链滴

GA 算法

作者：Lonery

原文链接：https://ld246.com/article/1615287948925

来源网站：链滴

遗传算法(genetic algorithm)是模拟达尔文生物进化论的自然选择和遗传学机理的生物进化过程计算型，是一种通过模拟自然进化过程搜索最优解的方法。下面我将分享自己在做GA模型的心得与困惑。

先来整理一下GA的基本步骤：

1. 随机生成一定数量的种群。
2. 对种群的个体进行编码与评估。
3. 选用合适的方法对现有种群中的个体做出选择。
4. 对选择出来的个体进行“交叉”并获得新的个体。
5. 对下一代进行”突变“操作。

第一步:随机生成一定数量的种群。首先应该构思存储个体的数据结构。我选择的是嵌套的列表(list)。个个体的形式为[[个体],[编码],评估值],一个种群则是[[[个体1],[编码1],评估值1],[[个体2] ,[编码2],评估2],[[个体3],[编码3],评估值3],...]。采用随机的方式生成种群/个体：

```
'''
popnum:单位种群中个体数量,cistern:存储单个种群的数据池,lbound,ubound:个体的取值范围
precision:取值的精度,n:问题的维度
'''
def initpop():
    cistern=[]
    for i in range(popnum):
        particle=[]
        for j in range(n):
            particle.append(round(random.uniform(lbound,ubound),precision))
        cistern.append([particle,[],None])
    return cistern
```

<p1 style="text-indent:2em">

第二步:选择合适的编码方案以及评估函数。我采用的是经典的二进制编码形式，这样的编码有个好处可以利用Python自带的bin()函数快速实现,但是一般情况下各个维度的数字是需要编在一起的，而bin(函数不能保证二进制数值的长度，往往导致在转化较小数字的时候其二进制长度过短，即便是将多个度编码后的二进制数值简单拼接，我们也需要在拼接处设置标志位，这样才能在“交叉”过后将各维分开，如果不设标志位而随机分割，那么将会在一定层面上导致“二次交叉”或“自我交叉”，这样失去了“交叉”的意义，为了解决这一问题我们可以采用 format()函数，也可以对各个维度分别编码分别“交叉”，在这里我选择后者。对于评估函数，我采用问题本身，即目标函数的函数值。

</p1>

```
def D2B(num):  #编码(十进制2二进制)
    return bin(round((num-lbound)*(pow(2,l)-1)/(ubound-lbound)))

def B2D(num):  #解码(二进制2十进制)
    return round(lbound+(ubound-lbound)/(pow(2,l)-1)*int(num,2),2)
```

```
def trans(cistern): #对种群进行二进制编码
    for i in cistern:
        for j in range(n):
            i[1].append(D2B(i[0][j]))
    return cistern

def fitness(cistern): #对种群进行评估
    for i in cistern:
        i[2]=fit(i[0])
    return cistern

<p3 style="text-indent:2em">
```

第三步:选用合适的方法对现有种群中的个体做出选择。个体选择的方法有很多，轮盘赌、 锦标赛选、排名选择，等。在这里我选择最为简单的轮盘赌。由于函数对于种群池的修改是直接地 所以我将采“一夫一妻制”，即三个父代个体不能产生两个子代个体，这也是为了避免多余“交叉”。

```
</p3>

def select(cistern):
    reservoir=[]
    choice=[]
    resualt={}
    s=0
    for i in cistern:  #将种群中所有的适应值相加
        s=s+i[2]
    for j in cistern:  #求每个个体的占比
        reservoir.append(j[2]/s)
    for k in range(1,popnum):  #求每个个体的累计占比
        reservoir[k]=reservoir[k]+reservoir[k-1]
    reservoir.insert(0,0)   #补充轮盘的开端，设为0
    for c in range(popnum):  #选择次数为种群容量
        r=random.random()    #选择概率
        for t in range(1,len(reservoir)-1):
            if r<reservoir[t] and r>reservoir[t-1]:
                choice.append(t-1)
    for d in set(choice):   #采用字典的方式存储各个个体以及被选中的次数
        resualt[d]=choice.count(d)
    resualt=sorted(resualt.items(), key=lambda item: item[1],reverse=True) #以被选中的次数大排序
    choice=[]
    for u in resualt:
      choice.append(u[0])
    return choice #返回选择的结果(下标)

<p4 style="text-indent:2em">
```

第四步:对选择出来的个体进行“交叉”并获得新的个体。这一步地操作，我采用对原种群直接修改的式。 对于两个个体的编码，选择一个交叉点位并将交叉点位以及其后面的二进制位一并交换，而且也用前面提到的分别“交叉”。

```
</p4>
```

```
def crossover(a,b): #染色体(个体)单位
    ch=random.random()  #交叉概率
    if ch<overchance:
        length=9999999
        for i in a[1]+b[1]:  #对较小长度的二进制选择交叉位作为各个维度的交叉位
            if len(i)<length:
                length=len(i)
        r=random.randint(2,length-1) #选择交叉位
        for j in range(n):  #对二进制数值进行剪切和拼接
            temp=a[1][j][r:]
            a[1][j]=a[1][j][:r]+b[1][j][r:]
            b[1][j]=b[1][j][:r]+temp
            a[0][j]=B2D(a[1][j])  #更改十进制数值
            b[0][j]=B2D(b[1][j])
        a[2]=fit(a[0])    #重新评估函数
        b[2]=fit(b[0])

<p5 style="text-indent:2em">
```

第五步:对下一代进行”突变“操作。“突变”是此算法中比较简单的流程，一般采用单点变异。

```
</p5>

def variation(cistern):
    for i in cistern:
        c=random.random()  #变异概率
        if c<variantchance:  #达成条件进行变异
            m=99999
            for x in i[1]:  #以长度较小的二进制值选择点位
                if len(x)<m:
                    m=len(x)
            r=random.randint(2,m-1) #变异点位
            for j in range(n):    #更换点位的值
                if i[1][j][r]=='0':
                    i[1][j]=replace_char(i[1][j],'1',r)
                    i[0][j]=B2D(i[1][j])
                else:
                    i[1][j]=replace_char(i[1][j],'0',r)
                    i[0][j]=B2D(i[1][j])
            i[2]=fit(i[0]) #重新评估函数
    return cistern

<p6 style="text-indent:2em">
```

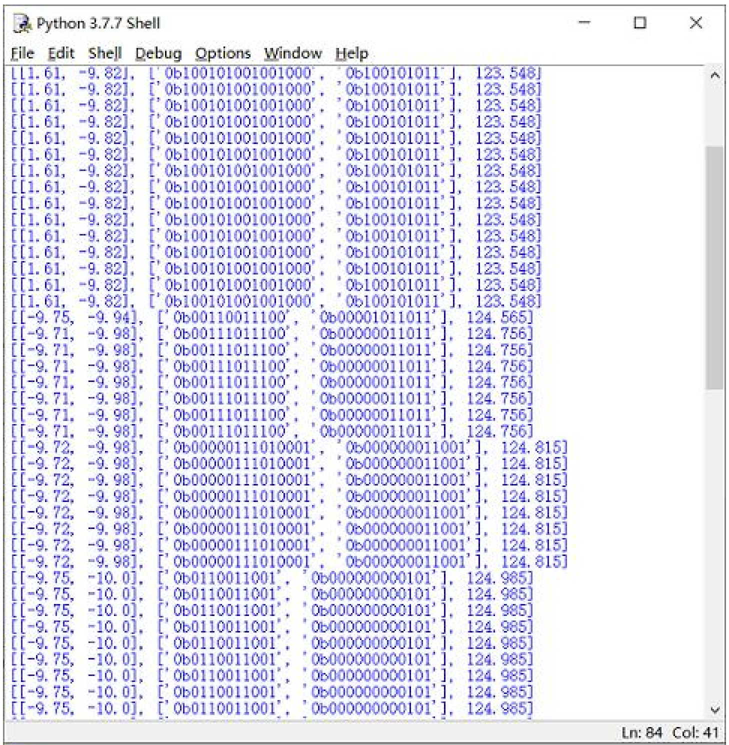
最后进行一下总结，第一步以树的结构存储种群中的个体，第二部采用分别编码，第三步采用“一夫妻制”选择 并且优先选择被选中次数多的个体进行下一步的操作，第四步采用分别“交叉”，第五步单变异。剩下还有一些特殊的函数 匹配函数，保持最优函数，目标函数和字符交叉函数。

```
</p6>

def replace_char(string,char,index): #单字符交叉，用于“变异”
    string=list(string)
    string[index]=char
```

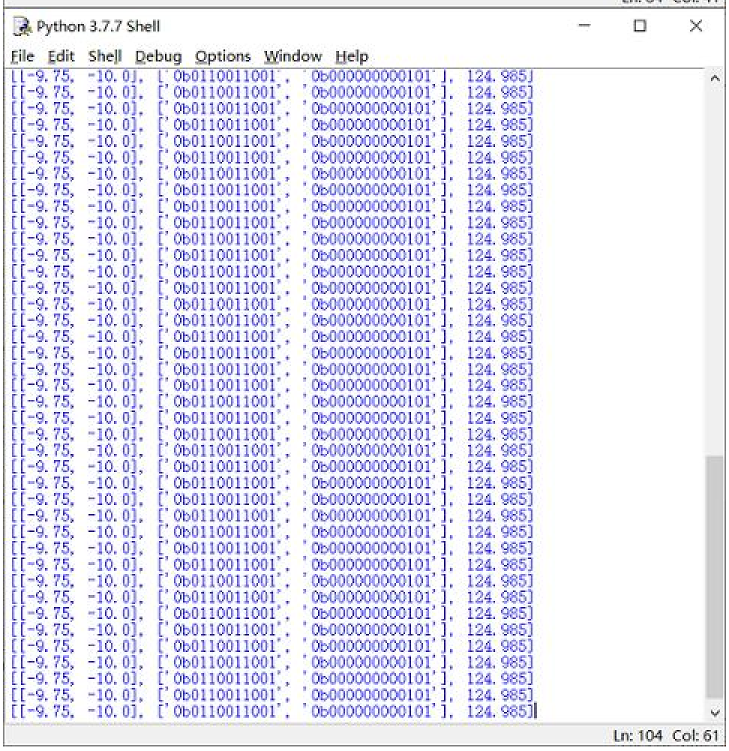
```
    return ''.join(string)

def fit(x):  #目标函数，函数值即为适应度。
    value=round(10*math.sin(5*x[0])+7*abs(x[1]-5)+10,precision)
    return value

def match(choice):  #匹配函数，防止选选择的个体个数为奇数。
    if len(choice)%2!=0:
        choice.pop()
    return choice

def best(cistern,Best):  #最优个体保持函数
    tem=sorted(cistern,key=lambda x:x[2],reverse=True)[0]
    if not Best:
        Best=copy.deepcopy(tem)
    elif tem[2]>Best[2]:
        Best=copy.deepcopy(tem)
    return Best
```

接下来以上面代码的目标函数对GA进行测试：

```
import math
import random
import copy
popnum=50
ubound=10
lbound=-10
precision=3
l=0
while pow(2,l)<(ubound-lbound)/(1/pow(10,precision)): #计算所需二进制长度
    l=l+1
n=2 #2-D
overchance=0.8
variantchance=0.3
Best=[]
a=initpop()
trans(a)
fitness(a)
for j in range(100):
    s=select(a)
    if s:
        c=match(s)
        for i in range(0,len(c),2):
            crossover(a[c[i]],a[c[i+1]])
    variation(a)
    Best=best(a,Best)
    print(Best)
```



```
Python 3.7.7 Shell
File Edit Shell Debug Options Window Help
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[1.61, -9.82], ['0b100101001001000', '0b100101011'], 123.548]
[[-9.75, -9.94], ['0b00110011100', '0b00001011011'], 124.565]
[[-9.71, -9.98], ['0b00111011100', '0b00000011011'], 124.756]
[[-9.71, -9.98], ['0b00111011100', '0b00000011011'], 124.756]
[[-9.71, -9.98], ['0b00111011100', '0b00000011011'], 124.756]
[[-9.71, -9.98], ['0b00111011100', '0b00000011011'], 124.756]
[[-9.71, -9.98], ['0b00111011100', '0b00000011011'], 124.756]
[[-9.71, -9.98], ['0b00111011100', '0b00000011011'], 124.756]
[[-9.71, -9.98], ['0b00111011100', '0b00000011011'], 124.756]
[[-9.72, -9.98], ['0b00000111010001', '0b000000011001'], 124.815]
[[-9.72, -9.98], ['0b00000111010001', '0b000000011001'], 124.815]
[[-9.72, -9.98], ['0b00000111010001', '0b000000011001'], 124.815]
[[-9.72, -9.98], ['0b00000111010001', '0b000000011001'], 124.815]
[[-9.72, -9.98], ['0b00000111010001', '0b000000011001'], 124.815]
[[-9.72, -9.98], ['0b00000111010001', '0b000000011001'], 124.815]
[[-9.72, -9.98], ['0b00000111010001', '0b000000011001'], 124.815]
[[-9.72, -9.98], ['0b00000111010001', '0b000000011001'], 124.815]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
Ln: 84 Col: 41
```



```
Python 3.7.7 Shell
File Edit Shell Debug Options Window Help
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
[[-9.75, -10.0], ['0b0110011001', '0b000000000101'], 124.985]
Ln: 104 Col: 61
```

原文链接：GA 算法