文章编号: 1001-0920 (2015) 09-1567-08 DOI: 10.13195/j.kzyjc.2014.0895

求解大规模系统可靠性问题的修正和声搜索算法

欧阳海滨, 高立群, 孔祥勇

(东北大学 信息科学与工程学院，沈阳 110004)

摘 要: 针对大规模系统可靠性问题, 提出一种修正和声搜索 (MHS) 算法. 该算法修改了和声搜索 (HS) 算法的搜索机制, 以当前最优解为研究对象, 随机选取不同维数进行即兴创作, 并修正步长 (BW) 的调整方式, 均衡算法的全局搜索和局部搜索. 对经典的大规模系统可靠性问题进行求解, 数值结果表明, 所提出算法优于其他文献中的 6 种和声搜索算法. 与最近提出的求解此类问题的各种算法进行实验对比, 实验结果表明所提出算法在整体上具有良好的优化性能.

关键词: 系统可靠性；和声搜索算法；探索能力；优化

中图分类号: TP391 **文献标志码:** A

Modified harmony search algorithm for solving large scale system reliability problem

OUYANG Hai-bin, *GAO Li-qun*, *KONG Xiang-yong*

(College of Information Science and Engineering，Northeastern University，Shenyang 110004，China．Correspondent: OUYANG Hai-bin，E-mail：oyhb1987@163.com)

Abstract: A modified harmony search(MHS) algorithm is proposed for solving large-scale system reliability problem. This algorithm amends the searching mechanism of HS algorithm, which takes the best-so-far solution as a study subject, randomly selects different dimensions to conduct improvisation, and modifies the adjustment method of parameter bandwidth(BW) to balance global and local searching. The classical large-scale system reliability problem is solved. Numerical results show that the proposed MHS algorithm is better than all the reported 6 kinds of HS algorithms. The MHS algorithm has better optimization performance on the whole compared to some excellent algorithms reported for solving large-scale system reliability problems in the recent year.

Keywords: system reliability；harmony search algorithm；exploration ability；optimization

0 引 言

在电子通信网络系统迅猛发展的今天, 对于系统的可靠性要求越来越苛刻, 进一步提高可靠性在系统设计上变得越来越重要. 仅仅依赖高可靠性的元器件来提高系统可靠度的方式已经难以满足现在系统可靠性的要求, 而系统冗余的合理配置是进一步提高系统可靠性的另一有效途径. 系统冗余配置问题是系统可靠性问题的一个分支, 指在元器件系统可靠性给定的条件下, 如何分配每个子系统的元器件数目, 实现系统的可靠性最大化. 当前, 运用启发式或元启发式自然算法 (如粒子群算法、差分进化算法等) 求解系统可靠性问题是近年来研究工作的主要焦点. 文献 [1] 总结了 1977 年以前系统可靠性问题的研究情况, 阐述了一系列系统可靠性问题的模型和系统结构; 文献 [2-3] 进一步概述了 2007 年以前系统可靠性问题的发展状况, 重点提出了蚁群算法以及近年来提出的元启发式算法在求解系统可靠性优化问题中具有的重要意义. 尽管精确数学规划方法、粒子群优化 (PSO)、差分进化 (DE)、和声搜索 (HS) 和人工蜂群 (ABC) 等算法已对系统可靠性问题进行了优化, 但对于大规模的系统可靠性问题的研究还有待进一步发展. 大规模的系统可靠性问题是一种非线性整数规划问题, 它是由 Prasad 等[4]于 2000 年提出的; Gen 等[5]于 2006 年使用了一种软计算方法求解了该问题; Zou 等[6]于 2010 年利用新颖全局和声搜索 (NGHS) 算法对大规模的系统可靠性问题进行了求解; 2013

收稿日期: 2014-06-08；**修回日期:** 2014-09-22.

基金项目: 国家自然科学基金项目(60674021, 61403174).

作者简介: 欧阳海滨(1987−), 男, 博士生, 从事智能优化与系统建模的研究；高立群(1949−), 男, 教授, 博士生导师, 从事智能优化与图象处理等研究.

年, 布谷鸟(CS)算法[7]及其改进算法(ICS)[8]也被应用于此问题中, 获得了较好的解. 但这些算法所得到的结果均不令人满意, 算法的收敛性和鲁棒性较差, 难以满足工程设计的要求. 因此, 进一步研究大规模的系统可靠性问题的求解方法是很有意义的.

和声搜索算法(HS)是一种简单有效的启发式智能算法, 是由Geem等[9]于2001年提出的. HS算法模拟即兴创作过程, 具有机制简单、随机性强、全局优化效果好等特点. 通过近十几年的研究, HS算法及其改进算法已被成功地应用于许多实际优化问题中. 但是, HS算法存在着搜索盲目、易陷入局部最优和多样性差等缺陷. 近几年, 针对和声搜索算法的不足, 学者提出了一些较为优秀的改进算法, 如改进和声搜索算法(IHS)[10]、全局和声搜索算法(GHS)[11]、新颖全局和声搜索算法(NGHS)[12]、探索和声搜索算法(EHS)[13]、全局动态和声搜索算法(GDHS)[14]等. 其中, NGHS算法已应用于大规模系统可靠性问题[6], 但所得到的结果并不理想, 这也是本文研究的动力所在.

为了更好地应用和声搜索算法求解大规模系统可靠性问题, 获得合理的冗余配置, 得到高而稳定的系统可靠性, 本文提出一种求解大规模系统可靠性问题的修正和声搜索算法. 与和声搜索算法及其改进算法, 以及近年文献中的算法进行了比较研究, 比较结果验证了所提出算法的有效性.

1 大规模系统可靠性问题模型

大规模系统可靠性问题是一类多维不连续, 并且带有多个局部极值点的复杂非线性多约束问题. 文献[15]分析了此问题, 并证明此问题为NP难问题, 具体的数学模型为[4,6]

$$\max\ f(x)=\prod_{j=1}^{n}[1-(1-r_j)^{x_j}].$$

$$\text{s.t.}\begin{cases} g_1(x)=\sum\limits_{j=1}^{n}\alpha_j x_j^2-\mu(\theta)\sum\limits_{j=1}^{n}a_j l_j^2\leqslant 0;\\ g_2(x)=\sum\limits_{j=1}^{n}\beta_j \mathrm{e}^{0.5x_j}-\mu(\theta)\sum\limits_{j=1}^{n}\beta_j \mathrm{e}^{0.5l_j}\leqslant 0;\\ g_3(x)=\sum\limits_{j=1}^{n}\gamma_j x_j-\mu(\theta)\sum\limits_{j=1}^{n}\gamma_j l_j\leqslant 0;\\ g_4(x)=\sum\limits_{j=1}^{n}\delta_j\sqrt{x_j}-\mu(\theta)\sum\limits_{j=1}^{n}\delta_j\sqrt{l_j}\leqslant 0;\\ \mu(\theta)=1+0.01\theta;\\ 1\leqslant x_j\leqslant 10,\ j=1,2,\cdots,n. \end{cases} \tag{1}$$

其中: $f(x)$表示所有n个子系统的总可靠度, $g_1(x)\sim g_4(x)$表示最基本的4个约束分别为重量、体积、成本和冗余度, x_j和r_j分别表示第j个子系统中元器件的个数和可靠度, l_j表示变量x_j的下界, r_j在区间[0.95, 1.0]内均匀随机产生, 容忍度θ表示每个资源l_j能容忍的最小要求, α_j、β_j、γ_j和δ_j分别在区间[6,10]、[1, 5]、[11,20]和[21,40]内随机产生.

由式(1)可知, 大规模系统可靠性问题即是在给定子系统个数和子系统元器件可靠度的情况下, 通过合理配置元器件的个数, 在满足系统重量、体积、成本和冗余度等约束的前提下, 使得系统的总可靠性最高. 考虑到此模型的变量为整数, 为了有效地应用和声搜索算法, 对算法的连续变量进行临近取整.

2 HS算法及其改进算法

本节简单介绍HS算法, 提供一些相关的概念和优化机制, HS算法[9]的基本步骤如下.

Step 1: 算法参数初始化. 给定和声记忆库大小(HMS)、和声记忆库考虑概率(HMCR)、基音调整概率(PAR)、和声微调步长(BW)和最大迭代次数K.

Step 2: 给定范围$[x_i^L, x_i^U]$, 其中x_i^L和x_i^U分别为第i维变量的下限和上限. 根据下式随机初始化, 产生HMS个和声向量存入和声记忆库(HM):

$$x_i=x_i^L+\mathrm{rand}\cdot(x_i^U-x_i^L). \tag{2}$$

Step 3: 基于HMCR、PAR和BW进行即兴创作, 产生新的和声向量, 具体伪代码如下:

For i =1 to N

  If rand < HMCR % 和声记忆库考虑

    $r\in\{1,2,\cdots,\mathrm{HMS}\}$, $x_i^{\mathrm{new}}=x_i^r$;

    If rand < PAR % 基音调整

      $x_i^{\mathrm{new}}=x_i^r\pm\mathrm{rand}\cdot\mathrm{BW}$.

    End If

  Else % 随机变异

    $x_i^{\mathrm{new}}=x_i^L+\mathrm{rand}\cdot(x_i^U-x_i^L)$.

  End If

End For

其中: N为问题的维数, 在进行基音调整时, 当随机数大于0.5时, 取‘+’号; 小于0.5时, 取‘−’号.

Step 4: 更新和声记忆库. 判断新和声x^{new}是否优于当前HM内的最差和声x^{worst}, 若是, 则用x^{new}代替x^{worst}.

Step 5: 判断终止准则. 如果当前迭代次数k大于最大迭代次数K, 则终止运行HS算法, 否则重复执行Step 3和Step 4.

利用IHS算法[15]对HS算法的关键参数PAR和

BW 进行动态调节, 具体表达式为

$$\mathrm{PAR}_k = \mathrm{PAR}_{\min} + \frac{(\mathrm{PAR}_{\max} - \mathrm{PAR}_{\min})k}{K}, \quad (3)$$

$$\mathrm{BW}_k = \mathrm{BW}_{\max} \exp\Big(\frac{k\mathrm{In}(\mathrm{BW}_{\min}/\mathrm{BW}_{\max})}{K}\Big). \quad (4)$$

其中: k表示当前迭代次数, K 表示最大迭代次数, $\mathrm{PAR}_{\max}$ 和 $\mathrm{PAR}_{\min}$ 分别表示最大基音调整概率和最小基音调整概率, $\mathrm{BW}_{\max}$ 和 $\mathrm{BW}_{\min}$ 分别表示最大微调步长和最小微调步长.

利用 EHS[18] 对 HS 算法探索能力进行分析, 设计微调步长, 具体表达式为

$$\mathrm{BW} = \lambda\sqrt{E(x(\mathrm{var}))}, \quad (5)$$

$$\mathrm{E}(x(\mathrm{var})) = \mathrm{E}\Big(\frac{1}{\mathrm{HMS}}\sum_{\mathrm{l}=1}^{\mathrm{HMS}}(x^l - \bar{x})^2\Big). \quad (6)$$

其中: $\lambda = 1.17$ 表示缩放比例参数, $\mathrm{E}(x)$ 表示对 x 求期望. GHS 算法[11]受群智能思想的影响, 在基音调整中引入了最好和声, 加快了算法的收敛. 高效和声搜索 (EGHS) 算法应用位置更新和随机选择操作进行即兴创作, 并应用于系统可靠性优化问题中[16]. NGHS 算法[12]在即兴创作过程中引入了位置更新和变异操作, 取代了 HS 算法的即兴创作过程中的和声记忆考虑, 基音调整和随机产生操作, 直接利用新产生的和声取代和声记忆库中的最差和声. 然而, HS 算法具有很强的随机性, 同时也伴随着很大程度的盲目性, 其优化精度和搜索速度仍有待进一步提高.

3　修正和声搜索算法

3.1　不确定维随机搜索

在和声搜索算法中, 即兴创作过程是通过 3 个准则对每一维变量进行选择和调整的: 和声记忆库考虑、基音调整和随机选择. 然而, 在算法搜索中会出现这样一种情况: 一个解与期望的全局最优解相比, 不是所有维的变量都需要进行选择和调整, 可能只有一部分维的变量需要进行修正. 例如, 目标函数 $f(x) = (x_1-1.2)^2+(x_2-2.5)^2+(x_3-3.3)^2+(x_4-6)^2$, 当前解为 (1.2, 2.49, 4, 5.1), 全局最优解为 (1.2, 2.5, 3.3, 6), 最希望的调整的维数应该为 x_3 和 x_4, 即并不需要所有的维数都进行调整. 基于这样的思想, 本文提出一种不确定维随机搜索方法来提高搜索的快速性, 具体流程如下.

Step 1: 在原有的和声记忆库中, 确定一个基准解作为需要进行即兴创作的对象 x^t.

Step 2: 随机选择需要调整的维数数目 D_n, 确定一个搜索周期 $T_n = K/D$.

Step 3: 在这个周期内, 对 x^t 中随机选择的 D_n 维进行即兴创作. 例如, 总维数为 $D = 10$, 序号为 $(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$, $D_n = 4$, 则不确定随机选择的维数为 (1, 5, 7, 8) 或是别的 4 个维数, 然后只对这 4 个维数进行调整.

在这里需要确定一个基准解, 本文考虑到当前的最优解具有指导搜索方向的作用, 同时也有利于加速算法的收敛, 因此, 基准解直接采用当前的最优解.

3.2　BW 调整策略

在和声搜索算法中, 步长 BW 的大小在一定程度上决定了探索的幅度和范围, 对于算法的全局搜索和局部搜索有重要影响.

当前调整 BW 的方式主要有以下 3 个方面: 1) 根据求解问题的精度直接赋予步长 BW 一个确定的值, 如 HS 算法[9]. 这种方式既难以有效确定比较准确的值, 也不能满足迭代进程的需求. 2) 根据迭代进程, 基于数学思想设计动态的 BW 调整方式 (线性、指数或者梯度等), 如 IHS 算法[10]. 这种方式虽然在一定程度上使 BW 值适应迭代进程的需求, 但与当前和声记忆库中的解没有联系, 大小取值比较盲目. 3) 根据和声记忆库 HM 中的信息进行自适应调整, 如 SAHS 算法[17]和 EHS 算法[13]. SAHS 利用每一维上最小和最大值来调整 BW, 这样 BW 的大小总是在这个最大和最小范围内进行有效的调节, 促进算法的探索, 但是单纯的最大最小值不能有效描述整个和声记忆库的状态. EHS 算法在分析了HS算法探索能力的基础上, 提出了利用整个和声记忆库中解的标准差信息来微调 BW 值, 然而, 整个和声记忆库中解的标准差信息衡量的是整个解的差异性, 这样调整的幅度极小, 同时也会使种群的差异慢慢减小, 多样性降低. 因此, 本文提出如下基于均值的 BW 调整策略:

$$\mathrm{BW}_i = \bar{x}_i. \quad (7)$$

均值客观地描述当前和声记忆库中第 i 维变量的期望. 如果将均值赋予步长 BW, 一方面调整的幅度更宽, 保证了种群的多样性, 另一方面更切合当前解调整的需要. 在迭代前期, 由于解比较分散, 均值相对较大, 有利于拓宽算法的全局搜索. 随着迭代的进行, 较差的解慢慢被取代. 在迭代后期, 均值也相应地越来越靠近最优值, 这有利于算法的局部精细搜索. 此外, 就数量级而言, 与标准差的数量级相比, 均值的数量级与变量的数量级更切合, 调整空间相对比较均匀, 不会出现较大的浮动. 基于以上分析, 修正和声搜索算法的具体伪代码显示如下:

1) 算法参数初始化.

对最大迭代次数 K、和声记忆库考虑 HMCR、基音调整概率 PAR 和问题维数 D 初始化.

2) 和声记忆库初始化.

3) 即兴创作产生新和声 $x^{\rm new}$.

① 不确定随机选择初始维数设为 $D_n = D$, 迭代初始值为 k.

While $k < K$, $x^{\rm new} = x^t$, 其中 x^t 为目前最优解.

If $\mathrm{mod}(k, T_n) = 0$ 满足搜索周期, 随机选择维数数目 $D_n = \mathrm{ceil}(D \cdot \mathrm{rand})$, 其中 $\mathrm{ceil}(x)$ 表示对 x 取整.

End If

$x_d = \mathrm{randperm}(D)$, 对原有的维数进行随机排序, 例如

$$D = 10,\ d = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$$

可能变为

$$x_d = (2, 4, 5, 6, 3, 7, 1, 8, 10, 9),$$

这样 $x_d(1) = 2$ 不是原有序号 1, 而是一个不确定的随机顺序.

②即兴创作.

For $tt = 1 : D_n$,

$j = x_d(tt)$;

If rand < HMCR % 和声记忆库考虑

$r \in \{1, 2, \cdots, \mathrm{HMS}\}$, $x_j^{\rm new} = x_j^r$;

If rand < PAR % 基音调整

$\mathrm{BW}_j = \bar{x}_j$,

$x_j^{\rm new} = x_j^r \pm \mathrm{rand} \cdot \mathrm{BW}$.

End If

Else % 随机变异

$x_i^{\rm new} = x_j^L + \mathrm{rand} \cdot (x_j^U - x_j^L)$.

End If

End For

4) 更新和声记忆库.

If $f(x^{\rm new}) < f(x^{\rm worst})$,

$x^{\rm worst} = x^{\rm new}$.

End If

5) 终止准则.

$k = k + 1$,

End While

6) 记录结果.

3.3 约束处理方法

罚函数法是最常见的约束处理方法, 它将 $f(x)$ 转化为无约束函数 $F(x)$, 由目标函数值和加权的约束函数构成. 罚函数法主要通过不可行解与可行解之间的距离来对不可行解实施惩罚, 通过利用罚函数法, 所有解向量都倾向于朝着可行解的方向搜索. 若问题 $f(x)$ 是最大化问题, 则可以将它转化为最小化问题 $\min(-f(x))$, 因此, 本文中的罚函数 $F(x)$ 可表示为

$$\min\ -f(x) + \varphi \sum_{j=1}^{m} \max(0, g_j). \tag{8}$$

其中: φ 为罚函数系数, 一般都被设置为一个较大的正常数, 本文中设定罚函数系数为 10^5.

4 实验结果与分析

4.1 实验准备

本文将 MHS 算法应用于大规模系统可靠性问题中, 以验证 MHS 算法的优化性能. 文献 [6] 对大规模系统可靠性问题进行了求解, 且设定子系统个数分别为 36、38、40、42 和 50, 具体参数详见文献 [6]. 首先, 进行各种和声搜索算法之间的比较; 其次, 分别与文献中的粒子群算法、差分进化算法和布谷鸟算法进行比较分析. 本文中对比的和声搜索算法包括 HS[9]、IHS[10]、GHS[11]、EGHS[16]、NGHS[6]和 EHS[13], 各算法的参数设置如表 1 所示.

表 1 各算法的参数设置

算法	参数
HS	$\mathrm{HMCR} = 0.9,\ \mathrm{HMS} = 5,\ \mathrm{PAR} = 0.33,$ $\mathrm{BW} = 0.5 \times 10^{-4}(x^U - x^L)$
GHS	$\mathrm{HMS} = 5,\ \mathrm{HMCR} = 0.9,$ $\mathrm{PAR}_{\min} = 0.01,\ \mathrm{PAR}_{\max} = 0.99$
EGHS	$\mathrm{HMS} = 5,\ \mathrm{LUP} = 0.9,\ C_{\min} = 0,\ C_{\max} = 1$
MHS	$\mathrm{HMS} = 50,\ \mathrm{HMCR} = 0.99,\ \mathrm{PAR} = 0.25,\ \mathrm{BW} = \bar{x}$
IHS	$\mathrm{HMS} = 5,\ \mathrm{HMCR} = 0.9,\ \mathrm{PAR}_{\min} = 0.01,$ $\mathrm{PAR}_{\max} = 0.99,\ \mathrm{BW}_{\min} = 0.5 \times 10^{-6}(x^U - x^L),$ $\mathrm{BW}_{\max} = 0.5 \times 10^{-2}(x^U - x^L)$
NGHS	$P_m = 0.1,\ \mathrm{HMS} = 5$
EHS	$\mathrm{HMS} = 50,\ \mathrm{HMCR} = 0.99,$ $\mathrm{PAR} = 0.33,\ \mathrm{BW} = 1.17\sqrt{x(\mathrm{var})}$

本文均采用 Matlab 7.14 实验仿真软件, 在操作系统为 Pentium(R) 4、主板 CPU 为 2.93 Ghz、内存为 1 G 的电脑上独立运行. 为了公平比较, 对比算法中的参数均按照原文进行设置, 各个算法采用罚函数法处理约束, 惩罚因子为 10^5.

4.2 实验结果与分析

依据表 1 的参数设置, 最大目标函数评价次数为 50 000, 将 7 个 HS 算法应用于 5 个经典的大规模系统可靠性算例中, 50 次独立运行所得的结果见表 2.

表 2 中: 达优率是指 50 次运行所获得的值与当前文献提供的全局最优值的偏差小于 5×10^{-3} 的次数与总运行次数的百分比 (即偏差小于 5×10^{-3} 的次数/50), “–”表示该算法无法得到可行解.

表 2　5 个大规模系统可靠性问题的测试结果对比

维数	算法	最好值	中间值	最差值	平均值	方差	达优率/%
	HS	0.517 958 111 6	0.501 905 398 8	0.473 628 578 5	0.501 073 909 4	8.873 6 e-03	10
	IHS	0.519 471 134 3	0.503 359 027 3	0.481 161 733 2	0.501 986 946 9	8.460 4 e-03	40
	GHS	0.519 471 134 3	0.504 422 092 1	0.489 356 797 1	0.502 924 890 2	7.567 1 e-03	40
36	EGHS	—	—	—	—	—	0
	EHS	0.519 975 965 4	0.517 456 215 7	0.492 194 895 1	0.513 539 438 8	6.766 6 e-03	70
	NGHS	0.519 975 965 4	0.515 286 278 0	0.464 973 907 4	0.514 168 557 2	5.396 9 e-02	64
	MHS	0.519 975 965 4	0.519 975 965 4	0.519 975 965 4	0.519 975 965 4	3.364 5 e-16	100
	HS	0.509 996 385 6	0.492 374 479 7	0.466 986 026 1	0.491 988 471 3	9.939 8 e-03	16
	IHS	0.506 069 560 7	0.491 803 463 9	0.467 008 015 0	0.492 306 754 1	1.008 6 e-02	5
	GHS	0.509 500 280 2	0.494 668 354 6	0.467 904 547 5	0.490 831 296 5	8.937 4 e-03	12
38	EGHS	—	—	—	—	—	0
	EHS	0.510 988 596 5	0.506 561 369 3	0.488 485 963 0	0.505 325 592 4	4.557 9 e-03	54
	NGHS	0.508 974 737 6	0.504 413 764 7	0.433 279 429 8	0.500 739 233 8	6.220 9 e-02	40
	MHS	0.510 988 596 5	0.510 988 596 5	0.510 988 596 5	0.510 988 596 5	3.364 5 e-16	100
	HS	0.501 826 592 6	0.486 513 526 3	0.458 258 339 2	0.483 728 831 8	1.137 1 e-02	30
	IHS	0.502 323 691 3	0.488 805 801 0	0.450 542 276 2	0.486 401 546 9	9.920 1 e-03	44
	GHS	0.505 992 421 2	0.485 814 005 3	0.463 121 590 7	0.490 333 547 3	8.687 8 e-03	40
40	EGHS	—	—	—	—	—	0
	EHS	0.505 992 421 2	0.501 592 161 6	0.474 140 206 9	0.499 045 309 9	4.843 6 e-03	60
	NGHS	0.502 323 634 7	0.502 323 691 3	0.441 374 694 5	0.500 203 612 6	6.976 8 e-02	68
	MHS	0.505 992 421 2	0.503 292 493 1	0.503 292 493 1	0.503 670 483 0	9.463 5 e-04	100
	HS	0.472 044 573 2	0.456 905 596 5	0.424 181 709 0	0.454 539 593 6	1.051 6 e-02	0
	IHS	0.472 502 424 3	0.452 166 129 3	0.429 639 244 6	0.454 091 516 7	1.134 0 e-02	0
	GHS	0.475 707 382 7	0.453 683 748 7	0.426 873 160 0	0.459 188 526 8	1.041 8 e-02	5
42	EGHS	—	—	—	—	—	0
	EHS	0.479 663 551 5	0.472 275 259 7	0.458 039 061 8	0.472 468 308 6	4.192 6 e-03	46
	NGHS	0.475 142 960 9	0.471 951 686 6	0.403 930 854 0	0.458 959 145 7	8.568 8 e-02	46
	MHS	0.479 663 551 5	0.478 731 260 8	0.475 707 382 7	0.478 251 048 5	1.339 0 e-03	100
	HS	0.393 968 617 1	0.376 325 941 8	0.349 573 656 8	0.374 489 590 1	1.128 8 e-02	0
	IHS	0.397 221 940 2	0.376 842 359 4	0.354 433 531 6	0.375 616 430 6	1.048 5 e-02	0
	GHS	0.402 256 074 2	0.376 298 436 5	0.349 634 710 5	0.376 788 329 4	1.395 3 e-02	10
50	EGHS	—	—	—	—	—	0
	EHS	0.406 571 428 3	0.398 964 331 6	0.367 862 983 2	0.399 351 120 8	6.896 7 e-03	44
	NGHS	0.406 937 101 2	0.405 244 053 4	0.320 502 109 8	0.400 274 256 0	1.342 5 e-01	68
	MHS	0.406 954 745 1	0.405 779 692 0	0.401 265 291 2	0.405 593 453 9	1.103 3 e-03	90

由表 2 结果可知, MHS 算法所获得的最好值、中间值、最差值、平均值和方差都是这 7 个算法中最优的, 表明了本文算法的有效性. 在此值得一提的是: 尽管 EGHS 算法在处理可靠冗余问题中表现出良好的优化性能[16], 但是在处理大规模系统可靠性问题时却无法通过优化得到一个可行解, 优化的效果是最差的; EHS 算法能够搜索到维数为 36、38、40 和 42 的系统可靠性问题的最好解, 但找不到维数为 50 的大规模系统可靠性问题的最优解; NGHS 算法所得的结果虽然优于 HS 算法和 IHS 算法, 但并不理想, 文献 [6] 也验证了这个结论. 在平均值和方差方面, 本文算法优于所有的其他和声搜索算法, 表现出强壮的鲁棒性和良好的收敛精度, 并且在达优率上, 本文算法优于所有其他和声搜索算法, 甚至在大多数情况下都达到了 100%, 显示出较强的搜索稳定性.

MHS 算法所获得的最好解见表 3.

表 3 中: VTV 表示最好解中值为 2 的变量, 除了值为 2 的变量之外, 其他变量的值都为 1; Slack 表示未使用的资源.

此外, 问题维数的增加使算法越容易陷入局部最优的状态, 而本文算法的优势愈加明显. 在求解的过程中, 为了显示各算法的优化进程, 本文以目标函数评价次数为横轴, 目标函数值为纵轴给出了 7 个算法的仿真优化曲线, 如图 1 ～ 5 所示.

表 3　MHS 所获得的最好解

Dim	VTV in optimum	$f(x)$	Slack (g_1)	Slack (g_2)	Slack (g_3)	Slack (g_4)
36	5, 10, 15, 21, 33	0.519 975 965	1	49.125 763 52	109	301.353 247
38	10, 13, 15, 21, 33	0.510 988 596	1	53.638 550 81	115	317.039 538 5
40	4, 10, 11, 21, 22, 33	0.505 992 421	0	51.047 141 67	119	333.240 548 6
42	4, 10, 11, 15, 21, 33	0.479 663 551	2	52.718 250 39	129	354.583 694 4
50	4, 10, 15, 21, 33, 42, 45	0.406 954 745	0	61.955 982 59	154	433.914 646 8

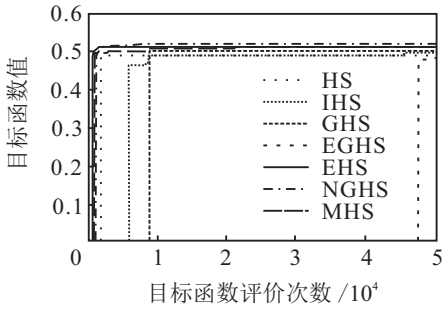

图 1 各算法的优化 ($n = 36$)

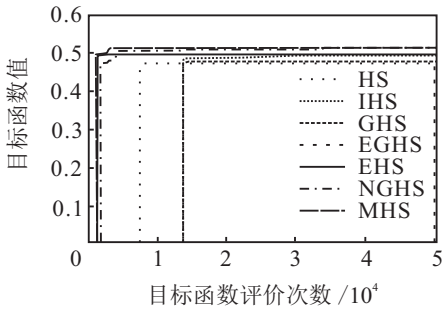

图 2 各算法的优化 ($n = 38$)

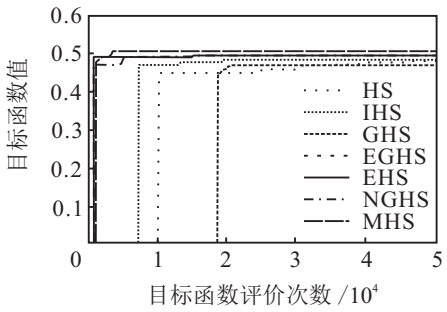

图 3 各算法的优化 ($n = 40$)

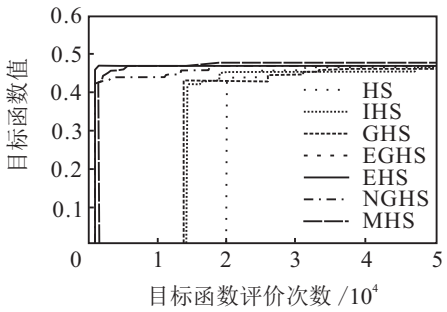

图 4 各算法的优化 ($n = 42$)

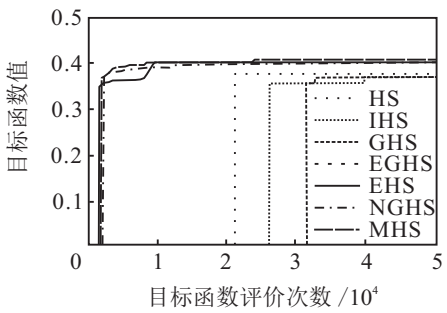

图 5 各算法的优化 ($n = 50$)

由图 1 ~ 图 5 可知, 本文算法具有良好的搜索效率, 收敛速度优于 HS、IHS 和 GHS 算法. 与 EHS 算法和 NGHS 算法相比, 除了子系统为 42 时, 本文算法的收敛速度略慢, 其他情况下的收敛速度都优于 EHS 算法和 NGHS 算法. 值得一提的是, EGHS 算法搜索的结果完全偏离了可行域, 除了子系统为 36 和 38 时, 能显示比较的结果外, 其余的结果都超出了范围, 没有显示出来, 表明 EGHS 算法陷入了维数灾, 搜索的结果是最差的.

除了与一系列 HS 算法进行比较外, 将 MHS 算法的优化结果与文献中的 PSO 及其改进算法[18]、DE 及其改进算法[19]、CS[7]及其改进算法[8]所求的结果进行比较.

在文献 [18] 中, 各 PSO 算法的目标函数评价次数为 13 500; 在文献 [8] 中, CS 算法和 ICS 算法的最大目标函数评价次数为 25 000; 在文献 [19] 中, 各 DE 算法的最大目标函数评价次数为 50 000, 其中子系统个数为 50 时, 最大目标函数评价次数为 50 000. 依据文献给定的最大目标函数评价次数设定本文算法的最大目标函数评价次数, 以确保公平地竞争和比较, 将本文算法所得结果与上述文献中的结果进行比较, 比较结果分别如表 4、表 5 和表 6 所示. 其中, 表 6 中的“–”表示文献中没有结果.

表 4 MHS 算法与 PSO 算法及其改进算法的结果比较

维数	算法	最好值	中间值	最差值	方差
36	PSO	0.507 770 38	0.490 056 77	3.885 3 e+55	5.49 e+54
	IWPSO	0.501 415 76	0.490 485 31	0.459 531 01	1.04 e-002
	BBPSO	0.515 467 5	0.494 182 9	1.081 6 e+54	1.53 e+53
	BBPSO+GJ	0.516 450 95	0.496 808 64	0.457 705 5	1.30 e-002
	IPSO	0.519 975 97	0.508 748 73	0.497 053 68	5.86 e-003
	MHS	0.519 975 965	0.519 471 134	0.518 461 472	4.88 e-04
38	PSO	0.505 087 85	0.477 678 14	1.055 6 e+56	1.49 e+55
	IWPSO	0.499 202 39	0.481 893 43	2.916 7 e+55	4.12 e+54
	BBPSO	0.502 168 27	0.484 240 27	1.562 5 e+54	2.23 e+53
	BBPSO+GJ	0.507 053 18	0.486 110 93	0.447 796 05	1.38 e-002
	IPSO	0.510 988 6	0.505 577 52	0.487 077 8	5.34 e-003
	MHS	0.510 988 597	0.509 996 386	0.506 067 651	1.52 e-03
40	PSO	0.492 212 36	0.47139574	0.449 034 79	1.17 e-002
	IWPSO	0.498 925 84	0.471 605 13	6.081 6 e+55	1.07 e+55
	BBPSO	0.500 376 7	0.473 520 32	0.439 660 35	1.35 e-002
	BBPSO+GJ	0.497 484 35	0.479 796 24	2.890 0 e+52	4.52 e+51
	IPSO	0.505 992 42	0.497 724 92	0.480 167 79	5.43 e-003
	MHS	0.505 992 421	0.502 317 12	0.498 932 885	1.73 e-03
42	PSO	0.458 517 51	0.443 203 81	0.417 565 23	8.76 e-003
	IWPSO	0.462 966 95	0.443 635 77	5.943 5 e+55	9.21 e+54
	BBPSO	0.471 590 68	0.450 403 89	0.412 668 81	1.42 e-002
	BBPSO+GJ	0.467 493 83	0.447 035 15	4.096 0 e+53	6.40 e+52
	IPSO	0.479 663 55	0.470 903 24	0.457 884 22	4.69 e-003
	MHS	0.479 663 552	0.474 580 412	0.469 310 699	2.64 e-03
50	PSO	0.398 561 87	0.362 346 3	1.984 4 e+56	2.86 e+55
	IWPSO	0.382 980 62	0.363 645 35	5.751 6 e+55	1.38 e+55
	BBPSO	0.392 039 21	0.368 843 74	8.880 4 e+54	1.50 e+54
	BBPSO+GJ	0.394 714 4	0.372 582 88	7.569 0 e+53	1.10 e+53
	IPSO	0.406 571 43	0.393 388 27	0.366 824 1	6.58 e-003
	MHS	0.406 954 745	0.406 164 542	0.402 039 935	9.39 e-04

由表 4 可知, 在处理子系统个数分别为 36、38、40 和 42 的系统可靠性问题时, 利用本文算法进行实验所取得的最优值与 IPSO 算法的最优值一样好, 且在子系统个数为 50 时, 所得结果优于 IPSO 算法. 同时, 本文算法所得结果在中间值、最差值和方差方面是所有算法中最好的, 说明本文算法相对于粒子群算法在处理大规模系统可靠性问题时具有很强的竞争优势.

表 5　MHS 算法与 CS 算法及其改进算法的结果比较

维数	算法	最好值	中间值	最差值	方差
36	ICS	0.519 975 97	0.519 085 9	0.510 722 53	2.40 e-03
	CS	0.519 975 97	0.501 979 13	0.473 776 99	8.81 e-03
	MHS	0.519 975 965	0.519 975 965	0.519 471 134	1.21 e-04
38	ICS	0.510 988 6	0.510 737 26	0.507 545 94	7.78 e-04
	CS	0.509 996 39	0.493 859 64	0.464 702 18	9.17 e-03
	MHS	0.510 988 597	0.510 988 597	0.507 541 667	5.66 e-04
40	ICS	0.505 992 42	0.503 285 48	0.502 317 12	3.23 e-04
	CS	0.503 292 49	0.485 745 31	0.455 793 44	9.50 e-03
	MHS	0.505 992 421	0.503 292 493	0.501 829 434	9.86 e-04
42	ICS	0.479 663 55	0.476 707 4	0.472 957 63	1.19 e-03
	CS	0.474 330 72	0.458 620 87	0.432 252 03	8.40 e-03
	MHS	0.479 663 552	0.476 631 086	0.474 783 679	1.13 e-03
50	ICS	0.406 954 75	0.405 072 36	0.401 265 29	2.16 e-03
	CS	0.400 314 71	0.386 970 83	0.358 977 96	8.10 e-03
	MHS	0.406 954 745	0.405 779 692	0.401 265 291	1.10 e-03

表 6　MHS 算法与 DE 算法及其改进算法的结果比较

维数	算法	最好值	最差值	平均值	方差
36	DE	0.519 975 965	0.512 966 693	0.518 049 771	2.24 e-03
	SaDE	0.519 975 965	0.513 966 142	0.519 210 9	1.21 e-03
	CoDE	0.516 451 932	0.491 710 577	0.506 407 449	6.80 e-03
	GAADE	0.519 975 965	0.507 041 712	0.518 256 289	3.25 e-03
	MHS	0.519 975 965	0.519 975 965	0.519 975 965	3.36 e-16
38	DE	0.510 988 596	0.502 654 863	0.507 231 779	2.68 e-03
	SaDE	0.510 988 596	0.504 596 524	0.507 680 27	1.62 e-03
	CoDE	0.499 731 51	—	—	—
	GAADE	0.510 988 596	0.504 123 183	0.509 217 262	2.05 e-03
	MHS	0.510 988 597	0.510 988 597	0.510 988 597	3.36 e-16
40	DE	0.505 992 421	—	—	—
	SaDE	0.505 992 421	0.495 077 754	0.500 153 554	2.62 e-03
	CoDE	0.485 570 002	—	—	—
	GAADE	0.505 992 421	0.498 452 218	0.502 589 162	1.94 e-03
	MHS	0.505 992 421	0.503 292 493	0.503 670 483	9.46 e-04
42	DE	0.474 791 68	—	—	—
	SaDE	0.479 663 551	0.462 512 182	0.471 511 647	4.70 e-03
	CoDE	—	—	—	—
	GAADE	0.479 663 551	0.471 132 4	0.476 800 442	2.79 e-03
	MHS	0.479 663 552	0.475 707 383	0.478 251 049	1.34 e-03
50	DE	—	—	—	—
	SaDE	0.397 221 94	—	—	—
	CoDE	—	—	—	—
	GAADE	0.406 954 745	0.401 481 014	0.404 949 391	1.75 e-03
	MHS	0.406 954 745	0.401 265 291	0.405 593 454	1.10 e-03

由表 5 可知, 本文算法所找到的最好值与 ICS 算法所获得的最好值基本相同, 同时本文算法搜索到的中间值除了子系统为 42 时略微比 ICS 算法的结果差一些以外, 其他所有的结果都是最好的. 在标准差方面, 除了子系统为 40 时, ICS 算法所得值是最好的以外, 本文算法得到的值均优于 CS 算法和 ICS 算法, 这表明本文算法具有良好的优化性能. 此外, 就最差值而言, 本文算法也具有一定的优势.

表6给出了本文算法和各DE算法所求得的结果. 数值结果表明, 除了在维数为50时所求得的最差值略差于GAADE算法以外, 本文算法在最好值、最差值、平均值和方差方面都优于DE、SaDE、CoDE和GAADE算法, 显示出良好的优化潜力, 从整体上说明了本文算法能有效解决大规模系统可靠性问题, 并具有良好的稳定性和收敛性.

综上所述, 对于大规模系统可靠性问题, 与一系列的HS算法及其改进算法、PSO算法及其改进算法、CS算法和ICS算法、DE算法及其改进算法相比, MHS算法能够优化出更有效更稳定的结果, 显示出良好的优化性能.

5 结 论

本文针对工程应用中的大规模系统可靠性问题, 提出一种修正和声搜索算法. 该算法改变了搜索策略, 采用不确定维随机搜索, 设计了BW的调整策略, 有效地均衡了全局搜索和局部搜索. 与现有的HS算法及其改进算法、PSO算法、DE算法和CS算法, 以及近年提出的ICS算法相比, 数值统计结果表明了MHS算法的有效性, 也说明了MHS算法在优化较大规模的复杂系统可靠性问题上具有明显的优势, 对于处理复杂工程问题具有较强的竞争力.

参考文献(References)

[1] Tillman F A, Hwang C L, Kuo W. Optimization techniques for system reliability with redundancy-a review[J]. IEEE Trans on Reliability, 1977, 26(3): 148-155.

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

[3] Kuo W, Wan R. Recent advances in optimal reliability allocation[J]. IEEE Trans on Systems, Man, and Cybernetics — Part A, 2007; 37(2): 143-156.

[4] Prasad V R, Kuo W. Reliability optimization of coherent systems[J]. IEEE Trans on Reliability, 2000, 49(3): 323-330.

[5] Gen M, Yun Y. Soft computing approach for reliability optimization: State-of-the-art survey[J]. Reliability Engineering & System Safety, 2006, 91(9): 1008-1026.

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

[7] Valian E, Valian E. A cuckoo search algorithm by Lévy flights for solving reliability redundancy allocation problems[J]. Engineering Optimization, 2013, 45(11): 1273-1286.

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

[9] Geem Z W, Kim J H, Loganathan G V. A new heuristic optimization algorithm: Harmony search[J]. Simulation, 2001, 76(2): 60-68.

[10] Mahdavi M, Fesanghary M, Damangir E. An improved harmony search algorithm for solving optimization problems[J]. Applied Mathematics and Computation, 2007, 188(2): 1567-1579.

[11] Omran M G H, Mahdavi M. Global-best harmony search[J]. Applied Mathematics and Computation, 2008, 198(2): 643-656.

[12] Zou D, Gao L, Wu J, et al. Novel global harmony search algorithm for unconstrained problems[J]. Neurocomputing, 2010, 73(16): 3308-3318.

[13] Das S, Mukhopadhyay A, Roy A, et al. Exploratory power of the harmony search algorithm: Analysis and improvements for global numerical optimization[J]. Systems, Man, and Cybernetics — Part B, 2011, 41(1): 89-106.

[14] Khalili M, Kharrat R, Salahshoor K, et al. Global dynamic harmony search algorithm: GDHS[J]. Applied Mathematics and Computation, 2014, 228: 195-219.

[15] Chern M S. On the computational complexity of reliability redundancy allocation in a series system[J]. Operations Research Letters, 1992, 11(5): 309-315.

[16] Zou D, Gao L, Li S, et al. An effective global harmony search algorithm for reliability problems[J]. Expert Systems with Applications, 2011, 38(4): 4642-4648.

[17] Wang C M, Huang Y F. Self-adaptive harmony search algorithm for optimization[J]. Expert Systems with Applications, 2010, 37(4): 2826-2837.

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

[19] 孔祥勇, 高立群, 欧阳海滨, 等. 求解大规模可靠性问题的改进差分进化算法[J]. 东北大学学报: 自然科学版, 2014, 35(3): 328-332.
(Kong X Y, Gao L Q, Ouyang H B, et al. Application of improved differential evolution algorithm on large scale reliability problem[J]. J of Northeastern University: Natural Science, 2014, 35(3): 328-332.)

(责任编辑: 闫 妍)