Lab3 – AI

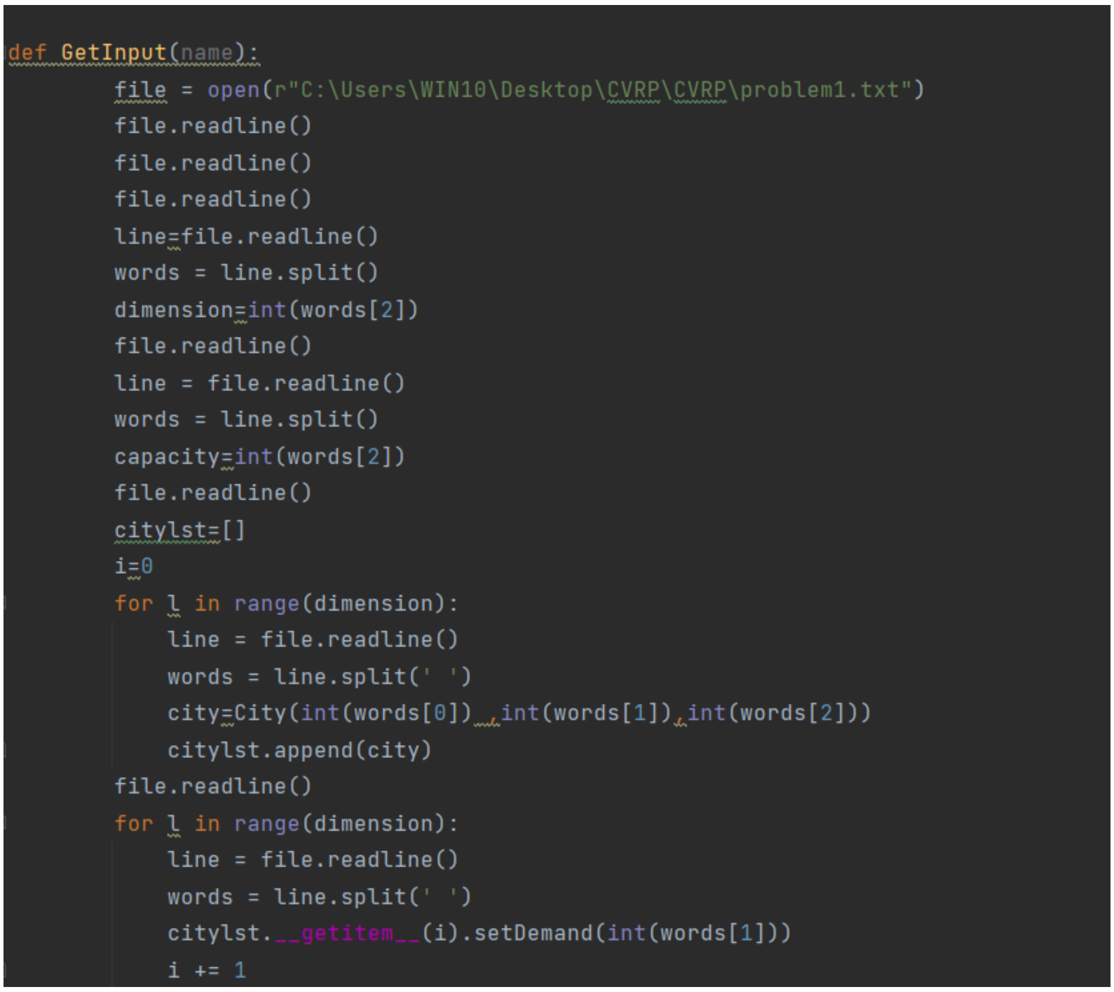
Zinat Abo Hgool   206721714

Elie Hadad   207931536

How we translated the input:

In order to implement our algorithms according to the input samples we have got, we added a new file 'GetInput' that help us with taking the relevant information from the samples file and assign the propitiate values to the arguments.

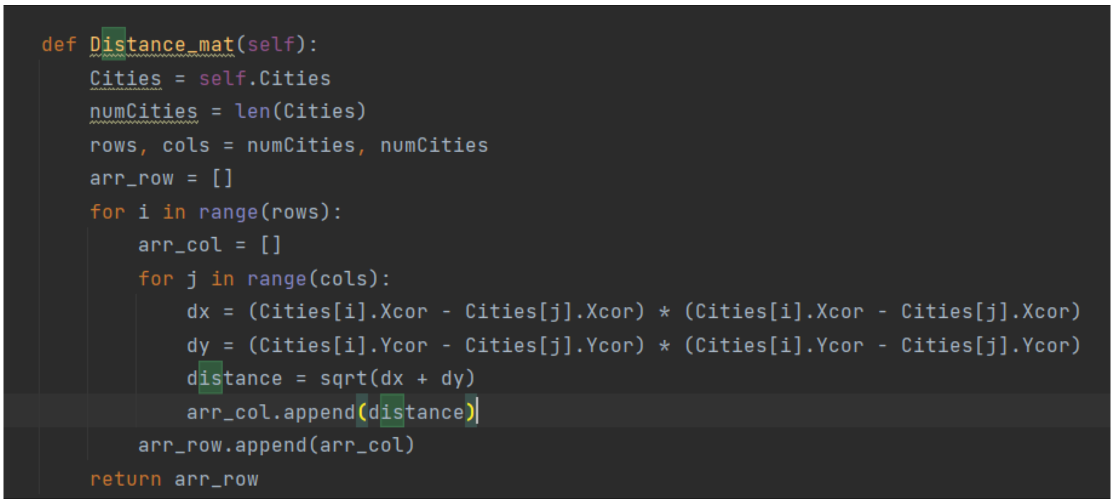
Here is a little bit of our code: (you can check the source file for the full code)



```python
def GetInput(name):
        file = open(r"C:\Users\WIN10\Desktop\CVRP\CVRP\problem1.txt")
        file.readline()
        file.readline()
        file.readline()
        line=file.readline()
        words = line.split()
        dimension=int(words[2])
        file.readline()
        line = file.readline()
        words = line.split()
        capacity=int(words[2])
        file.readline()
        citylst=[]
        i=0
        for l in range(dimension):
            line = file.readline()
            words = line.split(' ')
            city=City(int(words[0]),int(words[1]),int(words[2]))
            citylst.append(city)
        file.readline()
        for l in range(dimension):
            line = file.readline()
            words = line.split(' ')
            citylst.__getitem__(i).setDemand(int(words[1]))
            i += 1
```

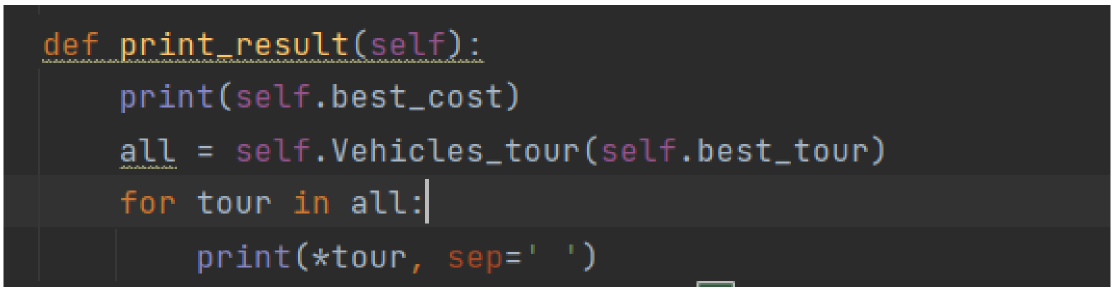
In order to translate the problem that was given in the assignment "CVRP", we added new file 'CVRP' which included the following functions:

1)function to calculate the distance between two cities in the map and to place this distance in the propriate indices in the distance matrix.



```
def Distance_mat(self):
    Cities = self.Cities
    numCities = len(Cities)
    rows, cols = numCities, numCities
    arr_row = []
    for i in range(rows):
        arr_col = []
        for j in range(cols):
            dx = (Cities[i].Xcor - Cities[j].Xcor) * (Cities[i].Xcor - Cities[j].Xcor)
            dy = (Cities[i].Ycor - Cities[j].Ycor) * (Cities[i].Ycor - Cities[j].Ycor)
            distance = sqrt(dx + dy)
            arr_col.append(distance)
        arr_row.append(arr_col)
    return arr_row
```

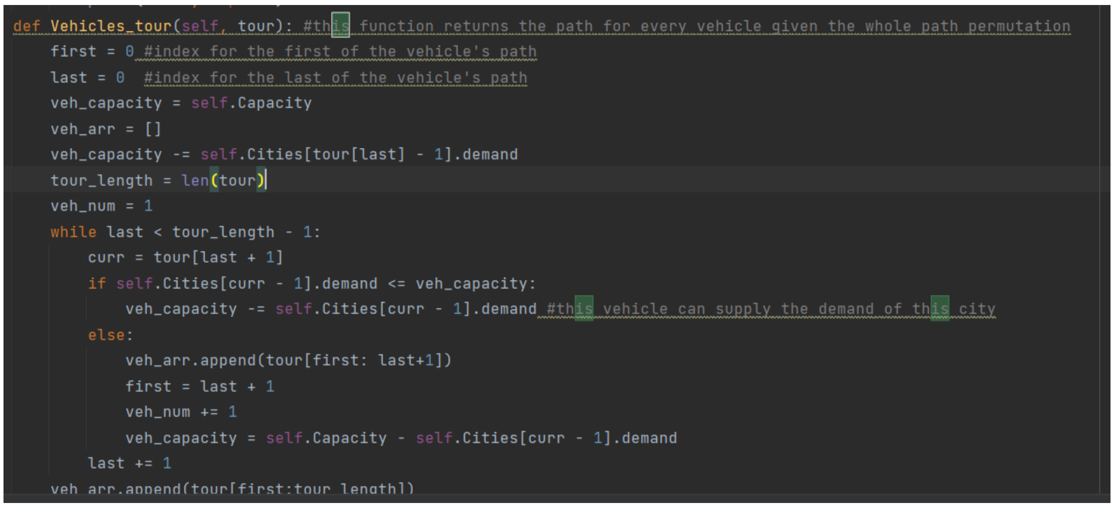
2) function to print the output in the specific order(matrix) as the instruction given in the assignment



```
def print_result(self):
    print(self.best_cost)
    all = self.Vehicles_tour(self.best_tour)
    for tour in all:
        print(*tour, sep=' ')
```

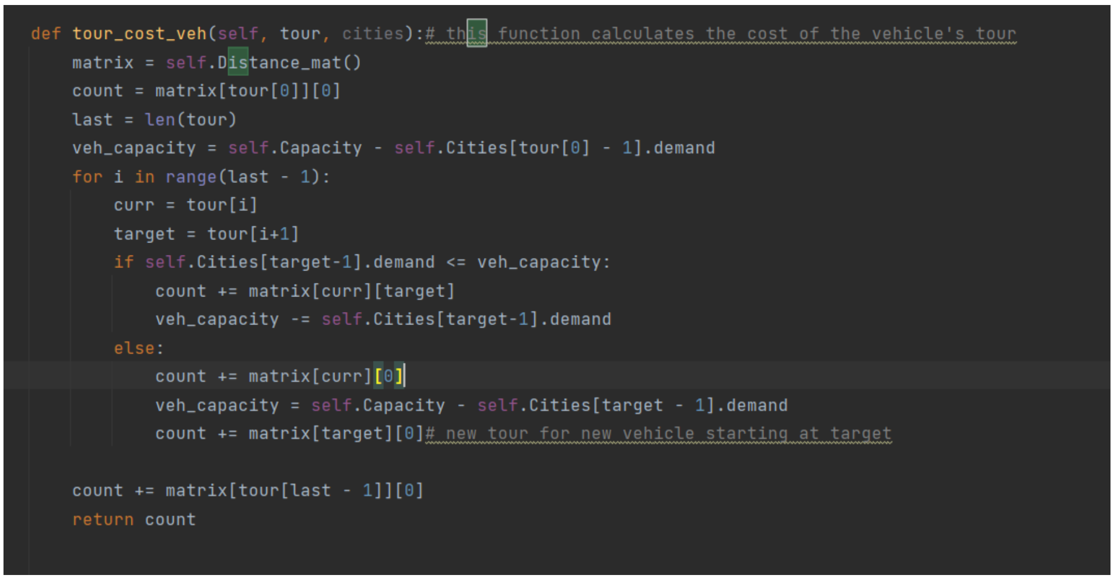
3) function that returns the path for every vehicle given the whole path permutation.

Here is a little bit of the code:



```python
def Vehicles_tour(self, tour): #this function returns the path for every vehicle given the whole path permutation
    first = 0 #index for the first of the vehicle's path
    last = 0  #index for the last of the vehicle's path
    veh_capacity = self.Capacity
    veh_arr = []
    veh_capacity -= self.Cities[tour[last] - 1].demand
    tour_length = len(tour)
    veh_num = 1
    while last < tour_length - 1:
        curr = tour[last + 1]
        if self.Cities[curr - 1].demand <= veh_capacity:
            veh_capacity -= self.Cities[curr - 1].demand #this vehicle can supply the demand of this city
        else:
            veh_arr.append(tour[first: last+1])
            first = last + 1
            veh_num += 1
            veh_capacity = self.Capacity - self.Cities[curr - 1].demand
        last += 1
    veh_arr.append(tour[first:tour_length])
```

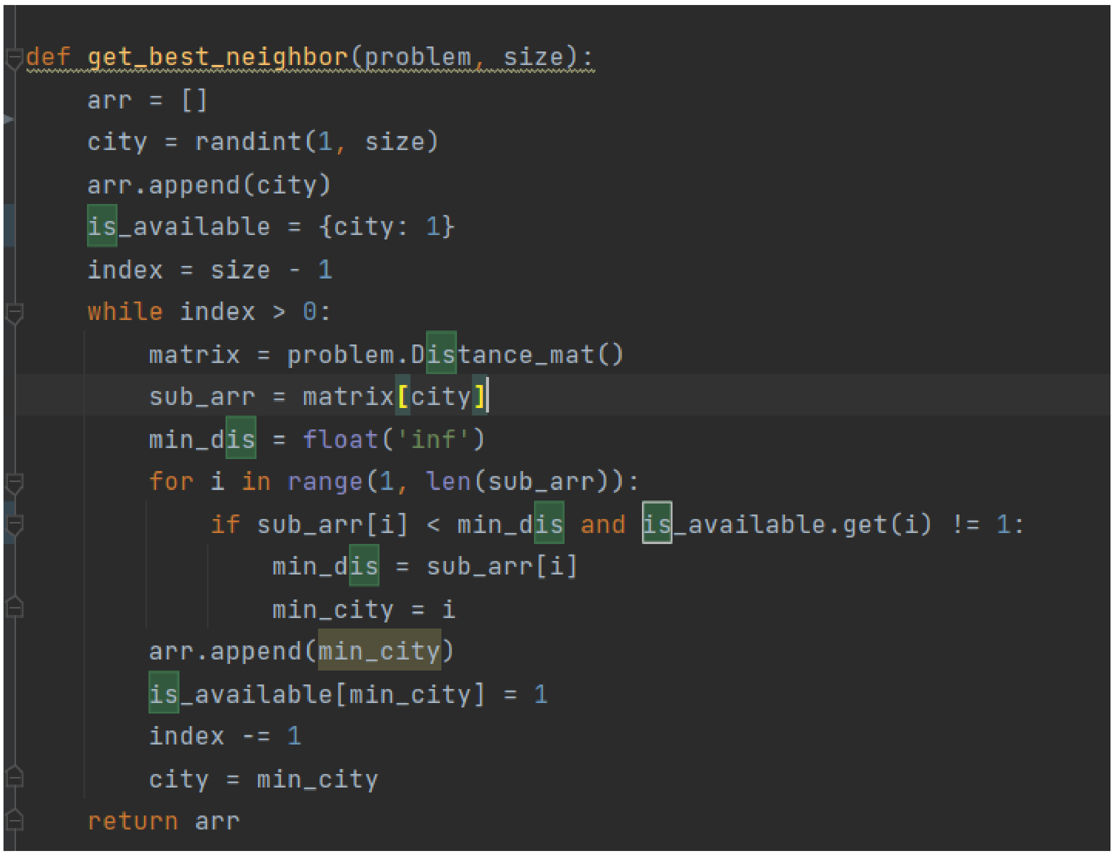
4) function that calculates the cost of a given a tour



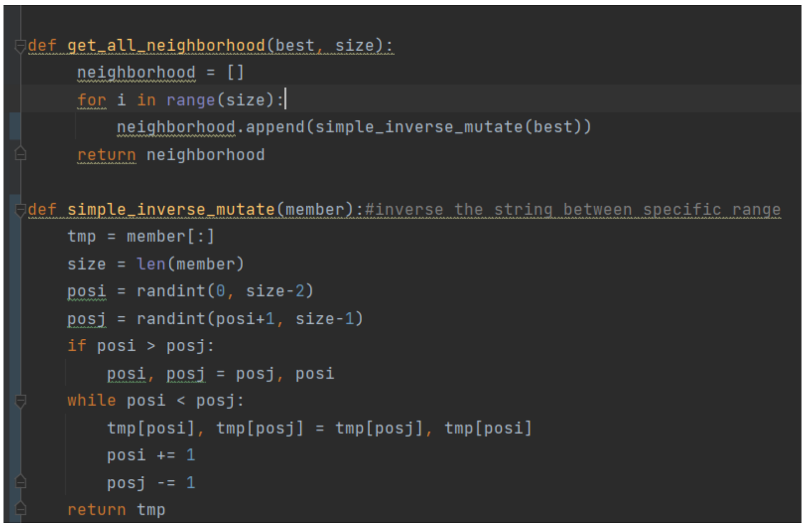
```python
def tour_cost_veh(self, tour, cities):# this function calculates the cost of the vehicle's tour
    matrix = self.Distance_mat()
    count = matrix[tour[0]][0]
    last = len(tour)
    veh_capacity = self.Capacity - self.Cities[tour[0] - 1].demand
    for i in range(last - 1):
        curr = tour[i]
        target = tour[i+1]
        if self.Cities[target-1].demand <= veh_capacity:
            count += matrix[curr][target]
            veh_capacity -= self.Cities[target-1].demand
        else:
            count += matrix[curr][0]
            veh_capacity = self.Capacity - self.Cities[target - 1].demand
            count += matrix[target][0]# new tour for new vehicle starting at target

    count += matrix[tour[last - 1]][0]
    return count
```

Heuristics we used:

We added new file named 'heuristic' with in it we declared the nearest neighbor heuristic that we learned, using this heuristic we select at first random member of the permutation and apply inverse mutation that we used in previous lab, here is a little bit of the code:



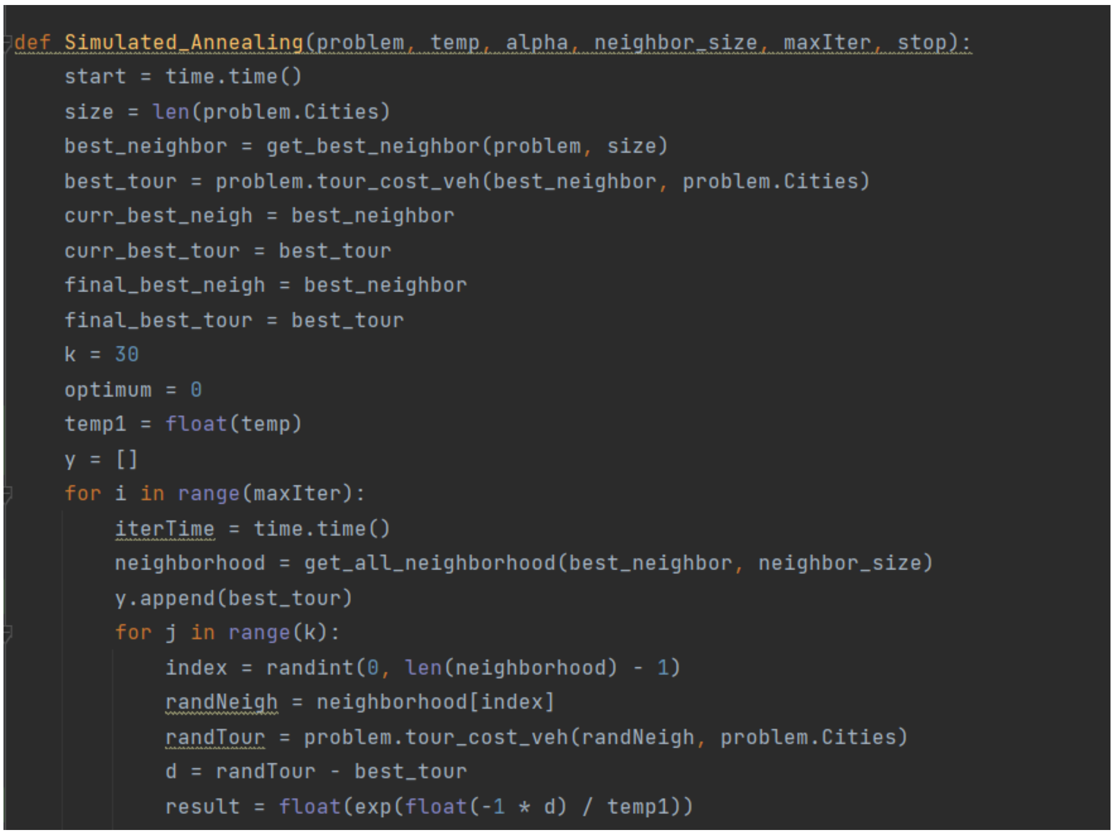
```python
def get_best_neighbor(problem, size):
    arr = []
    city = randint(1, size)
    arr.append(city)
    is_available = {city: 1}
    index = size - 1
    while index > 0:
        matrix = problem.Distance_mat()
        sub_arr = matrix[city]
        min_dis = float('inf')
        for i in range(1, len(sub_arr)):
            if sub_arr[i] < min_dis and is_available.get(i) != 1:
                min_dis = sub_arr[i]
                min_city = i
        arr.append(min_city)
        is_available[min_city] = 1
        index -= 1
        city = min_city
    return arr
```



```python
def get_all_neighborhood(best, size):
    neighborhood = []
    for i in range(size):
        neighborhood.append(simple_inverse_mutate(best))
    return neighborhood

def simple_inverse_mutate(member):#inverse the string between specific range
    tmp = member[:]
    size = len(member)
    posi = randint(0, size-2)
    posj = randint(posi+1, size-1)
    if posi > posj:
        posi, posj = posj, posi
    while posi < posj:
        tmp[posi], tmp[posj] = tmp[posj], tmp[posi]
        posi += 1
        posj -= 1
    return tmp
```

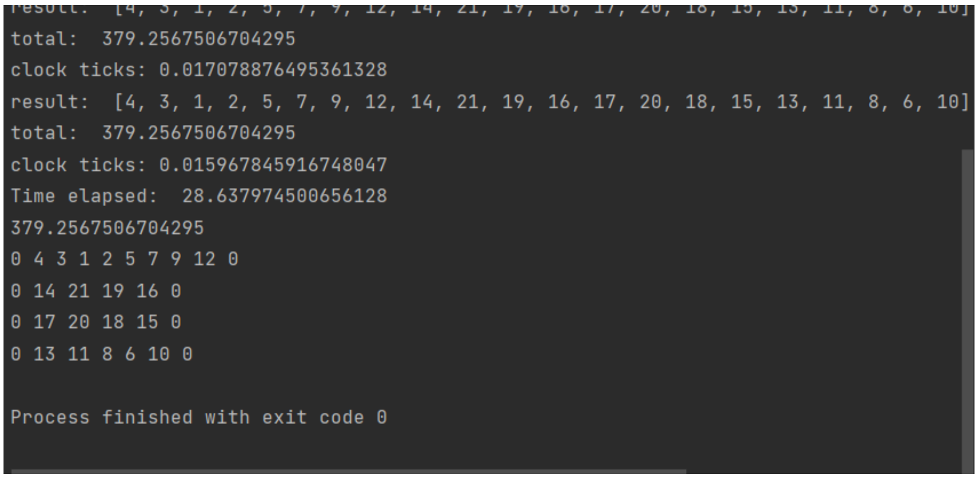
Simulated Annealing:

Here is a little bit of our code implementing the simulated annealing algorithm:



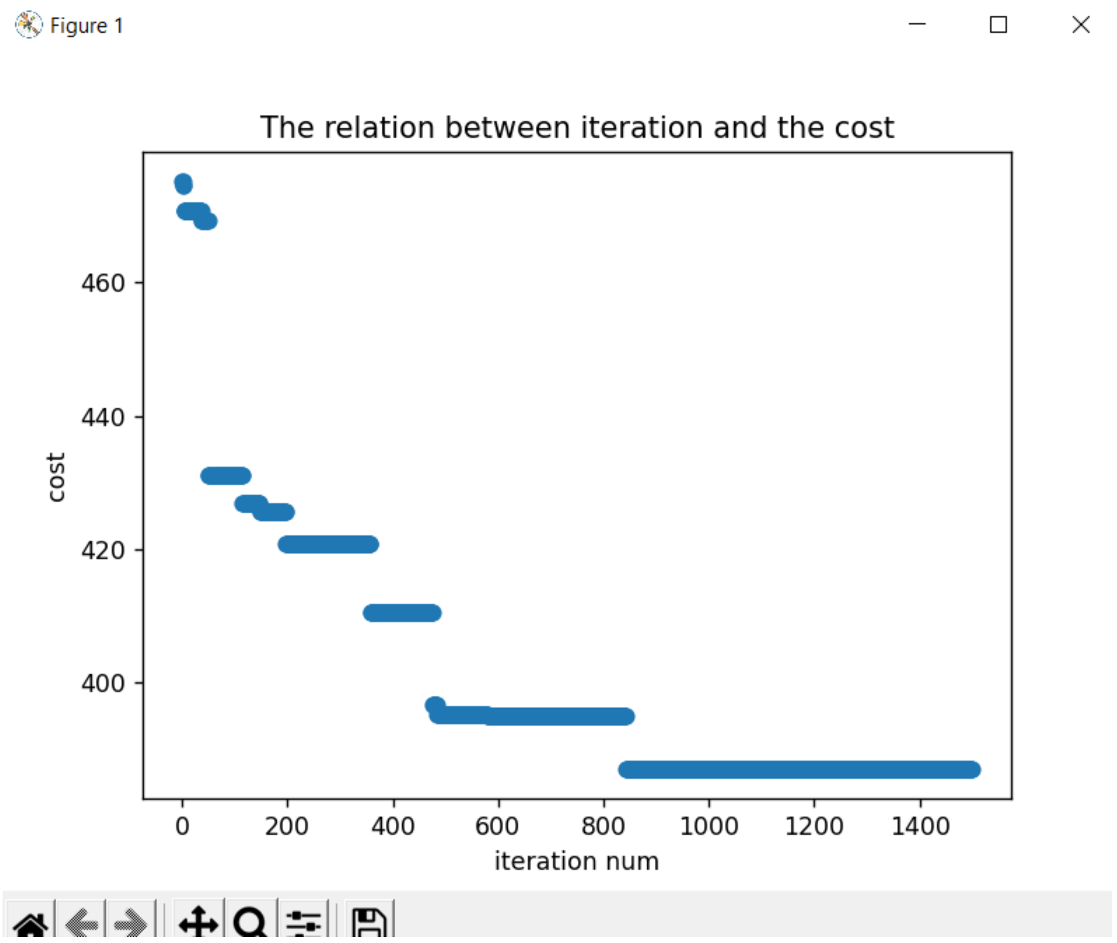
```python
def Simulated_Annealing(problem, temp, alpha, neighbor_size, maxIter, stop):
    start = time.time()
    size = len(problem.Cities)
    best_neighbor = get_best_neighbor(problem, size)
    best_tour = problem.tour_cost_veh(best_neighbor, problem.Cities)
    curr_best_neigh = best_neighbor
    curr_best_tour = best_tour
    final_best_neigh = best_neighbor
    final_best_tour = best_tour
    k = 30
    optimum = 0
    temp1 = float(temp)
    y = []
    for i in range(maxIter):
        iterTime = time.time()
        neighborhood = get_all_neighborhood(best_neighbor, neighbor_size)
        y.append(best_tour)
        for j in range(k):
            index = randint(0, len(neighborhood) - 1)
            randNeigh = neighborhood[index]
            randTour = problem.tour_cost_veh(randNeigh, problem.Cities)
            d = randTour - best_tour
            result = float(exp(float(-1 * d) / temp1))
```

And when we run our code on the 'sample 1' file (E-n22-k4) we have got the following output, with the following elapsed and CPU time:



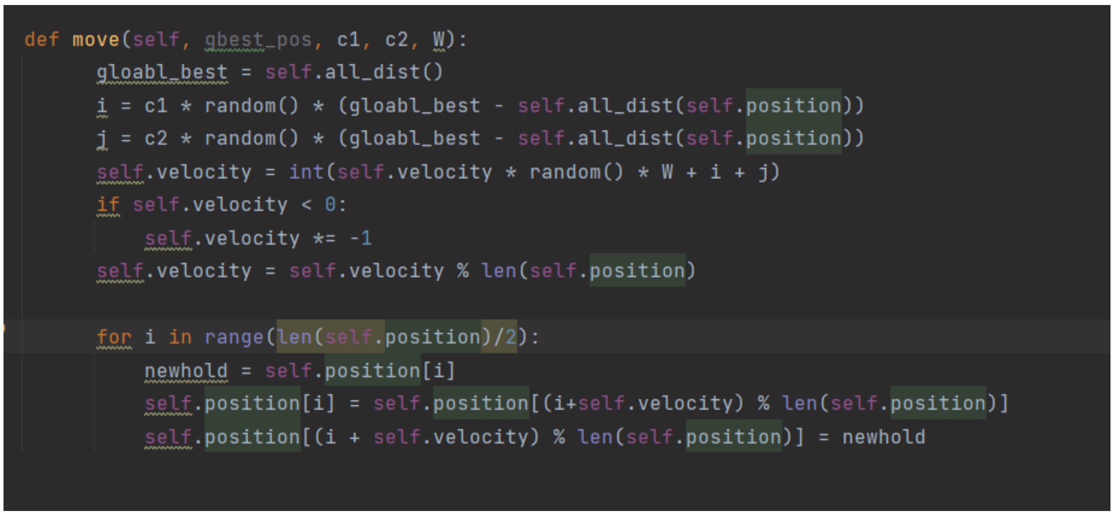
```
result:  [4, 3, 1, 2, 5, 7, 9, 12, 14, 21, 19, 16, 17, 20, 18, 15, 13, 11, 8, 6, 10]
total:  379.2567506704295
clock ticks: 0.017078876495361328
result:  [4, 3, 1, 2, 5, 7, 9, 12, 14, 21, 19, 16, 17, 20, 18, 15, 13, 11, 8, 6, 10]
total:  379.2567506704295
clock ticks: 0.015967845916748047
Time elapsed:  28.637974500656128
379.2567506704295
0 4 3 1 2 5 7 9 12 0
0 14 21 19 16 0
0 17 20 18 15 0
0 13 11 8 6 10 0

Process finished with exit code 0
```

Here is a graph that shows the relation between the fitness(cost) and the num of the iteration:



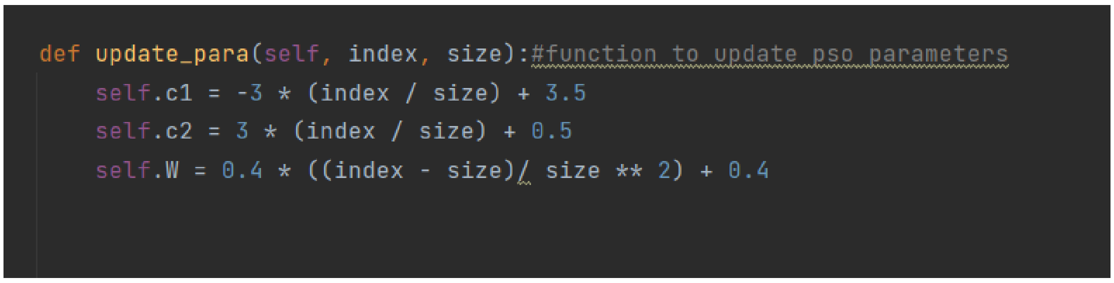
PSO:

In order to implement cooperative PSO algorithm that we learned in the lecture we updated the velocity of each particle and the position to be equal to:



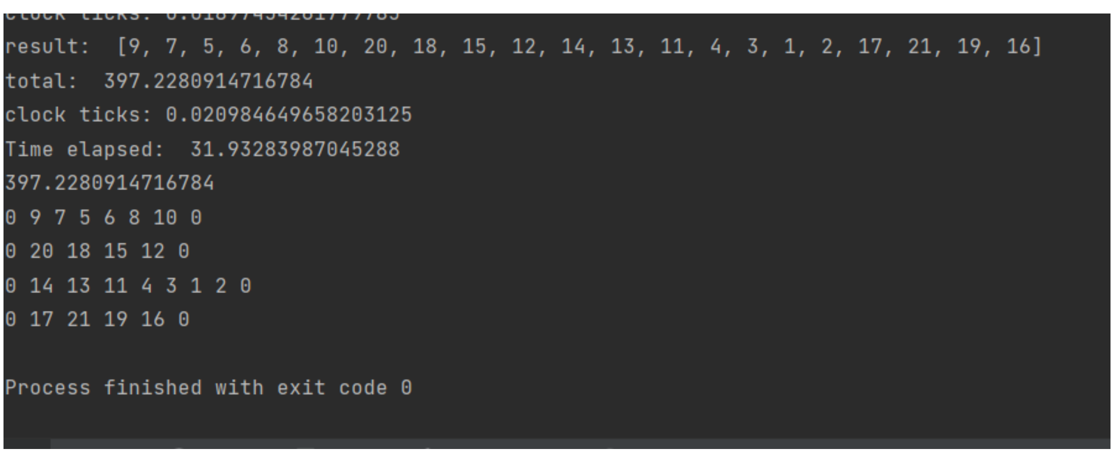
```python
def move(self, gbest_pos, c1, c2, W):
    gloabl_best = self.all_dist()
    i = c1 * random() * (gloabl_best - self.all_dist(self.position))
    j = c2 * random() * (gloabl_best - self.all_dist(self.position))
    self.velocity = int(self.velocity * random() * W + i + j)
    if self.velocity < 0:
        self.velocity *= -1
    self.velocity = self.velocity % len(self.position)

    for i in range(len(self.position)/2):
        newhold = self.position[i]
        self.position[i] = self.position[(i+self.velocity) % len(self.position)]
        self.position[(i + self.velocity) % len(self.position)] = newhold
```

And we updated the PSO parameters as the equation we saw in the lecture:



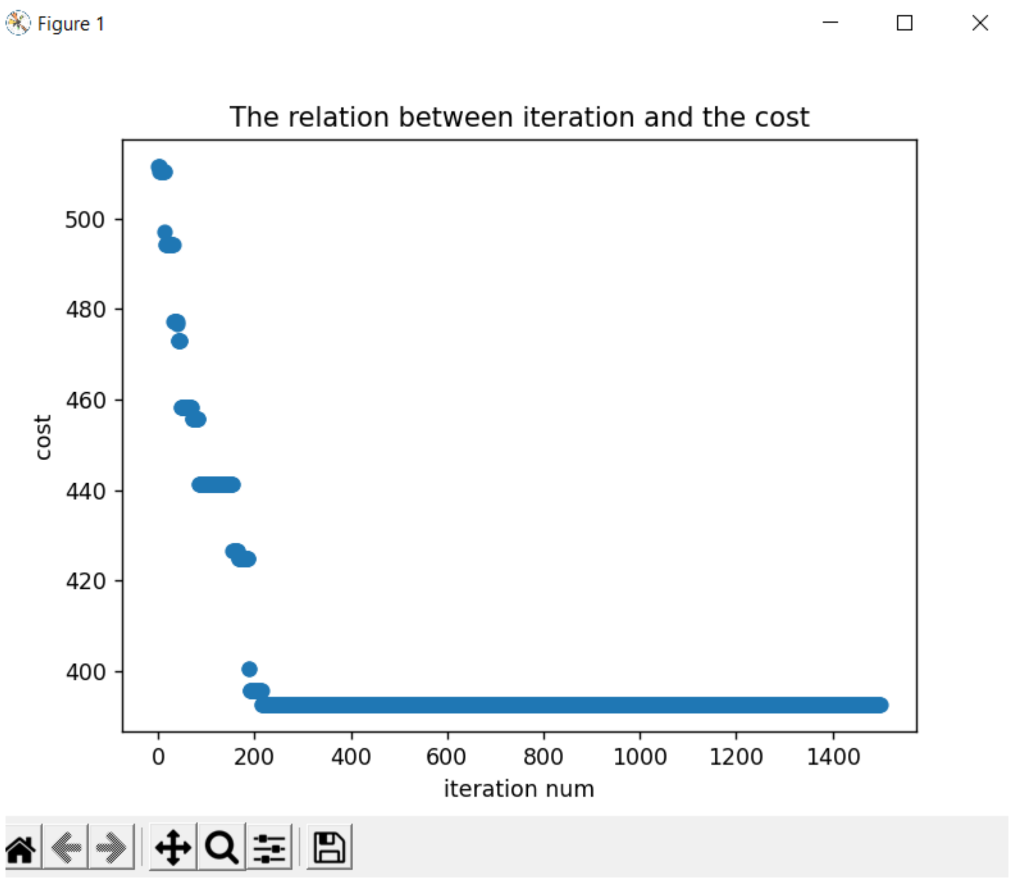
```python
def update_para(self, index, size):#function to update pso parameters
    self.c1 = -3 * (index / size) + 3.5
    self.c2 = 3 * (index / size) + 0.5
    self.W = 0.4 * ((index - size)/ size ** 2) + 0.4
```

Here is the output of PSO on file (E-n22-k4) :



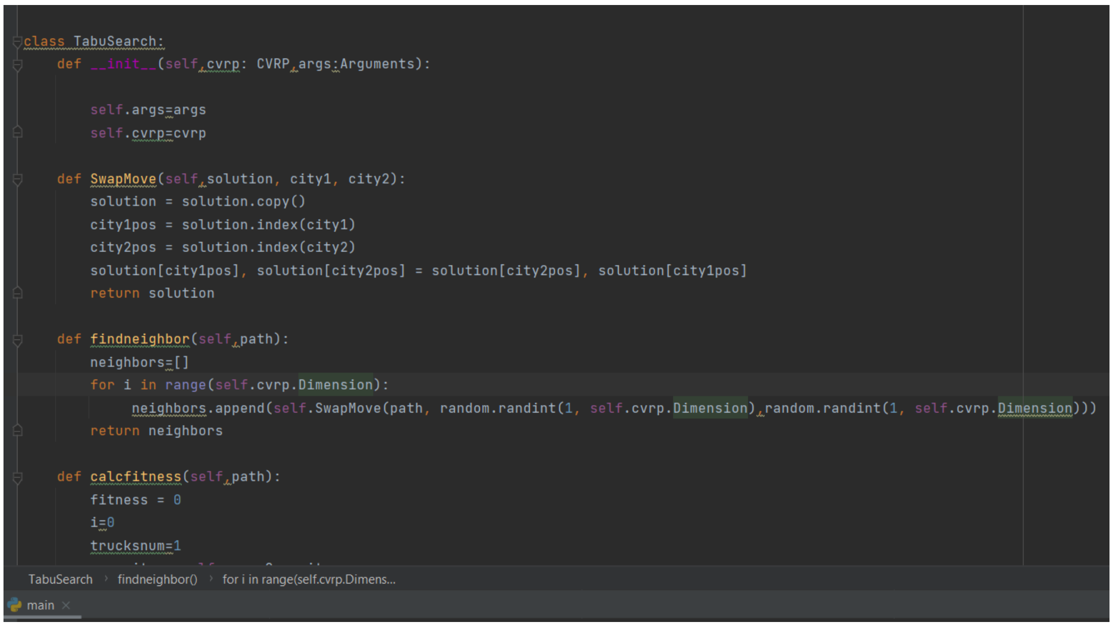
```
result:  [9, 7, 5, 6, 8, 10, 20, 18, 15, 12, 14, 13, 11, 4, 3, 1, 2, 17, 21, 19, 16]
total:  397.2280914716784
clock ticks: 0.020984649658203125
Time elapsed:  31.93283987045288
397.2280914716784
0 9 7 5 6 8 10 0
0 20 18 15 12 0
0 14 13 11 4 3 1 2 0
0 17 21 19 16 0

Process finished with exit code 0
```

Here is the graph:



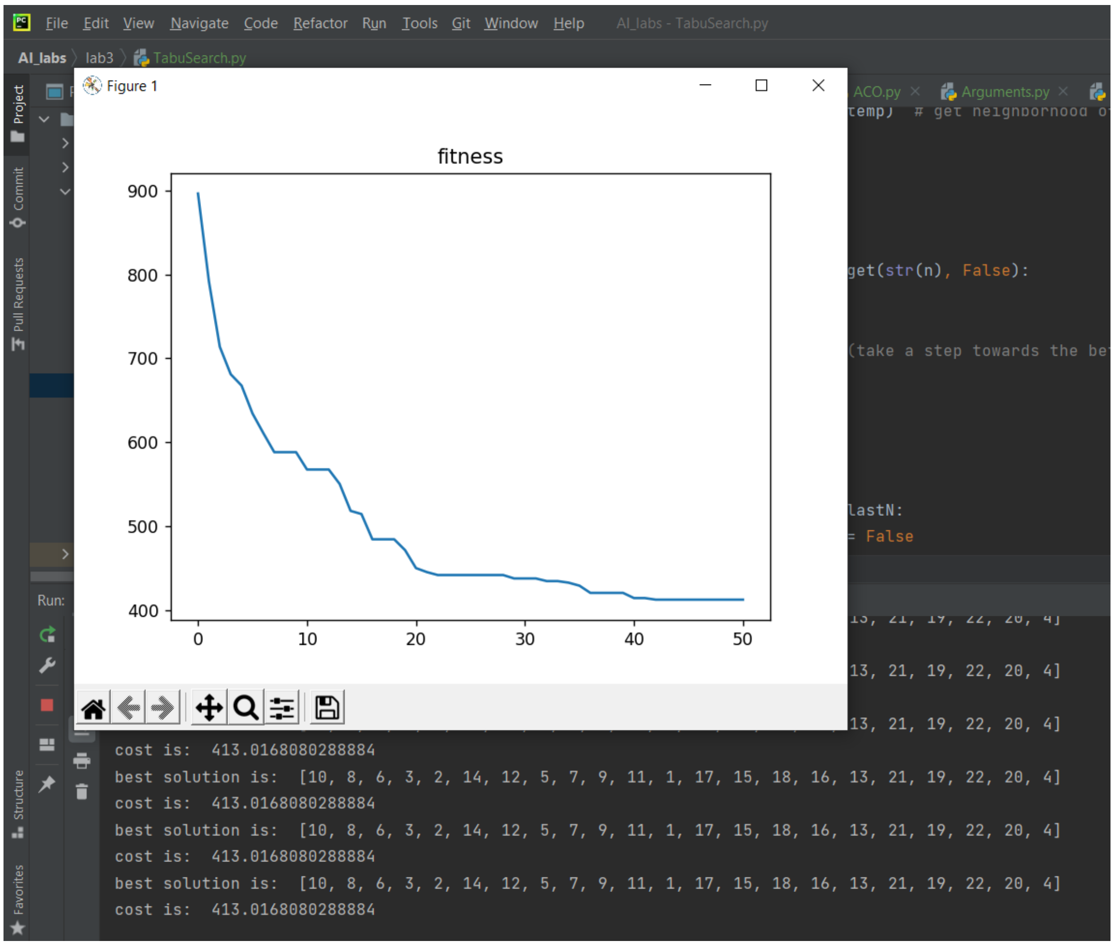
TabuSearch:

here is a little bit of our code implementing Tabu search algorithm:



```python
class TabuSearch:
    def __init__(self,cvrp: CVRP,args:Arguments):

        self.args=args
        self.cvrp=cvrp

    def SwapMove(self,solution, city1, city2):
        solution = solution.copy()
        city1pos = solution.index(city1)
        city2pos = solution.index(city2)
        solution[city1pos], solution[city2pos] = solution[city2pos], solution[city1pos]
        return solution

    def findneighbor(self,path):
        neighbors=[]
        for i in range(self.cvrp.Dimension):
            neighbors.append(self.SwapMove(path, random.randint(1, self.cvrp.Dimension),random.randint(1, self.cvrp.Dimension)))
        return neighbors

    def calcfitness(self,path):
        fitness = 0
        i=0
        trucksnum=1
```

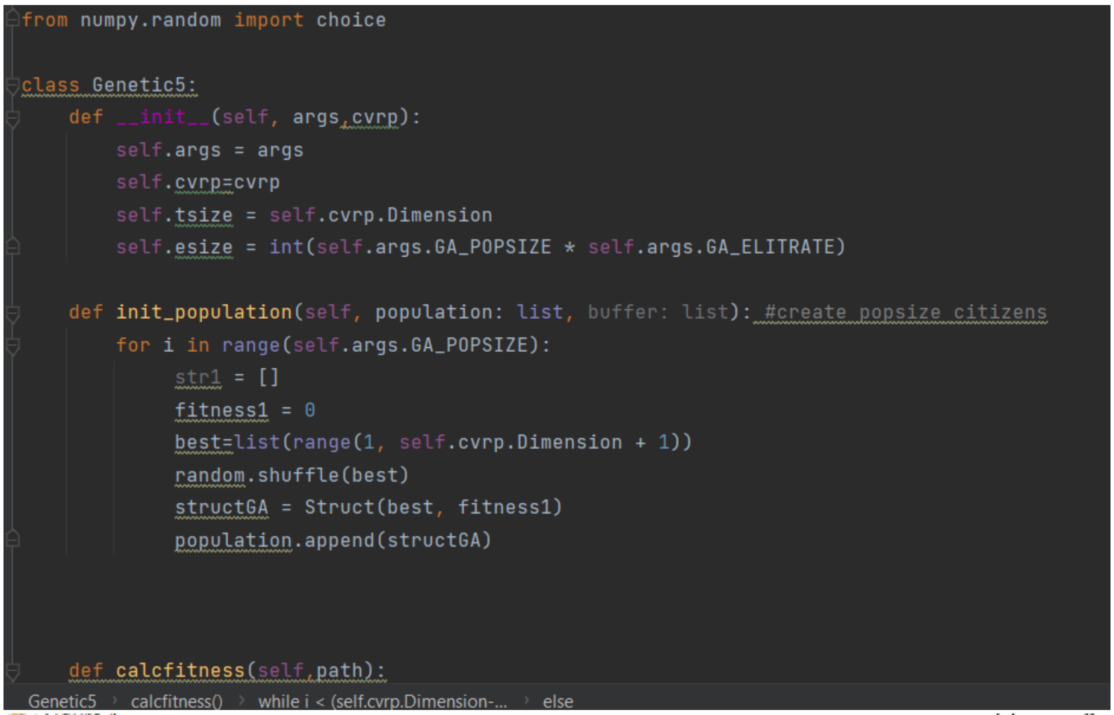
Here is the output:


File Edit View Navigate Code Refactor Run Tools Git Window Help AI_labs - TabuSearch.py
AI_labs lab3 TabuSearch.py
Figure 1
ACO.py Arguments.py
fitness
900
800
700
600
500
400
0
10
20
30
40
50
Run:
Project
Commit
Pull Requests
Structure
Favorites

```
temp)  # get neighborhood o
get(str(n), False):
(take a step towards the be
lastN:
= False
13, 21, 19, 22, 20, 4]
13, 21, 19, 22, 20, 4]
13, 21, 19, 22, 20, 4]
cost is:  413.0168080288884
best solution is:  [10, 8, 6, 3, 2, 14, 12, 5, 7, 9, 11, 1, 17, 15, 18, 16, 13, 21, 19, 22, 20, 4]
cost is:  413.0168080288884
best solution is:  [10, 8, 6, 3, 2, 14, 12, 5, 7, 9, 11, 1, 17, 15, 18, 16, 13, 21, 19, 22, 20, 4]
cost is:  413.0168080288884
best solution is:  [10, 8, 6, 3, 2, 14, 12, 5, 7, 9, 11, 1, 17, 15, 18, 16, 13, 21, 19, 22, 20, 4]
cost is:  413.0168080288884
```

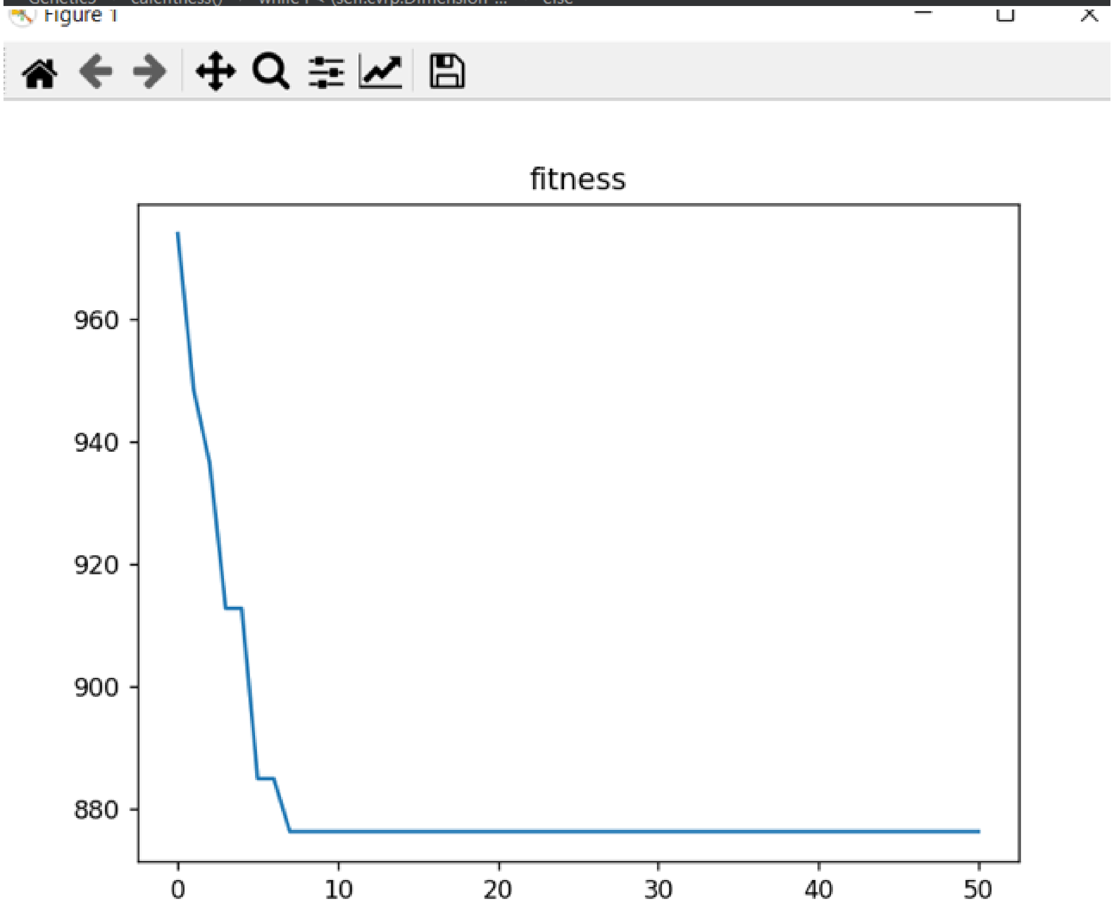
<u>GA- Island model:</u>

Here is a little bit of our code implementing GA in island model.



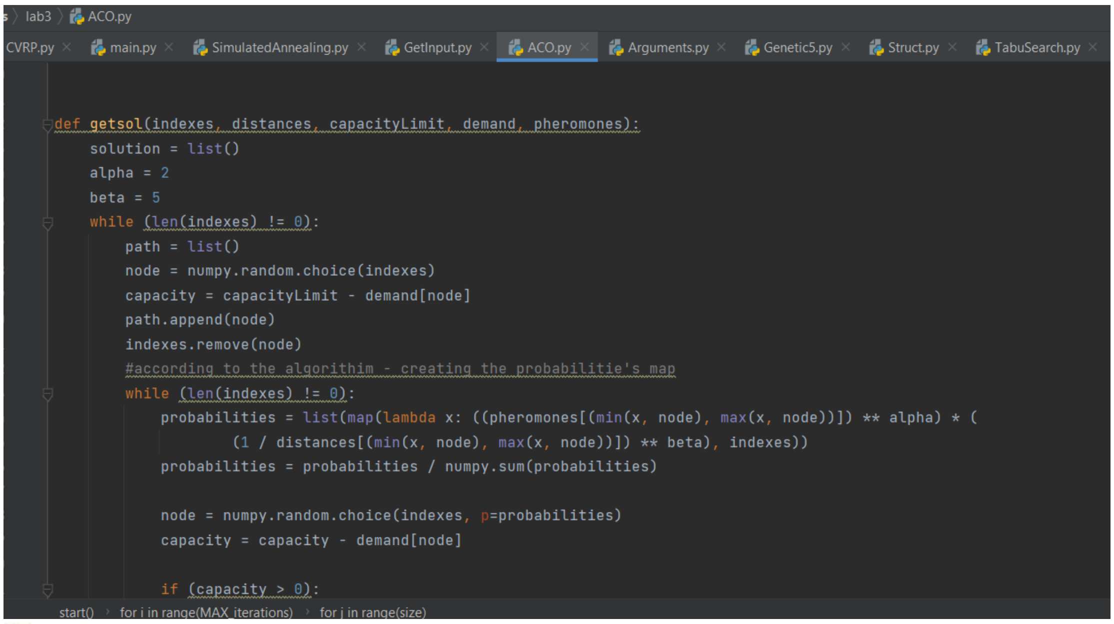
```
from numpy.random import choice

class Genetic5:
    def __init__(self, args,cvrp):
        self.args = args
        self.cvrp=cvrp
        self.tsize = self.cvrp.Dimension
        self.esize = int(self.args.GA_POPSIZE * self.args.GA_ELITRATE)

    def init_population(self, population: list, buffer: list): #create popsize citizens
        for i in range(self.args.GA_POPSIZE):
            str1 = []
            fitness1 = 0
            best=list(range(1, self.cvrp.Dimension + 1))
            random.shuffle(best)
            structGA = Struct(best, fitness1)
            population.append(structGA)


    def calcfitness(self,path):
```

Genetic5 › calcfitness() › while i < (self.cvrp.Dimension-... › else



ACO:

Here is a little bit of our code implementing ACO



```python
def getsol(indexes, distances, capacityLimit, demand, pheromones):
    solution = list()
    alpha = 2
    beta = 5
    while (len(indexes) != 0):
        path = list()
        node = numpy.random.choice(indexes)
        capacity = capacityLimit - demand[node]
        path.append(node)
        indexes.remove(node)
        #according to the algorithim - creating the probabilitie's map
        while (len(indexes) != 0):
            probabilities = list(map(lambda x: ((pheromones[(min(x, node), max(x, node))]) ** alpha) * (
                    (1 / distances[(min(x, node), max(x, node))]) ** beta), indexes))
            probabilities = probabilities / numpy.sum(probabilities)

            node = numpy.random.choice(indexes, p=probabilities)
            capacity = capacity - demand[node]

            if (capacity > 0):
```

start()
for i in range(MAX_iterations)
for j in range(size)

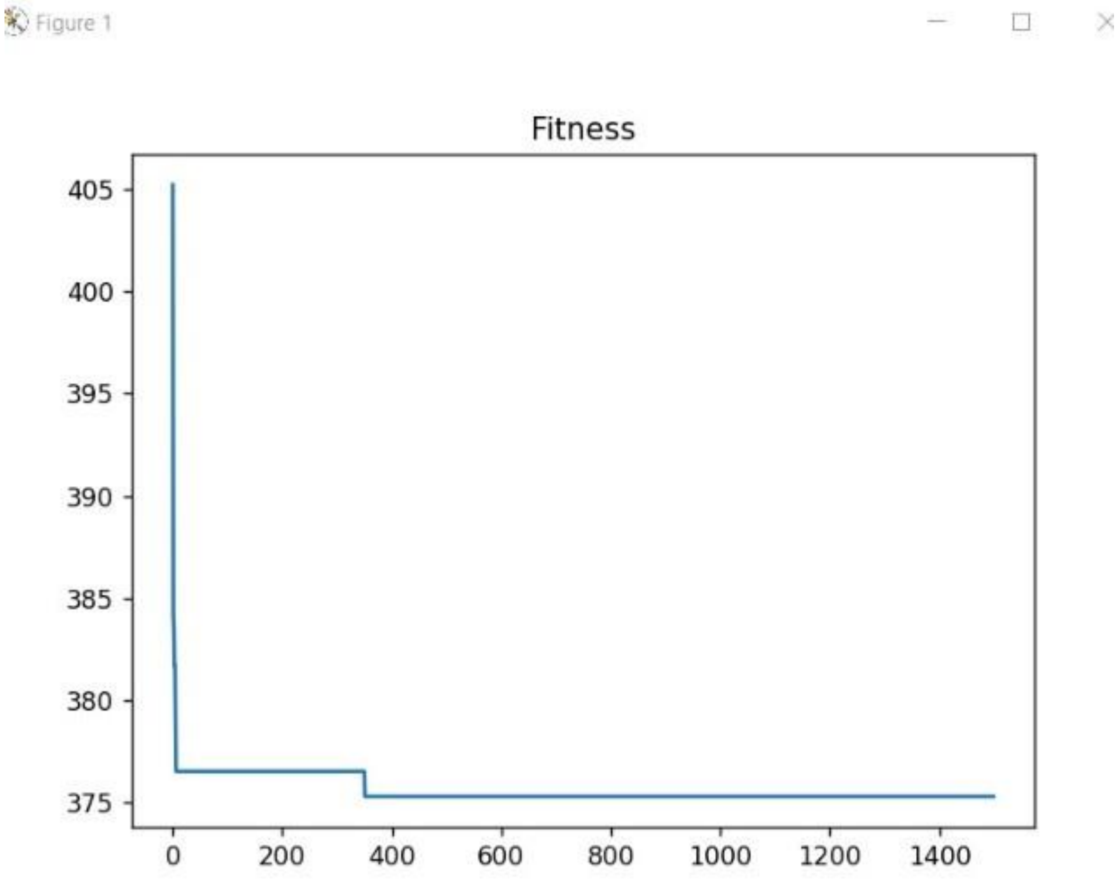


Figure 1
Fitness