文章编号: 1000-5641(2015)06-0036-10

一种新的自适应惩罚函数在遗传算法中的应用

蔡海鸾[1,2], 郭学萍[1,2]

(1. 华东师范大学 数学系, 上海 200241;
2. 华东师范大学 上海市核心数学与实践重点实验室, 上海 200241)

摘要: 惩罚函数是遗传算法中解决非线性约束最优化问题最常用的方法之一. 但传统的惩罚函数运用到遗传算法中往往难以控制惩罚因子, 因此本文引进了一种结构简单、通用性强的新自适应惩罚函数, 并证明了其收敛性. 随后构建了基于新自适应惩罚函数的遗传算法, 使得种群能快速进入可行域, 并且提高了遗传算法的局部搜索能力. 理论分析及仿真结果表明该算法具有参数少、稳定性强、收敛快等优点.

关键词: 约束最优化; 惩罚函数; 遗传算法; 自适应惩罚函数
中图分类号: O241-43 **文献标识码**: A **DOI:** 10.3969/j.issn.1000-5641.2015.06.006

A new adaptive penalty function in the application of genetic algorithm

CAI Hai-luan[1,2], GUO Xue-ping[1,2]

(1. *Department of Mathematics, East China Normal University, Shanghai* 200241, *China*;
2. *Shanghai Key Laboratory of PMMP, East China Normal University, Shanghai* 200241, *China*)

Abstract: Penalty function is one of the most commonly used method in genetic algorithm (GA) to solve nonlinear constraint optimization problems. For traditional penalty functions, it is always not easy to control penalty factors. In this paper we present a new adaptive penalty function with simpler construction and prove its convergence. Then based on this adaptive penalty function we present a new genetic algorithm, which can make populations quickly access to feasible regions and improve local search capacity of genetic algorithms. Theoretical analysis and simulation results show that this algorithm has stronger stability and better convergence but needs less parameters than other ones.

Key words: nonlinear constrained optimization; penalty function; genetic algorithm; adaptive penalty function

0 引 言

非线性约束优化问题是优化问题中最复杂的问题之一, 而遗传算法是约束优化中运用最为广泛的一种算法. 它以生物模型为原型, 快速地搜索出解空间中的全体解, 而不会陷入

收稿日期: 2014-11
基金项目: 国家自然科学基金(11471122, 44107310); 上海市科学技术委员会项目(13dz2260400)
第一作者: 蔡海鸾, 女, 硕士研究生, 研究方向为数值最优化. E-mail: caihailuan@qq.com.
通信作者: 郭学萍, 女, 副教授, 研究方向为数值代数、数值优化. E-mail: xpguo@math.ecnu.edu.cn.

局部最优解的快速下降陷阱之中[1], 并且利用它的内在并行性, 可以方便地进行分布式计算, 加快求解速度. 但是遗传算法的局部搜索能力较差, 导致单纯地运用遗传算法比较费时, 从而使得在进化后期搜索效率较低[1]. 在实际应用中, 遗传算法容易产生早熟收敛[2]. 如何既保留优良个体, 又维持群体的多样性, 一直是遗传算法中较难解决的问题. 早期人们提出用惩罚遗传算法来提高种群的适应度, 很好地解决了遗传算法的适应度问题[3], 但是新的问题是, 惩罚函数的惩罚系数往往难以控制, 过大或过小都会导致算法病态[4]. 因此提出一种新的自适应惩罚遗传算法来解决约束优化问题. 根据当前群体中的可行解的比例对目标函数和违反约束条件的程度作出一个合适的权衡, 使得种群快速进入可行区域, 提高遗传算法的局部搜索能力.

1　传统的惩罚函数

惩罚函数是将约束条件的违反度作为惩罚项加到目标函数中, 从而构造出带参数的增广目标函数. 其思想是把一系列的约束优化问题转化成无约束的优化问题求解. 在新的目标函数中, 惩罚因子的不断变化, 导致最优解也不断变化, 最终趋于原问题的最优解. 非线性约束优化问题一般可表示为[5]

$$\min f(x), \quad x=(x_1,x_2,\cdots,x_n)\in \mathbf{R}^n, \tag{1}$$

使得

$$\begin{cases} g_j(x)\leqslant 0, j=1,2,\cdots,q, \\ h_j(x)=0, j=q+1,\cdots,m, \end{cases} \tag{2}$$

其中, $f(x)$为目标函数, $g_j(x)$和 $h_j(x)$分别为不等式约束条件和等式约束条件, $x\in\Omega\subseteq S$. $S\subseteq\mathbf{R}^n$ 是搜索空间, Ω 是可行域, 即满足所有约束条件的空间.

惩罚函数方法即把问题(1)转化为: $\min G(x)=f(x)+c\varphi(x)$. 其中, $G(x)$叫做惩罚函数, c为惩罚因子, $\varphi(x)$称为惩罚项.

一般地, 惩罚项是基于个体到可行域的距离. 个体 x 到第 j 个约束条件的距离可以表示为:

$$\varphi_j(x)=\begin{cases} \max\{0,g_j(x)\}, & 1\leqslant j\leqslant q, \\ \max\{0,|h_j(x)|-\varepsilon\}, & q+1\leqslant j\leqslant m. \end{cases}$$

上式中, ε 是一个正的不等式约束条件容忍值, 记 $\varphi(x)=\sum_{j=1}^{m}\varphi_j(x)$为个体 x 违反约束的程度, 它反映了个体 x 到可行域的距离[6].

引理 1.1[4]　假设 f,g,h 是 $\mathbf{R}^n$ 上的连续函数, 又设对于每一个 c, 都有一个 $x_c\in\mathbf{R}^n$, 使得 $\theta(c)=f(x_c)+c\varphi(x_c)$成立, 则

(1) $\inf\{f(x):g(x)\leqslant 0,h(x)=0\}\geqslant\sup_{c>0}\theta(c)$, 其中$\theta(c)=\inf\{f(x)+c\varphi(c)\}$;

(2) 当 $c>0$ 时, $f(x)$关于 x 单调不降, $\theta(c)$ 关于 c 单调不降, $\varphi(x_c)$ 关于 c 单调不增.

惩罚函数方法可以直接利用无约束最优化问题的算法来求解原约束最优化问题, 因此有简单实用等优点. 由于惩罚函数方法在每一迭代步不涉及可行方向和投影的计算, 因此惩罚函数方法更适合求解非线性约束优化问题. 虽然惩罚函数方法能简单有效地解决非线性约束最优化问题, 但是惩罚函数方法在迭代过程中有时要求惩罚因子趋于无穷大, 从而使得子问题越来越病态, 并且惩罚函数方法的收敛速度也很慢. 在遗传算法中惩罚函数是处理非

线性约束优化最常用且最有效的一种方法, 因此本文构造一种新的自适应惩罚函数来处理遗传算法的约束条件.

2 新的自适应惩罚函数

综合各种观点, 要设计一个好的惩罚函数处理遗传算法的约束条件有以下原则[3]:

(1) 结构简单, 参数尽可能的少;

(2) 能在进化过程中正确权衡目标函数和障碍函数.

考虑一般线性约束规划问题(1), 可以构建如下形式的惩罚函数[3]:

$$G(x) = f(x) + c \times \varphi(x). \tag{3}$$

由惩罚函数的设计原则知, 只需对式(3)的惩罚系数 c 进行设计. 下面分析在约束过程中 c 应该如何变化.

在优化过程中的早期, 群体中没有或只有少量的可行解, 这时, c 应该是相对大的值, 以引导搜索进入可行解区域. 随着进化的进行, 一些可行解将进入群体, 这时, 惩罚应该逐渐减小. 而且, 可行解越多, 惩罚应该越小, 因此 c 应随着可行解的比例(ρ)的增大而逐渐减小. 从而使得搜索策略的重心从搜索可行解转移到搜索好的目标函数上来. 同时, 也可以让有较小目标函数值和较小违法约束条件的非可行解进入群体, 这对于求解最优解在可行域与不可行域的边界上的约束规划问题非常重要. 当群体中全部是可行解时, 惩罚系数 c 减到最小值 0.

从上面分析可以得出: 惩罚系数 c 在优化过程中的变化依赖于当前群体可行解的比例. 记当前可行解的比例为 ρ, 那么系数 c 可以表示为 ρ 的函数 $c(\rho)$. ρ 代表可行解的比例, ρ 越大表示可行解越多, c 应该越小, 因此 $c(\rho)$是一个减函数.

设计的主要问题是寻求满足以上性质的惩罚系数 $c(\rho)$, 综合各种性质, 本文设计函数 $c(\rho)$为

$$c(\rho) = 10^{\alpha(1-\rho)} - 1,$$

其中, $\alpha > 0$ 是一个需要调整的参数, 具体可取 [0, 10] 之间的一个整数. 基于我们设计的 $c(\rho)$, 构建如下新的惩罚函数:

$$G(x) = f(x) + [10^{\alpha(1-\rho)} - 1] \times \varphi(x). \tag{4}$$

新的惩罚函数有如下性质:

(1) $c(\rho)$ 随 ρ 的增大而减小, $c(\rho)$是一个减函数.

(2) 惩罚系数 $c(\rho)$的初始值是随群体中的可行解的比例 ρ 变化而变化的. 如果初始群体中可行解的比例较大, 那么 $c(\rho)$的值就较小, 反之较大. 在优化的过程中我们并没有强加给 $c(\rho)$一个小的或者大的初始值, 而是随种群当前状态所决定.

(3) 当 $\rho = 0$ 时有最大值 $c(0) = 10^{\alpha} - 1$, 这时群体没有可行解, 惩罚系数达到最大, 引导搜索进入可行区域.

(4) 当 $\rho = 1$ 时有最小值 $c(1) = 0$, 这意味着都是可行解, 惩罚系数最小, 非常有利于非可行解进入群体.

(5) 在从 0 到 1 变化的早期过程中, 惩罚系数急剧减小, 一段时间后则缓慢减小. 这可使得搜索重心从可行解快速地转移到目标函数上来.

(6) 在整个优化过程中惩罚系数不会过大或过小, 可以看出这是动态的自适应的.

若新构造的自适应惩罚函数是收敛的, 则新自适应惩罚函数处理约束条件是有效的[7]. 下面分析新自适应惩罚函数的收敛性.

定理 2.1(新自适应惩罚函数的收敛性)　考虑问题(1), 其中 f,g,h 是 $\mathbf{R}^n$ 上的连续函数, 假设此问题有一个可行解, 此外对于每个 α, 无约束最优化问题 $\min G(x,\alpha)$, 存在一个解 $x_\alpha \in \mathbf{R}^n$, 其中 x_α 包含在 $\mathbf{R}^n$ 中的紧子集中, 则

(1) $\inf\{f(x) : g(x) \leqslant 0, h(x) = 0\} = \sup_{\alpha\geqslant 0}\theta(\alpha) = \lim_{\alpha\to\infty}\theta(\alpha)$;

(2) x_α 的任一收敛子列的极限值 x^* 是问题(3)的最优解, 并且当 $\alpha\to\infty$ 时, $[10^{\alpha(1-\rho)}-1]\varphi(x_\alpha)\to 0$.

证　明　由引理 1.1 的(2)知 $\theta(c)$ 是关于 c 的单调不降的函数. 又由于 $c = 10^{\alpha(1-\rho)}-1$ 是关于 α 的单调增函数, 由复合函数的性质知, $\theta(\alpha)$ 是关于 α 的单调不降函数, 由极限的单调有界定理[8], 可知 $\sup_{\alpha\geqslant 0}\theta(\alpha) = \lim_{\alpha\to\infty}\theta(\alpha)$. 下证 $\alpha\to\infty$ 时 $\varphi(x_\alpha)\to 0$.

设问题(1)的惩罚函数 $\min G(x,\alpha)$ 对 $\alpha = 1$ 的最优解是 x_1, 如果对 $\forall\epsilon > 0$ 满足

$$\alpha \geqslant \frac{1}{\epsilon}[f(y) - f(x_1)] + 2,$$

则有 $f(x_\alpha)\geqslant f(x_1)$(证明过程见文 [4], 第 204 页定理 10.1.1).

因此, 当 $\alpha \geqslant \frac{1}{\epsilon}[f(y)-f(x_1)]+2$ 时 $\varphi(x_\alpha)\leqslant 0$; 又 $\epsilon > 0$ 是任意的, 故当 $\alpha\to\infty$ 时, $\varphi(x_\alpha)\to 0$. 现令 x_{α_k} 表示 x_α 的任一收敛子列, x^* 是它的极限, 则

$$\sup_{\alpha\geqslant 0}\theta(\alpha) \geqslant \theta(\alpha_k) = f(x_{\alpha_k}) + [10^{\alpha_k(1-\rho)}-1]\varphi(x_{\alpha_k}) \geqslant f(x_{\alpha_k}).$$

由于 $x_{\alpha_k}\to x^*$ 并且 f 是连续函数, 因此

$$\sup_{\alpha\geqslant 0}\theta(\alpha) \geqslant f(x^*). \tag{5}$$

当 $\alpha\to\infty$ 时即 $\varphi(x_\alpha)\to 0$, $\varphi(x^*) = 0$, 这表示 x^* 是原问题(1) 的可行解. 由式(5)和引理 1.1 的(1)知 x^* 是问题(1)的最优解, 并且 $\sup_{\alpha\geqslant 0}\theta(\alpha) = f(x^*)$. 当 $\alpha\to\infty$ 时

$$[10^{\alpha(1-\rho)}-1]\varphi(x_\alpha) = \theta(\alpha) - f(x_\alpha),$$

$$\theta(\alpha)\to f(x^*), f(x_c)\to f(x^*),$$

则当 $\alpha\to\infty$ 时 $[10^{\alpha(1-\rho)}-1]\varphi(x_\alpha)\to 0$.

3　基于新的自适应惩罚函数的遗传算法

数值方法求解约束最优化问题的主要手段是迭代运算. 一般的数值迭代方法容易陷入局部极小的陷阱而出现"死循环"现象, 使得迭代无法进行. 遗传算法很好地克服了这一缺点, 是一种全局优化算法, 但用于控制染色体的遗传算子通常会产生非线性规划的不可行后代, 因此解决非线性优化的主要问题是如何处理约束条件. 一般有如下几种方法：拒绝方法; 修补方法; 惩罚函数法. 其中惩罚函数法是遗传算法处理非线性约束条件最常用的一种方法, 惩罚项可以取常数也可以取变量, 常数对于解决复杂的约束优化问题效率比较低, 若对每个约束条件都独立给予惩罚因子, 目标函数对参数的依赖将会变得非常的强, 因此本文采用新

的自适应惩罚函数. 遗传算法从含有大量个体的初始种群中搜索最优解, 通过选择算子把具有较好适应值的个体选择出来, 进行交叉变异, 然后扩大搜索空间, 形成下一代种群. 遗传算法中惩罚函数是对任意违反约束条件的个体, 乘以一个惩罚项, 然后加到进化函数中, 使得违反约束的个体的适应度降低, 再选择算子, 生成下一代种群, 从而在种群中保持一定的不可行解, 使得遗传算法从可行解和不可行解两个区域进行搜索, 找到全局最优解.

3.1 适应度函数

遗传算法中使用适应度来度量群体中各个个体在优化计算中有可能达到或接近于最优解的程度. 适应度越大的个体遗传到下一代的概率就越大; 反之, 则概率越小. 度量个体适应度的函数称为适应度函数(Fitness Function). 适应度函数的设计有以下要求[7]:

(1) 单值, 非负, 连续, 最大化;

(2) 合理, 一致性;

(3) 计算量小;

(4) 通用性强.

基于以上要求本文选择以下适应度函数:

(1) 若目标函数为最小值问题, 则

$$\text{fitness}(x) = \max G(x) - G(x);$$

(2) 若目标函数为最大值问题, 则

$$\text{fitness}(x) = G(x) - \min G(x),$$

其中, $G(x) = f(x) + [10^{\alpha(1-\rho)} - 1] \times \varphi(x)$, n 表示种群的大小. 设 p 表示根据目标函数的大小所确定个体在种群中的位置. 例如: 目标函数 $G(x)$最大个体 $p = 1$, 个体记作 x_1; 目标函数 $G(x)$最小个体 $p = n$, 个体记作 x_n.

由以上适应度函数的构造可知, fitness(x)越大, 表明个体越接近最优解. 下记 x_p 的 fitness(x)为 $f(x_p)$.

3.2 交叉算子

交叉算子一般作用在父代两个个体上, 有时也可以作用在两个以上的个体上, 它通过融合父代基因型的信息来产生新的子代基因型. 根据适应度 fitness(x)的大小, 用轮盘赌选择算子从种群$P(t)$(t 表示当前种群的代数)中选出 n(n 为偶数)个父母点(设父母点的集合为 P), 对每对父母点x_i和x_j构造新的交叉算子, 两个后代按以下方式产生:

$$\begin{cases} \sigma_i = \dfrac{r_{11}}{r_{11} + r_{12}} x_i + \dfrac{r_{12}}{r_{11} + r_{12}} x_j; \\ \sigma_j = \dfrac{r_{21}}{r_{21} + r_{22}} x_i + \dfrac{r_{22}}{r_{21} + r_{22}} x_j. \end{cases}$$

上式中 $\sigma_i, \sigma_j \in p(t+1), p(t+1)$为杂交后代. $r_{ij}(i = 1, 2; j = 1, 2)$为 [0,1] 间的随机数.

3.3 变异算子

简单遗传以初始种群为基点, 经过选择、交叉、变异等操作生成新的种群, 以此更新种群, 直到满足终止条件. 遗传算法中变异算子的具体操作是以 pm 为变异概率, 用新的基因代

替原有的基因, 从而产生新的个体. 变异算子对个体编码串的基因做了局部改变, 维持种群的多样性, 有利于防止种群出现早熟现像.

常见的变异算子主要有基本变异、均匀变异以及高斯变异等等(可参考文[7]), 从这些变异算子的设计中可以看出变异算子的设计会涉及变异点的位置确定和基因值的替换两方面. 综合以上变异操作, 为了保证点跳出局部最优, 我们做如下变异操作:

$$\begin{cases} \bar{x_j} = x_j + \dfrac{(u_j - l_j)}{t} r, & \text{若 } r \leqslant 0.5; \\ \bar{x_j} = x_j - \dfrac{(u_j - l_j)}{t} r, & \text{若 } r > 0.5. \end{cases}$$

其中, r 是 [0,1] 之间的一个随机数. $x_j \in p(t+1)$, u_j, l_j 分别是搜索空间中 x_j 的上界与下界.

3.4 基于新的自适应惩罚函数的遗传算法

基于问题(1)本文做出一个详细的研究, 算法具体操作如下:

(1) 将原约束优化问题(1)转化为无约束优化问题, 构建新的自适应惩罚函数

$$G(x) = f(x) + [10^{\alpha(1-\rho)} - 1] \times \varphi(x),$$

其中 ρ 为可行解在种群中所占的比例, 且 $\varphi(x) = \sum_{j=1}^{n} \varphi_j(x)$,

$$\varphi_j(x) = \begin{cases} \max\{0, g_j(x)\}, & 1 \leqslant j \leqslant q; \\ \max\{0, |h_j(x)| - \epsilon\}, & q+1 \leqslant j \leqslant n. \end{cases}$$

(2) 给定搜索精度 ϵ_1, ϵ_2, 初始化种群大小 n 及繁殖概率, 交叉概率 pc, 变异概率 pm, 进化代数 t.

(3) 计算每一个体的适应度: 采用如下函数作为适应度函数

$$\text{fitness}(x) = \max G(x) - G(x).$$

(4) 初始化种群: 随机生成 n 个初始值作为初始种群.

(5) 自适应产生 ρ: 初始 $s = 0$, 若 $\varphi(x_i) < \epsilon_1$, 则 $s = s + 1, \rho = \frac{s}{n}$, 获取最优个体 $x^{(k)}$.

(6) 若 $||x^{(k)} - x^{(k-1)}|| \leqslant \epsilon_2$, 则输出最优解, 否则转下一步.

(7) 选择繁殖操作: 将种群中的个体按适应度由大到小排序, 然后根据单个个体所对应的适应度确定其繁殖后在交配池中所占的比例 $p_i = \frac{f(x^{(i)})}{\sum_{i=1}^{n} f(x^{(i)})}$.

(8) 产生 n 个 0 到 1 之间的随机数, 依据该随机数出现在上述哪一个概率区域内来确定各个个体被选中的次数.

(9) 交叉操作: 交叉运算是遗传算法中产生新个体的主要操作过程, 它以某一概率相互交换某两个个体或两个以上个体的部分染色体.

本例采用第 3.2 节的交叉方法, 其具体操作过程如下：

① 对每串产生 [0,1] 间随机数 r, 若 $r < pc$, 则该串参加交叉操作, 如此选出参加交叉的一组后, 随机配对.

② 对每一对, 产生 [0,1] 间的随机数 $r_{ij}, i = 1, 2; j = 1, 2$. 两个后代按以下方式产生:

$$\begin{cases} \sigma_i = \dfrac{r_{11}}{r_{11} + r_{12}} x_i + \dfrac{r_{12}}{r_{11} + r_{12}} x_j; \\ \sigma_j = \dfrac{r_{21}}{r_{21} + r_{22}} x_i + \dfrac{r_{22}}{r_{21} + r_{22}} x_j. \end{cases}$$

(10) 变异操作: 设变异概率为 pm, 则每一位基因值以等概率变异. 对每一串中的每一位产生 [0, 1] 间的随机数 r, 若 $r < pm$, 则该位发生变异, 本文按(3.3)方式进行变异, 具体操作如下:

$$\begin{cases} x_i = \sigma_i + \dfrac{(u_i - l_i)}{t} r, & \text{若 } r \leqslant 0.5; \\ x_i = \sigma_i - \dfrac{(u_j - l_i)}{t} r, & \text{若 } r > 0.5. \end{cases}$$

$$\begin{cases} x_j = \sigma_j + \dfrac{(u_j - l_j)}{t} r, & \text{若 } r \leqslant 0.5; \\ x_j = \sigma_j - \dfrac{(u_j - l_j)}{t} r, & \text{若 } r > 0.5. \end{cases}$$

(11) 若种群未达到最大进化代数, 则转至(3), 否则此时种群中适应度最大的个体所对应的目标函数值为全局最优解, 输出最优解.

由于算法的收敛性决定算法的有效性, 下面分析新自适应遗传算法的全局收敛性.

3.5 新自适应遗传算法的收敛性

根据 Banach 压缩原理, 如果能够找到一个合适的度量空间, 使得新算法成为一个压缩映射, 那么, 就可以证明, 任意选取初始群体 $P(0)$, 新算法收敛于某个唯一的不动点 x^*, 下面具体分析新自适应遗传算法的收敛性.

定义 3.1[9] 设 X 是一个非空集合, 若 d 是一个 $X \times X$ 到 $\mathbf{R}$ 的映射, 并且对于 $\forall x, y, z \in X$ 满足:

(1) $d(x, y) \geqslant 0$, 当且仅当 $x = y$ 时 $d(x, y) = 0$;

(2) $d(x, y) = d(y, x)$;

(3) $d(x, y) \leqslant d(x, z) + d(z, y)$.

则称 d 为 X 上的度量, 称 (X, d) 为度量空间.

定义 3.2[9] 设 (X, d) 为度量空间, $\{x_n\}$ 是度量空间 (X, d) 中的序列, 若 $\forall \varepsilon > 0$, 存在一个正整数 N, 使得对一切 $m, n > N$, 有 $d(x_m, x_n) < \varepsilon$, 则称序列 $\{x_n\}$ 是 (X, d) 中的 Cauchy 序列, 若其中的每一个 Cauchy 序列都收敛, 则称 (X, d) 为完备度量空间.

定义 3.3[10] 设 (X, d) 为完备度量空间, 对于映射 $f : X \to X$, $\exists \varepsilon \in [0, 1)$, 使得对 $\forall x, y \in X$, 满足: $d(f(x), f(y)) \leqslant \varepsilon \cdot d(x, y)$, 则称 f 为压缩映射.

定理 3.4(Banach 压缩映射原理)[10] 设 (X, d) 为完备度量空间, $f : X \to X$ 为压缩映射, 则 f 有且仅有一个不动点 $x^* \in X$, 并且对于 $\forall x_0 \in X$ 满足: $x^* = \lim_{k \to \infty} f^k(x_0)$. 其中 $f^0(x_0) = x_0, f^{k+1}(x_0) = f(f^k(x_0))$.

定理 3.5(新算法的收敛性) 考虑问题(1). 设 S 是所有群体的集合, 若存在映射 $\delta : S \times S \to \mathbf{R}$, 使得 (S, δ) 是一个完备的度量空间, 又设存在映射 $g : S \to S$, 使得 g 是压缩映射, 则对任意选取的初始群体 $P(0)$, 新算法都全局收敛于某个唯一的不动点 x^*.

证 明 设 P 为新遗传算法的一个种群, 种群大小为 n, 即 $P = \{x_1, x_2, \ldots, x_n\}$. 种群 P 的平均适应度函数为: $F(P) = \frac{1}{n} \Sigma_{x_i \in P} f(x_i)$. 求解问题(1)的最小值, 定义

$$\delta(P_1, P_2) = \begin{cases} 0, & P_1 = P_2; \\ |f_{\min}(P_1) + M - F(P_1)| + |f_{\min}(P_1) + M - F(P_2)|, & P_1 \neq P_2. \end{cases}$$

其中, $f_{\min}(P_1)$ 是第一代中最小的适应度, M 是 F 的上确界.

(1) $\delta(P_1,P_2)=\delta(P_2,P_1),\forall P_2,P_1\in S$;

(2) $\delta(P_1,P_2)\geqslant 0,\forall P_2,P_1\in S$;

(3) $\delta(P_1,P_2)+\delta(P_2,P_3)\geqslant\delta(P_1,P_3),\forall P_2,P_1,P_3\in S$.

因此, (S,δ)是一个度量空间. 由于 S 是有限集合, 所以 S 中的任意 Cauchy 序列都收敛, 那么, (S,δ) 是一个完备的度量空间.

根据遗传算法的选择原理[1], $F(P(t+1))>F(P(t))$, 即种群的适应度随着代数的增加而增大. 所以, 对于映射 g, $\exists\varepsilon=\frac{2M}{2M+2f_{\min}(P_1)}\in(0,1)$ 使得:

$$\begin{aligned}\delta(g(P_1(t)),g(P_2(t)))&=|f_{\min}(P_1)+M-F(g(P_1(t)))|+|f_{\min}(P_1)+M-F(g(P_2(t)))|\\&<\varepsilon\cdot(|f_{\min}(P_1)+M-F(P_1(t))|+|f_{\min}(P_1)+M-F(P_2(t))|)\\&=\varepsilon\cdot\delta(P_1(t),P_2(t))\ (0<\varepsilon<1).\end{aligned}$$

所以, g 是一个压缩映射, 由 Banach 压缩映射原理可知, 该新的自适应遗传算法收敛于唯一的不动点, 即 $P^*=\lim_{t\to\infty}g^t(P(0))$. 并且, P^* 与初始种群的选择无关. 因此, P^* 是全局最优解.

4　数值实验

以下给出两组测试函数:

(1) $g1$[11]

$$\min f(x)=5\sum_{i=1}^{4}x_i-5\sum_{i=1}^{4}x_i^2-\sum_{i=1}^{4}x_i^{13},$$

使得

$$\begin{cases}g_1(x)=2x_1+2x_2+x_{10}+x_{11}-10\leqslant 0,\\g_2(x)=2x_1+2x_3+x_{10}+x_{12}-10<0,\\g_3(x)=2x_2+2x_3+x_{11}+x_{12}-10\leqslant 0,\\g_4(x)=-8x_4+x_{10}\leqslant 0,\\g_5(x)=-8x_2+x_{11}\leqslant 0,\\g_6(x)=-8x_3+x_{12}\leqslant 0,\\g_7(x)=-2x_4-x_5+x_{10}\leqslant 0,\\g_8(x)=-2x_4+x_7+x_{11}\leqslant 0,\\0\leqslant x_i\leqslant 1\ (i=1,2,\ldots,9),\\0\leqslant x_i\leqslant 100\ (i=10,11,12),0\leqslant x_{13}\leqslant 1.\end{cases}$$

已知最优精确解为 $x^*=(1,1,1,1,1,1,1,1,1,3,3,3,1),f(x^*)=-15.$

(2) $g2$[12]

$$\min f(x)=\mathrm{e}^{x_1x_2x_3x_4x_5},$$

使得

$$\begin{cases} h_1(x) = x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 - 10 = 0, \\ h_2(x) = x_2x_4 - 5x_4x_5 = 0, \\ h_3(x) = x_1^2 + x_2^2 + 1 = 0, \\ -2.3 \leqslant x_i \leqslant 2.3\ (i = 1, 2), \\ -3.2 \leqslant x_i \leqslant 3.2\ (i = 3, 4, 5). \end{cases}$$

下面是实验结果分析.

数值实验在 matlab 中完成, 运行代数为 1 000, 每个测试在相同条件下独立运行 30 次, 记录其最优值, 平均值, 最差的值以及 cpu 运行时间. 本文选取了三种目前较好的最优算法作为比较, 文献 [12] 中的随机排序法(简称 RY), 文献 [13] 中的自适应惩罚函数法(简称 HW), 文献 [14] 中的静态函数法(简称 HF). 结果如表 1 所示.

表 1 新算法与其他算法的结果比较

Tab. 1 Results comparing our new method with other methods

测试函数	Status	RY	FW	HF	新算法	精确值
	参数的个数	2	1	13	1	
g1	最优值	−15.000	−15.000	−15.000	−15.000	−15.000
	平均值	−15.000	−15.000	−14.998	−15.000	−15.000
	最差值	−14.886	−14.995	−14.489	−14.998	−15.000
	标准方差	0.0E−00	0.0E−00	0.3E−00	0.0E−00	0
	运行时间	0.003 1 t	0.368 0 t	0.004 2 t	0.002 1 t	
g2	最优值	0.053 905	0.053 905	0.053 905	0.053 905	0.053 905
	平均值	0.053 905	0.053 905	0.053 905	0.053 905	0.053 905
	最差值	0.053 899	0.053 905	0.053 968	0.053 905	0.053 905
	标准方差	0.0E−00	0.3E−00	0.7E−00	0.0E−00	0
	运行时间	0.002 8 t	0.003 6 t	0.004 0 t	0.001 9 t	

根据表 1 所得数据结果对 $g1, g2$ 做详细分析.

(1) 新算法与 RY, FW, HF 相比, 这几种方法都能得出最优解, 但新算法参数显然较 RY, HF 少. 下面对参数的敏感性进行分析.

(2) 对参数的敏感性分析如表 2.

表 2 参数的敏感性比较

Tab. 2 Comparison of the sensitivity of parameters

参数	RY	FW	HF	新算法
第 1 次变动	−14.88	−15.00	−13.68	−15.00
第 2 次变动	−15.00	−14.98	−14.99	−15.00
第 3 次变动	−13.36	−14.86	−11.89	−15.00
第 4 次变动	−15.00	−14.89	−14.93	−15.00
第 5 次变动	−14.96	−14.99	−14.46	−14.99
第 6 次变动	−15.00	−14.99	−14.97	−15.00
第 7 次变动	−13.80	−15.00	−14.99	−15.00
第 8 次变动	−15.00	−14.98	−14.99	−15.00

从表 2 中可以看出新方法对参数的依赖性较小, 随参数的变动, 最优解基本保持稳定. RY, HF 最优解的变化较大, 显然新方法敏感性优于 RY, HF.

(3) 综合(1), (2), 知 FW 与新算法都比 RY, HF 参数少, 且对参数的敏感性较弱. 以下分析 FW 与新算法的稳定性. 由本文第 2 部分所列新的自适应函数的性质可知, 可行解的收敛性及稳定性可观察其 ρ(row) 的分布, 当 ρ(row) 达到 1 时表示, 群体全部收敛到可行解[15]. 而由本文第 3 部分对新自适应惩罚函数的遗传算法的介绍, 可知适应度代表最优解与精确解之间的比例, 因此当适应度为 1 时, 群体中的最优解为精确解.

图 1、图 2 为实验结果图.

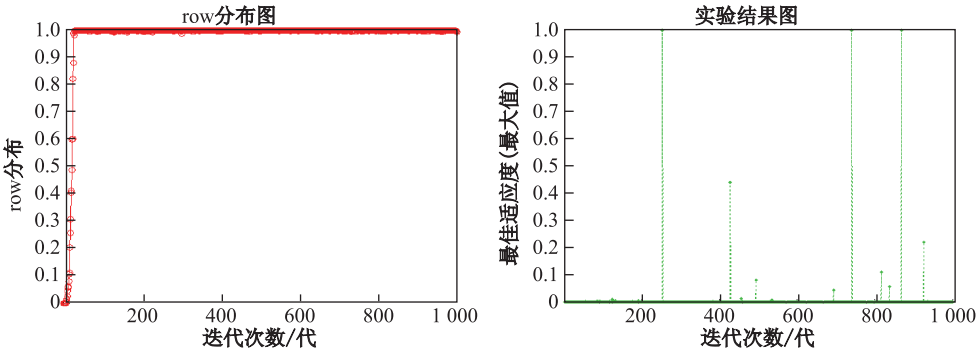

图 1　新算法

Fig. 1　New method

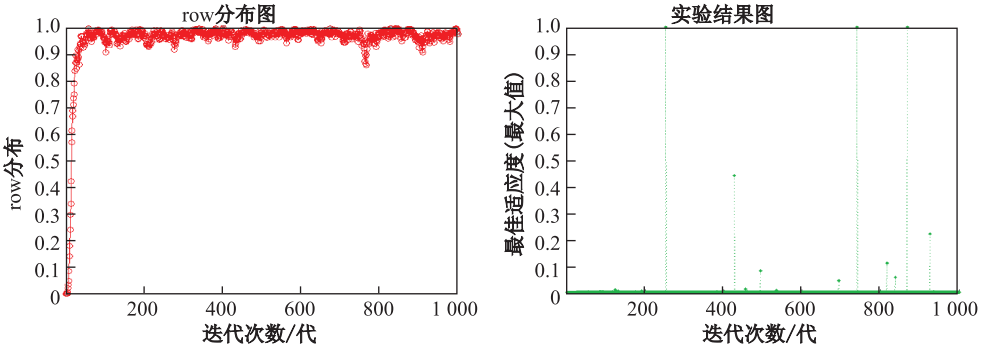

图 2　FW 算法

Fig. 2　FW method

从实验结果上可以看出, 新算法很快就全部收敛到可行域中, 并且保持平稳, 而 Farmani 和 Wright 提出的自适应惩罚法, 波动明显, 证明其稳定性弱, 显然新算法的收敛性以及稳定性都强于 FW.

综上所述新算法对 $g1, g2$ 都能得出最优解, 参数较 RY, HF 少, 且对参数的依赖性小, 与 Farmani 和 Wright 提出的自适应惩罚法比较, 新方法的稳定性更强. 新算法与 RY, FW, HF 相比取得了绝对优势, 因此新算法有着非常大的潜力解决约束优化问题.

[参　考　文　献]

[1] 程润伟, 玄光男. 遗传算法与工程优化 [M]. 周根贵, 译. 北京: 清华大学出版社, 2004: 5-13.

[2] RASHEED K. An adaptive penalty approach for constrained genetic algorithm optimization [C]//Proceedings of the 3rd An-null Genetic Programming Conference. San Francisco, CA, USA: Morgan, 1998: 584-590.

(下转第 52 页)