PBA1 Q1

August 22, 2021

[1]:
```
import numpy as np
import itertools
import copy
```

[2]:
```
values = {'s1':8,'s2':11,'s3':9,'s4':12,'s5':14,'s6':10,'s7':6,'s8':7,'s9':13}
weights = {'s1':1,'s2':2,'s3':3,'s4':2,'s5':3,'s6':4,'s7':1,'s8':5,'s9':3}
limit = 16
```

[3]:
```
n = 9
combinations = [list(i) for i in itertools.product([0, 1], repeat=n)]
```

[4]:
```
initial1 = [0,1,1,1,0,0,1,1,1]
initial2 = [0,1,0,1,0,0,0,1,0]
```

[5]:
```
def calc_weight(solution): #calculates the weight of the proposed solution
    weight = 0
    for count,bit in enumerate(solution):
        i = count+1
        weight = weight + solution[count]*weights["s"+str(i)]
    return weight

def calc_value(solution): #calculates the value of the proposed solution
    value = 0
    for count,bit in enumerate(solution):
        i = count+1
        value = value + solution[count]*values["s"+str(i)]
    return value
```

[6]:
```
def Neighbourhood(current_solution): #generates the neighbourhood from the␣
 ↪current proposed solution
    possible_neighbours=[]
    for count,bit in enumerate(current_solution):
        new_neighbour = copy.copy(current_solution)
        if(bit==1):
            bit = 0
        else: bit = 1
        new_neighbour[count] = bit
```

```
        possible_neighbours.append(new_neighbour)
    return possible_neighbours
```

[7]:
```
def find_solution(initial):
    weight = calc_weight(initial)
    current_solution = initial
    max_value = calc_value(current_solution)
    print("current max",max_value)
    while(weight<16):
        sol_list = []
        for neighbour in Neighbourhood(current_solution): #creates a dictionary␣
↪with the neighbourhood and
                                                           #each of the␣
↪potential solutions attributes
            value = calc_value(neighbour)
            weight = calc_weight(neighbour)
            attributes = {"vector":neighbour,"value":value,"weight":weight}
            sol_list.append(attributes)
        temp_max = max_value
        temp_solution = current_solution
        print("\n")
        for entry in sol_list: #check the validty of each neighbourhood␣
↪solution and decide if it is a new best
            print("\n",entry['vector'])
            print("considering weight:",entry['weight'],"and value:
↪",entry['value'],end = '')
            if(entry['value']>temp_max and entry['weight']<=16):
                print(" NEW BEST",end = '')
                temp_solution = entry['vector']
                temp_max = entry['value']
        if(temp_max>max_value):
            current_solution = temp_solution
            max_value = temp_max


        elif(temp_max<=max_value):
            print("\n No new solution found")
            break
    return current_solution
```

[8]:
```
answer = find_solution(initial2) #The run for initial2
print("\n")
print("starting=",initial2)
print("value:",calc_value(initial2))
print("weight:",calc_weight(initial2))
```

```python
print("final=",answer)
print("value:",calc_value(answer))
print("weight:",calc_weight(answer))
```

```
current max 30



 [1, 1, 0, 1, 0, 0, 0, 1, 0]
considering weight: 10 and value: 38 NEW BEST
 [0, 0, 0, 1, 0, 0, 0, 1, 0]
considering weight: 7 and value: 19
 [0, 1, 1, 1, 0, 0, 0, 1, 0]
considering weight: 12 and value: 39 NEW BEST
 [0, 1, 0, 0, 0, 0, 0, 1, 0]
considering weight: 7 and value: 18
 [0, 1, 0, 1, 1, 0, 0, 1, 0]
considering weight: 12 and value: 44 NEW BEST
 [0, 1, 0, 1, 0, 1, 0, 1, 0]
considering weight: 13 and value: 40
 [0, 1, 0, 1, 0, 0, 1, 1, 0]
considering weight: 10 and value: 36
 [0, 1, 0, 1, 0, 0, 0, 0, 0]
considering weight: 4 and value: 23
 [0, 1, 0, 1, 0, 0, 0, 1, 1]
considering weight: 12 and value: 43


 [1, 1, 0, 1, 1, 0, 0, 1, 0]
considering weight: 13 and value: 52 NEW BEST
 [0, 0, 0, 1, 1, 0, 0, 1, 0]
considering weight: 10 and value: 33
 [0, 1, 1, 1, 1, 0, 0, 1, 0]
considering weight: 15 and value: 53 NEW BEST
 [0, 1, 0, 0, 1, 0, 0, 1, 0]
considering weight: 10 and value: 32
 [0, 1, 0, 1, 0, 0, 0, 1, 0]
considering weight: 9 and value: 30
 [0, 1, 0, 1, 1, 1, 0, 1, 0]
considering weight: 16 and value: 54 NEW BEST
 [0, 1, 0, 1, 1, 0, 1, 1, 0]
considering weight: 13 and value: 50
 [0, 1, 0, 1, 1, 0, 0, 0, 0]
considering weight: 7 and value: 37
 [0, 1, 0, 1, 1, 0, 0, 1, 1]
```

```
considering weight: 15 and value: 57 NEW BEST


 [1, 1, 0, 1, 1, 0, 0, 1, 1]
considering weight: 16 and value: 65 NEW BEST
 [0, 0, 0, 1, 1, 0, 0, 1, 1]
considering weight: 13 and value: 46
 [0, 1, 1, 1, 1, 0, 0, 1, 1]
considering weight: 18 and value: 66
 [0, 1, 0, 0, 1, 0, 0, 1, 1]
considering weight: 13 and value: 45
 [0, 1, 0, 1, 0, 0, 0, 1, 1]
considering weight: 12 and value: 43
 [0, 1, 0, 1, 1, 1, 0, 1, 1]
considering weight: 19 and value: 67
 [0, 1, 0, 1, 1, 0, 1, 1, 1]
considering weight: 16 and value: 63
 [0, 1, 0, 1, 1, 0, 0, 0, 1]
considering weight: 10 and value: 50
 [0, 1, 0, 1, 1, 0, 0, 1, 0]
considering weight: 12 and value: 44


 [0, 1, 0, 1, 1, 0, 0, 1, 1]
considering weight: 15 and value: 57
 [1, 0, 0, 1, 1, 0, 0, 1, 1]
considering weight: 14 and value: 54
 [1, 1, 1, 1, 1, 0, 0, 1, 1]
considering weight: 19 and value: 74
 [1, 1, 0, 0, 1, 0, 0, 1, 1]
considering weight: 14 and value: 53
 [1, 1, 0, 1, 0, 0, 0, 1, 1]
considering weight: 13 and value: 51
 [1, 1, 0, 1, 1, 1, 0, 1, 1]
considering weight: 20 and value: 75
 [1, 1, 0, 1, 1, 0, 1, 1, 1]
considering weight: 17 and value: 71
 [1, 1, 0, 1, 1, 0, 0, 0, 1]
considering weight: 11 and value: 58
 [1, 1, 0, 1, 1, 0, 0, 1, 0]
considering weight: 13 and value: 52
 No new solution found


starting= [0, 1, 0, 1, 0, 0, 0, 1, 0]
value: 30
weight: 9
final= [1, 1, 0, 1, 1, 0, 0, 1, 1]
```

```
value: 65
weight: 16
```

[9]:

```
answer = find_solution(initial1)
print("\n")
print("starting=",initial1)
print("value:",calc_value(initial1))
print("weight:",calc_weight(initial1))


print("final=",answer)
print("value:",calc_value(answer))
print("weight:",calc_weight(answer))
```

```
current max 58


starting= [0, 1, 1, 1, 0, 0, 1, 1, 1]
value: 58
weight: 16
final= [0, 1, 1, 1, 0, 0, 1, 1, 1]
value: 58
weight: 16
```

[ ]:

```

```

[ ]:

```

```

PBA1 Q2

August 22, 2021

[1]:
```
import numpy as np
import copy
import random
import operator
import pandas as pd
import matplotlib.pyplot as plt
```

[2]:
```
distances = [
    [0,32,39,42,29,35],
    [32,0,36,27,41,25],
    [39,36,0,28,33,40],
    [42,27,28,0,27,38],
    [29,41,33,27,0,26],
    [35,25,40,38,26,0]
]

cities = [1,2,3,4,5,6]

def distance_calc(me,neighbour): #returns distances from the distance matrix
    return distances[me-1][neighbour-1]

class city:

    def __init__(self,me):
        self.me = me-1

    def d(self,neighbour):
        return distances[self.me][neighbour-1]
    def num(self):
        return self.me

cityList = []
for entry in cities:
    cityList.append(city(entry))

class route:
```

```
    def __init__(self,cities,rand):
        self.route = cities
        if(rand == 1):
            self.route = random.sample(cities,len(cities))

    def getRoute(self):
        return self.route

    def getCities(self): #returns a list of ints for breeding and display␣
↪purposes
        lst = []
        for city in self.route:
            if(type(city)==int):
                lst.append(city)
                continue
            lst.append(city.num()+1)
        return lst


    def calcFitness(self): #calculates the fitness of the route as 1/distance
        routeDistance = 0
        dist = 0
        for i in range(len(self.route)):
            dist = 0
            if(type(self.route[0])==int):
                if(i==len(self.route)-1):
                    dist = distance_calc(self.route[i],self.route[0])
                    routeDistance += dist
                    self.fitness = float(1/float(routeDistance))
                    return self.fitness
                dist = distance_calc(self.route[i],self.route[i+1])
                routeDistance += dist

            else:
                if(i==len(self.route)-1):
                    dist = distance_calc(self.route[i].num()+1,self.route[0].
↪num()+1)
                    routeDistance += dist
                    self.fitness = float(1/float(routeDistance))
                    return self.fitness
                dist = distance_calc(self.route[i].num()+1,self.route[i+1].
↪num()+1)
                routeDistance += dist
        return self.fitness

def selectRoutes(routeList): #sorts different routes into order with highest␣
↪fitness first
```

```
    sortedFitness = {}
    for count,entry in enumerate(routeList):
        sortedFitness[count] = entry.calcFitness()
    temp = copy.copy(sortedFitness)
    sortedFitness = sorted(temp.items(),key = operator.itemgetter(1), reverse =␣
↪True)
    sortedFitness = list(sortedFitness)
    sortedFitness = [list(element) for element in sortedFitness]
    return sortedFitness

def tournament_select(sortedFitness,final_list): #tournament with 3 individuals␣
↪randomly selected with proportion to their fitness
    #returns all 6 parents
    parent_list = []
    for i in range(6):
        tournament_list = []
        for k in range(3):
            selection_number = random.random()
            if(selection_number<sortedFitness[0][1]):
                selection_number = random.random()
            if(selection_number==1.0):
                chosen_route = sortedFitness[-1][0]
            for element in sortedFitness:
                if(element[1]<=selection_number):
                    chosen_route = element[0]

            for trip in sortedFitness:
                if(chosen_route==trip[0]):
                    competitor = final_list[chosen_route]
                    tournament_list.append(competitor)

        tournament_list = sorted(tournament_list,key=lambda entry: entry.
↪fitness,reverse=True)
        winner = tournament_list[0] #sort tournament list and select highest␣
↪fitness
        parent_list.append(winner)
    print("size of parents = ",len(parent_list))
    return parent_list

def crossover(parent_list): #handles the 2 point crossover process
    #returns all offspring from all parents
    offspring = []
    for i in range(int(len(parent_list)/2)):
        parent1 = parent_list[2*i].getCities()
        parent2 = parent_list[2*i+1].getCities()
        for j in range(2):
            if(parent1 == parent2):
```

```
                print("Parent 1:",parent1)
                print("          --------X--------")
                print("Parent 2:",parent2)
                print("Identical parents \n")
                offspring.append(route(parent1,rand=0))
                break
            child = ['x','x','x','x','x','x']
            crossover1 = random.randrange(len(parent1)/2)
            crossover2 = random.randrange(len(parent2))
            while(crossover2<=crossover1): #need different crossover points
                crossover2 = random.randrange(len(parent2))
            print("Parent 1:",parent1)
            print("          --------X--------")
            print("Parent 2:",parent2)
            print("     Crossovers at ",crossover1,"and ",crossover2)
            child[:crossover1] = parent1[:crossover1]
            child[crossover2:] = parent1[crossover2:]
            print("Showing cut points for child:")
            print(child)
            for count,k in enumerate(child):
                if(k=='x'):
                    for c in parent2:
                        if(c not in child):
                            child[count] = c
                            break
            print(child)
            offspring.append(route(child,rand=0))
            print()
    return offspring
```

[3]:

```
route_list = []
population_size = 8
for i in range(population_size):
    route_list.append(route(cityList,rand=1))
print("Initial population (8 random):")
for trip in route_list:
    print(trip.getCities())
```

```
Initial population (8 random):
[2, 4, 5, 3, 1, 6]
[2, 4, 1, 6, 5, 3]
[3, 2, 5, 6, 1, 4]
[3, 4, 5, 1, 2, 6]
[6, 4, 1, 3, 2, 5]
[3, 4, 1, 5, 6, 2]
[5, 4, 3, 1, 2, 6]
```

```
[6, 4, 1, 2, 3, 5]
```

[4]:
```
incumbent_counter = 0
final_list = route_list
bestFitness = 0
iteration = 0
sortedFitness = selectRoutes(final_list)
bestFitness  = sortedFitness[0][1]
parent_list=[]

while(incumbent_counter<9):
    iteration +=1
    print("________________________________")
    print("Iteration Number:",iteration)
    sortedFitness = selectRoutes(final_list)
    newBest = sortedFitness[0][1]
    if(bestFitness>=newBest):
        incumbent_counter+=1
    else:
        bestFitness=newBest
        incumbent_counter=0
    print("Current Population:")
    mean_fitness = 0
    for thing in final_list:
        mean_fitness+=thing.calcFitness()
        print(thing.getCities(),"fitness = ", thing.calcFitness())
    mean_fitness = mean_fitness/len(final_list)
    print()
    print("best fitness:",bestFitness)
    print()
    print("mean_fitness = ",mean_fitness )

    totalFitness = 0
    for element in sortedFitness:
        totalFitness += element[1]

    for element in sortedFitness:
        element[1] = element[1]/totalFitness

    for count,element in enumerate(sortedFitness):

        if(count>=1):
            element[1] += sortedFitness[count-1][1]

    #Result of the above is to have a cumulative total of fitness
    #which allows easy random selection proportional to fitness values
```

```
    #Generating parent list through tournament
    print(sortedFitness)
    parent_list = tournament_select(sortedFitness,final_list)
    for parent in parent_list:
        print(parent.getCities())
    print()
    #Starting crossover
    offspring = crossover(parent_list)

    #The below mutation section was not made modular since it changes just one element of the children.
    #If the mutation occured more often it would make sense to make a separate function

    mutation_selector = random.randrange(len(offspring))
    mutated = offspring[mutation_selector].getCities() #select the array of ints to be mutated

    print("Child from all offspring selected for mutation --->",mutated)
    print()
    swapped1 = random.randrange(len(mutated))
    swapped2 = random.randrange(len(mutated))
    while(swapped2==swapped1):
        swapped2 = random.randrange(len(mutated))
    temp = mutated[swapped1]
    mutated[swapped1] = mutated[swapped2]
    mutated[swapped2] = temp
    print("Swapping locations:", swapped1, "and", swapped2)
    print("Result of mutation process of child            --->",mutated)
    print()
    offspring[mutation_selector] = route(mutated,rand=0)

    for parent in parent_list:
        offspring.append(parent)

    sortedOffspring = selectRoutes(offspring) #Order all offspring and parents by fitness

    final_list = []
    for count,element in enumerate(sortedOffspring): #select 8 most fit to go into final_list and continue to next population
        if(count>=population_size):
            break
        final_list.append(offspring[element[0]])
    print("Best Distance so far:",1/bestFitness)
    print("Best Distance this generation:",1/newBest)
    print("________________________________")
```

```
--------------------------------
Iteration Number: 1
Current Population:
[2, 4, 5, 3, 1, 6] fitness =  0.005376344086021506
[2, 4, 1, 6, 5, 3] fitness =  0.005025125628140704
[3, 2, 5, 6, 1, 4] fitness =  0.004807692307692308
[3, 4, 5, 1, 2, 6] fitness =  0.0055248618784530384
[6, 4, 1, 3, 2, 5] fitness =  0.0045045045045045045
[3, 4, 1, 5, 6, 2] fitness =  0.005376344086021506
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[6, 4, 1, 2, 3, 5] fitness =  0.004830917874396135

best fitness: 0.005649717514124294

mean_fitness =  0.005136938484919249
[[6, 0.13747773919014294], [3, 0.27191729629873573], [0, 0.4027428868183879],
[5, 0.53356847733804], [1, 0.6558476724971118], [7, 0.7734011016596978], [2,
0.890389370105156], [4, 0.9999999999999997]]
size of parents =  6
[3, 4, 1, 5, 6, 2]
[3, 4, 1, 5, 6, 2]
[3, 4, 5, 1, 2, 6]
[5, 4, 3, 1, 2, 6]
[2, 4, 5, 3, 1, 6]
[3, 4, 5, 1, 2, 6]

Parent 1: [3, 4, 1, 5, 6, 2]
          --------X--------
Parent 2: [3, 4, 1, 5, 6, 2]
Identical parents

Parent 1: [3, 4, 5, 1, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
     Crossovers at  2 and  3
Showing cut points for child:
[3, 4, 'x', 1, 2, 6]
[3, 4, 5, 1, 2, 6]

Parent 1: [3, 4, 5, 1, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
     Crossovers at  2 and  5
Showing cut points for child:
```

```
[3, 4, 'x', 'x', 'x', 6]
[3, 4, 5, 1, 2, 6]

Parent 1: [2, 4, 5, 3, 1, 6]
          --------X--------
Parent 2: [3, 4, 5, 1, 2, 6]
     Crossovers at  0 and  3
Showing cut points for child:
['x', 'x', 'x', 3, 1, 6]
[4, 5, 2, 3, 1, 6]

Parent 1: [2, 4, 5, 3, 1, 6]
          --------X--------
Parent 2: [3, 4, 5, 1, 2, 6]
     Crossovers at  1 and  3
Showing cut points for child:
[2, 'x', 'x', 3, 1, 6]
[2, 4, 5, 3, 1, 6]

Child from all offspring selected for mutation ---> [3, 4, 5, 1, 2, 6]

Swapping locations: 5 and 3
Result of mutation process of child            ---> [3, 4, 5, 6, 2, 1]

Best Distance so far: 177.0
Best Distance this generation: 177.0
--------------------------------
--------------------------------
Iteration Number: 2
Current Population:
[3, 4, 5, 6, 2, 1] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[3, 4, 5, 1, 2, 6] fitness =  0.0055248618784530384
[3, 4, 5, 1, 2, 6] fitness =  0.0055248618784530384
[3, 4, 5, 1, 2, 6] fitness =  0.0055248618784530384
[3, 4, 1, 5, 6, 2] fitness =  0.005376344086021506
[2, 4, 5, 3, 1, 6] fitness =  0.005376344086021506
[3, 4, 1, 5, 6, 2] fitness =  0.005376344086021506

best fitness: 0.005649717514124294

mean_fitness =  0.005500381615209028
[[0, 0.12839376222784113], [1, 0.25678752445568226], [2, 0.38234385547406835],
[3, 0.5079001864924544], [4, 0.6334565175108404], [5, 0.7556376783405603], [6,
0.8778188391702801], [7, 1.0]]
size of parents =  6
[5, 4, 3, 1, 2, 6]
[3, 4, 5, 1, 2, 6]
```

```
[5, 4, 3, 1, 2, 6]
[3, 4, 5, 6, 2, 1]
[5, 4, 3, 1, 2, 6]
[5, 4, 3, 1, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [3, 4, 5, 1, 2, 6]
     Crossovers at  0 and  1
Showing cut points for child:
['x', 4, 3, 1, 2, 6]
[5, 4, 3, 1, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [3, 4, 5, 1, 2, 6]
     Crossovers at  2 and  4
Showing cut points for child:
[5, 4, 'x', 'x', 2, 6]
[5, 4, 3, 1, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [3, 4, 5, 6, 2, 1]
     Crossovers at  2 and  4
Showing cut points for child:
[5, 4, 'x', 'x', 2, 6]
[5, 4, 3, 1, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [3, 4, 5, 6, 2, 1]
     Crossovers at  2 and  4
Showing cut points for child:
[5, 4, 'x', 'x', 2, 6]
[5, 4, 3, 1, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
Identical parents

Child from all offspring selected for mutation ---> [5, 4, 3, 1, 2, 6]

Swapping locations: 3 and 0
Result of mutation process of child             ---> [1, 4, 3, 5, 2, 6]

Best Distance so far: 177.0
```

```
Best Distance this generation: 177.0
--------------------------------
--------------------------------
Iteration Number: 3
Current Population:
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[3, 4, 5, 6, 2, 1] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294

best fitness: 0.005649717514124294

mean_fitness =  0.005649717514124294
[[0, 0.125], [1, 0.25], [2, 0.375], [3, 0.5], [4, 0.625], [5, 0.75], [6, 0.875],
[7, 1.0]]
size of parents =  6
[3, 4, 5, 6, 2, 1]
[3, 4, 5, 6, 2, 1]
[5, 4, 3, 1, 2, 6]
[3, 4, 5, 6, 2, 1]
[5, 4, 3, 1, 2, 6]
[5, 4, 3, 1, 2, 6]

Parent 1: [3, 4, 5, 6, 2, 1]
          --------X--------
Parent 2: [3, 4, 5, 6, 2, 1]
Identical parents

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [3, 4, 5, 6, 2, 1]
     Crossovers at  0 and  1
Showing cut points for child:
['x', 4, 3, 1, 2, 6]
[5, 4, 3, 1, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [3, 4, 5, 6, 2, 1]
     Crossovers at  2 and  4
Showing cut points for child:
[5, 4, 'x', 'x', 2, 6]
[5, 4, 3, 1, 2, 6]
```

```
Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
Identical parents

Child from all offspring selected for mutation ---> [3, 4, 5, 6, 2, 1]

Swapping locations: 0 and 3
Result of mutation process of child            ---> [6, 4, 5, 3, 2, 1]

Best Distance so far: 177.0
Best Distance this generation: 177.0
--------------------------------
--------------------------------
Iteration Number: 4
Current Population:
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[3, 4, 5, 6, 2, 1] fitness =  0.005649717514124294
[3, 4, 5, 6, 2, 1] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[3, 4, 5, 6, 2, 1] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294

best fitness: 0.005649717514124294

mean_fitness =  0.005649717514124294
[[0, 0.125], [1, 0.25], [2, 0.375], [3, 0.5], [4, 0.625], [5, 0.75], [6, 0.875],
[7, 1.0]]
size of parents =  6
[5, 4, 3, 1, 2, 6]
[3, 4, 5, 6, 2, 1]
[5, 4, 3, 1, 2, 6]
[3, 4, 5, 6, 2, 1]
[5, 4, 3, 1, 2, 6]
[5, 4, 3, 1, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [3, 4, 5, 6, 2, 1]
     Crossovers at  2 and  3
Showing cut points for child:
[5, 4, 'x', 1, 2, 6]
[5, 4, 3, 1, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
```

```
Parent 2: [3, 4, 5, 6, 2, 1]
     Crossovers at  2 and  3
Showing cut points for child:
[5, 4, 'x', 1, 2, 6]
[5, 4, 3, 1, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [3, 4, 5, 6, 2, 1]
     Crossovers at  2 and  4
Showing cut points for child:
[5, 4, 'x', 'x', 2, 6]
[5, 4, 3, 1, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [3, 4, 5, 6, 2, 1]
     Crossovers at  2 and  5
Showing cut points for child:
[5, 4, 'x', 'x', 'x', 6]
[5, 4, 3, 2, 1, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
Identical parents

Child from all offspring selected for mutation ---> [5, 4, 3, 1, 2, 6]

Swapping locations: 0 and 3
Result of mutation process of child            ---> [1, 4, 3, 5, 2, 6]

Best Distance so far: 177.0
Best Distance this generation: 177.0
--------------------------------
--------------------------------
Iteration Number: 5
Current Population:
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[3, 4, 5, 6, 2, 1] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[3, 4, 5, 6, 2, 1] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294

best fitness: 0.005649717514124294
```

```
mean_fitness =  0.005649717514124294
[[0, 0.125], [1, 0.25], [2, 0.375], [3, 0.5], [4, 0.625], [5, 0.75], [6, 0.875],
[7, 1.0]]
size of parents =  6
[5, 4, 3, 1, 2, 6]
[5, 4, 3, 1, 2, 6]
[5, 4, 3, 1, 2, 6]
[5, 4, 3, 1, 2, 6]
[5, 4, 3, 1, 2, 6]
[3, 4, 5, 6, 2, 1]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
Identical parents

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
Identical parents

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [3, 4, 5, 6, 2, 1]
     Crossovers at  1 and  4
Showing cut points for child:
[5, 'x', 'x', 'x', 2, 6]
[5, 3, 4, 1, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [3, 4, 5, 6, 2, 1]
     Crossovers at  2 and  3
Showing cut points for child:
[5, 4, 'x', 1, 2, 6]
[5, 4, 3, 1, 2, 6]

Child from all offspring selected for mutation ---> [5, 4, 3, 1, 2, 6]

Swapping locations: 1 and 3
Result of mutation process of child           ---> [5, 1, 3, 4, 2, 6]

Best Distance so far: 177.0
Best Distance this generation: 177.0
--------------------------------
--------------------------------
Iteration Number: 6
```

```
Current Population:
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294

best fitness: 0.005747126436781609

mean_fitness =  0.005661893629456459
[[0, 0.1268817204301075], [1, 0.25161290322580643], [2, 0.3763440860215054], [3,
0.5010752688172043], [4, 0.6258064516129033], [5, 0.7505376344086022], [6,
0.8752688172043012], [7, 1.0]]
size of parents =  6
[5, 4, 3, 1, 2, 6]
[5, 4, 3, 1, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 4, 3, 1, 2, 6]
[5, 4, 3, 1, 2, 6]
[5, 4, 3, 1, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
     Crossovers at  0 and  2
Showing cut points for child:
['x', 'x', 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
     Crossovers at  0 and  2
Showing cut points for child:
['x', 'x', 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]

Parent 1: [5, 4, 3, 1, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
```

```
Identical parents

Child from all offspring selected for mutation ---> [5, 4, 3, 1, 2, 6]

Swapping locations: 4 and 3
Result of mutation process of child           ---> [5, 4, 3, 2, 1, 6]

Best Distance so far: 174.0
Best Distance this generation: 174.0
--------------------------------
--------------------------------
Iteration Number: 7
Current Population:
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294
[5, 4, 3, 1, 2, 6] fitness =  0.005649717514124294

best fitness: 0.005747126436781609

mean_fitness =  0.005686245860120788
[[0, 0.12633832976445394], [1, 0.2526766595289079], [2, 0.37901498929336186],
[3, 0.5032119914346894], [4, 0.627408993576017], [5, 0.7516059957173445], [6,
0.8758029978586721], [7, 0.9999999999999997]]
size of parents =  6
[5, 1, 3, 4, 2, 6]
[5, 4, 3, 1, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
     Crossovers at  1 and  2
Showing cut points for child:
[5, 'x', 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 4, 3, 1, 2, 6]
     Crossovers at  2 and  5
```

```
Showing cut points for child:
[5, 1, 'x', 'x', 'x', 6]
[5, 1, 4, 3, 2, 6]

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Child from all offspring selected for mutation ---> [5, 1, 4, 3, 2, 6]

Swapping locations: 1 and 2
Result of mutation process of child            ---> [5, 4, 1, 3, 2, 6]

Best Distance so far: 174.0
Best Distance this generation: 174.0
--------------------------------
--------------------------------
Iteration Number: 8
Current Population:
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609

best fitness: 0.005747126436781609

mean_fitness =  0.005747126436781609
[[0, 0.125], [1, 0.25], [2, 0.375], [3, 0.5], [4, 0.625], [5, 0.75], [6, 0.875],
[7, 1.0]]
size of parents =  6
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]

Parent 1: [5, 1, 3, 4, 2, 6]
```

```
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Child from all offspring selected for mutation ---> [5, 1, 3, 4, 2, 6]

Swapping locations: 3 and 5
Result of mutation process of child            ---> [5, 1, 3, 6, 2, 4]

Best Distance so far: 174.0
Best Distance this generation: 174.0
--------------------------------
--------------------------------
Iteration Number: 9
Current Population:
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609

best fitness: 0.005747126436781609

mean_fitness =  0.005747126436781609
[[0, 0.125], [1, 0.25], [2, 0.375], [3, 0.5], [4, 0.625], [5, 0.75], [6, 0.875],
[7, 1.0]]
size of parents =  6
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]

Parent 1: [5, 1, 3, 4, 2, 6]
```

```
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Child from all offspring selected for mutation ---> [5, 1, 3, 4, 2, 6]

Swapping locations: 1 and 3
Result of mutation process of child            ---> [5, 4, 3, 1, 2, 6]

Best Distance so far: 174.0
Best Distance this generation: 174.0
--------------------------------
--------------------------------
Iteration Number: 10
Current Population:
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609

best fitness: 0.005747126436781609

mean_fitness =  0.005747126436781609
[[0, 0.125], [1, 0.25], [2, 0.375], [3, 0.5], [4, 0.625], [5, 0.75], [6, 0.875],
[7, 1.0]]
size of parents =  6
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]

Parent 1: [5, 1, 3, 4, 2, 6]
```

```
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Child from all offspring selected for mutation ---> [5, 1, 3, 4, 2, 6]

Swapping locations: 0 and 3
Result of mutation process of child            ---> [4, 1, 3, 5, 2, 6]

Best Distance so far: 174.0
Best Distance this generation: 174.0
--------------------------------
--------------------------------
Iteration Number: 11
Current Population:
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609

best fitness: 0.005747126436781609

mean_fitness =  0.005747126436781609
[[0, 0.125], [1, 0.25], [2, 0.375], [3, 0.5], [4, 0.625], [5, 0.75], [6, 0.875],
[7, 1.0]]
size of parents =  6
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]

Parent 1: [5, 1, 3, 4, 2, 6]
```

```
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Child from all offspring selected for mutation ---> [5, 1, 3, 4, 2, 6]

Swapping locations: 3 and 0
Result of mutation process of child            ---> [4, 1, 3, 5, 2, 6]

Best Distance so far: 174.0
Best Distance this generation: 174.0
--------------------------------
--------------------------------
Iteration Number: 12
Current Population:
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609

best fitness: 0.005747126436781609

mean_fitness =  0.005747126436781609
[[0, 0.125], [1, 0.25], [2, 0.375], [3, 0.5], [4, 0.625], [5, 0.75], [6, 0.875],
[7, 1.0]]
size of parents =  6
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]

Parent 1: [5, 1, 3, 4, 2, 6]
```

```
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Child from all offspring selected for mutation ---> [5, 1, 3, 4, 2, 6]

Swapping locations: 0 and 4
Result of mutation process of child            ---> [2, 1, 3, 4, 5, 6]

Best Distance so far: 174.0
Best Distance this generation: 174.0
--------------------------------
--------------------------------
Iteration Number: 13
Current Population:
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609

best fitness: 0.005747126436781609

mean_fitness =  0.005747126436781609
[[0, 0.125], [1, 0.25], [2, 0.375], [3, 0.5], [4, 0.625], [5, 0.75], [6, 0.875],
[7, 1.0]]
size of parents =  6
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]

Parent 1: [5, 1, 3, 4, 2, 6]
```

```
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Child from all offspring selected for mutation ---> [5, 1, 3, 4, 2, 6]

Swapping locations: 0 and 2
Result of mutation process of child            ---> [3, 1, 5, 4, 2, 6]

Best Distance so far: 174.0
Best Distance this generation: 174.0
--------------------------------
--------------------------------
Iteration Number: 14
Current Population:
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609

best fitness: 0.005747126436781609

mean_fitness =  0.005747126436781609
[[0, 0.125], [1, 0.25], [2, 0.375], [3, 0.5], [4, 0.625], [5, 0.75], [6, 0.875],
[7, 1.0]]
size of parents =  6
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]

Parent 1: [5, 1, 3, 4, 2, 6]
```

```
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Child from all offspring selected for mutation ---> [5, 1, 3, 4, 2, 6]

Swapping locations: 5 and 1
Result of mutation process of child            ---> [5, 6, 3, 4, 2, 1]

Best Distance so far: 174.0
Best Distance this generation: 174.0
--------------------------------
--------------------------------
Iteration Number: 15
Current Population:
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609
[5, 1, 3, 4, 2, 6] fitness =  0.005747126436781609

best fitness: 0.005747126436781609

mean_fitness =  0.005747126436781609
[[0, 0.125], [1, 0.25], [2, 0.375], [3, 0.5], [4, 0.625], [5, 0.75], [6, 0.875],
[7, 1.0]]
size of parents =  6
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]
[5, 1, 3, 4, 2, 6]

Parent 1: [5, 1, 3, 4, 2, 6]
```

```
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Parent 1: [5, 1, 3, 4, 2, 6]
          --------X--------
Parent 2: [5, 1, 3, 4, 2, 6]
Identical parents

Child from all offspring selected for mutation ---> [5, 1, 3, 4, 2, 6]

Swapping locations: 3 and 1
Result of mutation process of child            ---> [5, 4, 3, 1, 2, 6]

Best Distance so far: 174.0
Best Distance this generation: 174.0
--------------------------------
```

PBA1 Q3

August 22, 2021

[1]:
```
#Question 3a
```

[2]:
```
import numpy as np
import itertools
import copy
import random
```

[3]:
```
values = {'s1':8,'s2':11,'s3':9,'s4':12,'s5':14,'s6':10,'s7':6,'s8':7,'s9':13}
weights = {'s1':1,'s2':2,'s3':3,'s4':2,'s5':3,'s6':4,'s7':1,'s8':5,'s9':3}
limit = 16
```

[4]:
```
n = 9
combinations = [list(i) for i in itertools.product([0, 1], repeat=n)]#all␣
↪possible combinations
```

[5]:
```
initial3 = [0,0,0,0,0,0,0,0,0]
```

[6]:
```
def calc_weight(solution):
    weight = 0
    for count,bit in enumerate(solution):
        i = count+1
        weight = weight + solution[count]*weights["s"+str(i)]
    return weight

def calc_value(solution):
    value = 0
    for count,bit in enumerate(solution):
        i = count+1
        value = value + solution[count]*values["s"+str(i)]
    return value
```

[7]:
```
def Neighbourhood(current_solution):
    possible_neighbours=[]
    for count,bit in enumerate(current_solution):
        new_neighbour = copy.copy(current_solution)
        if(bit==1):
            bit = 0
```

```
        else: bit = 1
        new_neighbour[count] = bit
        possible_neighbours.append(new_neighbour)
    return possible_neighbours
```

[8]:
```
print(Neighbourhood(initial3))
```

```
[[1, 0, 0, 0, 0, 0, 0, 0, 0], [0, 1, 0, 0, 0, 0, 0, 0, 0], [0, 0, 1, 0, 0, 0, 0,
0, 0], [0, 0, 0, 1, 0, 0, 0, 0, 0], [0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 0, 0, 0, 0,
1, 0, 0, 0], [0, 0, 0, 0, 0, 0, 1, 0, 0], [0, 0, 0, 0, 0, 0, 0, 1, 0], [0, 0, 0,
0, 0, 0, 0, 0, 1]]
```

[9]:
```
def random_walk(initial):
    walk_count = 0
    landscape = []
    current_solution = initial
    while(walk_count<100):
        walk_count+=1
        weight = calc_weight(initial)
        max_value = calc_value(current_solution)
        sol_list = []
        for neighbour in Neighbourhood(current_solution):
            value = calc_value(neighbour)
            weight = calc_weight(neighbour)
            attributes = {"vector":neighbour,"value":value,"weight":weight}
            sol_list.append(attributes)
        temp_max = max_value
        temp_solution = current_solution
        #print("\n")
        test = random.choice(sol_list)
        if(test['weight']>16):
            test = random.choice(sol_list)
        landscape.append(test['value'])
        current_solution = test['vector']

    return landscape
```

[10]:
```
walk = random_walk(initial3)
```

[11]:
```
print(walk)
```

```
[10, 0, 8, 20, 34, 45, 51, 64, 56, 42, 36, 45, 55, 43, 55, 46, 34, 43, 33, 39,
46, 35, 26, 34, 27, 39, 26, 18, 31, 18, 31, 18, 28, 41, 50, 38, 46, 33, 45, 58,
48, 35, 27, 41, 49, 35, 49, 56, 50, 41, 47, 35, 29, 38, 30, 21, 29, 22, 29, 40,
26, 39, 31, 20, 31, 20, 30, 23, 13, 0, 12, 19, 33, 39, 32, 26, 34, 41, 33, 46,
```

```
39, 27, 13, 24, 34, 43, 49, 61, 69, 56, 44, 38, 30, 21, 29, 43, 55, 47, 54, 43]
```

[12]:
```
var = np.var(walk) #variance
m=len(walk)
d = 1
m_d = m-d
r = ((1/(m_d)*var))
ave = sum(walk)/len(walk)
result = 0
for count,entry in enumerate(walk[:-d]):
    result += (entry-ave)*(walk[count]-ave)
r = r*result
print("r(1)=",r)
```

```
r(1)= 33025.10571874999
```

[13]:
```
L = 1/(np.log(r))
#now to normalise
#Diam(G) = the max distance between [0,0,0,0,0,1,0,1,0] and [1,1,1,1,1,0,1,0,1]␣
↪which corresponds
#to weight 15 (this is max moves without exceeding weight limit). This is 9␣
↪moves. Remove the two and add the rest.
epsilon = L/9
print("xi:",epsilon)
```

```
xi: 0.010678602784559484
```

[14]:
```
#Question 3b
U = []
for entry in Neighbourhood(initial3):
    for pos in Neighbourhood(entry):
        if(pos.count(1)==2):
            U.append(pos)
U = list(set(tuple(i)for i in U)) #removing any duplicates from process of␣
↪making U
U = list(list(i)for i in U)
#for sanity sake here is the length of U
len(U)
```

[14]: 36

[15]:
```
def find_solution(initial):
    iteration = -1
    weight = calc_weight(initial)
    current_solution = initial
```

```
    max_value = calc_value(current_solution)
    print("Starting point = \t  ",initial,"value = ",max_value)
    while(weight<16):
        iteration+=1
        sol_list = []
        for neighbour in Neighbourhood(current_solution): #creates a dictionary␣
 ↪with the neighbourhood and
                                                          #each of the␣
 ↪potential solutions attributes
            value = calc_value(neighbour)
            weight = calc_weight(neighbour)
            attributes = {"vector":neighbour,"value":value,"weight":weight}
            sol_list.append(attributes)
        temp_max = max_value
        temp_solution = current_solution
        for entry in sol_list: #check the validty of each neighbourhood␣
 ↪solution and decide if it is a new best
            if(entry['value']>temp_max and entry['weight']<=16):
                temp_solution = entry['vector']
                temp_max = entry['value']
        if(temp_max>max_value):
            current_solution = temp_solution
            max_value = temp_max


        elif(temp_max<=max_value):
            #print("\n No new solution found")
            break
    return current_solution,iteration,max_value
```

[16]:
```
lengths = []
solutions = []
for initial in U:
    sol,it,value = find_solution(initial)
    print("took",it,"to find solution = ",sol,"value = ",value,"\n")
    lengths.append(it)
    solutions.append(sol)
```

```
Starting point =           [0, 1, 0, 0, 0, 0, 0, 1, 0] value =  18
took 4 to find solution =  [1, 1, 0, 1, 1, 0, 0, 1, 1] value =  65

Starting point =           [0, 0, 0, 0, 0, 1, 0, 1, 0] value =  17
took 3 to find solution =  [1, 0, 0, 0, 1, 1, 0, 1, 1] value =  52

Starting point =           [1, 0, 0, 0, 1, 0, 0, 0, 0] value =  22
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74
```

```
Starting point =           [0, 0, 1, 0, 1, 0, 0, 0, 0] value =  23
took 5 to find solution =  [1, 1, 1, 1, 1, 0, 1, 0, 1] value =  73

Starting point =           [1, 0, 1, 0, 0, 0, 0, 0, 0] value =  17
took 5 to find solution =  [1, 1, 1, 1, 1, 0, 1, 0, 1] value =  73

Starting point =           [1, 0, 0, 0, 0, 0, 0, 1, 0] value =  15
took 4 to find solution =  [1, 1, 0, 1, 1, 0, 0, 1, 1] value =  65

Starting point =           [0, 1, 0, 0, 0, 0, 1, 0, 0] value =  17
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 0, 0, 0, 0, 1, 0, 0, 1] value =  23
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [1, 0, 0, 0, 0, 0, 0, 0, 1] value =  21
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 0, 0, 0, 0, 0, 1, 1, 0] value =  13
took 4 to find solution =  [0, 1, 0, 1, 1, 0, 1, 1, 1] value =  63

Starting point =           [0, 0, 1, 0, 0, 1, 0, 0, 0] value =  19
took 4 to find solution =  [1, 0, 1, 1, 1, 1, 0, 0, 1] value =  66

Starting point =           [0, 1, 0, 0, 0, 0, 0, 0, 1] value =  24
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 0, 0, 0, 1, 1, 0, 0, 0] value =  24
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 0, 0, 0, 0, 1, 1, 0, 0] value =  16
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [1, 0, 0, 0, 0, 0, 1, 0, 0] value =  14
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 1, 0, 0, 0, 1, 0, 0, 0] value =  21
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 0, 0, 1, 0, 0, 1, 0, 0] value =  18
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 0, 0, 0, 0, 0, 1, 0, 1] value =  19
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [1, 0, 0, 1, 0, 0, 0, 0, 0] value =  20
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74
```

```
Starting point =           [0, 0, 1, 0, 0, 0, 0, 1, 0] value =  16
took 3 to find solution =  [0, 0, 1, 1, 1, 0, 0, 1, 1] value =  55

Starting point =           [0, 0, 0, 0, 1, 0, 0, 1, 0] value =  21
took 4 to find solution =  [1, 1, 0, 1, 1, 0, 0, 1, 1] value =  65

Starting point =           [0, 0, 0, 1, 0, 0, 0, 0, 1] value =  25
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 0, 0, 1, 0, 0, 0, 1, 0] value =  19
took 4 to find solution =  [1, 1, 0, 1, 1, 0, 0, 1, 1] value =  65

Starting point =           [1, 0, 0, 0, 0, 1, 0, 0, 0] value =  18
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 0, 0, 0, 1, 0, 1, 0, 0] value =  20
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 0, 1, 0, 0, 0, 1, 0, 0] value =  15
took 5 to find solution =  [1, 1, 1, 1, 1, 0, 1, 0, 1] value =  73

Starting point =           [0, 0, 0, 0, 1, 0, 0, 0, 1] value =  27
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 0, 1, 0, 0, 0, 0, 0, 1] value =  22
took 5 to find solution =  [1, 1, 1, 1, 1, 0, 1, 0, 1] value =  73

Starting point =           [0, 1, 0, 1, 0, 0, 0, 0, 0] value =  23
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 0, 1, 1, 0, 0, 0, 0, 0] value =  21
took 5 to find solution =  [1, 1, 1, 1, 1, 0, 1, 0, 1] value =  73

Starting point =           [0, 1, 1, 0, 0, 0, 0, 0, 0] value =  20
took 5 to find solution =  [1, 1, 1, 1, 1, 0, 1, 0, 1] value =  73

Starting point =           [0, 0, 0, 1, 1, 0, 0, 0, 0] value =  26
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 1, 0, 0, 1, 0, 0, 0, 0] value =  25
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [1, 1, 0, 0, 0, 0, 0, 0, 0] value =  19
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74

Starting point =           [0, 0, 0, 0, 0, 0, 0, 1, 1] value =  20
took 4 to find solution =  [1, 1, 0, 1, 1, 0, 0, 1, 1] value =  65
```

```
Starting point =            [0, 0, 0, 1, 0, 1, 0, 0, 0] value =  22
took 5 to find solution =  [1, 1, 0, 1, 1, 1, 1, 0, 1] value =  74
```

[17]:
```
print("LLM(U)=",(sum(lengths)/len(lengths)))
```

```
LLM(U)= 4.694444444444445
```

[18]:
```
#Question 3c
#Assuming the mathematical definition of a set as mentioned in the question␣
 ↪where all elements must be unique.
#making sure there are no duplicates in the set O.
O = list(set(tuple(i)for i in solutions)) #removing any duplicates from process␣
 ↪of making O
O = list(list(i)for i in O)
#for sanity sake here is the length of O
print(len(O))
O
```

```
7
```

[18]:
```
[[1, 1, 1, 1, 1, 0, 1, 0, 1],
 [1, 1, 0, 1, 1, 0, 0, 1, 1],
 [0, 0, 1, 1, 1, 0, 0, 1, 1],
 [1, 0, 1, 1, 1, 1, 0, 0, 1],
 [1, 0, 0, 0, 1, 1, 0, 1, 1],
 [1, 1, 0, 1, 1, 1, 1, 0, 1],
 [0, 1, 0, 1, 1, 0, 1, 1, 1]]
```

[19]:
```
def Amp(P):
    vals = [] #array containing all fitness values (value of knapsack for given␣
 ↪solution set)
    for element in P:
        vals.append(calc_value(element))

    return (len(P)*(max(vals)-min(vals)))/sum(vals) #formula for Amp(P)
```

[20]:
```
Amp(U)
```

[20]: 0.7

[21]:
```
Amp(O)
```

[21]: 0.34375

[22]:
```
delta_amp = (Amp(U)-Amp(O))/Amp(U)
delta_amp
```

[22]: 0.5089285714285714

[23]:
```
def Gap(P):
    vals = []
    for element in P:
        vals.append(calc_value(element))
    array = []
    for val in vals:
        array.append(val-max(vals))
    return sum(array)/(len(P)*max(vals))
```

[24]:
```
Gap(O)
```

[24]: -0.13513513513513514

With a small gap it is a relatively easy problem to solve, LLM =4.7 means that there is a short distance to find the correct solution. With such a low value for xi(normalised correlation length) the landscape is not flat or smooth and as such there are many local optima to get trapped in. Combined with a delta amp being not very high and therefore not a large difference between the best solution and the worst it can be concluded that while this is not a difficult problem, it is not the easiest. It would not be very hard to solve the problem.

[ ]:
```

```