遗传算法中交叉和变异算法的比较研究

计算机科学与技术 周顺 401030918069

**1 代码介绍**

我要写的代码是在标准的遗传算法的基础上，研究不同的交叉算 法和变异算法，比较不同算法得到的目标函数的最优值所需要的 迭代次数，收敛效果等。从而找到最优的交叉算法和变异算法，使得遗传算法符合自然选择的适者生存的法则。

**2 代码运行环境配置说明**

**2.1 python安装**

**2.1.1 Unix和Linux平台安装python：**

大部分Unix和Linux系统都自带Python，只要在命令行输入python，就可以进入Python的编辑环境，输入exit（）就可以退出编辑环境，返回命令行。

若没有python，就只有自己装了，方法如下：

1 打开“https://www.python.org/downloads/source/”链接。

2 下载适合的源码压缩包，将其下载解压。

3 将路径转到解压的文件夹 执行：cd 文件夹路径

4 之后执行 ./configure

Make

Make install

Ok,安装完成。

**2.1.2 window平台下安装Python**

打开“https://www.python.org/downloads/windows/”链接。直接下载你需要的Python版本，解压安装到电脑上即可。

**2.2运行环境的配置**：

**2.2.1 Unix和linux设置环境变量：**

在命令行上输入：

exprot PATH=”$PATH:/usr/local/bin/python

window设置环境变量：

在命令提示框输入cmd，然后输入：

Path=%path%;D:\python

其中/usr/local/bin/python和D:\python都是Python的安装目录。

**2.2.2 安装matplotlib模块**

安装完Python后，我们还要下载一个模块matplotlib。

打开https://pypi.org/project/matplotlib/ 链接，下载该模块安装即可。

在Python运行环境输入import matplotlib，运行没有报错，即安装成功。

**3 详细代码说明**

**3.1 代码的基本结构：**

**1首先导入基本的模块，**

**2之后创建一个类GA，设置类中的初始值。**

**3在GA类中构造函数species\_origin、translation、 function、fitness、crossover、mutation、main等函数。**

**4设置参数值，利用类GA创建对象，调用里面的主函数，得到运行结果**

**3.2 源代码及注释**

1. # -\*-coding:utf-8 -\*-
2. # 文件名：genetic.py
3. # 文件编号：NCU-18-0130
4. # 版本号：1.3
5. # 作者：周顺
6. # 创建时间：2018-11-22
7. # 文件描述：目标求解2\*cos(x)+sinx的最大值
8. #
9. #
10. #----------------------------------------------------
11. #版本号：random 2 1.0.1
12. #作者：PSF
13. #维护人员：Srichter
14. #时间：2013.4.15
15. #random是随机数的moudle，常用操作有randint(begin,end),
16. #其中begin是开始的整数，end是结束的整数#如random.randint(0,1)
17. #就是随机取0和1中的一个数字.#还有random.random()随机产生0到1的
18. #浮点数#random.uniform(begin,end),随机产生begin到end的浮点数。
19. **import** random
20. #版本号：math123 0.4
21. #作者：Jaga123
22. #维护人员：Jagamtsl
23. #时间：2017.6.22
24. #math是数学公式的moudle，使用这个模块可以调用很多数学上复杂的函数，
25. #如：math.cos(),math.tan(),math.e"e=2.718 是一个常量"#math.factorrial(x)"
26. #这是求x的阶乘"，等等。你可以使用很多数学符号。
27. **import** math
28. #版本号：matplotlib 3.0.2
29. #作者：John D.Hunter,Michael Droettboom
30. #维护人员：cmoad，jdh2368,matthew.brett,mdboom2,Thomas.Caswell
31. #时间：2018.11.11
32. #matplotlib是一个可视化的模块,可以将数据以图表、曲线等方式可显现出来。
33. #这里重点介绍一下plt，plt.plot(x,y),其中x和y可以#是一个列表，也可以
34. #是一个数组，plt.plot(x,y)是以x和y相对应的点组成坐标，然后绘制曲线
35. #如plt.plot([1,2,3,4],[1,2,3,4]),绘制就就是y=x这条线段（0<x，y<4).
36. #plt.show()可以将曲线可视化。
37. **import** matplotlib.pyplot as plt
39. #构造一个GA的类。这里构造的方法和Python2版本不一样，
40. #Python2 “class GA：”就可以了，而Python3要加上object，即构造object的
41. #子类。这样统一了类和实例的类型，都为“type”。
42. **class** GA(object):
44. #构造一个\_\_init\_\_ 设置初值的函数，可以内部赋值，也可以外部赋值。
45. #生成chromosome\_length大小的population\_size个个体的种群
47. **def** \_\_init\_\_(self,population\_size,chromosome\_length,max\_value,pc,pm):
49. self.population\_size=population\_size#种群的数量
50. self.choromosome\_length=chromosome\_length#每个个体基因的长度]
51. self.max\_value=max\_value#添加一个常量，之后会用到
52. self.pc=pc#发生基因重组的概率
53. self.pm=pm#发生基因变异的概率

56. # 构造一个species\_origin的初始化种群的函数，给这个种群的每一个个体赋值随机的
57. # 基因，即种群的编码。编码方式有很多种：浮点数编码、格雷编码、二进制编码等等。
58. # 这里我们用二进制编码，二进制是由0和1组成的二进制作为个体的基因对。这里举个例子
59. # 比如：10110101011010101111，这样一串代码即表示为一个个体的基因对。
60. # 执行这函数，将会生成一个随机的种群。返回的这个种群是一个二维列表，第一维是个体
61. # ，第二维是该个体的基因。
62. **def** species\_origin(self):
63. population=[[]]#构造一个二维列表
64. **for** i **in** range(self.population\_size):#生成想要的数量的个体，数量为self.population\_size
66. temporary=[]#构造一个空的列表，作为基因对的暂存器
67. **for** j **in** range(self.choromosome\_length):#生成每一个个体的基因,
68. #每一个基因对的长度为self.choromosome\_length.
69. temporary.append(random.randint(0,1))
70. #随机产生一个基因对,由二进制数组成,append
71. #函数是将生成的0或者1加进temporary列表中
73. population.append(temporary)
74. #将一位列表temporary加进population二维列表中
75. #即将每个个体加进种群中
76. **return** population[1:]
77. # 将种群返回，种群是个二维列表，个体和基因对两维
79. #构造一个translation的从二进制转变成十进制的函数，使得可以对基因对进行处理。
81. **def** translation(self,population):
83. temporary=[]
84. **for** i **in** range(len(population)):#len是求长度，len(population)是得到种群的个数
85. total=0#暂存十进制的值
86. **for** j **in** range(self.choromosome\_length):
87. total+=population[i][j]\*(math.pow(2,j))
88. #math.pow(a,b)即求a的b次方的值。
89. #从第一个基因开始，每位对2求幂，再求和
90. # 如：1100 转成十进制为：0 \* 2(0) + 0 \* 2(0) + 1 \* 2(2) + 1 \* 2(3) =12
91. temporary.append(total)
92. #一个基因转变完成，由一个二进制数转变为一个十进制数
93. **return** temporary
94. # 返回种群中所有个体编码完成后的十进制数


98. #这里构造一个function函数，来模拟自然环境。
99. # 目标函数就相当于一个自然环境 对个体进行筛选，这里用2\*math.cos(x)+math.sin(x)作为目标函数
100. # 输入的是一个种群，输出的是这个种群每个个体对自然环境的产生的反应，也可以称为适应值，即函数的值。
101. **def** function(self,population):
102. temporary=[]#用来暂存基因对的十进制值
103. function1=[]#用来存储通过目标函数转换的值。
104. temporary=self.translation(population)#调用transslation函数，将二进制转换为十进制
105. **for** i **in** range(len(temporary)):
106. #这里的x取值可以取很多值，比如x=temporary[i]或者x=temporary[i]
107. #/(math.pow(2,self.chromosom\_length)-1),或者是下面的值。但是每
108. #一种取值的方式都会产生不同的优化效果，而且种群的进化的效果也不一样。
109. x=temporary[i]\*self.max\_value/(math.pow(2,self.choromosome\_length)-1)
110. #执行目标函数，将其结果加进function1列表中。
111. function1.append(2\*math.cos(x)+math.sin(x))
112. #返回function1.
113. **return** function1
115. #构造一个fitness函数，重新适应值进行改变，即为了模拟自然过程，为了使适应值都在>=0,
116. #所以当适应值为负数是，将其赋值为0.
117. #输入function1列表，输出改变后的适应值列表。
118. **def** fitness(self,function1):
119. #构造一个适应值列表
120. fitness\_value=[]
121. #得到function1列表的长度
122. num=len(function1)
123. #循环num次
124. **for** i **in** range(num):
125. #如果funtion[i]>0,将function1[i]的值加进fitness\_value函数中
126. #否则将0 加进fitness\_value函数中
127. **if**(function1[i]>0):
128. temporary=function1[i]
129. **else**:
130. temporary=0.0
132. fitness\_value.append(temporary)
133. #返回新的适应值列表
134. **return** fitness\_value
136. #构造一个sum函数，用来计算适应值的和。
137. #输入适应值列表，输出所有适应值的和。
139. **def** sum(self,fitness\_value):
140. total=0
142. **for** i **in** range(len(fitness\_value)):
143. total+=fitness\_value[i]
144. **return** total
146. #构造一个cumsum函数，计算适应值斐伯那契列表，从后面开始计算，
147. #如第n项的值赋值为前n项的值之和、第n-1项的值赋值为前n-1值之和。
148. #输入一个适应值列表，输出适应值对应的斐波那契列表。
149. **def** cumsum(self,fitness1):
150. #这里的range(begin,end,step),begin是开始的位置，end是
151. #结束的位置，step是步长。
152. #下面的循环是从后往前循环。
153. **for** i **in** range(len(fitness1)-1,-1,-1):
154. total=0#设置一个存储和的变量
155. j=0#循环的变量
157. **while**(j<=i):#while循环
158. total+=fitness1[j]
159. j+=1#j加一
160. #将和从新赋值给列表。
161. fitness1[i]=total
163. #构造一个selection函数，用来选择适应值较大的个体，首先将通过fitness
164. #函数的得到的适应值列表，将其每个个体的适应值概率化，也可以说是正则化。
165. #然后再通过轮盘赌的方法将适应值较大的个体选择出来。
166. #输入是种群和适应值列表，输出是新的适应值更大的种群
167. **def** selection(self,population,fitness\_value):
168. #构造一个列表，存储每个个体适应值占总适应值的比例。
169. new\_fitness=[]
170. #计算总的适应值
171. total\_fitness=self.sum(fitness\_value)
172. #通过这个循环计算每个个体所占比例
173. **for** i **in** range(len(fitness\_value)):
174. new\_fitness.append(fitness\_value[i]/total\_fitness)
175. #cumsum是计算列表的斐波那契列表。
176. self.cumsum(new\_fitness)
177. #用来存储存活的种群，
178. ms=[]
179. #计算种群的长度
180. population\_length=pop\_len=len(population)
182. #随机一连串0到1的数字，可能大也可能小，将其存储在ms数组中。
184. **for** i **in** range(pop\_len):
185. ms.append(random.random())
186. #将ms数组排序
187. ms.sort()
189. fitin=0#设置一个变量来记录new\_fitness的进度
190. newin=0#设置一个变量来记录ms的进度
191. new\_pop=population#复制种群
193. #轮盘赌方法是一种选取的方式。
194. #举个例子，我要从十个个体中选取适应值更大的个体延续下去，
195. #假如，每个个体适应值位1，0，1，4，1，3，1，0，1，0.
196. #它们对应的比例为：0.083，0，0.083，0.33，0.083，0.25，0.083，0，0.083，0.
197. #它们对于的斐波那契列表：0.083，0.083，0.166，0.496，0.579，0.829，0912，0.912，1，1.
198. #假设ms数组为：0.1，0.2，0.3，0.4，0.5，0.6，0.7，0.8，0.9，0.99.
199. #执行轮盘赌方法后，0.166>0.1,所以选择一次第三个个体，之后，0.496>0.4>0.3>0.2,所以选择3次第四个个体
200. #下面类似，以此可见，当适应值更高的个体，被选择的次数越多。
201. **while** newin<pop\_len:
202. **if**(ms[newin]<new\_fitness[fitin]):#斐波那契列表内数值大于ms数组内的数值。
203. new\_pop[newin]=population[fitin]#将个体保存在新种群内。
204. newin+=1#ms数组加1
205. **else**:
206. fitin+=1#列表加1
207. population=new\_pop#将旧种群替换成新种群
209. #4.构造一个crossover函数，用来实现基因重组。
210. #简单点说就是将一个基因对的一部分和另一个基因对的另一相同位置的部分交换。
211. #可以是单点，也可以是多点，在这个实验，我们比较一下双点和单点。
212. #输入一个种群，输出一个新的种群。
213. **def** crossover(self,population):
214. pop\_len=len(population)#种群的长度
216. **for** i **in** range(pop\_len-1):
217. #pc是基因重组的概率，
218. **if**(random.random()<self.pc):
220. # 双点基因重组
221. #cpoint是开始的点，epoint是终点
222. #cpoint在0到10内取一点
223. #epoint在10到20内取一点
224. cpoint=random.randint(0,len(population[0])-10)
225. epoint=random.randint(10,len(population[0]))
226. #temporary1 和2是用来暂存基因对。
227. temporary1=[]
228. temporary2=[]
229. #将第i个基因对的0到cpoint、第i+1个基因对的cpoint到epoint、第i个基因对的epoint到结尾
230. #。加到temporary1列表里面
231. temporary1.extend(population[i][0:cpoint])
232. temporary1.extend(population[i+1][cpoint:epoint])
233. temporary1.extend(population[i][epoint:len(population[i])])
234. #将第i+1个基因对的0到cpoint、第i个基因对的cpoint到epoint、第i+1个基因对的epoint到结尾
235. #。加到temporary2列表里
236. temporary2.extend(population[i+1][0:cpoint])
237. temporary2.extend(population[i][cpoint:epoint])
238. temporary2.extend(population[i+1][epoint:len(population[i])])
239. #然后将tem1赋值个第i个个体，tem2赋值个第i+1个个体。
240. population[i]=temporary1
241. population[i+1]=temporary2
243. #单点基因重组
244. # cpoint=random.randint(0,len(population[0]))
245. # temporary1=[]
246. # temporary2=[]
247. # temporary1.extend(population[i][0:cpoint])
248. # temporary1.extend(population[i+1][cpoint:len(population[i])])
249. # temporary2.extend(population[i+1][0:cpoint])
250. # temporary2.extend(population[i][cpoint:len(population[i])])
252. # population[i]=temporary1
253. # population[i+1]=temporary2
255. #构造一个mutation函数，用来模拟基因突变，即某个个体有可能突然变得很好，变得很坏。
256. #这里我们采用两种变异方式来模拟基因突变，一种是正交变异算子，这种方法是选取两个基因
257. #对执行异或操作，得到新的基因对，然后执行取反操作。第二种是直接执行取反操作。
258. #输入一个种群，输出一个新的种群
259. **def** mutation(self,population):
260. #px是种群的长度减1
261. px=len(population)-1
262. #py是基因对的长度
263. py=len(population[0])
264. #pm是发生基因突变的概率
265. **for** i **in** range(px):
266. **if**(random.random()<self.pm):
267. #正交基因突变
268. tem1=[]#用来存储异或操作后的新的基因对
269. **for** j **in** range(len(population[0])):
270. #执行异或操作，当相同时为0，不相同时为1
271. **if**(population[i][j]==population[i+1][j]):
272. tem1.append(0)
273. **else**:
274. tem1.append(1)
275. #将tem1覆盖给第i+1个基因对
276. population[i+1]=tem1
278. #取反基因突变
279. #随机在基因对中取一点，用来取反操作
280. mpoint=random.randint(0,py-1)
281. **if**(population[i][mpoint]==1):
282. population[i][mpoint]=0
283. **else**:
284. population[i][mpoint]=1
286. #构造一个b2d函数，
287. # 用来每一个基因对转化成十进制,
288. **def** b2d(self,best\_individual):
289. total=0
290. b=len(best\_individual)
291. **for** i **in** range(b):
292. total=total+best\_individual[i]\*math.pow(2,i)
294. total=total\*self.max\_value/(math.pow(2,self.choromosome\_length)-1)
295. **return** total
297. #构造一个best函数，用来寻找适应值最大的个体
298. #输入种群，适应值列表，输出适应值最大的个体的基因对和适应值
300. **def** best(self,population,fitness\_value):
302. px=len(population)
303. bestindividual=[]
304. bestfitness=fitness\_value[0]
305. # print(fitness\_value)
306. #找到适应值最大的个体，并且将其保存起来
307. **for** i **in** range(1,px):
308. **if**(fitness\_value[i]>bestfitness):
310. bestfitness=fitness\_value[i]
311. bestindividual=population[i]
312. #返回一个列表，列表包含个体基因对和适应值。
313. **return** [bestindividual,bestfitness]
315. #构造一个plot函数，用来画出（x，y）的函数。
316. #输入results列表，输出x，y的函数。
317. **def** plot(self, results):
318. X = []
319. Y = []
320. **for** i **in** range(500):
321. X.append(i)
322. Y.append(results[i][0])
323. #plot是将函数画出
324. plt.plot(X, Y)
325. #show时将函数显示
326. plt.show()
327. #主函数，实现种群的初始化，选择、基因重组、突变，所有的过程。
328. **def** main(self):
330. results = [[]]#二维列表，存储每一轮自然选择后的个体适应值和基因对对应的x的值
331. fitness\_value = []
333. population = pop = self.species\_origin()#随机给出一个初始的种群
335. **for** i **in** range(500):#500次自然选择
336. #建立目标函数，以该函数来计算每个个体的适应值，并且保存起来，调用了function函数
337. function\_value = self.function(population)
338. #选择个体，大于0的个体保持不变，小于0的个体赋值为0.调用了fitness函数
339. fitness\_value = self.fitness(function\_value)
340. #选择最优的个体，将其保存起来，调用了best，b2d函数
341. best\_individual, best\_fitness = self.best(population,fitness\_value)
342. results.append([best\_fitness, self.b2d(best\_individual)])
343. self.selection(population,fitness\_value)#某个个体的适应度越强，让其个体数越多，调用了selection函数。
344. self.crossover(population)#实现基因重组
345. self.mutation(population)#实现基因变异
346. results = results[1:]#选取1到500次的结果
347. results.sort()#排序
348. #画出函数图
349. self.plot(results)
351. **if** \_\_name\_\_ == '\_\_main\_\_':

354. population\_size=400#设置种群个体数为400
355. max\_value=10#设置这个常数为10
356. chromosome\_length=20#设置基因对长度为20
357. pc=0.6#设置基因重组的概率为0.6
358. pm=0.01#设置基因突变的概率为0.01
359. g=GA(population\_size,chromosome\_length,max\_value,pc,pm)#构造一个对象g
360. g.main()#调用里面的主函数

**4 代码测试过程**

**4.1 测试方法的介绍。**

**使用不同的基因重组的方式和基因突变的方式，比较种群进化过程，从而得出较为合理的基因重组和基因突变的组合方式。**

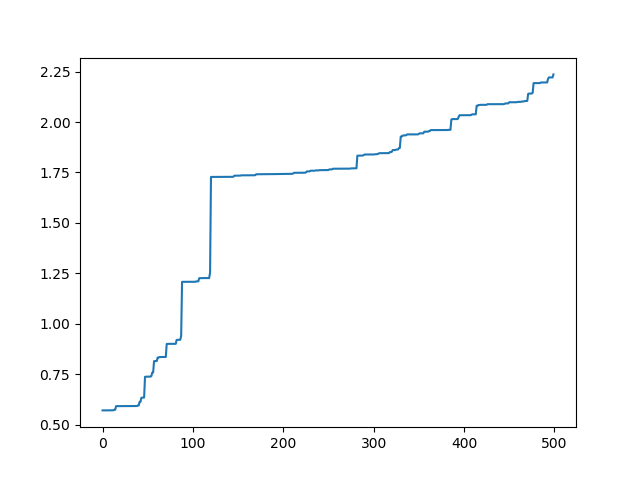
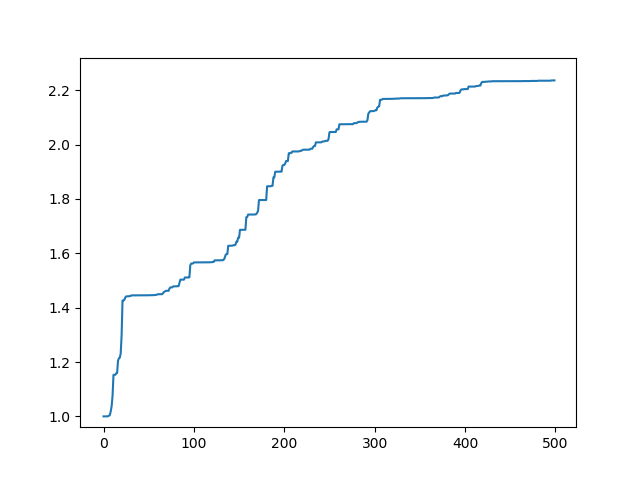
**4.2测试结果**

图4.2.1 单点基因重组和取反突变结果图 图 4.2.2 单点基因重组和正交基因突变结果图

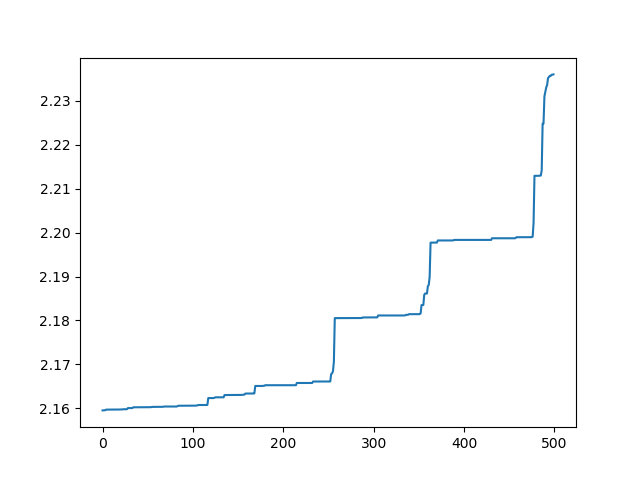
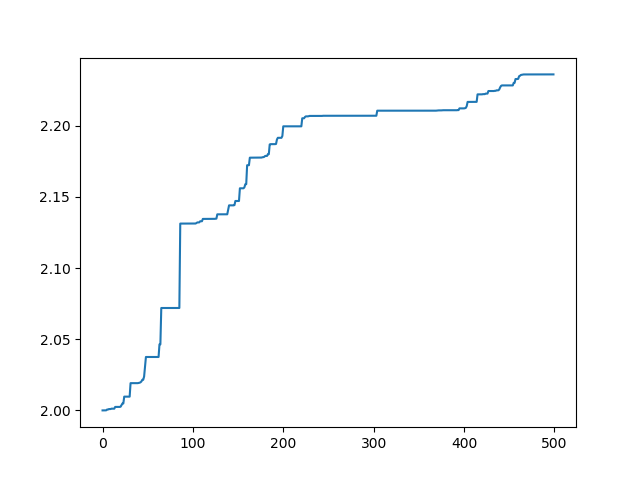


图4.2.3 双点基因重组和取反突变结果图 图 4.2.4 双点基因重组和正交基因突变结果图

比较一下图4.2.1和图4.2.2，使用正交基因突变种群的进化速度更快，而且更加平稳，不像取反突变那样急促的上升，或者又一直平缓。

比较图4.2.2和图4.2.4，一开始种群的适应值就有2，说明双点基因重组更快找到环境的适应度，所以一开始种群的适应值就较高。

结论：为了符合自然选择的优胜劣汰的法则，和自然循序渐进的规律，使用单点基因重组和正交基因突变的遗传算法较为符合自然的选择。

5 总结

通过对遗传算法的探究，我模拟了自然选择的过程，选取最优的个体存活下来，模拟了基因重组和基因突变的过程，比较了两种基因重组的方式，和两种基因突变的方式。并选取了一个目标函数作为环境选择的过程，找到该函数的最优值。我理解和掌握了遗传算法。