IE 313
SUPPLY CHAIN MANAGEMENT

Fall 2022-2023

Assignment III

09.01.2023

Selin Coşkun

2018402213

Ali Can Şahin

2018402189

Table of Contents

Case Definition	2
Model Analysis	3
Interpretation of the Results	5
Drawbacks of the Model	6
References	7

CASE DEFINITION

In this case, it is tried to identify the possible route with minimum cost for a salesman to travel all 81 cities in Turkey. Here, cost is defined as the total kilometer that a salesman traverses and there is only one vehicle, which is salesman. Therefore, this case is an example of a Traveling Salesman Problem (TSP) and a model corresponding to this problem is built.

There are 3 different model can be used for this type of problem. These 3 types of models have a common part, which is assignment constraints.

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j:j\neq i} x_{ij} = 1 \qquad j = 1, 2, \cdots, n \\ & \sum_{i:i\neq j} x_{ij} = 1 \qquad i = 1, 2, \cdots, n \end{aligned}$$

Figure 1: Base Model with objective function and assignment constraints.

In this base model, we have 1 decision variable x_{ij} and one parameter c_{ij}. x_{ij} values indicate whether salesman goes from city i to j and c_{ij} values indicate corresponding cost (in this case, it is distance from i to j) of traversing from city i to j. First two constraints ensure that all cities must be visited, and they are called assignment constraints. However, here the model does not account for the subtours. Subtour means that salesman can visit all the cities, but the route may not have connected to all cities. In this case, it is not desirable. Therefore, an additional constraint should be added to the model.

While working on the model, the subtour constraint in figure 2 is selected.

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1 \text{ for } S \subset N, 2 \leq |S| \leq n - 1$$

Figure 2: Subtour Elimination Constraint

The main reason of this choice is actually from experience. At the beginning, it is tried to solve the model with the MTZ formulation. However, it takes too much time for computer to compile it and the base model with the subtour constraint in figure 2 is used. This model gives better solution in terms of the time that is needed to solve the problem.

Finally, in this case only the distance matrix between cities was given and used. Additionally, the coordinates of the cities were collected from an outsource and used for visualization purposes.

MODEL ANALYSIS

In this model, Gurobi solver is used to identify the best possible route for the problem because of its efficiency. The model has 2 stages one is for data preparation and one for the model construction.

Data Preparation

To be able to build the model and run it, we need to make some data adjustments in the program. Firstly, the data is read, and a very large number is assigned to the cost of traversing from city i to city i. By doing that it is ensured that model will not provide solution in which salesman can go from one city to the same city. Then a list which is called "city" is declared and a distance dictionary in which keys are the 2-way combinations of all cities and values are the distance between the corresponding cities. These data structures and data will be used in the model constructing part.

Model Construction

In this part, model is constructed by using the prepared data in the earlier stages for random 15 cities and for all 81 cities separately. In this section, the model for the random 15 cities will be explained. The model for the other part has the exact same logic and the main idea is the same.

```
#Model Construction
from gurobipy import GRB

m = gurobi.Model()

vars = m.addVars(random_distance.keys(),obj = random_distance, vtype=GRB.BINARY, name='x')

for i, j in vars.keys():
    vars[j, i] = vars[i, j]

cons1 = m.addConstrs(vars.sum(c, '*') == 2 for c in random_cities)
```

Figure 3: Construction of the Model for by using Gurobi Solver

In the first line of the figure 3, a package from gurobipy is installed for solving problem, and then m is the defined as the model which will be prepared and solved.

After initializing the model m, decision variable is defined. This variable corresponds x_{ij} values in the model in figure 1. These variables are binary variables, and its indices are the cities. All these necessary information is provided to the model in the third line of the figure 3.

Then to be more efficient x_{ij} and x_{ji} values are made equal and a constraint which indicates that the sum of the x_{ij} values is equal to 2 because of the equality of the decision variables.

After these constraints, it must be ensured that subtours are eliminated. Here, the constraint in figure 2 is used. However, usage of these constraint creates so many lazy constraints and because of this, it may take too much time to solve the problem. To avoid this, a feature of the Gurobi solver is used. This feature is Callback.MIPSOL. To use this feature two different functions are defined, subtourelim() and subtour(). Subtourelim function helps the solver to eliminate lazy constraints, which are the infeasible subtours, from the model and decreases the number of constraints to examine. It calls subtour function in its body and determine the subtours. Then it identifies whether it is feasible or not. If it is not feasible these constraints removed from the execution. That improves the efficiency of the program besides eliminating subtours.

```
# Callback - use lazy constraints to eliminate sub-tours

def subtourelim(model, where):
    if where == GRB.Callback.MIPSOL:
        # make a list of edges selected in the solution
        vals = model.cbGetSolution(model._vars)
        selected = gurobi.tuplelist((i, j) for i, j in model._vars.keys()
                                    if vals[i, j] > 0.5)
        # find the shortest cycle in the selected edge list
        tour = subtour(selected)
        if len(tour) < len(cities):
            # add subtour elimination constr. for every pair of cities in subtour
            model.cbLazy(gurobi.quicksum(model._vars[i, j] for i, j in combinations(tour, 2))
                         <= len(tour)-1)

# Given a tuplelist of edges, find the shortest subtour

def subtour(edges):
    unvisited = cities[:]
    cycle = cities[:] # Dummy - guaranteed to be replaced
    while unvisited:  # true if list is non-empty
        thiscycle = []
        neighbors = unvisited
        while neighbors:
            current = neighbors[0]
            thiscycle.append(current)
            unvisited.remove(current)
            neighbors = [j for i, j in edges.select(current, '*')
                         if j in unvisited]
        if len(thiscycle) <= len(cycle):
            cycle = thiscycle # New shortest subtour
    return cycle
```

Figure 4: Subtourelim and Subtour Functions to eliminate lazy constraints and subtours.

Final part of the model is the execution phase. Here, the whole program is run and optimal solution is found.

```
m._vars = vars
m.Params.lazyConstraints = 1
m.optimize(subtourelim)
```

Figure 5: Execution of the program

INTERPRATION OF THE RESULTS

Random 15 City

Here randomly 15 different cities are selected, and it is tried to find an optimal route. Gurobi could find an optimal route very fast and optimal value was 4794 km. It indicates that all these cities can be traversed and total km that should be traversed is this value. When the decision variables are examined, it can be seen that mostly the adjacent cities are merged in order to minimize the total distance. For the 81 city part, a map is drawn and it will be seen that similar situation is also occurred there. (The results of the decision variables will be provided along with the code file)

```
Optimal solution found (tolerance 1.00e-04)
Best objective 4.794000000000e+03, best bound 4.794000000000e+03, gap 0.0000%
```

Figure 5 : Output of the Program for Random 15 Cities

Random 81 City

A similar output is reached here, and the optimal value is 9938 km. Here, a map is drawn from the results that are obtained from the program. From this map, it can be seen that generally adjacent cities are linked together to minimize the total cost. Additionally, there are some arcs that can be infeasible in the real life. These drawbacks will be examined in the following subsection.

```
Optimal solution found (tolerance 1.00e-04)
Best objective 9.938000000000e+03, best bound 9.938000000000e+03, gap 0.0000%
```

Figure 6: Output of the Program for 81 Cities

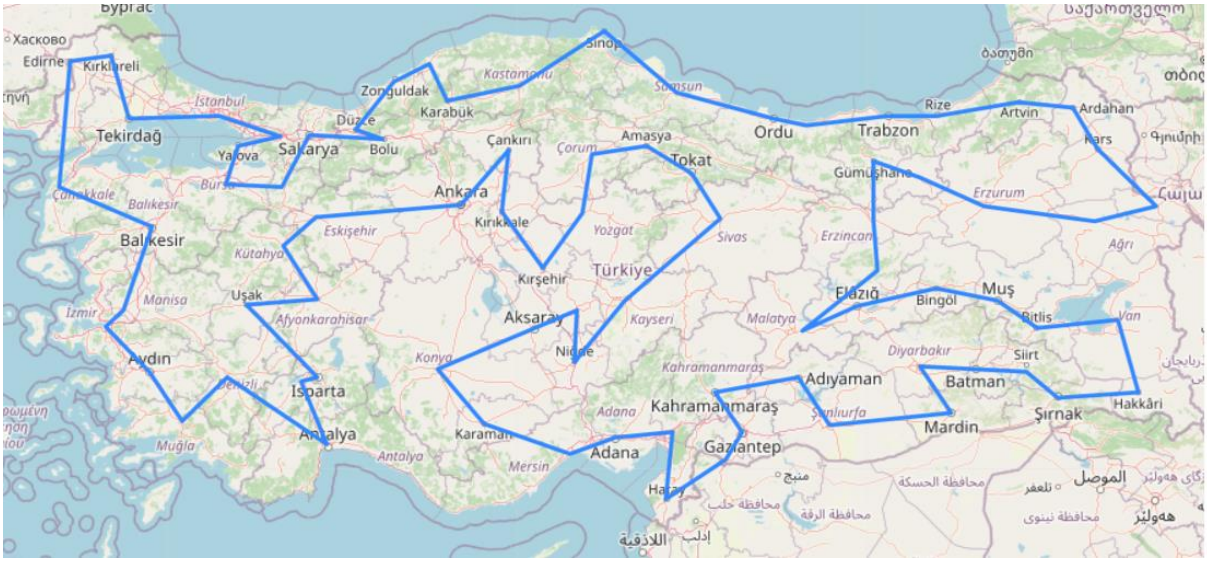

Figure 7: Optimal Route on the Turkish Map

DRAWBACKS OF THE MODEL

From figure 7, it can be seen that there are some infeasible solutions for the real life. For example, in the east region, salesman leaves Turkey for a moment (From Kars to Ağrı). Similarly, from Bitlis to Van salesman traverses directly over the Lake Van. In real life, these routes cannot be achieved. Similar situations occur also for some another arcs.

Additionally, other factors like whether there is a road in the shortest path from one city to another, how many hours it takes regarding the conditions of different regions and performance of different vehicles, how much are the other costs, and the budget allowance are not examined in this study. Therefore, it can be said that this model is not appropriate for the real life, but it is a good example for the TSP problem.

REFERENCES

https://www.gurobi.com/

Lecture Notes