Evrimsel Hesaplama Ödevi

Öğrenci İbrahim Mert Küni - 16253801

CENG 415 - Evrimsel Hesaplama

06 Ocak 2021

(1/2) Travelling Salesman Problem via Python DEAP

Merhabalar. Evrimsel Hesaplama dersinin ödevini Python Script şeklinde baştan sona gösterimini kod ile, anlatımı ekran görüntüsü ile yapmaya çalışacağım. 2 problem sonunda kaynak kodlari sade metin şeklinde belgenin en altında paylaşacağım.

Temsil biçimi, mutasyon ve çaprazlama yöntemleri, ebeveyn ve hayatta kalanların seçilme algoritmaları kullanımları: İçerik açıklaması anlam bozukluğu olmaması adına ingilizce bırakılmıştır. ilk önce kullanım şekli aşağıdaki resimdeki gibidir. Source github.com/imkuni

```
## permutation setup for individual,
toolbox.register("indices", random.sample, range(INDIVIDUAL_SIZE), INDIVIDUAL_SIZE)
# noinspection PyUnresolvedReferences
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.indices)

## population setup,
# noinspection PyUnresolvedReferences
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
######################################## 6
# noinspection PyShadowingNames
def EVALUATE(individual):
    summation = 0
    start = individual[0]
    for i in range(1, len(individual)):
        end = individual[i]
        summation += distances[start][end]
        start = end
    return summation

toolbox.register("evaluate", EVALUATE)
######################################## 7
toolbox.register("mate", tools.cxOrdered)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.01)
toolbox.register("select", tools.selTournament, tournsize=10)
```

tools.cxOrdered

Executes an ordered crossover (OX) on the input individuals. The two individuals are modified in place. This crossover expects sequence individuals of indices, the result for any other type of individuals is unpredictable.

Parameters:
ind1 – The first individual participating in the crossover.
ind2 – The second individual participating in the crossover.
Returns:
A tuple of two individuals.

Moreover, this crossover generates holes in the input individuals. A hole is created when an attribute of an individual is between the two crossover points of the other individual. Then it rotates the element so that all holes are between the crossover points and fills them with the removed elements in order. For more details see [Goldberg1989].

tools.mutShuffleIndexes

Shuffle the attributes of the input individual and return the mutant. The individual is expected to be a sequence. The indpb argument is the probability of each attribute to be moved. Usually this mutation is applied on vector of indices.

Parameters:
individual – Individual to be mutated.
indpb – Independent probability for each attribute to be exchanged to another position.
Returns:
A tuple of one individual.

tools.selTournament

Select the best individual among tournsize randomly chosen individuals, k times. The list returned contains references to the input individuals.

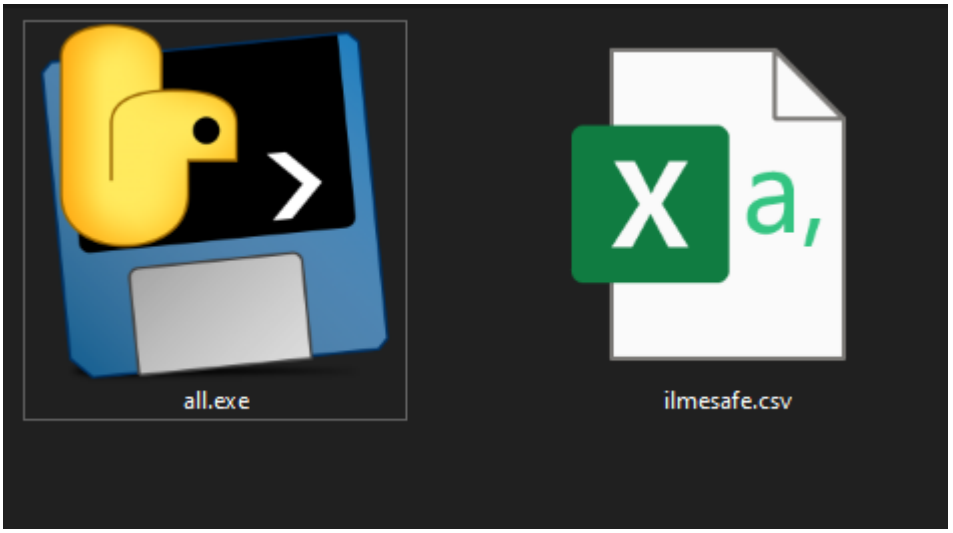
Parameters:
individuals – A list of individuals to select from.
k – The number of individuals to select.
tournsize – The number of individuals participating in each tournament.
fit_attr – The attribute of individuals to use as selection criterion
Returns:
A list of selected individuals.

How to

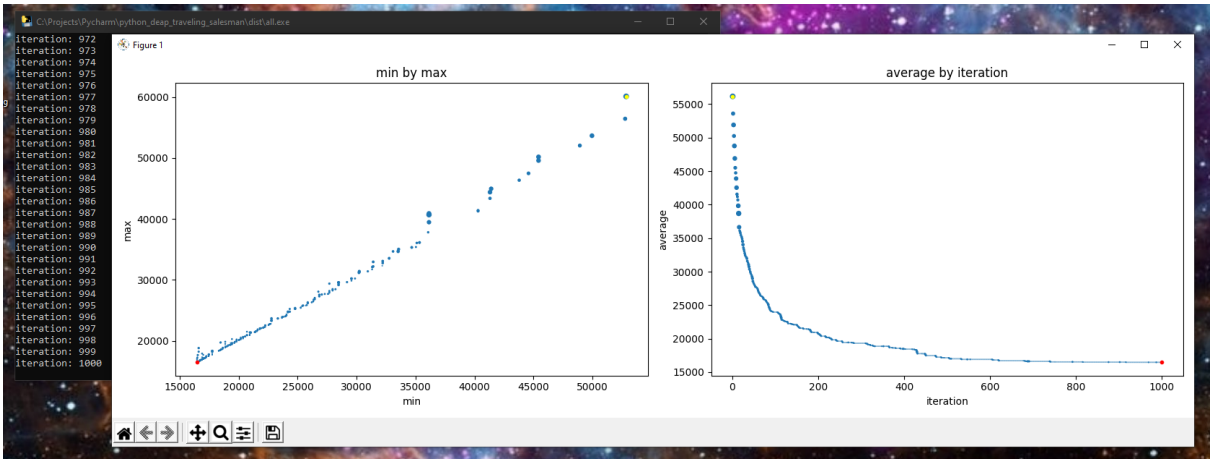
Compile

C:\Projects\Pycharm\python_deap_traveling_salesman>pyinstaller --onefile all.py

Execute



Main Script Runtime Pictures



```
En kısa yolun uzunluğu
{'i': 1000, 'mu': 16469.0, 'std': 0.0, 'max': 16469.0, 'min': 16469.0}
Denizli'den başlayacak şekilde dolaşılacak illerin sıralaması
0: DENİZLİ-20
1: AFYONKARAHİSAR-3
2: ÇORUM-19
3: AMASYA-5
4: TOKAT-60
5: ERZİNCAN-24
6: BİNGÖL-12
7: BATMAN-72
8: SİİRT-56
9: MARDİN-47
10: DİYARBAKIR-21
11: YOZGAT-66
12: DÜZCE-81
13: SAKARYA (ADAPAZARI)-54
14: KOCAELİ (İZMİT)-41
15: KIRKLARELİ-39
```

Important Parts of Code

Imports

```
import csv
import random
from pathlib import Path
import numpy as np
from deap import base, creator, tools
import matplotlib.pyplot as plt
```

Globals

```
INDIVIDUAL_SIZE = NUMBER_OF_CITIES = 81
POPULATION_SIZE = 200
N_ITERATIONS = 25
N_MATINGS = 50
```

Fitnesses

```
@staticmethod
def pull_stats(population, iteration=1):
    fitnesses = [individual.fitness.values[0] for individual in population]
    return {
        'i': iteration,
        'mu': np.mean(fitnesses),
        'std': np.std(fitnesses),
        'max': np.max(fitnesses),
        'min': np.min(fitnesses)
    }
```

Main stats variable creation

```
@property
def Run(self):
    population = self.toolbox.population(n=self.population_size)
```

```
    self.set_fitness(population)

    stats = []
    for iteration in list(range(1, self.iterations + 1)):
        current_population = list(map(self.toolbox.clone, population))
        offspring = list(self.get_offspring(current_population))
        for child in offspring:
            current_population.append(child)

        ## reset fitness,
        self.set_fitness(current_population)

        population[:] = self.toolbox.select(current_population,
len(population))
        stats.append(Runner.pull_stats(population, iteration))
        print('iteration: ' + str(len(stats)))
        # if len(stats) == 100:
        #     break
        # else:
        #     print('iteration: ' + str(len(stats)))
    return stats, population
```

Csv read

```
## City names,
csvDir = str(Path(__file__).parent) + '\\ilmesafe.csv'
with open(csvDir,encoding='utf8') as csvFile:
    data = list(csv.reader(csvFile))
npa = np.asarray(data)  # To Matrix
npa = np.delete(npa,0,1)    # First column
npa = np.delete(npa,0,1)    # Second column
cities = npa[:1].tolist()
npa = np.delete(npa,0,0)    # First row
for i in range(len(npa)):   # Arrange empty values in distance Matrix
    for j in range(len(npa[i])):
        if npa[i,j] == '':
            npa[i, j] = '0'
distances = npa.astype('float64')
```

Individual distance summation

```
def EVALUATE(individual):
    summation = 0
    start = individual[0]
    for i in range(1, len(individual)):
        end = individual[i]
        summation += distances[start][end]
        start = end
    return summation
```

Dolaşma

```
print('En kısa yolun uzunluğu')
print(stats[N_ITERATIONS-1])
print('Denizli den başlayacak şekilde dolaşılacak illerin sıralaması')
arrBefore19 = []
arrAfter19 = []
arrSwitch = False
for plate in population[POPULATION_SIZE-1]:
   if plate == 19:
       arrSwitch = True
   if arrSwitch:
       arrAfter19.append(plate)
   else:
       arrBefore19.append(plate)
concatenatedPlates = arrAfter19 + arrBefore19
for i in range(len(concatenatedPlates)):
   concatenatedPlates[i] = concatenatedPlates[i]+1
cities_to_list = []
for i in range(len(concatenatedPlates)):
   cp_index = concatenatedPlates[i]
   ctl_index = cities[cp_index-1] + '-' + str(cp_index)
   cities_to_list.append(ctl_index)
for i in range(len(cities_to_list)):
   print(str(i) + ': ' + cities_to_list[i])
```

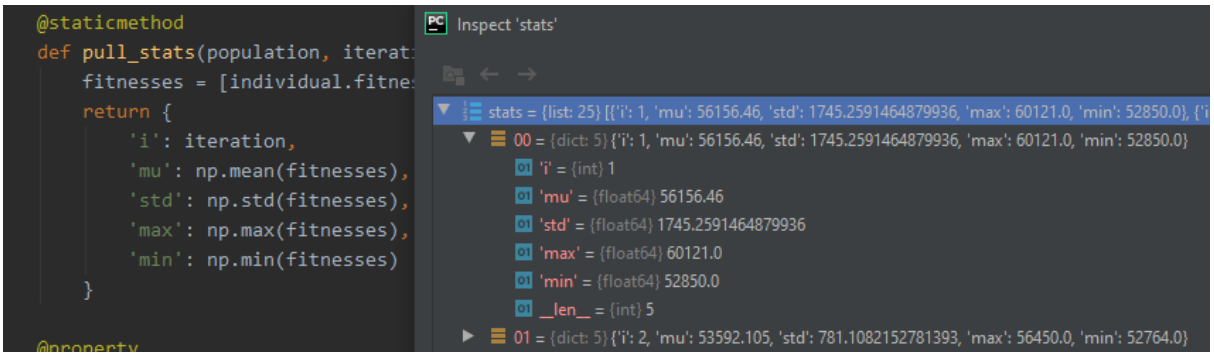
Plot

```
plt.figure(figsize=(15,5))
plt.subplot(1,2,1)

_ = plt.scatter([ s['min'] for s in stats ], [ s['max'] for s in stats ],
marker='.', s=[ (s['std'] + 1) / 20 for s in stats ])

_ = plt.title('min by max')
_ = plt.xlabel('min')
_ = plt.ylabel('max')

_ = plt.plot(stats[0]['min'], stats[0]['max'], marker='.', color='yellow')
_ = plt.plot(stats[-1]['min'], stats[-1]['max'], marker='.', color='red')

plt.subplot(1,2,2)

_ = plt.scatter([ s['i'] for s in stats ], [ s['mu'] for s in stats ],
marker='.', s=[ (s['std'] + 1) / 20 for s in stats ])

_ = plt.title('average by iteration')
_ = plt.xlabel('iteration')
_ = plt.ylabel('average')
```

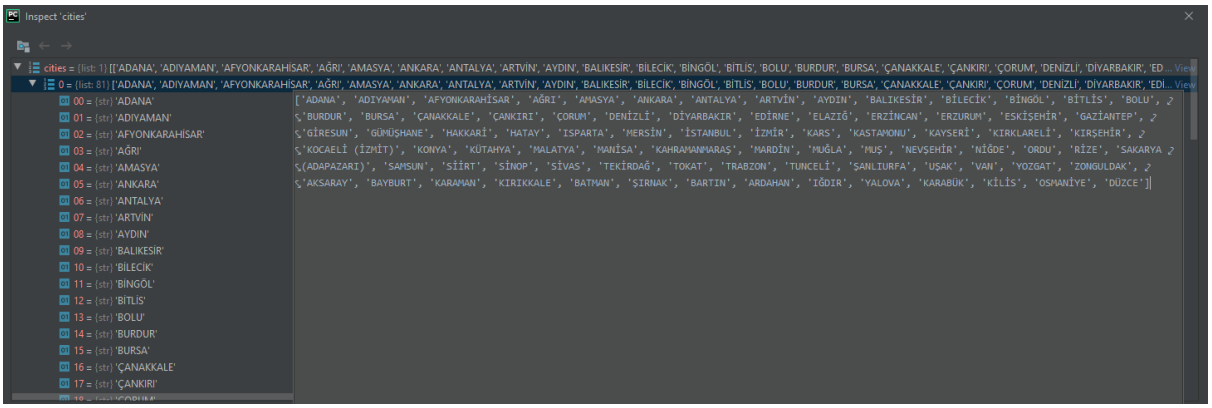
```
_ = plt.plot(stats[0]['i'], stats[0]['mu'], marker='.', color='yellow')
_ = plt.plot(stats[-1]['i'], stats[-1]['mu'], marker='.', color='red')

plt.tight_layout()
plt.show()
```

Inspect Variables

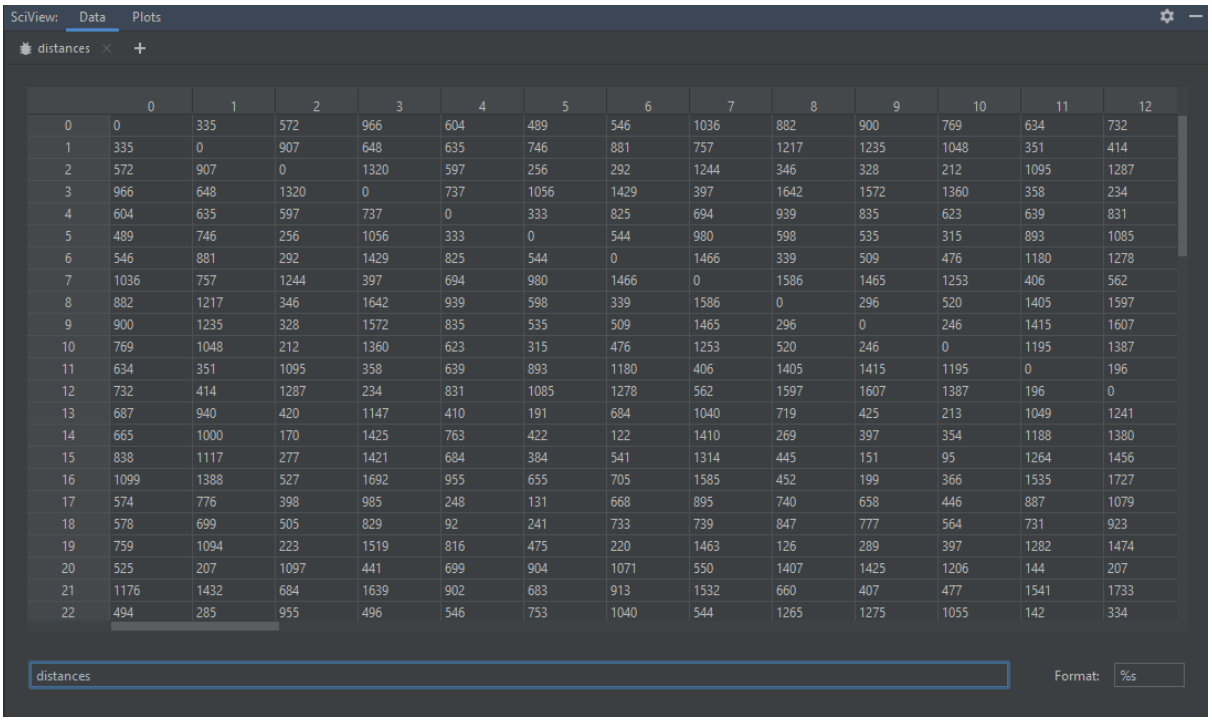
Stats (Variable that Plotted)



Cities



Distances

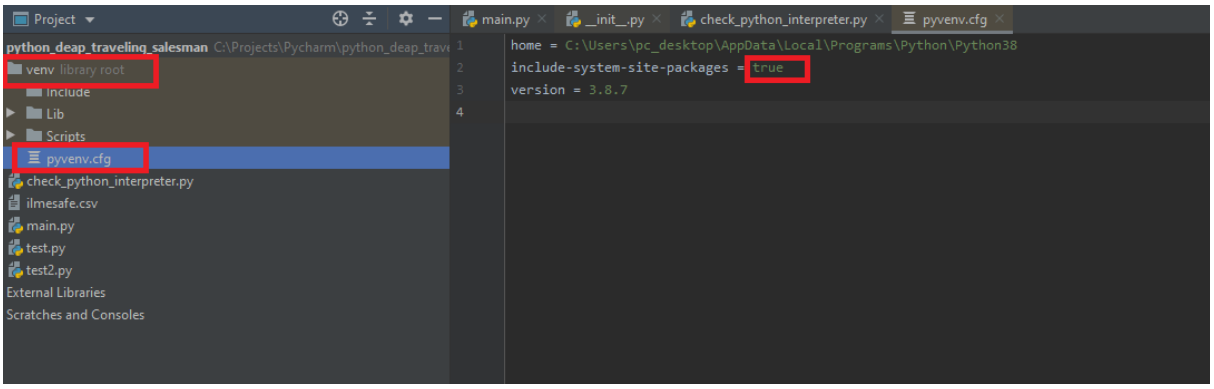


	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	335	572	966	604	489	546	1036	882	900	769	634	732
1	335	0	907	648	635	746	881	757	1217	1235	1048	351	414
2	572	907	0	1320	597	256	292	1244	346	328	212	1095	1287
3	966	648	1320	0	737	1056	1429	397	1642	1572	1360	358	234
4	604	635	597	737	0	333	825	694	939	835	623	639	831
5	489	746	256	1056	333	0	544	980	598	535	315	893	1085
6	546	881	292	1429	825	544	0	1466	339	509	476	1180	1278
7	1036	757	1244	397	694	980	1466	0	1586	1465	1253	406	562
8	882	1217	346	1642	939	598	339	1586	0	296	520	1405	1597
9	900	1235	328	1572	835	535	509	1465	296	0	246	1415	1607
10	769	1048	212	1360	623	315	476	1253	520	246	0	1195	1387
11	634	351	1095	358	639	893	1180	406	1405	1415	1195	0	196
12	732	414	1287	234	831	1085	1278	562	1597	1607	1387	196	0
13	687	940	420	1147	410	191	684	1040	719	425	213	1049	1241
14	665	1000	170	1425	763	422	122	1410	269	397	354	1188	1380
15	838	1117	277	1421	684	384	541	1314	445	151	95	1264	1456
16	1099	1388	527	1692	955	655	705	1585	452	199	366	1535	1727
17	574	776	398	985	248	131	668	895	740	658	446	887	1079
18	578	699	505	829	92	241	733	739	847	777	564	731	923
19	759	1094	223	1519	816	475	220	1463	126	289	397	1282	1474
20	525	207	1097	441	699	904	1071	550	1407	1425	1206	144	207
21	1176	1432	684	1639	902	683	913	1532	660	407	477	1541	1733
22	494	285	955	496	546	753	1040	544	1265	1275	1055	142	334

Must Know

Python version = 3.8.7

Fix site-packages



(2/2) Magic Square via Python DEAP

• Uygulamanın en az 1 adet ekran görüntüsü

&

• n = 7, 9, 10 değerleri için çözümler

7x7

```
-- Generation 299 --
Best  [42, 26, 23, 28, 12, 10, 34, 45, 8, 2, 11,
Fitnesses:
    Min 27.0
    Max 567.0
    Avg 131.5
    Std 126.0
-- Generation 300 --
Best  [42, 26, 23, 28, 12, 10, 34, 45, 8, 2, 11,
Fitnesses:
    Min 27.0
    Max 455.0
    Avg 145.0
    Std 128.9
--Draw Best--7x7
[42, 26, 23, 28, 12, 10, 34]
[45, 8, 2, 11, 24, 48, 37]
[1, 17, 35, 36, 31, 39, 18]
[14, 47, 46, 4, 30, 13, 21]
[27, 49, 33, 29, 19, 15, 3]
[7, 9, 32, 25, 16, 44, 40]
[38, 20, 5, 41, 43, 6, 22]
```

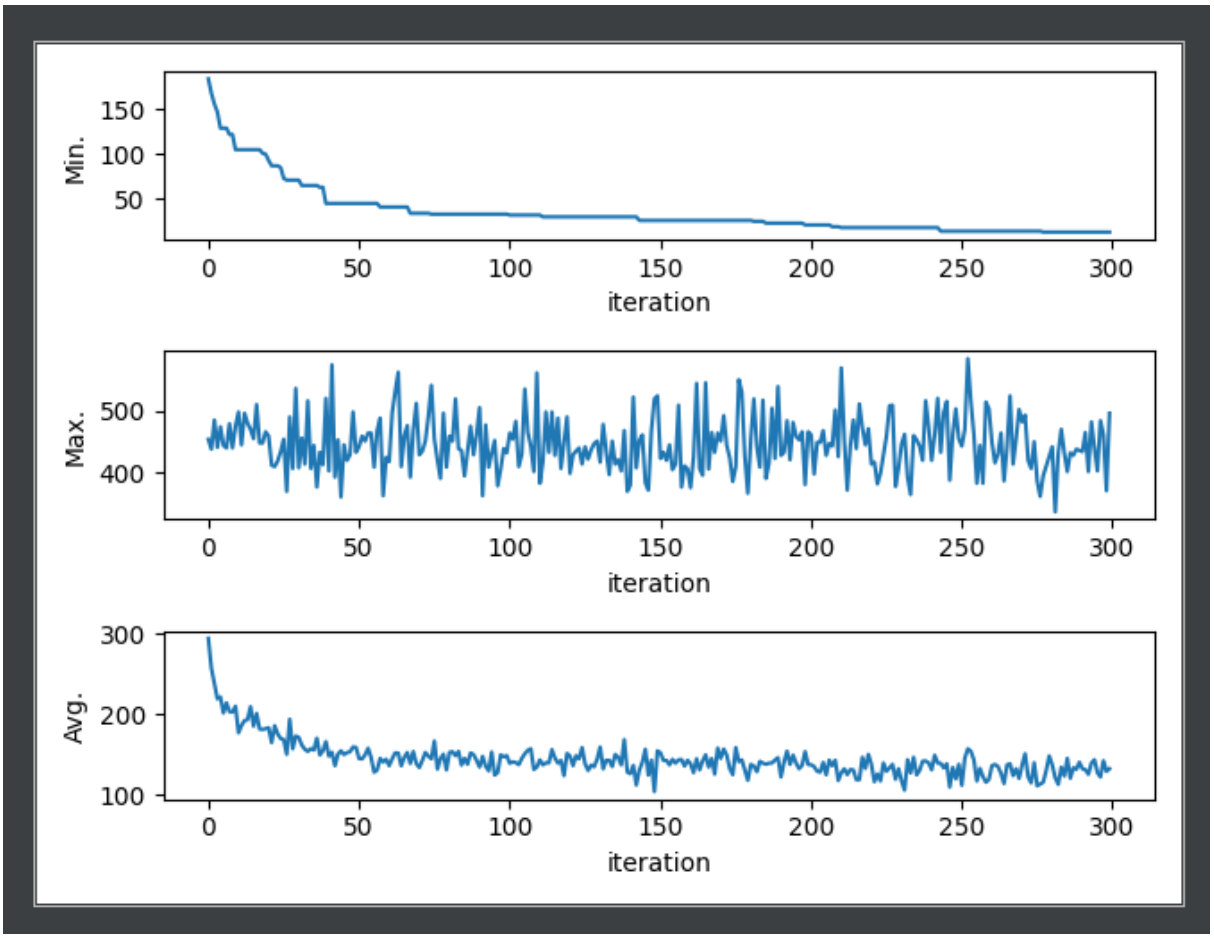
9x9

```
-- Generation 299 --
Best  [55, 6, 62, 24, 61, 47, 1, 57, 79, 11, 36, 69, 33, 29, 50,
Fitnesses:
    Min 149.0
    Max 1278.
    Avg 352.2
    Std 252.2
-- Generation 300 --
Best  [55, 6, 62, 24, 61, 47, 1, 57, 79, 11, 36, 69, 33, 29, 50,
Fitnesses:
    Min 149.0
    Max 1056.
    Avg 396.4
    Std 261.6
--Draw Best--9x9
[55, 6, 62, 24, 61, 47, 1, 57, 79]
[11, 36, 69, 33, 29, 50, 65, 32, 45]
[80, 8, 31, 53, 16, 44, 25, 64, 52]
[42, 59, 46, 43, 14, 56, 10, 71, 21]
[12, 48, 77, 35, 60, 70, 7, 41, 5]
[38, 74, 26, 34, 78, 17, 54, 39, 15]
[20, 68, 19, 23, 27, 3, 72, 58, 73]
[37, 40, 9, 63, 66, 75, 67, 2, 4]
[81, 30, 28, 49, 18, 22, 76, 13, 51]
```

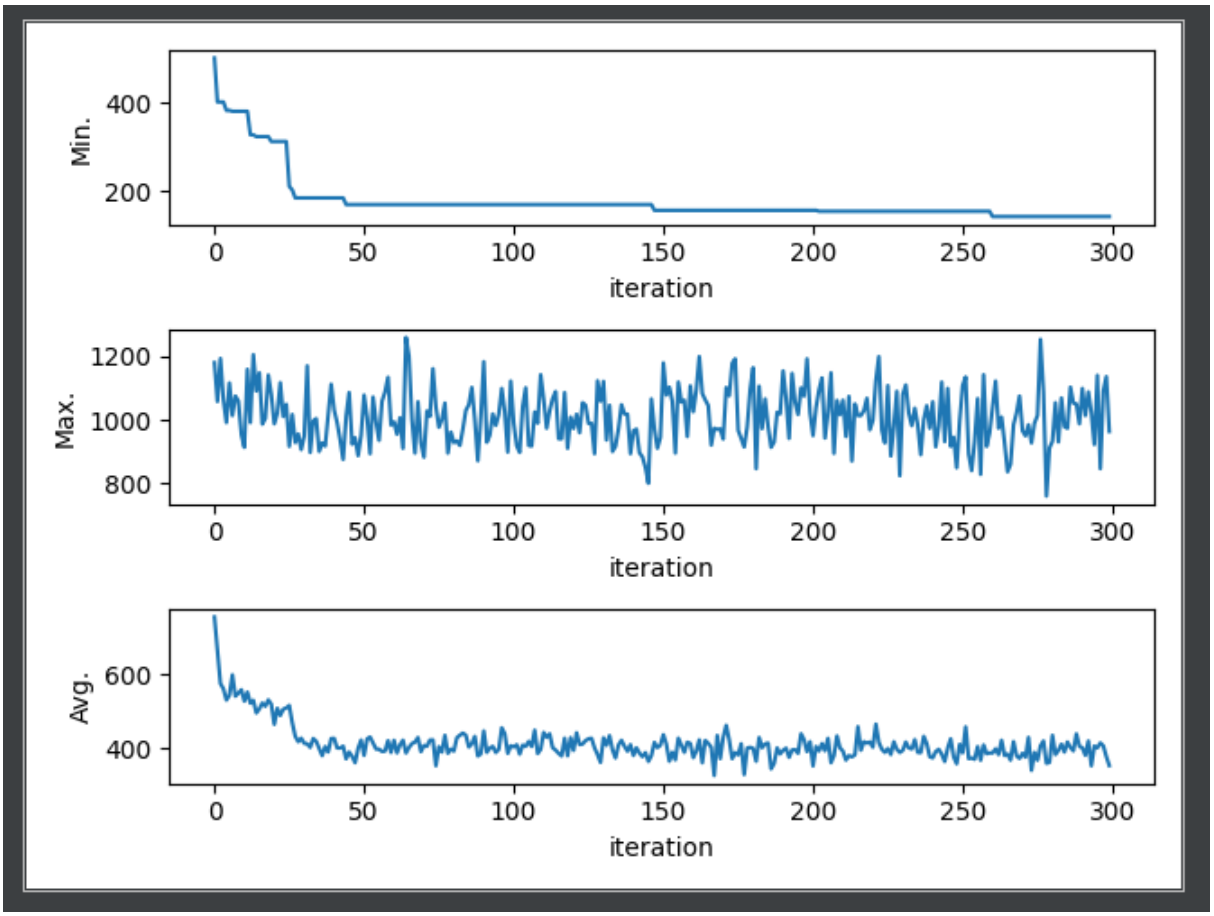
10x10

```
-- Generation 299 --
Best  [65, 26, 58, 3, 35, 92, 60, 86, 48, 40, 100, 31,
Fitnesses:
    Min 229.0
    Max 1473.
    Avg 627.7
    Std 363.8
-- Generation 300 --
Best  [65, 26, 58, 3, 35, 92, 60, 86, 48, 40, 100, 31,
Fitnesses:
    Min 229.0
    Max 1644.
    Avg 598.1
    Std 366.9
--Draw Best--10x10
[65, 26, 58, 3, 35, 92, 60, 86, 48, 40]
[100, 31, 51, 82, 46, 7, 33, 11, 37, 98]
[53, 14, 36, 38, 66, 84, 44, 24, 70, 74]
[9, 64, 18, 75, 76, 41, 2, 88, 43, 62]
[15, 68, 69, 4, 61, 90, 32, 34, 27, 94]
[42, 39, 79, 10, 21, 85, 80, 29, 95, 20]
[96, 16, 49, 56, 47, 52, 19, 5, 78, 91]
[28, 67, 63, 73, 54, 25, 71, 97, 55, 12]
[23, 99, 59, 93, 30, 17, 83, 77, 45, 1]
[72, 81, 22, 50, 89, 8, 87, 57, 6, 13]
```

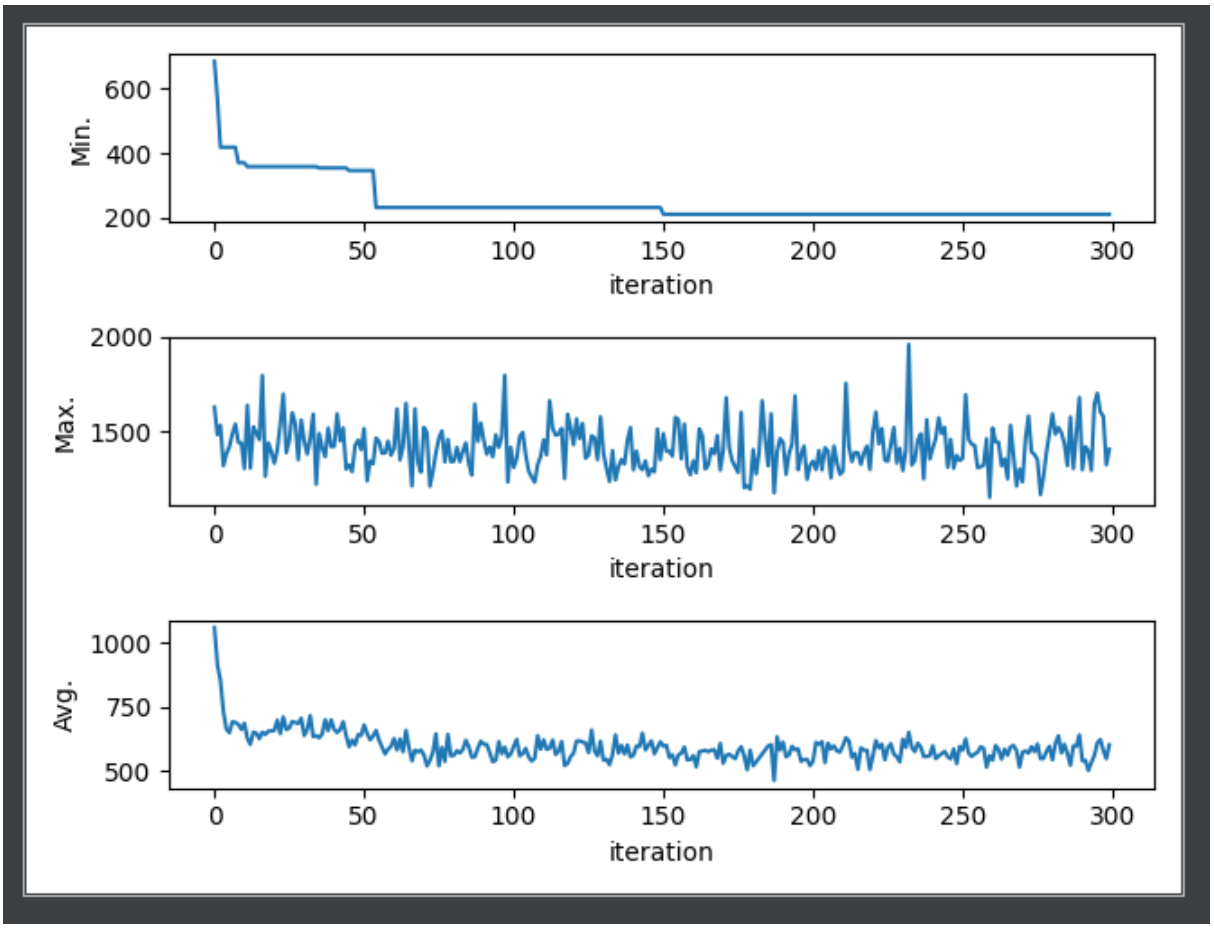
• n = 7, 9, 10 değerleri için eğitim grafikleri(minimum, maksimum ve ortalama fitness değerleri)
7x7



9x9


Min.
400
200
0
50
100
150
200
250
300
iteration
Max.
1200
1000
800
0
50
100
150
200
250
300
iteration
Avg.
600
400
0
50
100
150
200
250
300
iteration

10x10



• Seçilen temsil biçiminin belirtilmes

```
n_sons = int(pop_size*cros_over);
parents = toolbox.select(pop, n_sons)
offspring = list(map(toolbox.clone, parents))
pop[:(pop_size-n_sons)] = sorted_pop[:(pop_size-n_sons)]
pop[(pop_size-n_sons):] = offspring
```

• Temsil biçimi, çeşitlilik operatörleri(mutasyon ve çaprazlama), ebeveyn ve hayatta kalanların seçme algoritmalarının belirtilmesi

Mutasyon

```
# Clone the selected individuals
offspring = list(map(toolbox.clone, parents))
for child1, child2 in zip(offspring[::2], offspring[1::2]):
   toolbox.mate(child1, child2)

for mutant in offspring:
   if random.random() < mutation_rate:
       toolbox.mutate(mutant)
fitnesses = list(map(toolbox.evaluate, offspring))

for ind, fit in zip(offspring, fitnesses):
```

```
    ind.fitness.values = fit
    ind.invfitness.values = (100-fit[0],)
sorted_pop = sorted(pop, key=lambda ind: ind.fitness.values[0])
pop[:(pop_size-n_sons)] = sorted_pop[:(pop_size-n_sons)]
pop[(pop_size-n_sons):] = offspring
sorted_pop = sorted(pop, key=lambda ind: ind.fitness.values[0])
```

Algoritma

```
BOARD_SIZE = 10

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin,
invfitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("indices", random.sample, range(1,
BOARD_SIZE*BOARD_SIZE+1), BOARD_SIZE*BOARD_SIZE)
toolbox.register("individual", tools.initIterate, creator.Individual,
                 toolbox.indices)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("evaluate", evalMagicSquare)
toolbox.register("mate", cxCycle)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.10)
toolbox.register("select", tools.selRoulette, fit_attr='invfitness')
# toolbox.register("select", tools.selTournament, tournsize=3)
#toolbox.register("select", tools.selBest)

pop_size = 100;
cros_over = 0.8;
n_sons = int(pop_size*cros_over);
n_ger = 300;
# n_ger = 50;
mutation_rate = 0.7;
```

- Amaç(değerlendirme) fonksiyonunun yapısının anlatılması
  1. Fitness; konsola yazılmaktadır, değerler min, max, avg, std.
  2. Draw; konsola son jenerasyon bireyi matrix halinde çizmektedir.
  3. Plot; tüm jenerasyon min, max, avg değerlerinin grafiğini çizmektedir.

Fitness

```
av_min = sorted_pop[0].fitness.values[0]
arrGraphic[0].append(av_min)
av_max = sorted_pop[-1].fitness.values[0]
arrGraphic[1].append(av_max)
fits = [ind.fitness.values[0] for ind in pop]
```

```
mean = sum(fits) / pop_size
arrGraphic[2].append(mean)
sum2 = sum(x*x for x in fits)
std = abs(sum2 / pop_size - mean**2)**0.5

print("Best ", sorted_pop[0])
print("Fitnesses:")
print("\tMin %0.05s" % av_min)
print("\tMax %0.05s" % av_max)
print("\tAvg %0.05s" % mean)
print("\tStd %0.05s" % std)
```

Draw

```
print("--Draw Best--" + str(BOARD_SIZE) + "x" + str(BOARD_SIZE))
best = []
insertList = []
# noinspection PyUnboundLocalVariable
for i in range(len(list(sorted_pop[0]))):
    insertList.append(list(sorted_pop[0])[i])
    if ((i+1) % BOARD_SIZE == 0) & (i != 0):
        best.append(insertList)
        insertList = []
for row in best:
    for i in range(len(row)):
        if i == 0:
            print(end='[')
        if ((i + 1) % BOARD_SIZE == 0) & (i != 0):
            print(row[i], end=']\n')
        else:
            print(row[i], end=", ")
```

Plot

```
#Graphic # arrGraphic
# Some example data to display
arrX = [[], [], []]
arrY = [[], [], []]
fig, axs = plt.subplots(3)
# fig.suptitle('İbrahim Mert Küni - 16253801')
for i in range(len(arrGraphic[0])):
    arrX[0].append(i)
    arrX[1].append(i)
    arrX[2].append(i)
    arrY[0].append(arrGraphic[0][i])
    arrY[1].append(arrGraphic[1][i])
    arrY[2].append(arrGraphic[2][i])
axs[0].plot(arrX[0],arrY[0])
axs[0].set_xlabel('iteration')
```

```
axs[0].set_ylabel('Min.')

axs[1].plot(arrX[1],arrY[1])
axs[1].set_xlabel('iteration')
axs[1].set_ylabel('Max.')

axs[2].plot(arrX[2],arrY[2])
axs[2].set_xlabel('iteration')
axs[2].set_ylabel('Avg.')

plt.tight_layout()
plt.show()
```

Code Section ***
(1/2) *** Travelling Salesman Problem via Python DEAP

```
########################################## 1
import csv
import random
from pathlib import Path
import numpy as np
from deap import base, creator, tools
import matplotlib.pyplot as plt
########################################## Globals
INDIVIDUAL_SIZE = NUMBER_OF_CITIES = 81
POPULATION_SIZE = 200
N_ITERATIONS = 1000
N_MATINGS = 50
########################################## 2
# noinspection PyShadowingNames,PyAttributeOutsideInit
class Runner:

    def __init__(self, toolbox):
        self.toolbox = toolbox
        self.set_parameters(10, 5, 2)

    def set_parameters(self, population_size, iterations, n_matings):
        self.iterations = iterations
        self.population_size = population_size
        self.n_matings = n_matings
```

```
    def set_fitness(self, population):
        fitnesses = [
            (individual, self.toolbox.evaluate(individual))
            for individual in population
        ]

        for individual, fitness in fitnesses:
            individual.fitness.values = (fitness,)

    def get_offspring(self, population):
        n = len(population)
        for _ in range(self.n_matings):
            i1, i2 = np.random.choice(range(n), size=2, replace=False)

            offspring1, offspring2 = \
                self.toolbox.mate(population[i1], population[i2])

            yield self.toolbox.mutate(offspring1)[0]
            yield self.toolbox.mutate(offspring2)[0]

    @staticmethod
    def pull_stats(population, iteration=1):
        fitnesses = [individual.fitness.values[0] for individual in
population]
        return {
            'i': iteration,
            'mu': np.mean(fitnesses),
            'std': np.std(fitnesses),
            'max': np.max(fitnesses),
            'min': np.min(fitnesses)
        }

    @property
    def Run(self):
        population = self.toolbox.population(n=self.population_size)
        self.set_fitness(population)

        stats = []
        for iteration in list(range(1, self.iterations + 1)):
            current_population = list(map(self.toolbox.clone,
population))
            offspring = list(self.get_offspring(current_population))
            for child in offspring:
                current_population.append(child)

            ## reset fitness,
```

```
            self.set_fitness(current_population)

            population[:] = self.toolbox.select(current_population,
len(population))
            stats.append(Runner.pull_stats(population, iteration))
            print('iteration: ' + str(len(stats)))
            # if len(stats) == 100:
            #     break
            # else:
            #     print('iteration: ' + str(len(stats)))
        return stats, population
######################################### 3
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
# noinspection PyUnresolvedReferences
creator.create("Individual", list, fitness=creator.FitnessMin)
######################################### 4
random.seed(11)
np.random.seed(121)

## City names,
csvDir = str(Path(__file__).parent) + '\\ilmesafe.csv'
with open(csvDir,encoding='utf8') as csvFile:
    data = list(csv.reader(csvFile))
npa = np.asarray(data)  # To Matrix
npa = np.delete(npa,0,1)    # First column
npa = np.delete(npa,0,1)    # Second column
cities = npa[:1].tolist()
cities = cities[0]
npa = np.delete(npa,0,0)    # First row
for i in range(len(npa)):   # Arrange empty values in distance Matrix
    for j in range(len(npa[i])):
        if npa[i,j] == '':
            npa[i, j] = '0'
distances = npa.astype('float64')
######################################### 5
toolbox = base.Toolbox()

## permutation setup for individual,
toolbox.register("indices", random.sample, range(INDIVIDUAL_SIZE),
INDIVIDUAL_SIZE)
# noinspection PyUnresolvedReferences
toolbox.register("individual", tools.initIterate, creator.Individual,
toolbox.indices)

## population setup,
# noinspection PyUnresolvedReferences
```

```
toolbox.register("population", tools.initRepeat, list,
toolbox.individual)
######################################### 6
# noinspection PyShadowingNames
def EVALUATE(individual):
    summation = 0
    start = individual[0]
    for i in range(1, len(individual)):
        end = individual[i]
        summation += distances[start][end]
        start = end
    return summation

toolbox.register("evaluate", EVALUATE)
######################################### 7
toolbox.register("mate", tools.cxOrdered)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.01)
toolbox.register("select", tools.selTournament, tournsize=10)
######################################### 8
a = Runner(toolbox)
a.set_parameters(POPULATION_SIZE, N_ITERATIONS, N_MATINGS)

stats, population = a.Run
######################################### Dolaşma
print('En kısa yolun uzunluğu')
print(stats[N_ITERATIONS-1])
print('Denizli'den başlayacak şekilde dolaşılacak illerin sıralaması')
arrBefore19 = []
arrAfter19 = []
arrSwitch = False
for plate in population[POPULATION_SIZE-1]:
    if plate == 19:
        arrSwitch = True
    if arrSwitch:
        arrAfter19.append(plate)
    else:
        arrBefore19.append(plate)
concatenatedPlates = arrAfter19 + arrBefore19
for i in range(len(concatenatedPlates)):
    concatenatedPlates[i] = concatenatedPlates[i]+1
cities_to_list = []
for i in range(len(concatenatedPlates)):
    cp_index = concatenatedPlates[i]
    ctl_index = cities[cp_index-1] + '-' + str(cp_index)
    cities_to_list.append(ctl_index)
for i in range(len(cities_to_list)):
```

```
    print(str(i) + ': ' + cities_to_list[i])
# print('daa')
######################################### Plot
plt.figure(figsize=(15,5))

plt.subplot(1,2,1)

_ = plt.scatter([ s['min'] for s in stats ], [ s['max'] for s in stats ],
marker='.', s=[ (s['std'] + 1) / 20 for s in stats ])

_ = plt.title('min by max')
_ = plt.xlabel('min')
_ = plt.ylabel('max')

_ = plt.plot(stats[0]['min'], stats[0]['max'], marker='.',
color='yellow')
_ = plt.plot(stats[-1]['min'], stats[-1]['max'], marker='.', color='red')


plt.subplot(1,2,2)

_ = plt.scatter([ s['i'] for s in stats ], [ s['mu'] for s in stats ],
marker='.', s=[ (s['std'] + 1) / 20 for s in stats ])

_ = plt.title('average by iteration')
_ = plt.xlabel('iteration')
_ = plt.ylabel('average')

_ = plt.plot(stats[0]['i'], stats[0]['mu'], marker='.', color='yellow')
_ = plt.plot(stats[-1]['i'], stats[-1]['mu'], marker='.', color='red')

plt.tight_layout()
plt.show()
```

(2/2) *** Magic Square via Python DEAP

```
import random
import math
from deap import base
from deap import creator
from deap import tools
import matplotlib.pyplot as plt
import numpy as np

def evalMagicSquare(individual):
```

```
    size = len(individual)
    n = int(math.sqrt(size))
    magic_n = (n*(n*n +1))/2

    fitness = 0
    diag_aux = 0
    diag2_aux = 0
    for i in range(n):
        row_aux = 0
        column_aux = 0
        for j in range(n):
            row_aux = row_aux + individual[i*n + j]
            column_aux = column_aux + individual[i + j*n]

            if i==j:
                diag_aux = diag_aux + individual[i*n +j]
            elif i == (n-1) -j:
                diag2_aux = diag2_aux + individual[i*n + j]

        fitness = fitness + abs(column_aux - magic_n) + abs(row_aux -
magic_n)

    fitness = fitness + abs(diag_aux - magic_n) + abs(diag2_aux -
magic_n)
    return fitness,
# noinspection PyTrailingSemicolon,PyRedundantParentheses
def cxCycle(ind1, ind2):
    length = len(ind1)
    index = random.randint(0, length-1)
    cycle = [0]*length

    while(not cycle[index]):
        cycle[index] = 1;
        for i in range(0, length):
            if ind1[index] == ind2[i]:
                index = i
                break

    for j in range(0,length):
      if (cycle[j] == 1):
        temp = ind1[j]
        ind1[j] = ind2[j];
        ind2[j] = temp;

    return ind1,ind2
# noinspection PyTrailingSemicolon,PyShadowingNames
```

```
def main():
    BOARD_SIZE = 10

    creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
    creator.create("FitnessMax", base.Fitness, weights=(1.0,))
    creator.create("Individual", list, fitness=creator.FitnessMin,
invfitness=creator.FitnessMax)

    toolbox = base.Toolbox()
    toolbox.register("indices", random.sample, range(1,
BOARD_SIZE*BOARD_SIZE+1), BOARD_SIZE*BOARD_SIZE)
    toolbox.register("individual", tools.initIterate, creator.Individual,
                     toolbox.indices)
    toolbox.register("population", tools.initRepeat, list,
toolbox.individual)

    toolbox.register("evaluate", evalMagicSquare)
    toolbox.register("mate", cxCycle)
    toolbox.register("mutate", tools.mutShuffleIndexes, indpb=0.10)
    toolbox.register("select", tools.selRoulette, fit_attr='invfitness')
    # toolbox.register("select", tools.selTournament, tournsize=3)
    #toolbox.register("select", tools.selBest)

    pop_size = 100;
    cros_over = 0.8;
    n_sons = int(pop_size*cros_over);
    n_ger = 300;
    # n_ger = 50;
    mutation_rate = 0.7;

    pop = toolbox.population(n=pop_size)

    # Evaluate the entire population
    fitnesses = list(map(toolbox.evaluate, pop))
    for ind, fit in zip(pop, fitnesses):
        ind.fitness.values = fit
        ind.invfitness.values = (100-fit[0],)

    # Variable keeping track of the number of generations
    g = 0
    arrGraphic = [[], [], []]
    while g < n_ger:
        # A new generation
        g = g + 1
        print("-- Generation %i --" % g)
```

```
    #Select parents
    parents = toolbox.select(pop, n_sons)
    # Clone the selected individuals
    offspring = list(map(toolbox.clone, parents))
    for child1, child2 in zip(offspring[::2], offspring[1::2]):
        toolbox.mate(child1, child2)

    for mutant in offspring:
        if random.random() < mutation_rate:
            toolbox.mutate(mutant)

    fitnesses = list(map(toolbox.evaluate, offspring))
    for ind, fit in zip(offspring, fitnesses):
        ind.fitness.values = fit
        ind.invfitness.values = (100-fit[0],)

    sorted_pop = sorted(pop, key=lambda ind: ind.fitness.values[0])
    pop[:(pop_size-n_sons)] = sorted_pop[:(pop_size-n_sons)]
    pop[(pop_size-n_sons):] = offspring

    sorted_pop = sorted(pop, key=lambda ind: ind.fitness.values[0])
    av_min = sorted_pop[0].fitness.values[0]
    arrGraphic[0].append(av_min)
    av_max = sorted_pop[-1].fitness.values[0]
    arrGraphic[1].append(av_max)
    fits = [ind.fitness.values[0] for ind in pop]

    mean = sum(fits) / pop_size
    arrGraphic[2].append(mean)
    sum2 = sum(x*x for x in fits)
    std = abs(sum2 / pop_size - mean**2)**0.5

    print("Best ", sorted_pop[0])
    print("Fitnesses:")
    print("\tMin %0.05s" % av_min)
    print("\tMax %0.05s" % av_max)
    print("\tAvg %0.05s" % mean)
    print("\tStd %0.05s" % std)

    if av_min==0:
        break
# Draw
print("--Draw Best--" + str(BOARD_SIZE) + "x" + str(BOARD_SIZE))
best = []
insertList = []
# noinspection PyUnboundLocalVariable
```

```
    for i in range(len(list(sorted_pop[0]))):
        insertList.append(list(sorted_pop[0])[i])
        if ((i+1) % BOARD_SIZE == 0) & (i != 0):
            best.append(insertList)
            insertList = []
    for row in best:
        for i in range(len(row)):
            if i == 0:
                print(end='[')
            if ((i + 1) % BOARD_SIZE == 0) & (i != 0):
                print(row[i], end=']\n')
            else:
                print(row[i], end=", ")

    #Graphic # arrGraphic
    # Some example data to display
    arrX = [[], [], []]
    arrY = [[], [], []]
    fig, axs = plt.subplots(3)
    # fig.suptitle('İbrahim Mert Küni - 16253801')
    for i in range(len(arrGraphic[0])):
        arrX[0].append(i)
        arrX[1].append(i)
        arrX[2].append(i)
        arrY[0].append(arrGraphic[0][i])
        arrY[1].append(arrGraphic[1][i])
        arrY[2].append(arrGraphic[2][i])
    axs[0].plot(arrX[0],arrY[0])
    axs[0].set_xlabel('iteration')
    axs[0].set_ylabel('Min.')

    axs[1].plot(arrX[1],arrY[1])
    axs[1].set_xlabel('iteration')
    axs[1].set_ylabel('Max.')

    axs[2].plot(arrX[2],arrY[2])
    axs[2].set_xlabel('iteration')
    axs[2].set_ylabel('Avg.')

    plt.tight_layout()
    plt.show()


    print()
if __name__ == '__main__':
    main()
```