Amir Hossein Sadeghi

تحلیل کد الگوریتم ژنتیک برای مسئله فروشنده دوره‌گرد (TSP)

1. Selection (انتخاب والدین)

روش: Fitness Proportional Selection (تناسب تناسبی یا روش چرخ رولت)

- انتخاب می‌شوند (fitness) والدین براساس میزان فیتنس.

- محاسبه فیتنس:

  - فیتنس هر مسیر برابر با معکوس فاصله کل مسیر است:

  - هر چه مسیر کوتاه‌تر باشد، فیتنس بیشتر است.

- احتمال انتخاب:

  - احتمال انتخاب هر مسیر متناسب با سهم فیتنس آن از مجموع فیتنس کل جمعیت است:

probabilities = [f / total_fitness for f in fitness]

  - دو والد با استفاده از این احتمالات انتخاب می‌شوند:

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

2. Crossover (ترکیب والدین برای تولید فرزندان)

روش: Partial-Mapped Crossover (PMX)

- از این روش برای ترکیب دو مسیر استفاده شده است:

  - بخش تصادفی والد اول (بین دو نقطه انتخاب شده تصادفی) مستقیماً به فرزند منتقل می‌شود.

  - باقی ژن‌ها از والد دوم وارد می‌شوند، با اطمینان از اینکه هیچ تکراری وجود ندارد.

- جزئیات پیاده‌سازی:

  - ابتدا دو اندیس تصادفی در مسیر انتخاب می‌شود:

start, end = sorted(np.random.choice(len(parent1), 2, replace=False))

- ژن‌های بین این دو اندیس از والد اول مستقیماً به فرزند منتقل می‌شوند:

child[start:end + 1] = parent1[start:end + 1]

- سپس باقی ژن‌ها از والد دوم به ترتیب و بدون تکرار اضافه می‌شوند.

3. Mutation (جهش)

روش: Swap Mutation

- هر مسیر، با احتمال کمی (برابر با نرخ جهش `mutation_rate`)، دو ژن تصادفی جابجا می‌شوند.

- **پیاده‌سازی:**

- دو اندیس تصادفی در مسیر انتخاب و مقادیرشان با هم جابجا می‌شوند:

idx1, idx2 = np.random.choice(len(tour), 2, replace=False)

tour[idx1], tour[idx2] = tour[idx2], tour[idx1]

- هدف:

- ایجاد تنوع در جمعیت و جلوگیری از گیر افتادن الگوریتم در کمینه محلی.

4. Generation (تولید نسل جدید)

- در هر نسل:

1. والدین با روش Fitness Proportional Selection انتخاب می‌شوند.
2. ترکیب والدین با روش PMX Crossover انجام می‌شود.
3. فرزندان با احتمال کمی جهش می‌یابند.
4. جمعیت فعلی با نسل جدید جایگزین می‌شود:

self.population = new_population

5. Fitness Calculation (محاسبه فیتنس)

معیار: کل فاصله مسیر

- فیتنس معکوس فاصله کل مسیر است.

- فاصله بین دو نقطه از فرمول اقلیدسی محاسبه می‌شود:

distance += np.linalg.norm(np.array(point1) - np.array(point2))

مروری بر فرآیند کلی:

1. جمعیت اولیه: با ترتیب‌های تصادفی از شهرها ایجاد می‌شود
2. فیتنس: برای هر مسیر محاسبه می‌شود.
3. انتخاب: دو والد براساس فیتنس انتخاب می‌شوند
4. ترکیب: والدین برای تولید فرزندان ترکیب می‌شوند.
5. جهش: فرزندان با احتمال کمی دچار جهش می‌شوند
6. تکرار: فرآیند برای تعداد مشخصی نسل ادامه پیدا می‌کند.
7. نتیجه نهایی: بهترین مسیر و کوتاه‌ترین فاصله بازگردانده می‌شود.

از روش **Tournament Selection** استفاده می‌کنیم

```
import numpy as np
import pandas as pd

# تعریف داده‌های شهرها
data = {
    'City': ['Tehran', 'Isfahan', 'Tabriz', 'Shiraz', 'Mashhad', 'Kermanshah', 'Yazd', 'Karaj', 'Ahvaz', 'Qom'],
    'Latitude': [35.6892, 32.6546, 38.0962, 29.5918, 36.2605, 34.3293, 31.8974, 35.8325, 31.3193, 34.639],
```

```
  'Longitude': [51.3890, 51.6570, 46.2913, 52.5836, 59.5443, 47.1167, 54.3660, 51.9792, 48.6692, 50.8764]
}
iran_df = pd.DataFrame(data)
points = list(zip(iran_df['Longitude'], iran_df['Latitude']))

# الگوریتم ژنتیک ساده
class SimpleGeneticAlgorithm:
  def __init__(self, points, population_size=50, generations=200, mutation_rate=0.05):
    self.points = points
    self.population_size = population_size
    self.generations = generations
    self.mutation_rate = mutation_rate
    self.population = self.initialize_population()

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

  def calculate_distance(self, tour):
    distance = 0
    for i in range(len(tour)):
      point1 = self.points[tour[i]]
      point2 = self.points[tour[(i + 1) % len(tour)]]
      distance += np.linalg.norm(np.array(point1) - np.array(point2))
    return distance
```

```
def tournament_selection(self, k=5):
    selected = np.random.choice(self.population, k)
    return min(selected, key=self.calculate_distance)

def order_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]
    pointer = (end + 1) % len(parent1)
    for gene in parent2:
        if gene not in child:
            child[pointer] = gene
            pointer = (pointer + 1) % len(parent1)
    return child

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]

def run(self):
    for generation in range(self.generations):
        new_population = []
        for _ in range(self.population_size):
            parent1 = self.tournament_selection()
            parent2 = self.tournament_selection()
```

```
            child = self.order_crossover(parent1, parent2)
            self.mutate(child)
            new_population.append(child)
        self.population = new_population
    best_tour = min(self.population, key=self.calculate_distance)
    best_distance = self.calculate_distance(best_tour)
    return best_tour, best_distance

# اجرا
ga = SimpleGeneticAlgorithm(points)
best_tour, best_distance = ga.run()

# نمایش نتایج
cities = iran_df['City'].tolist()
print("Best tour:", [cities[i] for i in best_tour])
print("Best distance:", best_distance)
```