浙江工业大学

人工智能及其应用实验报告

(2021 级)

实验三：遗传算法实验

学生姓名：温家伟

学生学号：202103151422

学科专业：大数据分析 2101 班

所在学院：理学院

提交日期：2024 年 1 月 2 日

目录

1 实验目的

熟悉和掌握遗传算法的原理、流程和编码策略，并利用遗传求解函数优化问题，理解求解流程并测试主要参数对结果的影响。

2 实验原理

遗传算法的基本思想正是基于模仿生物界遗传学的遗传过程。它把问题的参数用基因代表，把问题的解用染色体代表（在计算机里用二进制码表示），从而得到一个由具有不同染色体的个体组成的群体。这个群体在问题特定的环境里生存竞争，适者有最好的机会生存和产生后代。后代随机化地继承了父代的最好特征，并也在生存环境的控制支配下继续这一过程. 群体的染色体都将逐渐适应环境，不断进化，最后收敛到一族最适应环境的类似个体，即得到问题最优的解。

3 实验条件

python+Linux

4 实验内容

用遗传算法求解下面一个 Rastrigin 函数的最小值，设定求解精度到 10 位小数。

$f(x_1, x_2) = 20 + x_1^2 + x_2^2 - 10(cos2\pi x_1 + cos2\pi x_2) \qquad -5 \leq x_i \leq 5 \quad i = 1, 2$

以下给出了最小化的适应度函数。

$$f(x_1, x_2) = 20 + x_1^2 + x_2^2 - 10(cos2\pi x_1 + cos2\pi x_2)$$

表 1: 参数设置

编码	编码方式	实数编码
种群参数	种群规模	**100**
	初始种群的个体取值范围	**[0,10]**
选择操作	个体选择方法	轮盘赌选择
最佳个体保存	优良个体保存方法	每代只存最优的一个
交叉操作	交叉概率	**1**
	交叉方式	单点交叉
变异操作	变异方式	均匀变异
停止参数	最大迭代步数	**100**
	停止阈值	适应度值是否有效

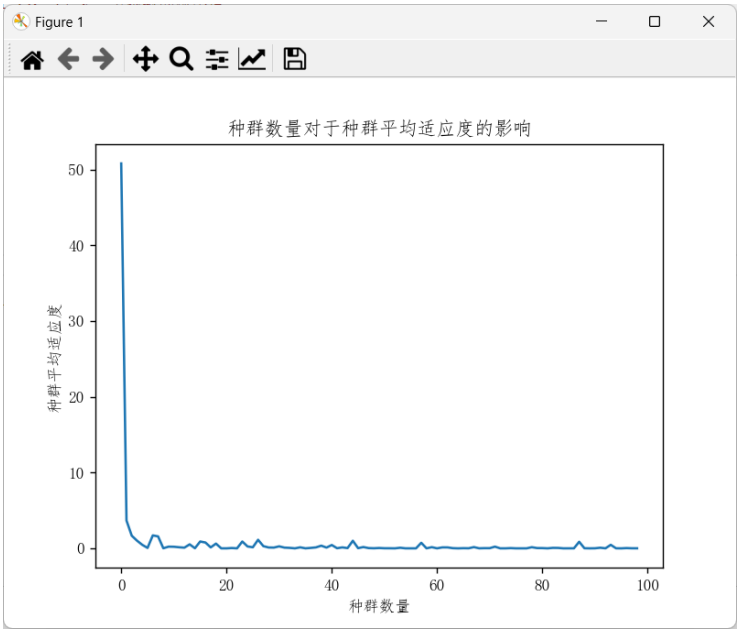

图 1: 种群数量

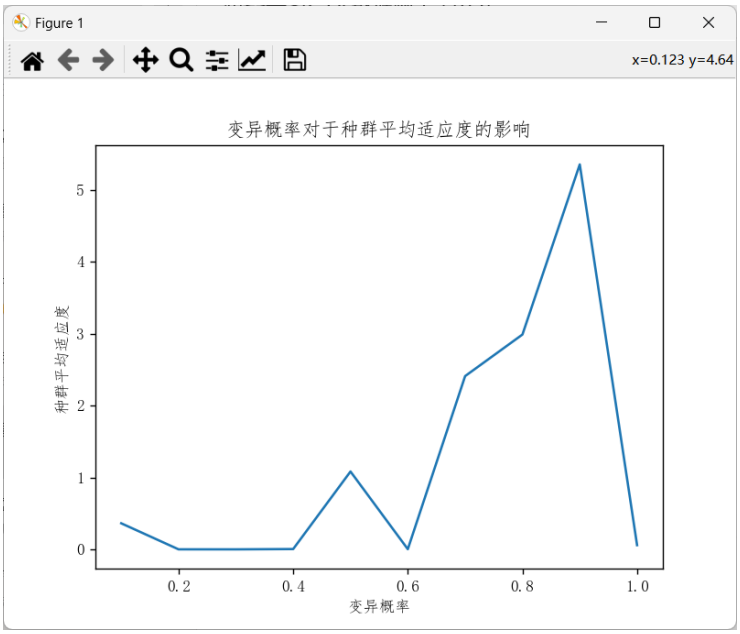
Figure 1
x=0.123 y=4.64
变异概率对于种群平均适应度的影响
种群平均适应度
变异概率
0
1
2
3
4
5
0.2
0.4
0.6
0.8
1.0

图 2: 变异概率

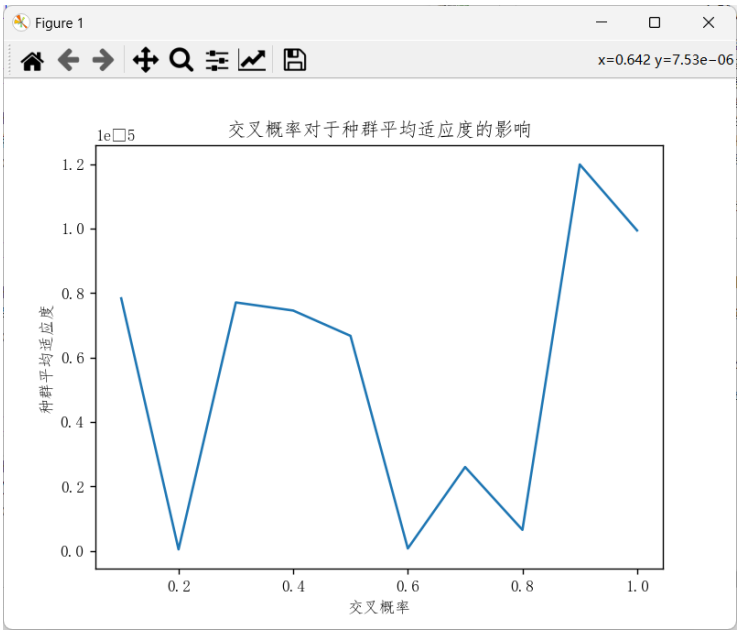

图 3: 交叉概率

5 附录

```
# """
#
    设计高效遗传算法的实验中，求解的问题是在-5<=x1,x2<=5上求一下函数的极
# y = 20 + x1^2 + x2^2 - 10*(cos 2PIx1 + cos
    2PIx2 )
# """
# import matplotlib.pyplot as plt

# from problem1_GA import problem1_GA

# plt.rcParams["font.family"] = "FangSong"
# SCORE_ZQ = []
# for ZQ in range(1, 100):
#     test = problem1_GA(1, 1, ZQ)
```

```
13      #     test.initpopulation()
14      #     for i in range(50):
15      #         test.next_generation()
16      #         generation, answer, score =
            test.get_what_we_need()
17      #         if i == 99:
18      #             print("x = " + str(answer[0])+", y
            = "+str(answer[1]))
19      #         pass
20
21      #     x1, x2, x3 = test.draw()
22      #     SCORE_ZQ.append(score)
23
24      # plt.plot(range(99), SCORE_ZQ)
25      # plt.xlabel("种群数量")
26      # plt.ylabel("种群平均适应度")
27      # plt.title("种群数量对于种群平均适应度的影响")
28      # plt.show()
29
30      # """
31      #
            设计高效遗传算法的实验中，求解的问题是在-5<=x1,x2<=5上求一下函数的极
32      # y = 20 + x1^2 + x2^2 - 10*(cos 2PIx1 + cos
            2PIx2 )
33      # """
34      # import matplotlib.pyplot as plt
35
36      # from problem1_GA import problem1_GA
37
38      # plt.rcParams["font.family"] = "FangSong"
39      # SCORE_ZQ = []
40      # p_list = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
            0.8, 0.9, 1]
```

```
41      # for p in p_list:
42      #     test = problem1_GA(1, p, 100)
43      #     test.initpopulation()
44      #     for i in range(50):
45      #         test.next_generation()
46      #         generation, answer, score =
           test.get_what_we_need()
47      #         if i == 99:
48      #             print("x = " + str(answer[0])+", y
           = "+str(answer[1]))
49      #         pass
50
51      #     x1, x2, x3 = test.draw()
52      #     SCORE_ZQ.append(score)
53
54
55      # plt.plot(p_list, SCORE_ZQ)
56      # plt.xlabel("变异概率")
57      # plt.ylabel("种群平均适应度")
58      # plt.title("变异概率对于种群平均适应度的影响")
59      # plt.show()
60
61
62      """
63      设计高效遗传算法的实验中，求解的问题是在-5<=x1,x2<=5上求一下函数的极小值
64      y = 20 + x1^2 + x2^2 - 10*(cos 2PIx1 + cos 2PIx2 )
65      """
66      import matplotlib.pyplot as plt
67
68      from problem1_GA import problem1_GA
69
70      plt.rcParams["font.family"] = "FangSong"
71      SCORE_ZQ = []
```

```
72      p_list = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
            0.9, 1]
73      for p in p_list:
74          test = problem1_GA(p, 1, 100)
75          test.initpopulation()
76          for i in range(1000):
77              test.next_generation()
78              generation, answer, score =
                    test.get_what_we_need()
79
80              pass
81
82          x1, x2, x3 = test.draw()
83          SCORE_ZQ.append(score)
84
85
86      plt.plot(p_list, SCORE_ZQ)
87      plt.xlabel("交叉概率")
88      plt.ylabel("种群平均适应度")
89      plt.title("交叉概率对于种群平均适应度的影响")
90      plt.show()
91
92
93  class problem1_individual(object):
94      def __init__(self, n1, n2):
95          self.gene = [n1, n2]
96          self.score = 0
97      pass
98
99
100     import problem1_individual
101     import math
102     import random
```

```
103
104
105     class problem1_GA:
106         def __init__(self, crossrate, variationrate,
                size):
107             """
108             :param crossrate: 基因交叉发生的概率
109             :param variationrate: 基因变异发生的概率
110             :param cities: 输入的城市的矩阵
111             1 crossrate: 单点交叉概率
112             2 variationrate: 突变概率
113             3 cities: 城市的权重矩阵
114             4 size: 需要计算的种群大小
115             5 generation: 迭代次数
116             6 city_num: 计算的城市数量
117             7 best: 种群最好的个体
118             8 list: 种群中所有的个体
119             """
120             self.crossrate = crossrate
121             self.variationrate = variationrate
122             self.size = size
123             self.generation = 1
124             self.gene_num = 2
125             self.best = None
126             self.list = []
127             pass
128
129         def initpopulation(self):
130             # 初始化一个种群大小的函数
131             for i in range(self.size):
132                 n1 = random.uniform(-5, 5)
133                 n2 = random.uniform(-5, 5)
```

```
134                 life =
                        problem1_individual.problem1_individual(n1,
                        n2)
135                 self.list.append(life)
136                 pass
137             pass
138
139         # 个体数值计算公式
140         def function(self, individual):
141             score = 20+pow(individual.gene[0],
                    2)+pow(individual.gene[1],
                    2)-10*(math.cos(2*math.pi*individual.gene[0])+math.cos(2*math
142             return score
143
144         def caculate(self):
145             self.best = self.list[0]
146             for i in range(self.size):
147                 self.list[i].score =
                        self.function(self.list[i])
148                 if self.best.score >
                        self.list[i].score:
149                     self.best = self.list[i]
150             pass
151
152         def roulette_wheel(self):
153             """
154             轮盘赌普通方法得到被选中的基因型
155             :param individuals:
                    种群中各个基因类型的数量
156             :return: 返回被选中的基因型的代号
157             """
158             all_individual = 0
159             for i in range(len(self.list)):
```

```
160                 if (self.list[i].score):
161                     all_individual += 1 /
                            self.list[i].score
162             probabilities = []
163             for i in range(len(self.list)):
164                 if (self.list[i].score):
165                     probabilities.append((1 /
                            self.list[i].score) /
                            all_individual)
166             selected_individual = random.uniform(0, 1)
167             now_individual = 0.0
168             for ind, val in enumerate(probabilities):
169                 now_individual += val
170                 if now_individual >
                        selected_individual:
171                     return self.list[ind]
172
173         def crossgene(self, parent1, parent2, index1,
                index2):
174             child = [0, 0]
175             child[0] = parent1.gene[0] * index1 +
                    parent2.gene[0] * index2
176             child[1] = parent1.gene[1] * index1 +
                    parent2.gene[1] * index2
177             return child
178
179         def crossover(self, father, mother):
180             """
181             :param father: 需要进行遗传的父类个体
182             :param mother: 需要进行遗传的母类个体
183             :return:
184             """
185             x = random.randint(0, 9999)
```

```
186             #
                    变异有概率，大于某个值就不发生，小于某个值就发生变异
187             k = 10000 * self.variationrate
188             if x < k:
189                 # 进行单点交换
190                 index1 = random.uniform(0, 1)
191                 index2 = 1 - index1
192                 child1 = self.crossgene(father,
                        mother, index1, index2)
193                 child2 = self.crossgene(mother,
                        father, index1, index2)
194             else:
195                 child1 = father.gene
196                 child2 = mother.gene
197             return child1, child2
198
199         def variation(self, individual):
200             """
201             :param individual:
                    需要判断是否进行变异的个体的基因
202             :return: 变异或者没有变异以后的个体
203             """
204             x = random.randint(0, 9999)
205             #
                    变异有概率，大于某个值就不发生，小于某个值就发生变异
206             k = 10000 * self.variationrate
207             if x < k:
208                 index = random.randint(0, 1)
209                 individual[index] =
                        random.uniform(-5, 5)
210             return individual
211
212         # 进行繁衍，得到孩子
```

```
213         def get_children(self):
214             father = self.roulette_wheel()
215             mother = self.roulette_wheel()
216             child1, child2 = self.crossover(father,
                    mother)
217             self.variation(child1)
218             self.variation(child2)
219             return
                    problem1_individual.problem1_individual(child1[0],
                    child1[1]), \
220                     problem1_individual.problem1_individual(child2[0],
                            child2[1])
221
222         # 得到新一代种群
223         def next_generation(self):
224             new_list = []
225             self.caculate()
226             new_list.append(self.best)  #
                    最好的个体一定要,虽说回交有点不太好......
227             while len(new_list) < len(self.list):
228                 individual1, individual2 =
                        self.get_children()
229                 new_list.append(individual1)
230                 new_list.append(individual2)
231             self.list = new_list
232             self.generation += 1
233             pass
234
235         def get_what_we_need(self):
236             return self.generation, self.best.gene,
                    self.best.score
237
238         def draw(self):
```

```
239                 return self.size, self.crossrate,
                        self.variationrate
240             pass
```