Travel salesman Problem Genetic algorithms have a variety of applications, and one of the basic applications of genetic algorithms can be the optimization of problems and solutions. Use optimization for finding the best solution to any problem. Optimization using genetic algorithms can be considered genetic optimization

https://en.wikipedia.org/wiki/Travelling_salesman_problem

```
# Imports
import numpy as np
import random

from datetime import datetime


# Parameters
n_cities = 20

n_population = 100

mutation_rate = 0.3


# Generating a list of coordenades representing each city
coordinates_list = [[x,y] for x,y in zip(np.random.randint(0,100,n_cities),np.random.randint(0,100,n_cities))]
names_list = np.array(['Berlin', 'London', 'Moscow', 'Barcelona', 'Rome', 'Paris', 'Vienna', 'Munich', 'Istanbul', 'Kyiv', 'Bucharest', 'Mins
cities_dict = { x:y for x,y in zip(names_list,coordinates_list)}

# Function to compute the distance between two points
def compute_city_distance_coordinates(a,b):
    return ((a[0]-b[0])**2+(a[1]-b[1])**2)**0.5

def compute_city_distance_names(city_a, city_b, cities_dict):
    return compute_city_distance_coordinates(cities_dict[city_a], cities_dict[city_b])

cities_dict
```

```
 {'Berlin': [40, 25],
  'London': [45, 33],
  'Moscow': [46, 84],
  'Barcelona': [82, 0],
  'Rome': [77, 48],
  'Paris': [6, 72],
  'Vienna': [98, 67],
  'Munich': [28, 56],
  'Istanbul': [43, 89],
  'Kyiv': [45, 85],
  'Bucharest': [99, 95],
  'Minsk': [82, 7],
  'Warsaw': [27, 22],
  'Budapest': [56, 77],
  'Milan': [98, 17],
  'Prague': [71, 57],
  'Sofia': [49, 22],
  'Birmingham': [37, 54],
  'Brussels': [92, 68],
  'Amsterdam': [33, 75]}
```

```
# First step: Create the first population set
def genesis(city_list, n_population):

    population_set = []
    for i in range(n_population):
        #Randomly generating a new solution
        sol_i = city_list[np.random.choice(list(range(n_cities)), n_cities, replace=False)]
        population_set.append(sol_i)
    return np.array(population_set)

population_set = genesis(names_list, n_population)
population_set
```

```
 array([['Milan', 'Barcelona', 'Vienna', ..., 'Kyiv', 'Berlin', 'Moscow'],
        ['Berlin', 'London', 'Budapest', ..., 'Bucharest', 'Rome',
         'Barcelona'],
        ['Moscow', 'Paris', 'Berlin', ..., 'Amsterdam', 'Bucharest',
         'Minsk'],
        ...,
```

```
            ['Amsterdam', 'Warsaw', 'Rome', ..., 'Bucharest', 'Prague',
             'London'],
            ['Barcelona', 'Berlin', 'Paris', ..., 'Bucharest', 'Minsk',
             'Budapest'],
            ['Budapest', 'Moscow', 'Brussels', ..., 'Bucharest', 'Minsk',
             'Paris']], dtype='<U10')
```

```
def fitness_eval(city_list, cities_dict):
    total = 0
    for i in range(n_cities-1):
        a = city_list[i]
        b = city_list[i+1]
        total += compute_city_distance_names(a,b, cities_dict)
    return total
```

```
  def get_all_fitnes(population_set, cities_dict):
    fitnes_list = np.zeros(n_population)

    #Looping over all solutions computing the fitness for each solution
    for i in  range(n_population):
        fitnes_list[i] = fitness_eval(population_set[i], cities_dict)

    return fitnes_list

fitnes_list = get_all_fitnes(population_set,cities_dict)
fitnes_list
```

```
     array([ 898.99469475,  882.85418748, 1100.11479126,  946.36415103,
             871.67572296, 1039.5913772 ,  848.10247503, 1050.8802423 ,
             955.01966595, 1159.10070297,  954.21121773, 1069.2026347 ,
             948.938909  , 1058.16925776, 1038.73354695, 1018.32388418,
            1027.57053977,  924.71507825,  847.46986004,  935.30790558,
            1025.90457391,  960.51620294,  948.79425209,  882.40704462,
             810.66030964,  835.78340877, 1085.34074437, 1073.30829206,
            1124.35423738, 1129.00910089,  890.05740812, 1040.50923559,
             984.62220291,  920.42659841,  938.92691874,  928.10233503,
            1069.9776899 , 1004.75085998,  909.44801133, 1077.30262702,
             856.73097288,  880.0335961 ,  951.84389454,  900.12359927,
            1261.79839539, 1038.76794467,  982.81167546,  935.47476431,
            1011.13529085, 1004.48543721,  810.09803542, 1047.7969524 ,
             913.75847096, 1002.96330918, 1058.4895399 ,  970.58505553,
            1032.79779834, 1128.4697065 ,  859.19446941,  994.21231553,
             965.18730762,  925.54262345, 1024.51573981,  988.78384524,
            1068.91906394,  968.24237062, 1084.81267705,  770.27966803,
            1016.52416962,  997.50947741,  976.21912806,  909.36905432,
            1069.75640841, 1141.13378458,  990.9400788 ,  963.7666192 ,
             964.1549149 ,  978.01309912,  963.32195588,  960.18224173,
            1072.39305174,  914.15776917, 1042.15300957,  896.30430677,
             980.98625243,  968.1943718 ,  865.35668976, 1134.71412559,
             979.75675685,  976.77880447,  911.12237453,  840.45218126,
            1141.15545867, 1026.96652855,  916.62307769, 1095.30522816,
             963.59027395, 1040.15930834, 1047.78253987,  916.76330229])
```

```
def progenitor_selection(population_set,fitnes_list):
    total_fit = fitnes_list.sum()
    prob_list = fitnes_list/total_fit

    #Notice there is the chance that a progenitor. mates with oneself
    progenitor_list_a = np.random.choice(list(range(len(population_set))), len(population_set),p=prob_list, replace=True)
    progenitor_list_b = np.random.choice(list(range(len(population_set))), len(population_set),p=prob_list, replace=True)

    progenitor_list_a = population_set[progenitor_list_a]
    progenitor_list_b = population_set[progenitor_list_b]


    return np.array([progenitor_list_a,progenitor_list_b])
```

```
progenitor_list = progenitor_selection(population_set,fitnes_list)
progenitor_list[0][2]
```

```
     array(['Munich', 'London', 'Barcelona', 'Paris', 'Bucharest', 'Prague',
            'Rome', 'Budapest', 'Vienna', 'Amsterdam', 'Kyiv', 'Berlin',
            'Sofia', 'Milan', 'Warsaw', 'Istanbul', 'Moscow', 'Minsk',
            'Brussels', 'Birmingham'], dtype='<U10')
```

```
def mate_progenitors(prog_a, prog_b):
    offspring = prog_a[0:5]

    for city in prog_b:

        if not city in offspring:
            offspring = np.concatenate((offspring,[city]))

    return offspring



def mate_population(progenitor_list):
    new_population_set = []
    for i in range(progenitor_list.shape[1]):
        prog_a, prog_b = progenitor_list[0][i], progenitor_list[1][i]
        offspring = mate_progenitors(prog_a, prog_b)
        new_population_set.append(offspring)

    return new_population_set

new_population_set = mate_population(progenitor_list)
new_population_set[0]

     array(['Sofia', 'Budapest', 'Berlin', 'Moscow', 'Prague', 'Paris',
            'Istanbul', 'Warsaw', 'London', 'Minsk', 'Vienna', 'Bucharest',
            'Kyiv', 'Birmingham', 'Amsterdam', 'Milan', 'Munich', 'Barcelona',
            'Rome', 'Brussels'], dtype='<U10')


def mutate_offspring(offspring):
    for q in range(int(n_cities*mutation_rate)):
        a = np.random.randint(0,n_cities)
        b = np.random.randint(0,n_cities)

        offspring[a], offspring[b] = offspring[b], offspring[a]

    return offspring


def mutate_population(new_population_set):
    mutated_pop = []
    for offspring in new_population_set:
        mutated_pop.append(mutate_offspring(offspring))
    return mutated_pop

mutated_pop = mutate_population(new_population_set)
mutated_pop[0]

     array(['Sofia', 'Budapest', 'Berlin', 'Moscow', 'Istanbul', 'Bucharest',
            'Barcelona', 'Warsaw', 'London', 'Vienna', 'Minsk', 'Paris',
            'Kyiv', 'Prague', 'Milan', 'Amsterdam', 'Munich', 'Birmingham',
            'Rome', 'Brussels'], dtype='<U10')


best_solution = [-1,np.inf,np.array([])]
for i in range(10000):
    if i%100==0: print(i, fitnes_list.min(), fitnes_list.mean(), datetime.now().strftime("%d/%m/%y %H:%M"))
    fitnes_list = get_all_fitnes(mutated_pop,cities_dict)

    #Saving the best solution
    if fitnes_list.min() < best_solution[1]:
        best_solution[0] = i
        best_solution[1] = fitnes_list.min()
        best_solution[2] = np.array(mutated_pop)[fitnes_list.min() == fitnes_list]

    progenitor_list = progenitor_selection(population_set,fitnes_list)
    new_population_set = mate_population(progenitor_list)

    mutated_pop = mutate_population(new_population_set)
```

```
5000 753.4448224652133 977.1177413939051 28/03/23 05:06
5100 718.4692195645611 991.1243328697954 28/03/23 05:07
5200 702.1003303489756 984.633354211684 28/03/23 05:07
5300 765.5184982196453 985.4579591636617 28/03/23 05:07
5400 828.4965865057676 974.984783063635 28/03/23 05:07
5500 734.2713154534049 981.5974048908496 28/03/23 05:07
5600 733.2468326192889 964.1616039406554 28/03/23 05:07
5700 766.0942033755032 972.3110584951542 28/03/23 05:07
5800 739.5723996435436 991.5384922420718 28/03/23 05:07
5900 726.1695798642232 977.9455337643553 28/03/23 05:07
6000 779.2567284877994 992.903406535533 28/03/23 05:07
6100 787.4414542147761 986.5529695557314 28/03/23 05:07
6200 733.5820346529707 982.0580324974896 28/03/23 05:07
6300 700.5621325749289 974.7041263313597 28/03/23 05:07
6400 812.1532690668797 988.0542241090426 28/03/23 05:07
6500 775.6645884434258 986.2251304202629 28/03/23 05:07
6600 732.7679706649909 981.2343531680814 28/03/23 05:07
6700 783.6693390156631 997.192698253225 28/03/23 05:07
6800 816.8505587736694 978.2225934120403 28/03/23 05:07
6900 734.0114891003376 991.1569671000673 28/03/23 05:07
7000 745.8124977502748 973.2116186749929 28/03/23 05:07
7100 826.1263806281926 990.7416353579254 28/03/23 05:08
7200 707.7894044170221 968.6615877238249 28/03/23 05:08
7300 729.5022233206532 1000.7940487749089 28/03/23 05:08
7400 752.2839665613564 986.8952823952241 28/03/23 05:08
7500 828.0754651328057 998.9642166136495 28/03/23 05:08
7600 746.1393931087591 980.2560232668502 28/03/23 05:08
7700 784.723387012894 989.6182215521493 28/03/23 05:08
7800 815.8890914271556 1004.1974185470639 28/03/23 05:08
7900 744.5314891921712 977.1975882229493 28/03/23 05:08
8000 786.972481762025 996.7146520657724 28/03/23 05:08
8100 747.4204201368992 961.6659912371092 28/03/23 05:08
8200 724.7354781805948 974.0289336524095 28/03/23 05:08
8300 814.9013977073683 983.788163918463 28/03/23 05:08
8400 736.9901017084769 981.8467659441955 28/03/23 05:08
8500 771.945508018802 997.3471216848903 28/03/23 05:08
8600 699.7072819297188 983.5051848584698 28/03/23 05:08
8700 838.1438804195731 989.0103045439517 28/03/23 05:08
8800 796.6979222858959 975.3991884631198 28/03/23 05:08
8900 806.0414561649119 984.6745173705812 28/03/23 05:08
9000 772.071342337953 980.6807375546279 28/03/23 05:08
9100 780.654475246742 975.5112735022293 28/03/23 05:09
9200 789.1315552286712 975.5627141480632 28/03/23 05:09
9300 755.8495068038047 983.9455007329058 28/03/23 05:09
9400 784.7933646655003 979.3015347778143 28/03/23 05:09
9500 745.6799931587389 977.5603596135818 28/03/23 05:09
9600 674.3089953363809 974.6195732369981 28/03/23 05:09
9700 800.861239227156 993.760073838869 28/03/23 05:09
9800 768.113883328225 982.7154189236884 28/03/23 05:09
9900 797.3666437353972 990.0003062598179 28/03/23 05:09
```

```
best_solution
```

```
[8060,
 573.9394114970532,
 array([['Prague', 'Rome', 'Minsk', 'Barcelona', 'Milan', 'Birmingham',
         'Munich', 'Amsterdam', 'Kyiv', 'Brussels', 'Bucharest', 'Vienna',
         'Budapest', 'Istanbul', 'Moscow', 'Warsaw', 'Berlin', 'Paris',
         'London', 'Sofia']], dtype='<U10')]
```

✓ 0s completed at 10:39 AM