



Setup

```
pip install -r requirements.txt
```

 A default adjacency matrix is filled out in a text file called `adj_mat.txt` for each structure inside `matrices` folder. You can create another one with the same structure of the default one from your graph and put it in its related directory.

Usage Example

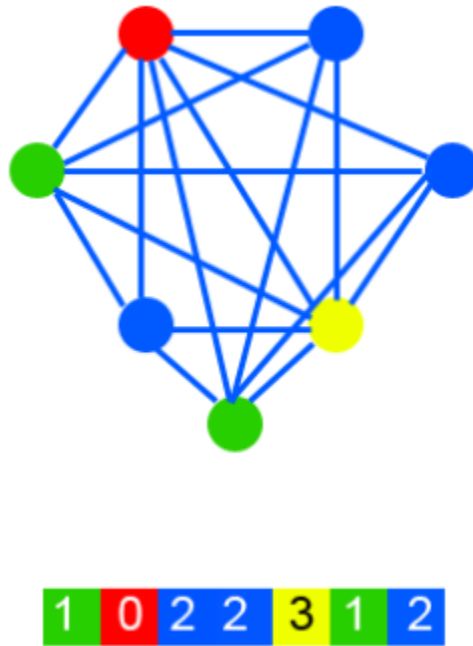
```
python colory.py --adj-mat utils/matrices/tree/adj_mat.txt --colors  
orange red blue green --chromosomes 50 --generations 20 --parents 30 -  
-selection-method tournament --crossover-method 3-point --mutation-  
method creep --alpha-rate 0.30 --mutation-rate 0.20 --crossover-rate  
0.80
```

 Run `python colory.py --help` for argument details.

Procedures

Encoding & Initial Population


Chromosome Representation of a Colored Graph



We represent the graph using adjacency matrices. The user gives the colors.

There is no need to use **GA** if the number of colors is more than or equal to the number of nodes. If so, an error will pop up, showing that we do not need to use **GA**.

The encoding that we used is discrete. We have an array filled with the numbers mapped to each color, and the length of the array is equal to the nodes' number.

 Note that the user have to use the full name of the colors as input. Refer to the setup section.

We use a uniform random function to create the initial population using the given adjacency matrices and colors. The user gives the total number of the population.

✦ Objective Functions

The obtained fitness function consists of two parts: The first part detects the `invalid_genes` and tries to minimize the number of them. In the second part, we have a variable called `m_prime`, which is the minimum number of colors used for coloring the graph. We need to use constant values for both parts of the objective function to show which part is more important. Alpha and beta are the constant variables. Assume that the sum of Alpha and beta is equal to one. The issue is to color the graph using the minimum valid colors. In order to find the proper multi-objective function, both `invalid_genes` and `m_prime` should have the same range. To fix this, we need to normalize the values as shown below:

```
invalid_normalized = invalid_genes / total_edges  
m_prime_normalized = (minimum_number_of_colors - 1) / colors  
total_fitness = (alpha*invalid_normalized) + (beta*m_prime_normalized)
```

⚠ Depending on the method that we have used during the process, the fitness function could be equal to the objective function or inverted.

✦ Optimization

We have a two-part goal for this problem. The first and the essential part is to find the valid coloring for the graph in which none of the connected nodes have the same color; the second part is to color the graph using the minimum number of colors. Since **GA** is a stochastic search algorithm based on natural competition principles between individuals, it is possible not to reach the intended minimum colors for each graph depending on the problem inputs. Nonetheless, the achieved result is that the minimum number of colors reached up to the assumed generation with the number of chromosomes and parents using each operation's selected methods. The code is entirely operative.

✈ Genetic Operators

Order of operations are as follows:

Selection

As selection methods, we have used the roulette wheel (**FPS**), rank, and tournament.

- In the roulette wheel (**FPS**), the fitness function is equal to the inversed objective function.
- In tournament and ranked based selection, the fitness function is equal to the objective function.
- In the tournament method, we used equal probability distribution for each gene in the selected chromosome.

Crossover

As crossover methods, we have used t-point and uniform.

Mutation

As mutation methods, we have use swap, creep, and inversion.

Replacement

As replacement methods, we have used steady-state, generational gap, and generational method using elitism depending on the proportion of the replaced population defined by the alpha rate, which should be initialized by the user.

⚠ The user should initialize crossover and mutation rates. As we have used the **GA**, the crossover rate should be much higher than the mutation rate.

✈ Stopping Criterion

Since the total fitness function never equals zero (The minimum number of colors is equal to one.), we considered the stopping criterion to reach the maximum number of generations.

✈ Coloring the Graph

To color the graph, we have defined two **NumPy** arrays and have stored the valid chromosomes and the valid chromosomes with the minimum number of colors in each generation called `total_generations_valid_chromosomes` and `total_generations_minimum_colors_valid_chromosomes`, respectively. Using these two arrays, we plot the colored graph based on the our coloring problem inputs' size.

Results

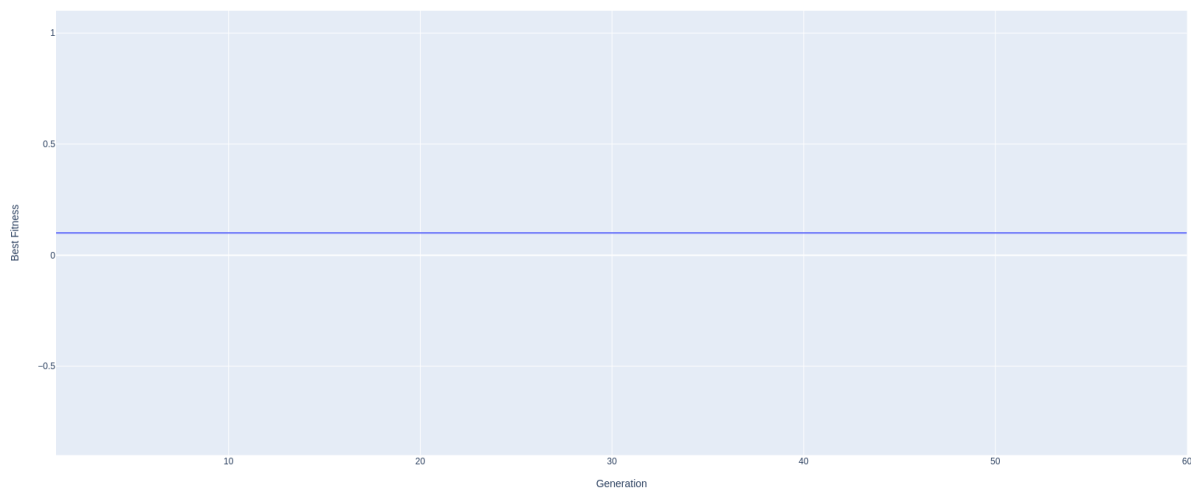
We ran the **GA** for different test-cases and store their results inside [utils/results](#) folder. Below are two of the test-cases that our algorithm reached state-of-the-art of solving coloring problem.

Test-case-1

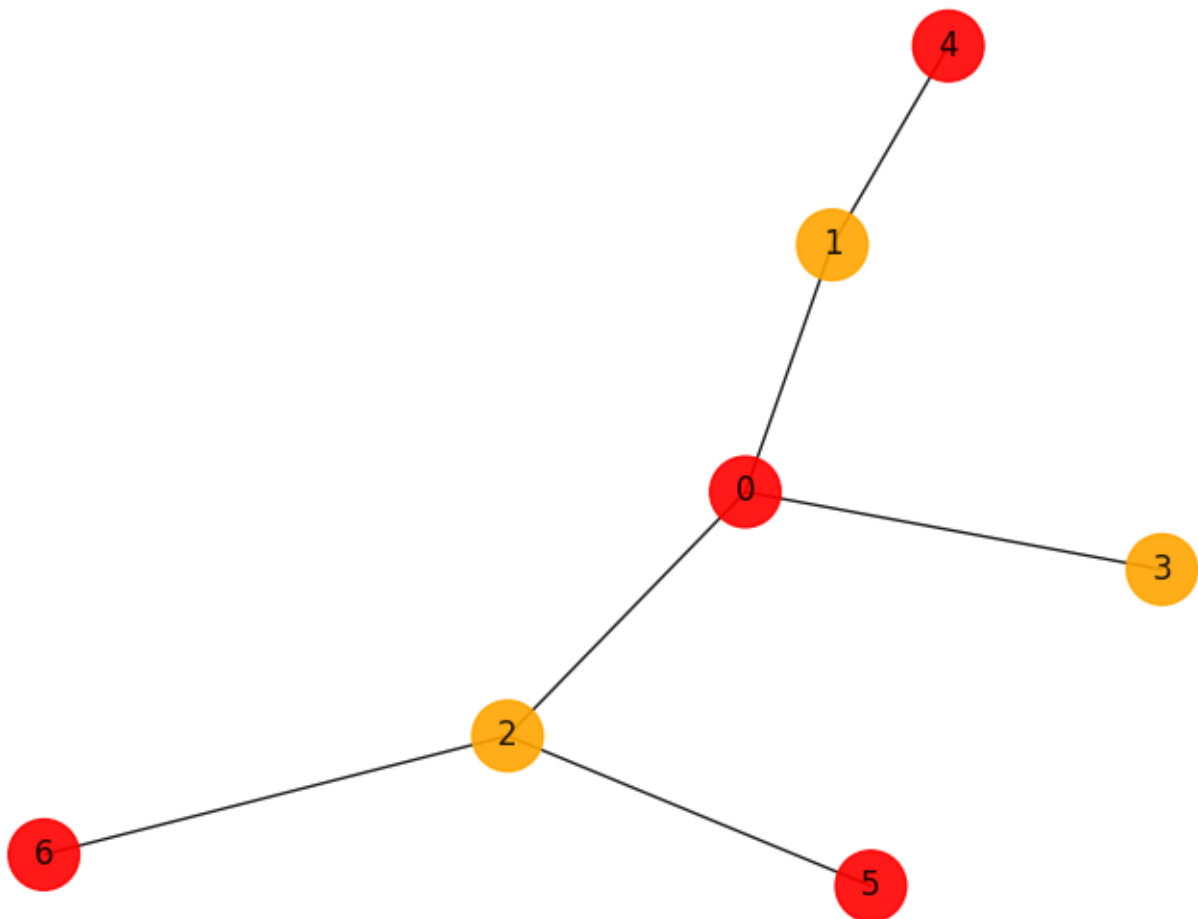
Statistical Logs

```
⌚ Saving Valid Chromosomes, Chromosomes with Minimum Colors and Best Fitness Scores
📁 Videos    ✖ It's not Possible to Color the Graph with Less than 2 Colors
📁 Trash      ✔ It's Possible to Color the Graph with 2 Colors using Random Selected Valid Chromosome
               ▶ Random Selected Valid Chromosome --- [1 0 0 0 1 1 1]
+ On         ▶ Average Fitness Scores --- 0.09999999999999996
               ▶ Median Fitness Scores --- 0.1
               ▶ Total Valid Chromosomes in Generation 1 --- (1, 7)
```

Fitness Generations



Colored Graph using Valid Chromosomes



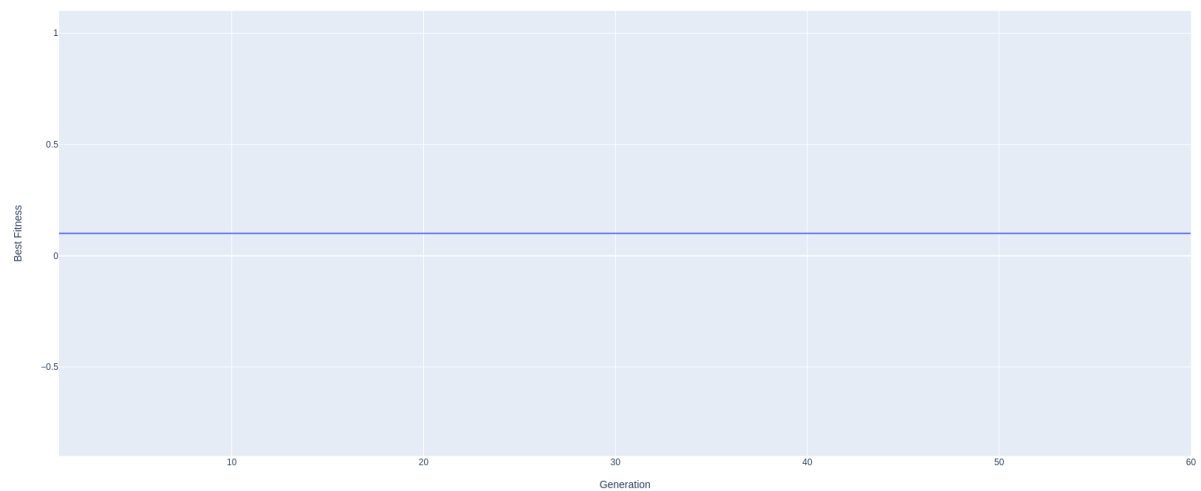
Test-case-2

Statistical Logs

```

Saving Valid Chromosomes, Chromosomes with Minimum Colors and Best Fitness Scores.
It's Possible to Color the Graph with Less than 4 Colors using Random Selected Valid Chromosome with Minimum Colors
Random Selected Valid Chromosome with Minimum Colors --- [0 3 3 1 1 0 0]
Average Fitness Scores --- 0.09999999999999996
Median Fitness Scores --- 0.1
Total Valid Chromosomes in Generation 1 --- (12, 7)
Total Valid Chromosomes with Minimum Colors in Generation 1 --- (3, 7)
```

Fitness Generations



Colored Graph using Valid Chromosomes with Minimum Colors

