{m}\exp(\frac{f_i(x)}{\mu}) \tag{8.23}$$

显然，当 $\mu>0$ 时，我们有

$$f(x)\le f(x,\mu)\le f(x)+\mu\ln m \tag{8.24}$$

定义点 x*为一个稳定点，如果存在向量 $y^*=(y_1^*,y_2^*,...,y_m^*)$，使得下式成立

$$\sum_{j=1}^{m}y_j^*\nabla f_j(x^*)=0 \tag{8.25}$$

而且 $y^*=(y_1^*,y_2^*,...,y_m^*)$ 满足

$$y_j^*\ge 0,\ j=1,2,...,m$$

$$\sum_{j=1}^{m}y_j^*=1$$

$$y_j^*=0, if\ (f_j(x^*)<\max\{f_1(x^*),f_2(x^*),...,f_m(x^*)\}) \tag{8.26}$$

对于光滑函数而言，我们给出下面两个简单的定理。

定理 1、若 x*是(8.22)的局部最优解，则它是一个稳定点，且满足(8.25)、(8.26)。反之，设 $f(x)$ 为凸函数，如果 x*为稳定点，则它为最大最小问题的全局最优解。

定理 2、设 $f_i(x),i=1,2,...,m$ 二次连续可微， $f(x),f(x,\mu)$ 分别满足(8.23)和(8.24)，则

1） 当 μ 给定时， $f(x,\mu)$ 为增函数；

2） 对于所有的 $\mu>0$，$f(x,\mu)$ 二次连续可微，且

$$\nabla_x f(x,\mu)=\sum_{i=1}^{m}\lambda_i(x,\mu)\nabla f_i(x) \tag{8.27}$$

$$\nabla_x^2 f(x,\mu)=\sum_{i=1}^{m}(\lambda_i(x,\mu)\nabla^2 f_i(x)+\frac{1}{\mu}\lambda_i(x,\mu)\nabla f_i(x)\nabla f_i(x)^T)$$

$$-\frac{1}{\mu}(\sum_{i=1}^{m}\lambda_i(x,\mu)\nabla f_i(x))(\sum_{i=1}^{m}\lambda_i(x,\mu)\nabla f_i(x))^T \tag{8.28}$$

其中，

$$\lambda_i(x,\mu)=\frac{\exp(\frac{f_i(x)}{\mu})}{\sum_{j=1}^{m}\exp(\frac{f_j(x)}{\mu})}\in(0,1) \tag{8.29}$$

且

$$\sum_{j=1}^{m}\lambda_j(x,\mu)=1 \tag{8.30}$$

3） 对于所有的 $\mu>0$ ， $0\le f_{\mu}^{'}(x,\mu)\le \ln m$

当我们考虑非线性规划问题时，可以将其转化为最大最小问题，即对于非线性规划问题：

$$\begin{aligned}&\min F(x)\\&s.t.\ g_i(x)\ge 0, i=1,2,...,m\end{aligned} \tag{8.31}$$

转化后的形式为

$$\begin{aligned}&\min_x f(x)\\&f(x)=\max_{1\le i\le m} f_i(x)\\&f_1(x)=F(x)\\&f_i(x)=F(x)-\alpha_i g_i(x),\alpha_i>0,i=1,2,...,m\end{aligned} \tag{8.32}$$

为了验证算法的有效性，对下面 13 个测试函数进行了测试，其中，微粒群算法的最大迭代次数为 500，精度为 10^{-8} ，惯性因子 w 从 1.2 动态降到 0.4，$c_1=c_2=0.5$ ，每个测试函数运行 25 次。实验结果利用平均收敛率、迭代次数的均值与函数适应值的均值表示。

测试函数 1、

$$\min_x F_1(x)$$

其中

$$\begin{aligned}&F_1(x)=\max\{f_i(x)\}, i=1,2,3\\&f_1(x)=x_1^2+x_2^4\\&f_2(x)=(2-x_1)^2+(2-x_2)^2\\&f_3(x)=2e^{-x_1+x_2}\end{aligned}$$

测试函数 2、

$$\min_x F_2(x)$$

其中

$$\begin{aligned}&F_2(x)=\max\{f_i(x)\}, i=1,2,3\\&f_1(x)=x_1^4+x_2^2\\&f_2(x)=(2-x_1)^2+(2-x_2)^2\\&f_3(x)=2e^{-x_1+x_2}\end{aligned}$$

算法定义域为 $[-5,5]^2$ ，且整个种群含有 20 个微粒，结果见表 8.3。

表 8.3、测试函数 f_1, f_2 结果

函数	收敛次数/运行次数	平均收敛代数	标准差
F1	25/25	331.2	6644.0
F2	25/25	297.36	5967.2

测试函数 3、

$$F_3(x) = x_1^2 + x_2^2 + 2x_3^2 + x_4^2 - 5x_1 - 5x_2 - 21x_3 + 7x_4$$
$$g_2(x) = -x_1^2 - x_2^2 - x_3^3 - x_4^2 - x_1 + x_2 - x_3 + x_4 + 8$$
$$g_3(x) = -x_1^2 - 2x_2^2 - x_3^2 - 2x_4^2 + x_1 + x_4 + 10$$
$$g_4(x) = -x_1^2 - x_2^2 - x_3^2 - 2x_1 + x_2 + x_4 + 5$$

测试函数 4、

$$F_4(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + 3(x_4 - 11)^2 + x_3^4 + 10x_5^6$$
$$+ 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7$$
$$g_2(x) = -2x_1^2 - 3x_3^4 - x_3 - 4x_4^2 - 5x_5 + 127$$
$$g_3(x) = -7x_1 - 3x_2 - 10x_3^2 - x_4 + x_5 + 282$$
$$g_4(x) = -23x_1 - x_2^2 + 3x_1x_2 - 2x_3^2 - 5x_6 + 11x_7$$

算法定义域为 $[-5,5]^D$，其中，D 为所考虑问题的维数，且整个种群含有 20 个微粒，结果见表 8.4。

表 8.4、测试函数 f_3, f_4 结果

函数	收敛次数/运行次数	平均收敛代数	标准差
F3	25/25	365.56	7331.2
F4	25/25	335.08	6721.6

测试函数 5、 $\min \max\{|x_1 + 2x_2 - 7|, |2x_1 + x_2 - 5|\}$

测试函数 6、 $\min \max\{|x_j|\}, 1 \le j \le 10$

算法定义域为 $[-5,5]^D$，其中，D 为所考虑问题的维数，且对于函数 f_5 种群含有 20 个微粒，对于函数 f_6 种群含有 100 个微粒。结果见表 8.5。

表 8.5、测试函数 f_5, f_6 结果

函数	收敛次数/运行次数	平均收敛代数	标准差
F5	25/25	285.44	5728.8
F6	25/25	320.8	32180.0

对于测试函数 7 到测试函数 13，限于篇幅，我们仅给出相应的结果及参数设置。

表 8.6、测试函数 7~13 的参数设置

函数	维数	$\#\ f_i(x)$
CB2	2	3
WF	2	3
SPIRAL	2	2
EVD52	3	6
POLAK6	4	4
WONG1	7	5
WONG2	10	9

（$\#\ f_i(x)$ 表示函数 $f_i(x)$ 的数目）

表 8.7、测试函数 $f_7 - f_{13}$ 结果

函数	收敛次数/运行次数	平均收敛代数	标准差
CB2	25/25	319.48	6409.6
WF	25/25	306.04	6140.8
SPIRAL	25/25	257.52	5170.4
EVD52	25/25	319.12	6402.4
POLAK6	25/25	337.8	6777.6
WONG1	24/25	366.04	25692.8
WONG2	25/25	353.48	35448.0

8.5 在整数规划问题中的应用

整数规划的一般形式为

$$\min_x f(x), x \in S \subseteq Z^n \tag{8.33}$$

本文为了应用微粒群算法，首先在实数空间 R^n 中考虑问题，然后将求得的结果离散化，考虑距离该点最近的整数点并取代之。

为了验证本节算法，我们使用了 6 个测试函数，它们分别为

测试函数 1、 $F_1(x) = \| x \|_1 = | x_1 | + ... + | x_D |$

其中， $x = (x_1, x_2, ..., x_D) \in [-100,100]^D$ ，D 为问题的维数。

测试函数 2、 $F_2(x) = x^T x = (x_1 ... x_D) \begin{pmatrix} x_1 \\ ... \\ x_D \end{pmatrix}$

其中， $x = (x_1, x_2, ..., x_D) \in [-100,100]^D$ ，D 为问题的维数。

测试函数 3、 $$F_3(X) = -(15,27,36,18,12)\vec{X} + \vec{X}^T \begin{pmatrix} 35 & -20 & -10 & 32 & -10 \\ -20 & 40 & -6 & -31 & 32 \\ -10 & -6 & 11 & -6 & -10 \\ 32 & -31 & -6 & 38 & -20 \\ -10 & 32 & -10 & -20 & 31 \end{pmatrix}$$

测试函数 4、 $$F_4(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2$$

测试函数 5、 $$F_5(x) = (9x_1^2 + 2x_2^2 - 11)^2 + (3x_1 + 4x_2^2 - 7)^2$$

测试函数 6、 $$F_6(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$

测试函数 7、 $$F_7(x) = (x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$$

对于测试函数 F_1, F_2，我们考虑了多种情形，并对每种情况分别作用了 100 次实验。它们的不同的参数设置及实验结果见表 8.8 与表 8.9。

对于测试函数 $F_3 \sim F_7$，它们的参数设置可见表 8.10，其中，对每种情况分别作用了 100 次试验，且分别在不同的定义域上（$[-100,100]^D$，$[-50,50]^D$，$[-25,25]^D$）进行了测试，其实验结果可见表 8.11~8.13。

表 8.8、对于函数 F_1, F_2 的不同参数设置

维数 D	种群大小	最大截止代数
5	20	1000
10	50	1000
15	100	1000
20	200	1000
25	250	1500
30	300	2000

表 8.9、函数 F_1, F_2 的测试结果

函数	维数	平均收敛率(%)	平均收敛代数	标准差
F1	5	100	440.72	8834.3
F2	5	100	440.92	8838.4
F1	10	100	453.48	22724.0
F2	10	100	454.88	22794.0
F1	15	100	459.44	46044.0
F2	15	100	462.88	46388.0
F1	20	100	465.24	93248.0
F2	20	100	467.08	93616.0
F1	25	100	683.44	171110.0
F2	25	100	685.44	171610.0
F1	30	80	914.64	274692.0
F2	30	84	913.28	274284.0

表 8.10、函数 $F_3 \sim F_7$ 的不同参数设置

函数	维数	种群大小	最大截止代数
F3	5	150	1000
F4	2	20	1000
F5	2	20	1000
F6	2	20	1000
F7	4	40	1000

表 8.11、函数 $F_3 \sim F_7$ 在定义域 $[-100,100]^D$ 上的实验结果

函数	平均收敛率	平均收敛代数	标准差
F3	80%	499.4	75060.0
F4	100%	402.32	8066.4
F5	100%	415.08	8321.6
F6	100%	418.4	8388.0
F7	92%	460.92	18476.8

表 8.12、函数 $F_3 \sim F_7$ 在定义域 $[-50,50]^D$ 上的实验结果

函数	平均收敛率	平均收敛代数	标准差
F3	92%	447.72	67308.0
F4	100%	328.48	6589.6
F5	100%	357.12	7162.4
F6	100%	342.64	6872.8
F7	100%	440.8	17672.0

表 8.13 函数 $F_3 \sim F_7$ 在定义域 $[-25,25]^D$ 上的实验结果

函数	平均收敛率	平均收敛代数	标准差
F3	96%	420.28	63192.0
F4	100%	246.16	4943.2
F5	100%	228.28	4585.6
F6	100%	332.08	6661.6
F7	100%	429.32	17212.8

8.6 使用微粒群算法寻找多峰函数的最小点

为了考虑方便，我们假设问题的最小点为 0。为了寻找多峰函数的极值点，首先选择一个适当的阈值 $\varepsilon > 0$，对微粒群体进化，如果存在某个微粒 x*，其适应值低于阈值 ε，则将该微粒分离出种群，并随机产生一个新微粒以保证微粒种群个数保持不变。然后对整个种群的所有微粒修改其适应值，即

$$f(x) \leftarrow \frac{f(x)}{\| x - x^* \|} \tag{8.34}$$

如果微粒 x*没有达到最小值的精度要求，则在其附近生成一个小微粒群体，用以对该微粒附近的点进行搜索。同时，原微粒群体继续进化，这样，通过不断的分离，我们可以在有限的时间内找到大部分极值点。

该算法的基本框架可以表示如下：

Step0. 设置误差 $\varepsilon > 0$ ；

Step1. 初始化种群及参数；

Step2. 执行单个微粒群算法；

Step3. 若最优微粒的适应值满足 $f_{best} \le \varepsilon$，隔离 f_{best} ；

Step4. 在 f_{best} 附近执行局部搜索；

Step5. 对适应值函数执行变换 $f(x) \leftarrow \dfrac{f(x)}{\| x - x^* \|}$ ；

Step6. 若满足结束条件，停止；否则，转 step2。

为了表明该算法的有效性，我们考虑下面的测试函数

$$f(x_1, x_2) = \cos(x_1)^2 + \sin(x_2)^2 \tag{8.35}$$

它在 $(k\dfrac{\pi}{2}, \lambda\pi), k = \pm 1, \pm 3, ...; \lambda = 0, \pm 2, ...$ 上得到全局最小值，其图像为

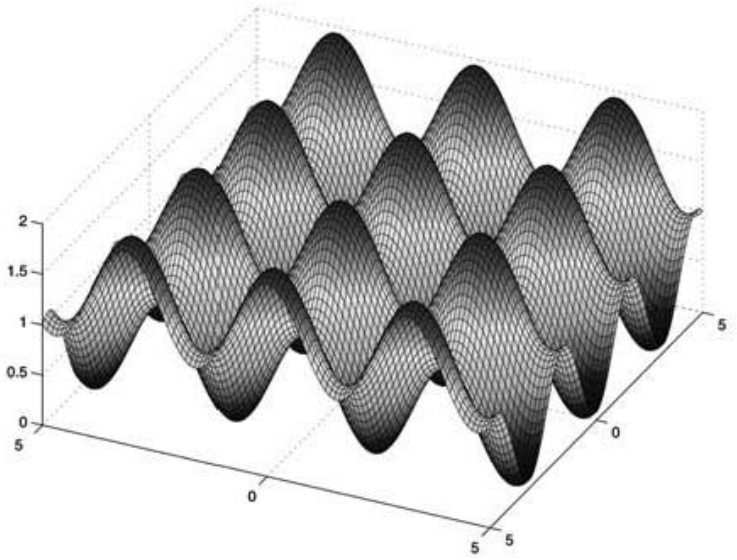

图 8.5 测试函数曲线图

为了解释算法，我们首先考虑只含有一个微粒的群体，其进化轨迹为图 8.6，显然，由于只有一个微粒，虽然经过许多全局极值点的邻域，但都没有搜索到更好的结果。但对于含有多个微粒的群体时，效果非常好。图 8.7 就是其中一幅搜索到的全局最优解，其误差为10^{-5}。

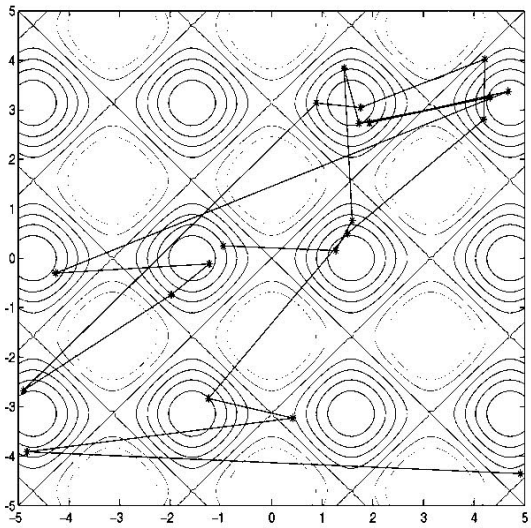

图 8.6、单个微粒的运行轨迹图

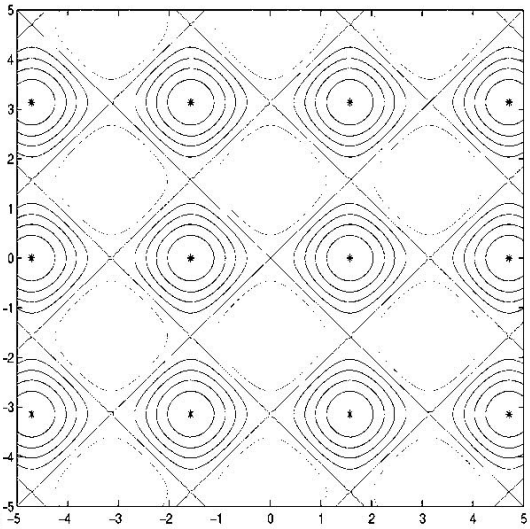

图 8.7、误差为10^{-5}的搜索结果