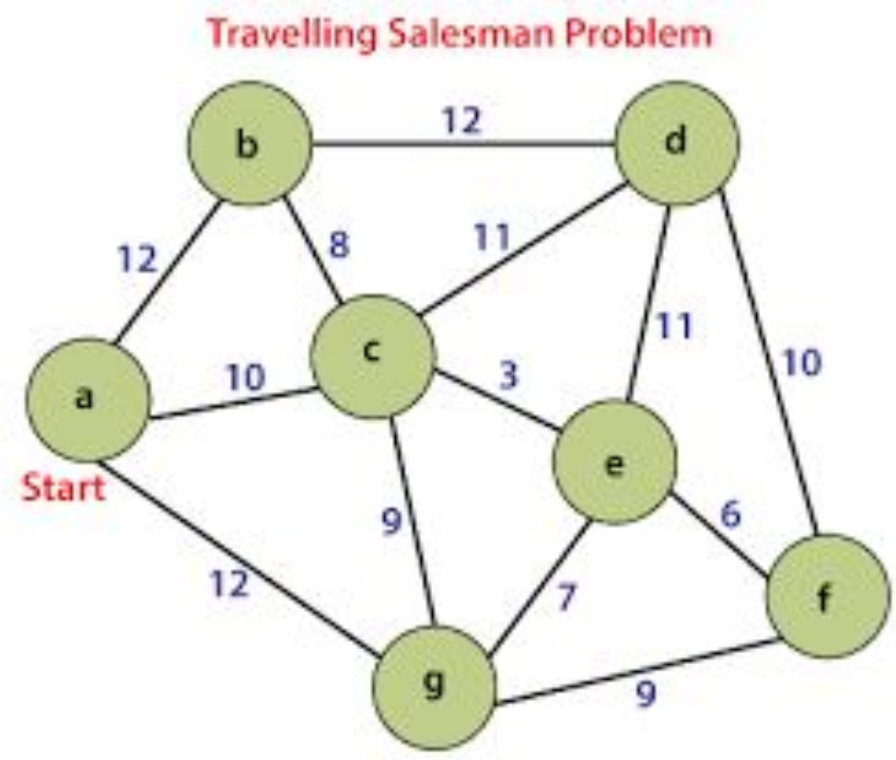
Genetic Algorithm - Traveling Salesman Problem

Daeyeon Kim

1. Traveling Salesman Problem

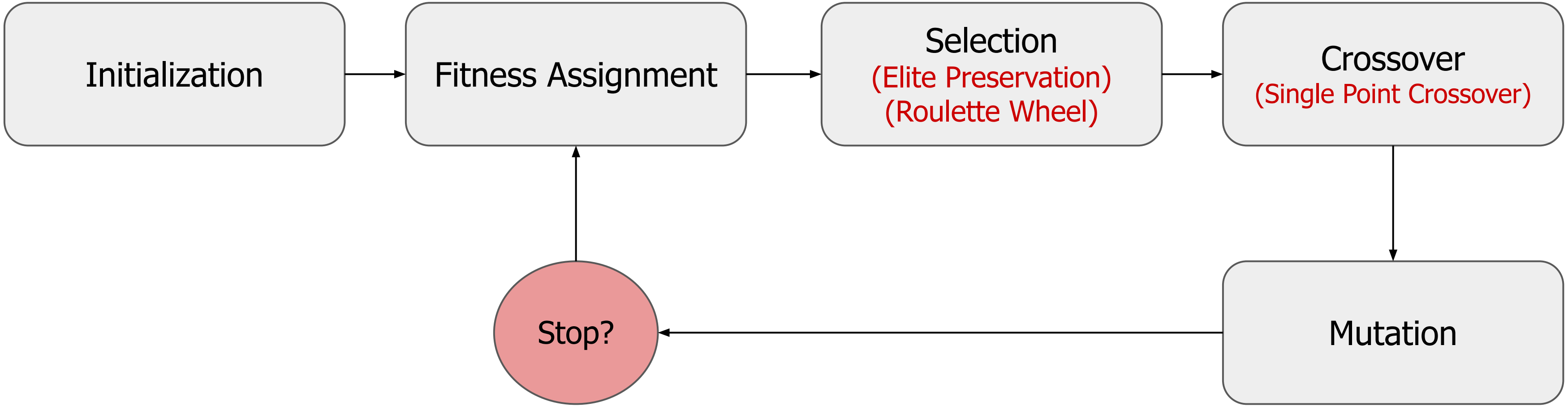


"Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?"

→Apply genetic algorithm to solve the traveling salesman problem

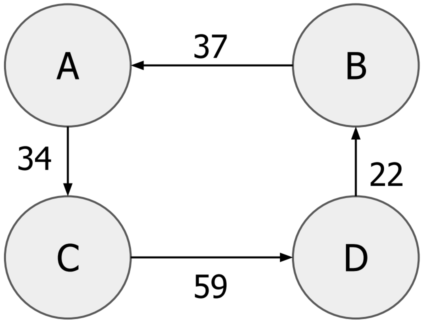
2.Genetic Algorithm

- Metaheuristic algorithm based on Darwin’s evolution of biological systems



2.Genetic Algorithm

- 2.1 Chromosome
- Each chromosome contains the route information and the total distance for a route

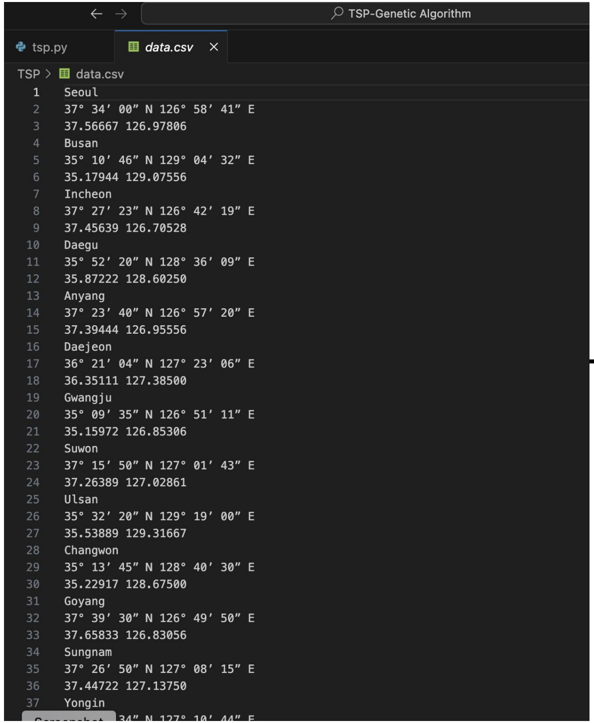


Route	Distance
A→C→D→B	152

<Chromosome>

2.Genetic Algorithm

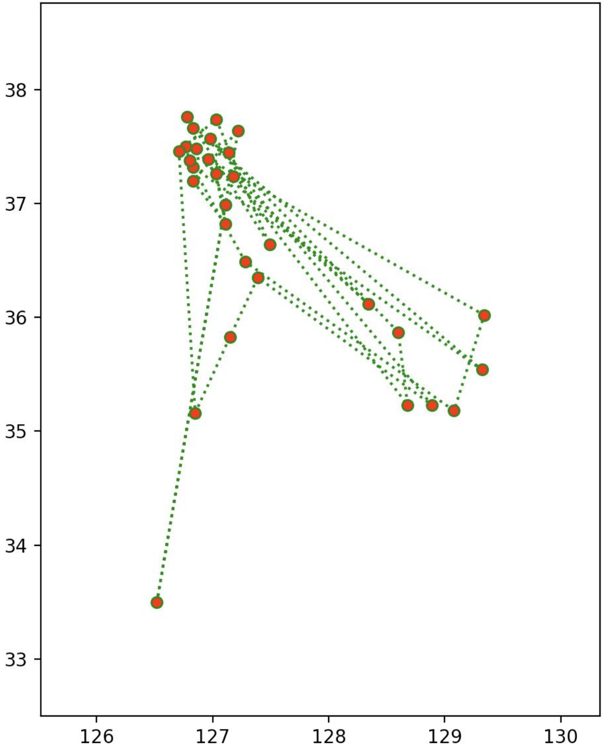
- 2.2 Initialization



```
Seoul
37° 34’ 00” N 126° 58’ 41” E
37.56667 126.97806
Busan
35° 10’ 46” N 129° 04’ 32” E
35.17944 129.07556
Incheon
37° 27’ 23” N 126° 42’ 19” E
37.45639 126.70528
Daegu
35° 52’ 20” N 128° 36’ 09” E
35.87222 128.60250
Anyang
37° 23’ 40” N 126° 57’ 20” E
37.39444 126.95556
Daejeon
36° 21’ 04” N 127° 23’ 06” E
36.35111 127.38500
Gwangju
35° 09’ 35” N 126° 51’ 11” E
35.15972 126.85306
Suwon
37° 15’ 50” N 127° 01’ 43” E
37.26389 127.02861
Ulsan
35° 32’ 20” N 129° 19’ 00” E
35.53889 129.31667
Changwon
35° 13’ 45” N 128° 40’ 30” E
35.22917 128.67500
Goyang
37° 39’ 30” N 126° 49’ 50” E
37.65833 126.83056
Sungnam
37° 26’ 50” N 127° 08’ 15” E
37.44722 127.13750
Yongin
           34” N 127° 10’ 44” E
```

List of cities in South Korea

- Longtitude
- Latitude



2.Genetic Algorithm

- 2.3 Elite Preserving Selection

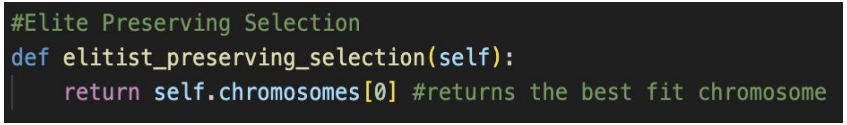
Route(1)	Distance
A→C→D→B	152

Route(2)	Distance
A→C→B→D	155

Route(3)	Distance
A→B→C→D	171

Route(4)	Distance
A→B→D→C	198

Selects the best fit chromosome for the next generation



```python
#Elite Preserving Selection
def elitist_preserving_selection(self):
    return self.chromosomes[0] #returns the best fit chromosome
```

<Code for elite preserving selection>

2.Genetic Algorithm

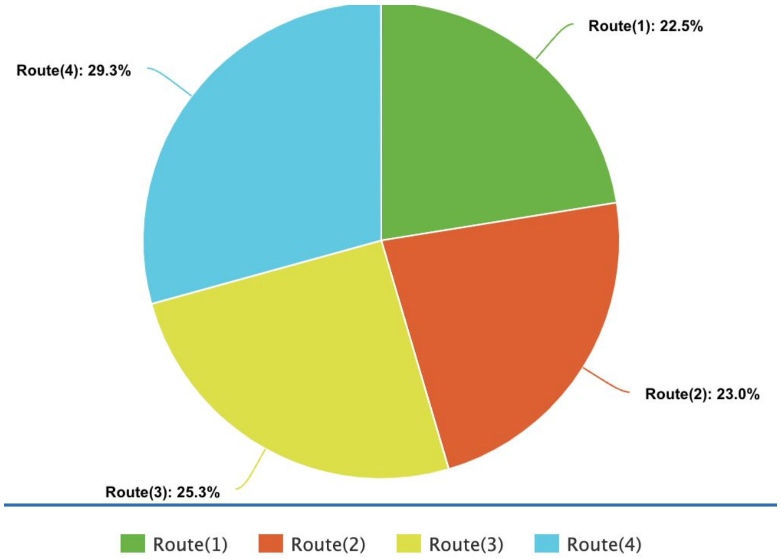
- 2.4 Roulette Wheel Selection

Route(1)	Distance
A→C→D→B	152

Route(2)	Distance
A→C→B→D	155

Route(3)	Distance
A→B→C→D	171

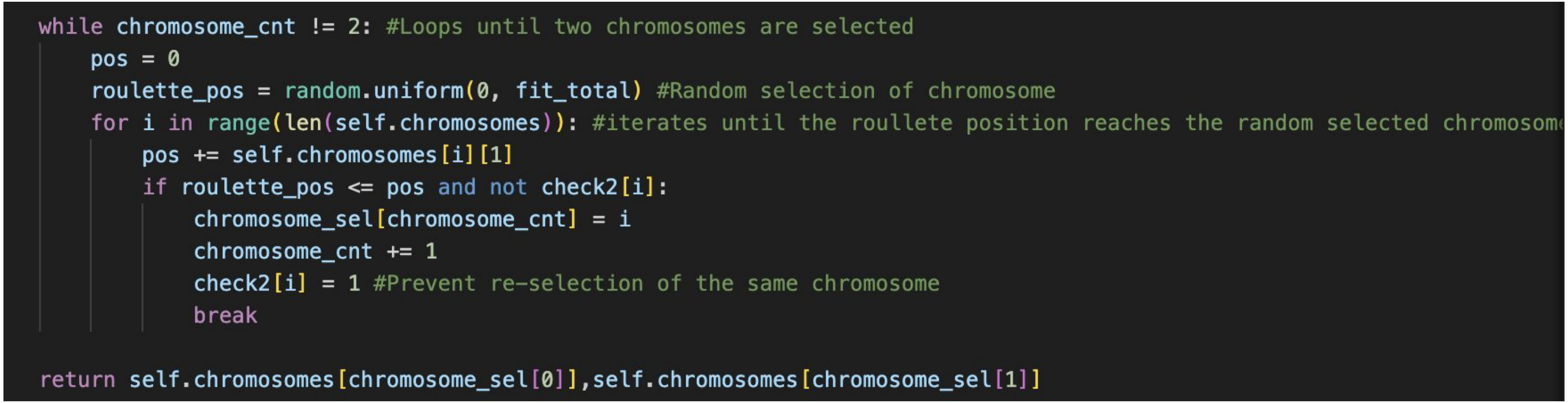
Route(4)	Distance
A→B→D→C	198



- Randomly selects a chromosome for the next generation by placing them in a roulette and assigning them probabilities based on their fitness scores

2.Genetic Algorithm

- 2.4 Roulette Wheel Selection



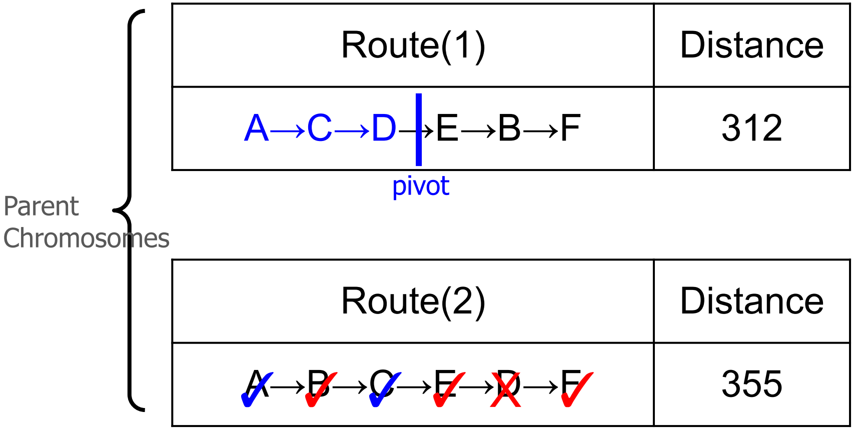
```
while chromosome_cnt != 2: #Loops until two chromosomes are selected
    pos = 0
    roulette_pos = random.uniform(0, fit_total) #Random selection of chromosome
    for i in range(len(self.chromosomes)): #iterates until the roullete position reaches the random selected chromosom
        pos += self.chromosomes[i][1]
        if roulette_pos <= pos and not check2[i]:
            chromosome_sel[chromosome_cnt] = i
            chromosome_cnt += 1
            check2[i] = 1 #Prevent re-selection of the same chromosome
            break

return self.chromosomes[chromosome_sel[0]],self.chromosomes[chromosome_sel[1]]
```

<Code for roulette wheel selection>

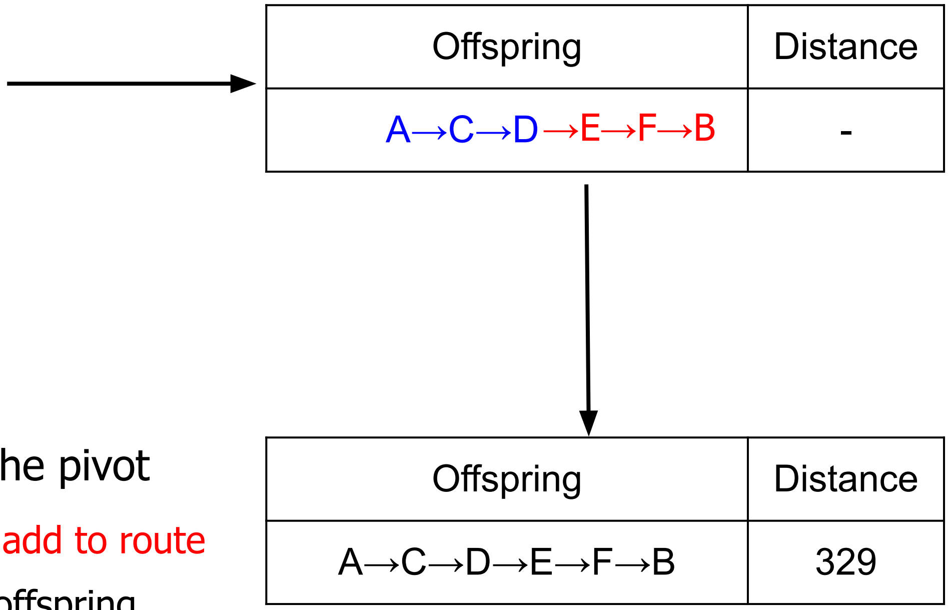
2.Genetic Algorithm

- 2.5 Single Point Crossover
- 1) Picks a random pivot for crossover



Route(1)	Distance
A→C→D | →E→B→F	312

Route(2)	Distance
A✓→B✓→C✓→E✓→D✗→F✓	355



Offspring	Distance
A→C→D→E→F→B	-

Offspring	Distance
A→C→D→E→F→B	329

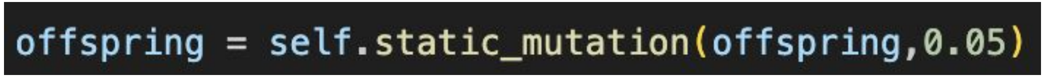
- 2) Adds new route to the offspring from the pivot
  - Checks for a duplicate city visit, and does not add to route
  - Evaluates new fitness score, produces a new offspring

2.Genetic Algorithm

- 2.6 Static Mutation
- Selects two points and swaps them

Offspring	Distance
A→C→D→E→F→B	329

pivot1 pivot2



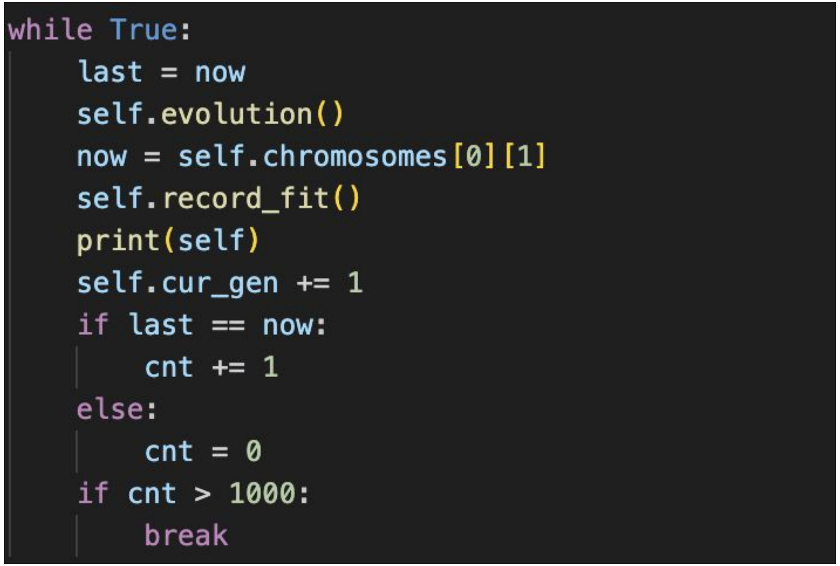
```
offspring = self.static_mutation(offspring,0.05)
```

- Casts mutation by 5% chance

Offspring	Distance
A→F→D→E→C→B	335

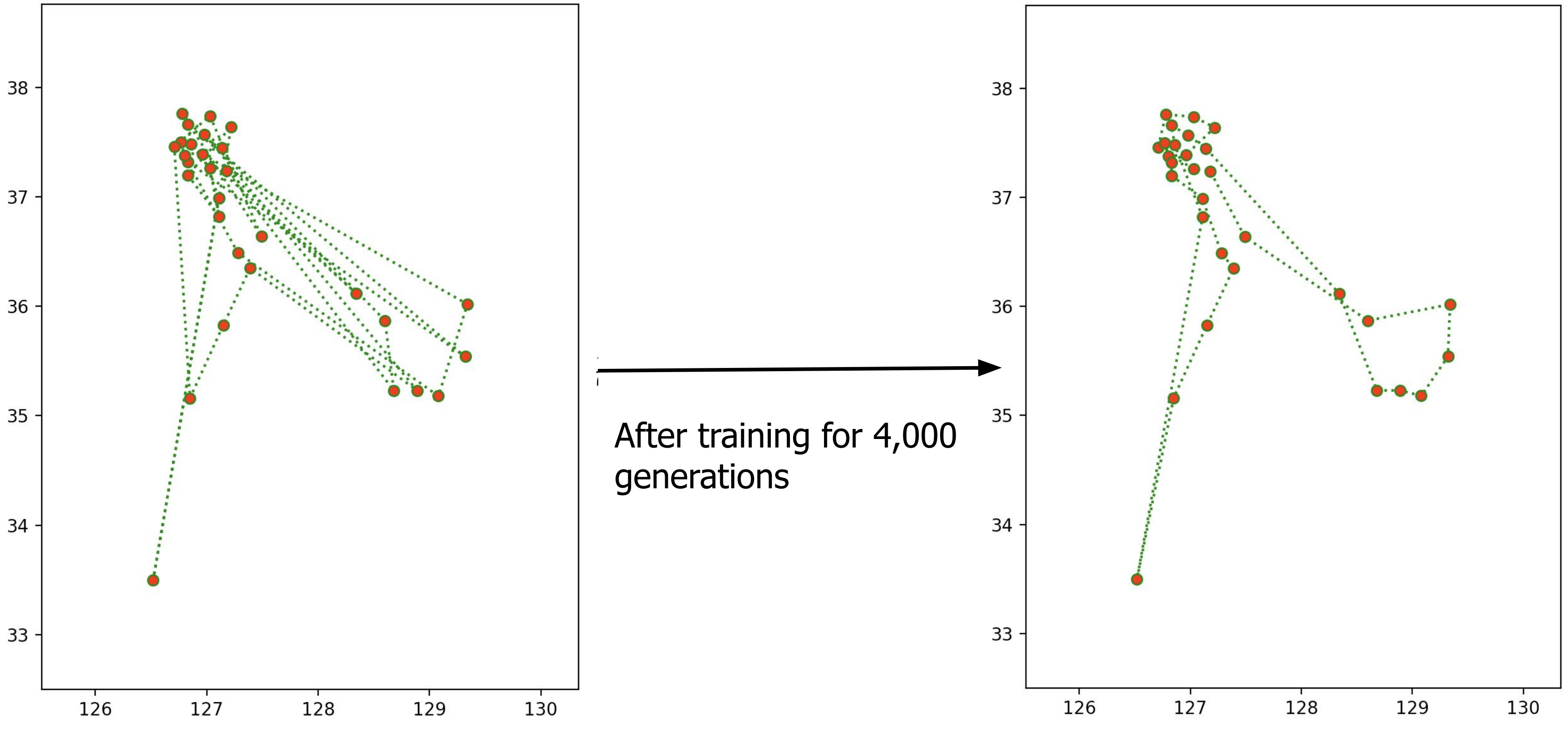
2.Genetic Algorithm

- 2.7 Termination Condition
- When fitness scores does not change for over 1,000 generations



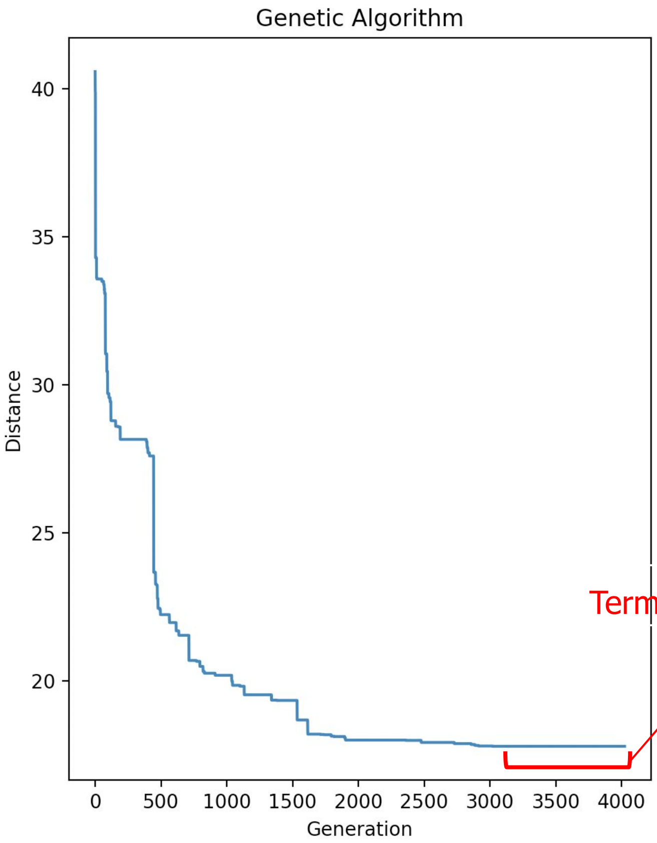
```
while True:
    last = now
    self.evolution()
    now = self.chromosomes[0][1]
    self.record_fit()
    print(self)
    self.cur_gen += 1
    if last == now:
        cnt += 1
    else:
        cnt = 0
    if cnt > 1000:
        break
```

3. Results

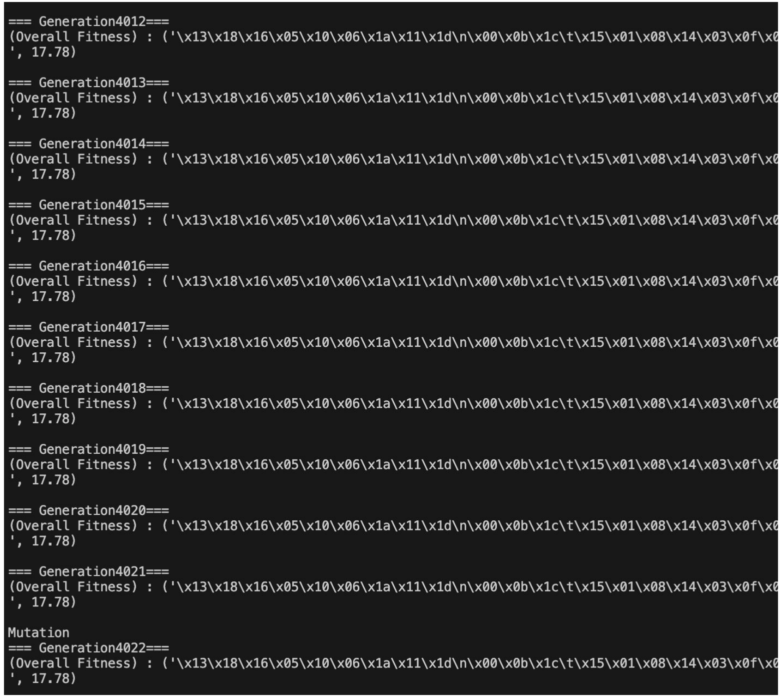


After training for 4,000 generations

3. Results



Termination condition



```
=== Generation4012===
(Overall Fitness) : ('\x13\x18\x16\x05\x10\x06\x1a\x11\x1d\n\x00\x0b\x1c\t\x15\x01\x08\x14\x03\x0f\x0
', 17.78)

=== Generation4013===
(Overall Fitness) : ('\x13\x18\x16\x05\x10\x06\x1a\x11\x1d\n\x00\x0b\x1c\t\x15\x01\x08\x14\x03\x0f\x0
', 17.78)

=== Generation4014===
(Overall Fitness) : ('\x13\x18\x16\x05\x10\x06\x1a\x11\x1d\n\x00\x0b\x1c\t\x15\x01\x08\x14\x03\x0f\x0
', 17.78)

=== Generation4015===
(Overall Fitness) : ('\x13\x18\x16\x05\x10\x06\x1a\x11\x1d\n\x00\x0b\x1c\t\x15\x01\x08\x14\x03\x0f\x0
', 17.78)

=== Generation4016===
(Overall Fitness) : ('\x13\x18\x16\x05\x10\x06\x1a\x11\x1d\n\x00\x0b\x1c\t\x15\x01\x08\x14\x03\x0f\x0
', 17.78)

=== Generation4017===
(Overall Fitness) : ('\x13\x18\x16\x05\x10\x06\x1a\x11\x1d\n\x00\x0b\x1c\t\x15\x01\x08\x14\x03\x0f\x0
', 17.78)

=== Generation4018===
(Overall Fitness) : ('\x13\x18\x16\x05\x10\x06\x1a\x11\x1d\n\x00\x0b\x1c\t\x15\x01\x08\x14\x03\x0f\x0
', 17.78)

=== Generation4019===
(Overall Fitness) : ('\x13\x18\x16\x05\x10\x06\x1a\x11\x1d\n\x00\x0b\x1c\t\x15\x01\x08\x14\x03\x0f\x0
', 17.78)

=== Generation4020===
(Overall Fitness) : ('\x13\x18\x16\x05\x10\x06\x1a\x11\x1d\n\x00\x0b\x1c\t\x15\x01\x08\x14\x03\x0f\x0
', 17.78)

=== Generation4021===
(Overall Fitness) : ('\x13\x18\x16\x05\x10\x06\x1a\x11\x1d\n\x00\x0b\x1c\t\x15\x01\x08\x14\x03\x0f\x0
', 17.78)

Mutation
=== Generation4022===
(Overall Fitness) : ('\x13\x18\x16\x05\x10\x06\x1a\x11\x1d\n\x00\x0b\x1c\t\x15\x01\x08\x14\x03\x0f\x0
', 17.78)
```

Result of training : prints the top fitness score chromosome of each generation