文章编号: 1001-0920 (2015) 05-0917-06 DOI: 10.13195/j.kzyjc.2014.0647

改进差分进化算法在可靠性冗余分配问题中的应用

刘宏志[1,2], 高立群[1], 孔祥勇[1], 欧阳海滨[1]

(1. 东北大学 信息科学与工程学院, 沈阳 110004; 2. 辽宁工程技术大学 电气与控制工程学院, 辽宁 葫芦岛 125105)

摘 要: 提出一种改进差分进化算法 (IDE), 以解决系统可靠性冗余分配问题. 在罚函数法的基础上, 对约束处理方法进行改进. 新约束处理方法在搜索过程中不需要在每一步都计算惩罚函数值, 加快了寻优速度. 具有良好的通用性, 可以引入到其他智能优化算法中. 将改进的算法用于求解 4 类典型的系统可靠性冗余分配问题, 实验结果表明了所提出的改进算法具有很好的寻优精度和收敛速度.

关键词: 差分进化算法; 约束处理; 可靠性冗余分配问题

中图分类号: TP391 **文献标志码:** A

Improved differential evolution algorithm for solving reliability redundancy allocation problem

LIU Hong-zhi[1,2], *GAO Li-qun*[1], *KONG Xiang-yong*[1], *OUYANG Hai-bin*[1]

(1. College of Information Science and Engineering, Northeastern University, Shenyang 110004, China; 2. Faculty of Electrical and Control Engineering, Liaoning Technical University, Huludao 125105, China. Correspondent: LIU Hong-zhi, E-mail: 19048851@qq.com)

Abstract: A new improved differential evolution(IDE) algorithm is proposed for solving the system reliability redundancy allocation problem. The constraints handling method is improved based on the punishing function method. The new constraints handling method doesn't need to calculate the every punishing function value in the search process, which greatly speeds up the searching for the optimized solutions. The proposed method has good generality, which can be introduced completely to other intelligent optimization algorithms. Using the improved algorithm to solve four typical system reliability redundancy allocation problems, the experimental results show that the algorithm has good optimization precision and convergence speed.

Keywords: differential evolution algorithm; constraints handling; reliability redundancy allocation problem

0 引 言

可靠性优化着眼于确定一个系统结构, 利用更小的预算或更少的资源获得更高水平的可靠性, 通常描述为带有成本、重量、体积等约束的非线性规划问题. 为了获得更高的系统可靠性, 通常有两种主要的方法: 一是增强每个系统组件的可靠性; 二是在不同的子系统中使用冗余元件, 也被称作可靠性冗余分配问题 (RAP)[1].

20 世纪 70 年代以来, 系统可靠性优化问题引起人们的高度重视, 并且已通过传统方法取得了许多可喜的成果[2]. 然而, 作为具有较高计算复杂度的非线性约束问题, RAP 问题的求解有很大的难度. 为了解决该问题, 基于启发式算法的求解方法不断涌现, 如遗传算法[3]、免疫算法[4]、模拟退火算法[5]、和声搜索算法[6]、粒子群算法[7]、帝国竞争算法[8]和布谷鸟算法[9-10]等. 这些研究主要是通过改进算法来提高求解的精度, 而没有涉及其约束的处理. 处理约束优化问题最常见、最有力的工具是罚函数法, 它在实际中得到了广泛的应用, 但也存在一定的不足, 即在搜索过程中每一步都需要计算惩罚函数值, 当约束较多、较复杂时会大大降低寻优速度.

本文提出一种改进的差分进化算法, 并基于罚函数法给出一种新的约束处理方法, 可以提高寻优效率, 缩短寻优时间. 新的方法具有很好的通用性, 完全可

收稿日期: 2014-04-29; **修回日期:** 2014-08-28.

作者简介: 刘宏志(1977−), 男, 讲师, 博士生, 从事数学建模与智能优化理论的研究; 高立群(1949−), 男, 教授, 博士生导师, 从事模式识别和智能优化理论等研究.

以方便地引入到其他启发式算法. 将本文所提算法用于求解4类典型的RAP问题, 实验结果表明了该算法具有很好的寻优精度和收敛速度.

1 改进的差分进化算法

1.1 基本差分进化算法

差分进化算法(DE)是Storn等[11]于1995年提出的一种基于群体智能的启发式算法, 包括变异、交叉和选择3个重要操作. 对于规模为N的种群而言, DE算法的3个主要操作如下所示.

1) 变异. 对于第i个目标向量, 以最常用的DE/rand/1算子为例, 在当前解向量中, 随机选取3个不同于当前目标向量的解向量, 试验向量的构建如下:

$$v_{i,j}^{k+1}=x_{i_3,j}^{k}+F(x_{i_1,j}^{k}-x_{i_2,j}^{k}),\ j=1,2,\cdots,D. \quad (1)$$

其中: F为尺度因子, 用来控制差向量$x_{i_1,j}^{k}-x_{i_2,j}^{k}$的缩放尺度, 其取值范围通常为$[0,2]$; D为所求解问题自变量的维数.

2) 交叉. 记交叉后的解向量为u_i^{k+1}, u_i^{k+1}中的元素根据概率从父代个体x_i^k和试验向量v_i^{k+1}中随机选择, 即

$$u_{i,j}^{k+1}=\begin{cases}v_{i,j}^{k+1},\ \text{rand}()<\text{CR or } j=r_{1\sim D};\\ x_{i,j}^{k},\ \text{otherwise.}\end{cases} \quad (2)$$

其中: $r_{1\sim D}$是区域$[1,D]$内的随机整数; CR是交叉率, 取值范围为$[0,1]$.

3) 选择. DE算法采取贪婪策略对解向量进行选择, 即只有在u_i^{k+1}优于x_i^k的情况下, u_i^{k+1}才会被保留到下次迭代, 具体过程如下所示:

$$x_i^{k+1}=\begin{cases}u_i^{k+1},\ f(u_i^{k+1})<f(x_i^k);\\ x_i^k,\ \text{otherwise.}\end{cases} \quad (3)$$

1.2 改进的差分进化算法

"DE/rand/2"算子和"DE/best/2"算子是DE及改进算法中的两种典型变异策略, "DE/rand/2"算子侧重于对解空间的全局搜索, 而"DE/best/2"算子侧重于搜索速度的提高. 本文将两类算子融合, 提出一种改进策略, 具体如下:

$$v_{i,j}^{k+1}=x_{i_5,j}^{k}+F(x_{i_1,j}^{k}+x_{i_2,j}^{k}-x_{i_3,j}^{k}-x_{i_4,j}^{k}),$$
$$j=1,2,\cdots,D. \quad (4)$$

其中: i_1、i_2、i_3、i_4、i_5是从$\{1,2,\cdots,N\}$中随机选取的不同的整数, $x_{i_5,j}^{k}=\arg\{\min[f(x_{i_1,j}^{k}),f(x_{i_2,j}^{k}),f(x_{i_3,j}^{k}),f(x_{i_4,j}^{k}),f(x_{i_5,j}^{k})]\}$. 变异过程包含了种群中5个解向量, 能够较全面地包含进化过程中种群的个体信息, 提高全局搜索效率; 同时随机使用5个解向量中适应度值最优的$x_{i_5,j}^{k}$作为基向量进行引导, 加强局部寻优能力, 故改进策略的性能介于"DE/rand/2"与"DE/best/2"之间, 具有较好的全局寻优和局部寻优能力.

在基本DE算法中, 尺度因子F为常数, 为提高算法性能, 本文利用Logistic映射产生的混沌序列来确定DE算法的尺度因子.

Logistic方程表示为

$$y(k)=\mu\times y(k-1)\times[1-y(k-1)]. \quad (5)$$

其中: k是迭代次数; $y(0)$的取值范围为$[0,1]$; μ是控制参数, $0\leqslant\mu\leqslant 4$, μ的取值控制着混沌序列的变化情况. 本文取$\mu=4$, $y(0)\notin\{0,0.25,0.5,0.75,1.0\}$.

令尺度因子为

$$F(k)=$$
$$[(F_{\max}-F_{\min})(k/k_{\max})+F_{\min}]y(k). \quad (6)$$

其中: $k_{\max}$为最大迭代次数; $F_{\min}$和$F_{\max}$分别为设定的$F(k)$的最小值和最大值; $F(k)$为迭代次数k的函数, 随迭代次数的增加不断变化.

2 处理约束的新方法

实际优化问题通常情况下都含有约束, 问题越复杂, 约束越多. 一般约束优化问题的数学模型可以表示为

$$\max\ f(x);$$
$$\text{s.t. } g_j(x)\leqslant 0,\ j=1,2,\cdots,n_g. \quad (7)$$

其中: $f(x)$代表目标函数, $g_j(x)$代表第j个约束条件, n_g代表约束条件的个数.

处理约束问题最简单常见的方法是罚函数法, 该方法通过惩罚策略将带有约束的优化问题转化为无约束优化问题, 表示为

$$\max\ J(x)=f(x)-\lambda\sum_{i=1}^{n_g}\max(0,g_i), \quad (8)$$

其中λ代表惩罚系数, 本文中设置为10^5. 因为$J(x)$的最大值问题可以等效为$-J(x)$的最小值问题, 式(8)可以表示为

$$\min\ J(x)=-f(x)+\lambda\sum_{i=1}^{n_g}\max(0,g_i). \quad (9)$$

在应用罚函数法处理约束问题时, 寻优的每一步通常都需要计算罚函数值, 当约束较多、较复杂时, 计算$J(x)$的时间要远远多于计算$f(x)$的时间, 这大大增加了计算时间, 降低了寻优效率. 针对该问题, 本文提出一种新的处理方法, 不需要在搜索的每一步都计算惩罚函数值, 提高了寻优效率.

为方便起见, 将式(9)改写为

$$\min\ J(x)=f+\lambda f_1. \quad (10)$$

其中: $f=-f(x)$, $f_1=\sum_{i=1}^{n_g}\max(0,g_i)$.

在每一次寻优中, 首先计算 f 的值, 然后将其与上一代可行解中的最优值 $f_{\rm best}$ 进行比较. 如果 f 优于 $f_{\rm best}$, 则进一步计算 f_1, 按照式 (10) 对 $J(x)$ 进行更新, 进入下一代寻优. 如果 f 不优于 $f_{\rm best}$ 且 $f_1 = 0$, 则使用对应的 f 对 $J(x)$ 进行更新; 若 $f_1 \neq 0$, 则直接进入下一代寻优. 显然, 本文方法只是对具有较好 f 值的解进行约束计算, 利用惩罚策略使其向可行解空间移动, 而不是对每一个解都进行约束计算.

上述处理方法的具体实现步骤如下.

Step 1: 算法初始化, 随机产生若干个初始解向量, 对采用的相关启发式优化算法进行参数设置.

Step 2: 根据式 (10) 计算目标函数值, 当 $f_1 = 0$ 时, 保存对应可行解中的最优值 $f_{\rm best}$.

Step 3: 采用相关的启发式优化算法对解向量进行寻优.

Step 4: 将得到的解向量按照下面步骤进行计算:

if $f < f_{\rm best}$

$J(x) = f + \lambda f_1$;

else if $f_1 = 0$

$J(x) = f$;

end

Step 5: 对 $J(x)$ 进行更新, 同时保存新的 $f_{\rm best}$.

Step 6: 判断迭代次数是否达到预先设置的最大迭代次数, 如果满足则输出最优解, 停止迭代; 否则重复步骤 Step 3 ∼ Step 5.

将新的约束处理方法引入本文提出的改进差分进化算法中, 并记为 IDE 算法.

IDE 算法对处理具有复杂约束的优化问题效果显著. 对于简单约束问题, 罚函数的计算量小于目标函数时, 仍有一定优势, 但不明显. 在这种情况下也可以采取另外一种方法, 先判定是否满足约束, 然后再决定是否计算目标函数.

3　可靠性冗余分配问题

本文针对文献中常见的 4 个系统可靠性冗余分配问题进行研究, 即复杂桥系统、串联系统、串并联系统和超速保护系统.

3.1　复杂桥系统

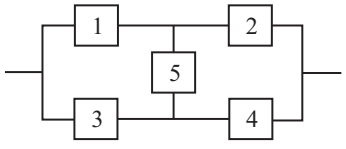

图 1　复杂桥系统原理

图 1 为含有 5 个子系统的复杂桥系统, 其数学模型为

$$\begin{aligned}\max\ & f(r,n) = \\ & R_1R_2 + R_3R_4 + R_1R_4R_5 + R_2R_3R_5 - \\ & R_1R_2R_3R_4 - R_1R_2R_3R_5 - R_1R_2R_4R_5 - \\ & R_1R_3R_4R_5 - R_2R_3R_4R_5 + 2R_1R_2R_3R_4R_5.\end{aligned}$$

$$\text{s.t. } g_1(r,n) = \sum_{i=1}^{m} w_i v_i^2 n_i^2 - V \leqslant 0;$$

$$g_2(r,n) = \sum_{i=1}^{m} \alpha_i \Big(-\frac{1\,000}{\ln(r_i)} \Big)^{\beta_i} [n_i + \exp(n_i/4)] - C \leqslant 0;$$

$$g_3(r,n) = \sum_{i=1}^{m} w_i n_i \exp(n_i/4) - W \leqslant 0;$$

$$0 \leqslant r_i \leqslant 1,\ n_i \in Z^+,\ 1 \leqslant i \leqslant m.$$

其中: $R_i(n_i) = (1 - q_i^n)$ 代表子系统 i 的可靠度, $f(r,n)$ 代表整个系统的可靠度; m 代表子系统的个数, n_i 代表子系统 i 中元件的个数; r_i 代表子系统 i 中每个元件的可靠度, $q_i = 1 - r_i$ 代表子系统 i 中每个元件的失效概率; w_i、v_i、c_i 代表子系统 i 中元件的重量、体积和成本; 参数 α_i 和 β_i 代表系统元件的物理特性; W、V 和 C 代表系统总重量、总体积和总成本的上限. 复杂桥系统所有的参数如表 1 所示.

表 1　复杂桥系统参数

i	$10^5\alpha_i$	β_i	$w_iv_i^2$	w_i	V	C	W
1	2.330	1.5	1	7			
2	1.450	1.5	2	8			
3	0.541	1.5	3	8	110	175	200
4	8.050	1.5	4	6			
5	1.950	1.5	2	9			

3.2　串联系统

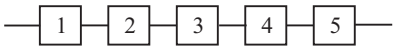

图 2　串联系统原理

图 2 为含有 5 个子系统的串联系统, 系统参数与复杂桥系统相同, 其数学模型为

$$\max\ f(r,n) = \prod_{i=1}^{m} R_i(n_i).$$

$$\text{s.t. } g_1(r,n),\ g_2(r,n),\ g_3(r,n);$$

$$0 \leqslant r_i \leqslant 1,\ n_i \in Z^+,\ 1 \leqslant i \leqslant m.$$

3.3　串并联系统

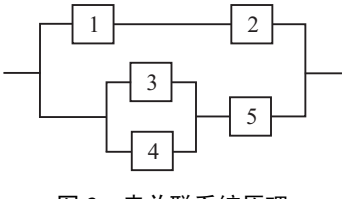

图 3　串并联系统原理

图3为含有5个子系统的串并联系统, 其数学模型为

$$\max\ f(r,n)=1-(1-R_1R_2)(1-(R_3+R_4-R_3R_4)R_5).$$

$$\text{s.t. } g_1(r,n),\ g_2(r,n),\ g_3(r,n);$$

$$0\leqslant r_i\leqslant 1,\ n_i\in Z^+,\ 1\leqslant i\leqslant m.$$

串并联系统与复杂桥系统一样, 也有3个非线性约束条件, 但是系统参数不同, 具体见表2.

表 2 串并联系统所用数据

i	$10^5\alpha_i$	β_i	$w_iv_i^2$	w_i	V	C	W
1	2.500	1.5	2	3.5			
2	1.450	1.5	4	4			
3	0.541	1.5	5	4	180	175	100
4	0.541	1.5	8	3.5			
5	2.100	1.5	4	4.5			

3.4 超速保护系统

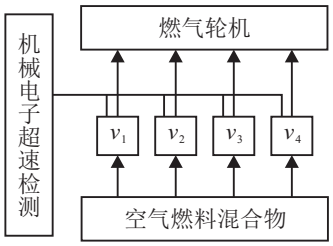

图 4 超速保护系统原理

图4所示为燃气轮机的超速保护系统, 常用于机械系统和电子系统的实时超速检测. 该控制系统可被视为一个4阶串联系统, 其数学模型为

$$\max\ f(r,n)=\prod_{i=1}^{m}[1-(1-r_i)^{n_i}].$$

$$\text{s.t. } g_1(r,n)=\sum_{i=1}^{m}v_in_i^2-V\leqslant 0;$$

$$g_2(r,n)=\sum_{i=1}^{m}C(r_i)[n_i+\exp(n_i/4)]-C\leqslant 0;$$

$$g_3(r,n)=\sum_{i=1}^{m}w_in_i\exp(n_i/4)-W\leqslant 0;$$

$$0.5\leqslant r_i\leqslant 1-10^{-6},\ r_i\in R^+;$$

$$1\leqslant n_i\leqslant 10,\ n_i\in Z^+.$$

其中: n_i、r_i、v_i 和 w_i 分别表示在阶段 i 中冗余元件的个数、可靠度、体积和重量; $C(r_i)=\alpha_i\Big(-\dfrac{T}{\ln(r_i)}\Big)^{\beta^i}$ 表示阶段 i 中可靠性为 r_i 的每个元件的成本; α_i 和 β_i 表示阶段 i 中每个元件物理特性的常数; T 表示每个元件的使用寿命. 超速保护系统的系统参数如表3所示.

表 3 超速保护系统所用数据

i	$10^5\alpha_i$	β_i	v_i	w_i	V	C	W	T
1	1.0	1.5	1	6				
2	2.3	1.5	2	6				
3	0.3	1.5	3	8	250	400	500	1 000 h
4	2.3	1.5	2	7				

4 仿真实验

在硬件环境为Intel(R)Core(TM)2 Quad CPU, Q9400@2.66 GHz, 3.50 GB RAM的计算机上使用Matlab 7.5.0编程, 进行相关仿真实验. IDE算法的参数设置为 $F_{\min}=0.5$, $F_{\max}=1.5$, CR $=0.6$, $N=10$, 最大迭代次数 $k_{\max}=1\,000$, 算法独立运行30次实验.

4.1 寻优精度实验

对上述4个系统的可靠性冗余分配问题进行求解, 结果与文献[3-10]的GA、IA、SAA、NGHS、IPSO、AR-ICA、ICS、CS-GA等算法进行对比实验. 实验中以最优值和MPI值作为算法的评价指标. MPI是Coelho[12]在2009年提出的性能评价指标, 计算方法如下:

$$\text{MPI}(\%)=(f_{\text{IDE}}-f_{\text{other}})/(1-f_{\text{other}}). \tag{11}$$

其中: f_{IDE} 表示由IDE算法得到的最优系统可靠性, f_{other} 表示由其他方法得到的最优系统可靠性. MPI值越大, 表明算法的改进程度越大, 如果MPI值接近于零, 则说明该算法几乎没有改进. 该指标在算法评价中具有重要意义. 比较结果见表4～表7, 其中"–"表示对应文献没有关于此系统的结果.

表 4 复杂桥系统实验结果

算法	$(n_1,n_2,\cdots,n_5)$	r_1	r_2	r_3	r_4	r_5	$f(x)$	g_1	g_2	g_3	MPI
GA	(3,3,3,3,1)	0.814 09	0.864 614	0.890 291	0.701 19	0.734 731	0.999 879 16	18	0.376 347	4.264 8	8.672 6
IA	(3,3,3,3,1)	0.812 485	0.867 661	0.861 221	0.713 852	0.756 699	0.999 889 21	19	0.001 494	4.264 8	0.388 1
SAA	(3,3,3,3,1)	0.807 263	0.868 116	0.872 862	0.712 673	0.751 034	0.999 887 64	18	0.104 323	4.264 8	1.78
NGHS	(3,3,2,4,1)	0.828 831 48	0.858 367 89	0.913 349 96	0.647 794 51	0.701 787 37	0.999 889 62	5	0.000 041	1.560 5	0.018 1
IPSO	(3,3,2,4,1)	0.828 683 61	0.858 025 67	0.913 646 16	0.648 034 07	0.702 275 95	0.999 889 63	5	0.000 004	1.560 5	0.009 1
AR-ICA	(3,3,2,4,1)	0.827 642 57	0.857 478 45	0.914 196 77	0.649 273 79	0.704 092	0.999 889 63	5	0.000 044	1.560 5	0.009 1
ICS	(3,3,2,4,1)	0.828 094 04	0.858 004 49	0.914 162 92	0.647 907 79	0.704 565 98	0.999 889 64	5	0.000 079	1.560 5	0
CS-GA	(3,3,2,4,1)	0.828 085 67	0.857 806 05	0.914 240 06	0.648 143 75	0.704 182 28	0.999 889 64	5	0	1.560 5	0
IDE	(3,3,2,4,1)	0.828 081 59	0.857 707 49	0.914 332 4	0.648 223 61	0.703 714 68	0.999 889 64	5	0	1.560 5	0

表 5　串联系统实验结果

算法	$(n_1, n_2, \cdots, n_5)$	r_1	r_2	r_3	r_4	r_5	$f(x)$	g_1	g_2	g_3	MPI
GA	(3,2,2,3,3)	0.779 427	0.869 482	0.902 674	0.714 038	0.786 896	0.931 578	27	0.121 5	7.518 9	0.152 565
IA	(3,2,2,3,3)	0.779 266	0.872 513	0.902 634	0.710 648	0.788 406	0.931 678	27	0.001 6	7.518 9	0.006 423
SAA	(3,2,2,3,3)	0.782 391	0.866 712	0.901 747	0.717 266	0.783 795	0.931 46	27	0.053 2	7.518 9	0.324 465
NGHS	–	–	–	–	–	–	–	–	–	–	–
IPSO	(3,2,2,3,3)	0.780 373 07	0.871 783 43	0.902 408 9	0.711 473 56	0.787 387 6	0.931 68	27	0.000 1	7.518 9	0.003 495
AR-ICA	(3,2,2,3,3)	0.779 874	0.872 057	0.903 426	0.710 96	0.786 902	0.931 679 39	27	0.000 1	7.518 9	0.004 388
ICS	(3,2,2,3,3)	0.779 416 94	0.871 833 28	0.902 885 08	0.711 393 87	0.787 803 71	0.931 682 387	27	0	7.518 9	0.000 146
CS-GA	(3,2,2,3,3)	0.779 398 87	0.871 837 02	0.902 885 36	0.711 402 52	0.787 799 49	0.931 682 388	27	0	7.518 9	0
IDE	(3,2,2,3,3)	0.779 398 88	0.871 837 02	0.902 885 36	0.711 402 52	0.787 799 48	0.931 682 388	27	0	7.518 9	0

表 6　串并联系统实验结果

算法	$(n_1, n_2, \cdots, n_5)$	r_1	r_2	r_3	r_4	r_5	$f(x)$	g_1	g_2	g_3	MPI
GA	(2,2,2,2,4)	0.785 452	0.842 998	0.885 333	0.917 958	0.870 318	0.999 974 18	40	1.194 4	1.609 3	9.566 23
IA	(2,2,2,2,4)	0.812 485	0.843 155	0.897 385	0.894 516	0.870 59	0.999 976 58	40	0.002 6	1.609 3	0.298 89
SAA	(2,2,2,2,4)	0.812 161	0.853 346	0.897 597	0.900 71	0.866 316	0.999 976 31	40	0.007 3	1.609 3	1.435 21
NGHS	–	–	–	–	–	–	–	–	–	–	–
IPSO	(2,2,2,2,4)	0.819 185 26	0.843 664 21	0.894 729 92	0.895 376 28	0.869 127 24	0.999 976 64	40	0.000 6	1.609 3	0.042 81
AR-ICA	(2,2,2,2,4)	0.822 012 64	0.843 656 4	0.891 290 92	0.898 698 86	0.868 249 39	0.999 976 61	40	0.000 4	1.609 3	0.171 01
ICS	(2,2,2,2,4)	0.819 927 09	0.845 267 66	0.895 491 55	0.895 440 69	0.868 318 78	0.999 976 649	40	0	1.609 3	0.004 28
CS-GA	(2,2,2,2,4)	0.819 660 26	0.844 981 62	0.895 519 31	0.895 492 25	0.868 447 59	0.999 976 65	40	0	1.609 3	0
IDE	(2,2,2,2,4)	0.818 953 45	0.844 386 93	0.895 504 15	0.896 086 93	0.868 651 4	0.999 976 65	40	0	1.609 3	0

表 7　超速保护系统实验结果

算法	$(n_1, n_2, \cdots, n_4)$	r_1	r_2	r_3	r_5	$f(x)$	g_1	g_2	g_3	MPI
GA	–	–	–	–	–	–	–	–	–	–
IA	(5, 5, 5, 5)	0.903 8	0.874 992	0.919 898	0.890 609	0.999 942	50	0.002 152	28.803 7	21.853 45
SAA	(5, 5, 5, 5)	0.895 644	0.885 878	0.912 184	0.887 785	0.999 945	50	0.938	28.803 7	17.590 91
NGHS	(5, 6, 4, 5)	0.901 861 94	0.849 684 07	0.948 426 96	0.888 005 9	0.999 954 67	55	0.001 204	24.801 9	0.011 03
IPSO	(5, 6, 4, 5)	0.901 631 64	0.849 970 2	0.948 218 28	0.888 128 85	0.999 954 67	55	0.000 009	24.081 9	0.011 03
AR-ICA	(5, 6, 4, 5)	0.901 489 88	0.850 035 26	0.948 129 52	0.888 238 33	0.999 954 673	55	0.002 138	24.801 9	0.004 412
ICS	(5, 5, 4, 6)	0.901 614 6	0.888 223 37	0.948 141 03	0.849 920 9	0.999 954 675	55	0	15.363 5	0
CS-GA	(5, 5, 4, 6)	0.901 613 41	0.888 223 38	0.948 142 11	0.849 920 79	0.999 954 675	55	0	15.363 5	0
IDE	(5, 5, 4, 6)	0.901 617 17	0.888 221 1	0.948 142 66	0.849 920 16	0.999 954 675	55	0	15.363 5	0

从表 4～表 7 可以看出: 对于复杂桥系统和超速保护系统, 无论最优值, 还是 MPI 值, 本文结果都与文献 [9] 和文献 [10] 中的结果相近, 优于其他几篇对比文献中算法的结果; 而在串连系统和串并联系统中, 本文算法与文献 [10] 接近, 更优于其他几篇对比文献中的结果.

4.2　寻优速度实验

鉴于优化算法性能的不断提高, 不同算法优化结果之间的差异变得越来越小, 算法性能的进一步提高变得越来越困难, 相比之下, 寻优速度显得更为重要. 为了说明本文提出的约束处理方法的运行效率和通用性, 将其引入到基本 PSO、ABC、HS 和 DE 这 4 类基本常用的算法中, 分别记为 RAP-PSO、RAP-ABC、RAP-HS 和 RAP-DE, 并进行对比实验.

4 种基本算法参数设置如下:

PSO: Colony = 40, $C_1 = 1.496\,2$, $C_2 = 1.496\,2$, $w = 0.729\,8$, Iterations = 1 000;

ABC: Colony = 20, Limit = 100, Iterations = 1 000;

HS: HMS = 5, HMCR = 0.9, PAR = 0.3, BW = 0.01, Iterations = 10 000;

DE: Colony = 10, F = 0.9, CR = 0.6, Iterations = 1 000.

每种算法分别独立运行 30 次, 算法的平均运行时间和最优值如表 8 所示.

从表 8 可以看出: 在引入本文提出的约束处理方法后, 各种算法的运行效率都得到了很大提高,和声搜索算法运行效率的提高最为显著, 运行时间不到原

表 8 平均寻优时间及最优值比较

算法	复杂桥系统		串联系统		串并联系统		超速保护系统	
	时间/ s	最优值	时间/ s	最优值	时间/ s	最优值	时间/ s	最优值
PSO	0.616 6	0.999 804 87	0.556 1	0.624 028 46	0.558 8	0.999 914 93	0.536 6	0.999 954 53
RAP-PSO	0.483 2	0.999 854 45	0.463 8	0.860 133 36	0.419 9	0.9999456	0.369 6	0.999 954 6
ABC	5.830 2	0.999 696 82	5.879 6	0.840 938 11	5.907 3	0.999 829 11	5.779	0.903 183 52
RAP-ABC	4.560 7	0.999 838 36	3.951 4	0.816 458 92	3.959 3	0.999 873 55	3.904 5	0.962 216 08
HS	2.299 1	0.999 813 09	2.056 8	0.874 127 73	2.052 5	0.999 943 85	2.0295	0.989 484 79
RAP-HS	0.752 7	0.999 863 54	0.402	0.892 486 95	0.573	0.999 967 68	0.396 2	0.999 888 69
DE	0.557 5	0.999 847 17	0.497 4	0.928 488 19	0.494	0.999 952 91	0.493 3	0.999 954 43
RAP-DE	0.394 7	0.999 889 6	0.338 3	0.931 682 33	0.334	0.999 970 13	0.328 5	0.999 954 56

来的 1/3; 粒子群算法的提高程度最小,但其运行时间也减少了 20 % 左右. 实验结果表明, 对于约束优化问题, 本文所提出的约束处理方法可以显著提高寻优效率和精度.

5 结 论

本文将现有的 2 种典型变异算子融合, 给出了一种新的改进差分进化算法. 针对复杂约束, 提出了一种新的处理方法, 可以大大减少惩罚函数的计算量, 提高寻优效率. 利用所提出的算法解决 4 类典型系统可靠性冗余分配问题, 结果表明, 该方法可以大幅度提高求解效率. 本文约束处理方法具有很好的寻优效率和通用性, 可以引入到粒子群、和声搜索、人工蜂群等其他智能寻优算法.

参考文献(References)

[1] Kuo W, Prasad R. An annotated overview of system reliability optimization[J]. IEEE Trans on Reliability, 2000, 49(2): 176-187.

[2] Xu Z, Kuo W, Lin H. Optimization limits in improving system reliability[J]. IEEE Trans on Reliability, 1990, 39(1): 51-60.

[3] Hsieh Y C, Chen T C, Bricker D L. Genetic algorithms for reliability design problems[J]. Microelectronics Reliability, 1998, 38(10): 1599-1605.

[4] Chen T C. IAs based approach for reliability redundancy allocation problems[J]. Applied Mathematics and Computation, 2006, 182(2): 1556-1567.

[5] Kim H G, Bae C O, Park D J. Reliability-redundancy optimization using simulated annealing algorithms[J]. J of Quality in Maintenance Engineering, 2006, 12(4): 354-363.

[6] Zou Dexuan, Gao Liqun, Li Steven, et al. A novel global harmony search algorithm for reliability problems[J]. Computers & Industrial Engineering, 2010, 58(2): 307-316.

[7] Wu P, Gao L, Zou D, et al. An improved particle swarm optimization algorithm for reliability problems[J]. ISA Transactions, 2011, 50(1): 71-81.

[8] Afonso L D, Mariani V C, Coelho L dos S. Modified imperialist competitive algorithm based on attraction and repulsion concepts for reliability-redundancy optimization[J]. Expert Systems with Applications, 2013, 40(9): 3794-3802.

[9] Valian E, Tavakoli S, Mohanna S, et al. Improved cuckoo search for reliability optimization problems[J]. Computers & Industrial Engineering, 2013, 64(1): 459-468.

[10] Kanagaraj G, Ponnambalam S G, Jawahar N. A hybrid cuckoo search and genetic algorithm for reliability-redundancy allocation problems[J]. Computers & Industrial Engineering, 2013, 66(4): 1115-1124.

[11] Storn R, Price K. Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces[R]. Berkley: Int Computer Science Institute, 1995.

[12] Coelho L d S. An efficient particle swarm approach for mixed-integer programming in reliability-redundancy optimization applications[J]. Reliability Engineering & System Safety, 2009, 94(4): 830-837.

(责任编辑: 齐 霁)