

Test Problem

Task:

Arrange m dogs and n cats ($n > 1$) in a circle such that there are either no cats or at least two cats between each pair of consecutive dogs.

*Observation: The problem can be simplified if we consider that an invalid placement of dogs and cats is the case in which there is **only** one cat between two dogs.*

Individual representation:

We consider x an individual. The size of x is the sum of the number of cats n and the number of dogs $m = n + m$:

$$x = \{x_0, x_1, x_2, \dots, x_n, \dots, x_{n+m-1}\}$$

For each $x_i, i = \overline{0, n+m-1}$, we consider

$$x_i = \begin{cases} \text{cat}, & \text{if } 0 \leq x_i < n \\ \text{dog}, & n \leq x_i < m+n \end{cases}$$

Example:

for $n = 5$

$m = 6$

→ we have 5 cats and 6 dogs → the size of x will be 11

$x = [1, 2, 0, 7, 9, 10, 6, 5, 4, 3, 8]$

→ x can be translated to $x = [\text{cat}, \text{cat}, \text{cat}, \text{dog}, \text{dog}, \text{dog}, \text{dog}, \text{dog}, \text{dog}, \text{cat}, \text{cat}, \text{dog}]$.

→ mathematically, x represents a permutation of $m + n - 1$

Fitness function (lines 30 – 47):

Our goal is to have no invalid sequences. An invalid sequence is represented by the presence of only one cat between two dogs. Thus, we can consider the cost as $\text{cost} = \text{sum}(\text{invalid_seq})$ (lines 32 – 46). We want to find the best possible solution, in other words an x with no invalid sequences. In order to minimize the cost and maximize the fitness value we can consider the following equation:

$$\text{fitness}_{\max} = \frac{1}{1 + \text{cost}}$$

We consider $1 + \text{cost}$ instead of just cost to avoid division by zero in the case of $\text{cost} = 0$.

Restrictions / feasibility check:

It is not necessary that we check if an individual is feasible, since we use permutation operators and it is impossible to have invalid permutations (invalid individuals x).

Parameters

For the ease of working with our input data, we have the following structure:

```
@dataclass
class Parameters:
    pop_size: int = 40
    n: int = 5      # number of cats
    m: int = 6      # number of dogs
    pets: int = 0
    max_iterations: int = 500
    crossover_probability: float = 0.8
    mutation_probability: float = 0.1

parameters = Parameters()
parameters.pets = parameters.n + parameters.m
```

parameters.pets represents the size of individual x .

Initial population generation (lines 51 – 62):

Each individual x in the population is randomly generated, by generating a random permutation of **parameters.pets** size.

```
for i in range(parameters.pop_size):
    x = np.random.permutation(parameters.pets)
    pop += [x]
    init_pop_fitness += [fitness_func(x)]
```

After the generation, we simply add it to the initial population and we calculate its' fitness and store it in the array of fitnesses.

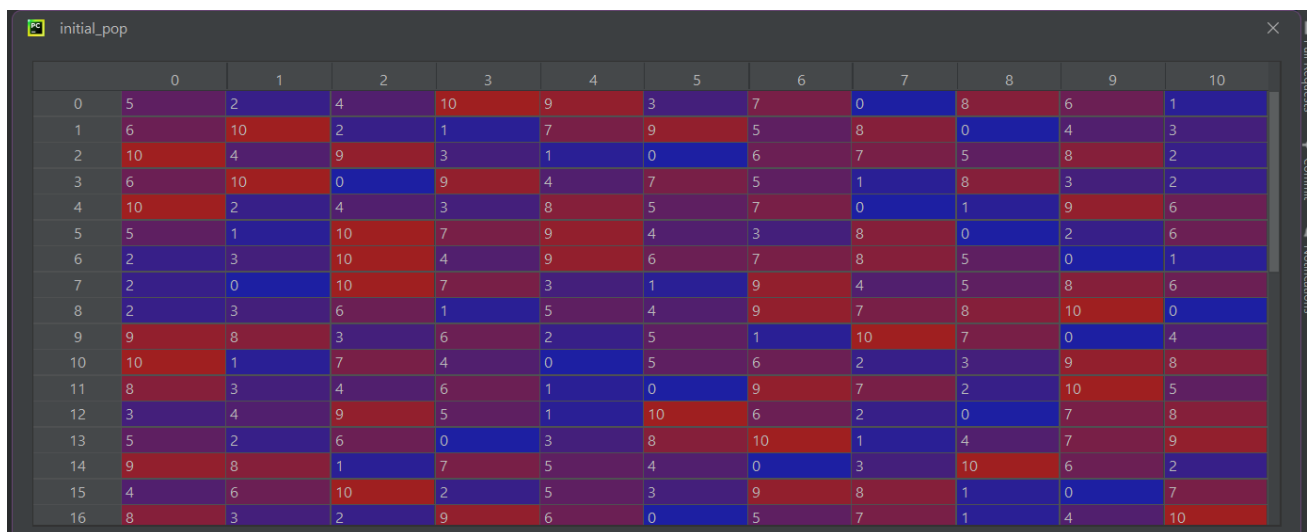
This mechanism will be repeated **parameters.pop_size** times where *parameters.pop_size* = *population size*.

Initial population example:

for $n = 5$

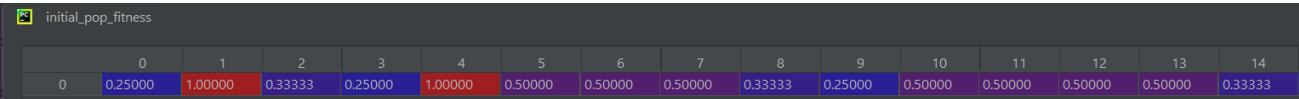
$m = 6$

population size = 40



	0	1	2	3	4	5	6	7	8	9	10
0	5	2	4	10	9	3	7	0	8	6	1
1	6	10	2	1	7	9	5	8	0	4	3
2	10	4	9	3	1	0	6	7	5	8	2
3	6	10	0	9	4	7	5	1	8	3	2
4	10	2	4	3	8	5	7	0	1	9	6
5	5	1	10	7	9	4	3	8	0	2	6
6	2	3	10	4	9	6	7	8	5	0	1
7	2	0	10	7	3	1	9	4	5	8	6
8	2	3	6	1	5	4	9	7	8	10	0
9	9	8	3	6	2	5	1	10	7	0	4
10	10	1	7	4	0	5	6	2	3	9	8
11	8	3	4	6	1	0	9	7	2	10	5
12	3	4	9	5	1	10	6	2	0	7	8
13	5	2	6	0	3	8	10	1	4	7	9
14	9	8	1	7	5	4	0	3	10	6	2
15	4	6	10	2	5	3	9	8	1	0	7
16	8	3	2	9	6	0	5	7	1	4	10

Fitness values:



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0.25000	1.00000	0.33333	0.25000	1.00000	0.50000	0.50000	0.50000	0.33333	0.25000	0.50000	0.50000	0.50000	0.50000	0.33333

→ *coincidentally we have found the best global solution (or one of them) in the initial population. This can be due to the fact that our individual size is not very big.*

Parent selection (lines 66 – 105):

Stochastic Universal Sampling selection was used in order to allow the individuals with higher fitness values to have a higher probability of selection (probabilities have been assigned through Fitness Proportional selection based on sigma scaling). We take the parents 2 by 2 and we perform crossover, keeping the order the same in the parents array. We select *pop_size* parents because we have to keep the population size constant.

Crossover (lines 110 – 177):

To crossover the parents' genes we use partially mapped crossover (PMX) because it is specific to permutation representations. We randomly select two crossover points, compute the set of alleles corresponding to the genes between the two points from the second parent so that they do not belong in the first offspring. Then we perform the PMX crossover. The probability of an individual to be recombined is 0.8 (80%). If the crossover is not performed, we perform asexual crossover, where the child takes its' parent genes directly.

Mutation (lines 181 – 207):

The chosen mutation method is swap mutation because it is specific to permutation representation and the mutation probability is 0.1 (10%). The swap mutation is applied to the entire chromosome by selecting two random genes and swapping them. If the probability of mutation is not met, then the offspring remains unchanged.

Next generation selection (lines 211 – 223):

To advance the generation, we use the elitism method in which we compare the best parent (the parent with the highest fitness value) to the best child (the child with the highest fitness value). If the best parent is better than the best child, we randomly replace a child with the best parent. Otherwise, the children population remains the same.

GA Termination Conditions:

The genetic algorithm will try to compute the best individual through a certain number of *max_iterations*. However, the GA will stop and retain the current best solution even if it is not the best global solution in certain conditions as follows:

- if the number of generations equals the number of *max_iterations*

```
iteration == parameters.max_iterations
```

- if the max fitness value of the population remains constant over *max_iterations* / 3 generations

```
nrm == int(parameters.max_iterations / 3)
```

Where *nrm* = number of generations with the same max fitness value

- if the best individual's population is equal to 1 which means the best global solution has been achieved

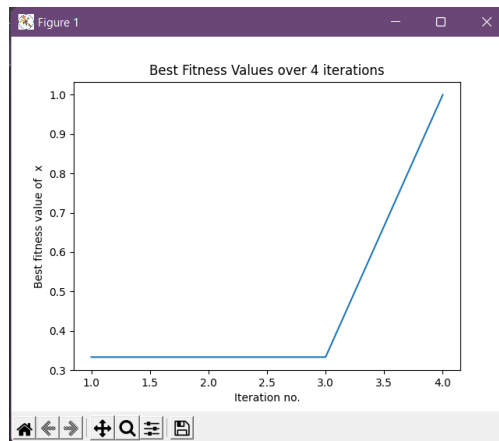
```
next_pop_max == 1
```

The GA Solution:

The solution of the genetic algorithm will be an individual from the last generation that has the best fitness value (not necessarily the best global solution) and is represented in the manner mentioned above. At the end of the GA, we will display the best individual from the last generation as well as its' fitness value.

Graph Interpretation:

The graph generated by the problem represents the best fitness value of each generation throughout all of the iterations performed.



Disclaimer:

All constant parameters of the problem have been added to a Parameters structure:

```
@dataclass
class Parameters:
    pop_size: int = 40
    n: int = 5      # number of cats
    m: int = 6      # number of dogs
    pets: int = 0
    max_iterations: int = 500
    crossover_probability: float = 0.8
    mutation_probability: float = 0.1
```

However the input data can be changed when running the problem:

```
size = (int)(input("Population size ( > 0 ): "))
parameters.pop_size = size
it = (int)(input("\nMaximum number of iterations:"))
parameters.max_iterations = it
cp = (float)(input("\nCrossover probability (between 0 and 1):"))
parameters.crossover_probability = cp
mp = (float)(input("\nMutation probability (between 0 and 1):"))
parameters.mutation_probability = mp
```