Laboratory 1

Import important libraries

In [1]:

```
import copy
import random
import numpy as np
import matplotlib.pyplot as plt
```

Loading data

Dataset, actualy flow and distances matrix can be found on http://anjos.mgi.polymtl.ca/qaplib/inst.html#HRW (http://anjos.mgi.polymtl.ca/qaplib/inst.html#HRW) website. Data in file are saved in the following order *n_facilities, flow_matrix, distance_matrix*.

In [2]:

```
data_path = "./data/had12.dat"
data_file = open(data_path, 'r')
n_facilities = int(data_file.readline().lstrip().rstrip())
data_file.close()

data_matrix = np.loadtxt(data_path, skiprows=2)
distance_matrix = data_matrix[:n_facilities]
flow_matrix = data_matrix[n_facilities:]
```

Quadratic assigment problem

The **quadratic assignment problem** (*QAP*) is one of the fundamental combinatorial optimization problems in the branch of optimization or operations research in mathematics, from the category of the facilities location problems.

The problem models the following real-life problem:

There are a **set of n facilities** and a **set of n locations**. For each pair of locations, a **distance** is specified and for each pair of facilities a **weight** or **flow** is specified (*e.g., the amount of supplies transported between the two facilities*). The problem is to *assign all facilities to different locations with the goal of minimizing the sum of the distances multiplied by the corresponding flows*. Intuitively, the cost function encourages factories with high flows between each other to be placed close together.

More infor can be found on: https://neos-guide.org/content/quadratic-assignment-problem (https://neos-guide.org/content/quadratic-assignment-problem)

COST Function

$$\sum_{a,b \in P} w(a,b) \cdot d(f(a), f(b))$$

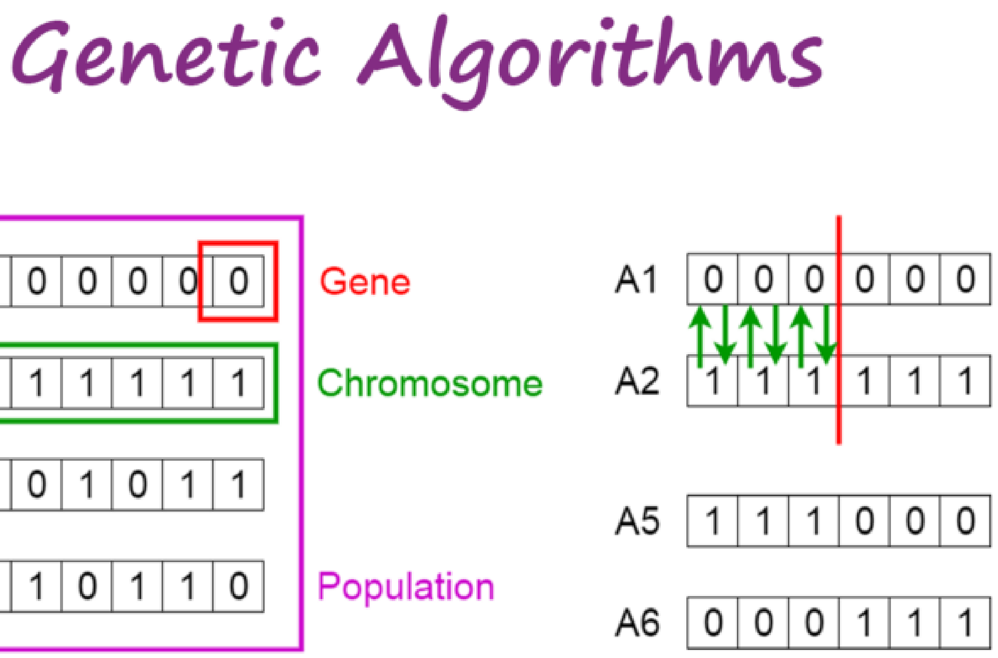
Genetic Algorithm

1. Introduction

A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation.

Popular terms in Genetic algorithm:

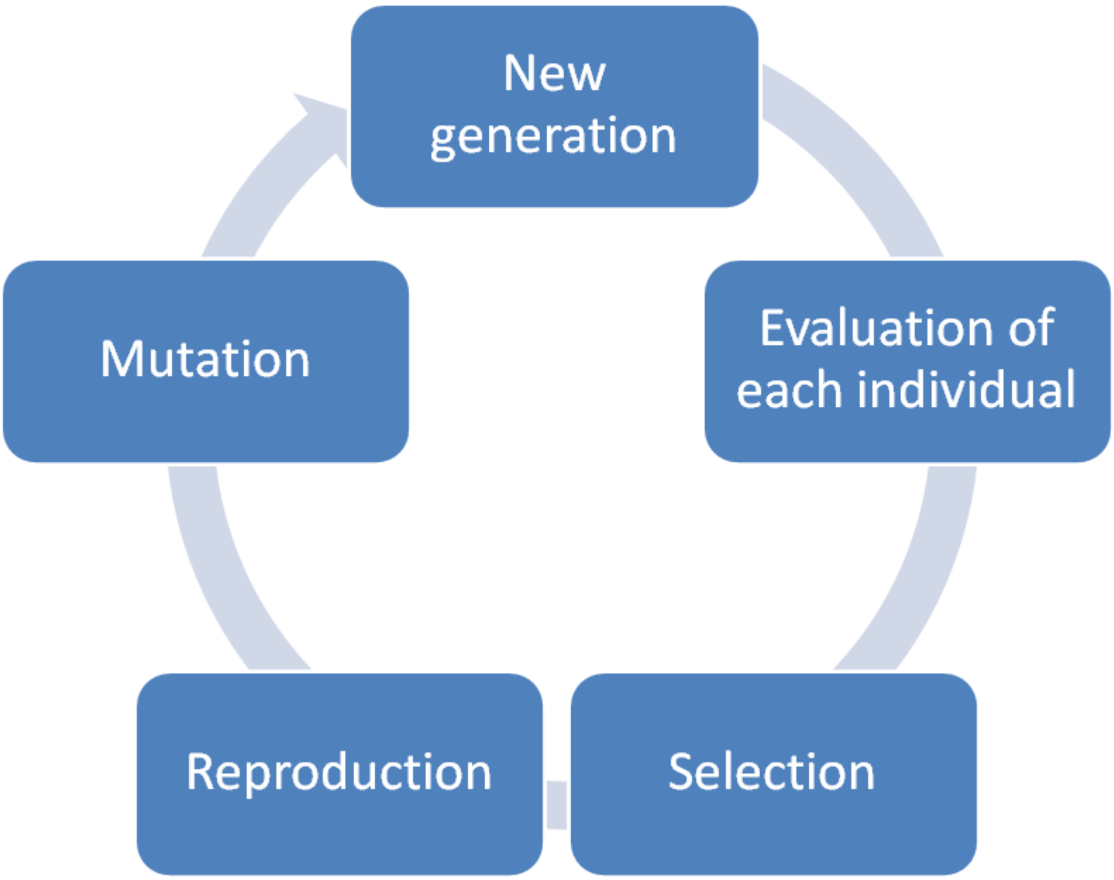
- Population - set of individuals,
- Genes - set of parameters (variables/features),
- Chromosome - solution,



2. Genetic algorithm steps

Genetic algorithm can be described in few very important steps, each of them is derived from evolution theory. Whole algorithm can be implemented in following steps:

- START
- Generate the initial population
- Compute fitness
- REPEAT
  - Selection
  - Crossover
  - Mutation
  - Compute fitness
- UNTIL population has converged
- STOP



Set initial parameters

Set initial parameters for genetic algorithm. The most important hyperparameters are population_size, crossover_proobability and mutation_probability

In [3]:

```
population_size = 1000
crossover_probability = 0.8
mutation_probability = 0.008
```

Generate random population

In first step Population (set of solutions is randomly generated)

In [4]:

```
def generate_random_population(n_facilities, n_chromosomes):
    population_list = []
    for ch_idx in range(n_chromosomes):
        rand_chromosome = list(range(0, n_facilities))
        random.shuffle(rand_chromosome)
        population_list.append(rand_chromosome)
    return population_list
```

Compute fitness

Fitness function is the most important element in implementing genetic algorithm. Best chromosomes will be choosen according to this function, so it is significant to choose it correctly. It gives a fitness score to each individual. The probability that an individual will be selected for reproduction is based on its fitness score.

In [5]:

```
def fitness_score(population_list, flow_matrix, distance_matrix):
    n_facilities = len(population_list[0])
    fitness_scores_list = []
    for chromosome_list in population_list:
        chromosome_fitness = 0
        for loc_idx_f in range(0, n_facilities):
            fac_und_loc_f =  chromosome_list[loc_idx_f]
            for loc_idx_s in range(0, n_facilities):
                fac_und_loc_s =  chromosome_list[loc_idx_s]
                ft_s = flow_matrix[fac_und_loc_f, fac_und_loc_s] * distance_matr
ix[loc_idx_f, loc_idx_s]
                chromosome_fitness += ft_s
        fitness_scores_list.append(1. / (chromosome_fitness / 2.))
    return fitness_scores_list
print(1. / fitness_score([[4, 3, 6, 0, 1, 2, 5]], flow_matrix, distance_matrix)
[0])
```

```
213.0
```

There are few requirements for fitness score, first of them tell that fitness function can't be negative and second tells that results should sum to one.

In [6]:

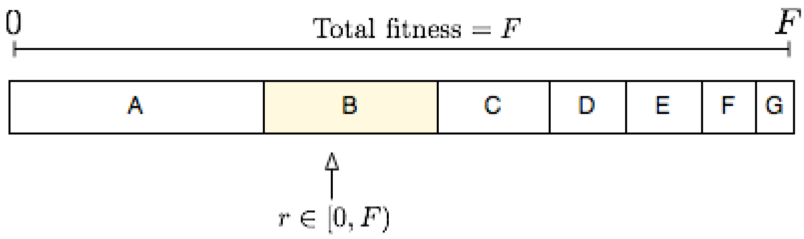
```
def normalise_fitness_score(fitness_score):
    return np.array(fitness_score) / np.sum(fitness_score)
```

Selection

To select the best individuals from the population, the selection method is used. In selection process few variation can be met like **roulette selection**, **tournament selection**, **elitism**.

Roulette selection

Roulette selection uses mechanism very similiar to roulette wheel in gambling. Actualy it uses cumulative distribution to select appropriate chromosomes. This method is very similiar to inverse-transform methods used for sampling in Machine Learning.



In [7]:

```
def roulette_selection(population_list, fitness_scores_list, elitism=True):
    new_species = []
    population_size = len(fitness_scores_list)
    population_size = population_size - 1 if elitism else population_size

    cum_sum = np.cumsum(fitness_scores_list, axis=0)
    for _ in range(0, population_size):
        rnd = random.uniform(0, 1)
        # If small than first
        if rnd < cum_sum[0]:
            new_species.append(population_list[0])
            continue
        # for others
        counter = 0
        while rnd > cum_sum[counter]:
            counter += 1
        new_species.append(population_list[counter])
    new_species.append(population_list[np.argmax(fitness_scores_list)])
    return new_species
```

Tournament selection

Tournament selection is a method of selecting an individual from a population of individuals in a genetic algorithm. Tournament selection involves **running several "tournaments"** among a few individuals (or "chromosomes") chosen at random from the population. **The winner of each tournament (the one with the best fitness) is selected for crossover**.

- Choose few individuals at random from the population (a tournament).
- The individual with the best fitness (the winner) is selected for crossover.

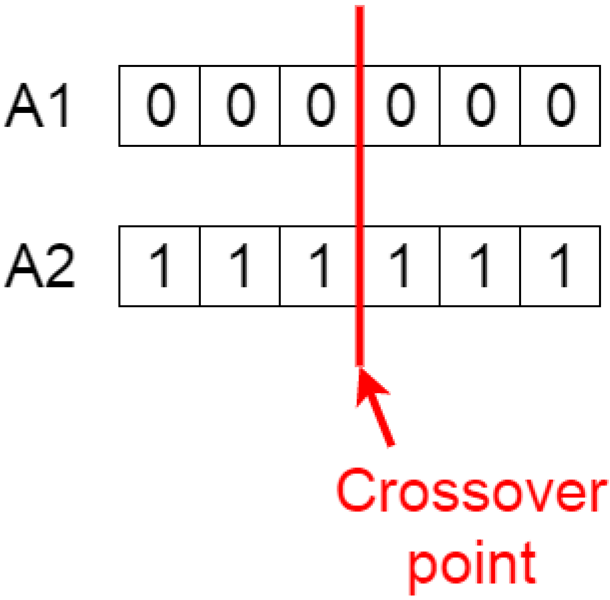
In [8]:

```
def tournament_selection(population_list, fitness_scores_list, elitism=True):
    # Create new population
    new_species = []
    population_size = len(fitness_scores_list)
    population_size = population_size - 1 if elitism else population_size
    for _ in range(0, population_size):
        # Take best
        of_parent_idx = random.randint(0, len(fitness_scores_list) - 1)
        tf_parent_idx = random.randint(0, len(fitness_scores_list) - 1)
        if fitness_scores_list[of_parent_idx] > fitness_scores_list[tf_parent_idx]:
            ch_winner = population_list[of_parent_idx]
        else:
            ch_winner = population_list[tf_parent_idx]
        new_species.append(ch_winner)
    new_species.append(population_list[np.argmax(fitness_scores_list)])
    return new_species
```

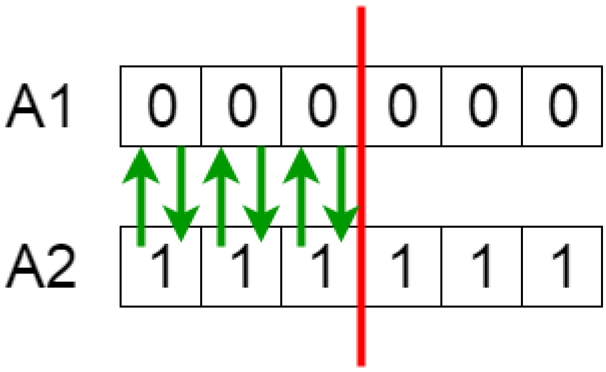
Crossover

Crossover is the most significant phase in a genetic algorithm. For each pair of parents to be mated, a crossover point is chosen at random from within the genes.

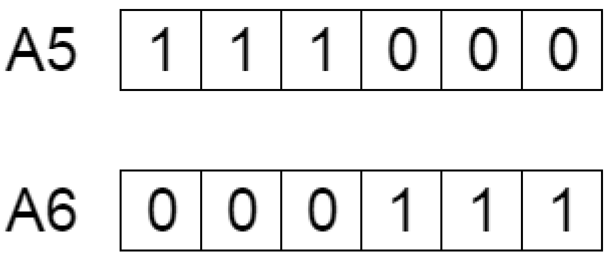
For example, consider the crossover point to be 3 as shown below.



Offspring are created by exchanging the genes of parents among themselves until the crossover point is reached.



The new offspring are added to the population.



In [9]:

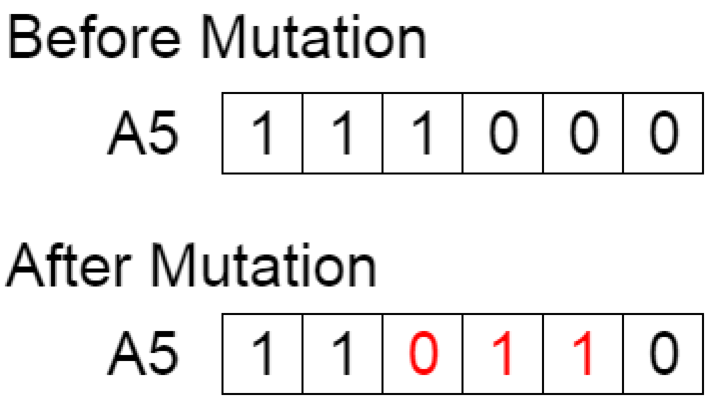
```
def chromosome_crossover(chromosome_o, chromosome_s):
    # Random point to crossover
    chr_o, chr_s = copy.copy(chromosome_o), copy.copy(chromosome_s)
    pos = random.randint(0, len(chromosome_o) - 1)

    # Change chromosome
    for ch_idx in range(0, pos):
        # values on ind
        fac_o = chr_o[ch_idx]
        fac_s = chr_s[ch_idx]

        # Values for swap
        fac_os_idx = chr_o.index(fac_s)
        fac_so_idx = chr_s.index(fac_o)

        # Save values
        chr_o[fac_os_idx] = fac_o
        chr_s[fac_so_idx] = fac_s

        # Change values
        chr_o[ch_idx] = fac_s
        chr_s[ch_idx] = fac_o
    return chr_o, chr_s
```

In [10]:

```
def crossover_population(new_species, crossover_probability):
    # Select for crossover
    species_nc = []
    crossover_list = []
    for n_chrom in new_species:
        rnd = random.uniform(0, 1)
        if rnd < crossover_probability:
            crossover_list.append(n_chrom)
        else:
            species_nc.append(n_chrom)

    crossover_tuples = []
    # Create crossover buddies
    cr_iterate = list(enumerate(crossover_list))
    while cr_iterate:
        cch_idx, c_chrom = cr_iterate.pop()
        if not cr_iterate:
            species_nc.append(c_chrom)
            break
        cb_idx, cross_buddy = random.choice(cr_iterate)
        cr_iterate = [(x_k, x_v) for x_k, x_v in cr_iterate if x_k != cb_idx]
        crossover_tuples.append((c_chrom, cross_buddy))

    # Crossover to list
    after_cover = []
    for cr_tup in crossover_tuples:
        cr_o, cr_t = chromosome_crossover(cr_tup[0], cr_tup[1])
        after_cover.append(cr_o)
        after_cover.append(cr_t)

    # New population
    new_species = after_cover + species_nc
    return new_species
# new_species = [[7, 4, 3, 2, 5, 6, 1, 8], [4, 6, 5, 2, 1, 8, 7, 3]]
# print(crossover_population(new_species, 1.0))
```

Mutation

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. It is analogous to biological mutation. Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to a better solution by using mutation. Mutation occurs during evolution according to a user-definable mutation probability. This probability should be set low. If it is set too high, the search will turn into a primitive random search.



In TSP and QAP problem mutation will have slightly different form. We will choose two genes and swap them.

In [11]:

```
def mutation_population(new_species, mutation_probability):
    # # MUTATION
    mutated = []
    for chromosome in new_species:
        for b_idx in range(0, len(chromosome)):
            rnd = random.uniform(0, 1)
            if rnd < mutation_probability:
                swap_idx = random.randint(0, len(chromosome) - 1)
                old_mut_val = chromosome[b_idx]
                chromosome[b_idx] = chromosome[swap_idx]
                chromosome[swap_idx] = old_mut_val
        mutated.append(chromosome)
    return mutated
```

Concatenate all steps

In [12]:

```
population_list = generate_random_population(n_facilities, population_size)
for epoch in range(0, 100000):
    fit_scores = fitness_score(population_list, flow_matrix, distance_matrix)
    fit_scores_norm = normalise_fitness_score(fit_scores)
    selected_ch = tournament_selection(population_list, fit_scores_norm,
elitism=False)
    crossed_ch = crossover_population(selected_ch, crossover_probability)
    mutated_ch = mutation_population(crossed_ch, mutation_probability)
    max_fitness = np.max(fit_scores)
    max_chromosome = population_list[np.argmax(fit_scores)]
    max_chromosome = [x + 1 for x in max_chromosome]
    print("Epoch: {}, Population fitness score: {}, Max score: {}, Max chromosom
e: {}".format(epoch, 1. / np.mean(fit_scores),

              1. / max_fitness, max_chromosome))
    population_list = crossed_ch
```

```
Epoch: 0, Population fitness score: 943.135661066, Max score: 871.0,
Max chromosome: [8, 1, 7, 5, 6, 11, 12, 10, 3, 2, 4, 9]
Epoch: 1, Population fitness score: 935.06002662, Max score: 866.0,
 Max chromosome: [8, 11, 4, 3, 2, 7, 6, 10, 12, 5, 1, 9]
Epoch: 2, Population fitness score: 931.612001164, Max score: 864.0,
Max chromosome: [8, 3, 7, 2, 5, 12, 11, 6, 1, 10, 4, 9]
Epoch: 3, Population fitness score: 930.179909126, Max score: 864.0,
Max chromosome: [3, 8, 7, 2, 5, 12, 11, 6, 1, 10, 4, 9]
Epoch: 4, Population fitness score: 927.841822936, Max score: 852.0,
Max chromosome: [8, 10, 12, 11, 2, 7, 5, 6, 3, 4, 1, 9]
Epoch: 5, Population fitness score: 926.403306656, Max score: 852.0,
Max chromosome: [8, 10, 12, 11, 2, 7, 5, 6, 3, 4, 1, 9]
Epoch: 6, Population fitness score: 924.898971632, Max score: 841.0,
Max chromosome: [3, 12, 10, 11, 2, 5, 6, 7, 8, 1, 9, 4]
Epoch: 7, Population fitness score: 923.18868538, Max score: 841.0,
 Max chromosome: [3, 12, 10, 11, 2, 5, 6, 7, 8, 1, 9, 4]
Epoch: 8, Population fitness score: 925.11052329, Max score: 841.0,
 Max chromosome: [3, 12, 10, 11, 2, 5, 6, 7, 8, 1, 9, 4]
Epoch: 9, Population fitness score: 922.398905929, Max score: 841.0,
Max chromosome: [3, 10, 6, 2, 7, 5, 12, 11, 8, 1, 9, 4]
Epoch: 10, Population fitness score: 922.064181757, Max score: 837.
0, Max chromosome: [10, 3, 11, 2, 12, 5, 6, 7, 8, 1, 9, 4]
Epoch: 11, Population fitness score: 920.797286201, Max score: 837.
0, Max chromosome: [10, 3, 11, 2, 12, 5, 6, 7, 8, 1, 9, 4]
Epoch: 12, Population fitness score: 919.541850687, Max score: 837.
0, Max chromosome: [10, 3, 11, 2, 12, 5, 6, 7, 8, 1, 9, 4]
Epoch: 13, Population fitness score: 920.737834849, Max score: 837.
0, Max chromosome: [10, 3, 11, 2, 12, 5, 6, 7, 8, 1, 9, 4]
Epoch: 14, Population fitness score: 920.747139211, Max score: 837.
0, Max chromosome: [10, 3, 11, 2, 12, 5, 6, 7, 8, 1, 9, 4]
Epoch: 15, Population fitness score: 919.350738404, Max score: 837.
0, Max chromosome: [10, 3, 11, 2, 12, 5, 6, 7, 8, 1, 9, 4]
Epoch: 16, Population fitness score: 920.107430861, Max score: 827.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 6, 7, 8, 1, 9, 4]
Epoch: 17, Population fitness score: 922.540903402, Max score: 842.
0, Max chromosome: [3, 10, 8, 2, 12, 5, 6, 7, 11, 1, 9, 4]
Epoch: 18, Population fitness score: 920.860279231, Max score: 839.
0, Max chromosome: [9, 4, 5, 7, 6, 2, 12, 11, 1, 10, 8, 3]
Epoch: 19, Population fitness score: 919.378903035, Max score: 842.
0, Max chromosome: [9, 4, 5, 7, 6, 2, 12, 11, 1, 10, 8, 3]
Epoch: 20, Population fitness score: 919.390400153, Max score: 842.
0, Max chromosome: [9, 4, 5, 7, 6, 2, 12, 11, 1, 10, 8, 3]
Epoch: 21, Population fitness score: 918.138743692, Max score: 849.
0, Max chromosome: [3, 6, 10, 2, 7, 1, 12, 11, 8, 5, 4, 9]
Epoch: 22, Population fitness score: 916.317736585, Max score: 847.
0, Max chromosome: [8, 2, 7, 10, 6, 11, 5, 1, 3, 12, 4, 9]
Epoch: 23, Population fitness score: 915.122170792, Max score: 847.
0, Max chromosome: [8, 2, 7, 10, 6, 11, 5, 1, 3, 12, 4, 9]
Epoch: 24, Population fitness score: 915.112853056, Max score: 851.
0, Max chromosome: [3, 5, 10, 2, 7, 6, 11, 1, 8, 12, 4, 9]
Epoch: 25, Population fitness score: 916.112120315, Max score: 849.
0, Max chromosome: [3, 2, 10, 7, 6, 5, 11, 12, 8, 1, 9, 4]
Epoch: 26, Population fitness score: 915.803466659, Max score: 849.
0, Max chromosome: [3, 2, 10, 7, 6, 5, 11, 12, 8, 1, 9, 4]
Epoch: 27, Population fitness score: 913.265574288, Max score: 837.
0, Max chromosome: [3, 10, 12, 2, 11, 5, 6, 7, 8, 4, 1, 9]
Epoch: 28, Population fitness score: 913.883832543, Max score: 849.
0, Max chromosome: [3, 2, 10, 7, 6, 5, 11, 12, 8, 1, 9, 4]
Epoch: 29, Population fitness score: 912.980106611, Max score: 847.
0, Max chromosome: [8, 6, 10, 7, 2, 11, 12, 1, 3, 5, 9, 4]
Epoch: 30, Population fitness score: 912.900551794, Max score: 852.
```

```
0, Max chromosome: [3, 10, 11, 12, 6, 7, 2, 1, 8, 5, 9, 4]
Epoch: 31, Population fitness score: 911.457518934, Max score: 853.
0, Max chromosome: [3, 2, 6, 5, 7, 1, 12, 10, 8, 11, 4, 9]
Epoch: 32, Population fitness score: 909.521426711, Max score: 849.
0, Max chromosome: [3, 2, 12, 6, 5, 7, 11, 4, 10, 1, 9, 8]
Epoch: 33, Population fitness score: 908.256561756, Max score: 845.
0, Max chromosome: [3, 10, 2, 6, 12, 5, 11, 7, 8, 1, 9, 4]
Epoch: 34, Population fitness score: 906.701619348, Max score: 827.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 9, 4]
Epoch: 35, Population fitness score: 905.786969951, Max score: 845.
0, Max chromosome: [3, 10, 12, 2, 11, 5, 7, 9, 8, 6, 1, 4]
Epoch: 36, Population fitness score: 904.678310478, Max score: 840.
0, Max chromosome: [3, 2, 7, 10, 6, 12, 5, 1, 8, 11, 4, 9]
Epoch: 37, Population fitness score: 901.607136411, Max score: 840.
0, Max chromosome: [3, 10, 5, 2, 7, 6, 12, 11, 8, 1, 4, 9]
Epoch: 38, Population fitness score: 898.722854796, Max score: 843.
0, Max chromosome: [3, 8, 5, 12, 11, 6, 7, 1, 2, 10, 4, 9]
Epoch: 39, Population fitness score: 895.897010854, Max score: 835.
0, Max chromosome: [3, 10, 2, 5, 12, 11, 6, 1, 8, 7, 9, 4]
Epoch: 40, Population fitness score: 894.136926981, Max score: 835.
0, Max chromosome: [3, 10, 2, 5, 12, 11, 6, 1, 8, 7, 9, 4]
Epoch: 41, Population fitness score: 893.423798053, Max score: 835.
0, Max chromosome: [3, 10, 2, 5, 12, 11, 6, 1, 8, 7, 9, 4]
Epoch: 42, Population fitness score: 888.907322457, Max score: 835.
0, Max chromosome: [3, 10, 11, 5, 12, 2, 6, 1, 8, 7, 9, 4]
Epoch: 43, Population fitness score: 885.683893032, Max score: 834.
0, Max chromosome: [3, 10, 11, 2, 12, 7, 6, 5, 8, 1, 4, 9]
Epoch: 44, Population fitness score: 884.704085338, Max score: 834.
0, Max chromosome: [3, 10, 11, 2, 12, 7, 6, 5, 8, 1, 4, 9]
Epoch: 45, Population fitness score: 882.620286101, Max score: 835.
0, Max chromosome: [3, 10, 11, 2, 12, 7, 6, 5, 8, 1, 9, 4]
Epoch: 46, Population fitness score: 880.078105531, Max score: 836.
0, Max chromosome: [3, 10, 11, 5, 12, 1, 2, 6, 8, 7, 4, 9]
Epoch: 47, Population fitness score: 876.858566511, Max score: 833.
0, Max chromosome: [3, 10, 2, 11, 12, 5, 6, 7, 8, 1, 9, 4]
Epoch: 48, Population fitness score: 874.300857699, Max score: 832.
0, Max chromosome: [3, 10, 5, 2, 12, 7, 11, 1, 8, 6, 4, 9]
Epoch: 49, Population fitness score: 872.037583439, Max score: 827.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 9, 4]
Epoch: 50, Population fitness score: 869.105252162, Max score: 827.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 9, 4]
Epoch: 51, Population fitness score: 867.561174701, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 6, 7, 8, 1, 4, 9]
Epoch: 52, Population fitness score: 866.321217844, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 6, 7, 8, 1, 4, 9]
Epoch: 53, Population fitness score: 863.268518851, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 6, 7, 8, 1, 4, 9]
Epoch: 54, Population fitness score: 862.201866356, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 6, 7, 8, 1, 4, 9]
Epoch: 55, Population fitness score: 859.234647083, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 6, 7, 8, 1, 4, 9]
Epoch: 56, Population fitness score: 856.924646441, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 57, Population fitness score: 855.124549087, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 58, Population fitness score: 851.383622781, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 59, Population fitness score: 849.252226396, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 60, Population fitness score: 847.446978192, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
```

```
Epoch: 61, Population fitness score: 846.136302297, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 62, Population fitness score: 844.157485896, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 63, Population fitness score: 842.522820676, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 64, Population fitness score: 840.531886664, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 65, Population fitness score: 839.724426361, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 66, Population fitness score: 838.2739618, Max score: 826.0,
 Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 67, Population fitness score: 837.210419602, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 6, 7, 8, 1, 4, 9]
Epoch: 68, Population fitness score: 835.917307699, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 6, 7, 8, 1, 4, 9]
Epoch: 69, Population fitness score: 834.860859153, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 6, 7, 8, 1, 4, 9]
Epoch: 70, Population fitness score: 833.681291809, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 71, Population fitness score: 832.534236011, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 72, Population fitness score: 831.644752445, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 73, Population fitness score: 831.615689513, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 74, Population fitness score: 831.799228851, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 75, Population fitness score: 831.676966088, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 76, Population fitness score: 831.509222553, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 77, Population fitness score: 830.326839934, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 78, Population fitness score: 830.343595457, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 79, Population fitness score: 829.617783992, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 80, Population fitness score: 829.042728341, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 81, Population fitness score: 828.359791305, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 82, Population fitness score: 828.321759622, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 83, Population fitness score: 829.055836063, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 84, Population fitness score: 829.455540777, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 85, Population fitness score: 829.218555641, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 86, Population fitness score: 828.631154217, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 87, Population fitness score: 829.337243569, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 88, Population fitness score: 829.272946361, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 89, Population fitness score: 828.745036308, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 90, Population fitness score: 828.900269247, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 91, Population fitness score: 828.795340426, Max score: 826.
```

```
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 92, Population fitness score: 828.33547284, Max score: 826.0,
Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 93, Population fitness score: 828.820567227, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 94, Population fitness score: 828.911297675, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 95, Population fitness score: 829.162542961, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 96, Population fitness score: 829.676521471, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 97, Population fitness score: 829.361468193, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 98, Population fitness score: 829.193098405, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 99, Population fitness score: 828.728951234, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 100, Population fitness score: 828.82706106, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 101, Population fitness score: 828.233485798, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 102, Population fitness score: 828.760258164, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 103, Population fitness score: 828.774793988, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 104, Population fitness score: 828.594134412, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 105, Population fitness score: 829.0190958, Max score: 826.0,
Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 106, Population fitness score: 829.461248564, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]
Epoch: 107, Population fitness score: 829.178988115, Max score: 826.
0, Max chromosome: [3, 10, 11, 2, 12, 5, 7, 6, 8, 1, 4, 9]

--------------------------------------------------------------------
-------
KeyboardInterrupt                         Traceback (most recent cal
l last)
<ipython-input-12-ba509c5b4829> in <module>()
      1 population_list = generate_random_population(n_facilities, p
opulation_size)
      2 for epoch in range(0, 100000):
----> 3     fit_scores = fitness_score(population_list, flow_matrix,
 distance_matrix)
      4     fit_scores_norm = normalise_fitness_score(fit_scores)
      5     selected_ch = tournament_selection(population_list, fit_
scores_norm, elitism=False)

<ipython-input-5-a9390cfeb737> in fitness_score(population_list, flo
w_matrix, distance_matrix)
      8             for loc_idx_s in range(0, n_facilities):
      9                 fac_und_loc_s =  chromosome_list[loc_idx_s]
---> 10                 ft_s = flow_matrix[fac_und_loc_f, fac_und_lo
c_s] * distance_matrix[loc_idx_f, loc_idx_s]
     11                 chromosome_fitness += ft_s
     12         fitness_scores_list.append(1. / (chromosome_fitness
/ 2.))

KeyboardInterrupt:
```

In [ ]:

```

```