گزارش پروژه ی اختیاری

مهسا امینی ۹۸۱۷۸۲۳

در چنین مسائلی دو فضا برای ما تعریف شده است یکی فضای محاسباتی و دیگری فضای واقعی که به فضای محاسباتی ژنوتایپ و به فضای واقعی فنوتایپ گفته میشود.

اگر یک کروموزوم را به یک جواب تبدیل کنیم به این کار decoding گفته میشود و تبدیل فنوتایپ به کروموزوم را encoding میگویند .

در ابتدا یک عملیات encoding انجام میدهیم و یک کروموزوم تولید میکنیم که این کروموزوم مربوط به عکس اصلی است و نام آن را target میگذاریم:

```
# encoding
target = numpy.reshape(a=image, newshape=(functools.reduce(operator.mul, im_shape)))
```

یکی از مهم ترین قسمت ها در الگوریتم ژنتیک تابع برازندگی است که ارزیابی میکند راه حل ما چقدر به راه حل بهینه نزدیک است. تابع برازندگی که برای این سوال ایجاد کردیم حداکثر تفاوت بین تصویر اصلی و تصویر ساخته شده را محاسبه میکند که این روشی است برای اینکه تشخیص دهیم تصویرایجاد شده چقدر به تصویر اصلی نزدیک است.

تابع برازندگی دو مقدار میگیرد اولی target است که یک کروموزم است که از روی عکس اصلی ساخته شده است و پارامتر دوم آن population است که برابر مجموعه ای از راه حل ها در زمان اکنون است. هر گاه جمعیت جدیدی داریم مقدار کروموزوم ساخته شده از روی عکس اصلی و جمعیت جدید به تابع برازندگی میدهیم .

تابع برازندگی به شکل زیر پیاده سازی میشود:

```
def fitness(target, population):

    n = population.shape[0]
    res = numpy.zeros(n)
    for i in range(n):
        res[i] = numpy.sum(target) - numpy.mean(numpy.abs(target- population[i,:]))
    return res
```

در الگوریتم ژنتیک در ابتدا به یک جمعیت اولیه نیاز داریم ما دو راهکار برای ایجاد این جمعیت داریم. راه حل اول این است که جمعیت اولیه را به صورت رندوم مقداردهی کنیم و راه حل دوم این است که بر اساس یک heuristic استفاده کنیم. که برای حل این سوال ما از روش اول استفاده کردیم و به صورت تصادفی مقداردهی میکنیم.

```
primary_population = numpy.empty(shape=(n,reduce(operator.mul, im_shape)))
for i in range(n):
    primary_population[i,:] = numpy.random.random(reduce(operator.mul, im_shape))
    primary_population[i,:] *=256
```

پس از اینکه یک نسل ایجاد شد در این نسل ما باید بهترین افراد را انتخاب کنیم و از افراد انتخاب شده برای تولید مثل استفاده کنیم برای اینکار تابع زیر را تعریف کرده ایم:

```
def choose_parents(res_fitness,population,n):

    n1 = population.shape[1]
    parents = numpy.empty((n,n1))

    for i in range(n):
        index = numpy.where(res_fitness == numpy.max(res_fitness))[0][0]
        parents[i, :] = population[index, :]
        res_fitness[index] = -1

    return parents
```

حال والد هارا داریم و باید فرزندان را تولید کنیم که از عملگر crossover استفاده میکنیم این عملگر برای ترکیب اطلاعات ژنتیکی دو والد برای تولید فرزندان جدید استفاده میشود.

که این کار را به روش های مختلفی میتوان انجام داد یکی از روش های آن one point crossover است که یک نقطه را مشخص میکنیم و از دو نقطه دو تکه از والدین را با هم جا به جا میکنیم که به شکل زیر انجام میشود:

برای حل این سوال نقطه ی مورد نظر را در وسط کروموزوم در نظر گرفتیم.

و همچنین در crossover فرزندان میتوانند زمانی جایگزین والد خود شوند که از والد خود برتر باشند. همچنین روی والد ها نیز باید عملیات جایگشت انجام شود. تابع به شکل زیر خواهد بود:

```
def crossover(n,parents,im_shape):

    n1 = parents.shape[0]
    population = numpy.empty(shape=(n, reduce(operator.mul, im_shape)), dtype=numpy.uint8)
    population[0:n1,:] = parents
    n2 = n-n1

    # permutation
    p_parents= list(permutations(iterable=numpy.arange(0,n1), r=2))
    f_permutation = random.sample(range(len(p_parents)), n2)

    for i in range(len(f_permutation)):

        c0 = p_parents[f_permutation[i]][0]
        c1 = p_parents[f_permutation[i]][1]
        population[n1+i, 0:int(population.shape[1]/2)] = parents[c0,0:int(population.shape[1]/2)]
        population[n1+i, int(population.shape[1]/2):] =  parents[c1, int(population.shape[1]/2):]

    return population
```

و در نهایت mutation یا جهش داریم که برای حفظ تنوع استفاده میشود و برای جهش باید یک نرخ مشخص کنیم. جهش به روش های مختلفی اتفاق می افتد مانند inversion mutation، scramble mutation ، bit flip mutation و ...

در اینجا برای این سوال تعدادی از ژن هارا به صورت تصادفی انتخاب میکنیم و مقدار آن ها را باز هم به صورت تصادفی تغییر میدهیم.

که تابع آن به صورت زیر است:

```
def mutation(n,population, mutation_rate):

    for i in range(n, population.shape[0]):
        n1 =population.shape[1]
        index = numpy.uint32(numpy.random.random(size=numpy.uint32(mutation_rate/100*n1))*n1)
        population[i, index] = numpy.uint8(numpy.random.random(size=index.shape[0])*256)
    return population
```

و در نهایت باید کروموزوم را به جواب تبدیل کنیم. و نشان دهیم.