MODA - Bicycle Routing

Myriana Miltiadous (s3699463), Katerina Zacharia (s3783049)

May 26, 2023

For the second assignment, we decided to choose the third case, with some small changes, which is about planning a route that has the minimum distance and the maximum comfort. This is a multi-objective problem where we have two objectives: Distance and Comfort, which consists of four factors; scenic beauty, roughness, safety, and slope. We utilize Python and the DESDEO library for the solution.

Problem Definition

Considering a network of connections between bicycle vertices, as shown in Figure 1, we define an environment which consists of 20 nodes and we set as starting point the node 0, and as a finish point the node 19. The routes that we allow for our data set consist of 7-10 nodes.

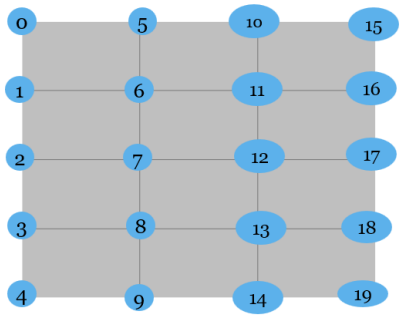

Figure 1: Permutation representation of the network with 20 nodes

In this network, it is possible to visit a node only if you visited one of the neighbouring nodes, e.g., in order to visit node 12, you have to be either at position 11, 7, 13 or 17. With math equations, we define that for each node i, there is a transition to node j when the $x_{i,j}$ element of a matrix, that will be explained later in this report, is one and there is no transition when it is zero (i.e for example $x_{12,17} = x_{17,12} = 1$ but $x_{12,1} = 0$).

For each of these transitions, the following characteristics are defined:

- $a_{ij} \in \{1, 2, 3, 4, 5\}$. This is a vector for scenic beauty, where 1 is the worst and 5 is the best score, i.e., the most beautiful,
- $b_{ij} \in \{1, 2, ..., 5\}$ is a vector about roughness, where 1 stands for very rough path, and 5 for a smooth bicycle path,
- $s_{ij} \in \{1, 2, ..., 5\}$ is a variable that signifies the safety of a connection, where 1 stands for dangerous traffic situation, and 5 for very safe,
- $l_{ij} \in \{1, 2, ..., 5\}$ stands for the slope. If the connection is steep the score is 1, whilst the score is 5 if it is gentle descent.
- $d_{ij} \in [50, 200]$ is a distance matrix, which is the total distance between the two knots in meters.Even though our representation consists of squares, the distances are not necessarily the same. For example $d_{0,5}$ should not be equal to $d_{1,6}$.

The way we approach this kind of problem is by taking the average of the first four factors to extract the comfort, which we want to maximize, whilst also minimizing the distance. In order to achieve this, we find Pareto front for the efficient set. However, we first need to designate the constraints of the problem.

$$14 \leq \sum_{i,j}(a_{i,j}), \sum_{i,j}(b_{i,j}), \sum_{i,j}(l_{i,j}), \sum_{i,j}(s_{i,j}) \leq 50 \tag{1}$$

$$AVG = \frac{\sum_{i,j}(a_{i,j}) + \sum_{i,j}(b_{i,j}) + \sum_{i,j}(l_{i,j}) + \sum_{i,j}(s_{i,j})}{4} \geq 17 \tag{2}$$

$$D = \sum_{i,j}(d_{i,j}) \leq \frac{max(d) + min(d)}{2} \tag{3}$$

Constraint 1 is set within that boundaries because we considered that if every independent factor that constitute comfort has a minimum score of 2, and the path has a minimum of 7 edges, then the total minimum we want is 14. The upper bound, i.e., 50, depict the maximum which is exported when all the independent factors indicate a score of 5 but we also have routes that include 10 edges.
Regarding Constraint 2, we set the average of the sums of the independent factors (a,b,s,l) to be higher or equal to 17. This comes from the need of having an even better comfort average than the average of all the sums to be at least equal to 14, as set in the previous Constraint. This way we make sure that some of those sums will be greater than 14, meaning that at least one path has a factor with a score greater than 2.
Concerning the last constraint (Constraint 3), we set the sum of distance to be less than the average of the route with the highest and the one with the lowest sum of distance in the data set, which contains all the possible routes that can get someone from node 0 to node 19.

Dataset

In order to create the data-set, except the above matrices, it was necessary to initiate another one, that stores the neighbouring relations between the nodes. This matrix works as adjacency array, where 1 is given when node i is connected to node j, and 0 when this does not apply. We named this array as *'x'*, and we multiply it with all the other arrays so as to be correctly defined. For example, node 5 is connected to nodes 0, 6, and 10, and therefore only towards these nodes, there are scenic beauty, roughness, slope, safety, and distance. We end up with five symmetric matrices.

To find all the possible paths from starting point (0) to the ending point (19), we implement an algorithm that is explained in the below list.

1. Create an empty list called *paths* to store the valid paths.
2. Define a recursive procedure called backtrack, where append the current position to the path. If current is the end position, create a copy of path and add it to paths. If not, find valid moves by examining non-zero values in the corresponding row of the matrix.
3. For each valid move, update current position.
4. Check if the new position is within the matrix bounds and not already visited in the current path.
5. If conditions are met, recursively call backtrack with updated path and current position.
6. Remove the last position from path to backtrack. Call backtrack with an empty path and the start position to initiate the algorithm.
7. Return the paths list containing valid paths from start to end position in the matrix.

However, there are some restrictions, such as: a node cannot be added twice in the path, i.e., they are only once visited, and the path does not have crossovers that may lead to the first restriction. Moreover, we only save the routes that consist of 7 to 10 nodes, extracting the first one.

Lastly, we combine all these arrays in one, so as to have only one array to import in our optimization problem. This 2-D array stores in rows all the possible paths retrieved, whilst the columns give the corresponding sum of the independent factors and total distance of the specific route. The total paths we gathered are 126. Some examples are given on Figure 2. Figure 3 illustrates the route of the first case of the data.

Paths	A	S	B	L	D
[0, 1, 2, 3, 4, 9, 8, 13, 14, 19]	38	31	38	23	1107
[0, 1, 2, 3, 4, 9, 8, 13, 18, 19]	36	29	38	25	1166
[0, 1, 2, 3, 4, 9, 14, 13, 18, 19]	33	31	34	26	1311

Figure 2: Example of data

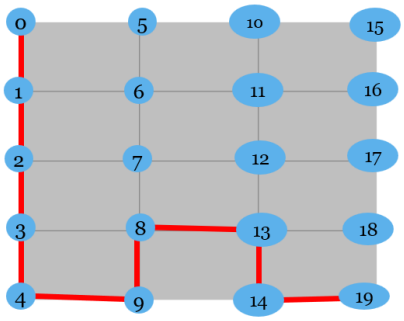

Figure 3: Representation of route {0,1,2,3,4,9,8,13,14,19}

Algorithmic Approach and Pareto Front

To solve the aforementioned multi-objective optimization problem and find the efficient set as well as the pareto front,the desdeo library was used.Firstly the building of the problem was performed with the following steps:

1. Definition of the variables names,

```
var_names = ["A", "S", "B","L", "D"]
```

   where A stands for the sum of beauty in the relative path, B for roughness, S for safety, L for slope and D for distance .
2. Definition of the lower, upper and initial values of those variables, taking into consideration the constraints that need to be satisfied.
3. Construction of the variables objects using the variable_builder function.

```
variables = variable_builder(var_names,
                             initial_values,
                             lower_bounds,
                             upper_bounds)
```

4. Definition of the objective functions
   $obj1 \longrightarrow AVG \longrightarrow max$
   $obj2 \longrightarrow D \longrightarrow min$
   and use ScalarObjective().
5. Definition of the constraint function and use ScalarConstraint().
6. Initialization of the problem with MOProblem() .

```
prob = MOProblem(objectives=[f1, f2], variables=variables,
    constraints=[cons1,cons2])
```

7. Evaluate the described data-set of the specified problem and get results as well as constraint violations.

```
y = prob.evaluate(data)
```

After building and evaluating the problem as mentioned, using the objectives results from the evaluation part, the non dominated front was found using the function nd2() from pygmo library and stored in the variable data_pareto. Then, after the addition of a new column 'AVG' in the data-set with the average comfort calculations, the definition of which objective should be maximize and which should be minimized was done. By using it to define the DataProblem class, the problem object was created.

```
maximize = pd.DataFrame([[False, True]], columns=[' D','AVG'])
problem = DataProblem(data=training_data, variable_names=[' A',' S','
    B',' L'], objective_names=[' D','AVG'], maximize=maximize)
```

That class uses the described data frame and provides a train method which trains all the objectives sequentially. Using that class the evaluation of two methods of training, Lipschitzian Regressor and Gaussian Process Regressor, was performed using the evolutionary algorithm NSGA-III with different selection strategies,"optimistic" and "robust".

The well-known multi-objective evolutionary algorithm NSGA-III makes use of a genetically based strategy to iteratively search the solution space and discover the Pareto front, which denotes a collection of non-dominated solutions. While the "robust" selection type prioritizes exploitation by choosing solutions that are closer to the present Pareto front, the "optimistic" selection type supports exploration by choosing alternatives with the highest predicted improvement.

Using the two training methods for both strategies of NSGA-III, the goal is to optimize the performance of both LR and GPR models by finding the best set of hyperparameters for the given problem. Each iteration of the evolution process entails producing new candidate solutions, assessing their fitness using the specified model, and updating the population in accordance with the selection strategy.

Results

The acquired results from the different algorithms were utilized to conduct a visualization of the fronts in the given problem. The values for the AVG objective are scaled and multiplied by 10 for better representation. The fronts on the left side of Figure 4 display all the Pareto-optimal solutions found through optimization. Finally, on the right of Figure 4 the initial data-set used for the specific problem is additionally visualized with its Pareto front.

The resulting Pareto front obtained through the optimization process, using the NSGA-III algorithm, it may better represent the true trade-offs and optimal solutions in the problem space, than the calculation of pareto front by only using the dataset, and is often preferred for large and complex data-sets.

As it is obvious from the two plots, the Lipschitzian regressor appears to be more effective in capturing the underlying patterns and trade-offs than Gaussian Process Regressor, because its results are comparable to those of the direct dataset-based approach(right figure). However, it is crucial to take into account elements like dataset characteristics, computational efficiency, and interpretability when deciding between the Lipschitzian regressor and the Gaussian Process Regressor for other optimization problems.

Two solutions from the efficient set of the provided data-set that deserve particular attention are the points (994,302.5) and (1054,355) of the right figure with the first having AVG=30.25,D=994 and path which includes the nodes 1, 6, 7, 12, 13, 8, 9, 14, 19. That point prioritizes minimum distance. On the other hand, the second point which path's includes the nodes 0, 5, 10, 11, 12, 13, 8, 9, 14, 19 has AVG=355, D=1054 and prioritizes maximum comfort. Those points are particularly mentioned since they can represent the trade-off between comfort and distance. If for example by seeing the first point the goal is to choose between its neighbors and itself, it can be observed from the figure that the point on its right depicts bigger distance but not as much comfort. Concerning its left point, it showcases less distance than before but considerably less comfort too. So, choosing another point between those three probably is not worth it. The same thought can be applied for all the points in the efficient set.

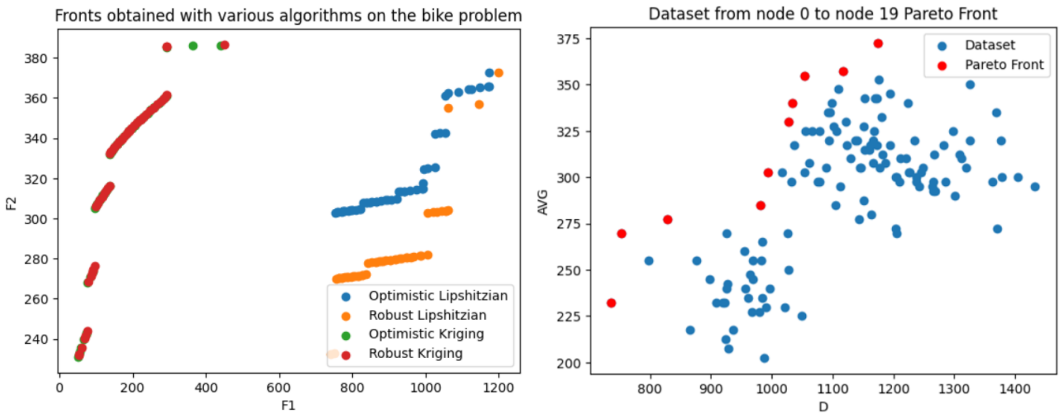

Figure 4: Left Figure:Pareto fronts obtained using Lipschitzian and Gaussian Process Regressor training with optimistic and robust selection with NSGA-III algorithm. Right Figure: Visualization of the data-set used for the Problem and its Pareto Front

Discussion

The above methodology can be also applied with different data-sets with any starting and any finish point and also with different values for comfort and distance variables. The data-sets can be obtained using the same procedure.

To sum up, following the same approach, it can lead to efficient sets for routes from any starting point A to any end point B. According to each person's preference, on which is the most important objective between distance and comfort, one can decide which option from the efficient set is the best one for them.

References

Desdeo framework from github link