تمرین چهارم به نام خدا

سید سجاد حسینی- علیرضا ظهوریان

نحوه محاسبه Fitness :

```
 def select_parents(self):
        fitness = [1 / self.calculate_distance(tour) for tour in
self.population]
        total_fitness = sum(fitness)
```

برای هر تور کروموزوم جمعیت که به صورت رندم تولید شده اند ، عدد ۱ را تقسیم بر میزان فاصله تور آن کروموزوم جمعیت میکنیم. از آنجا که فیتنس هر چه قدر بزرگتر باشد نشان دهنده بهتر بودن کروموزوم هست پس با استفاده از این روش زمانی که فاصله تور کروموزوم ما کمتر باشد ، مخرج کسر کوچک تر میشود و درنتیجه فیتنس بهتری به ما میدهد و بالعکس.

در نهایت تمام فیتنس کروموزم های موجود را جمع و درون total_fitness می‌ریزیم.

نحوه انتخاب ژن های اولیه :

```
def initialize_population(self):
        return [np.random.permutation(len(self.points)) for _ in
range(self.population_size)]
```

در این کد برای هر کروموزوم به صورت رندم به اندازه کروموزوم نقاط یا ژن ها تصادفی انتخاب میشوند و این کار به تعداد جمعیتی که انتخاب کردیم انجام می شود .

نحوه جهش :

```
def mutate(self, tour):
        if np.random.rand() < self.mutation_rate:
            idx1, idx2 = np.random.choice(len(tour), 2, replace=False)
            tour[idx1], tour[idx2] = tour[idx2], tour[idx1]
```

در این کد ابتدا یک اگر عدد رندم انتخابی کمتر از mutation_rate ما باشد کد اجرا می شود سپس دو ژن یا نقطه از کروموزوم به صورت رندم انتخاب می شوند و در خط بعد مقادیر آن ژن ها رو جا به جا می‌کند.

نحوه Selection :

```
    def select_parents(self):
        fitness = [1 / self.calculate_distance(tour) for tour in
self.population]
        total_fitness = sum(fitness)

        if total_fitness == 0:
            raise ValueError("Total fitness is zero. Something went wrong
with the distances.")

        probabilities = [f / total_fitness for f in fitness]

        population_indices = np.arange(len(self.population))

        return np.random.choice(population_indices, size=2,
p=probabilities)
```

بعد از محاسبه فیتنس ابتدا چک میکند که اگر فیتنسی برابر با صفر بود با مشکلی روبه رو شده و ارور می‌دهد. در ادامه اگر درست بود متغیری به نام probabilities یا احتمالات که داخل آن مقدار احتمال هر کروموزم تقسیم میشود بر مجموع فیتنس کروموزوم ها و سپس در متغیر population_indices لیست مرتبی به اندازه جمعیت ایجاد میکند.

در ادامه در بازه لیست جمعیتی که ما داریم دو کروموزوم که احتمال فیتنس بیشتری دارند انتخاب می شوند .

نحوه Crossover :

```
def crossover(self, parent1, parent2):
        start, end = sorted(np.random.choice(len(parent1), 2,
replace=False))
        child = [None] * len(parent1)
        child[start:end + 1] = parent1[start:end + 1]

        current_position = (end + 1) % len(parent1)
        for gene in parent2:
            if gene not in child:
                child[current_position] = gene
                current_position = (current_position + 1) % len(parent1)
        return np.array(child)
```

ابتدا دو کروموزم والد را دریافت می‌کند. در خط بعد به اندازه دامنه والد اول ۲ نقطه که تکراری نباشد را انتخاب می‌کند، یکی برای شروع و دیگری برای پایان. سپس برای فرزند به اندازه والد اول لیستی خالی درست می‌کند. در ادامه از ابتدا تا نقطه پایانی که انتخاب شده را، ژن های آن را جایگزین می‌کند با همان نقطه شروع و پایان در والد . در خط بعدی مکان فعلی که نقطه پایان قرار دارد را ذخیره می‌کنیم. در حلقه اگر ژنی با ژن والد دوم یکسان نبود وارد دستورات حلقه می‌شویم و باقی مانده ژن های فرزند که خالی هستند را با ژن های والد دوم جایگزین میکنیم و در نهایت فرزند را return می‌کنیم.

تغییر روش Crossover به PMX :

و

```
def pmx_crossover(self, parent1, parent2):
        size = len(parent1)
        start, end = sorted(np.random.choice(size, 2, replace=False))

        child = [-1] * size
        child[start:end + 1] = parent1[start:end + 1]

        for i in range(start, end + 1):
            if parent2[i] not in child:
                val = parent2[i]
                pos = i
                while start <= pos <= end:
                    pos = np.where(parent2 == parent1[pos])[0][0]
                child[pos] = val

        for i in range(size):
            if child[i] == -1:
                child[i] = parent2[i]

        return np.array(child)
```

ابتدا اندازه والد اول را در size میریزم سپس در با استفاده از متد choice به اندازه دامنه size ۲ عدد را که تکراری نباشد انتخاب میکنیم و در ادامه این اعداد را مرتب میکنیم و در متغیر های start و end می‌ریزیم . سپس child را تشکیل میدهیم به اندازه والد اول که خانه های آن خالی هست . در ادامه با استفاده از نقطه start و end که به دست آوردیم نقاط را تا end پر میکنیم . سپس در حلقه for ادامه end به بعد را پر می‌کنیم. درادامه ایندکسی از والد دوم را انختاب میکنیم اگر این ایندکس در child نبود دستورات بعدی رو اجرا می‌کنیم.

در متغیر val مقداری که در ایندکس انتخاب شده بود را می‌ریزیم و در pos عدد ایندکس آن.

در حلقه while تا زمانی که ایندکس ما بین و یا مساوی با نقاط start و end بود که در والد اول انتخاب کرده بودیم، ادامه کد while اجرا می‌شود. سپس در متغیر pos اگر مقداری از نقاط والد دوم با مقدار ایندکس pos والد اول یکی باشد را پیدا میکند و ایندکس والد دوم که با والد اول یکی بوده را در pos می‌ریزد. در مرحله بعد مکانی را که در child با استفاده از pos پیدا کرده بودیم مقدار val را در آن ایندکس میریزیم و این کار را به اندازه start تا end ادامه میدهیم.

سپس در حلقه for بعدی اگر child ما خانه خالی داشته باشد همان خانه با خانه های والد دوم جایگزین می‌شود. و در آخر child را return می‌کنیم.

خروجی کد تغییر داده شده :

```
Best tour: ['Arak', 'Urmia', 'Kermanshah', 'Ahvaz', 'Tabriz', 'Bandar
Abbas', 'Shiraz', 'Gorgan', 'Zanjan', 'Tehran', 'Kerman', 'Qom',
'Isfahan', 'Yazd', 'Birjand', 'Mashhad', 'Bojnurd', 'Sabzevar', 'Sari',
'Karaj']
Best distance: 99.93798473314757
```

خروجی کد بدون تغییر:

```
Best tour: ['Bandar Abbas', 'Birjand', 'Sari', 'Ahvaz', 'Tehran',
'Gorgan', 'Urmia', 'Tabriz', 'Isfahan', 'Qom', 'Zanjan', 'Arak',
'Kermanshah', 'Karaj', 'Bojnurd', 'Sabzevar', 'Mashhad', 'Yazd', 'Shiraz',
'Kerman']
Best distance: 91.78347815295244
```