目录

第一部分 课程论文题目

论文题目：人工智能基本理论及应用研究综述
论文排版撰写工具：Vscode、LaTeX 和 Zotero。
论文全部配置文件仓库为：https://gitee.com/funpony/learn-tex。
本文参考文献格式：GB/T 7714-2015-numerical。

第二部分 课程论文内容

1 知识表示与推理

1.1 知识与知识表示

知识可以理解为事实和与事实相关的规则的集合。与之类似的是离散数学中的命题逻辑的概念，众所周知命题便是各种事实以及事实之间的推断。和命题逻辑一眼样，人工智能领域中的知识也是为了反应事物或者说是事实之间的联系与规则。相较与命题逻辑，此处的知识要求更高一些，我们希望知识至少是相对正确的（相对正确性），希望知识可以用符号等显式的表示出来以便能更好的利用这些知识（可表示、可利用性），当然知识也有很多不确定性。在不同的先决条件或者不同的规则下，所得到的知识也大相径庭。例如生物领域中“橘生淮南则为橘生于淮北则为枳”，计算机科学中二进制 10 与十进制的 10 并不相同。

知识表示不仅在人工智能领域中出现，其还是认知科学领域的重要内容，在 AI 领域，我们通常希望将知识表示训练的像人一样智慧。

1. AI 知识表示：

   表示方法 我们如何表示知识？

   表示范围 某一表示方法的使用范围？

   表示效果 某种表示方案的效果如何？

   本身性质 知识表示本身的性质问题？

知识表示分为两个步骤，将我们生活中的信息流转化为程序可以识别的信息流，然后模仿人的感知、认知、推理等思维来解决一些人类遇到的费时费力的艰巨任务。

对于 AI 领域的知识表示，通常有以下方法，见表 1。

对于 AI 中各种知识表示方法的优劣，见表 2。

而另一块是关于确定性推理的概念，即依据我们现有的初始信息或依据，按某种确定的推理策略不断运用库中已有知识来逐步得到确定的结论的推理过程。而在 AI 领域，我们的推理是借助程序也叫推理机实现的。

表 1: 知识表示方法及应用.

知识表示方法	应用
一阶谓词逻辑表示法	知识表示与推理
产生式表示法	基于遗传算法的问题求解系统 图搜索求解模型
框架表示法	复杂知识的框架网络
语义网络表示法	机器翻译、问答系统、自然语言理解

1.2 确定性推理

常见的确定性推理方法有图搜索策略，盲目搜索和启发式搜索等，还有最新的像消解原理、规则演绎系统以及产生式系统。确定性推理的特点见表 3，此处我选择启发式搜索 (Heuristic) 来介绍他的原理和应用。

表 2: 各种知识表示方法的优劣.

知识表示方法	优势	劣势
一阶谓词逻辑表示法	自然精确严密易实现	不能表示不确定的知识 耦合度高，效率低
产生式表示法	自然、模块性 有效性、清晰性	不能表达具有结构性的知识 效率不高
框架表示法	结构性、继承性、自然性	缺乏形式理论 适应能力不强
语义网络表示法	强调联系，符合人类思维 描述明确简洁直观 结构化显性描述语义关系	不能保证推论的严格有效 不能处理结点太多的推理 不便表达判断性深层知识

表 3: 确定性推理特点.

确定的推理策略、确定的结论
事实（条件）和知识是构成推理的两个基本要素
以数理逻辑的有关理论、方法和技术为理论基础
机械化的、可在计算机上加以实现

1.2.1 确定性推理方法之启发性搜索算法

启发式搜索方法是一种帮你不断试探出答案的方法，但它给出的答案是具有偶然性的，也就是 subject to chance。如在一个状态空间中，对一点的可能所在的每一个位置进行评估，先得到一个最可能的位置。再从这个位置出发持续进行下一轮的位置评估，这样的循环往复的搜索过程就被称为启发式搜索。而评估过程所用到的估价函数是启发式搜索核心，可以根据状态场景来选取合适的估价函数，常见的如式 1:

$$f(n) = g(n) + h(n) \tag{1}$$

1.2.2 一种启发性搜索算法的应用

天气被认为是影响航班路径规划的最主要因素，有 70% 的航班延误是由于危险天气[1]。Li He 和 Anfei Zhao[2] 使用启发式算法将飞机在危险天气下，不同飞行海拔高度对飞机飞行转态的影响离散化进入一个多边网格模型，结果发现，借助该网格所获得的最小代价的飞行路径比传统的以一个恒定海拔高度飞行的飞行路径要更短。本章认为，当飞行过程中海拔发生变化，则规划的飞行路径也应适当的变化，详情见图 1。

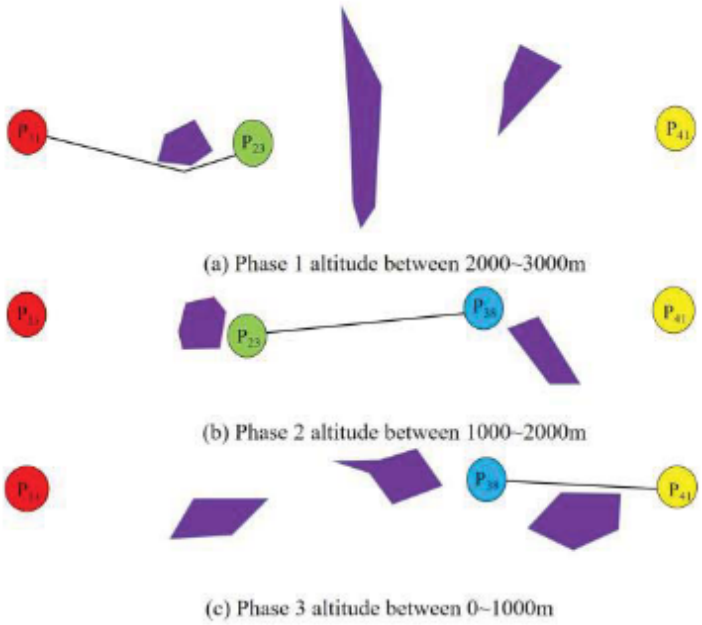

Fig. 3. Planned path after changing altitude

图 1: 海拔变化时路径节点变化图.

这篇文章讨论，如果在决策点改变飞行高度，则从决策点到达下一个路径点的成本不是水平面上的欧几里德距离。因为高度变化被视为在一定爬升角度下的直线飞行，计算到达所选高度后的下一个路径点的代价公式为式 2

$$C_{\text{height}} = \sqrt{d^2_{\text{height}} + h^2} \tag{2}$$

d_{height} 是当前决策点与选定高度中的下一个路径点在水平面上的欧几里得距离，h 是海拔的变化。同时，在网格系统中，使用曼哈顿距离作为代

价估计的启发式函数具有更好的计算速度和搜索效率[3]。这样做可以确保路径搜索方向始终靠近目标，简化搜索空间，排除远离目标的点。得到的启发式搜索函数中 $h(n)$ 为 式 3。

$$h(n) = \left|x_g - x_n\right| + \left|y_g - y_n\right| \tag{3}$$

n 表示当前网格点，g 表示目标点，x_g, x_n 等表示相应结点坐标。

则应用次启发式搜索的规划飞行路径的代价函数就为式 4,C_{height} 表示到达当前结点 n 之前的代价。

$$f(n) = C_{\text{height}} + \left|x_g - x_n\right| + \left|y_g - y_n\right| \tag{4}$$

最终的实验结果表明，根据危险天气和海拔的不同，基于启发式搜索的路径决策算法所得到的航班里程要小于恒定海拔下飞行的航班里程。通过海拔的不断调整，可以将飞机更好的锁定在危险天气影响较小的飞行区域，进而减少平均航行里程[2]。

1.3 不确定性推理

对比于上文中所讲的确定性推理，不确定性推理便好理解了。不确定性推理的初始的条件 (论据) 不确定，推理方法 (或策略) 也不确定，知识库也不确定，最后的推理结果不确定但合理。常见的知识不完备、不精确以及模糊知识等的推理也属于不确定推理的范畴。不确定性推理的特点见表 4

表 4: 不确定性推理特点.

证据的不确定性表示
不确定性的匹配
组合证据不确定性的计算
不确定性的更新
不确定性结论的合成

1.3.1 不确定性推理之主观贝叶斯方法

常见的不确定性推理方法有模糊推理、可信度、证据理论、主观贝叶斯方法等等。本文主要选取主观贝叶斯方法来着重介绍一下。主观贝叶斯方法隶属于规则推理方法，其以贝叶斯理论为依据，克服了实际问题中很难获取到先验概率的问题[4] [5]。我们都知道贝叶斯公式为式 5, 而将全概率公式代入后可得贝叶斯公式的另一种形式，即式 6。

$$P\left(A_i \mid B\right)=\frac{P\left(A_i\right) \times P\left(B \mid A_i\right)}{P(B)}, i=1,2, \ldots, n \tag{5}$$

$$P\left(A_i \mid B\right)=\frac{P\left(A_i\right) \times P\left(B \mid A_i\right)}{\sum_{j=1}^{n} P\left(A_j\right) \times P\left(B \mid A_j\right)} \tag{6}$$

引入产生式规则 IF E THEN Hi，用该规则的前提条件 E 代替 Bayes 公式中的 B，而 A_i 则由 H_i 来代替可得到公式 7。

$$P\left(H_i \mid E\right)=\frac{P\left(E \mid H_i\right) P\left(H_i\right)}{\sum_{j=1}^{n} P\left(E \mid H_j\right) P\left(H_j\right)}, i=1,2, \ldots, n \tag{7}$$

再引入一组不确定性数值对 $(\mathrm{LS}, \mathrm{LN})$ 来表示知识的强度，分别称为充分性度量和必要性度量，即式 8和式 9。

$$LS=\frac{P(E \mid H)}{P(E \mid \neg H)} \tag{8}$$

$$LN=\frac{P(\neg E \mid H)}{P(\neg E \mid \neg H)}=\frac{1-P(E \mid H)}{1-P(E \mid -\neg H)} \tag{9}$$

则此时得到主观贝叶斯公式为式 10。

$$\begin{aligned} P(H \mid E) &=\frac{P(H) \times P(E \mid H)}{P(E)} \\ P(\neg H \mid E) &=\frac{P(\neg H) \times P(E \mid \neg H)}{P(E)} \end{aligned} \tag{10}$$

借助几率函数即式 11最终可得到分段线插值表示的后验概率为式 12。

$$\begin{gathered} O(x)=\frac{P(x)}{1-P(x)}=\frac{P(x)}{P(-x)} \\ P(x)=\frac{O(x)}{1+O(x)} \\ P(x)=0 \text{ 时}, O(x)=0 \\ P(x)=1 \text{ 时}, O(x)=+\infty \end{gathered} \tag{11}$$

$$P(H \mid S)=\begin{cases} P(H \mid \neg E)+\frac{P(H)-P(H|\neg E)}{P(E)} \times P(E \mid S), 0 \leq P(E \mid S)<P(E) \\ P(H)+\frac{P(H|E)-P(H)}{1-P(E)} \times[P(E \mid S)-P(E)], P(E) \leq P(E \mid S) \leq 1 \end{cases} \tag{12}$$

其中 S 是对 E 的观察，而 P(E|S) 表示在观察 S 下，E 发生的概率。

1.3.2 一种主观贝叶斯方法的应用

袁杰等[6]借助改进后的主观贝叶斯方法来更好的进行电熔镁炉熔炼过程异常工况识别，同时利用模糊隶属度函数对个观测状态和证据进行匹配，对金属熔炼过程中的不确定性问题即异常工况获取到更好的识别效果。

这篇文章考虑到主观贝叶斯公式里 L_S 和 L_N 的取值较大，而选择将其值限制在 [0, 1] 中，并设置 P_x 为证据 A 发生时对结论 B 的支持度。并选择合适的映射函数 f，即 e 指数函数式 13。

$$1-k_1 \mathrm{e}^{-k_2 L_{\mathrm{S}}}=p_x \tag{13}$$

从而得到新的 L_S 和 L_N 的概率公式 14和 15。

$$L_{\mathrm{S}}=\frac{\ln\left(1-p_x\right)}{\ln(1-P(B))} \tag{14}$$

$$L_{\mathrm{N}}=\frac{\ln\left(1-p_y\right)}{\ln(1-P(B))} \tag{15}$$

并借助命题事件 $x \geq x_{\mathrm{a}}$ 模糊隶属度函数图 2来获得某证据 A 在观察 S 情况下匹配程度 $P(A \mid S)$，即特征变量观测值 S 特征变量限定范围 A 的匹配关系。改进了传统的主观贝叶斯方法中证据与结论的匹配非 0 即 1（即只有匹配或不匹配的结果）。也提高了异常工况识别中的识别准确率。

$$\boldsymbol{P(x \geqslant x_{\mathrm{a}} \mid x)=\begin{cases} 1-\exp(k \cdot(x_0-x)), & x>x_0; \\ P(x \geqslant x_{\mathrm{a}}), & x=x_0; \\ \exp(k \cdot(x-x_0))-1, & x<x_0. \end{cases}}$$

图 2: 命题模糊隶属度函数.

从现场收集并选取了 200 个异常工况样本，来制图显示电熔镁炉异常运行数据图，见图 3。可以看到，使用改进的主观贝叶斯方法对电熔镁炉

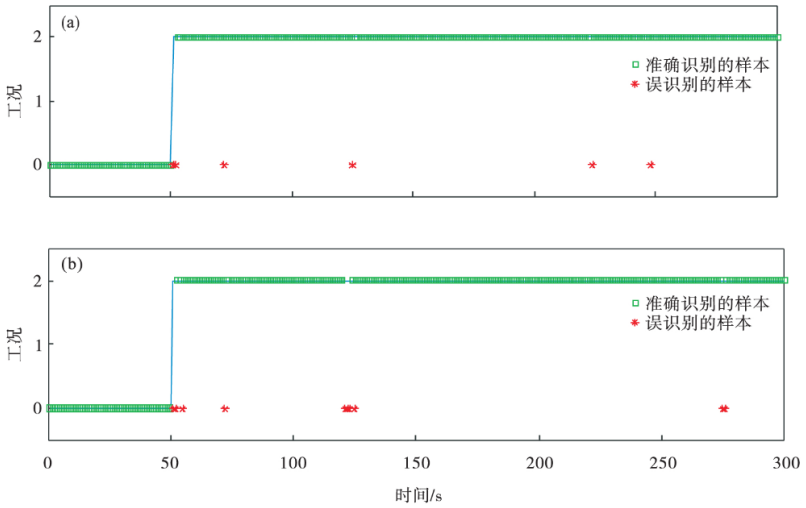

图 3: 两种贝叶斯方法异常工况识别结果.
(a) —改进主观贝叶斯方法; (b) —传统主观贝叶斯方法

熔炼工况进行识别后，相较于传统的主观贝叶斯方法，误识别样本减少了40%，说个改进的主观贝叶斯方法能对异常工况识别更快且提高诊断准确率[6]。

可以发现主观贝叶斯方法的优点是理论基础比较坚实（计算公式多是由概率论的基础所推导出来），且 L_S 和 L_N 由领域内的专家给出，避免了我们自己的数据统计工作。再加上很多时候 LS 和 LN 也反映了证据与结论间的因果关系，也使得结论更为可靠。缺点是应用时，相关领域专家需要同时给出先验概率 P(H)，又由于 Bayes 的独立性假设等等都很难满足，使得该方法使用受限。这也间接看出该方法主要应用在领域专家能给出 L_S 和 L_N 以及事件满足独立性假设的情况，常见的如气象预测、人口预测等。

2 搜索与问题求解

2.1 搜索与问题求解

无信息搜索指的是已知条件很少，只有自己定义的信息可用，所以也称盲目式搜索。而有信息搜索就是找到一种策略来判断某一种搜索到的状

态比另一种更好，进而能朝向更优的方向不断搜索。而超越经典搜索（局部搜索）则不太关心搜搜过程中的路径和代价，只关注解的状态。与之前的无信息搜索是在可观察、确定性以及已知的环境之下搜索不同，局部搜索跳出这些约束，不关心路径，从单个结点出发，通常只移动到该结点的邻近状态，如果存在一个最优解，最优局部搜索状态通常能找到最优解对应的最大或者最小值。

2.2 简单遗传算法解决八皇后问题中理论与设计

2.2.1 遗传算法简介

遗传算法是借鉴了达尔文生物进化学说中种群遗传、基因突变、自然选择和杂交等理论的一种进化搜索算法。

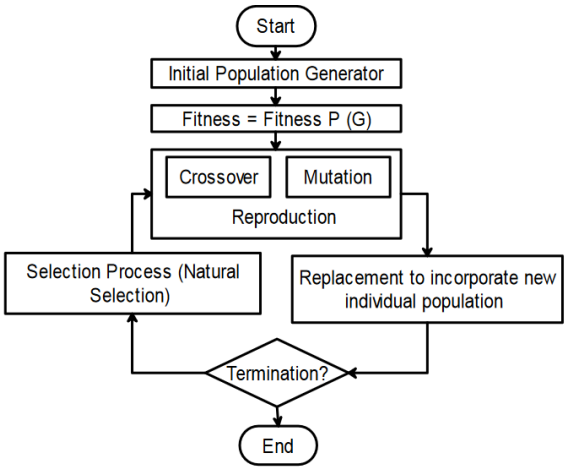

图 4: 简单遗传算法的步骤[7].

1. 遗传算法的具体实现步骤如图 4：

   生成随机候选群、初始化群基因个数。

   设计适应度函数计算 fitness_value。

   定义突变函数和交叉函数。

   定义进化策略，包括丢弃和增补函数。

   不断生成新种群，知道达到适应度要求的值。

2.2.2 遗传算法用于八皇后问题的介绍

我们这次要解决的问题是八皇后问题，用八位数字来代表每个皇后，八个皇后组成遗传种群，并设置遗传种群的大小为 8。用不相互攻击的皇后对的数目来代表 fitness_value（即适应度值）。而如果检查次数（检查当前皇后位置是否与其他所有行列的皇后冲突）达到 28，则代表已找到八皇后的解。而是否选择该种群的某些个体进行下一轮的遗传则取决于个体的适应度值，适应度值越大，被选中的概率越高。简单遗传算法的 Python 实现见列表1，语言环境 Python 3.8.13 64-bit。

Listing 1: 简单遗传算法的 Python 实现[8].

```python
import numpy as np
import copy


# 普通选择算子遗传算法：适应度函数
def fitness_function(individual):
    value = 0
    for i in  range(7):
        for j in  range(i+1, 8, 1):
            if individual[i] != individual[j]:
                x_distance = np. abs(individual[j] - individual[i])
                y_distance = j - i
                if x_distance != y_distance:
                    value += 1
    return value


# 普通选择算子遗传算法：把适应度函数转化为概率分布
def softmax( input):
     input = np.array( input, dtype=np. float)
     input = np.exp( input)
    output =  input /  input. sum()
    return output


# 随机变异
def mutation(individual, prob=0.1):
    p = np.random.rand(8)
    individual[p > prob] = np.random.choice( range(8), 8)[p > prob]
```

```
30
31      return individual
32
33
34  # Genetic Algorithm
35  def GA(size=4):
36      # 默认种群大小为4，可以多试几个，效果不一样
37      size = size
38      num_generation = 0
39      population = []
40      Generation_Fitness = {}
41      for i in  range(size):
42          population.append(np.random.choice( range(8), 8))
43      while (True):
44          fitness_list = []
45          selection = []
46          print("Generation : ", num_generation)
47
48          for individual in population:
49              fitness_value = fitness_function(individual)
50              if fitness_value == 28:
51                  print("Find Target!")
52                  print(individual)
53                  Generation_Fitness[num_generation] = 28
54                  return Generation_Fitness, individual
55              fitness_list.append(fitness_value)
56          Generation_Fitness[num_generation] =  max(fitness_list)
57          print("current max fitnessvalue:" +  str( max(fitness_list)))
58
59          # Selection is Here自然选择在这里
60          prob = softmax(fitness_list)
61          select_id = np.random.choice( range(size), size, replace=True, p=prob)
62          for idx in select_id:
63              selection.append(population[idx])
64          num_pair =  int(size/2)
65          position = np.random.choice( range(1, 7, 1), num_pair, replace=True)
66
67          # Crossover is Here基因片段的交叉互换在这里
68          for i in  range(0, size, 2):
69              start = position[ int(i/2)]
70              tempa = copy.deepcopy(selection[i][start:])
71              tempb = copy.deepcopy(selection[i+1][start:])
```

```
72              selection[i][start:] = tempb
73              selection[i+1][start:] = tempa
74
75          # Mutation is Here变异在这里
76          for i in  range(size):
77              selection[i] = copy.deepcopy(mutation(selection[i], prob=0.8))
78          population = selection
79          num_generation += 1
```

2.2.3 普通遗传算法解决八皇后问题的具体实现

根据种群中每个个体的 fitness_value 在当前种群中总 fitness_value 中的占比（即计算每个个体适应度值的概率分布）来选择一批用于繁衍下一代种群的新个体。实验软件为 Visual Studio Code. Version 1.56.2 测试平台：jupyter notebook，语言环境 Python 3.8.13 64-bit。采用这种策略的实验过程和结果见列表2，图 5和图 6。

Listing 2: 简单遗传算法解决八皇后问题的测试过程和结果.

```
# 导入测试所用到的库
mport numpy as np
import copy
import matplotlib.pyplot as plt
import seaborn as sns

# 导入我们需要用到的遗传相关函数
from Genetic_Algorithm_N_Queen import fitness_function
from Genetic_Algorithm_N_Queen import softmax, mutation
from Genetic_Algorithm_N_Queen import GA

# 调用种群数为100的遗传函数GA
GeneticDict, Queen = GA(size = 8)

# 测试结果
----------------------------------
Generation :  0
current  max fitnessvalue: 24
Generation :  1
current  max fitnessvalue: 25
Generation :  2
```

```
current  max fitnessvalue: 26
Generation :  3
current  max fitnessvalue: 25
Generation :  4
current  max fitnessvalue: 24
Generation :  5
current  max fitnessvalue: 26

--中间遗传过程省略--

Generation :  2285
current  max fitnessvalue: 24
Generation :  2286
Find Target!
[6 4 2 0 5 7 1 3]

示例总遗传代数：2286代
示例计算总耗时：  5.8s
```

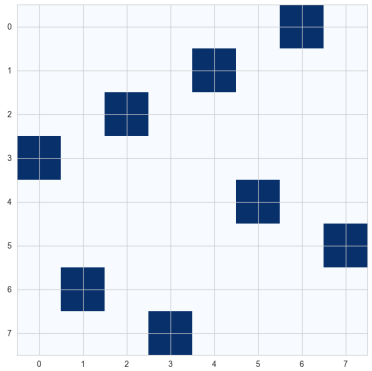

图 5: 简单遗传算法的解 8-Queen 图.

2.3 三种改进遗传算法解决八皇后问题的设计和结论

本文介绍三种遗传算法的改进，其本质是对图 4中某个环节进行了优化，三种改进方法分别是自适应遗传算法，双倍体遗传算法，双种群遗传算法[9]。实验软件为 Visual Studio Code. Version 1.56.2 。语言环境 Python

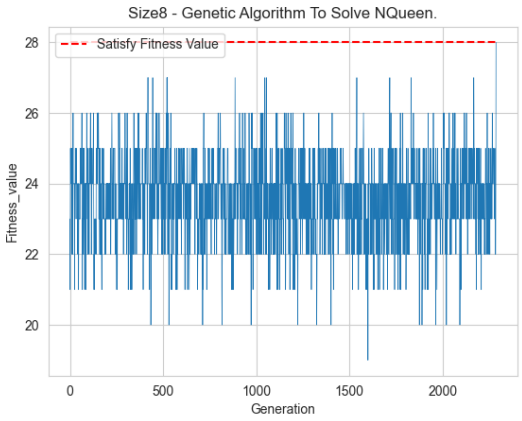

图 6: 简单遗传算法的解收敛过程.

3.8.13 64-bit。三种方法的种群大小均为 8，用每个方法找到所有解并记录每个解的遗传代数。实验结果统一放在代码板网站上，可以点击链接访问。

2.3.1 遗传算法改进之自适应遗传算法解决八皇后问题

自适应遗传算法的改进主要是突变函数和交叉函数的交叉算子方面，不再像普通遗传算法那样以固定的方式来计算突变概率和交叉概率，而是让这两种概率能随着每代种群的适应度变化而变化，所以该算法可以很好的改善局部最优问题，更有利于优良个体的生存。所以，该算法可以更好的优化突变和交叉过程，有利于跳出普通遗传算法容易出现的局部最优陷阱。

对比普通遗传算法，自适应遗传算法主要变化在 hybridization 函数来让每代种群突变和交叉概率随适应度而变化, 详情见 3。

Listing 3: 自适应遗传算法的 Python 实现.

```
# 更新适应度函数
def update_fitness_score(individual):
    value = 0
    for i in  range(8):
        for j in  range(i + 1, 8):
            if individual[i] != individual[j]:
                x = j - i
                y =  abs(individual[i] - individual[j])
                if x != y:
```

```
                    value += 1
    return value


# 交叉产生后代
def hybridization(bianyi1):
    # 选择两个父本
    first = select()
    second = select()
    selected_parents = copy.deepcopy([parent[first], parent[second]])
    # 交换从pos1到pos2的基因
    pos1 = random.randint(0, 6)
    pos2 = random.randint(0, 6)
    # 保证pos1 <= pos2
    if pos1 > pos2:
        pos1, pos2 = pos2, pos1
    # 交叉
    tmp = selected_parents[0][pos1:pos2]
    selected_parents[0][pos1:pos2] = selected_parents[1][pos1:pos2]
    selected_parents[1][pos1:pos2] = tmp
    # 一定的概率发生变异,假设概率为0.5
    may = random.random()
    if may > bianyi1:
        selected_parents[0] = mutation(selected_parents[0])
    may = random.random()
    if may > bianyi1:
        selected_parents[1] = mutation(selected_parents[1])
    # 更新适应度
    first_fit = update_fitness_score(selected_parents[0])
    second_fit = update_fitness_score(selected_parents[1])

    # 加入到子代中
    children.append(selected_parents[0])
    children.append(selected_parents[1])
    children_fitness.append(first_fit)
    children_fitness.append(second_fit)
    return
```

自适应遗传算法完整的实验代码见 https://paste.ubuntu.com/p/wyKvN54byn/, 实验结果（每次找到解的迭代次数）见 https://paste.ubuntu.com/p/g68KF3pxg5/。从实验结果可以看到，最小遗传代数 146 代即获得了

可用的解，相比于普通遗传算法还是提升较为明显。平均遗传代数在 10000 代左右。但其最大迭代次数为 52017 代，远超平均遗传代数和解的时间，且多于 40000 代的情况出现的次数较多，应该是频繁出现过局部收敛现象

2.3.2 遗传算法改进之双倍体遗传算法解决八皇后问题

普通遗传算法在选择和突变过程中是单条染色体，但在生物学上大多数生物是具有多条染色体的（如 AaBb 型），并且存在基因的显性和隐形之分。由于显性基因会在后代中显现所以我们在选择操作中按照染色体显性基因来计算下一代种群的选择概率，在交叉和突变过程也需要根据显隐性分别计算。通过增加显隐性基因重排可以解决下一代的基因显隐性划分，这样做不仅可以增加算法随机性，也能提高染色体基因为隐形的优良个体遗传概率。对比普通遗传算法，双倍体遗传算法主要的变化在 hybridization 函数，在突变、选择和交换过程中引入了隐形基因组成两对基因来进行遗传，详情见 4。

Listing 4: 双倍体遗传算法的 Python 实现.

```
# 交叉产生后代
def hybridization():
    # 选择两个父本
    first = select1()
    second = select1()
    selected_parents = copy.deepcopy([parent1[first], parent1[second]])
    # 交换从pos1到pos2的基因
    pos1 = random.randint(0, 6)
    pos2 = random.randint(0, 6)
    # 保证pos1 <= pos2
    if pos1 > pos2:
        pos1, pos2 = pos2, pos1
    # 交叉
    tmp = selected_parents[0][pos1:pos2]
    selected_parents[0][pos1:pos2] = selected_parents[1][pos1:pos2]
    selected_parents[1][pos1:pos2] = tmp
    # 一定的概率发生变异,假设概率为0.5
    may = random.random()
    if may > 0.5:
        selected_parents[0] = mutation(selected_parents[0])
    may = random.random()
```

```
    if may > 0.5:
        selected_parents[1] = mutation(selected_parents[1])
    # 更新适应度
    first_fit = update_fitness_score(selected_parents[0])
    second_fit = update_fitness_score(selected_parents[1])

    # 加入到子代中
    children1.append(selected_parents[0])
    children1.append(selected_parents[1])
    children_fitness1.append(first_fit)
    children_fitness1.append(second_fit)

    # 选择两个父本
    first = select2()
    second = select2()
    selected_parents = copy.deepcopy([parent2[first], parent2[second]])
    # 交换从pos1到pos2的基因
    pos1 = random.randint(0, 6)
    pos2 = random.randint(0, 6)
    # 保证pos1 <= pos2
    if pos1 > pos2:
        pos1, pos2 = pos2, pos1
    # 交叉
    tmp = selected_parents[0][pos1:pos2]
    selected_parents[0][pos1:pos2] = selected_parents[1][pos1:pos2]
    selected_parents[1][pos1:pos2] = tmp
    # 一定的概率发生变异,假设概率为0.5
    may = random.random()
    if may > 0.5:
        selected_parents[0] = mutation(selected_parents[0])
    may = random.random()
    if may > 0.5:
        selected_parents[1] = mutation(selected_parents[1])
    # 更新适应度
    first_fit = update_fitness_score(selected_parents[0])
    second_fit = update_fitness_score(selected_parents[1])

    # 加入到子代中
    children2.append(selected_parents[0])
    children2.append(selected_parents[1])
    children_fitness2.append(first_fit)
    children_fitness2.append(second_fit)
```

```
64      return
```

双倍体遗传算法完整的实验代码见 https://paste.ubuntu.com/p/sjhVnRjyZx/，实验结果见 https://paste.ubuntu.com/p/tjPmPBwsgy/。从该算法的实验结果可以看到，虽然双倍体遗传算法也出现了迭代 56014 代的现象，但是出现这种局部收敛现象的次数很少且解平均迭代代数为 8000 附近，其找到解的最小迭代代数为 118，相较于自适应遗传算法，其提升较为明显。

2.3.3 遗传算法改进之双种群遗传算法解决八皇后问题

当遗传种群长时间进化后可能会陷入某种相对优势而不是全局最优的状态，使整个遗传算法的效率降低，为了打破这种平衡态跳出局部最优状态故而提出了双种群的改进方案，即两个种群同时进行图 4的流程，然后从完成迭代的两个种群中选取随机数个最优个体打破平衡态。这也可以进一步提高遗传算法的随机性，提高遗传算法的迭代效率。首先，该改进算法同时定义了两个种群（即父种群和子种群），又像上面两种改进一样，双种群遗传算法在初始化函数，hybridization 函数以及 main 函数中全部进行两个种群的遗传操作，详情见 5。

Listing 5: 双种群遗传算法的 Python 实现.

```
1  # 种群大小
2  population_size = 8
3  # 父种群的编码列表
4  parent1 = []
5  parent2 = []
6  # 子种群的编码列表
7  children1 = []
8  children2 = []
9  # 父种群每个个体的适应度
10 parent_fitness1 = []
11 parent_fitness2 = []
12 # 子种群每个个体的适应度
13 children_fitness1 = []
14 children_fitness2 = []
15
16
```

```
17 # 初始化个体
18 def initial_individual():
19 
20     # 个体的编码
21     individual = []
22     # 8个编码
23     for i in  range(8):
24         a = random.randint(0, 7)
25         individual.append(a)
26     # 计算生成的个体的适应度
27     fit_score = update_fitness_score(individual)
28     # 加入到种群中
29 
30     parent_fitness1.append(fit_score)
31     parent1.append(individual)
32 
33     # 个体的编码
34     individual = []
35     # 8个编码
36     for i in  range(8):
37         a = random.randint(0, 7)
38         individual.append(a)
39     # 计算生成的个体的适应度
40     fit_score = update_fitness_score(individual)
41     # 加入到种群中
42 
43     parent_fitness2.append(fit_score)
44     parent2.append(individual)
45     return
46 
47 
48 # 交叉产生后代
49 def hybridization():
50     # 选择两个父本
51     first = select1()
52     second = select1()
53     selected_parents = copy.deepcopy([parent1[first], parent1[second]])
54     # 交换从pos1到pos2的基因
55     pos1 = random.randint(0, 6)
56     pos2 = random.randint(0, 6)
57     # 保证pos1 <= pos2
58     if pos1 > pos2:
```

```
59          pos1, pos2 = pos2, pos1
60      # 交叉
61      tmp = selected_parents[0][pos1:pos2]
62      selected_parents[0][pos1:pos2] = selected_parents[1][pos1:pos2]
63      selected_parents[1][pos1:pos2] = tmp
64      # 一定的概率发生变异,假设概率为0.5
65      may = random.random()
66      if may > 0.5:
67          selected_parents[0] = mutation(selected_parents[0])
68      may = random.random()
69      if may > 0.5:
70          selected_parents[1] = mutation(selected_parents[1])
71      # 更新适应度
72      first_fit = update_fitness_score(selected_parents[0])
73      second_fit = update_fitness_score(selected_parents[1])
74
75      # 加入到子代中
76      children1.append(selected_parents[0])
77      children1.append(selected_parents[1])
78      children_fitness1.append(first_fit)
79      children_fitness1.append(second_fit)
80
81      # 选择两个父本
82      first = select2()
83      second = select2()
84      selected_parents = copy.deepcopy([parent2[first], parent2[second]])
85      # 交换从pos1到pos2的基因
86      pos1 = random.randint(0, 6)
87      pos2 = random.randint(0, 6)
88      # 保证pos1 <= pos2
89      if pos1 > pos2:
90          pos1, pos2 = pos2, pos1
91      # 交叉
92      tmp = selected_parents[0][pos1:pos2]
93      selected_parents[0][pos1:pos2] = selected_parents[1][pos1:pos2]
94      selected_parents[1][pos1:pos2] = tmp
95      # 一定的概率发生变异,假设概率为0.5
96      may = random.random()
97      if may > 0.5:
98          selected_parents[0] = mutation(selected_parents[0])
99      may = random.random()
100     if may > 0.5:
```

```
101             selected_parents[1] = mutation(selected_parents[1])
102         # 更新适应度
103         first_fit = update_fitness_score(selected_parents[0])
104         second_fit = update_fitness_score(selected_parents[1])
105 
106         # 加入到子代中
107         children2.append(selected_parents[0])
108         children2.append(selected_parents[1])
109         children_fitness2.append(first_fit)
110         children_fitness2.append(second_fit)
111         return
112 
113 
114 while t < 100:
115         # 父种群的编码列表
116         parent1 = []
117         parent2 = []
118         # 子种群的编码列表
119         children1 = []
120         children2 = []
121         # 父种群每个个体的适应度
122         parent_fitness1 = []
123         parent_fitness2 = []
124         # 子种群每个个体的适应度
125         children_fitness1 = []
126         children_fitness2 = []
127         # 初始化个体
128         t = t + 1
129         cc.append(main())
```

双种群遗传算法完整的实验代码见 https://paste.ubuntu.com/p/QPvVR4KRPJ/，实验结果见https://paste.ubuntu.com/p/D6h9f4yXWC/。双种群的实验结果显示求解的最大迭代代数为 30640 代，局部收敛的状态套更好一点，且高迭代代数出现的频率很低，其解的最小迭代数仅为 33 代，平均迭代数不到 5000。显著优于自适应算法和双倍体遗传算法。

2.3.4 三种改进遗传算法解决八皇后问题的实验对比和结论

代码运行过程中可以很明显的看到，普通遗传算法找到第一个解的时间通常在 5 秒左右，三种改进遗传算法找到第一个解的速度基本在 3 秒以

内，虽然遗传代数要偏大一些，但是效率要比普通遗传算法好很多，而根据实验结果来看，三种改进遗传算法中效果最好的是双种群遗传算法，双倍体遗传算法次之，最后是自适应遗传算法。

3 机器学习基本理论与方法

3.1 机器学习的基本概念及其学习策略

机器学习也叫统计学习（Statistics Learning），就是利用现有数据或者知识并去寻找一些适合他的模型对其进行描述和表示，并利用合适的模型对事物的发展和变化做一些合理的预测和分析，从而优化事务处理流程，提高决策的效率。

1. 机器学习的三要素是：

   模型（Model） 模型就是数据里抽象出来的，那些刚好可以被拿来描述客观世界的数学特征或统计学框架。

   策略（Strategy） 通常在模型行为阶段用来评价模型的好坏，也可以称做是模型的比较标准和选择标准。

   算法（Algorithm） 有了数据和模型以及策略，就会在实际执行的过程中注重效率问题，这时候就需要优化模型和策略了，算法就是用来解决效率问题的优化方法。

2. 统计学习基本学习策略：

   获取一个有限合理的训练数据集；

   确定学习模型的集合（包含所有可能模型的假设空间）；

   确定学习策略（即选择准则）；

   实现学习算法；

   通过学习方法选择最优模型；

   利用学习到的最优化框架对新数据进行分析和预测。

3.2 人工神经网络及其应用

3.2.1 人工神经网络的结构设计

人工神经网络由 input（输入），nerve cell(人工神经元)，weights(权值)，transfer function(传递函数)，activation function(激活函数)，以及 threshold(阈值)。一个人工神经系统的结构见图 7。其中核心是神经元，每个神经

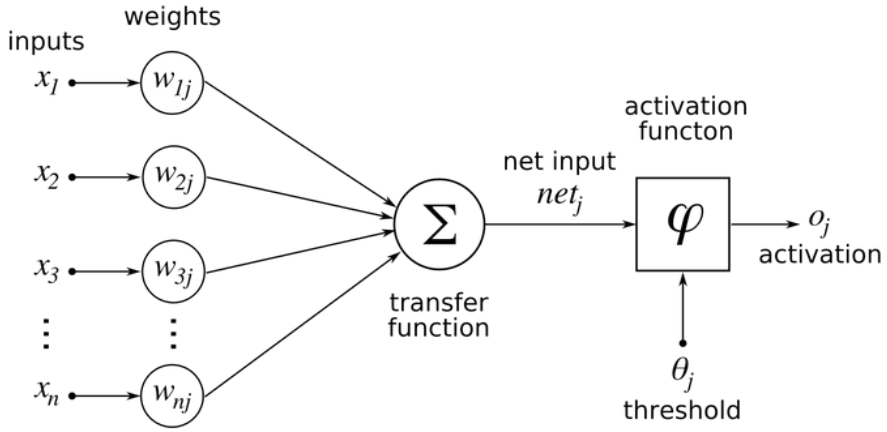

图 7: 人工神经系统的结构.

元 input = sum（其他神经元的 input x weight），并借助激活函数进行处理后将结果再传递给后面的神经元，直到输出。这虽然看起来像是一系列简单的数学运算，但是如果将有限多层神经元堆叠起来，你就会获得一个能处理复杂任务的人工神经网络。比如语音识别、图像识别分类等问题。

卷积神经网络、循环神经网络、玻尔兹曼机是三个常见的神经网络，下面我们用表格的形式来介绍他们各自的特点和应用领域，见表 5。玻尔兹曼机是一种无监督深度学习模型，也是随机神经网络的一种。玻尔兹曼机也是一种特殊形式的对数线性的马尔科夫随机场（Markov Random Field，MRF），即能量函数是自由变量的线性函数。通过引入隐含单元，我们可以提升模型的表达能力，表示非常复杂的概率分布[10]。他的特点如下：

1. 玻尔兹曼机的特点：

   学习阶段 其权值设置或改变的算法是不确定的，通过某种概率分布不断变化调整。即每训练一次就可能改变一次。

表 5: 卷积神经网络和循环神经网络的特点

名称	卷积神经网络
特点	核心层是 Convolution + Pooling + FullConnected 同时实现特征提取和分类 能根据局部获得全局信息
应用领域	图像分类，目标检测跟踪，语义分割等
名称	循环神经网络
特点	可以根据借助时间序列的变化建模 能有效处理序列特性的数据
应用领域	视频分析、机器翻译、语音识别等

运行阶段 状态的改变也不是按照非 0 即 1 这样的确定性方式，同样按照某种概率分布（状态改变取 1 或 0 的概率）来进行状态的转移。

玻尔兹曼机中的受限玻尔兹曼机应用较多，主要应用是权重的初始化（让权重更有效的模拟数据）以及模型降维、分类和协同过滤，特征学习与主题建模等。

3.2.2 人工神经网络的学习算法

用于人工神经网络的学习算法有很多，Deeplearning Algorithms Tutorial（深度学习算法教程）一书将所有的人工智能算法应用总结如下[11]：

- 自动编码器 (Autoencoder)
- 反向传播 (Backpropagation)
- 递归神经网络 (Recurrent Neural Network)
- 多层感知器 (Multilayer Perceptron)
- 玻尔兹曼机 (Boltzmann Machine)

- 卷积神经网络 (Convolutional Neural Network)
- Hopfield 网络 (Hopfield Network)
- 径向基函数网络 (Radial Basis Function Network)
- 受限玻尔兹曼机 (Restricted Boltzmann Machine)
- 自组织映射 (Self-Organizing Map)
- 尖峰神经网络 (Spiking Neural Network)

本文挑选比较常用的自动编码器算法、反向传播算法和受限玻尔兹曼机，并借助这些算法的相关应用文献来简要介绍。

3.2.3 人工神经网络在与预测和图像方面的应用

自动编码器（下文统称 Autoencoder）其实是一种无监督的学习算法，用途主要在数据的降维和特征提取。Autoencoder 可以很好的学习 input 中的元素特征，且通过 Autoencoder 编码的数据在完成解码的同时较好的保留了原始数据信息。Jiatong Shen 等[12]使用一种基于深度神经网络的 Autoencoder 建立了一个信用卡欺诈预警系统，相比于传统的预警系统，该混合系统的 AUC（Area Under the Curve）值提高到了 0.9577，被证明具有更好的准确性且稳定性更佳。

反向传播算法也称“误差反向传播”，可以计算网络中每个权重的损失函数梯度，，并反馈最优化方法用以不断更新权值的最小化损失函数。S. Prashant Mahasagara 和 Andry Alamsyah 等[13] 基于印度尼西亚基建和消费股股票价格历史走势，并借助人工神经网络的反向传播算法生成了一套相关行业股票价格的预测系统且可以以更小的误差率生成股票投资组合。实验结果证明，在当前市场投资风险较低、股票市场波动较小时，使用 ANN 反向传播的预测系统要优于实际的真实投资。

上一个小章节我们介绍过玻尔兹曼机，这次我们要介绍的受限玻尔兹曼机是玻尔兹曼机的一种特殊拓扑结构，实质是一种概率图模型。受限是指限定模型为二分图，模型包含两部分，即可见单元和隐单元，对应输入参数和训练结果。为了比一般的玻尔兹曼机的训练算法更高效，二分图各边

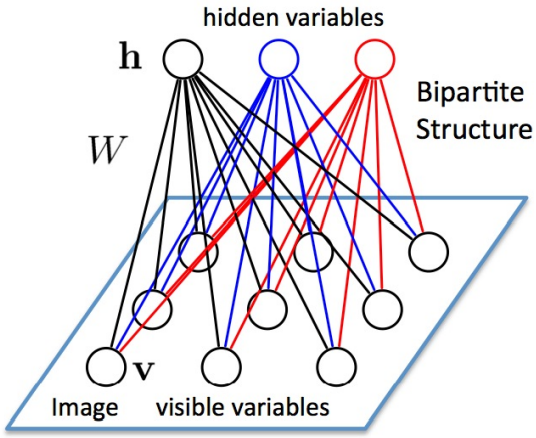

图 8: 受限玻尔兹曼机

均连接可见单元与隐单元。众所周知受限玻尔兹曼机（下统称 RBM）的计算密度或强度较大，目前也有很多更快的训练算法被发展，如 GPU/CPU 加速算法、硬件改进以及神经元友好型加速算法，这使得大型 RBMs 在未来的应用变得可能。

图像领域传统的字典学习算法收敛速度缓慢、训练稀疏表示的如图 9效果学习效果差、如果被噪声干扰学习效果会更糟糕。为了解决这个问题,Liu

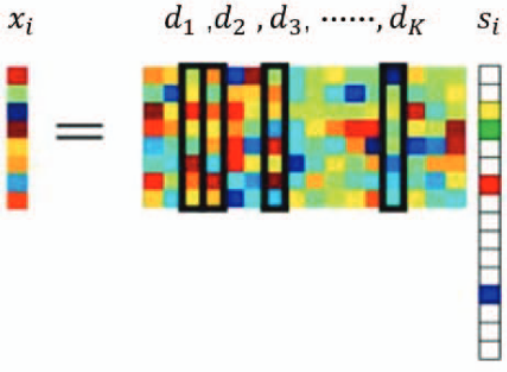

图 9: 图像稀疏表示图

Lian 等[14]提出了一种基于受限玻耳兹曼机的字典学习算法。字典学习的神经网络结构见图 10。通过采用分段的训练方法训练稀疏表示的字典，充分

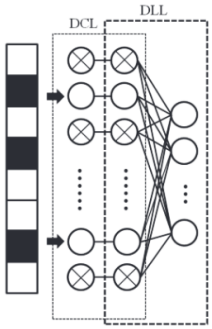

图 10: 字典学习神经网络结构

学习多样性图像的稀疏表示，得到了 RBM 隐藏层神经元的概率分布。神经

Sampling rates	PSNR/dB			
	DCT	KSVD	BPFA	TSDL
0.3	24.58	24.78	25.28	25.86
0.4	25.93	26.24	26.49	26.55
0.5	26.50	27.39	27..65	28.07
0.6	28.55	29.04	29.57	29.76
0.7	30.87	31.04	31.15	31.16
0.8	32.22	32.67	33.43	33.62

图 11: 不同采样率重构图片的 PSNR

网络的权值训练采用最优字典矩阵。测试结果见图 11，利用该算法重构的图片的 PSNR 随着采样率的下降而下降，图片清晰度也显著提高，见图 12。实验结果表明, 该算法能够提高去噪的效果和提高图片测试的精度。

4 总结

本文分为三大块，分别是知识表示与推理、搜索与问题求解以及机器学习的理论和方法。

Figure 6. Visible effect of reconstructed image: (a) DCT (b) KSVD(c) BPFA (d) TSDL

图 12: 重构图像的清晰度对比图

4.1 本文重点内容总结

其中重点内容有：

- 确定性推理之启发式搜索算法在危险天气下航班路径规划的应用。
- 不确定推理之主观贝叶斯方法在电熔镁炉熔炼过程异常工况识别的应用。
- 遗传算法在八皇后问题中的具体实现以及实验设计。
- 几种改进的遗传算法在八皇后问题中的具体实现以及实验设计。
- 介绍了了人工神经网络的基础架构和常用的几种学习算法。
- 介绍了人工神经网络自动编码器算法在信用卡欺诈预警系统的应用。
- 介绍了人工神经网络反向传播算法在印尼相关行业股票投资预测方面的应用。
- 介绍了基于人工神经网络受限玻尔兹曼机的新型字典学习算法在稀疏图像领域的应用。

4.2 结论、收获和致谢

结论和收获为：

- 人工智能是本专业很多研究课题的基础，用心学好这门课可以了解到很多深度学习与机器学习的算法常识。

- 从完全不了解人工智能的知识到对基础常用的人工智能算法和模型有了初步的认识。
- 从之前只学习过简单直接的经典算法解决问题到现在开始体会到人工智能解决问题的独特方式。
- 学习了遗传算法的实现及其使用，以及其在在八皇后问题的具体实现，体会到该算法仿生物学的特性的巧妙。
- 学到了文献检索与分类，学习了 Zotero 文献管理软件的使用，能更快的处理文章编辑过程中的文献插入问题。
- 学会了阅读英文文献的方法流程，磨练了自己的耐心。

最后，非常感谢赵晖老师半年来的授课和指导、同学们的陪伴，令我受益匪浅！

参考文献

[1] NARKAWICZ A, HAGEN G E. Algorithms for Collision Detection Between a Point and a Moving Polygon, with Applications to Aircraft Weather Avoidance[C/OL]//16th AIAA Aviation Technology, Integration, and Operations Conference. Washington, D.C.: American Institute of Aeronautics and Astronautics, 2016. DOI: 10.2514/6.2016-3598.

[2] HE L, ZHAO A, WANG X, et al. Path Planning Method for General Aviation under Hazardous Weather Using Heuristic Algorithm[C/OL]//2019 5th International Conference on Transportation Information and Safety (ICTIS). 2019: 920-926. DOI: 10.1109/ICTIS.2019.8883714.

[3] GRECHE L, JAZOULI M, Es-Sbai N, et al. Comparison between Euclidean and Manhattan distance measure for facial expressions classification[C/OL]//2017 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS). Fez, Morocco: IEEE, 2017: 1-4. DOI: 10.1109/WITS.2017.7934618.

[4] GÖNEN M, JOHNSON W O, LU Y, et al. Comparing Objective and Subjective Bayes Factors for the Two-Sample Comparison: The Classification Theorem in Action[J/OL]. The American Statistician, 2019, 73 (1): 22-31. DOI: 10.1080/00031305.2017.1322142.

[5] UZUNOĞLU B. An Adaptive Bayesian Approach With Subjective Logic Reliability Networks for Preventive Maintenance[J/OL]. IEEE Transactions on Reliability, 2020, 69(3): 916-924. DOI: 10.1109/TR.2019.2916722.

[6] 袁杰, 王姝, 王福利, 等. 基于改进主观贝叶斯方法识别电熔镁炉异常工况[J]. 东北大学学报 (自然科学版), 2021, 42(2): 153-159.

[7] PANDEY H M, DIXIT A, MEHROTRA D. Genetic algorithms: Concepts, issues and a case study of grammar induction[C/OL]// Proceedings of the CUBE International Information Technology Conference on - CUBE '12. Pune, India: ACM Press, 2012: 263. DOI: 10.1145/2381716.2381766.

[8] Eight queens puzzle | Kaggle[Z].

[9] 王万良. 人工智能导论. 第 3 版[M]. 人工智能导论. 第 3 版, 2011.

[10] 人工神经网络 (Artificial Neural Network) - 玻尔兹曼机 (Boltzmann Machine) - 《Deeplearning Algorithms Tutorial（深度学习算法教程）》 - 书栈网· BookStack[Z].

[11] Deeplearning Algorithms Tutorial - 《Deeplearning Algorithms Tutorial（深度学习算法教程）》 - 书栈网· BookStack[Z].

[12] SHEN J. Credit Card Fraud Detection Using Autoencoder-Based Deep Neural Networks[C/OL]//2021 IEEE 2nd International Conference on Big Data, Artificial Intelligence and Internet of Things Engineering (ICBAIE). 2021: 673-677. DOI: 10.1109/ICBAIE52039.2021.9389940.

[13] MAHASAGARA S P, ALAMSYAH A, RIKUMAHU B. Indonesia infrastructure and consumer stock portfolio prediction using artificial neural network backpropagation[C/OL]//2017 5th International Conference on Information and Communication Technology (ICoIC7). 2017: 1-4. DOI: 10.1109/ICoICT.2017.8074710.

[14] LIAN L. Dictionary Learning Algorithm based on Restricted Boltzmann Machine[C/OL]//2021 8th International Conference on Dependable Systems and Their Applications (DSA). 2021: 560-565. DOI: 10.1109/DSA52907.2021.00082.