



**JISHOU　UNIVERSITY**

**本科生毕业设计**

|  |  |  |  |
| --- | --- | --- | --- |
| **题 目：** | 基于改进和声搜索算法的模糊Petri网参数优化研究 | | |
| **作 者：** | 李航程 | | |
| **学 号：** | 2017403603 | | |
| **所属学院：** | 信息科学与工程学院 | | |
| **专业年级：** | 计算机科学与技术，2017级 | | |
| **指导教师：** | 周恺卿 | **职 称：** | 副教授 |
| **完成时间：** | 2020年 5 月 10 日 | | |

**吉首大学教务处**

目录

[摘 要 I](#_Toc31097)

[Abstract II](#_Toc1729)

[第1章 绪 论 1](#_Toc12767)

[1.1 本文研究的背景和意义 1](#_Toc8019)

[1.2 国内外研究现状 1](#_Toc4992)

[1.3 本文的研究内容和创新点 2](#_Toc9874)

[1.4 本文结构安排 2](#_Toc28415)

[第2章 基本HS算法和其他相关理论 4](#_Toc24445)

[2.1 基本HS算法 4](#_Toc1190)

[2.2 莱维飞行策略 5](#_Toc30063)

[2.3 反向学习策略 5](#_Toc30130)

[第3章 模糊Petri相关理论 7](#_Toc25280)

[3.1 模糊Petri网 7](#_Toc24835)

[3.2 模糊产生式规则 7](#_Toc10537)

[3.3 模糊推理函数 8](#_Toc3673)

[第4章 基于莱维飞行和反向学习改进的全局和声搜索算法 10](#_Toc9440)

[4.1 改进思路 10](#_Toc2658)

[4.2 具体改进措施 10](#_Toc2603)

[4.3 IGHS-LF-OBL算法 12](#_Toc10255)

[4.4 IGHS-LF-OBL仿真实验 13](#_Toc12665)

[第5章 基于IGHS-LF-OBL的模糊Petri网参数优化 19](#_Toc25044)

[5.1 确定FPN结构 19](#_Toc8540)

[5.2 确定目标函数 20](#_Toc25962)

[5.3 和声个体权值归一化操作 20](#_Toc32639)

[5.4 迭代得到FPN参数 20](#_Toc29684)

[5.5 FPN参数优化仿真实验 21](#_Toc11815)

[第6章 总结与展望 24](#_Toc32528)

[6.1 总结 24](#_Toc708)

[6.2 未来展望 24](#_Toc12891)

[参考文献 25](#_Toc28595)

[附 录 27](#_Toc7879)

[IGHS-LF-OBL主要改进函数代码 27](#_Toc14823)

[FPN参数优化对象代码 28](#_Toc5726)

基于改进和声搜索算法的模糊Petri网参数优化研究

摘 要

针对模糊Petri网自学习能力差、自身各项参数难以确定的缺陷，提出了一种改进的和声搜索算法对模糊Petri网进行参数寻优。提出了一个新颖的即兴创作方法和和声产生机制，利用莱维飞行和反向学习方法提高和声搜索算法的种群多样性和搜索区间。将所提出的算法与6种和声搜索算法在12个测试函数下进行不同维度的对比，实验结果表明所提出的改进算法具有更好的全局搜索和自适应能力。将所提出的改进算法对模糊Petri网进行参数优化，并与遗传算法、差分进化算法和粒子群算法在同一个模糊Petri网下对比，试验结果表明，所提出改进算法的模糊Petri网参数优化结果具有更好的精度和泛化能力。

关键词：模糊Petri网；改进的和声搜索算法；参数优化；莱维飞行；反向学习

**Parameter optimization of fuzzy Petri net based on modified harmony search algorithm**

**Abstract**

Aiming at the defects of fuzzy Petri net, such as poor self-learning ability and difficult to determine its own parameters, an modified harmony search algorithm is proposed to optimize the parameters of fuzzy Petri net. A novel improvisation method and harmony generation mechanism are proposed, levy flight strategy and opposition-based learning strategy are used to improve the population diversity and search range of harmony search algorithm. The modified harmony search algorithm is compared with 6 harmony search algorithms in different dimensions under 12 test functions. The experimental results show that the modified algorithm has better global search and adaptive ability. The modified harmony search algorithm is used to optimize the parameters of fuzzy Petri net, and compared with genetic algorithm, differential evolution algorithm and particle swarm optimization in the same fuzzy Petri net, the experimental results show that the modified algorithm has better precision and generalization ability.

**Key words:** Fuzzy Petri net；Modified harmony search algorithm；Parameter optimization；Levy flight；Opposition-based learning

第1章 绪 论

1.1 本文研究的背景和意义

Petri网是1962年由德国科学家Petri C A提出的一种网状模型，能够很好地表示系统中并行、同步和冲突等关系[1]，但对于专家系统和产生式规则中一些不确定、模糊和难预测的知识却难以表达。模糊Petri网(Fuzzy Petri Net, FPN)作为Petri网的一个重要分支[2]，主要由库所、变迁、确信度、阀值、权值五部分组成，它将Petri网中非0即1的输入输出状态模糊化到0到1之间，是一种基于模糊产生式规则知识库系统的良好建模工具，具有较好的知识表示和推理能力，目前已经得到广泛应用（如网络协议[3]、工业建模[4]、知识推理[5]、模型评估[6]等）。但是FPN的参数难以精确获得[7]，例如权值、阈值、可信度的参数确定很大程度的取决于专家的经验，这阻碍了FPN的知识推理和泛化能力，从而加大了实际应用难度。

和声搜索算法(harmony search, HS)是一种新型的元启发式随机搜索智能优化算法，模拟音乐家通过即兴创作达到美妙和声的过程[8]。该算法借助和声记忆库存储容量大小(harmony memory size, HMS)、和声记忆库存储考虑概率(harmony memory considering rate, HMCR)、基音微调概率(pitch adjusting rate, PAR)和带宽(bandwidth, BW)等4个参数，通过实施和声记忆库考虑、基音调整、更新和声记忆库等3个关键操作来反复调整记忆库中的解变量，使函数值随着迭代次数的增加不断收敛，从而来完成优化。相较于其他经典的元启式算法(如遗传算法[9]、蚁群算法[10]、粒子群算法[11]等)，HS无需对目标函数和约束函数进行严格的数学处理，只要在算法开始事先设置好参数，可针对各类复杂问题并行求解和分布式处理，目前已在许多实际优化问题求解过程得到了广泛的应用（如多目标优化[12]、知识规则抽取[13]、特征选择[14]、自动驾驶[15]等）。但基本HS在解决复杂优化问题时仍存在收敛速度慢和搜索精度低等问题。

1.2 国内外研究现状

针对基本HS算法自身固有的收敛速度慢和搜索精度低等问题，近年来各国学者相继提出了多种改进思路。文献[16]将遗传算法、粒子群算法与传统HS算法相结合，提出一种基于带宽自适应调整策略的改进HS算法NGHS，试验表明，与其他HS算法相比，NGHS的本地搜索能力、搜索精度和收敛速度优势明显；文献[17]利用种群间随机差分表示带宽，提出了一个新的即兴创作模式，并取得了很好的效果；文献[18]在差分的基础上，提出了一种搜索区域自适应放缩的HS算法，提高了了算法的搜索精度；文献[19]在NGHS的基础上，引入反向学习算法作为新的和声产生机制，大大扩宽了算法的搜索区间；文献[20]提出了一种新的和声位置更新机制，大大避免了算法陷入局部最优的可能性。从以上可以看出，当今HS算法的改进多集中在即兴创作和和声产生机制上，也都取得了不错的成果，但HS算法的全局寻优能力和自适应能力仍需要提高。

针对模糊Petri网自学习能力差、自身参数难以确定等缺陷，国内外学者都提出了相应的解决方法，大体上分为两类，一类是从FPN本身结构上进行改进，文献[21-22]提出了一种带自适应能力的FPN框架，一定程度上提高了FPN的自学习能力，文献[23]提出了一种广义动态的FPN框架，提高了FPN的推理精度和泛化能力，但以上方法总体来说都需要严格的限制条件和特定的适用情景；另一类是利用机器学习和群体智能等算法从外部对FPN参数进行优化，文献[7,24]提出将BP网络与FPN相结合，采用分层思想，对FPN的权值、阈值、可信度参数进行整体寻优，使得训练后的网络具有较好的泛化性，文献[25-27]将各种软计算方法运用在FPN的自适应能力研究上，分别利用克隆选择算法、差分进化算法、改进遗传算法等进化算法对FPN的参数进行寻优求解，使得算法实现不依赖数据经验，无需严格的初始输入，进一步提高了FPN模型推理的准确度，但以上方法或多或少存在一些问题，如算法迭代完成后FPN参数精度不够，在算法迭代过程中易陷入局部最优等，从而影响FPN参数优化的结果。

1.3 本文的研究内容和创新点

在HS算法运行过程中，即兴创作阶段产生种群个体的多样性会逐渐减少，搜索寻优范围也会逐渐单一，针对这些缺陷，受上述文献启发，提出了一种改进的HS算法，通过提出了一个新颖的即兴创作方法和和声产生机制来达到算法改进的目的。之后，将所提出的算法与6种和声搜索算法在12个测试函数下进行不同维度的对比，实验结果表明所提出的改进算法具有更好的全局搜索和自适应能力。在模糊Petri网实际应用中，自身参数难以确定，依赖于专家经验给出，针对这一缺陷，利用改进HS算法优良的寻优能力和自适应能力对FPN参数进行整体寻优，使得FPN参数的获取不依赖人的经验且对初始输入无要求，最后，将改进HS算法与遗传算法、差分进化和粒子群算法在同一个FPN下进行参数优化对比，实验结果表明，所提出改进HS算法的FPN优化结果具有更好的精度和泛化性。

本文主要创新点体现在：

1. 将莱维飞行策略引入HS算法的即兴创作阶段，增强产生种群个体的多样性，减小陷入局部最优的概率，并与其他改进算法结合，提出了一种新颖的即兴创作方法。
2. 利用反向学习策略扩宽搜索区域，从而加快收敛速度，提出了一种新的和声个体产生机制。
3. 首次将改进的HS算法对模糊Petri网的参数进行整体寻优，使得FPN参数的获取不依赖人的经验且对初始输入无要求，并与其他传统群体智能算法优化结果对比。

1.4 本文结构安排

第一章主要阐述了模糊Petri网和和声搜索算法的研究背景和意义、国内外研究现状，简要介绍了了本文研究内容和创新点。

第二章主要介绍了基本和声搜索算法、莱维飞行和反向学习策略基本理论。

第三章主要介绍了模糊Petri网的基本概念和理论。

第四章主要介绍了基于莱维飞行和反向学习改进的全局和声搜索算法的改进思路和算法步骤，并记录了与其他6种和声搜索算法的试验对比结果。

第五章主要介绍了改进的和声搜索算法对模糊Petri网的参数优化步骤，并记录了与遗传算法、进化差分算法和粒子群算法三种群体智能算法的试验对比结果。

第六章主要介绍了本文得出的结论与工作总结以及对下一步研究工作的一些展望。

第2章 基本HS算法和其他相关理论

2.1 基本HS算法

基本的和声搜索算法分为五个步骤[8,28]：（1）定义问题和参数数值；（2）初始化和声记忆库(harmony memory, HM)；（3）即兴创作，生成一个新的和声；（4）更新和声记忆库；（5）检查算法是否终止。

1. **定义问题和参数数值**

定义目标函数

 (2.1)

式中：为目标函数的解。相关参数如表1所示。

表1 参数及其对应含义和作用

|  |  |  |
| --- | --- | --- |
| 参数名称 | 含义 | 作用 |
| ***H*** | 和声记忆库（和声种群） | 用于存储和声向量 |
| *S*HM | 和声记忆库（和声种群）的大小 | 从解空间中随机生成和声记忆库，*S*HM控制和声记忆库的大小 |
| *P*HMCR | 和声记忆库存储考虑概率 | 从现有和声种群（和声库）中拿出一个和声的概率, *P*HMCR控制这个概率的大小 |
| *P*PAR | 基音微调概率 | 对拿出来的和声变量进行微调的概率，*P*PAR控制这个概率的大小 |
| *W*BW | 带宽 | 控制微调的幅度 |
| *T*max | 调整（迭代）的次数 | 控制总迭代次数 |
| *U*、*L* | 搜索范围上、下界 | 控制解空间的范围 |

**（2）初始化和声记忆库**

从解空间里随机生成*S*HM个和声放入和声记忆库***H***，并记录对应的*f(****X****)*,和声库的形式为：

 (2.2)

式中为算法的目标函数值。

**（3）即兴创作（生成一个新的和声）**

在[0,1]之间产生一个随机数*r*1，与*P*HMCR进行比较，若*r*1<*P*HMCR，则从和声记忆库中随机拿出一个和声变量；否则，从解空间随机生成一个和声变量。若该和声变量是从和声库中得到，则进行微调操作。具体而言，在[0,1]之间产生一个随机数*r*2，若*r*2*<P*PAR，根据微调带宽*W*BW对得到的和声变量进行调整，得到一个新的和声变量；否则，不做任何调整；最后得到一个新的和声。

**（4）更新和声记忆库**

对进行评估得。若至少优于***H***中的一个函数值，即，则将代替***H***中函数值最差的和声，否则，不做修改。

**（5）检查算法是否终止**

重复步骤(3)和(4)，直到迭代次数达到*T*max为止。

2.2 莱维飞行策略

莱维飞行(Lévy flight, LF)是指随机行走的过程中有相对较高的概率出现大跨步，是因为分布的路径酷似飞行一样，顾命名为莱维飞行[29]。这种长短距离兼备的策略主要被用于布谷鸟算法中下一次位置的确定[30]，用于扩大局部搜索的位置点，从而增大种群多样性。莱维飞行的实现主要依赖于莱维随机数的实现，本文Lévy随机数的实现如下:

服从概率分布，计算如下：

 (2.3)

即  (2.4)

其中*u*和*v*均服从标准正态分布，,的计算如下：

 (2.5)

把以上公式用python模拟，生成的二维平面萊维飞行轨迹如图1所示：

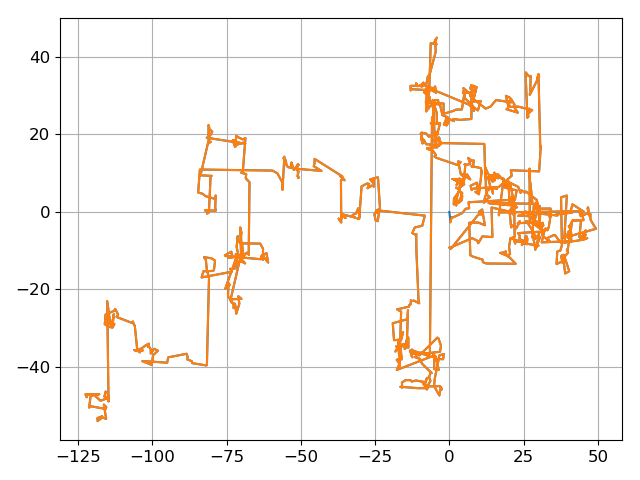


图1 二维平面莱维飞行轨迹

2.3 反向学习策略

反向学习(Opposition-based learning,OBL)通过一个解寻找到它在一个区间内反向的解，然后将这两个解比较，选出更优的解[31]，通过扩大搜索的区间，发挥自身良好的全局搜索能力。以下为反向学习技术的定义：

**定义1 全局反向数：**

设，且，则它对应的全局反向数被定义为；

**定义2 全局反向点：**

设为N维空间里的一个点，且，，则全局反向点可被定义为。反向学习策略可以有效扩宽解的搜索范围，如图2所示。

****

图2 反向数图解

第3章 模糊Petri相关理论

3.1 模糊Petri网

模糊Petri网是由普通Petri网发展而来的的[4]，将原来的非0即1的输入输出模糊化到[0,1]之间，FPN主要由库所、变迁、权值、确信度、阈值五个部分组成。

**定义3** 一个FPN被定义为一个八元组

，其中：

1. 表示库所位置节点的有限集合。
2. 表示位置变迁节点的有限集合。
3. *I*表示，是输入函数，反映库所到变迁的映射；*O*表示，是输出函

数，反映变迁到库所的映射。

1. *M*是一个映射，将库所*P*映射到[0,1]之间，对每一个库所节点,

都有一个标记值。

1. ，为变迁的阈值(*j*=1,2,...,*m*)，当输入大于阈值进行点火。
2. 是每条规则的权值集合，表示规则中的前提条件对结论的重要

程度

1. ，为变迁的确信程度，，且*j*=1,2,...,*m*。

3.2 模糊产生式规则

模糊产生式规则用来描述多个命题间的模糊关系，如果*R*是一个模糊产生式规则系统，即,则(*i*=1,2,...,*m*)可分为与规则和或规则两类：

**定义4** (1) 与规则:

: IF *d*1 and *d*2 and ... and *d*n THEN *d* , *u*,,。

其中，*dj*是前提命题，*d*是结果命题，*u*和分别是规则的确信度和阈值，是权值,且，满足( *j*=1,2,...,*n*)，图形化表示如图3所示：



图3 与规则FPN模型示例

1. 或规则：

: IF *d*1 or *d*2 or ... or *d*n THEN *d* ,。

其中的意义和与规则相同，但各库所到变迁的权值都为1，图形化表示

如图4所示：



图4 或规则FPN模型示例

模糊产生式规则的激活是通过变迁的点火来实现的，对于一个变迁*t*，若它的所有输入库所节点的标记值与对应弧上权值之积的和大于变迁的阈值，则称变迁t是使能的，定义如下：

**定义5 ，**若，， *j*=1,2,...,*n，*则称变迁t是使能的。

使能后变迁被点燃，进行模糊推理，变迁*t*所有输入库所标记值不变，向输出库所传送新的标记值。在与规则和或规则中，传送到输出库所的标记值结果有所不同，以下定义为或规则变迁传送到输出库所标记值的计算。

**定义6** 若库所*p*是输出库所，且有多个变迁(*i*=1,2,...,*n*)输入，则库所*p*的标记

是传送而来的n个值中的最大值：

，。

3.3 模糊推理函数

为了使模糊推理比较直观的计算，需要将变迁使能的过程转化为一个便于求导的连续函数，现在应用文献[21]中的Sigmoid函数将模糊推理函数分为变迁点燃连续函数和最大运算连续函数两部分。

3.3.1 变迁点燃连续函数

对于与规则，在定义5中，对于一个变迁*t*，当时，变迁*t*使能，设变迁点燃连续函数*g*(*x*)为一个Sigmoid函数，*b*为一个足够大的常量，*g*(*x*)表达式为。设，，则*g*(*x*)建立了变迁使能的判断：当*b*足够大时，有

1. 若,则,表示变迁*t*是使能的。
2. 若，则，表示变迁*t*是没有使能的。

因此，连续函数可用来判断*t*是否变迁和对库所输出标记值的计算。

3.3.2 最大运算连续函数

根据前面的*g*(*x*)函数，设有3个变迁已经使能，为使能输出值，当b足够大时，以下推导成立：





以此类推，当有多个与规则和或规则时，计算方式同理。

第4章 基于莱维飞行和反向学习改进的全局和声搜索算法

4.1 改进思路

HS算法能够不断收敛并且在区间内搜索更优解的原因是因为HS算法采用了贪婪准则和即兴创作阶段，然而这两种策略都有不同程度的缺陷，如贪婪准则通过每一次将新产生的个体与种群中最差的个体作比较，若比最差的个体好，则替换，这样虽然能够保证总体种群的进化，但很有可能将种群都置换成局部最优，导致算法陷入局部最优而出不来；即兴创作阶段通过两个个随机概率随机对种群间的个体进行调整，这样虽然保证了新个体的出现，但是HS算法中即兴创作策略方法却过于单一，且新个体的产生太过依赖于当前种群，一旦算法陷入坡度较大的局部最优，即使使用即兴创作策略，也很难跳出。针对这两个机制的缺陷，NGHS[16]提出了将调整幅度用最好个体值和最差个体值之差来代替，使得调整幅度能随具体情况自适应调整，并舍弃掉贪婪准则，直接替换最差的个体；IDHS提出将种群间差分进化方法应用到即兴创作阶段，利用种群差分来提高即兴创作阶段的自适应性和种群多样性，以增强算法全局性能。本文受NGHS和IDHS启发，并在其基础上，主要从种群多样性和搜索区间范围入手，提出了一种基于莱维飞行和反向学习改进的全局和声搜索算法(An improved global harmony search algorithm utilizing levy flight and opposition-based learning, IGHS-LF-OBL)。相较于其他主流改进HS算法，具体改进点体现如下：

1. 在即兴创作阶段中，当产生随机数大于和声记忆库考虑概率时，利用莱维飞行策略高度随机行走的特性产生新的和声个体，丰富种群和个体选择的多样性，降低陷入局部最优的风险。
2. 当即兴创作阶段结束后，提出一种新的和声产生机制，利用反向学习策略动态扩宽搜索区间的范围，产生出种群中最优和声个体和最差和声个体的反向和声个体，再进入和声记忆库更新阶段。
3. 对于新的和声产生机制采用新的和声更新机制，以降低过度使用贪婪准则带来的的负面影响，将产生出的最差反向和声个体与种群中最优的个体作比较，若比最优的个体好，则替换，将产生出的最优反向和声个体与种群中最差的个体作比较，若比最差的个体好，则替换。

4.2 具体改进措施

4.2.1 新增参数设置

(1) 新增加参数*P*HMCRmin, *P*HMCRmax, *P*PARmin , *P*PARmax, Levy(*i*)和,。其中，*P*HMCRmin为最小和声记忆库取值概率，*P*HMCRmax为最大和声记忆库取值概率，*P*PARmin为最小音调微调概率，*P*PARmax为最大音调微调概率，Levy(*i*)为产生的莱维随机数，和为莱维随机数调节因子。

(2) 带宽*W*BW不再在参数设置阶段直接赋值，而是由对应维度最差和声和最优和声之差确定。

(3) 新增参数和。其中是由种群中每个维度的最大值组成的和声个体，是由种群中每个维度的最小值组成的和声个体。

4.2.2 新颖的即兴创作计划

1. 当*r*1*<P*HMCR时，对进行微调，带宽由对应维度最差和声和最优和声之差确定即：

 (4.1)

其中，带宽，为当前种群***H***中效果最差的和声个体的第*i*个值，为当前种群***H***中效果最好的和声个体的第*i*个值。此方法的原理如图5所示：



图5 带宽扩展搜索范围原理

由图5可以看出，使用此种带宽表示方法可以有效开发种群中最优解周围的搜索空间，而莱维飞行的加入，是为了让搜索的信息更多样，进一步开发搜索空间的潜力。

1. 若*r*2<*P*PAR，则保留种群中效果最好的和声个体的一个随机和声变量值，即：

 (4.2)

其中*k*为维度范围内的一个随机整数，每次保存效果最好的和声有利于加快算法的收敛速度，同时随机数*k*可以增加维间扰动，若所求问题有两个或两个以上相同解时，当有一个正确解被搜索到时，其他几个相同解也会被随机数*k*快速搜索到。

1. 若*r*1>*P*HMCR，将随机挑选出一个和声个体值，并对其做莱维随机行走，即：

 (4.3)

其中为1~*S*HM之间的随机整数，利用莱维飞行增加种群多样性，降低局部最优风险，原理同图5。

4.2.3 新的和声个体产生机制

当即兴创作阶段结束后，新增一个新的和声个体产生机制，通过和作为反向学习区间，通过种群中最优和最差的和声个体产生两个反向和声个体，即：

 (4.4)

 (4.5)

其中，rand()为在0~1之间的一个随机小数，和是产生的两个反向和声个体。通过计算种群中最优和最差和声个体的反向数，再次扩宽了搜索区域，加快算法收敛速度，方法原理如图6：

****

图6 反向和声个体产生原理图

从图6可以看出，使用此方法可以充分开发种群中最优个体和最差个体到对应反向个体之间的搜索空间，进一步扩宽搜索范围。

4.2.4 修改的和声记忆库更新机制

本文多次使用贪婪准则，为削减贪婪准则带来的负面影响，修改了和声记忆库的更新机制，具体为：对于和声个体，在即兴创作阶段完成后，与种群中最差的个体作比较，若比最差的个体好，则替换；对于和，先将与与种群中最优的个体作比较，若比最优的个体好，则替换，再将与种群中最差的个体作比较，若比最差的个体好，则替换。

4.2.5 参数动态更新

每次迭代完成前，将和声记忆库取值概率和音调微调概率随迭代次数动态改变，设为当前迭代次数，具体操作如下：

 (4.6)

 (4.7)

4.3 IGHS-LF-OBL算法

IGHS-LF-OBL算法主要由五个部分组成：1.初始化和声记忆库 2. 新的即兴创作策略 3.新的和声产生机制 4.和声种群更新 5.参数动态调节。伪代码如下：

|  |
| --- |
| **IGHS-LF-OBL 算法伪代码** |
| *T*it = 0 |
| /\*初始化和声记忆库***H***\*/ |
| for *i*=1 to *S*HM do: |
| for *j*=1 to *n* do: |
|  |
| /\*即兴创作创作一个新的和声个体\*/ |
| While *T*it < *T*max ： |
| for *i*=1 to *n* do: |
| if (*r*1<*P*HMCR) then |
|  |
| if (*r*2<*P*PAR) then: |
|  |
| else (i.e. when *r*1>*P*HMCR) |
|  |
| If ***X***new(*i*) < *L* or ***X***new(*i*) > *U*: |
| ***X***new(*i*)=*L*+(*U*-*L*)rand(); |
| end for |
| if *f*(***X***new)*<f*(***X***worst)then ***X***worst = ***X***new; |
| /\*一个反向学习和声个体产生机制\*/ |
| for *i*=1 to *n* do: |
| ; |
| ; |
| for *i*=1 to *n* do: |
| ; |
| ; |
| if *f*()*<f*(***X***best)then ***X***best =; |
| if *f*()*<f*(***X***worst)then ***X***worst=; |
| *T*it = *T*it + 1; |
| ; |
| ; |
| end while |
| 注：*n*为维度大小，getDminvalue(*i*)和getDmaxvalue(*i*)为获取到种群中第*i*维值的最小值和最大值。 |

4.4 IGHS-LF-OBL仿真实验

为了验证IGHS-LF-OBL的有效性，本研究采用文献[16-20]涉及到的12个经典函数作为测试函数，这12个测试函数中有3个单峰值函数，6个多峰值函数和3个固定尺寸多峰值函数。让IGHS-LF-OBL与HS[8]、NGHS[16]、IDHS[17]、ID-HS-LDD[18]、GOGHS[19]、IMGHSA[20]六种HS算法在12个测试函数下优化求解，并各自单独运行30次，就优化结果进行对比分析，关注的指标有迭代完成的目标函数值、30次独立运行的平均函数值和函数值标准差。其中函数*F*1-*F*9分别在10维、30维和100维三种情况下分别迭代3000次后进行对比，函数*F*10-*F*12在固定维度下迭代3000次后进行对比。

4.4.1 测试函数说明

12个测试函数的表达式及特点如表2所示。

表2 12个经典测试函数的表达式及其特点

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 编号 | 函数名 | 表达式 | 函数类型 | 搜索空间 | 最小函数值 |
| *F*1 | Sphere |  | 单峰函数 | [-100,100] | 0 |
| *F*2 | Step |  | 单峰函数 | [-100,100] | 0 |
| *F*3 | Rosenbrock |  | 单峰函数 | [-30,30] | 0 |
| *F*4 | Ackley |  | 多峰函数 | [-32,32] | 0 |
| *F*5 | Griewanks |  | 多峰函数 | [-600,600] | 0 |
| *F*6 | Rastrigin |  | 多峰函数 | [-5.12,5.12] | 0 |
| *F*7 | Levy |  | 多峰函数 | [-10,10] | 0 |
| *F*8 | Bohachevsky |  | 多峰函数 | [-15,15] | 0 |
| *F*9 | Alpine 1 |  | 多峰函数 | [-10,10] | 0 |
| *F*10 | Shubert |  | 固定尺寸多峰函数 | [-10,10] | 0 |
| *F*11 | Matyas |  | 固定尺寸多峰函数 | [-10,10] | 0 |
| *F*12 | Trid 10 |  | 固定尺寸多峰函数 | [-100,100] | -210 |

4.4.2 实验环境及相关参数设置

试验在InTel（R）Core（TM）i5-3470 CPU @ 3.20 GHz、4 GB内存、Win10操作系统的条件下进行，编程语言为Python 3.5。

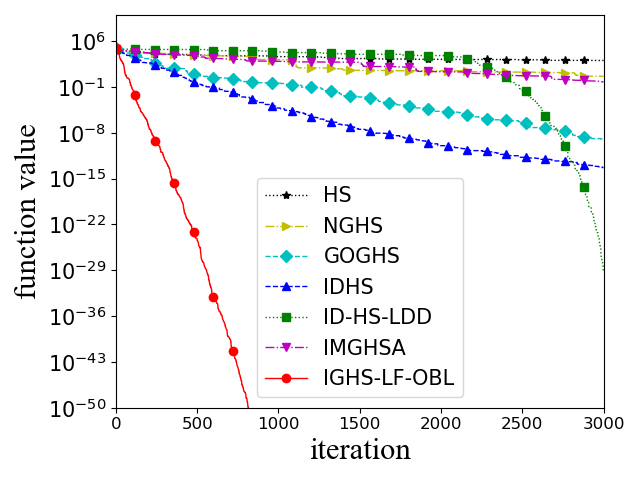
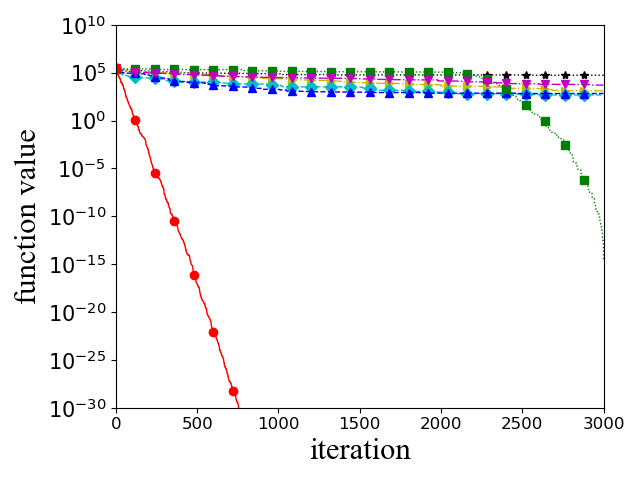
参数选取均为各文献中所记录参数，7种HS算法的参数设置如表3所示。

表3 不同和声搜索算法参数设置

|  |  |
| --- | --- |
| 算法名称 | 参数 |
| HS | *S*HM*=*5*,P*HMCR*=*0.9*,P*PAR*=*0.3*,W*BW*=*0.001 |
| NGHS | *P*m*=*0.005 |
| GOGHS | *P*m*=*0.005 |
| IDHS | *S*HM*=*20*, P*HMCRmin*=*0.8*, P*HMCRmax*=*0.9*,P*PARmin*=*0.1*, P*PARmax*=*0.9*, F*1*=*0.6*,F*2*=*0.6 |
| ID-HS-LDD | *S*HM*=*30*, P*HMCRmin*=*0.3*, P*HMCRmax*=*0.99*,P*PARmin*=*0.9*, P*PARmax*=*0.99 |
| IMGHSA | *S*HM*=*30*,P*HMCR*=*0.9*,P*PAR*=*0.3*,W*BW*=*0.01*,P*m*=*0.005 |
| IGHS-LF-OBL | *S*HM*=*8*, P*HMCRmin*=*0.8*, P*HMCRmax*=*0.9*,P*PARmin*=*0.1*, P*PARmax*=*0.9*,* |

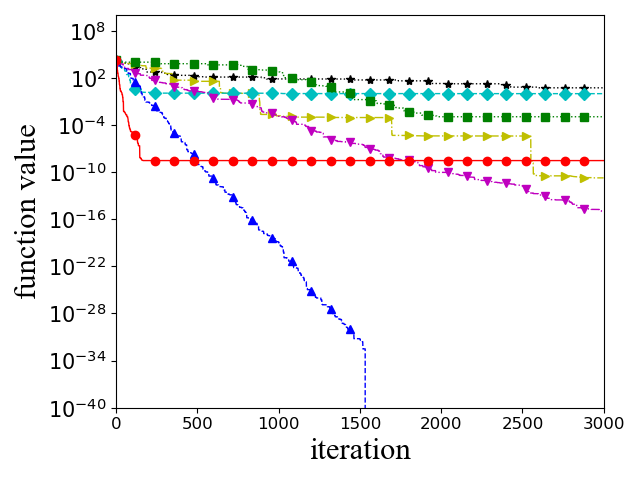
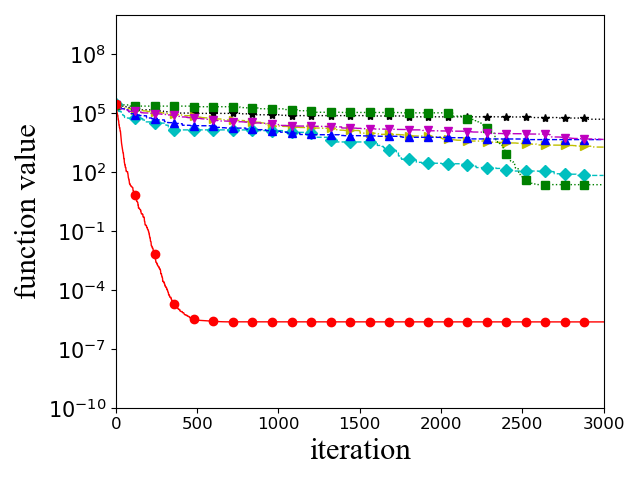
4.4.3 HS算法实验结果及分析

图7~18为12个测试函数在30维和100维情况下的进化曲线，目标函数值越小、收敛速度越快表示则算法性能更优，其中图例在图7~18中通用。

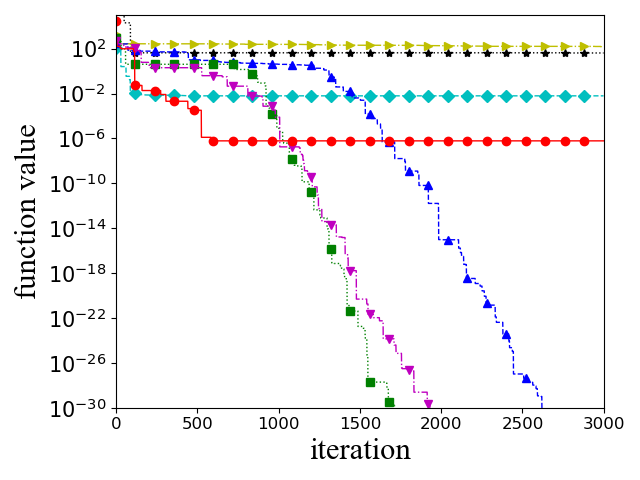
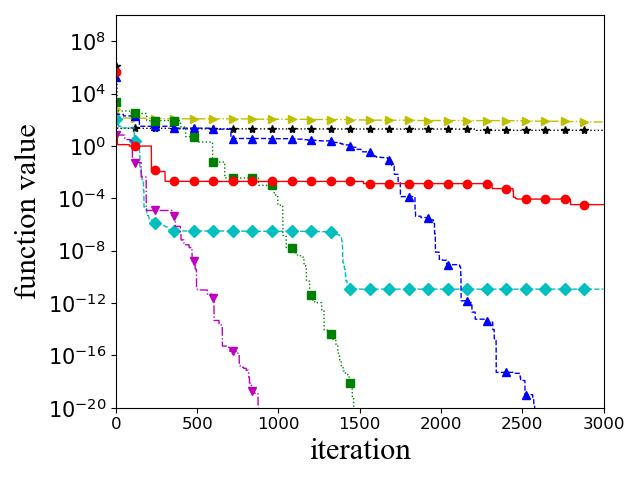
|  |  |
| --- | --- |
| 1. **30-dimension situation** | 1. **100-dimension situation** |

图7 Sphere函数进化曲线

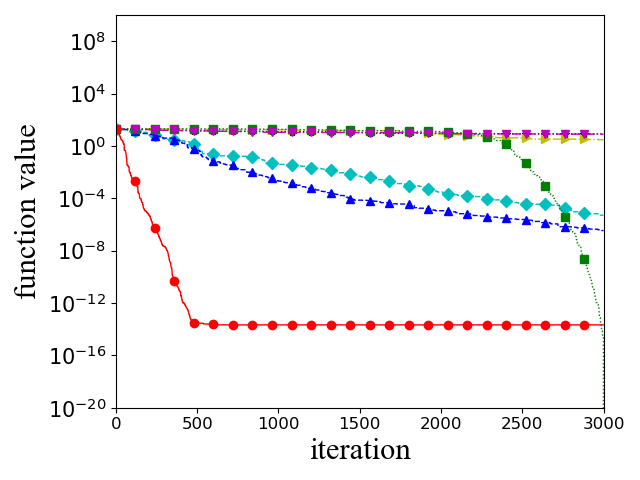
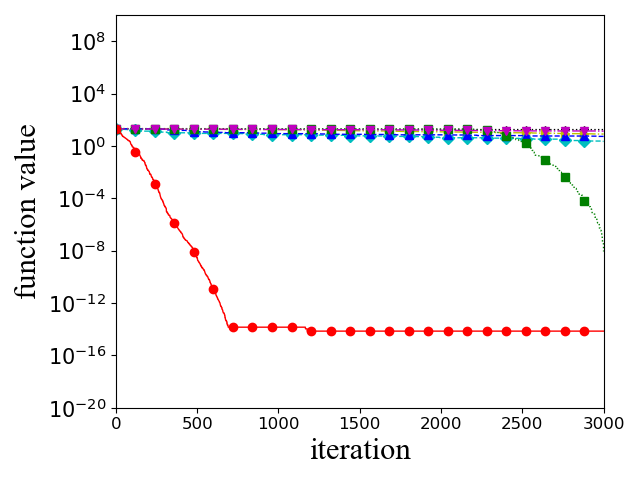
|  |  |
| --- | --- |
| (**a**) **30-dimension situation** | (**b**) **100-dimension situation** |

图8 Step函数进化曲线

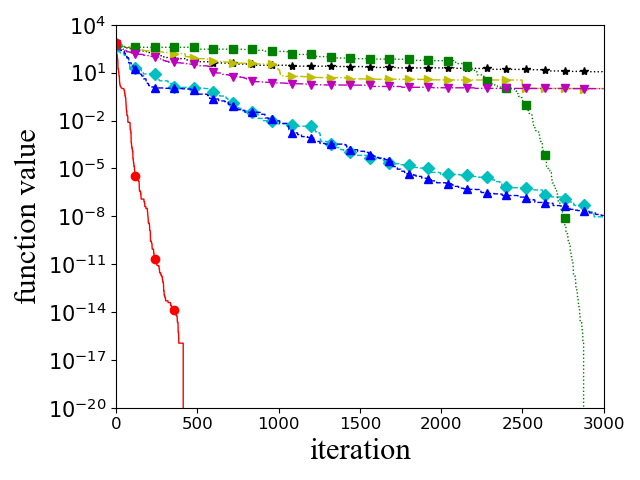
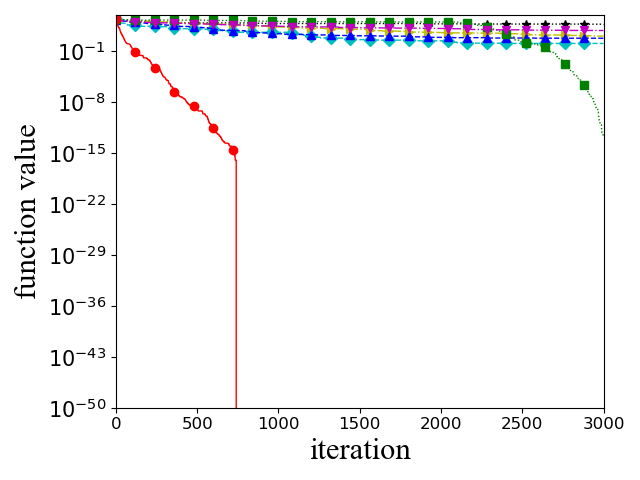
|  |  |
| --- | --- |
| 1. **30-dimension situation** | 1. **100-dimension situation** |

图9 Rosenbrock函数进化曲线

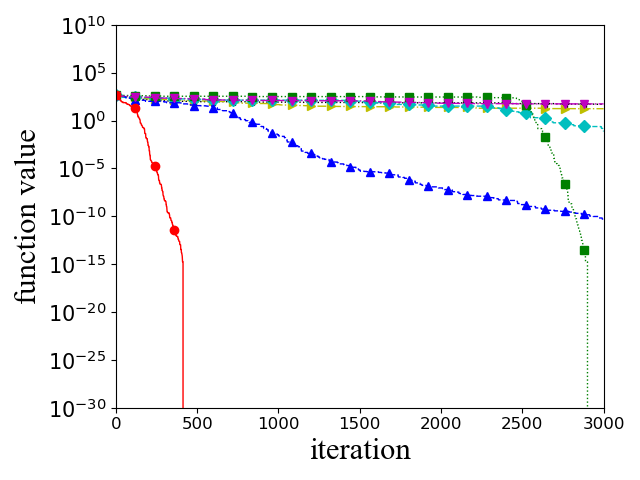
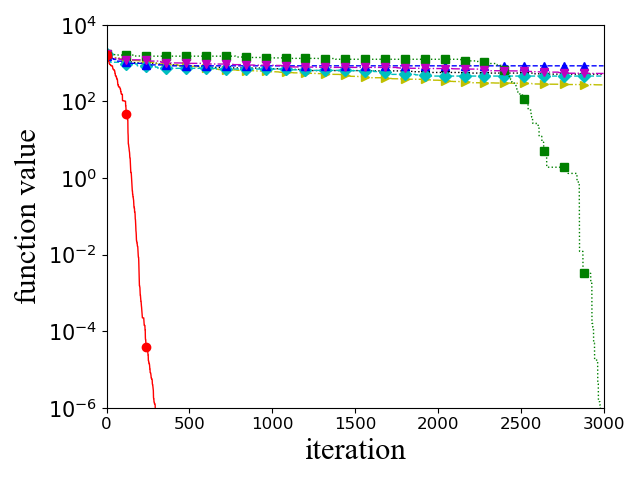
|  |  |
| --- | --- |
| 1. **30-dimension situation** | 1. **100-dimension situation** |

图10 Ackley函数进化曲线

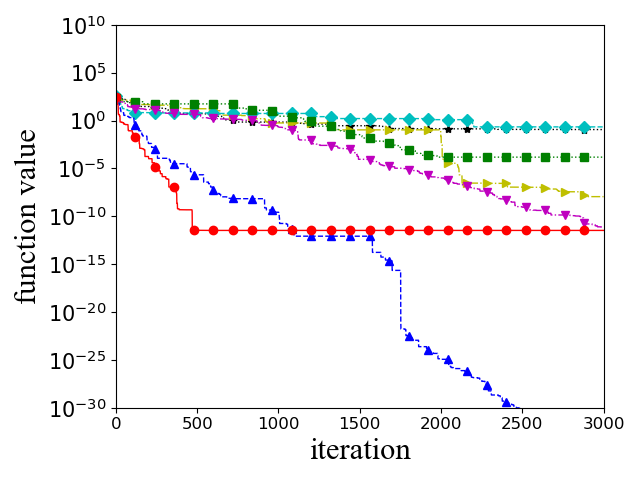
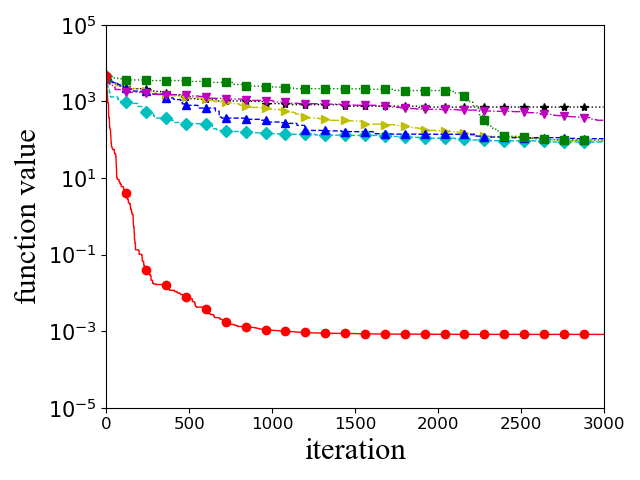
|  |  |
| --- | --- |
| 1. **30-dimension situation** | 1. **100-dimension situation** |

图11 Griewanks函数进化曲线

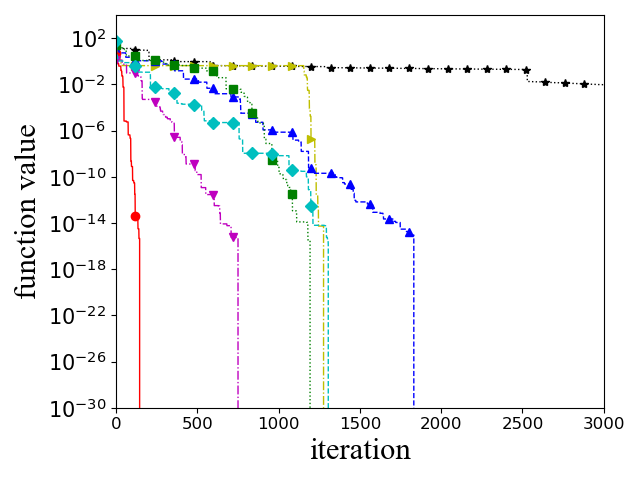
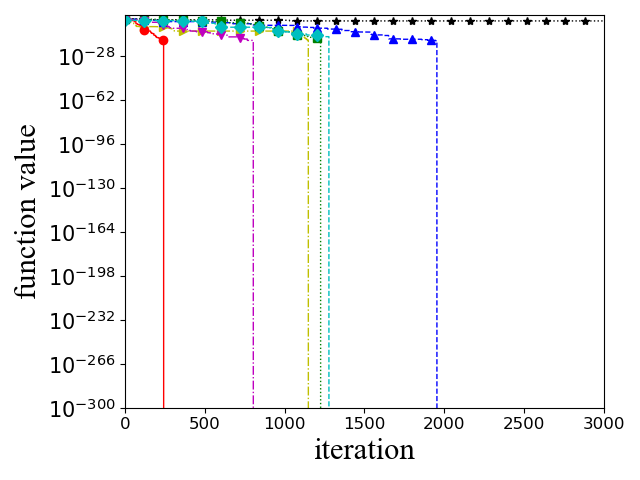
|  |  |
| --- | --- |
| 1. **30-dimension situation** | 1. **100-dimension situation** |

图12 Rastrigin函数进化曲线

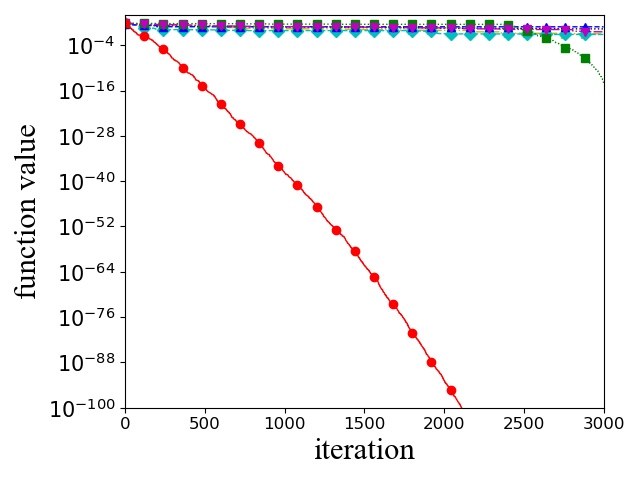
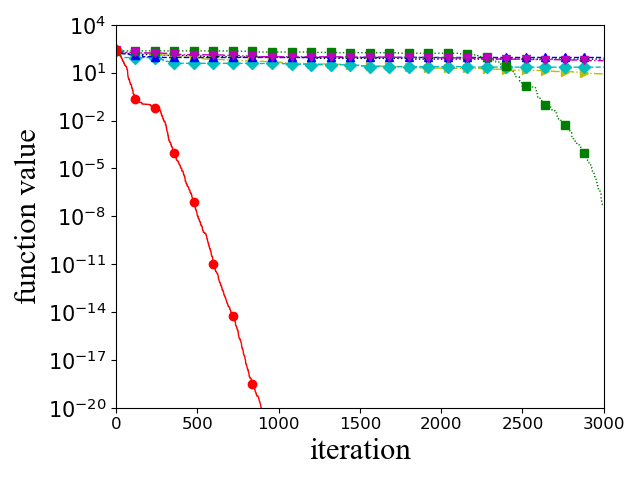
|  |  |
| --- | --- |
| 1. **30-dimension situation** | 1. **100-dimension situation** |

图13 Levy函数进化曲线

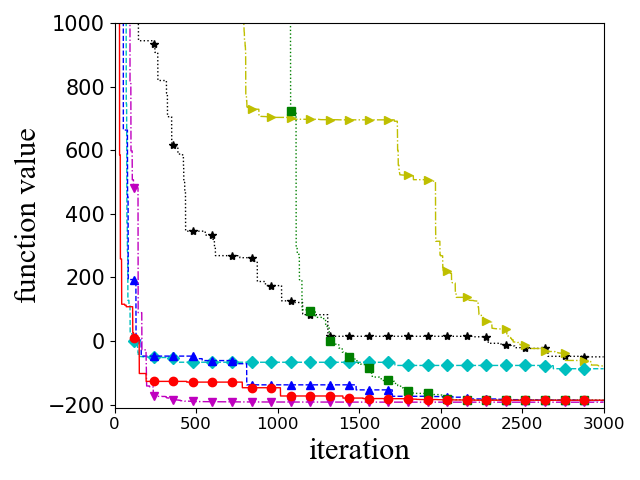
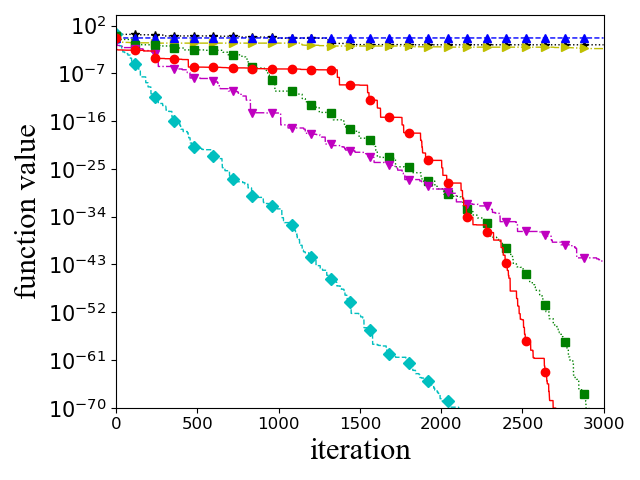
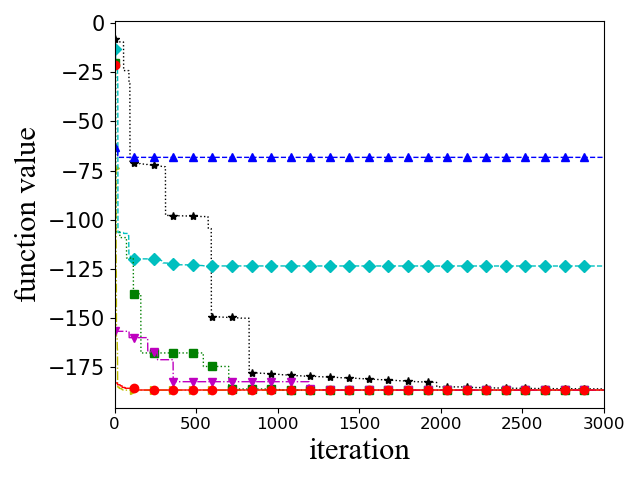
|  |  |
| --- | --- |
| 1. **30-dimension situation** | 1. **100-dimension situation** |

图14 Bohachevsky函数进化曲线

|  |  |
| --- | --- |
| 1. **30-dimension situation** | 1. **100-dimension situation** |

图15 Alpine 1函数进化曲线



|  |  |  |
| --- | --- | --- |
| 图16 Shubert函数进化曲线 | 图17 Matyas函数进化曲线 | 图18 Trid 10函数进化曲线 |

从图7~18的进化曲线结果来看，本文提出的IGHS-LF-OBL算法在与其他六种HS算法的对比中，在大多数情况下都具备较好的收敛速度和精度，且IGHS-LF-OBL算法在面对高纬度问题时，对比其他六种HS算法，也具备着更好的收敛速度和精度。比如，对于函数Sphere、Griewanks、Rastrigin、Bohachevsky和Alpine 1，无论是在30维和100维情况下，IGHS-LF-OBL算法的性能都优于其他六种HS算法；对于函数Step、Ackley和Levy，在30维时虽然各方面性能不是最好的，但在100维时，IGHS-LF-OBL能够跳出局部最优，表现出更好的收敛速度和精度；对于固定尺寸、最优值为固定负数的Shubert函数和Trid 10函数，IGHS-LF-OB的收敛速度和精度仍然具备较好的效果。

为了更好了解试验结果，表3从数值的角度记录了30次运行的试验比对结果，其中Dim表示维度，MEAN表示30次最优目标函数值均值，用于反映出算法的收敛精度；STDV表示30次最优和声函数值标准差，用于反映出算法的稳定性。对比效果好的值加黑表示。

分析表3中的实验结果，也可得出在针对高维函数优化时，IGHS-LF-OBL相较于其他HS算法具有较高的最终收敛精度，且具有相对较好的稳定性，如函数*F*2、*F*4、*F*7在维度为10维、30维时效果并不是最好的，但在100维时MEAN和STDV都最小，显示出较好的收敛精度和稳定性。从单个函数看，除IGHS-LF-OBL的其他六种HS算法在针对具体函数时都各有优势，但并不稳定；总体上看，IGHS-LF-OBL的全局最优和自适应能力较好。

综合图7~18及表3可知，较其他6钟改进和声搜索算法，本研究提出的IGHS-LF-OBL算法具有更好的全局搜索能力和自适应能力。

表3 7种HS算法在12个测试函数下的不同维度实验对比结果

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | HS | NGHS | GOGHS | IDHS | ID-HS-LDD | IMGHSA | IGHS-LF-OBL |
| Dim | MEANSTD | MEANSTD | MEANSTD | MEANSTD | MEANSTD | MEANSTD | MEANSTD |
| *F*1 | 10  30  100 | 5.67E+004.14E+00  1.75E+031.03E+02  5.61E+041.02E+03 | 4.91E-079.83E-07  3.88E+017.67E+01  1.25E+032.61E+00 | 1.01E-614.81E-62  6.15E-101.02E-10  5.05E+022.12E+01 | 1.84E-492.19E-50  1.12E-111.41E-12  1.70E+036.23E+01 | 1.77E-413.27E-41  2.57E-254.41E-25  2.23E-093.93E-09 | 3.19E-151.29E-15  2.64E+017.73E-01  5.02E+035.81E+01 | **0.00.0**  **1.49E-2910.0**  **3.51E-2500.0** |
| *F*2 | 10  30  100 | 5.35E+004.14E-01  1.00E+033.48E+01  5.23E+047.65E+02 | 1.83E-112.37E-14  8.31E+001.61E+01  3.63E+033.16E+03 | 9.65E-014.14E-01  7.24E+001.16E+00  5.94E+025.24E+01 | **0.00.0**  **3.94E-123.75E-13**  9.75E+023.34E+01 | 1.78E-032.69E-04  5.91E+001.91E-01  2.35E+012.95E+00 | 2.91E-095.17E-10  2.00E-013.29E-02  4.71E+032.29E+01 | 1.60E-154.00E-16  9.94E-085.17E-08  **5.28E-056.77E-06** |
| *F*3 | 10  30  100 | 3.86E+011.55E+00  6.61E+015.89E+01  1.54E+016.46E+00 | 2.89E+028.43E+01  2.49E-021.15E-04  6.71E+015.15E+00 | 3.41E-085.39E-08  2.16E-123.12E-12  1.14E-113.73E-12 | 6.81E-021.11E-02  **0.00.0**  1.52E-342.52E-34 | 3.19E-454.00E-45  1.01E-022.11E-02  6.37E-481.34E-47 | **0.00.0**  **0.00.0**  **0.00.0** | 2.87E-039.91E-04  4.13E-072.51E-08  3.27E-052.34E-05 |
| *F*4 | 10  30  100 | 3.59E+001.31E+00  8.93E+004.68E+00  1.76E+014.85E+00 | 5.86E-031.10E-04  2.98E+002.13E+00  8.28E+003.28E+00 | 7.10E-150.0  9.49E-061.01E-06  2.35E+007.28E-01 | 3.55E-150.0  1.33E-066.62E-08  5.65E+001.22E-01 | **0.00.0**  **0.00.0**  3.09E-063.52E-07 | 4.02E+001.52E+00  4.95E+006.33E-01  1.36E+019.54E+00 | 3.55E-150.0  3.55E-150.0  **3.55E-150.0** |
| *F*5 | 10  30  100 | 1.10E+004.43E-01  1.13E+012.16E+00  4.70E+023.83E+02 | 1.94E-011.63E-02  1.79E+013.37E+00  2.73E+011.75E+01 | 8.88E-160.0  1.23E-082.33E-09  5.47E+002.23E+00 | **0.00.0**  1.35E-081.66E-09  6.80E+003.44E+00 | **0.00.0**  **0.00.0**  6.44E-101.09E-09 | 2.21E-018.73E-02  1.00E+001.67E-01  5.19E+011.00E+01 | **0.00.0**  **0.00.0**  **0.00.0** |
| *F*6 | 10  30  100 | 4.28E+006.87E-01  5.89E+019.61E+00  6.36E+023.87E+02 | 6.82E+001.70E+00  2.86E+017.99E+00  2.57E+026.03E+00 | 3.55E-150.0  1.42E+029.30E+00  3.86E+028.21E+00 | **0.00.0**  1.93E-141.57E-15  8.73E+022.64E+01 | **0.00.0**  **0.00.0**  1.39E-071.06E-07 | 6.96E+001.49E+00  9.63E+012.00E+00  6.80E+021.23E+01 | **0.00.0**  **0.00.0**  **0.00.0** |
| *F*7 | 10  30  100 | 1.13E-018.86E-02  2.56E+015.41E+00  7.19E+029.77E+00 | 1.12E-081.96E-10  2.18E-013.43E-02  9.02E+012.50E+00 | 2.16E-012.77E-02  4.14E+001.63E+00  8.66E+012.41E+00 | **1.34E-312.53E-33**  **9.85E-098.77E-10**  1.17E+026.35E+00 | 2.63E-044.96E-05  2.46E+016.24E+00  9.93E+015.61E+00 | 1.16E-112.93E-12  1.19E+009.96E-01  3.35E+015.91E+00 | 3.31E-124.03E-15  4.43E-053.62E-05  **8.22E-043.15E-06** |
| *F*8 | 10  30  100 | 9.43E-044.35E-04  9.47E-025.29E-03  2.54E-017.22E-02 | 1.81E-143.62E-15  4.98E-099.97E-09  6.76E-085.54E-08 | **0.00.0**  **0.00.0**  **0.00.0** | **0.00.0**  **0.00.0**  **0.00.0** | **0.00.0**  **0.00.0**  **0.00.0** | **0.00.0**  **0.00.0**  **0.00.0** | **0.00.0**  **0.00.0**  **0.00.0** |
| *F*9 | 10  30  100 | 4.81E-021.33E-02  3.47E+008.39E-01  7.03E+021.13E+01 | 1.37E-093.84E-10  4.95E-022.29E-02  1.25E+012.94E+00 | 4.96E-028.04E-03  1.66E-012.69E-02  2.12E+015.85E+00 | 1.21E-073.30E-08  2.88E-021.76E-03  1.22E+025.33E+00 | 4.97E-207.12E-21  8.04E-141.03E-15  2.08E-052.31E-06 | 5.74E-061.74E-08  8.00E-018.47E-02  6.60E+017.01E+00 | **2.97E-2110.0**  **1.18E-1610.0**  **8.17E-1075.06E-111** |
| *F*10 | 2 | -171.778.65 | -184.531.16 | -175.675.64 | -180.392.96 | **-186.730.0** | **-186.730.0** | **-186.730.0** |
| *F*11 | 2 | 2.25E-015.35E-02 | 4.92E-039.57E-05 | **5.81E-1153.91E-117** | 1.47E-014.73E-02 | 4.86E-709.45E-71 | 1.57E-491.09E-50 | 4.79E-802.57E-81 |
| *F*12 | 10 | 146.3913.23 | 39.426.74 | -69.7710.21 | -174.0315.97 | -187.209.26 | **-191.247.71** | -190.778.13 |
| 注：当计算数量级小于1E-330时，计算机显示结果为0.0 | | | | | | | | |

第5章 基于IGHS-LF-OBL的模糊Petri网参数优化

一个FPN由若干个库所、变迁、权值、可信度和阈值组成，输入由若干个输入库所组成，输出由若干个输出库所组成。在实际应用中，库所和变迁是事先确定的，而权值、可信度和阈值大多依靠专家经验给出，这大大增加了FPN的应用难度和使用精度，若使用其他智能算法对FPN进行自适应参数寻优，则必定要以FPN的最终输出库所的模糊推理函数作为目标函数，随着FPN的扩大，最终的目标函数也会变得越来越复杂，局部最优也逐渐增多，要求智能算法必须要有强大的全局寻优能力和自适应能力。本文在第四章的仿真实验中验证了IGHS-LF-OBL具有良好的全局最优和自适应能力，本文通过将FPN的权值、可信度和阈值组成的若干个参数看成一个和声个体，多个和声个体组成一个种群，以输出库所的模糊推理函数作为目标函数，迭代完成后选取最优个体作为优化完成的结果参数。本文基于IGHS-LF-OBL的模糊Petri网参数优化大体上分为四个部分：1.确定FPN结构。2.确定目标函数。3.和声个体权值归一化。4.迭代得到FPN参数。

5.1 确定FPN结构

在一个专家系统中，有一个确定的FPN，已知库所各自对应着有关命题，它们各命题之间存在如下的模糊产生式规则：









根据上述模糊产生式规则，建立模糊产生式规则与之对应的FPN模型如图19所示。



图19 FPN模型

在图19的FPN模型中，有四层变迁，其中，第一层变迁有，第二、三、四层变迁分别为。从图中可以看出，和并不在同一层变迁中，但是输出对应同一输出库所节点，为了模型具有更良好的层次关系和模糊推理逻辑，在和之间引入一个虚部库所节点和虚部变迁，同时将指向，作为变迁，的输出指向，并且将虚部变迁的阈值和确信度设置为0和1，如图20。这样建立的FPN具有更好的层次结构和模糊推理逻辑，也不会改变规则库系统。



图20 添加虚部后的FPN模型

5.2 确定目标函数

利用第3章提到的模糊推理函数对FPN进行逐层模糊推理，推理顺序为：

1. 首先点燃第一层中的变迁



1. 接着点燃第二层中的变迁和



1. 再点燃第三层的变迁



1. 最后点燃第四层变迁



本文采用有监督学习训练方法，随机生成若干组输入库所标记值样本作为训练数据，计算出对应的期望输出库所标记值和实际输出库所标记值，以实际输出标记值与FPN期望输出标记值的差的平方和作为目标函数，目标函数定义为：，其中*S*为生成的样本组数，*i*=1,2,...,*S*。

5.3 和声个体权值归一化操作

在一个FPN中，如果出现多个库所指向同一个变迁时，也就是与规则，各路权值和

为1。在和声个体中同样也遵循同样的设置，即每次迭代计算目标函数时，将个体中对应权值做归一化操作，将，其中，为第T次迭代时，与规则中库所到变迁的权值。

5.4 迭代得到FPN参数

以搜索模糊推理函数最小值为目标，对参数进行寻优，迭代完成后输出最优和声个体值，得到FPN参数。

5.5 FPN参数优化仿真实验

使用上述相同的方法，将IGHS-LF-OBL算法与GA、PSO、DE三种群体智能算法针对图20中同一个FPN进行参数优化，其中IGHS-LF-OBL使用本文给出的算法流程，GA、PSO、DE算法使用python在github中的开源库scikit-opt进行计算[32]。搜索区间和维度都设置相同，在迭代次数为2500次时，评价指标为迭代完成时的目标函数值和各参数的MSE；在迭代次数为3000次时，同上方法，迭代完成后随机产生10组数据对各算法进行泛化性测试，评价指标为期望输出和实际输出的MSE和。

5.5.1 参数设置及实验环境

实验环境同第4章。迭代次数设置为2500次，搜索区间设置为[0.0,1.0]，维度设置为15，四种算法参数设置见表4。事先假设图20中FPN的权值、阈值和可信度等参数值，并将推理函数中的常量*b*设置为105，事先假设好的参数设置见表5。

表4 四种群体智能算法参数设置

|  |  |
| --- | --- |
| 算法名称 | 参数 |
| IGHS-LF-OBL | *S*HM*=*20*, P*HMCRmin*=*0.8*, P*HMCRmax*=*0.9*,P*PARmin*=*0.1*, P*PARmax*=*0.9*,* |
| GA | *S*pop=50,*P*cro*=*0.5,*P*var=0.001 |
| DE | *S*pop=50,*P*var=0.001,*F*var=0.5 |
| PSO | *S*pop=50,*w=*0.8,*C*1=0.5,*C*2=0.5 |

表5 FPN模型中的期望参数值

|  |  |
| --- | --- |
| 参数名 | 期望参数值 |
| 权值 |  |
| 确信度 |  |
| 阈值 |  |

5.5.2 FPN参数优化实验结果及分析

随机生成30组输入库所标记值作为训练样本，为了防止变迁输入值过小不能点燃从而产生过多无效训练数据影响参数优化结果，其中20组样本从[0.5,1.0]之间生成，10组样本在[0.0,1.0]之间生成。之后，将IGHS-LF-OBL与其他三种群体智能算法GA、DE、PSO对FPN进行参数寻优，各独立运行30次，取各自30次运行中迭代完成后目标函数值最小的一次做实验对比。图21为获得的四种参数优化算法的收敛曲线，迭代完成适应度值越小，说明该算法对FPN的推理目标函数收敛效果越好。图22为对应IGHS-LF-OBL在训练过程中各参数的MSE曲线图，其中*w*表示参数权值的MSE曲线，*u*表示参数可信度的MSE曲线，*t*表示参数阈值的MSE曲线，Sum表示三种参数的平均MSE曲线，MSE曲线值越小，说明优化得到的参数与假设期望参数越接近。

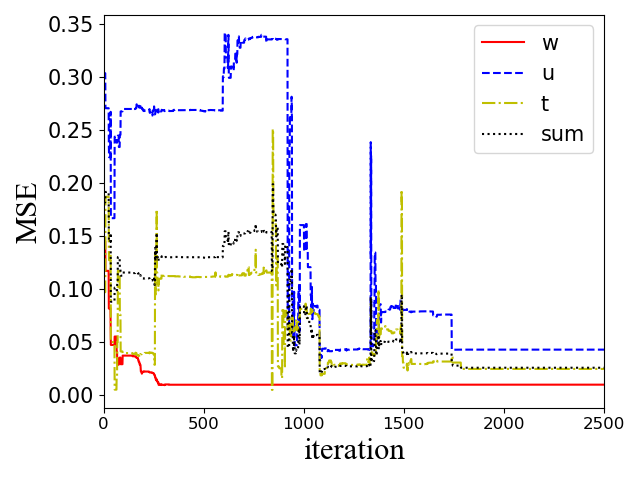
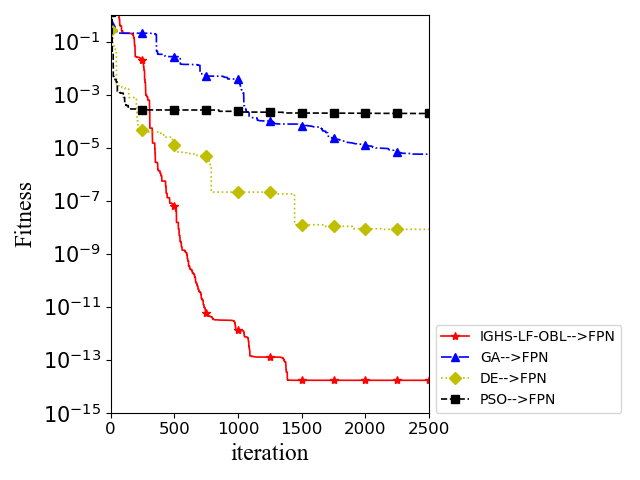


图21 四种算法FPN参数优化适应度曲线 图22 IGHS-LF-OBL各参数MSE进化曲线

图21中可见，随迭代次数增加，各算法的适应度收敛曲线都呈下降趋势，其中，最终收敛精度最高的是IGHS-LF-OBL，其次是DE、GA、PSO，再次验证了IGHS-LF-OBL良好的全局搜索和自适应性能。在图22中，当迭代次数达到2500次时，各曲线均已平稳且到达一个较小的数值，其中，权值*w*的优化效果最好，其次是阈值*t*和可信度*u*，总体MSE也达到了较小的数值。对比图21和图22，随着适应度函数值的减少，MSE并没有相应的减少，而是呈现出上下波动式减少的趋势，这是因为算法的目标函数是期望输出值与实际输出值差的平方和，并不是参数的MSE，且目标函数存在多个局部最优，因此两个收敛曲线间并没有直接关联。

为了更好的观察试验结果，表6列出了各参数优化算法迭代完成后的三种参数取值和各参数的MSE值，其中TMSE为参数总体平均MSE值。

表6 迭代2500次后四种算法的FPN参数取值

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 算法 | 参数 | 序号 | | | | | MSE(10-3) | TMSE(10-3) |
| 1 | 2 | 3 | 4 | 5 |
| IGHS-IF-OBL | w | 0.2297 | 0.4256 | 0.3446 | 0.3247 | 0.6752 | 9.8734 | 26.0086 |
| u | 0.8551 | 0.8229 | 0.3791 | 0.8718 | 0.6541 | 43.0223 |
| τ | 0.2260 | 0.4120 | 0.3768 | 0.6156 | 0.3988 | 25.1301 |
| GA | w | 0.2373 | 0.3971 | 0.3655 | 0.4892 | 0.5107 | 16.1001 | 130.3522 |
| u | 0.6469 | 0.9756 | 0.9414 | 0.5257 | 0.8033 | 105.5299 |
| τ | 0.9920 | 0.6119 | 0.2219 | 0.3819 | 0.3751 | 269.4267 |
| DE | w | 0.2202 | 0.4444 | 0.3352 | 0.3580 | 0.6419 | 4.1307 | 80.7368 |
| u | 0.9446 | 0.5130 | 0.6930 | 0.9655 | 0.6219 | 125.8619 |
| τ | 0.5567 | 0.3532 | 0.5875 | 0.5558 | 0.3450 | 112.2178 |
| PSO | w | 0.3222 | 0.2371 | 0.4405 | 0.3831 | 0.6168 | 52.1760 | 95.4878 |
| u | 0.8975 | 0.9999 | 0.9999 | 0.4765 | 0.7655 | 158.9819 |
| τ | 0.4341 | 0.6311 | 0.4147 | 0.3573 | 0.2871 | 75.3055 |

对比表5和表6可得，IGHS-LF-OBL对FPN的参数优化结果与假设期望参数最接近，且总体MSE也最小，其次是DE、PSO、GA。其中IGHS-LF-OBL的参数优化结果中，权值*w*的取值已经十分接近于期望参数，可信度u与期望参数的接近程度最小，这是因为算法迭代完成后也仅是获得一个较好的局部最优结果，并没有达到真正意义上的全局最优。

为了进一步了解各算法训练得到的FPN参数的泛化性，将迭代次数设置为3000，采用以上相同实验方法，随机在[0.0,1.0]之间生成10组测试样本对各算法的最终参数进行测试，表7记录了每个样本的期望输出值和四种算法的实际输出值，并记录相应MSE。

表7 迭代3000次后四种算法的泛化性比较

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 期望输出 | IGHS-IF-OBL输出 | GA输出 | DE输出 | PSO输出 |
| 1 | 0.504 | 0.504010 | 0.503283 | 0.504287 | 0.494400 |
| 2 | 0.40992 | 0.409914 | 0.413712 | 0.410717 | 0.0 |
| 3 | 0.50176 | 0.501759 | 0.501028 | 0.502138 | 0.491169 |
| 4 | 0.0 | 0.377999 | 0.0 | 0.0 | 0.0 |
| 5 | 0.4032 | 0.403209 | 0.403479 | 0.403349 | 0.0 |
| 6 | 0.56448 | 0.564481 | 0.562740 | 0.564987 | 0.573458 |
| 7 | 0.0 | 0.0 | 0.439718 | 0.0 | 0.434587 |
| 8 | 0.46984 | 0.469837 | 0.472074 | 0.470378 | 0.462585 |
| 9 | 0.0 | 0.0 | 0.397895 | 0.0 | 0.441570 |
| 10 | 0.40208 | 0.402084 | 0.403888 | 0.402635 | 0.0 |
| MSE(10-3) | | 71.4416 | 175.8495 | 0.0008 | 438.2302 |

由表7得，IGHS-LF-OBL的泛化性并不是最好的，总体MSE最小的是DE，其次才是IGHS-LF-OBL，对于期望输出为0.0的情况，也就是变迁并没有被点燃的情况下，并没有很好地计算出来，导致总体MSE较大，相比较下，DE的泛化性较好。

综合图21-22和表6-7，提出的IGHS-LF-OBL算法对比GA、DE、PSO三种群体智能算法，IGHS-LF-OBL算法对FPN的参数寻优总体上更优。

第6章 总结与展望

6.1 总结

针对模糊Petri网自学习能力差、各项参数难以确定等缺陷，提出了一种基于莱维飞行和反向学习策略改进的和声搜索算法IGHS-LF-OBL，并对模糊Petri网进行参数寻优。试验结果表明，所提出的IGHS-LF-OBL算法具有较好的全局搜索性能和自适应能力。将IGHS-LF-OBL算法与GA、DE和PSO三种群体智能算法针对同一模糊Petri网参数优化结果作比较，试验结果表明，IGHS-LF-OBL算法对模糊Petri网的参数寻优效果更好。

6.2 未来展望

1. 本文提出的改进和声搜索算法IGHS-LF-OBL与其他HS算法相比，虽然具有较好的全局寻优性能，但是算法本身时间复杂度较大算法也较为复杂，下一步工作将侧向于在保留当前精度的前提下，减少算法的时间复杂度。

2. 本文针对的FPN参数优化是只有一个输出库所的单目标函数优化，算法在单目标函数下效果较好，不一定在有多个输出库所的FPN下效果一样好，下一步研究方向将试图使用IGHS-LF-OBL针对有多个输出库所的FPN进行多目标函数参数优化。

参考文献

1. WOLFGANG R.Understanding petri nets : modeling techniques, analysis methods, case studies[M]. Berlin:Springer, 2016:1-10.
2. LOONEY C G.Fuzzy Petri nets and application[C]//Tzafestas S G.Fuzzy Reasoning in Information,Decision and Control Systems. Norwell,MA：Kluwer Academic Publishers,1994:511-527.
3. ALI A P, MEISAM Y T. FPN‐SAODV: using fuzzy petri nets for securing AODV routing protocol in mobile Ad hoc network[J]. International Journal of Communication Systems, 2017, 30(1) : 1-14.
4. ZHOU Kaiqing, AZLAN M Z. Fuzzy petri nets and industrial applications: A review[J]. Artificial Intelligence Review, 2016, 45(4) : 405-446.
5. LI Hui. Acquiring and sharing tacit knowledge based on interval 2-tuple linguistic assessments and extended fuzzy petri nets[J]. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 2018, 26(1) : 43-65.
6. YANG Hongyu, FENG Yuhao. A pythagorean fuzzy petri net based security assessment model for civil aviation airport security inspection information system[J]. International Journal of Intelligent Systems, 2021, 36(5) : 2122-2143.
7. 鲍培明.基于BP网络的模糊Petri网的学习能力[J].计算机学报.2004.27(5):695-702.
8. GEEM Z W, KIM J H, LOGANATHAN G V. A new heuristic optimization algorithm: harmony search[J]. Simulation, 2001, 76(2):60-68.
9. DEB K , PRATAP A , AGARWAL S , et al. A fast and elitist multiobjective genetic algorithm: NSGA-II[J]. IEEE Transactions on Evolutionary Computation, 2002, 6(2):182-197.
10. PUGH J, SEGAPELLI L, MARTINOLI A. Ant colony optimization and swarm intelligence[J]. Lecture Notes in Computer Science, 2004, 49(8):767-771.
11. LIANG J J, QIN A K, SUGANTHAN P N, et al. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions[J]. IEEE transactions on evolutionary computation, 2006, 10(3):281-295.
12. LEE K S, GEEM Z W. A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice[J]. Computer Methods in Applied Mechanics and Engineering, 2005, 194(36):3902-3933.
13. LI Hangcheng, ZHOU Kaiqing, MO Liping, et al. Weighted fuzzy production rule extraction using modified harmony search algorithm and bp neural network framework[J]. IEEE Access,2020,8:186620-186637.
14. MOAYEDIKIA A, ONG K L, BOO Y L, et al. Feature selection for high dimensional imbalanced class data using harmony search[J]. Engineering Applications of Artificial Intelligence, 2017, 57:38-49.
15. BEIGZADEH M M B, ABOLGHASEM M S. Harmony search path detection for vision based automated guided vehicle[J]. Robotics and Autonomous Systems, 2018, 107:156-166.
16. ZOU Dexuan, GAO Liqun . A novel global harmony search algorithm for task assignment problem[J]. Journal of Systems and Software, 2010, 83(10):1678-1688.
17. WANG Lin , HU Huanlin , LIU Rui , et al. An improved differential harmony search algorithm for function optimization problems[J]. Soft Computing, 2019, 23(13):4827-4852.
18. ZHU Qidan, TANG Xiangmeng, LI Yong. An improved differential-based harmony search algorithm with linear dynamic domain[J]. Knowledge-Based Systems,2020,187:1-14.
19. GUO Zhaolu. Global harmony search with generalized opposition-based learning[J]. Soft Computing, 2017, 21(8) : 2129-2137.
20. JAFAR G, KAREEM K A, HOSSAM M Z. A novel global harmony search algorithm for solving numerical optimizations[J]. Soft Computing, 2020: 1-13.
21. LI Xiaoou. Dynamic knowledge inference and learning under adaptive fuzzy petri net framework.[J]. IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 2000,30(4):442-449.
22. LI Xiaoou. Adaptive fuzzy petri nets for dynamic knowledge representation and inference[J]. Expert Systems with Applications, 2000, 19(3):235-241.
23. WU S Q. A fast approach for automatic generation of fuzzy rules by generalized dynamic fuzzy neural networks[J]. IEEE Trans on Fuzzy Systems,2001,9(4):578-594.
24. 郑寇全,雷英杰,王睿,王毅,申晓勇.基于BP算法的IFPN参数优化方法[J].控制与决策,2013,28(12):1779-1785.
25. 李洋,乐晓波.克隆选择算法在优化模糊Petri网参数中的应用[J].计算机工程与应用,2011,47(15):39-42.
26. 张弛,乐晓波,周恺卿,莫礼平.采用差分进化算法优化模糊Petri网参数[J].计算机工程与科学,2014,36(06):1095-1100.
27. JIANG Wei, ZHOU Kaiqing, MO Liping. Parameter optimization strategy of fuzzy petri net utilizing hybrid GA-SFLA algorithm[J].Simutools,2019,295:416-426.
28. 李航程,周恺卿,莫礼平.基于全局最优的自适应和声搜索算法[J/OL].山东大学学报(工学版):1-10[2021-04-21].http://kns.cnki.net/kcms/detail/37.1391.T.20210315.1033.002.html.
29. YANG X S, DEB S. Engineering optimisation by cuckoo search[J]. International Journal of Mathematical Modelling and Numerical Optimisation, 2010, 1(4):330-343.
30. SALGOTRA R, SINGH U, SAHA S, et al. Self adaptive cuckoo search: analysis and experimentation[J]. Swarm and Evolutionary Computation, 2020, 60:100751.
31. TIZHOOSH, H R. Opposition-based learning: a new scheme for machine intelligence[C]// International Conference on International Conference on Computational Intelligence for Modelling, Control and Automation. Vienna,Austria: IEEE, 2005:695-701.
32. KAI H, ZSCHECH P, JANIESCH C,et al. Process data properties matter: Introducing GCNN and KVP for next event prediction with deep learning[J]. Decision Support Systems, 2021:113494.

附 录

IGHS-LF-OBL主要改进函数代码

def improvise(self):

lhc\_levy = self.levy\_flight(1.5)

ind = HSIndividual(self.vardim, self.bound, 0)

ind.chrom = np.zeros(self.vardim)

for i in range(0, self.vardim):

worstIdx = np.argmax(self.fitness)

bestIdx = np.argmin(self.fitness)

r1 = random.random()

r2 = random.random()

if r1 <= self.params2[0]:

if r2 < self.params2[1]:

ind.chrom[i] = self.population[bestIdx].chrom[random.randint(0, self.vardim - 1)]

else:

ind.chrom[i] = self.population[worstIdx].chrom[i] + lhc\_levy[i] \* 0.005 \*(

self.population[bestIdx].chrom[i] - self.population[worstIdx].chrom[i])

if ind.chrom[i] < self.bound[0, i] or ind.chrom[i] > self.bound[1, i]:

ind.chrom[i]=self.bound[0, i]+(self.bound[1, i]-self.bound[0, i])\*random.random()

else:

ind.chrom[i] = self.population[random.randint(0, self.sizepop - 1)].chrom[i] + lhc\_levy[i] \* 0.01\* (

self.population[random.randint(0, self.sizepop - 1)].chrom[i] -

self.population[random.randint(0, self.sizepop - 1)].chrom[i])

if ind.chrom[i] < self.bound[0, i] or ind.chrom[i] > self.bound[1, i]:

ind.chrom[i] = self.bound[0, i] + (self.bound[1, i] - self.bound[0, i]) \* random.random()

ind.calculateFitness()

return ind

def levy\_flight(self, Lambda=1.5):

sigma1 = np.power((math.gamma(1 + Lambda) \* np.sin((np.pi \* Lambda) / 2)) \

/ math.gamma((1 + Lambda) / 2) \* np.power(2, (Lambda - 1) / 2), 1 / Lambda)

sigma2 = 1

u = np.random.normal(0, sigma1, size=self.vardim)

v = np.random.normal(0, sigma2, size=self.vardim)

step = u / np.power(np.fabs(v), 1 / Lambda)

return step

def second\_OBL\_best\_worst\_improvise(self):

inf = 0x3f3f3f3f

A = np.tile([inf], self.vardim)

B = np.tile([-inf], self.vardim)

for i in range(0, self.vardim):

for j in range(0, self.sizepop):

if A[i] > self.population[j].chrom[i]:

A[i] = self.population[j].chrom[i]

if B[i] < self.population[j].chrom[i]:

B[i] = self.population[j].chrom[i]

OBL\_X\_best=HSIndividual(self.vardim, self.bound, 0)

OBL\_X\_worst = HSIndividual(self.vardim, self.bound, 0)

bestIdx = np.argmin(self.fitness)

worstIdx = np.argmax(self.fitness)

for i in range(0, self.vardim):

OBL\_X\_best.chrom[i] = random.random()\*(A[i] + B[i]) - self.population[bestIdx].chrom[i]

OBL\_X\_worst.chrom[i] = random.random()\*(A[i] + B[i]) - self.population[worstIdx].chrom[i]

if OBL\_X\_best.chrom[i] < self.bound[0, i] or OBL\_X\_best.chrom[i] > self.bound[1, i]:

OBL\_X\_best.chrom[i] = self.bound[0, i] + (self.bound[1, i] - self.bound[0, i]) \* random.random()

if OBL\_X\_worst.chrom[i] < self.bound[0, i] or OBL\_X\_worst.chrom[i] > self.bound[1, i]:

OBL\_X\_worst.chrom[i] = self.bound[0, i] + (self.bound[1, i] - self.bound[0, i]) \* random.random()

OBL\_X\_best.calculateFitness()

OBL\_X\_worst.calculateFitness()

self.update\_best(OBL\_X\_worst)

self.update\_worst(OBL\_X\_best)

FPN参数优化对象代码

class Fuzzy\_petri\_net():

def init(self,chrom,test,input\_list):

###########################################

self.w1\_standard = 0.2;self.w2\_standard = 0.5;self.w3\_standard = 0.3;self.w4\_standard = 0.4;self.w5\_standard = 0.6

self.u1\_standard = 0.7;self.u2\_standard = 0.9;self.u3\_standard = 0.6;self.u4\_standard = 0.8;self.u5\_standard = 0.7

self.t1\_standard = 0.3;self.t2\_standard = 0.4;self.t3\_standard = 0.2;self.t4\_standard = 0.5;self.t5\_standard = 0.4

###########################################

self.w1 = chrom[0];self.w2 = chrom[1];self.w3 = chrom[2];self.w4 = chrom[3];self.w5 = chrom[4]

self.u1 = chrom[5];self.u2 = chrom[6];self.u3 = chrom[7];self.u4 = chrom[8];self.u5 = chrom[9]

self.t1 = chrom[10];self.t2 = chrom[11];self.t3 = chrom[12];self.t4 = chrom[13];self.t5 = chrom[14]

sum1=self.w1 + self.w2 + self.w3;sum2=self.w4 + self.w5

if sum1==0.0:

sum1+=1e-10

if sum2==0.0:

sum2+=1e-10

self.w1=self.w1\*1.0/sum1;self.w2=self.w2\*1.0/sum1;self.w3=self.w3\*1.0/sum1;self.w4=self.w4\*1.0/sum2;self.w5=self.w5\*1.0/sum2;

###########################################

self.input\_list = input\_list

self.p1=self.input\_list[test][0];self.p4=self.input\_list[test][1];self.p5=self.input\_list[test][2];self.p7=self.input\_list[test][3]

def fuzzy\_reason\_function(self,x,k):

return 1.0/(1.0+np.exp(-50000\*(x-k)))\*x

def max\_caculate\_con\_function(self,x1,x2):

return x1/(1.0+np.exp(-50000\*(x1-x2)))+x2/(1.0+np.exp(-50000\*(x2-x1)))

def hope\_FPN\_caculate(self):

###第一层

x1=self.fuzzy\_reason\_function(self.p1\*self.u1\_standard,self.t1\_standard)

p2=self.fuzzy\_reason\_function(self.p1\*self.u2\_standard,self.t2\_standard)

###第二层

x2=self.fuzzy\_reason\_function(p2\*self.u3\_standard,self.t3\_standard)

p3=self.max\_caculate\_con\_function(x1,x2)

###第三层

x3=self.p4\*self.w1\_standard+p3\*self.w2\_standard+self.p5\*self.w3\_standard

p6=self.fuzzy\_reason\_function(x3\*self.u4\_standard,self.t4\_standard)

###第四层

x4=p6\*self.w4\_standard+self.p7\*self.w5\_standard

p8=self.fuzzy\_reason\_function(x4\*self.u5\_standard,self.t5\_standard)

#print("hope\_p8=:",p8)

return p8

def real\_FPN\_caculate(self):

###第一层

x1 = self.fuzzy\_reason\_function(self.p1 \* self.u1, self.t1)

p2 = self.fuzzy\_reason\_function(self.p1 \* self.u2, self.t2)

###第二层

x2 = self.fuzzy\_reason\_function(p2 \* self.u3, self.t3)

p3 = self.max\_caculate\_con\_function(x1, x2)

###第三层

x3 = self.p4 \* self.w1 + p3 \* self.w2 + self.p5 \* self.w3

p6 = self.fuzzy\_reason\_function(x3 \* self.u4, self.t4)

###第四层

x4 = p6 \* self.w4 + self.p7 \* self.w5

p8 = self.fuzzy\_reason\_function(x4 \* self.u5, self.t5)

return p8

def MSE(self,chrom):

input\_list = np.load("随机生成的样本数据.npy")

input\_list.tolist()

sum\_mse=0.0

for test in range(0,len(input\_list)):

self.init(chrom,test,input\_list)

hope = self.hope\_FPN\_caculate()

real = self.real\_FPN\_caculate()

sum\_mse+=(hope-real)\*(hope-real)\*0.5

return sum\_mse