NTNU

Kunnskap for en bedre verden

Department of Mechanical and Industrial Engineering

TPK4186 - Advanced Tools for Performance Engineering

Assignment 1 - Prodled Bros

Authors:

Aaryan Neupane

Anne Torgersen

Tora Kristine Løtveit

Date: 12.01.2024

Table of Contents

1 Data Structures and Functions

In this section, we outline the design and rationale behind the data structures and functions developed to manage road networks. The key classes implemented for this purpose are "Edge," "Node," "Empirical Distribution," and "Road Network."

1.1 Definitions

The "Edge" class represents the roads connecting individual nodes, analogous to cities in real-life scenarios. Each edge encapsulates its unique traffic distribution, which fluctuates daily, and is modeled by the "Empirical Distribution" class. Together, these components form the "Road Network" class, mirroring real-world road networks with cities, interconnecting roads, and stochastic traffic patterns.

1.2 Edge

The constructor of the "Edge" class initializes default values for variables such as code, source node, target node, length, a unique distribution, and edge type (which can be uni- or bidirectional) if not provided. To ensure secure modification and customization of these values based on specific requirements, we have implemented a set of functions that safely handle adjustments, taking into account the edge type.

1.3 Node

Each "Node" is characterized by a unique code, along with lists of incoming and outgoing edges. Additionally, it contains a set of unique attributes, initialized to None by default. Similar to the Edge class, we have implemented functions to safely manage the addition, removal, or modification of these variables using helper functions. To facilitate algorithmic tasks, we have also developed helper functions to calculate both the time and distance between a node and all its directly connected neighbors where a path exists.

1.4 Empirical Distribution

An empirical distribution is represented by a dictionary where the keys correspond to each day of the week, and the values are lists of unique tuples. Each tuple contains two elements: the time of the day in minutes and the corresponding estimated driving time for that specific time of day.

Upon creation of an instance of the Edge class, an empirical distribution is automatically generated and assigned to that edge. This ensures that each edge possesses a unique distribution accounting for different days of the week.

To facilitate secure modification of these values, helper functions have been implemented. Additionally, functions are available to randomly generate these tuples, considering variations such as rush hours during weekdays and more consistent traffic patterns during weekends.

Since the randomly generated points may not cover every minute of the day, an interpolate function has been developed to calculate linearly interpolated values between the two closest time points provided.

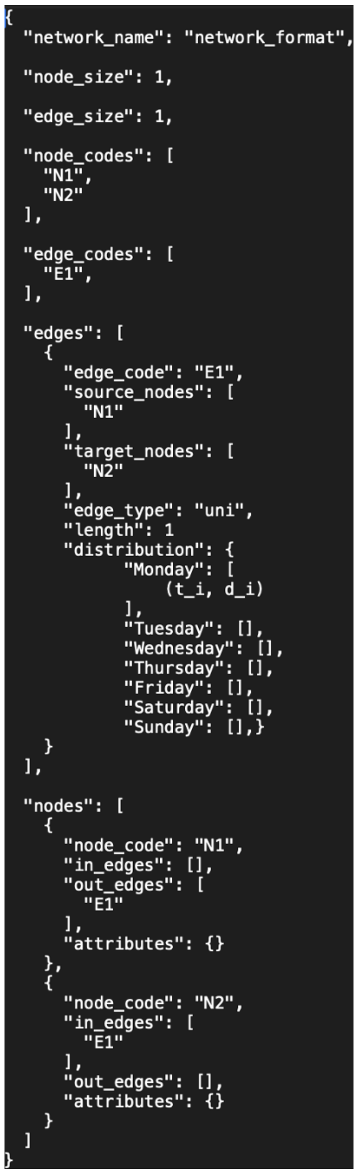
1.5 Road Network

A Road Network is composed of a list of nodes and edges, each with its unique name. This class includes various helper functions to safely manage the addition and removal of variables, as well as to locate specific elements within the network.

Additionally, the Road Network class provides functionality to calculate the shortest distance between two given nodes and also the shortest time required to travel between them, considering the traffic distribution. Furthermore, it offers functions to determine the existence of a path between nodes and whether the entire network is connected or not.

2 Network format

We opted to utilize a JSON file format to store the necessary information for the road networks in a clean and secure manner. This decision was driven by the simplicity of JSON, making it easy to both store and load various road networks. The format we finalized enables us to represent the different classes as objects, significantly simplifying the loading process. With the format below, we can effectively store and organize the essential data for the road networks.

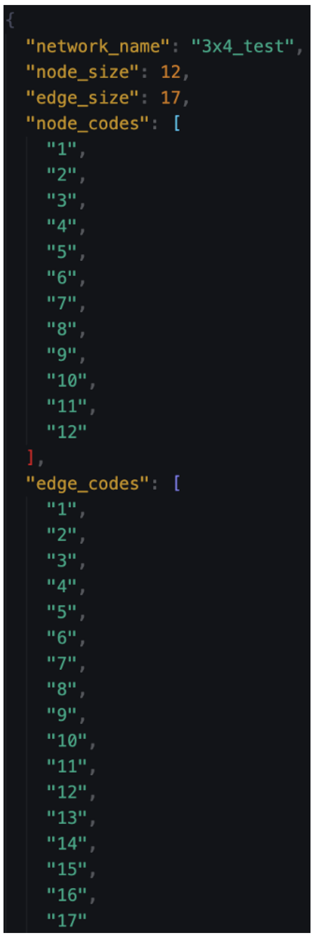


```
{
  "network_name": "network_format",

  "node_size": 1,

  "edge_size": 1,

  "node_codes": [
    "N1",
    "N2"
  ],

  "edge_codes": [
    "E1",
  ],

  "edges": [
    {
      "edge_code": "E1",
      "source_nodes": [
        "N1"
      ],
      "target_nodes": [
        "N2"
      ],
      "edge_type": "uni",
      "length": 1
      "distribution": {
            "Monday": [
                (t_i, d_i)
            ],
            "Tuesday": [],
            "Wednesday": [],
            "Thursday": [],
            "Friday": [],
            "Saturday": [],
            "Sunday": [],}
    }
  ],

  "nodes": [
    {
      "node_code": "N1",
      "in_edges": [],
      "out_edges": [
        "E1"
      ],
      "attributes": {}
    },
    {
      "node_code": "N2",
      "in_edges": [
        "E1"
      ],
      "out_edges": [],
      "attributes": {}
    }
  ]
}
```

Figure 1: Ultimate format

3 Generators

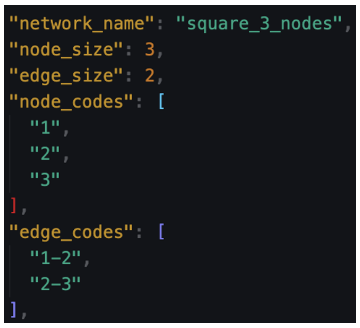
For the grid generator, we developed a function that automatically generates a grid based on the provided parameters of rows and columns. In this grid, the intersections between the rows and columns represent nodes, while the lines connecting them represent edges. We thoroughly tested this function by creating multiple networks, including an example of a 3x4 grid, which successfully matched our objectives.



```
{
  "network_name": "3x4_test",
  "node_size": 12,
  "edge_size": 17,
  "node_codes": [
    "1",
    "2",
    "3",
    "4",
    "5",
    "6",
    "7",
    "8",
    "9",
    "10",
    "11",
    "12"
  ],
  "edge_codes": [
    "1",
    "2",
    "3",
    "4",
    "5",
    "6",
    "7",
    "8",
    "9",
    "10",
    "11",
    "12",
    "13",
    "14",
    "15",
    "16",
    "17"
```

Figure 2: Grid experiments

The second generator we implemented is a square generator, which requires parameters such as name and an initial number of nodes. Using this initial node count, we construct the smallest possible square grid with the corresponding number of nodes. The determination of edges between nodes is facilitated by the helper function ”probability,” which utilizes a provided probability equation to decide the existence of edges based on the distances between initialized nodes. Through multiple tests with an initial node count of 3, we observed significant variation in the number of edges generated each time.

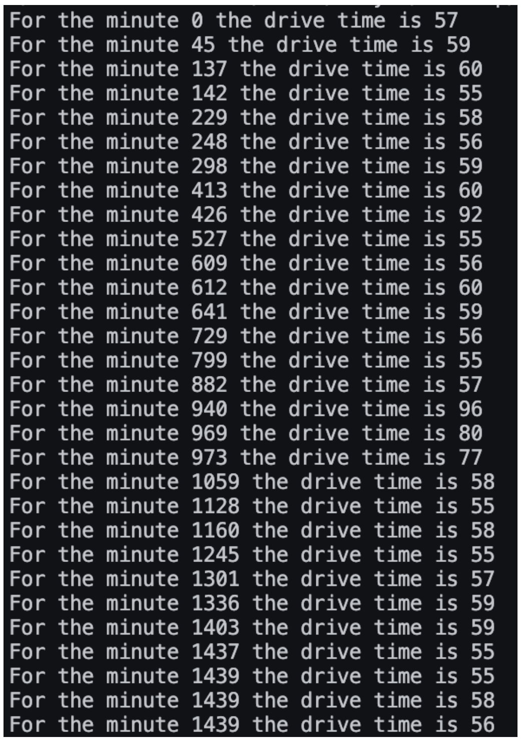


```
"network_name": "square_3_nodes",
"node_size": 3,
"edge_size": 2,
"node_codes": [
  "1",
  "2",
  "3"
],
"edge_codes": [
  "1-2",
  "2-3"
],
```

Figure 3: Square experiments

4 Traffic Statistics

As stated in the introduction, our implementation includes the ”Empirical Distribution” class, which comprises a dictionary containing tuples for each day of the week. This method allows us to assign a unique distribution to each edge upon its creation. In the experiment described below, we instantiated an edge with an initial length of 30 and printed out the distribution for Monday.



```
For the minute 0 the drive time is 57
For the minute 45 the drive time is 59
For the minute 137 the drive time is 60
For the minute 142 the drive time is 55
For the minute 229 the drive time is 58
For the minute 248 the drive time is 56
For the minute 298 the drive time is 59
For the minute 413 the drive time is 60
For the minute 426 the drive time is 92
For the minute 527 the drive time is 55
For the minute 609 the drive time is 56
For the minute 612 the drive time is 60
For the minute 641 the drive time is 59
For the minute 729 the drive time is 56
For the minute 799 the drive time is 55
For the minute 882 the drive time is 57
For the minute 940 the drive time is 96
For the minute 969 the drive time is 80
For the minute 973 the drive time is 77
For the minute 1059 the drive time is 58
For the minute 1128 the drive time is 55
For the minute 1160 the drive time is 58
For the minute 1245 the drive time is 55
For the minute 1301 the drive time is 57
For the minute 1336 the drive time is 59
For the minute 1403 the drive time is 59
For the minute 1437 the drive time is 55
For the minute 1439 the drive time is 55
For the minute 1439 the drive time is 58
For the minute 1439 the drive time is 56
```

Figure 4: A traffic distribution on a weekday with rush-hours

5 Fastest routes

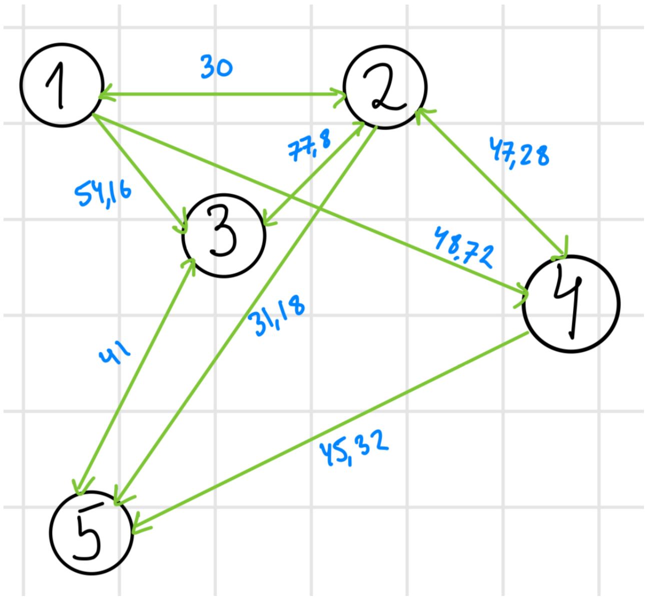
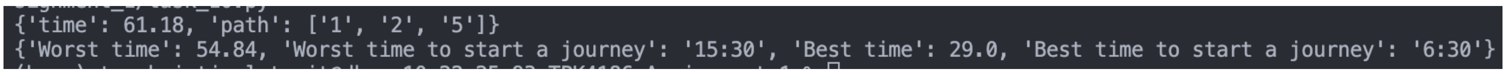


Figure 5: Outcome from the functions fastest_route and worst_best_time

The function fastest_route calculates and extracts the fastest route from a given node x to another node y at a given start time. In this experiment we loaded our network (figure 2) and provided the function with a start node (1), and an end node (5), the day (Monday) and time (06:40) of departure. Line 1 in figure 3 shows the result of running the code. The fastest route from node 1 to 5 is through 2.

The function worst_best_time calculates the best and the worst time to start a journey from a given node x to another node y in a given day and in the week. In this experiment we loaded or network (figure 2), a start node (1), an end node (2) and the day of departure. Line 2 in figure 3 shows the result of running the code.

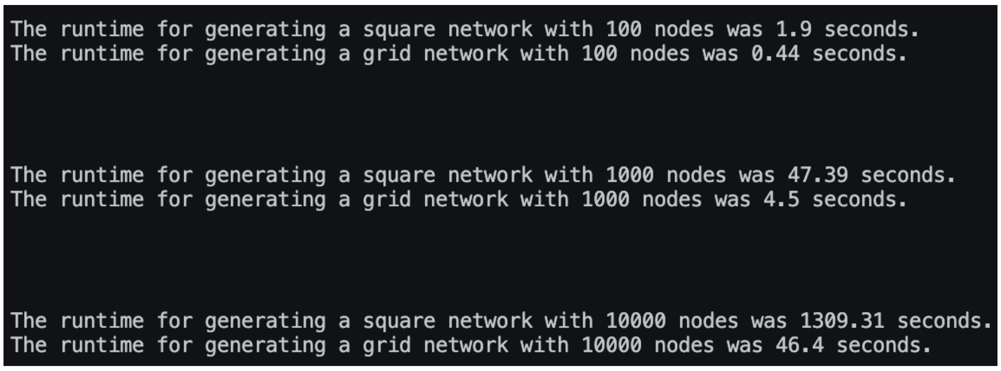


```
{'time': 61.18, 'path': ['1', '2', '5']}
{'Worst time': 54.84, 'Worst time to start a journey': '15:30', 'Best time': 29.0, 'Best time to start a journey': '6:30'}
```

Figure 6: Results from the experiment

6 Scalability

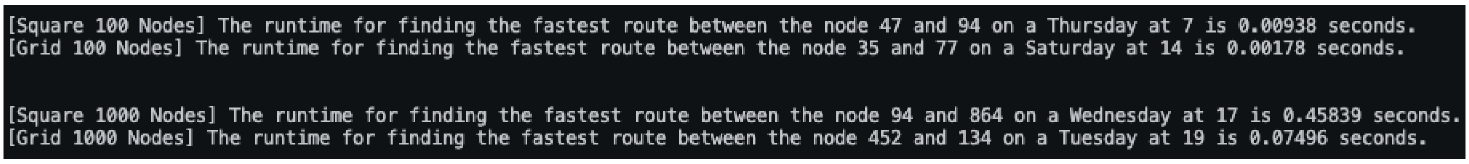
To assess the scalability of our program, we conducted experiments involving the creation of multiple networks with varying sizes and measured their run times. Subsequently, we utilized our built-in algorithms to calculate different distances within these networks.



```
The runtime for generating a square network with 100 nodes was 1.9 seconds.
The runtime for generating a grid network with 100 nodes was 0.44 seconds.



The runtime for generating a square network with 1000 nodes was 47.39 seconds.
The runtime for generating a grid network with 1000 nodes was 4.5 seconds.



The runtime for generating a square network with 10000 nodes was 1309.31 seconds.
The runtime for generating a grid network with 10000 nodes was 46.4 seconds.
```

Figure 7: Run times for the two generators with varying node sizes

As depicted in Figure [7], the run times for the square generator exhibit a noticeable increase as we increase the number of nodes. This behavior is expected, considering the current lack of optimization in our code for the generator.



```
[Square 100 Nodes] The runtime for finding the fastest route between the node 47 and 94 on a Thursday at 7 is 0.00938 seconds.
[Grid 100 Nodes] The runtime for finding the fastest route between the node 35 and 77 on a Saturday at 14 is 0.00178 seconds.


[Square 1000 Nodes] The runtime for finding the fastest route between the node 94 and 864 on a Wednesday at 17 is 0.45839 seconds.
[Grid 1000 Nodes] The runtime for finding the fastest route between the node 452 and 134 on a Tuesday at 19 is 0.07496 seconds.
```

Figure 8: Run times for the route algorithms with varying node sizes

Additionally, we conducted tests to measure the run times for the route calculator algorithms using different square and grid networks. As shown in Figure [8], these calculations are performed relatively quickly due to the utilization of Dijkstra's algorithm, which helps optimize the process.

These findings provide valuable insights into the scalability of our program and highlight areas where further optimization may be beneficial.