The Travelling Salesman Problem (TSP) for 15 and 81 cities of Turkey

The Travelling Salesman Problem (TSP) defines a set of cities with pairwise distances. The problem seeks a route for the salesman that visits each city exactly once and returns to the origin city while minimizing the total length of the cycle.

In this project, our goal is to find the optimal route for a salesman that must visit cities of Turkey. This salesman will start this travel from city 1 and after visiting all the 15 cities which are chosen randomly, they will return to city 1 in part 1. Then, this salesman will start this travel from city 1 and after visiting all the 81 cities, they will return to city 1 in part 2.

TSP Model Formulation

$$\min \sum_{(i,j\,\epsilon\, Pairings)} d_{i,j} \times x_{i,j} \qquad (1)$$

$$\text{s.t.} \quad x_{i,j} = x_{j,i} \qquad \forall (i,j)\,\epsilon\, pairings \qquad (2)$$

$$\sum_{(i,j)\,\epsilon\, pairings} x_{i,j} = 2 \qquad \forall i\,\epsilon\, cities \qquad (3)$$

$$\sum_{(i\neq j)\,\epsilon\, S} x_{i,j} \leq |S| - 1 \qquad \forall S \subset cities,\ 2 \leq |S| \leq n-1 \qquad (4)$$

Sets and Indices:

i,j ϵ cities: Indices and set of Turkey’ s cities

Pairings ={ (i,j) ϵ cities*cities } : set of allowed pairings

S⊂cities : A subset of Turkey’ s cities

G=(cities, pairings) : A graph where the set cities define the set of nodes and set pairings define the set of edges.

Parameters:

$d_{i,j} \in R^{+}$: Distance from city i to city j for all (i,j) $\in$ pairings. Distance from city i to j is equal to distance from city j to city. $d_{i,j} = d_{j,i}$

Decision Variable:

$x_{i,j}$ ={0,1} : This variable is equal to 1 if we decide to connect city i with city j. Otherwise this variable is equal to 0.

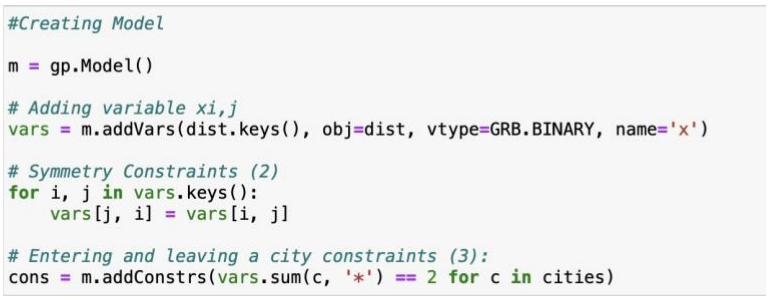
Objective Function(1):

Minimize the total distance of a route. A route is a sequence of cities where the salesman visits each city only once and returns to the starting city.

Constraints:

Symmetry constraints (2) : For each edge (i,j) ensure that the cities i and j are connected, if the previous city is visited immediately after visiting the next city.

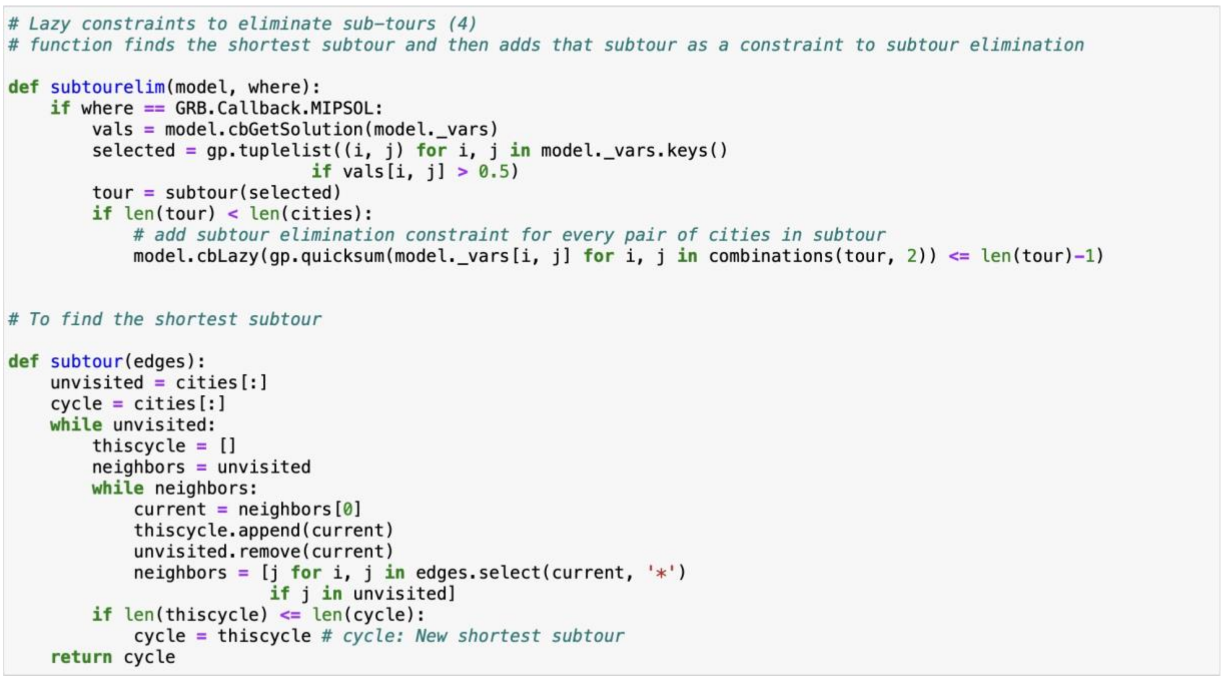
Entering and leaving a city constraints (3): For each city i, ensure that this city is connected to two other cities.



```
#Creating Model

m = gp.Model()

# Adding variable xi,j
vars = m.addVars(dist.keys(), obj=dist, vtype=GRB.BINARY, name='x')

# Symmetry Constraints (2)
for i, j in vars.keys():
    vars[j, i] = vars[i, j]

# Entering and leaving a city constraints (3):
cons = m.addConstrs(vars.sum(c, '*') == 2 for c in cities)
```

Subtour elimination constraints (4): These constraints ensure that for any subset of cities S of the set of cities, there is no cycle. That is, there is no route that visits all the cities in the subset and returns to the origin city. Each subset S, sub-tours are prevented, ensuring that the number of arcs selected in S is smaller than the number of S nodes. $x_{i,j}$ is a binary variable and is equal to 1 when the nodes of i,j are visited.S is a set of vertices between 2 and cities − 1 because two nodes cannot take a tour, and the minimum number for making a tour is 3.

*The number of cities of the TSP is 81, then the possible number of routes is 81!. Since there are an exponential number of constraints (2^81- 2) to eliminate cycles, lazy constraints are used to dynamically eliminate those cycles. Lazy constraints are constraints that should be satisfied by any solution to the problem but they are not generated upfront. The lazy constraint callback checks whether the optimal solution found by the solver contains subtours. If yes, then subtour elimination constraints are added that forbid these subtours. If not, then the optimal solution forms a true solution of the TSP problem, as it contains only on tour. First, a function finds the shortest subtour and then adds that subtour as a constraint to subtour elimination.



```python
# Lazy constraints to eliminate sub-tours (4)
# function finds the shortest subtour and then adds that subtour as a constraint to subtour elimination

def subtourelim(model, where):
    if where == GRB.Callback.MIPSOL:
        vals = model.cbGetSolution(model._vars)
        selected = gp.tuplelist((i, j) for i, j in model._vars.keys()
                                if vals[i, j] > 0.5)
        tour = subtour(selected)
        if len(tour) < len(cities):
            # add subtour elimination constraint for every pair of cities in subtour
            model.cbLazy(gp.quicksum(model._vars[i, j] for i, j in combinations(tour, 2)) <= len(tour)-1)


# To find the shortest subtour

def subtour(edges):
    unvisited = cities[:]
    cycle = cities[:]
    while unvisited:
        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 # cycle: New shortest subtour
    return cycle
```

This modeling requires the following libraries in Python:

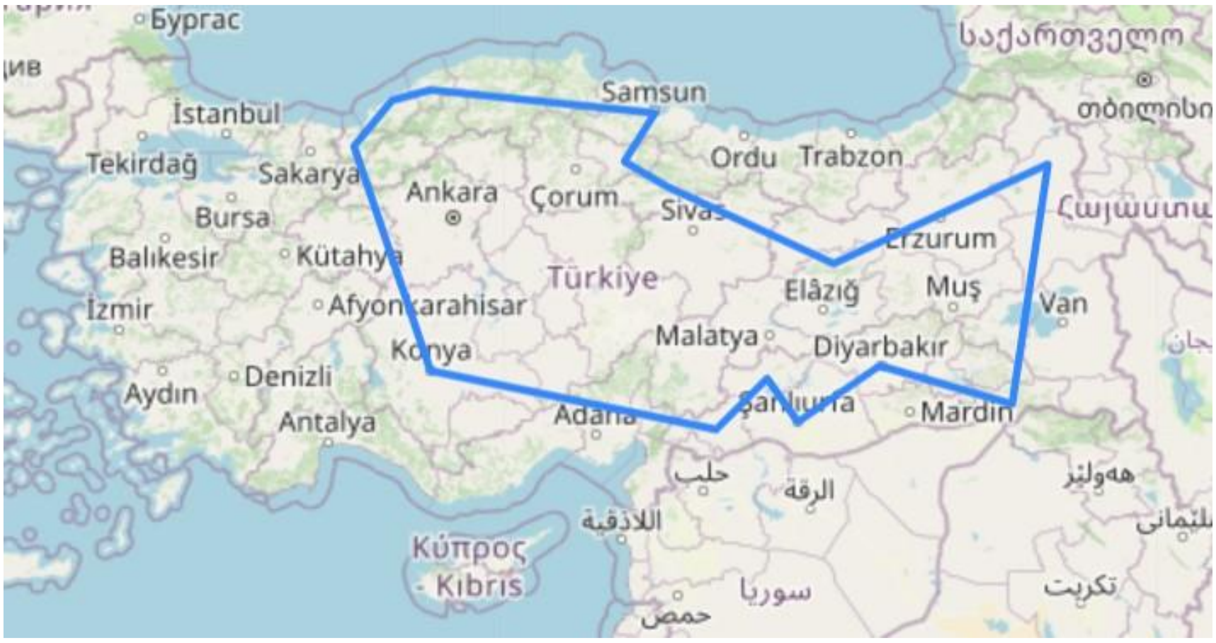
- **folium**: to create maps.
- **gurobipy**: provides Gurobi algorithms and lazy constraints to solve MIP models.

TSP for 15 cities:

In the first part of the assignment, the TSP algorithm applied 15 cities which were chosen randomly. The road distances between 81 cities in Turkey is given. Seed=10 is used. The number of pairing (i,j) for 15 cities is 105 which is a combination (15,2).

To illustrate 15 cities' routes on the map, 81 cities’ coordination is needed. The below link contains 81 cities' coordination:

https://gist.githubusercontent.com/ozdemirburak/4821a26db048cc0972c1beee48a408de/raw/4754e5f9d09dade2e6c461d7e960e13ef38eaa88/cities_of_turkey.json'



The optimal solution for random 15 cities : 4206 km

The route:

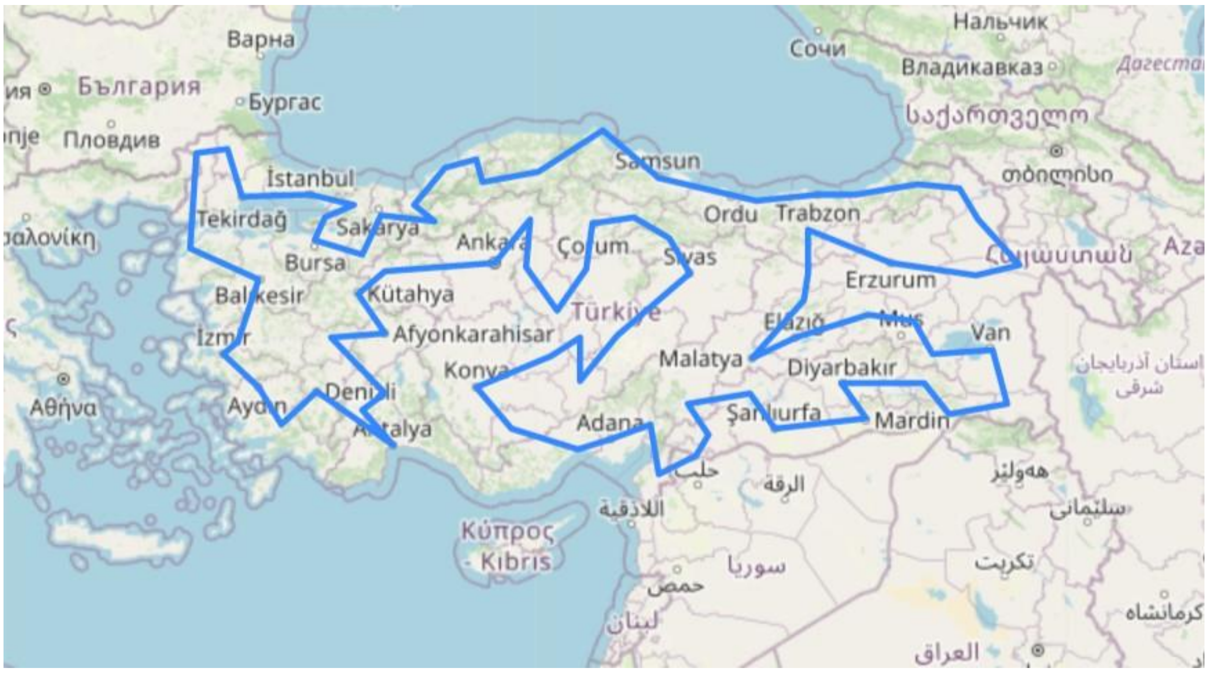
['Bartın','Samsun', 'Amasya', 'Tokat', 'Tunceli', 'Kars','Şırnak','Diyarbakır', 'Şanlıurfa', 'Adıyaman', 'Gaziantep', 'Osmaniye', 'Konya', 'Düzce', 'Zonguldak']

TSP for 81 cities:

In the second part of the assignment, the TSP algorithm applied 81 cities.The number of pairing (i,j) for 81 cities is 3240 which is a combination(81,2).

The road distances between 81 cities in Turkey is given. To illustrate 81 cities' routes on the map, 81 cities’ coordination is needed. The below link contains 81 cities' coordination:

https://gist.githubusercontent.com/ozdemirburak/4821a26db048cc0972c1beee48a408de/raw/4754e5f9d09dade2e6c461d7e960e13ef38eaa88/cities_of_turkey.json'



The optimal solution for random 81 cities : 9938 km

The route:

['Adana','Mersin', 'Karaman', 'Konya', 'Aksaray', 'Nevşehir', 'Niğde', 'Kayseri', 'Sivas', 'Tokat', 'Amasya', 'Çorum', 'Yozgat','Kırşehir', 'Kırıkkale', 'Çankırı', 'Ankara', 'Eskişehir', 'Kütahya','Afyonkarahisar','Uşak','Isparta', 'Burdur','Antalya', 'Denizli', 'Muğla', 'Aydın', 'İzmir', 'Manisa', 'Balıkesir', 'Çanakkale', 'Edirne', 'Kırklareli', 'Tekirdağ', 'İstanbul', 'Kocaeli', 'Yalova', 'Bursa', 'Bilecik', 'Sakarya', 'Bolu', 'Düzce', 'Zonguldak', 'Bartın', 'Karabük', 'Kastamonu', 'Sinop', 'Samsun', 'Ordu', 'Giresun', 'Trabzon', 'Rize', 'Artvin', 'Ardahan', 'Kars', 'Iğdır', 'Ağrı', 'Erzurum', 'Bayburt', 'Gümüşhane', 'Erzincan', 'Tunceli', 'Malatya', 'Elâzığ', 'Bingöl', 'Muş', 'Bitlis', 'Van', 'Hakkâri', 'Şırnak', 'Siirt', 'Batman', 'Diyarbakır', 'Mardin', 'Şanlıurfa', 'Adıyaman', 'Kahramanmaraş', 'Gaziantep', 'Kilis', 'Hatay', 'Osmaniye']

Conclusion

The Traveling Salesman Problem (TSP) is the most popular NP-hard optimization problem. In this model, we showed how to formulate the symmetric Traveling Salesman Problem as a MIP problem. Thanks to using lazy constraints, we also showed how to dynamically eliminate subtours.

References

- Comparison of the Sub-Tour Elimination Methods for the Asymmetric Traveling Salesman Problem Applying the SECA Method, Ramin Bazrafshan, Sarfaraz Hashemkhani Zolfani , and S. Mohammad J. Mirzapour Al-e-hashem
- https://colab.research.google.com/github/Gurobi/modeling-examples/blob/master/traveling_salesman/tsp_gcl.ipynb#scrollTo=78csMpPYxjtu
- https://gist.githubusercontent.com/ozdemirburak/4821a26db048cc0972c1beee48a408de/raw/4754e5f9d09dade2e6c461d7e960e13ef38eaa88/cities_of_turkey.json'