Name: Chenxu Yang

UNI:cy2554

Course name: Evolutionary Computation & Design Automation (MECS E4510)

Instructor: Hod Lipson

Date Submitted:2019/9/28/5PM.

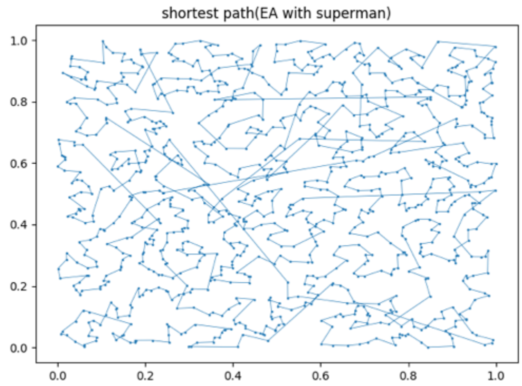
Grace hours used:0

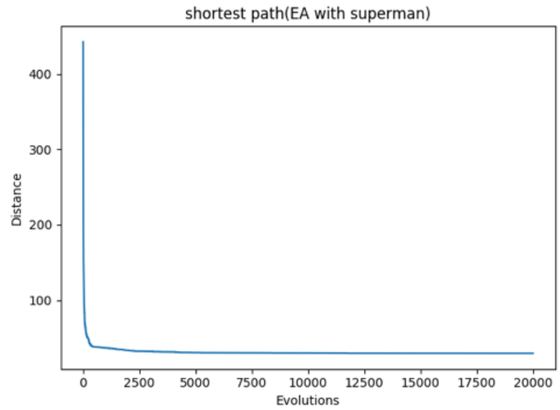
Grace hours remaining:96

Results summary table

Shortest path found

Distance=29.485244

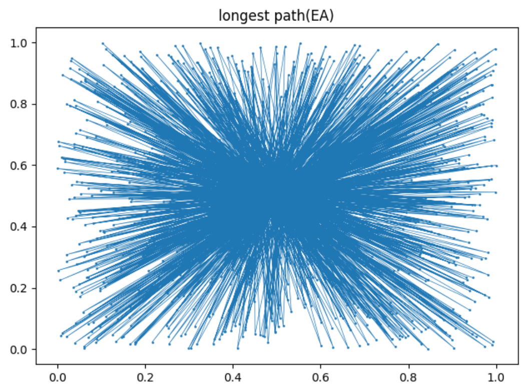


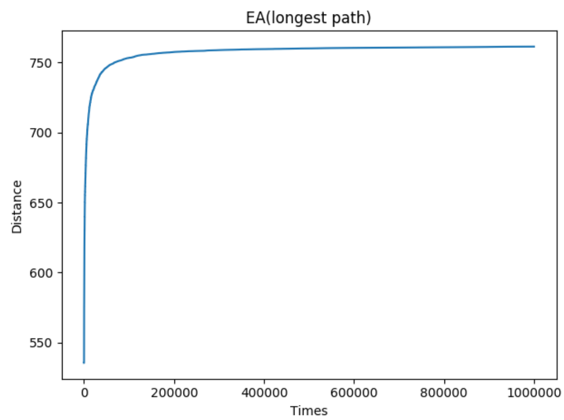


60individuals

Longest path found

Distance= 761.426125





60individuals

Method	evaluations	Shortest length
Rondom search	10000	491.343224
Hill climber	10000000	70.860866
EA	60000000	67.418042
EA with superman	1200000	29.485244

Methods

Random search:

for every time, generate a 1000_long list which represent the visiting sequence of the 1000 cities, calculate the total distance, keep the shortest distance, and draw it. *Poor performance*

Hill climber:

generate a 1000_long list which represent the visiting sequence of the 1000 cities, for every time, switch the sequence of two cities, if the distance is shorter, keep that switch, else switch them back.
Better performance than random search, but stuck at the distance of 70.

Find the nearest:

generate a 1000_long list which represent the visiting sequence of the 1000 cities, start at city[0], always visit the nearest city in the next visit.
great performance, but cannot the best answer.

EA:

1) generate a population which includes 60 individuals, each individual is a random 1000_long list, the list is also called gene
2) Get each individual's fitness(1000/distance), calculate the sum of total fitness and find the best individual (with smallest distance)
3) Generate a new population:
    (1) Cross: select two individuals, take one random part of one individual, and converge it with another individual to get a new individual. For example, take individual1[100,200], then the child become child [100,200], and using the individual2's gene to fix child[0,100] and child[200,1000] without repetition
    (2) Mutation: select an individual, switch the sequence of two genes
    (3) Select: according to every individual's fitness, use Roulette method to select individual
    (4) Always keep the best individual in the previous population

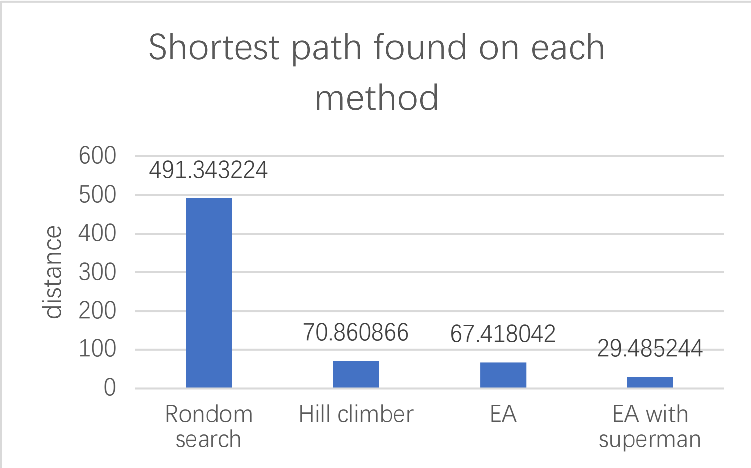
The outcome is a litter better than Hill climber, but cost more time to calculate

EA with superman:

Almost same as EA, except two different ways in initiating population and mutation:

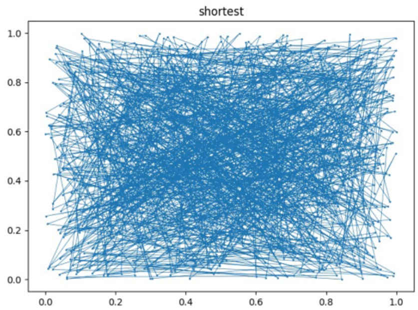
1) When generating a population, there is a chance to generate supermen, which have one 100_long gene generated by Find nearest method.
2) When a child is born, there is a chance for the child to become a superman, which have one 5_long gene generated by Find nearest method.

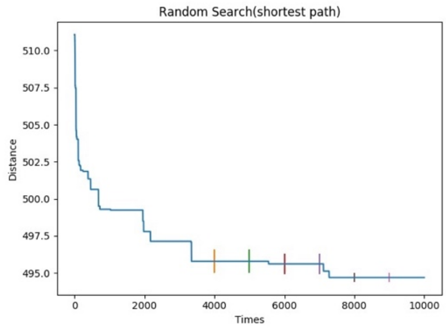
The outcome is much better than the three methods above, and cost less time than EA


Shortest path found on each method
distance
600
500
400
300
200
100
0
491.343224
70.860866
67.418042
29.485244
Rondom search
Hill climber
EA
EA with superman

Learning curves for each method(shortest)

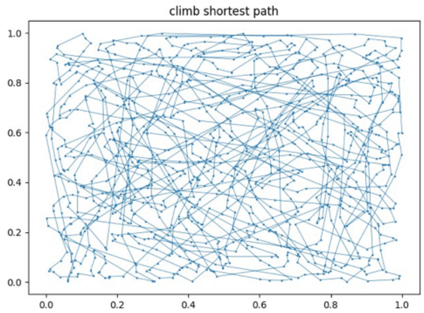
Random search:

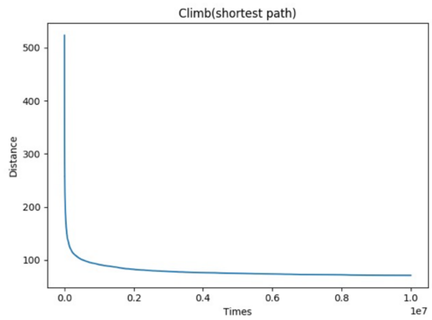




Shortest distance found:491.343224

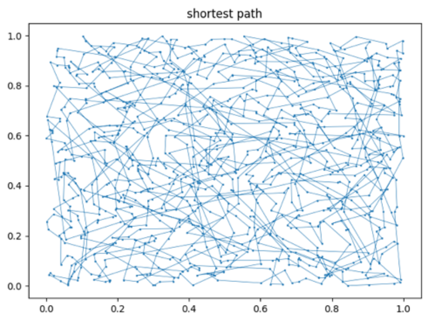
Climb hills

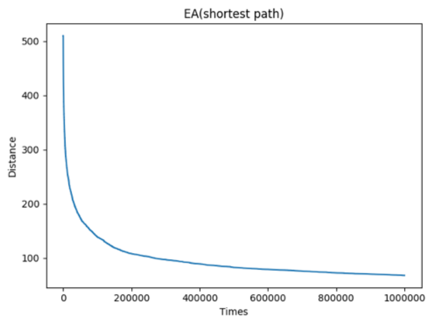




Shortest distance found:70.860866

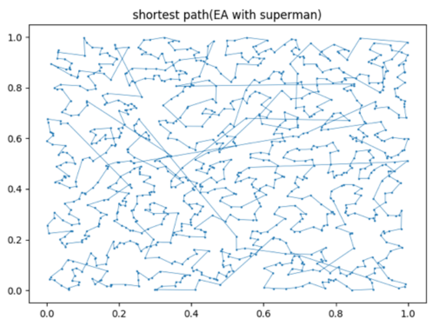
EA

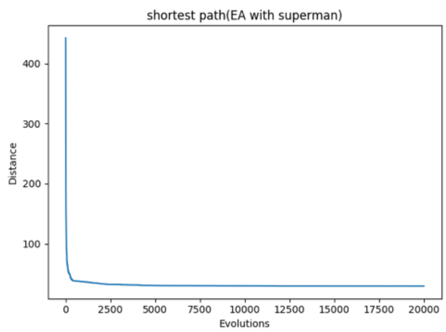




Shortest distance found:67.418042 people=60

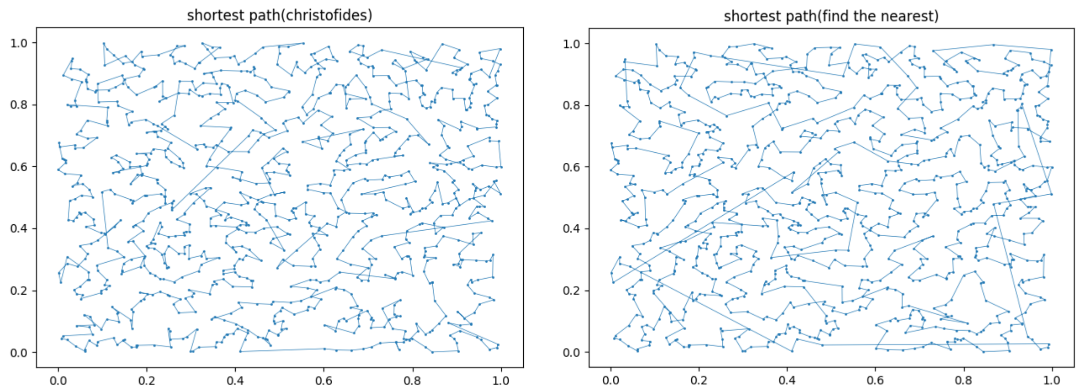
EA with superman





Shortest distance found:29.485244 people=60

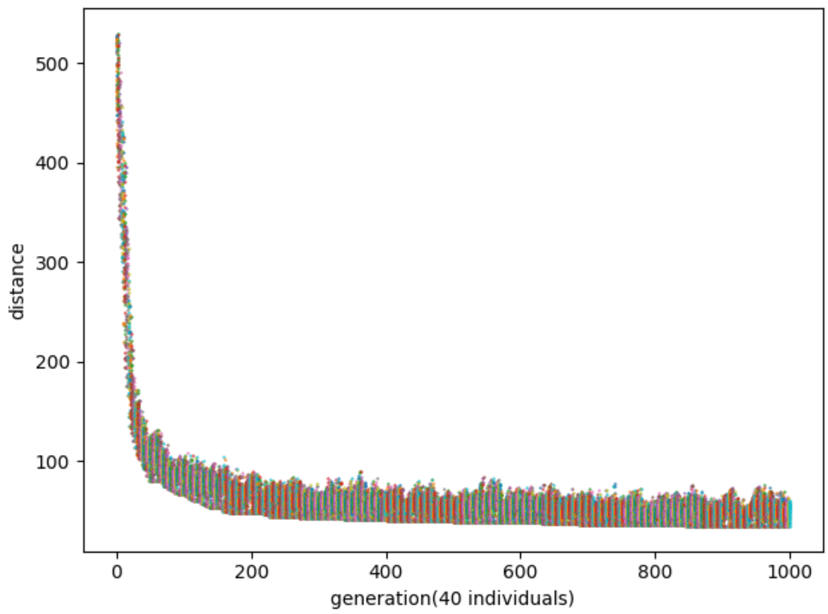
Christofides and Find the nearest



Shortest distance found:27.330767

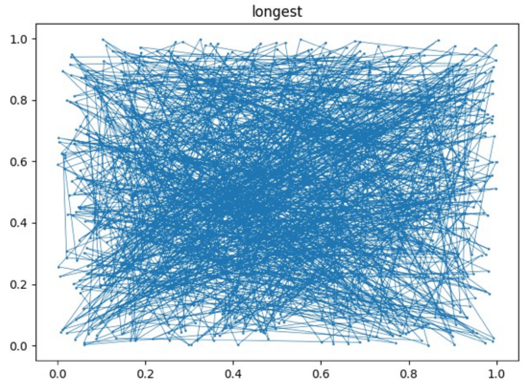
Shortest distance found:27.730631

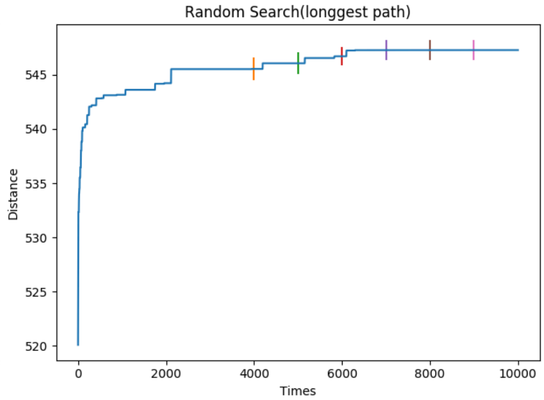
Dot plot for EA (with superman)



Learning curves for each method(longest)

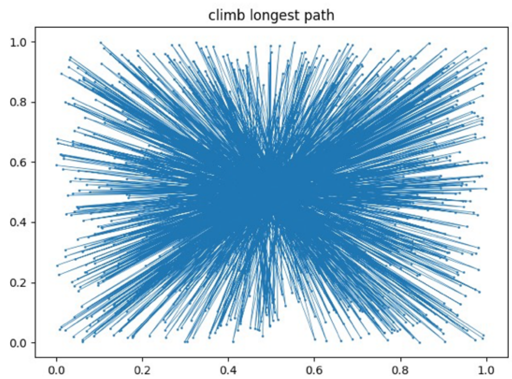
Random search

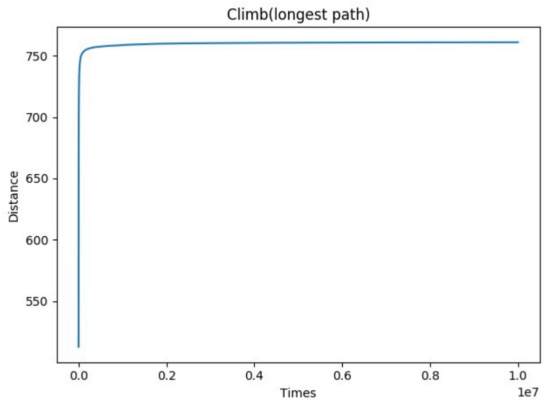




Longest distance found:548.432455

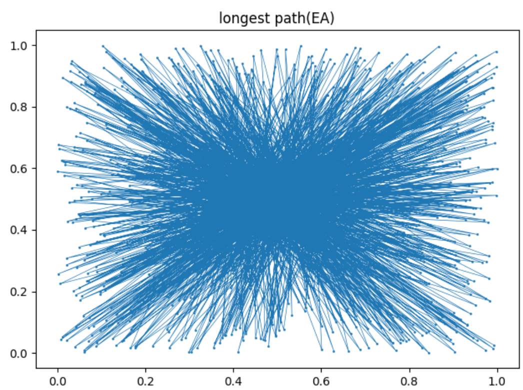
Hill Climber

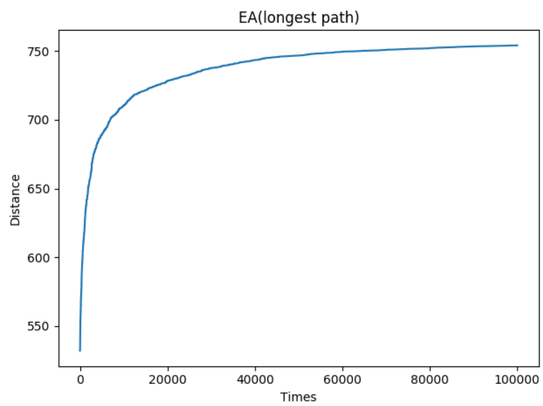




Longest distance found:760.832785 people=60

EA





Longest distance found: 754.087158 people=60

CODE

Code included:

1) EA with superman
2) EA
3) Find the nearest
4) Hill climber

1) EA with superman:

```
import numpy as np
from numba import njit
import random
import matplotlib.pyplot as plt
CORSSRATE=0.5
MUTATIONRATE=0.4
GENERATION=200
GENE_SIZE=1000
POPULATION_SIZE=30
SUPERGENERATE=0.5
a = np.loadtxt('tsp.txt')
CITY_POSITION = np.reshape(a, (1000, 2))
DOT=[]

def make_population():
    population=np.vstack([np.random.permutation(GENE_SIZE) for _ in
range(POPULATION_SIZE)])
    for i in range(0,len(population),2):
        population[i]=super_man(population[i])
    return population

def super_man(individual):
    index1 = random.randint(0, GENE_SIZE - 100)  # randomly get the mutation position #
    #index2 = random.randint(index1, GENE_SIZE - 1)
    index2=index1+99
    crossgene = []
    superman=[]
    crossgene.append(individual[index1])
    for i in range(0,index2-index1):
        crossgene.append(individual[nearest(crossgene,individual,crossgene[i])])
    for flag, item in enumerate(individual):
        if flag == index1:
            superman.extend(crossgene)
```

```
        if item not in crossgene:
            superman.append(item)
    return superman

def super_man1(individual):
    index1 = random.randint(0, GENE_SIZE - 6)  # randomly get the mutation position #
    #index2 = random.randint(index1, GENE_SIZE - 1)
    index2=index1+5
    crossgene = []
    superman=[]
    crossgene.append(individual[index1])
    for i in range(0,index2-index1):
        crossgene.append(individual[nearest(crossgene,individual,crossgene[i])])
    for flag, item in enumerate(individual):
        if flag == index1:
            superman.extend(crossgene)
        if item not in crossgene:
            superman.append(item)
    return superman
@njit
def nearest(crossgene,individual,i):
    best_distance=100
    best_index=-1
    for index,item in enumerate(individual):
        if item not in crossgene:
            distance = ((CITY_POSITION[individual[index]][0] - CITY_POSITION[i][0]) ** 2
+
                        (CITY_POSITION[individual[index]][1] - CITY_POSITION[i][1]) ** 2)
** 0.5
            if distance < best_distance:
                best_distance = distance
                best_index = index
    return best_index

#@njit
def fitness(population,y):
    best_individual=population[0]
    best_fitness=getfitness(best_individual)
    fitness_list=[0]*POPULATION_SIZE
    sumfitness=0
    for i,individual in enumerate(population):
        fitness_list[i]=getfitness(individual)
        plt.scatter(y,1000/fitness_list[i] ,marker='.',s=1)
        plt.xlabel('generation')
```

```
        plt.ylabel('distance')
        if fitness_list[i]>best_fitness:
            best_individual=individual
            best_fitness=fitness_list[i]
        sumfitness+=fitness_list[i]
    return sumfitness,best_individual,fitness_list,best_fitness
@njit
def getfitness(individual):
    distance=0.000000
    for i in range(1,len(individual)):
        distance += ((CITY_POSITION[individual[i]][0] - CITY_POSITION[individual[i-
1]][0]) ** 2 +
                     (CITY_POSITION[individual[i]][1] - CITY_POSITION[individual[i-
1]][1]) ** 2)**0.5

    return 1000/distance
#@njit
def cross(mom,dad):
    index1 = random.randint(0, GENE_SIZE - 3)  # randomly get the mutation position #
    index2 = random.randint(index1+1, GENE_SIZE - 1)
    crossgene = []
    for i in range(index1, index2):
        crossgene.append(mom[i])
    child=[]
    for flag,item in enumerate(dad):
        if flag==index1:
            child.extend(crossgene)
        if item not in crossgene:
            child.append(item)
    return child

def mutaion(individual):
    index = random.sample(range(0, GENE_SIZE), 2)
    new_individual=individual[:]

new_individual[index[0]],new_individual[index[1]]=new_individual[index[1]],new_individ
ual[index[0]]
    return new_individual
def select(population,sumfitness):
    flag=random.uniform(0,sumfitness)
    for individual in population:
        flag-=getfitness(individual)
        if flag<=0:
            return individual
```

```python
def born(population,sumfitness):
    dad=select(population,sumfitness)
    while dad is None:
        dad = select(population, sumfitness)
    mom = select(population, sumfitness)
    while dad is None:
        mom=select(population,sumfitness)
    rate = random.uniform(0, 1)
    if rate < CORSSRATE:
        child = cross(dad, mom)
    else:
        child = dad
    rate = random.uniform(0, 1)
    if rate < MUTATIONRATE:
        child = mutaion(child)

    return child
#@njit
def generation(population,y):
    sumfitness,best_individual,fitness_list,best_fitness=fitness(population,y)
    new_population=[]
    new_population.append(best_individual)
    while len(new_population) < POPULATION_SIZE:
        rate = random.uniform(0, 1)
        if rate<0.3:
            new_population.append(super_man(born(population,sumfitness)))
        else:
            new_population.append(super_man1(born(population, sumfitness)))
    return new_population
    #generation_times += 1

def draw(individual):
    x=[0]*(GENE_SIZE+1)
    y=[0]*(GENE_SIZE+1)
    for i in range(0,GENE_SIZE):
        x[i]=CITY_POSITION[individual[i]][0]
    x[GENE_SIZE]=x[0]
    for i in range(0,GENE_SIZE):
        y[i]=CITY_POSITION[individual[i]][1]
    y[GENE_SIZE] = y[0]
    fig_short = plt.figure()
    ax1 = fig_short.add_subplot(1, 1, 1)
    ax1.plot(x, y, lw=0.5, marker='o', markersize=1, mfc='w')
```

```
    ax1.set_title('shortest path(EA with superman)')
    plt.show()
if __name__=='__main__':
    population=make_population()
    i=GENERATION
    shortest_distance=[]
    while i>0:
        sumfitness, best_individual,
fitness_list,best_fitness=fitness(population,GENERATION-i)
        print("distance:%f,  generation:%d"%(1000/best_fitness,GENERATION-i))
        shortest_distance.append(1000/best_fitness)
        population=generation(population,GENERATION-i)
        i-=1
    draw(best_individual)
    np.savetxt('best_individual.txt',best_individual)
    np.savetxt('short_distance.txt', shortest_distance)
    fig_shortest = plt.figure()
    ax1 = fig_shortest.add_subplot(1, 1, 1)
    times=[i for i in range(GENERATION)]
    ax1.plot(times, shortest_distance)
    ax1.set_title('shortest path(EA with superman)')
    ax1.set_xlabel('Evolutions')
    ax1.set_ylabel('Distance')
    plt.show()
```

2) EA:

```
import numpy as np
from numba import njit
import random
import matplotlib.pyplot as plt
CORSSRATE=0.5
MUTATIONRATE=0.4
GENERATION=700000
GENE_SIZE=1000
POPULATION_SIZE=15
a = np.loadtxt('tsp.txt')
CITY_POSITION = np.reshape(a, (1000, 2))


def make_population():
    population=np.vstack([np.random.permutation(GENE_SIZE) for _ in
range(POPULATION_SIZE)])
```

```
    return population
#@njit
def fitness(population):
    best_individual=population[0]
    best_fitness=getfitness(best_individual)
    fitness_list=[0]*POPULATION_SIZE
    sumfitness=0
    for i,individual in enumerate(population):
        fitness_list[i]=getfitness(individual)
        if fitness_list[i]>best_fitness:
            best_individual=individual
            best_fitness=fitness_list[i]
        sumfitness+=fitness_list[i]
    return sumfitness,best_individual,fitness_list,best_fitness
@njit
def getfitness(individual):
    distance=0.000000
    for i in range(1,len(individual)):
        distance += ((CITY_POSITION[individual[i]][0] - CITY_POSITION[individual[i-
1]][0]) ** 2 +
                    (CITY_POSITION[individual[i]][1] - CITY_POSITION[individual[i-
1]][1]) ** 2)**0.5
    return 1000/distance
@njit
def cross(mom,dad):
    index1 = random.randint(0, GENE_SIZE - 1)
    index2 = random.randint(index1, GENE_SIZE - 1)
    crossgene = []
    for i in range(index1, index2):
        crossgene.append(mom[i])
    child=[]
    for flag,item in enumerate(dad):
        if flag==index1:
            child.extend(crossgene)
        if item not in crossgene:
            child.append(item)
    return child

def mutaion(individual):
    index = random.sample(range(0, GENE_SIZE), 2)
    new_individual=individual[:]

new_individual[index[0]],new_individual[index[1]]=new_individual[index[1]],new_individ
ual[index[0]]
```

```python
    return new_individual


def select(population,sumfitness):
    flag=random.uniform(0,sumfitness)
    for individual in population:
        flag-=getfitness(individual)
        if flag<=0:
            return individual
def born(population,sumfitness):
    dad=select(population,sumfitness)
    mom=select(population,sumfitness)
    rate = random.uniform(0, 1)
    if rate < CORSSRATE:
        child = cross(dad, mom)
    else:
        child = dad
    rate = random.uniform(0, 1)
    if rate < MUTATIONRATE:
        child = mutaion(child)
    return child
#@njit
def generation(population):
    sumfitness,best_individual,fitness_list,best_fitness=fitness(population)
    new_population=[]
    new_population.append(best_individual)
    while len(new_population) < POPULATION_SIZE:
        new_population.append(born(population,sumfitness))
    return new_population
    #generation_times += 1


def draw(individual):
    x=[0]*GENE_SIZE
    y=[0]*GENE_SIZE
    for i in range(0,GENE_SIZE):
        x[i]=CITY_POSITION[individual[i]][0]
    for i in range(0,GENE_SIZE):
        y[i]=CITY_POSITION[individual[i]][1]
    fig_short = plt.figure()
    ax1 = fig_short.add_subplot(1, 1, 1)
    ax1.plot(x, y, lw=0.5, marker='o', markersize=1, mfc='w')
    ax1.set_title('shortest path')
    plt.show()
if __name__=='__main__':
    population=make_population()
```

```
    i=GENERATION
    shortest_distance=[]
    while i>0:
        sumfitness, best_individual, fitness_list,best_fitness=fitness(population)
        print("distance:%f,  generation:%d"%(1000/best_fitness,GENERATION-i))
        shortest_distance.append(1000/best_fitness)
        population=generation(population)
        i-=1
    draw(best_individual)
    np.savetxt('best_individual.txt1',best_individual)
    np.savetxt('short_distance.txt1', shortest_distance)
    fig_shortest = plt.figure()
    ax1 = fig_shortest.add_subplot(1, 1, 1)
    times=[i for i in range(GENERATION)]
    ax1.plot(times, shortest_distance)
    ax1.set_title('EA(shortest path)')
    ax1.set_xlabel('Times')
    ax1.set_ylabel('Distance')
    plt.show()
```

3) Find the nearest

```
import numpy as np
import datetime
from numba import njit
import random
import matplotlib.pyplot as plt
GENERATION=10
GENE_SIZE=1000
a = np.loadtxt('tsp.txt')
CITY_POSITION = np.reshape(a, (1000, 2))
@njit
def nearest(crossgene,individual,i):
    best_distance=100
    best_index=-1
    for index,item in enumerate(individual):
        if item not in crossgene:
            distance = ((CITY_POSITION[individual[index]][0] -
CITY_POSITION[individual[i]][0]) ** 2 +
                     (CITY_POSITION[individual[index]][1] -
CITY_POSITION[individual[i]][1]) ** 2) ** 0.5
            if distance < best_distance:
                best_distance = distance
                best_index = index
    return best_index
```

```
@njit
def distance(i,j):
    return ((CITY_POSITION[START[i]][0]-CITY_POSITION[START[j]][0])**2+
          (CITY_POSITION[START[i]][1]-CITY_POSITION[START[j]][1])**2)**0.5
@njit
def getfitness(individual):

    distance=0.000000
    for i in range(1,len(individual)):
        distance += ((CITY_POSITION[individual[i]][0] - CITY_POSITION[individual[i-
1]][0]) ** 2 +
                     (CITY_POSITION[individual[i]][1] - CITY_POSITION[individual[i-
1]][1]) ** 2)**0.5
    return distance
def draw(individual):
    x=[0]*1001
    y=[0]*1001
    for i in range(0,GENE_SIZE):
        x[i]=CITY_POSITION[individual[i]][0]
    x[0]=CITY_POSITION[individual[0]][0]
    for i in range(0,GENE_SIZE):
        y[i]=CITY_POSITION[individual[i]][1]
    y[0]=CITY_POSITION[individual[0]][1]
    fig_short = plt.figure()
    ax1 = fig_short.add_subplot(1, 1, 1)
    ax1.plot(x, y, lw=0.5, marker='o', markersize=1, mfc='w')
    ax1.set_title('shortest path(find the nearest)')
    plt.show()
if __name__=='__main__':
    START = [i for i in range(0, 1000)]
    best_gene=[]
    best_gene.append(START[0])
    for i in range(0,1000-1):
        best_gene.append(nearest(best_gene,START,best_gene[i]))
    print(best_gene)
    print(getfitness(best_gene))
    draw(best_gene)
```

4) Hill Climber

```
import numpy as np
from numba import njit
import random
import matplotlib.pyplot as plt
```

```
a = np.loadtxt('tsp.txt')
CITY_POSITION = np.reshape(a, (1000, 2))


@njit
def getfitness(individual):
    distance=0.000000
    for i in range(1,len(individual)):
        distance += ((CITY_POSITION[individual[i]][0] - CITY_POSITION[individual[i-
1]][0]) ** 2 +
                    (CITY_POSITION[individual[i]][1] - CITY_POSITION[individual[i-
1]][1]) ** 2)**0.5
    return distance
def draw(individual):
    x=[0]*1001
    y=[0]*1001
    for i in range(0,1000):
        x[i]=CITY_POSITION[individual[i]][0]
    x[1000]=CITY_POSITION[individual[0]][0]
    for i in range(0,1000):
        y[i]=CITY_POSITION[individual[i]][1]
    y[1000] = CITY_POSITION[individual[0]][1]
    fig_short = plt.figure()
    ax1 = fig_short.add_subplot(1, 1, 1)
    ax1.plot(x, y, lw=0.5, marker='o', markersize=1, mfc='w')
    ax1.set_title('climb shortest path')
    plt.show()

def mutation(individual,distance,i):
    index1 = random.randint(0, 1000-1)
    index2 = random.randint(0, 1000-1)
    new_individual1 = individual[:]
    new_individual1[index1], new_individual1[index2] = new_individual1[index2],
new_individual1[index1]
    a=getfitness(new_individual1)
    plt.scatter(i, a, alpha=0.6)
    if a>distance:
        new_individual1[index1], new_individual1[index2] = new_individual1[index2],
new_individual1[index1]
    return new_individual1

if __name__=='__main__':

    shortest_distance=[]
```

```python
fig_shortest = plt.figure()
ax1 = fig_shortest.add_subplot(1, 1, 1)
start=np.random.permutation(1000)
generation=10000
best_individual=start
shortest=getfitness(best_individual)
i=1
while(generation>0):
    new=mutation(best_individual,shortest,i)
    best_individual=new
    shortest=getfitness(best_individual)
    shortest_distance.append(shortest)
    print("distance: %f.  generation:%d"%(shortest,i))
    i+=1
    generation-=1

draw(best_individual)
plt.show()
```