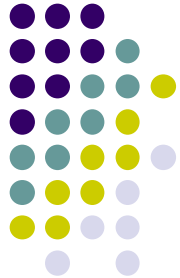
PSO求解约束优化问题

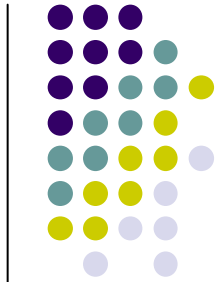


1）约束优化问题(Constrained Optimization， CO)

2）CPSO

3) 函数扩展技术

构造算法



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。

约束优化问题描述

$$\min\ f(\boldsymbol{x}) \tag{4-1}$$

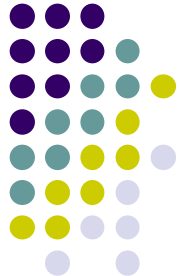
$$\text{s.t.}\ \ g_i(\boldsymbol{x}) \leqslant 0, i = 1,2,\cdots,n \tag{4-2}$$

$$h_j(\boldsymbol{x}) = 0, j = 1,2,\cdots,p \tag{4-3}$$

$$l_k \leqslant x_k \leqslant u_k, k = 1,2,\cdots,d \tag{4-4}$$

其中，$\boldsymbol{x}=[x_1,x_2,\cdots,x_d]^{\mathrm{T}}$ 表示决策解向量，d 是决策变量的维数，式(4-1)为目标函数，式(4-2)称为不等式约束，式(4-3)称为等式约束，l_k 和 u_k 分别表示第 k 维变量取值的上下界。

约束优化问题的求解难点

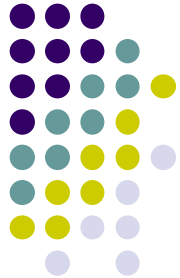


(1) 优化曲面的复杂性。复杂的优化曲面对约束优化带来的求解难点类似于无约束优化问题，

(2) 不可行域的存在性。约束的存在，导致决策变量的搜索空间产生了不可行域。

(3) 满足约束与优化目标的不平衡。

求解约束极值问题的传统方法



可行方向法（Feasible Direction）、

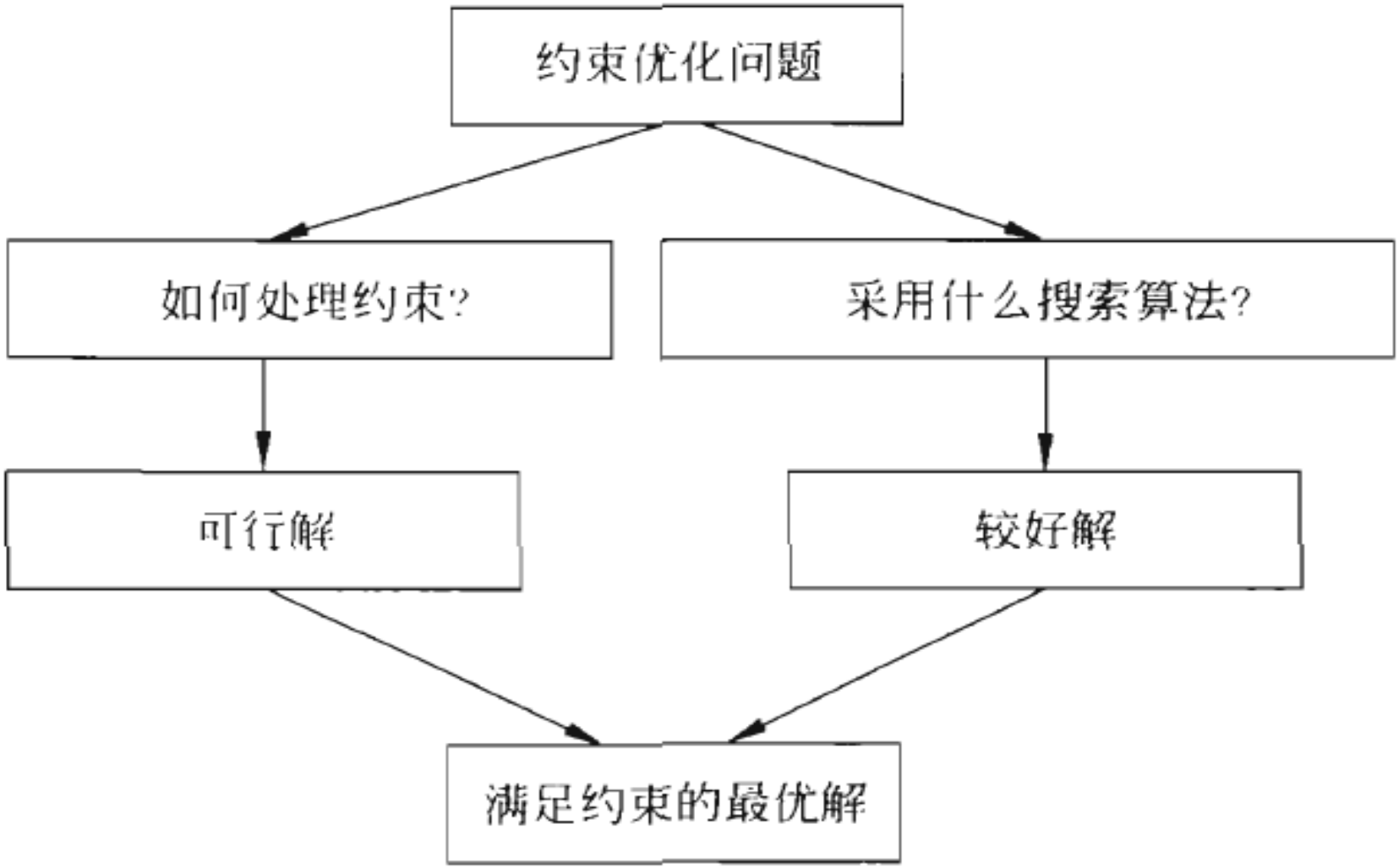
梯度投影法(Gradient Projection Method)、

约束集法(Active Set Method)、

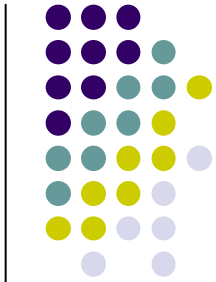
罚函数法（Penalty Function Method)等等。

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。

约束优化问题的求解思路



2.智能约束处理技术



1) 无约束化处理

罚函数法是最常用的约束处理技术，其基本思想是通过序列无约束最小化技术（Sequential Unconstrained Minimization Technique， SUMT），将约束优化问题转化为一系列无约束优化问题进行求解，其原理简单，实现方便，对问题本身没有苛刻要求。

罚函数法

罚函数法就是将目标函数和约束同时综合为一个罚函数，典型的罚函数如下:

$$\Phi(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{n} r_i \times G_i(\boldsymbol{x}) + \sum_{j=1}^{p} s_j \times H_j(\boldsymbol{x})$$

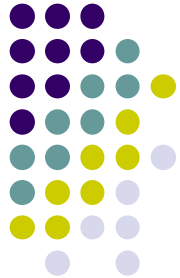
G(x),H(x)常用形式

$$G_i(x) = \max[0, g_i(x)] \quad 或 \quad G_i(x) = \{\max[0, g_i(x)]\}^2$$

$$H_j(x) = |h_j(x)| \quad 或 \quad H_j(x) = [h_j(x)]^2$$

如何合理设置罚因子，是利用罚函数法求解约束优化问题的一个瓶颈，也是约束优化领域的关键问题。

定常罚函数法



该类方法将罚因子在整个算法流程中设置为定值，但算法通常因数值的单一性而效果不佳。

分层罚因子法，采用如下罚函数：

$$\Phi(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{N} (R_{k.,} \times \max[0, g_i(\boldsymbol{x})]^2)$$

2.动态罚函数法

在动态罚函数法中.罚因子的数值是时变的，通常随进化代数增大而增大。

理由：在进化初期采用较小的罚因子，算法将有可能对不可行域进行一定程度的搜索；而进化后期采用较大的罚因子，将使算法的搜索集中在可行域，寻找目标更优的可行解。

动态罚函数

$$\Phi(\boldsymbol{x}) = f(\boldsymbol{x}) + (C \times t)^{\alpha} \times SVC(\beta, \boldsymbol{x})$$

3.退火罚函数法

一种基于模拟退火策略的罚函数法，罚因子每隔一定的进化代数就随温度的减小而增大，到算法后期罚因子则因温度很低而变得很大。该技术可认为是一种特殊的动态罚函数方法，但可通过控制温度来调整罚因子。

$$\Phi(x) = f(x) + \frac{1}{2\tau} \times \sum_{i} V_i^2(x)$$

其中，τ表示温度，$V_i(x)$为各约束违反量的函数。

4.适应性罚函数法

适应性罚函数法，把搜索过程中获得的信息作为反馈，来指导罚因子的调整。

$$\Phi(x) = f(x) + \lambda(t) \times \left\{ \sum_{i=1}^{n} \max[0, g_i(x)]^2 + \sum_{j=1}^{p} | h_j(x) | \right\}$$

其中，$\lambda(t)$在每一代按如下方式更新

$$\lambda(t+1) = \begin{cases} \lambda(t)/\beta_1 & \text{情况 1} \\ \beta_2 \lambda(t) & \text{情况 2} \\ \lambda(t) & \text{其他} \end{cases}$$

非固定多段映射罚函数法

- 通常，所构造的广义目标函数具有如下形式：

$$F(x) = f(x) + h(k)H(x), \qquad x \in S \subset R^n$$

其中f(x)代表原目标函数；h(k)H(x)称为惩罚项，H(x)表示惩罚力度, h(k)为惩罚因子。

如果在求解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。

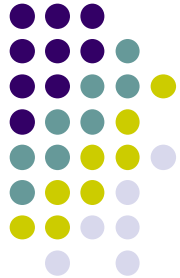
动态调整

- 其中**h(k)**可以被动态调整，**H(x)**具体定义如下

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

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)。

4.2.2基于排序的方法



基于排序的约束处理技术不再进行无约束化处理，而是通过综合考虑目标函数值和约束违反的程度来对不同候选解进行比较。

1）随机排序法

2）改进的随机排序法

3）基于可行性规则的方法

4. 2. 3基于多目标的方法

多目标优化问题通常可描述如下:

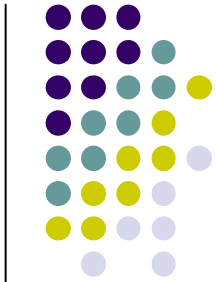
$$\min \quad f(x) = (f_1(x), \cdots, f_m(x)), x \in S$$

其中，$f_1, \cdots, f_m$ 为 m 个目标函数，S 为搜索空间。

对于问题的两个可行解x1和x2，若下式满足，则称解x1支配解x2，记作x1>x2

$$\begin{cases} \forall j \in \{1, 2, \cdots, m\}, f_j(x_1) \leqslant f_j(x_2) \\ \exists k \in \{1, 2, \cdots, m\}, f_k(x_1) < f_k(x_2) \end{cases}$$

Pareto最优解



给定一个可行解x’，若S中不存在支配x的解，则称x’为Pareto最优解或非支配解(non-dominated solution)或非劣解。所有Pareto最优解构成目标空间的Pareto前沿(Pareto front) 。

目标函数和约束函数当作并列的多个目标

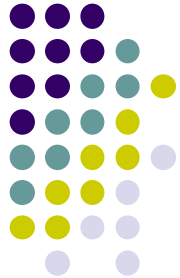
基于多目标的约束处理技术就是把目标函数和约束函数当作并列的多个目标来处理。譬如，第4. 1节描述的约束优化问题，可转化为式(4-15）描述的多目标优化问题或式(4-16)描述的双目标优化问题。

$$\min \quad (f(x), G_1(x), \cdots, G_n(x), H_1(x), \cdots, H_p(x)) \tag{4-15}$$

$$\min \quad (f(x), S(x)), S(x) = \sum_{i=1}^{n} G_i(x) + \sum_{j=1}^{p} H_j(x) \tag{4-16}$$

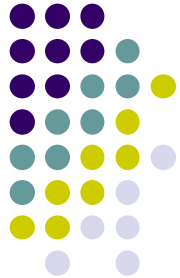
通过采用非支配解的概念来比较解，可使搜索向Pare-to前沿方向前进，其中非支配解集中第一个目标函数值最小且其余目标值均等于0的解就是原约束优化问题的最优解。

2.基于支配的选择技术



一种基于支配的约束处理规则。(1)可行解总优于不可行解;(2)若两个解均可行，则目标值好的解为优;(3)若两个解均不可行且其中一个解为非支配解而另一个解为受支配解，则非支配解为优;(4)若两个解均不可行且它们同为非支配解或受支配解，则约束违反量小的解为优。

基于协进化PSO算法的约束优化

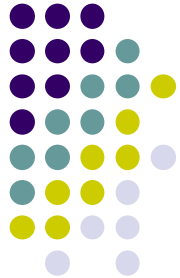


罚函数法求解约束优化问题的性能取决于两个关键环节:

(1)罚函数形式的设计。罚函数要综合考虑约束和目标两个方面，因此设计罚函数时应该考虑挖掘不可行解违反约束的信息，设计有效的约束违反函数，合理体现可行解与不可行解在综合评价上的差异。

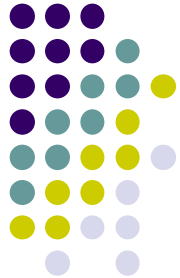
(2）罚因子的选取。罚因子直接平衡约束违反量和目标间的平衡。

一种基于协进化模型的微粒群优化



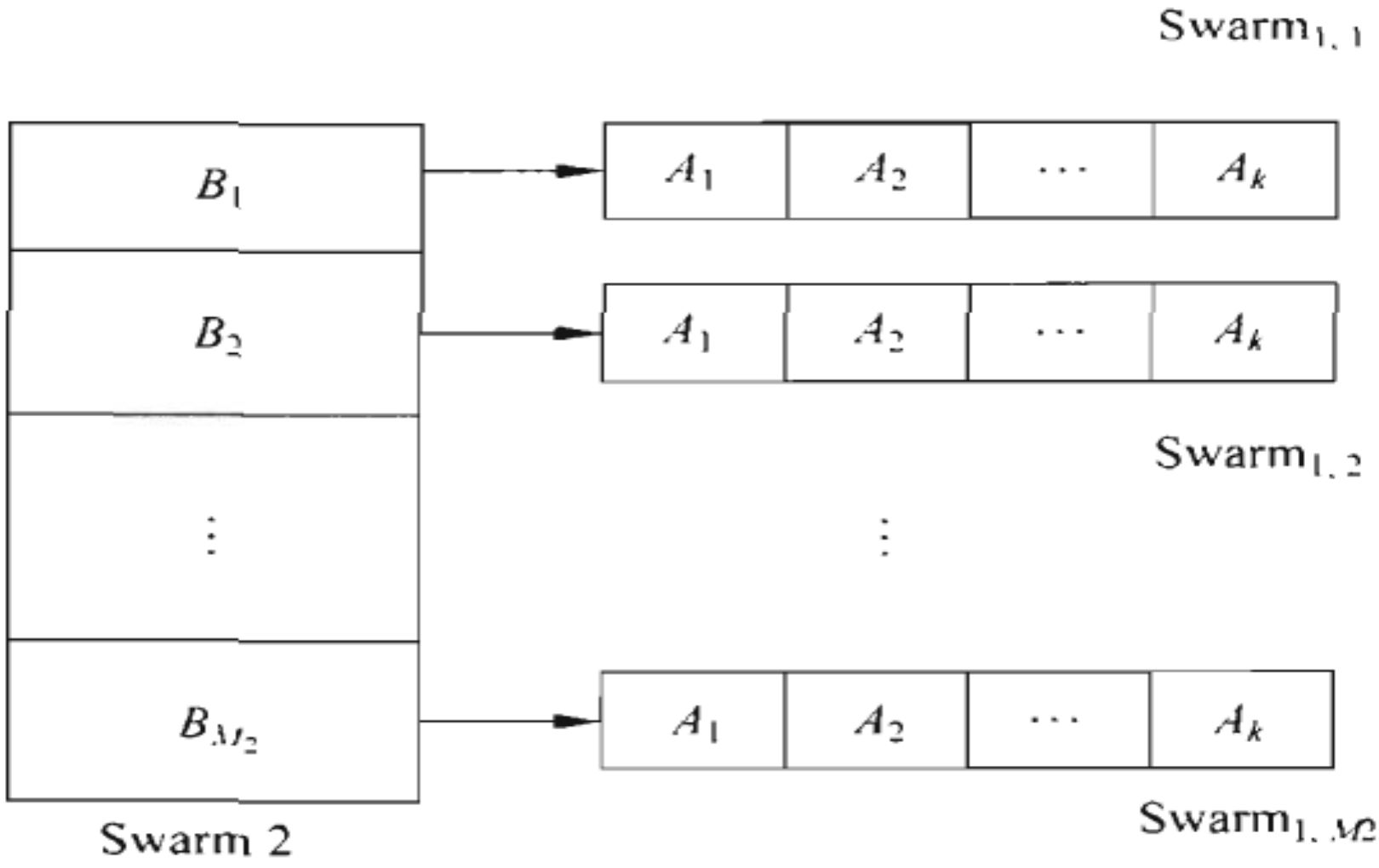
设计思路归纳为，在设计罚函数方面，不但考虑不可行解违反约束的总量，而且还利用各不可行解违反约束的个数这一信息;在罚因子选择方面，基于协进化模型，把罚因子也作为寻优变量，在搜索过程中利用PSO算法自适应地进行调整，使算法最终不仅获得约束优化问题的优良解，同时还获得适合于问题的最佳罚因子。

4. 3 .2 协进化模型

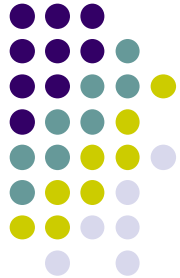


- 协进化的原理可解释为，算法采用多个种群，或者将一个种群分为多个部分，各种群在各自独立进化的同时相互间共享和交互信息，各种群不仅利用从外界获得的信息来指导自身的搜索，同时还把探索得到的经验与其他种群分享，从而使整个系统协同进化，直至获得最优解。
- 本节的CPSO算法采用如图4. 7所示的协进化模型。

协进化模型示意图（图4.7）



cpso算法包含两类种群



- 一类种群包含M2个子种群Swarm1,j(j=1, …, M2}，子种群规模均为M1，种群中的每个微粒Ai(i=I,…,k)则表示问题的一个决策解，该类种群用于进化决策解; ,
- 另一个规模为M2的种群Swarm 2，其每个微粒Bj{j=1,…,M2})代表一组罚因子。用于计算Swarm1 , j中各微粒的罚函数值(或称适配值)。

协进化的每代进化过程

Swarm1 , j中的每个微粒利用B，表示的罚因子计算适配值，

并连续采用PSO算法进化G、代获得一个新的解的种群Swarm1 ,j;

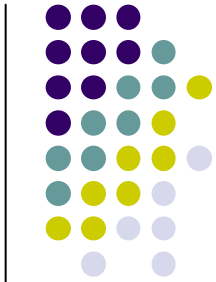
然后，根据Swarm1 , j中所有解的优劣信息，评价Swarm 2中微粒Bj的优劣，即评价罚因子;

当Swarm 2中所有微粒B，均得到评价后，Swarm 2采用PSO算法进化一代，从而获得新的种群Swarm 2，即得到M:组新的罚因子。

在一代协进化结束后Swarm1,j(j=1.2,…,M2})再分别用新的M2组罚因子进行评价，以此类推，直到满足算法终止准则，例如达到给定的最大协进化代数G2。

算法通过比较所有Swarm1,j得到的历史最好解，将最优者作为最终解输出，同时算法输出终止时Swarm 2中的最优微粒。即最佳罚因子。

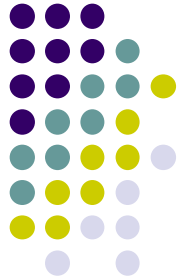
CPSO小结



协进化就是两类种群的交互进化工程，第一类种群Swarm1 ,j用PSO算法来进化解向量，第二类种群Swarm 2 用PSO算法来调整罚因子。

通过协进化模型下的PSO算法，不但算法在解空间进化探索决策解，而且在罚因子空间进化探索罚因子，最终同时获得最优决策解和一组合适的罚因子。

CPSO算法设计



1. Swarm1 , j的评价函数

罚函数既是违反约束数量的函数，又是违反约束程度的函数，则其效果要比罚函数仅为约束数量的函数的情况好。基此，采用如下适配值函数评价Swarm1,j中第i个微粒

$$F_i(\boldsymbol{x}) = f_i(\boldsymbol{x}) + sum\_viol \times w_1 + num\_viol \times w_2$$

其中，f(x)表示第i个微粒的目标值，sum_viol表示该微粒违反约束的总量，num_ viol表示该微粒违反约束的个数，w1,w2是由Swarm 2中微粒Bj对应的罚因子。

sum_ viol按如下公式计算

$$sum\_viol = \sum_{i=1}^{N} g_i(\boldsymbol{x}), \quad \forall\, g_i(\boldsymbol{x}) > 0$$

2. Swarm 2的评价函数

Swarm 2的每一个微粒B，均代表一组罚因子，即w1和w2。当Swarm1 ,j进化G1代后，Swarm 2的第j个微粒B，采用如下方法进行评价。

(1)若Swarm 1 , j中至少有一个可行解，则称B，为一个有效(valid)微粒，并按下式评价B,

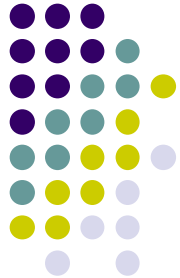
$$P(B_j) = \frac{\sum f_{feasible}}{num\_feasible} - num\_feasible$$

(2)若Swarm1 ,j中没有可行解(可认为惩罚项太小了)，则称B，为一个无效(invalid)微粒，并采用下式评价B;

$$P(B_j) = \max(P_{valid}) + \frac{\sum sum\_viol}{\sum num\_viol} + \sum num\_viol$$

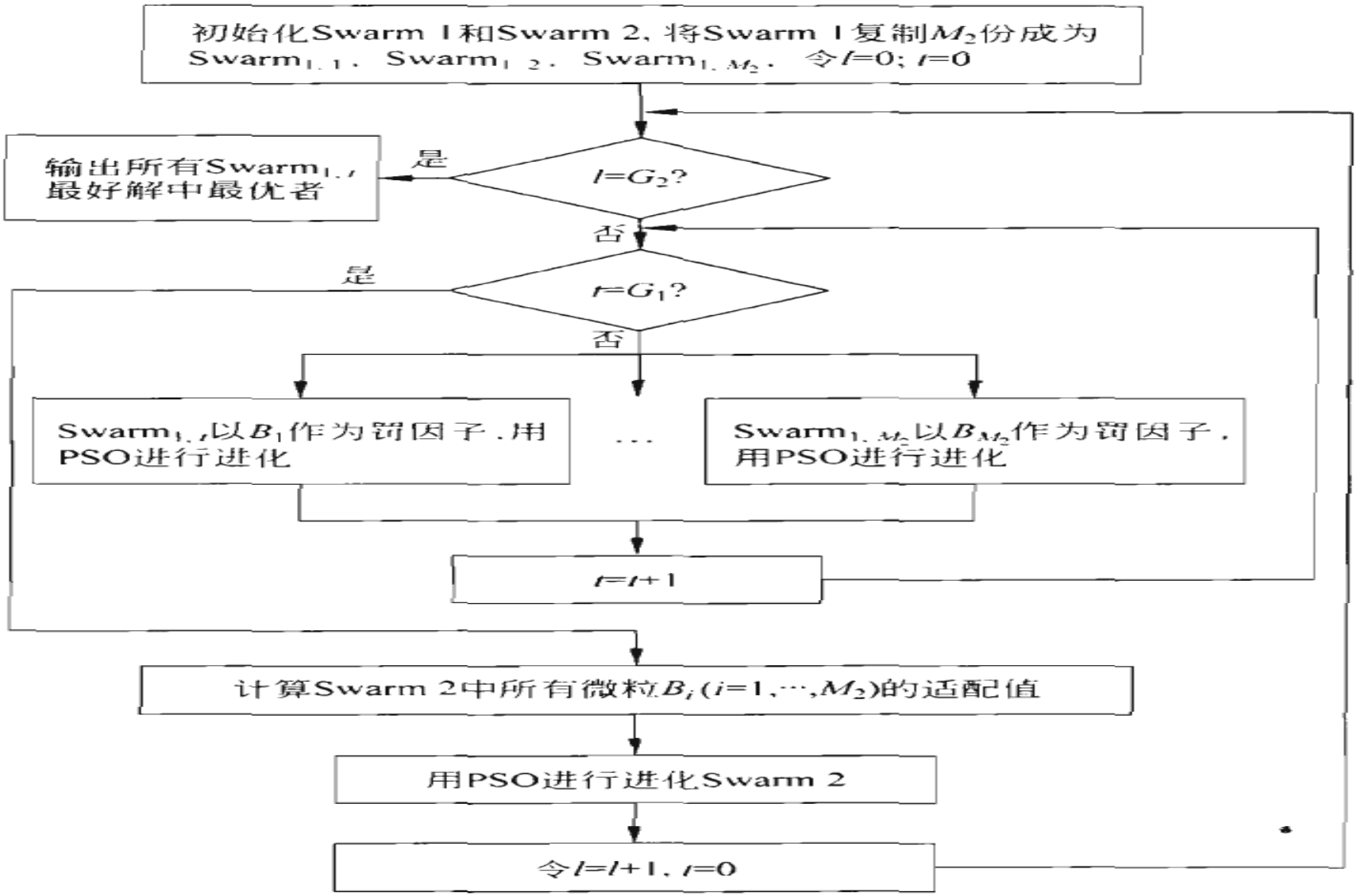
其中，$\max(P_{valid})$表示 Swarm 2 中所有有效微粒的最大适配值，$\sum sum_viol$ 表示 Swarm1,j 中所有微粒违反约束的总量，$\sum num\_viol$ 表示 Swarm1,j 中所有微粒违反约束的总数。

（3） Swarml , j和Swarm 2的进化方程



- 两类种群中的微粒均采用标准PSO算法进行进化。特别地，
- Swarm 2里的每个微粒表示一组罚因子，即w1和w2,
- Swarml , j里的每个微粒表示一个决策解向量。

（4）CPSO算法框架

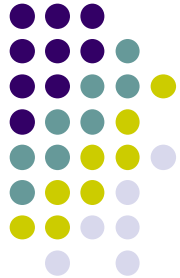

初始化Swarm 1和Swarm 2，将Swarm 1复制M_2份成为$Swarm_{1,1}$，$Swarm_{1,2}$，$Swarm_{1,M_2}$，令$l=0$；$t=0$
$l=G_2$?
是
输出所有$Swarm_{1,i}$最好解中最优者
否
$t=G_1$?
是
否
$Swarm_{1,1}$以B_1作为罚因子，用PSO进行进化
...
$Swarm_{1,M_2}$以B_{M_2}作为罚因子，用PSO进行进化
$t=t+1$
计算Swarm 2中所有微粒B_i ($i=1,\cdots,M_2$)的适配值
用PSO进行进化Swarm 2
令$l=l+1$，$t=0$

4. 3 .4 数值仿真与分析

- 对于每个测试问题，CPSO算法采用如下统一的一组参数。M1=50,M2 =20,G1＝25,G2 =8,惯性因子LDW法，0.9->0.2, 加速因子c1=c2=2。
- 另外，Swarm1, j中微粒位置的最大值和最小值(即X1,max和X1,min对应于各问题变量的取值上下界，Swarm 2中微粒位置的最大值和最小值(即X2, max和X2.min设置为w1max= w2.max=1000. w1min= w2.min=0。
- 第一个例子的可行域相对较小，算法单独将罚因子的上下界设置为w1max＝w2max=10000,w1min= w2min =5000。两类种群中微粒速度的上下界设置

$$V_{i,\max}=0.2\times(X_{i,\max}-X_{i,\min}),V_{i,\min}=-V_{i,\max}。$$

1. 对焊接条设计问题的求解结果



该问题来自文献[23]，焊接条结构如图4. 9所示。优化目标是寻求满足切应力τ、弯曲应力σ、杆条弯曲载荷Pc、末端偏差δ和边界条件等约束的4个设计变量*h(x1)*、*l(x2:)* ,*c(x2)*和*b(x4)*，使得制造焊接条所需的总费用最小。

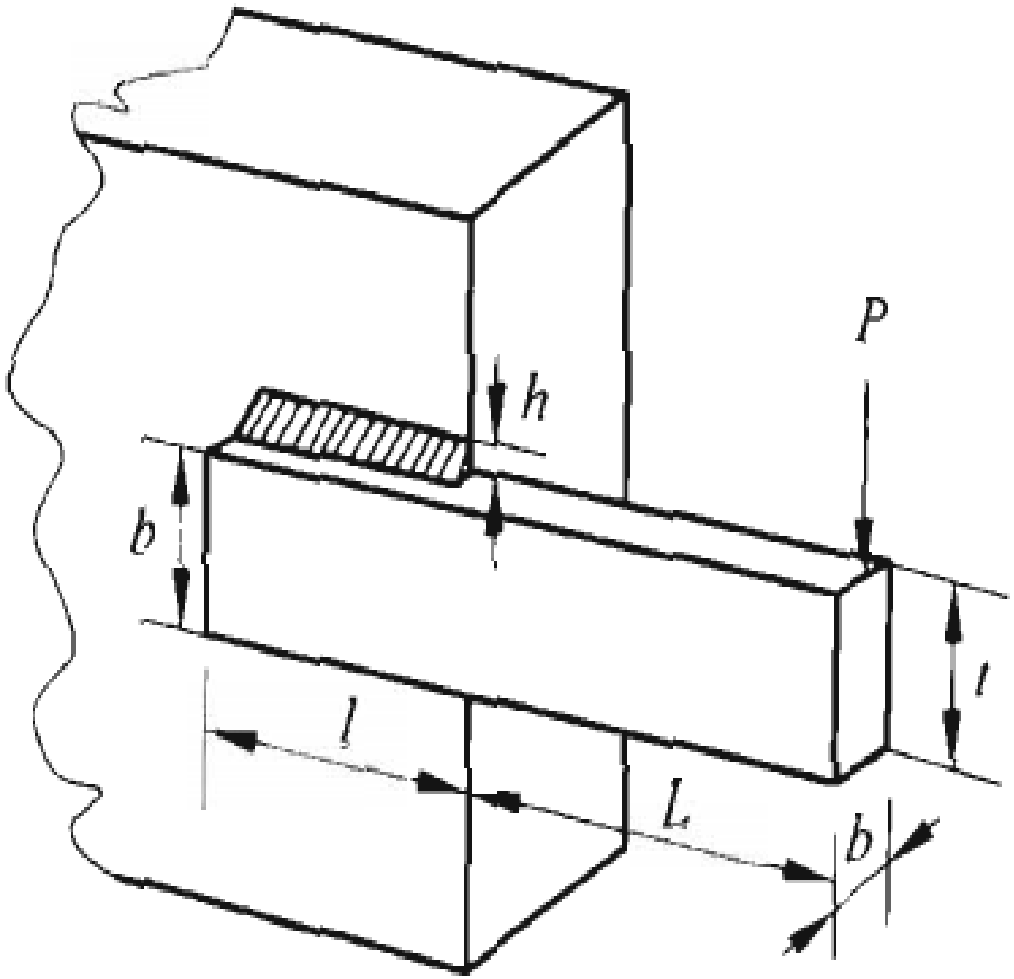


图 4.9　焊接条设计问题

该设计问题的数学模型描述如下:

$$\min \quad f(\boldsymbol{x}) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2)$$

$$\text{s.t.} \quad g_1(\boldsymbol{x}) = \tau(\boldsymbol{x}) - 13\,600 \leqslant 0$$

$$g_2(\boldsymbol{x}) = \sigma(\boldsymbol{x}) - 30\,000 \leqslant 0$$

$$g_3(\boldsymbol{x}) = x_1 - x_4 \leqslant 0$$

$$g_4(\boldsymbol{x}) = 0.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5.0 \leqslant 0$$

$$g_5(\boldsymbol{x}) = 0.125 - x_1 \leqslant 0$$

$$g_6(\boldsymbol{x}) = \delta(\boldsymbol{x}) - 0.25 \leqslant 0$$

$$g_7(\boldsymbol{x}) = 6000 - P_c(\boldsymbol{x}) \leqslant 0$$

其中

$$\tau(\boldsymbol{x}) = \sqrt{(\tau')^2 + 2\tau'\tau''x_2/(2R) + (\tau'')^2}$$

$$\tau' = 6000/(\sqrt{2}x_1x_2)$$

$$\tau'' = MR/J$$

$$M = 6000(14 + x_2/2)$$

$$R = \sqrt{[x_2^2 + (x_1 + x_3)^2]/4}$$

$$J = 2\{\sqrt{2}x_1x_2[x_2^2/12 + (x_1 + x_3)^2/4]\}$$

$$\sigma(\boldsymbol{x}) = 504\,000/(x_4x_3^2)$$

$$\delta(\boldsymbol{x}) = 2.1952/(x_3^3x_4)$$

$$P_c(\boldsymbol{x}) = 4.013 \cdot E \cdot (1 - 0.028\,234\,6x_3) \cdot x_3x_4^3/(6L^2)$$

[24]-GA传统罚函数，[25]几何规划法
[26][基于协进化模型的GA, [12]基于支配选择机制的GA

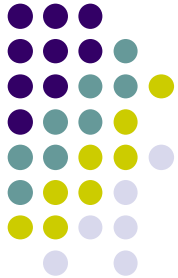


表 4.2　不同算法求解焊接条设计问题所得的最优解

决策变量	CPSO	文献[24]	文献[25]	文献[26]	文献[12]
$x_1(h)$	0.204 381	0.2455	0.2489	0.2088	0.2060
$x_2(l)$	3.505 107	6.1960	6.1730	3.4205	3.4713
$x_3(t)$	9.033 546	8.2730	8.1789	8.9975	9.0202
$x_4(b)$	0.205 878	0.2455	0.2533	0.21	0.2065
$g_1(x)$	−12.839 796	−5743.826 517	−5758.603 777	−0.337 812	−0.074 092
$g_2(x)$	−1.247 467	−4.715 097	−255.576 901	−353.902 604	−0.266 227
$g_3(x)$	−0.001 498	0	−0.004 400	−0.0012	−0.000 495
$g_4(x)$	−3.429 347	−3.020 289	−2.982 866	−3.411 865	−3.430 043
$g_5(x)$	−0.079 381	−0.120 500	−0.123 900	−0.0838	−0.080 986
$g_6(x)$	−0.235 536	−0.234 208	−0.234 160	−0.235 649	−0.235 514
$g_7(x)$	−11.681 355	−3604.275 002	−4465.270 928	−363.232 384	−58.666 44
$f(x)$	**1.728 024**	2.385 937	2.433 116	1.748 309	1.728 226

表 4.3　不同算法求解焊接条设计问题的统计结果

方法	最优解	平均解	最差解	标准偏差
CPSO	**1.728 024**	**1.748 831**	**1.782 143**	0.012 926
文献[24]	2.385 937	N/A	N/A	N/A
文献[25]	2.433 116	N/A	N/A	N/A
文献[26]	1.748 309	1.771 973	1.785 835	**0.011 220**
文献[12]	1.728 226	1.792 654	1.993 408	N/A

2.对伸缩绳设计问题的求解结果

- 该问题来自文献[27]，伸缩绳结构如

图4. 10所示。优化目标是寻求满足对最小偏差、切应力、湍振频率等一系列约束条件的3个决策变量，即平均卷直径D(x1)、线直径d(x2)和活动卷的数量P(x3)。使得伸缩绳的重量最小。

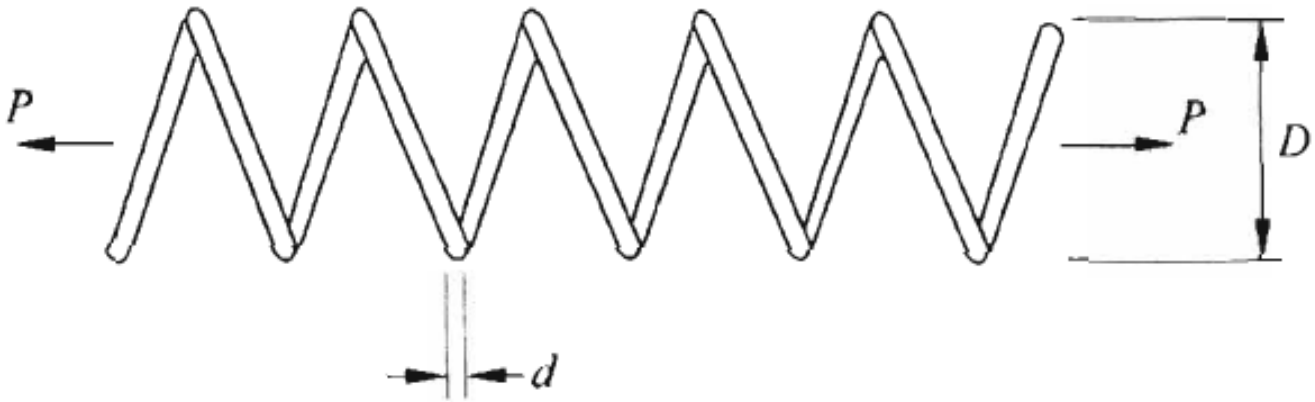


图 4.10　伸缩绳设计问题

该设计问题的数学模型描述如下

$$
\begin{aligned}
\min \quad & f(\boldsymbol{x}) = (x_3 + 2)x_2 x_1^2 \\
\text{s.t.} \quad & g_1(\boldsymbol{x}) = 1 - x_2^3 x_3/(71\,785x_1^1) \leqslant 0 \\
& g_2(\boldsymbol{x}) = (4x_2^2 - x_1 x_2)/[12\,566(x_2 x_1^3 - x_1^4)] \\
& \qquad + 1/(5108x_1^2) - 1 \leqslant 0 \\
& g_3(\boldsymbol{x}) = 1 - 140.45x_1/(x_2^2 x_3) \leqslant 0 \\
& g_4(\boldsymbol{x}) = (x_1 + x_2)/1.5 - 1 \leqslant 0
\end{aligned}
$$

决策变量的取值范围为 $0.05 \leqslant x_1 \leqslant 2, 0.25 \leqslant x_2 \leqslant 1.3, 2 \leqslant x_3 \leqslant 15$。

CPSO 算法与如下算法进行比较，包括包含 8 种数值优化算法的软件包[28]（只列出最好结果）、不变代价修复约束法[27]、基于协进化模型的 GA[26]、基于支配选择机制的 GA[12]。表 4.4 列出了各种算法求得的最优解，表 4.5 则给出了统计结果。

表 4.4　不同算法求解伸缩绳设计问题所得的最优解

决策变量	CPSO	文献[28]	文献[27]	文献[26]	文献[12]
$x_1(d)$	0.051 728	0.050 000	0.053 396	0.051 480	0.051 989
$x_2(D)$	0.357 644	0.315 900	0.399 180	0.351 661	0.363 965
$x_3(P)$	11.244 543	14.250 000	9.185 400	11.632 201	10.890 522
$g_1(x)$	−0.000 845	−0.000 014	0.000 019	−0.002 080	−0.000 013
$g_2(x)$	−0.000 013	−0.003 782	−0.000 018	−0.000 110	−0.000 021
$g_3(x)$	−4.051 300	−3.938 302	−4.123 832	−4.026 318	−4.061 338
$g_4(x)$	−0.727 090	−0.756 067	−0.698 283	−4.026 318	−0.722 698
$f(x)$	**0.012 675**	0.012 833	0.012 730	0.012 705	0.012 681

表 4.5 不同算法求解伸缩绳设计问题的统计结果

方法	最优解	平均解	最差解	标准偏差
CPSO	**0.012 675**	**0.012 730**	0.012 924	5.198 500e−5
文献[28]	0.012 833	N/A	N/A	N/A
文献[27]	0.012 730	N/A	N/A	N/A
文献[26]	0.012 705	0.012 769	0.012 822	**3.939 000e−5**
文献[12]	0.012 681	0.012 742	0.012 973	5.900 000e−5

3.对压力管设计问题的求解结果

- 该问题来自文献[29」，压力管结构如图4. 11所示。优化目标是寻求满足一系列约束条件的4个设计变量Ts(x1，圆柱形管子的厚度)、Th(x2，半球形盖子的厚度)、R(x3，圆柱形管子的内径)和L(x4，圆柱形管子的长度，不包括两端的盖子)，使得包括材料、焊接、铸造等费用在内的总费用最少。其中，Ts和Th均为0. 0625英寸的整数倍，R和L是连续变量。
- 在用CPSO算法求解时，Ts和Th均在实数空间搜索，只在求解目标函数值时才近似处理为0. 0625的整数倍。

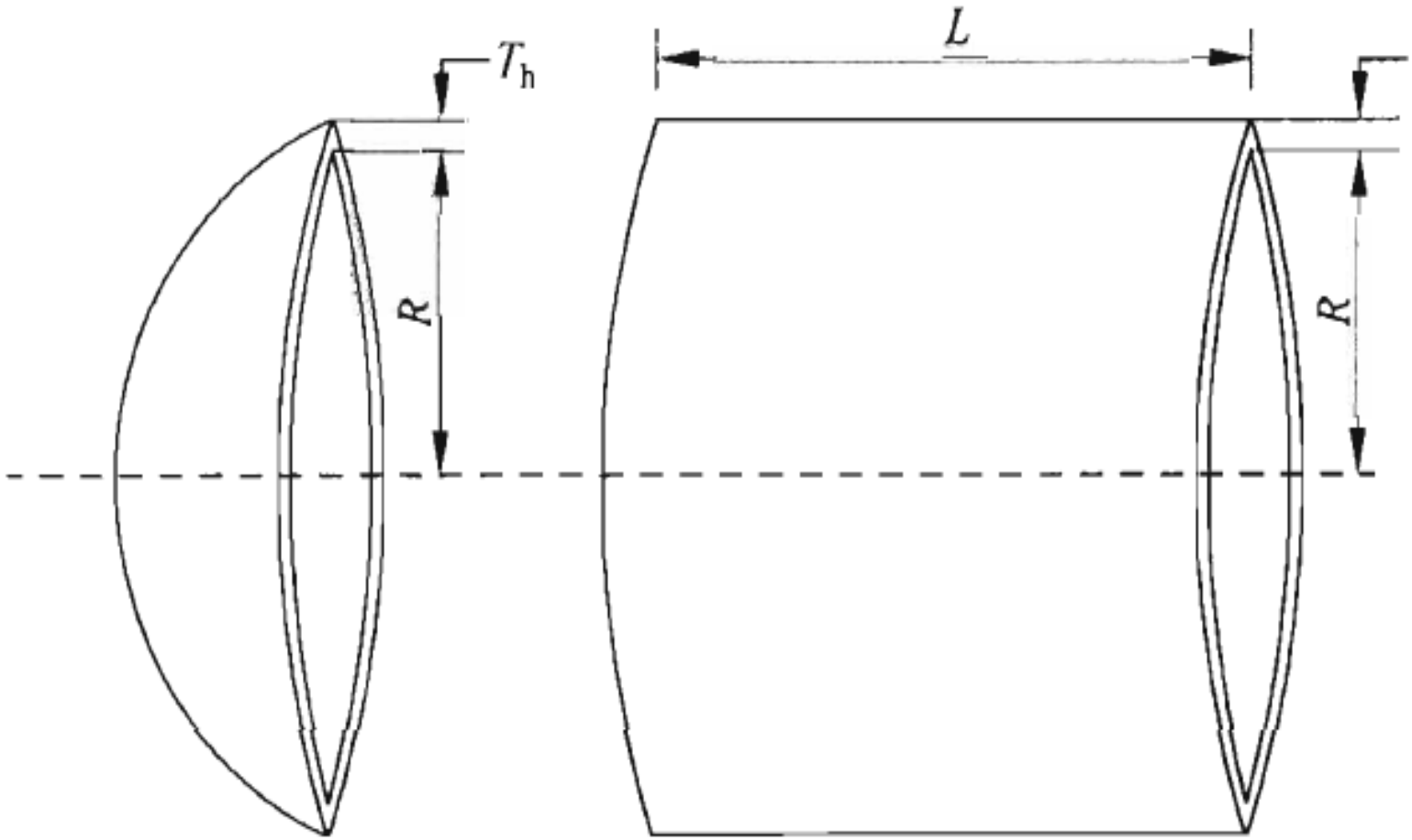


图 4.11　压力管设计问题

该问题的数学模型描述如下

$$\min \quad f(\boldsymbol{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$$

$$\text{s.t.}\ g_1(\boldsymbol{x}) = -x_1 + 0.0193x_3 \leqslant 0$$

$$g_2(\boldsymbol{x}) = -x_2 + 0.00954x_3 \leqslant 0$$

$$g_3(\boldsymbol{x}) = -\pi x_3^2x_4 - 4\pi x_3^3/3 + 1\,296\,000 \leqslant 0$$

$$g_4(\boldsymbol{x}) = x_4 - 240 \leqslant 0$$

决策变量的取值范围为 $1 \leqslant x_1 \leqslant 99, 1 \leqslant x_2 \leqslant 99, 10 \leqslant x_3 \leqslant 200, 10 \leqslant x_4 \leqslant 200$。

- CPSO算法与如下算法进行比较，包括遗传自适应搜索法[30]、扩张拉格朗日乘子法[29]、分支定界法[31]、基于协进化模型的GA[26]、基于支配选择机制的GA[12]。表4. 6列出了各种算法求得的最优解，表4. 7则给出了统计结果。

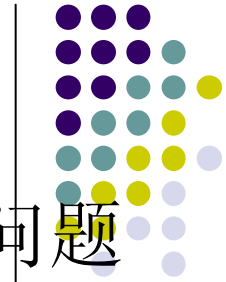
表 4.6　不同算法求解压力管设计问题所得的最优解

设计变量	CPSO	文献[30]	文献[29]	文献[31]	文献[26]	文献[12]
$x_1(T_s)$	0.8125	1.1250	1.1250	0.9375	0.8125	0.8125
$x_2(T_h)$	0.4375	0.6250	0.6250	0.5000	0.4375	0.4375
$x_3(R)$	42.0913	47.7000	58.2910	48.3290	40.3239	42.0974
$x_4(L)$	176.7465	117.7010	43.6900	112.6790	200.0000	176.6541
$g_1(x)$	−0.0001	−0.2044	0.000 02	−0.0048	−0.0343	−0.000 02
$g_2(x)$	−0.0359	−0.1699	−0.0689	−0.0389	−0.0528	−0.0359
$g_3(x)$	−116.3827	54.2260	−21.2201	−3652.8768	−27.1058	−27.8861
$g_4(x)$	−63.2535	−122.2990	−196.3100	−127.3210	−40.0000	−63.3460
$f(x)$	6061.0777	8129.1036	7198.0428	6410.3811	6288.7445	**6059.9463**

表 4.7 不同算法求解压力管设计问题的统计结果

方法	最优解	平均解	最差解	标准偏差
CPSO	6061.0777	**6147.1332**	6363.8041	86.4545
文献[30]	8129.1036	N/A	N/A	N/A
文献[29]	7198.0428	N/A	N/A	N/A
文献[31]	6410.3811	N/A	N/A	N/A
文献[26]	6288.7445	6293.8432	**6308.1497**	**7.4133**
文献[12]	**6059.9463**	6177.2533	6469.3220	130.9297

4）对标准测试问题的求解结果



- 13个不同类型的函数约束优化问题，作为标准问题常用于测试各种约束处理技术的性能和效率。下面，用其中的g04,g08, g09,g11和g12问题进一步检验CPSU算法的有效性和鲁棒性。
- CPSO算法对每个测试函数均独立运行30次，表4. 8、表4. 9和表4. 10给出了CPSO算法所得的最优解、平均解和最差解以及学术界最具代表性的几种约束优化算法的结果，包括共形映射法HM[2] ,随机排序法SR[3] , SMES[22], ASCHEA[10] ,

表 4.8 不同方法求解标准问题所得的最优解

问题	理论最优	HM[12]	SR[13]	ASCHEA[10]	SMES[22]	CPSO
g04	−30 665.539	−30 664.500	−30 665.539	−30 665.500	−30 665.539	−30 665.506
g08	0.095 825	0.095 825	0.095 825	0.095 825	0.095 825	**0.095 825**
g09	680.630 00	680.910 00	680.630 00	680.630 00	680.632 00	682.640 00
g11	0.750 000	0.750 000	0.750 000	0.750 000	0.750 000	0.750 011
g12	1.000 000	0.999 999	1.000 000	1.000 000	1.000 000	**1.000 000**

表 4.9　不同方法求解标准问题所得的平均解

问题	理论最优	HM[2]	SR[3]	ASCHEA[10]	SMES[22]	CPSO
g04	−30 665.539	−30 655.300	−30 665.539	−30 665.500	−30 665.539	−30 664.375
g08	0.095 825	0.089 157	0.095 825	0.095 825	0.095 825	**0.095 825**
g09	680.630 00	681.160 00	680.656 00	680.641 00	680.643 00	684.229 50
g11	0.750 000	0.750 000	0.750 000	0.750 000	0.750 000	0.751 227
g12	1.000 000	0.999 135	1.000 000	N/A	1.000 000	**1.000 000**

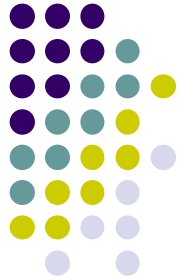
表 4.10　不同方法求解标准问题所得的最差解

问题	理论最优	HM[2]	SR[3]	ASCHEA[10]	SMES[22]	CPSO
g04	−30 665.539	−30 645.900	−30 665.539	N/A	−30 665.539	−30 661.778
g08	0.095 825	0.029 143 8	0.092 825	N/A	0.095 825	**0.095 825**
g09	680.630 00	683.180 00	680.763 00	N/A	680.643 00	687.000 00
g11	0.750 000	0.750 000	0.750 000	N/A	0.750 000	0.755 290
g12	1.000 000	0.991 950	1.000 000	N/A	1.000 000	**1.000 000**

讨论

- 其余8个标准问题测试时发现CPSO算法的性能不是太理想，尤其在求解高维问题(g02,20维)时求解结果与理论最优解相差较大，
- 在求解可行域很小(gO1,gOS, g07, g13,P值几乎为0)的问题时经常找不到可行解。这说明单一的CPSO算法在寻优性能上还有待改进，尤其对于强约束优化问题。与其他约束处理技术(如文献[10]中的特殊等式约束处理技术)相结合，有望进一步提高算法性能。

5）算法性能分析与参数设置



- 针对三个工程设计问题，首先考察算法首次获得最终最优解的代数，即在实验过程中对第一类种群中所有Swarm1，j的最优解进行逐代跟踪，
- CPSO算法30次独立运行的统计结果如表4. 11所示。由表可见，CPSO算法对于三个工程设计问题能够快速寻找到最优解。

表 4.11　获得最优解的首达代数

问题	最小首达代数	平均首达代数	最大首达代数
焊接条设计	30	34.5	39
伸缩绳设计	23	32.8	40
压力管设计	24	32.5	36

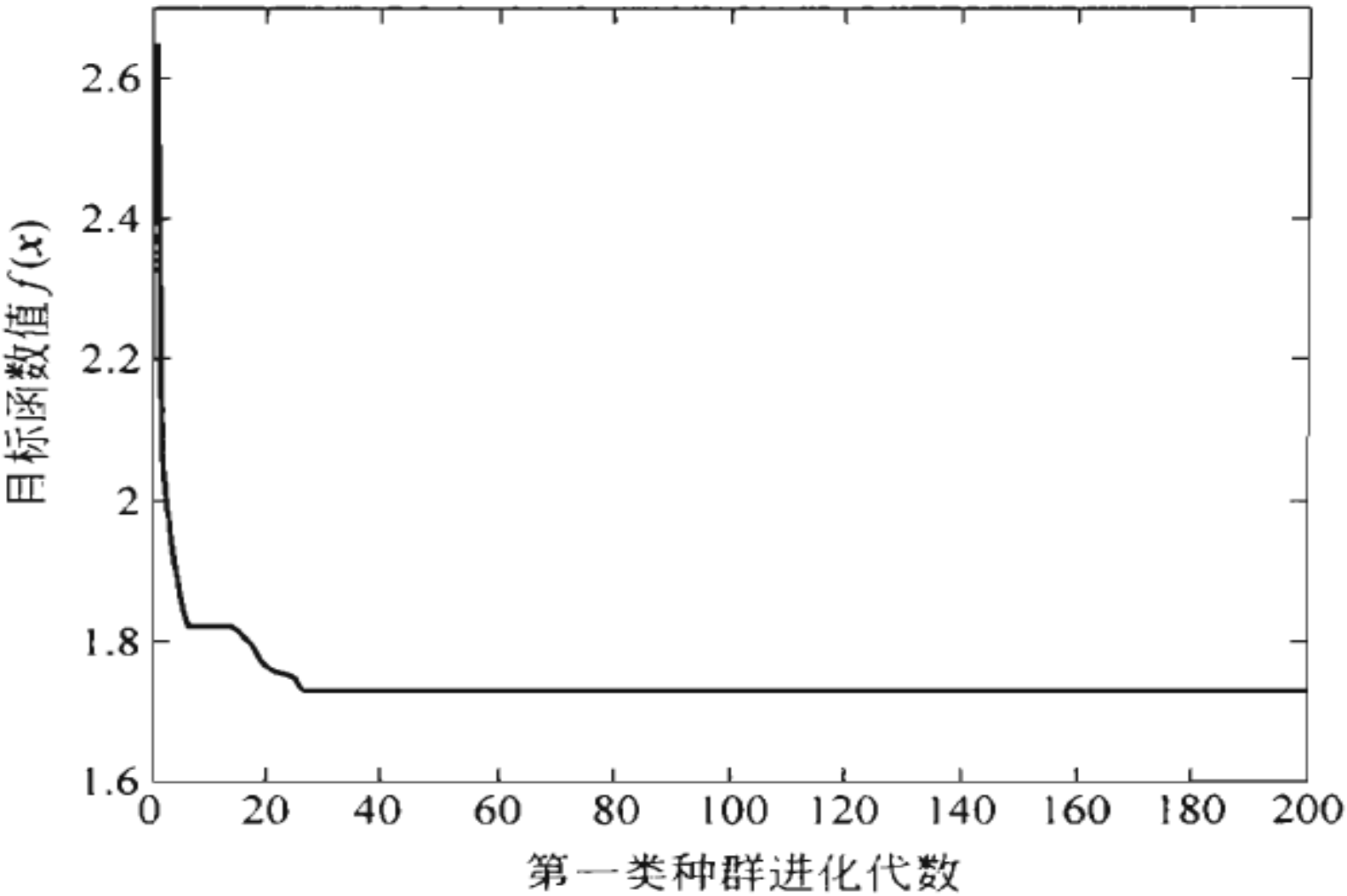


图 4.12　最优解进化动态曲线(焊接条设计问题)

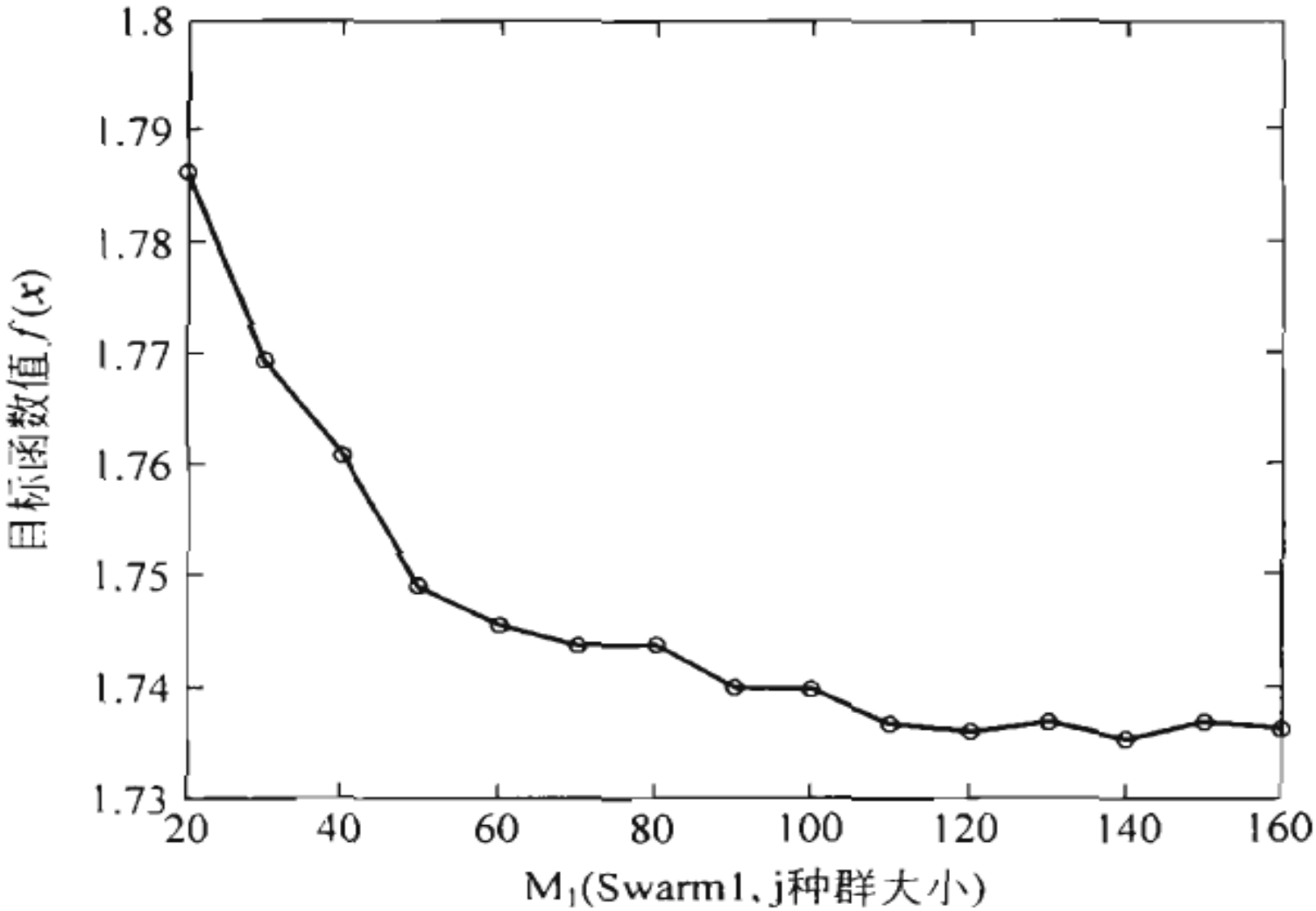


图 4.13　CPSO 平均解随 M_1 变化的动态曲线(焊接条设计问题)

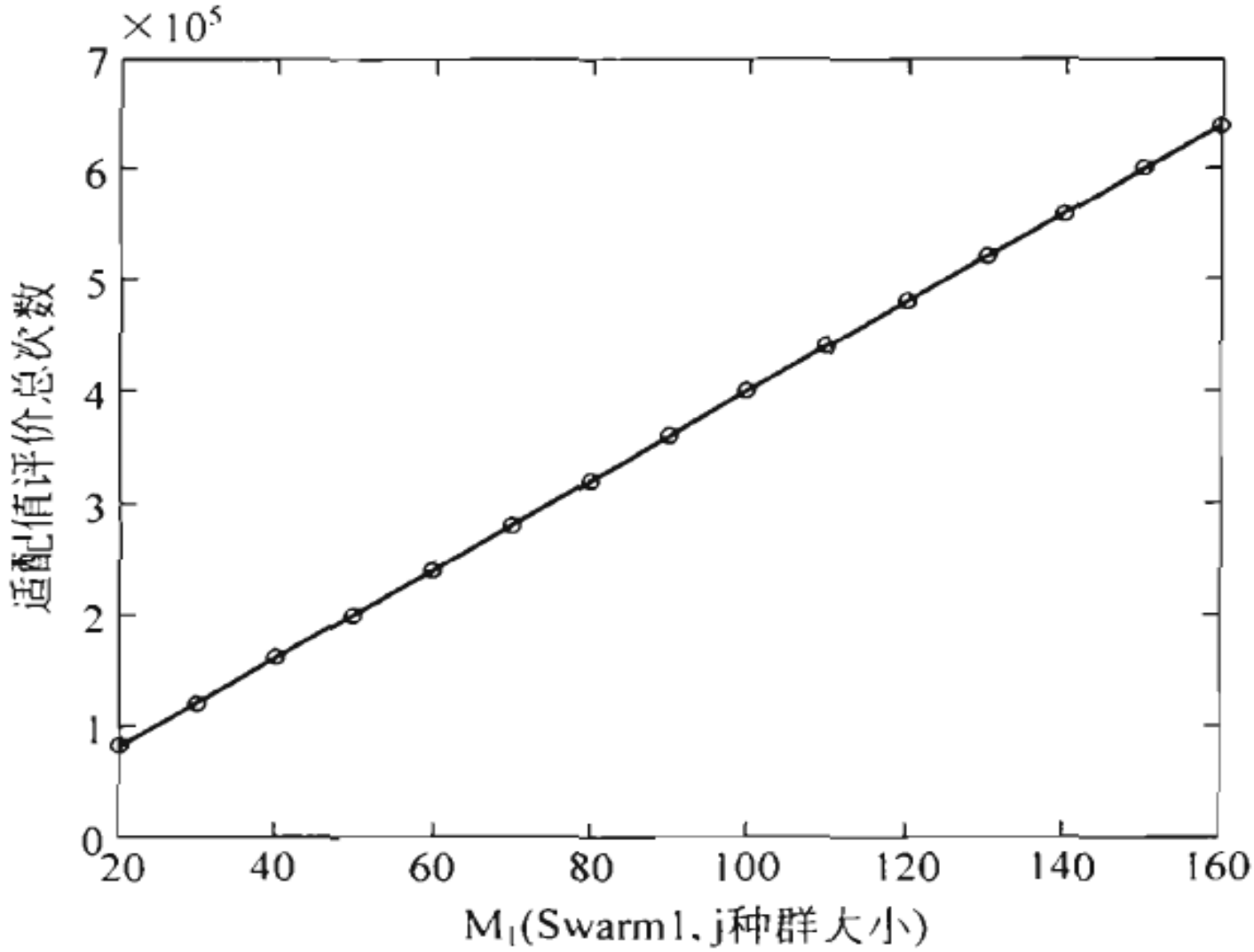


图 4.14　适配值评价总量随 M_1 变化的动态曲线(焊接条设计问题)

- 综合考虑算法搜索质量和计算效率，对于5维以内的问题，建议M1的取值选择在50-70之间。
- 另外，Swarm 2的目的是找到合适的罚因子，由于不是直接在决策解空间进行搜索，因此其种群规模M2过大并不能给算法性能带来明显的提高。考虑到Swarm 2中每个微粒为2维向量，遵循PSO参数选择的原则，M2取20比较合适。
- 此外，在实验过程中还发现G2的增大并不能对搜索的性能有很大改善，因此在CPSO算法中，建议G2取值在10左右。

4. 3. 5 CPSO算法的改进

1）改进措施

措施1　约束违反量的归一化处理

为了消除不同约束间尺度上的差别，在计算、sum_viol时按下式对每个约束的违反量进行了归一化处理

$$sum\_viol = \sum_{i=1}^{N} \frac{G_i(x_j)}{\max_j [G_i(x_j)]}$$

经这种处理后，不同约束的违反量都被限制在0到1之间。

措施II 使目标函数和约束违反量处于同一尺度

引入一个尺度因子，考虑目标与约束违反量之间的平衡，即

$$F_i(\boldsymbol{x}) = f_i(\boldsymbol{x}) + \max_i(f_i(\boldsymbol{x})) \times (sum\_viol \times w_1 + num\_viol \times w_2)$$

措施III 用优良解替代种群中的个体

对于可行域很小的问题，仅仅依靠PSO 算法对微粒的速度和位置进行更新，往往无法搜索到可行解，还甚至会因飞行速度过大而使搜索过程中找到的可行微粒丢失。为了尽可能利用可行解，在每代进化过程中可以将搜索到的优良解(通常是可行解)替换种群中随机选取的部分微粒，譬如用当前找到的最优解替换各

Swarm1, j种群中随机选取的10%的个体。尽管可能失去一定的种群多样性，但可加快进人可行域和找到最优解的速度。

2）数值仿真及分析

表 4.12　焊接条设计问题求解统计结果比较

方法	最优解	平均解	最差解	标准偏差
CPSO	1.728 024	1.748 831	1.782 143	0.012 926
Ⅰ+Ⅱ+Ⅲ	1.724 870	1.725 086	1.726 160	2.9276e−004
Ⅰ	1.725 217	1.728 408	1.734 958	0.002 714 3
Ⅱ	1.725 097	1.730 651	1.735 682	0.003 376 1
Ⅲ	1.725 042	1.729 845	1.745 277	0.006 241 6
Ⅰ+Ⅱ	1.725 735	1.733 143	1.754 402	0.007 853 9
Ⅰ+Ⅲ	1.724 863	**1.725 015**	**1.725 254**	**1.1468e−004**
Ⅱ+Ⅲ	**1.724 860**	1.725 075	1.725 664	2.1692e−004

表 4.13　压力管设计问题求解统计结果比较

方法	最优解	平均解	最差解	标准偏差
CPSO	6061.0777	6147.1332	6363.8041	86.4545
Ⅰ+Ⅱ+Ⅲ	**6059.7143**	6092.2145	6370.7798	67.3090
Ⅰ	*659.3802*	*1926.5345*	*3639.9858*	767.6291
Ⅱ	6059.7341	6082.4961	6151.5268	21.5595
Ⅲ	*1046.4521*	*1127.3367*	*1215.0715*	42.7009
Ⅰ+Ⅱ	6059.7157	**6075.1899**	6090.5412	15.2423
Ⅰ+Ⅲ	*43.9092*	*82.7870*	*140.8168*	27.1442
Ⅱ+Ⅲ	**6059.7143**	6080.1936	**6090.5291**	**14.5697**

表 4.14　伸缩绳设计问题求解统计结果比较

方法	最优解	平均解	最差解	标准偏差
CPSO	0.012 856	0.013 630	0.014 417	5.1011e−4
Ⅰ+Ⅱ+Ⅲ	0.012 718	**0.012 931**	0.013 322	1.6489e−4
Ⅰ	0.013 161	0.013 662	0.014 281	3.3238e−4
Ⅱ	0.012 953	0.013 671	0.014 850	4.9111e−4
Ⅲ	0.012 715	0.012 945	0.013 402	1.7816e−4
Ⅰ+Ⅱ	0.013 166	0.013 832	0.014 744	4.7759e−4
Ⅰ+Ⅲ	0.012 736	0.012 992	**0.013 308**	**1.5667e−4**
Ⅱ+Ⅲ	**0.012 681**	0.012 941	0.013 686	2.5289e−4

函数“Stretching”技术

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：

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

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

- 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）； γ1、γ2和μ是三个任意的正数常量；
- *sign* (•)为常见的符号函数：
- 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

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

做法

1）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。

2）当算法探测到某一局部极值点之后，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信息，被划分为两部分来考虑。

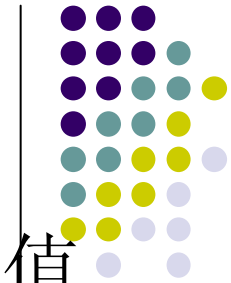
一部分为区域: $S1=\{x \mid f(x)<=f(x')\}$,

另一部分为区域: $S2=\{x \mid f(x)>f(x')\}$

分析8.4与8.5式可知，区域S1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∈S1，均有G(x)=f(x)=H(x)，同样原始目标函数在区域中的极值点（包括全局极小点在内）也始终保留不变。

区域S2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)，

$$G(x) = f(x) + 2\gamma_1 \left\| x - x' \right\|$$



相当于原始目标函数在区域S2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S2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)

$$H(x) = G(x) + \frac{2\gamma_2}{\tanh(\mu(G(x) - G(x')))}$$

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。

- 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。

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

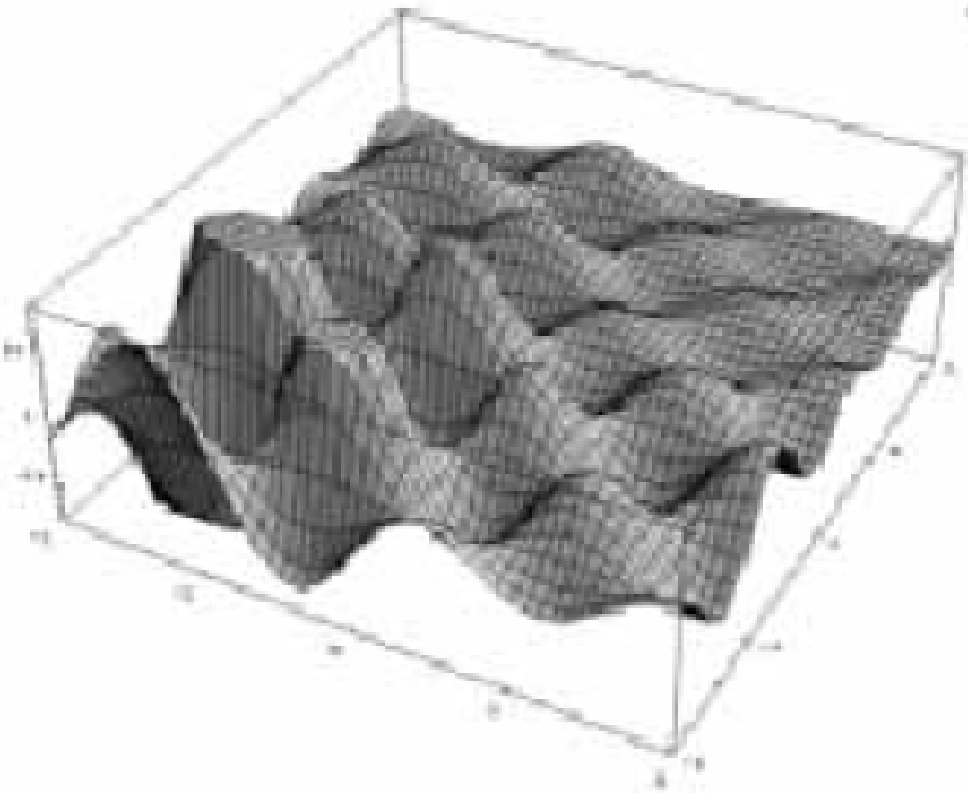


图8.1　Levy No.5函数原始空间曲面图

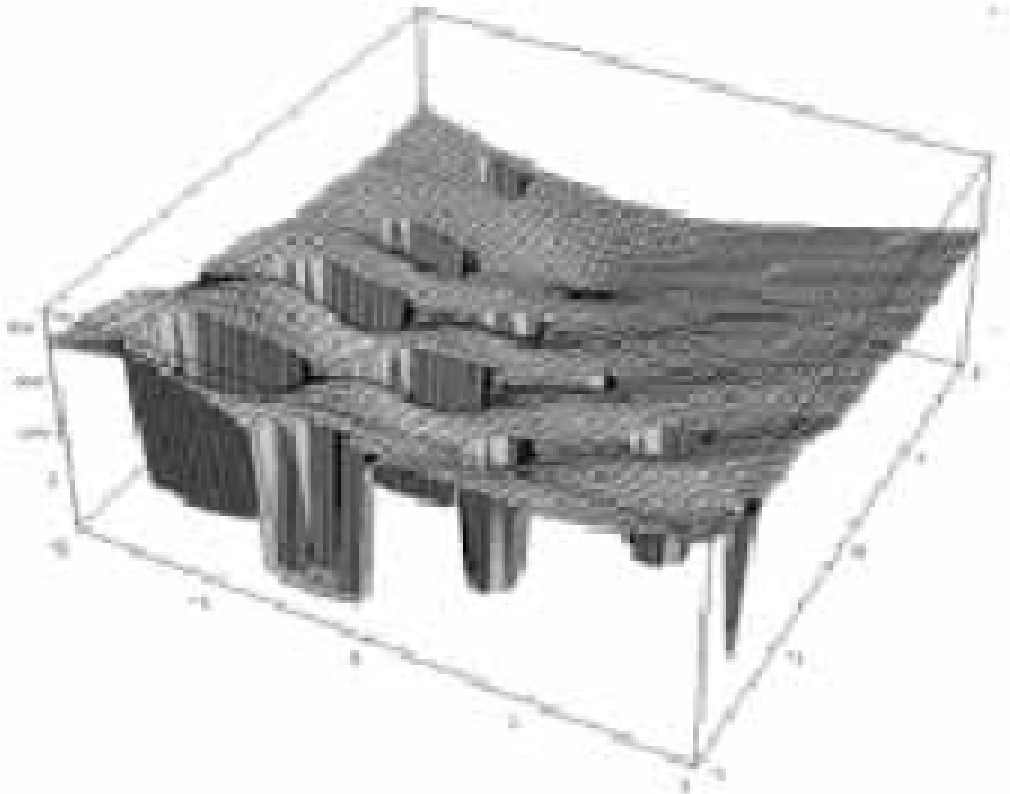


图8.2 Levy No.5函数第一次变换后的空间曲面图

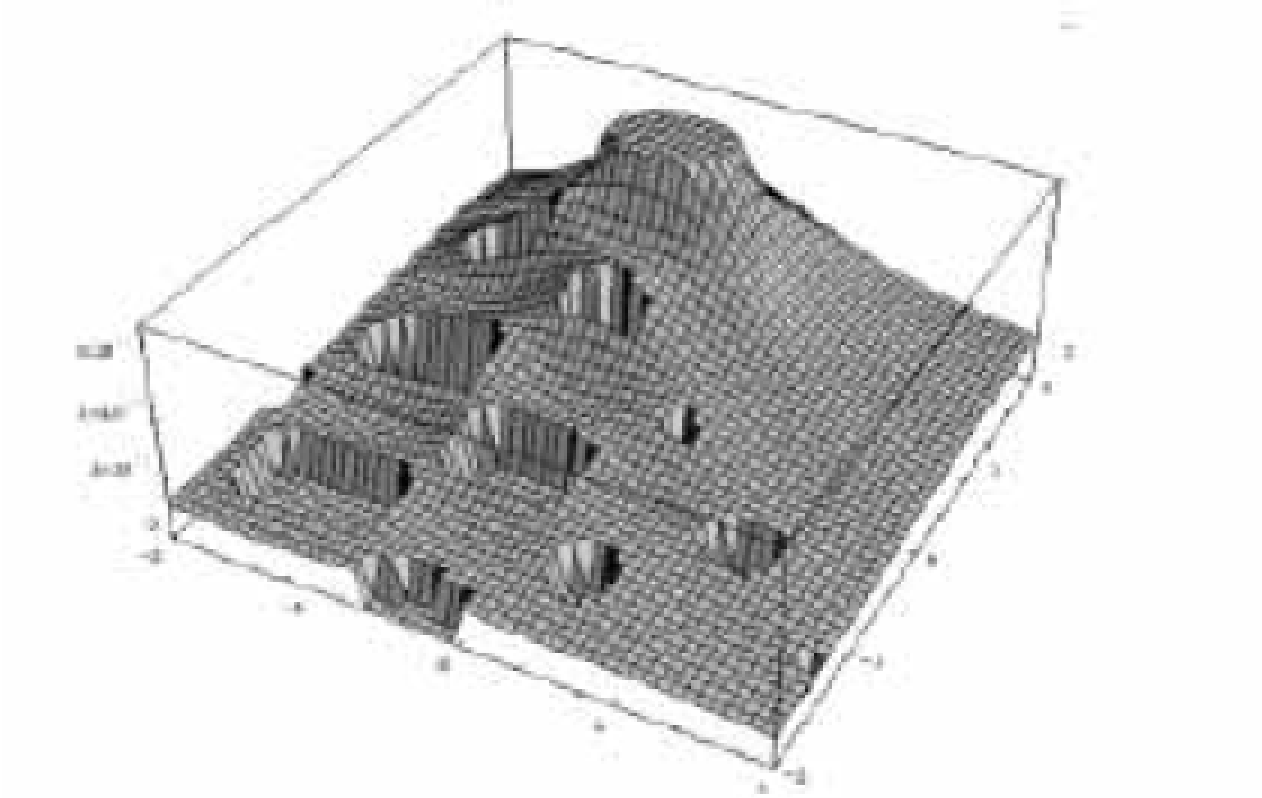
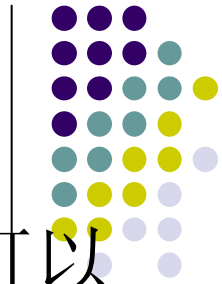


图8.3 Levy No.5函数第二次变换后的空间曲面图



- 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。

SPSO

- 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）。。

流程

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

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]

XOR函数

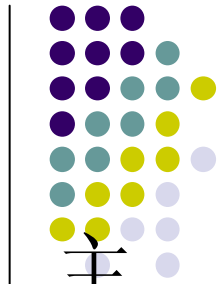
$$f(x) = [1 + \exp(-\frac{x_7}{1+\exp(-x_1 - x_2 - x_5)} - \frac{x_8}{1+\exp(-x_3 - x_4 - x_6)} - x_9)]^{-2}$$

$$+ [1 + \exp(-\frac{x_7}{1+\exp(-x_5)} - \frac{x_8}{1+\exp(-x_6)} - x_9)]^{-2}$$

$$+ [1 - \{1 + \exp(-\frac{x_7}{1+\exp(-x_1 - x_5)} - \frac{x_8}{1+\exp(-x_3 - x_6)} - x_9)\}^{-1}]^2$$

$$+ [1 - \{1 + \exp(-\frac{x_7}{1+\exp(-x_2 - x_5)} - \frac{x_8}{1+\exp(-x_4 - x_6)} - x_9)\}^{-1}]^2$$

实验条件

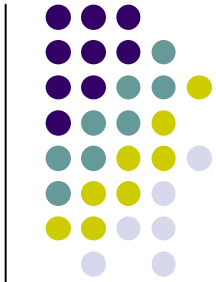


- 实验中采用带有惯性权重的PSO计算模型，主要的实验参数如下： γ1=5000， γ2＝0.5 mu=1e-10, c1=c2=0.5;
- 惯性权重系数*W*随时间递减，从1.0减小至0.4。

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%

小结



1）约束PSO要采用智能约束处理方法

2）CPSO有效

3）SPSO有效

1. BP网络模型

BP网络模型如图2.4所示:

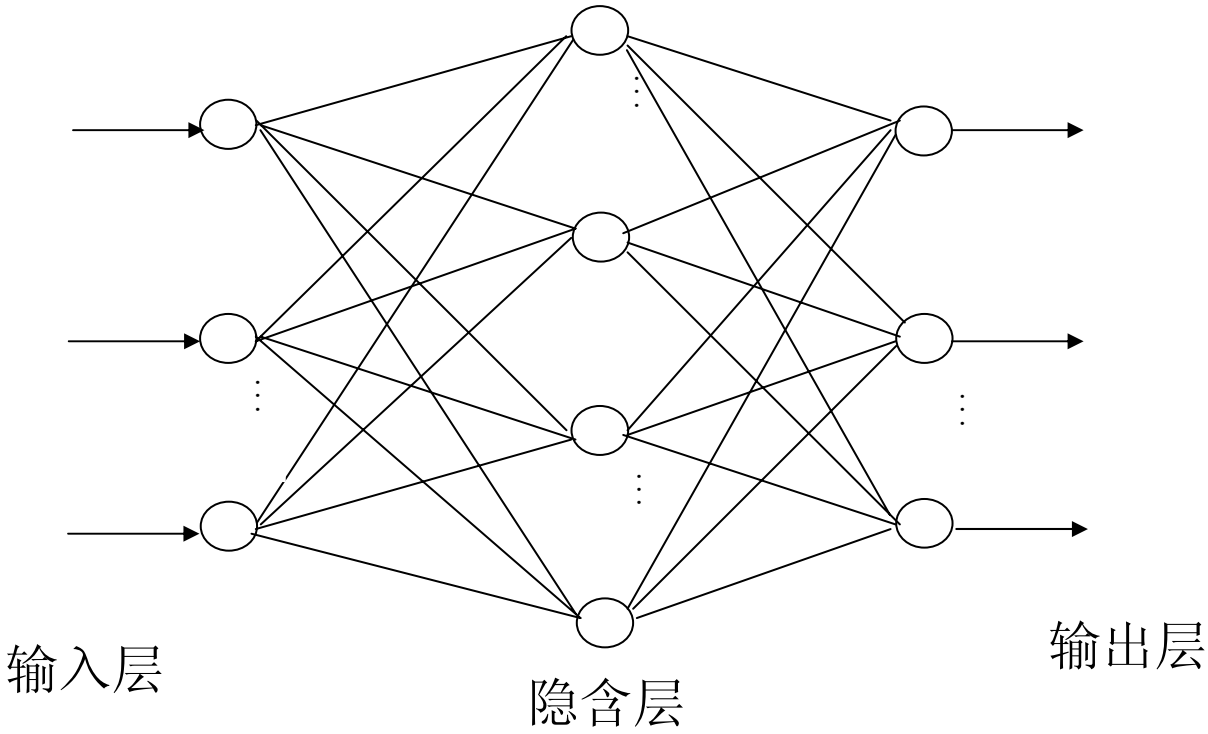
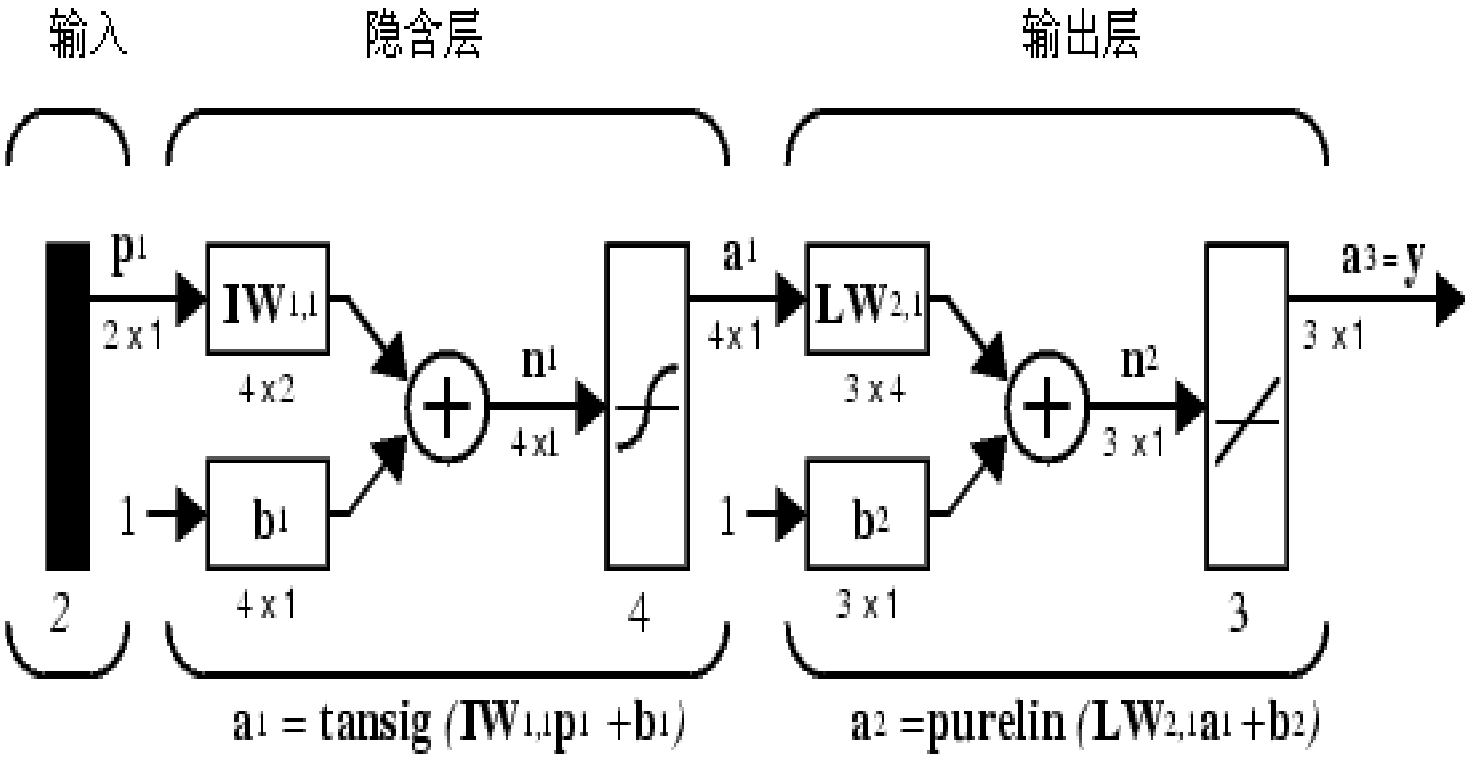


图2.4 多层前向神经网络结构

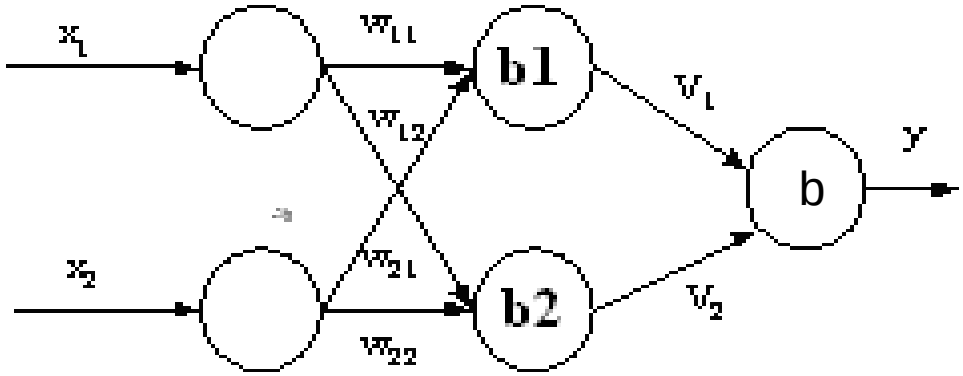
Matlab形式



Nnd11nf.m 描述输出与加权系数的关系

nndemos（chap11 demo1）network function

示例2：两层XOR网络



x1	x2	y
0	0	0
0	1	1
1	0	1
1	1	0

输入输出关系

$$y_i = \sum_{i=1}^{2} v_i f\left(\sum_{j=1}^{2}\left(w_{ij} x_j + b_i\right)\right)$$

$$= \sum_{i=1}^{2} v_i z_i + b$$

$$z_i = f\left(\sum_{j=1}^{2}\left(w_{ij} x_j + b_i\right)\right)$$

$$\overline{z_i} = \sum_{j=1}^{2}\left(w_{ij} x_j + b_i\right)$$

$$i = 1 \sim 4$$

适应度函数

$$J = \sum_{i=1}^{4} (t_i - y_i)^2 -> \min$$

$$f(x) = 1/(1+e^{-x})$$