DSAI 3202 – Parallel and distributed computing
Assignment 1 – Part 2/2: Navigating the City

This exercise is worth 60 points on your assignment.

The code in this project should be correctly structured and documented. Regardless of the provided files.

1. Objectives:

- Develop Python programs that run the uses genetic algorithms in a distributed fashion using MPI4PY or Celery.

2. Tools and Concepts:

- Python: Programming language.
- Packages: MPI4PY.

3. Deliverables:

- A link to your repository with the correct branch containing:
  - The code.
  - A readme file correctly formatted to explain your program and answer the questions, as well as the results.

4. The genetic algorithm

4.a. Description

Genetic algorithms (GAs) are a class of evolutionary algorithms inspired by the process of natural selection in biology. They are used to solve optimization and search problems by evolving a population of candidate solutions over time.

In a genetic algorithm, each solution is represented as an individual in the population. These individuals are often encoded as strings of binary digits, but other representations are also possible depending on the problem. The quality of each solution is assessed using a fitness function, which assigns a fitness score to each individual based on how well it solves the problem.

The genetic algorithm iteratively improves the population of solutions by applying genetic operators such as selection, crossover, and mutation:

i) Selection:

This operator selects individuals from the current population to form a mating pool. Individuals are chosen based on their fitness, with higher fitness individuals being more likely to be selected. This simulates the survival of the fittest principle in natural selection.

ii) Crossover (Recombination):

This operator combines the genetic information of two parent individuals to create offspring. Crossover is applied with a certain probability, known as the crossover rate. There are various crossover techniques, such as single-point crossover, two-point crossover, and uniform crossover.

iii) Mutation:

This operator introduces random changes to the genetic makeup of individuals. Mutation is applied with a certain probability, known as the mutation rate. It helps to maintain genetic diversity in the population and prevent premature convergence to suboptimal solutions.

The genetic algorithm repeats these steps for a fixed number of iterations or until a satisfactory solution is found. Over time, the population evolves towards better solutions, with the fittest individuals being more likely to survive and reproduce.

4.b. Simplified algorithm

```
1. Initialize a population of individuals (Can be done randomly).
2. Evaluate the fitness of each individual in the population.
3. While termination condition not met:
4.     Select parents from the population based on their fitness.
5.     Perform crossover on pairs of parents to create offspring.
6.     Apply mutation to the offspring.
7.     Evaluate the fitness of the offspring.
8.     Select individuals for the next generation from the offspring and/or
   current population.
9. Return the best individual from the final population.
```

The termination condition can be the number of iterations for instance.

Key Components of Genetic Algorithms:

- **Representation**: The way in which solutions are encoded as individuals in the population.
- **Fitness Function**: A function that evaluates how well an individual solves the problem at hand.
- **Selection**: A method for choosing individuals from the population to reproduce.
- **Crossove**r: A method for combining the genetic information of two parents to create offspring.
- **Mutation**: A method for introducing random changes to individuals.
- **Termination Condition**: A criterion for determining when the algorithm should stop, such as reaching a maximum number of iterations or achieving a desired fitness level.

5. Fleet management using genetic algorithms: *One car version, completing the sequential code* (15 pts).

5.a. Objective

The goal of this assignment is to use a genetic algorithm to optimize the routes for a fleet of delivery vehicles in a city. **The objective is to minimize the total distance traveled by all vehicles while ensuring that each delivery node in the city is visited exactly once by any vehicle**. In this version, we will start with one car that must go through all the nodes.

- **City Layout:** The city is represented as a graph with 20 nodes, where each node corresponds to a delivery location. The distances between nodes are given by a distance matrix. Figure 1 is a representation of the nodes and distance between them. These distances are also represented in the file `city_distances.csv.`
- **Constraints:**
  Minimize the total distance traveled by all vehicles in the fleet.
  - Each delivery node must be visited exactly once by any vehicle in the fleet.
  - Each vehicle must start and end its route at the depot (node 0).
  - The total distance traveled by all vehicles should be minimized.

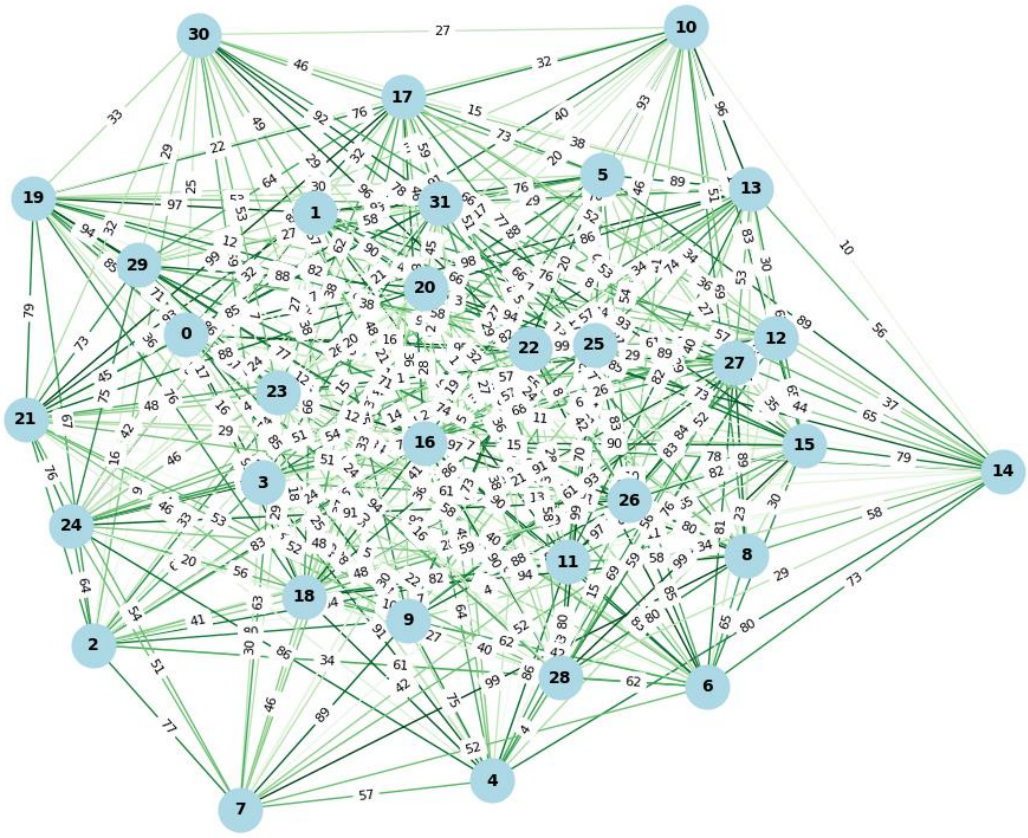

Figure 1. The map of the city with 32 nodes and 400 edges (links).

5.b. Description of the code

There are multiple files in the folder of this project:

- `genetic_algorithms_functions.py` contains the functions execution of the genetic algorithm. ***This file is partially empty, and you need to complete it.***
- `genetic_algorithm_trial.py` contains the script for the to run the trial for the genetic algorithm.
- `city_distances.csv` contains the matrix of the distances between the nodes. In this matrix, cells with values of 100000 are indicative of nodes that are not connected. The normal distance between two nodes is at max equal to 100.

5.c. Completing the functions (10 pts)

There are two function to complete in **`genetic_algorithms_functions.py`**:

- **`calculate_fitness.`**
- **`select_in_tournament.`**

The rest of the functions are:

- **`order_crossover.`**
- **`mutate.`**
- **`generate_unique_population.`**

These functions are described in the docstrings. I invited to read and analyze them.

i) calculate_fitness

The objective of this exercise is to minimize the distance travelled by the delivery car. The function takes the route proposed by the genetic algorithm. This the skeleton of the function:

```python
def calculate_fitness(route,
                      distance_matrix):
    """
    calculate_fitness function: total distance traveled by the car.

    Parameters:
        - route (list): A list representing the order of nodes visited in the route.
        - distance_matrix (numpy.ndarray): A matrix of the distances between nodes.
            A 2D numpy array where the element at position [i, j] represents the
            distance between node i and node j.
    Returns:
        - float: The negative total distance traveled
          (negative because we want to minimize distance).
          Returns a large negative penalty if the route is infeasible.
    """
    total_distance = 0

    # Add your code here.

    return total_distance
```

To complete the code:

- Initialize a variable **`total_distance`** to 0. This will accumulate the total distance traveled.
- Iterate through the route list, and for each consecutive pair of nodes (node1, node2):
    - Retrieve the distance between node1 and node2 from the **`distance_matrix`**.
    - If the distance is equal to 10000 (indicating an infeasible route), directly return a large negative penalty (e.g., 1e6).
    - Otherwise, add the distance to **`total_distance`**.
- After the loop, return the negative of **`total_distance`** as the fitness value.

ii) select_in_tournament

The **`select_in_tournament`** function performs tournament selection, a method used in genetic algorithms to select individuals for reproduction. In this method, a specified number of tournaments are run, and in each tournament, a subset of

individuals (defined by **`tournament_size`**) is randomly selected from the population to compete. The individual with the highest fitness score in each tournament is selected for crossover.

- Create an empty list selected to store the individuals selected through the tournament selection process.
- Use a for loop to run the specified number of tournaments (**`number_tournaments`**). In each iteration of the loop:
  - Randomly select **`tournament_size`** individuals from the population using **`np.random.choice`**.
  - Store the indices of the selected individuals in a list idx.
  - Find the index of the individual with the highest fitness score among the selected individuals using **`np.argmax(scores[idx])`**.
  - Store this index in **`best_idx`**.
  - Append the individual at **`best_idx`** from the population to the **`selected`** list.
- After all tournaments have been run, return the selected list, which contains the individuals selected for crossover.

```
def select_in_tournament(population,
                         scores,
                         number_tournaments=4,
                         tournament_size=3):
    """
    Tournament selection for genetic algorithm.

    Parameters:
        - population (list): The current population of routes.
        - scores (np.array): The calculate_fitness scores corresponding to each individual in the population.
        - number_tournaments (int): The number of the tournamnents to run in the population.
        - tournament_size (int): The number of individual to compete in the tournaments.

    Returns:
        - list: A list of selected individuals for crossover.
    """
    selected = []

    # Add your code here

    return selected
```

5.d. Explain and run the algorithm (5 pts).

- Explain the program outlined in the script **`genetic_algorithm_trial.py.`**
- Run and time the execution of this script.

6. Parallelize the code (20 pts)

After running the code sequentially, the current part of the assignment requires you to run the code in parallel over multiple machines.

- Define the parts to be distributed and parallelized, explain your choices (5 pts).
- Parallelize your program (10 pts).
- Run your code and compute the performance metrics (5 pts).

7. Enhance the algorithm (20 pts).

There are several improvements that can be implemented in the algorithm.

- Distribute your algorithm over 2 machines or more (10 pts).
- What improvements do you propose? Add them to your code (5 pts).
- After adding your improvements, recompute the performance metrics and compare with before the enhancements (5 pts).

8. Large scale problem (10 pts)

- Run the program using the extended city map: **`city_distances_extended.csv`**. Successful execution in feasible time is rewarded with 5 pts.
- How would you add more cars to the problem? (Just explain, 5 pts).

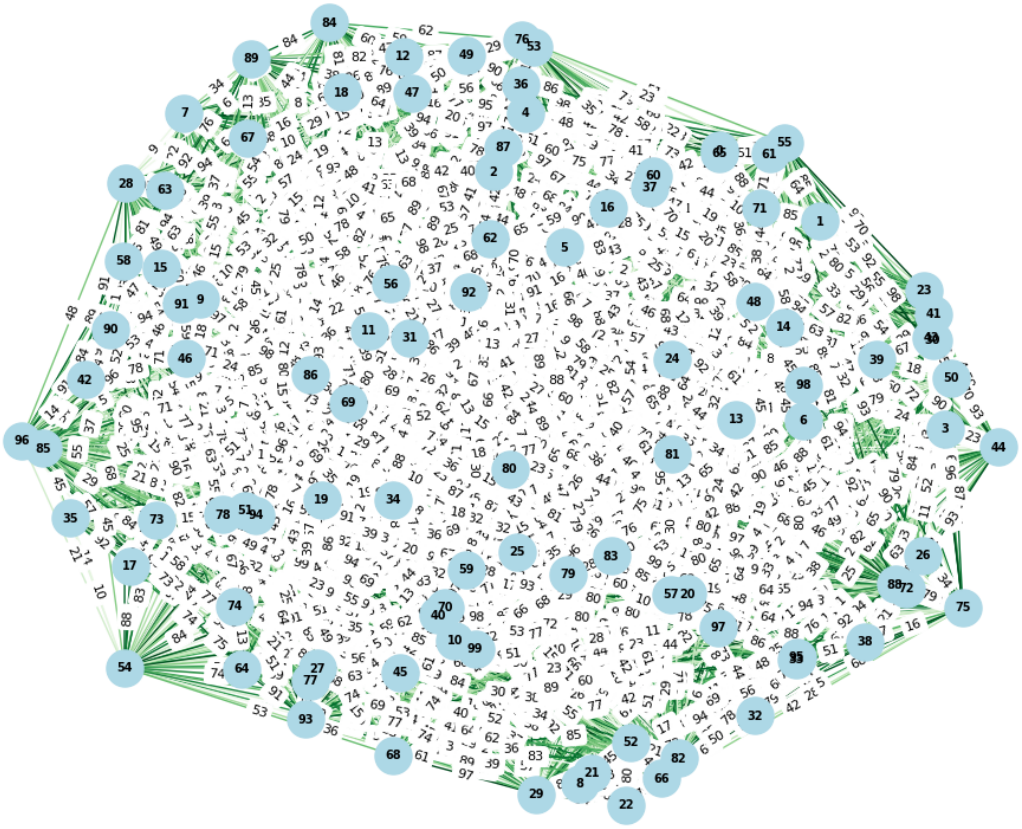

Figure 2. The city is now larger (100 nodes) and has more routes (4000).

9. Bonuses

- Implement and run the code correctly with multiple cars (5 pts).
- Use AWS to do the assignment on multiple machines (5 pts).
- Best solution or fastest execution:
  - In the first part (2 pts).
  - In the second part (5 pts).