INDR 371 / Homework No: 2

Sunoy Sanyal

November 11, 2024

Problem 1

Sets

$$
\begin{aligned}
F &: \text{Set of candidate locations for the new facilities} \\
\bar{F} &: \text{Set of open plants} \\
K &: \text{Set of customers} \\
K(f) &: \text{The set of customers a facility } f \in F \cup \bar{F} \text{ can serve}
\end{aligned}
$$

Parameters

$$
\begin{aligned}
q_f &: \text{The production capacity of a facility } f \in F \cup \bar{F} \\
d_k &: \text{The demand amount of a customer } k \in K \\
c &: \text{Per unit transportation cost per unit distance} \\
t_{fk} &: \text{the distance between a facility } f \in F \cup \bar{F} \text{ and a customer } k \in K
\end{aligned}
$$

Decision Variables

$$
x_f = \begin{cases} 1 & \text{if facility at location } f \in F \cup \bar{F} \text{ is built} \\ 0 & \text{otherwise} \end{cases}
$$

y_{fk} :Amount of product sent from facility $f \in F \cup \bar{F}$ to demand point $k \in K(f)$

Objective and Constraints

$$
\begin{aligned}
\text{minimize:} \quad & \sum_{k \in K} \sum_{f \in F \cup \bar{F}} c\, t_{fk}\, y_{fk} && (1)\\
\text{s.t.} \quad & \sum_{f \in F} x_f \leq 2 && (2)\\
& \sum_{f \in F \cup \bar{F}} y_{fk} \geq d_k \qquad \forall k \in K && (3)\\
& \sum_{k \in K} y_{fk} \leq q_f x_f \qquad \forall f \in F \cup \bar{F} && (4)\\
& y_{fk} \geq 0,\, x_f = \{1, 0\} \qquad \forall k \in K, \forall f \in F \cup \bar{F} && (5)
\end{aligned}
$$

The maximum number of new plants is 2, which can be seen added as a constraint in equation number (2) above. In equation number (3) above, for all customers the total amount of product received is greater than the demand. In equation number (4) above, production capacity for each facility is given.

Problem 2

(a) The given solution method during the lecture has been implemented in python by using ChatGPT's help. The python function used can be seen below.

```
def find_hub(I, f, u, d, flows, hubcosts):
    min_cost = math.inf
    hub = None

    f = np.array(f)
    u = np.array(u)
    d = np.array(d)
    flows = np.array(flows)
    hubcosts = np.array(hubcosts)
    costs = []
    for i in I:
        inflow_cost = 0
        outflow_cost = 0
        # Fixed hub cost
        fixed_hub_cost = hubcosts[i]

        # Calculate inflow_cost
        for j in I:
            #print(i,j)
            if j != i and flows[j][i] > 0:

                inflow_cost += f[j][i] + u[j][i] * d[j][i] * flows[j][i]

        # Calculate outflow_cost
        for j in I:
```

```
            if j != i and flows[i][j] > 0:
                outflow_cost += f[i][j] + u[i][j] * d[i][j] * flows[i][j]


        # Total cost for hub i
        c_i = inflow_cost + outflow_cost + fixed_hub_cost
        costs.append(c_i)
        if c_i <= min_cost:
            min_cost = c_i
            hub = i

    return hub+1, min_cost, costs
```

The optimal solution is given as $x^\star = 38$, identifying "Kayseri" as seen in Figure 1 showing the screen shot of the code.

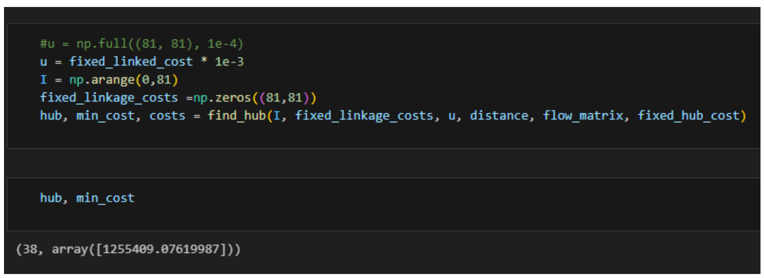


Figure 1: Screen shot of the code output

The code can also be found in the submission folder as "HW2_q2a.ipynb".

(b) For the given problem, the solution method during the lecture has been implemented in python by using ChatGPT's help. The python function used can be seen below.

```
def find_hubs(I, f, u, d, flows, hubcosts, r = 1000, cutoff = 69, alpha = 0.5, verbose_y = False, write_down = False, time = 1):

        """
        I: set of hubs
        f: fixed operating cost for open routes equal to 0
        u: unit cost per km per kg
        d: distances
        flows: required flow from i to j
        hubcosts: costs to open hubs
        r: connection cutoff range forhigh-connection
        cutoff: number of connections needed to be a candidate
        alpha: interhub transfer discount
        """
        hubs = []


        #fixed_hub_cost = {...}  # Dictionary of fixed costs
```

```
    u = np.array(u)
    f = np.array(flows)

    # Step 1: Select candidate hubs (e.g., lowest fixed costs)
    H = select_active_hubs(d, r, cutoff) #defining active nodes for hub locations
    #H = (44,25,39,43,11)
    n = len(H)

    print("Number of hubs",n)
    print("Number of cities", I.shape[0])
    # Step 2: Define Model
    model = gp.Model("Hub Location")

    # Decision variables
    x = model.addVars([(k) for k in H], vtype=GRB.BINARY, name="x_k")  # Hub indicator for candidates
    y_ijkl = model.addVars([(i, j, k, l) for i in I for j in I for k in H for l in H if i != j], vtype=GRB.CONTINUOUS, name="y", lb=0)  # Flow variables

    u_ijkl = {(i, j, k, l): d[i, k] + alpha * d[k, l] + d[l, j]
      for i in I for j in I for k in H for l in H if i != j}
    # Objective: Minimize total transportation and hub setup costs

    model.setObjective(
       gp.quicksum(u_ijkl[i, j, k, l] * f[i, j] * y_ijkl[i, j, k, l]
            for i in I for j in I for k in H for l in H if i != j),
            GRB.MINIMIZE
            )

    # Constraint: Total budget for hubs
    model.addConstr(gp.quicksum(x[i] * hubcosts[i] for i in H) <= 2400, name="hub_total_cost")

    # Flow sum constraints: sum over all hub pairs for each (i, j) should equal 1
    model.addConstrs(
                (gp.quicksum(y_ijkl[i, j, k, l] for k in H for l in H) == 1
                for i in I for j in I if i != j),
                name="connectedness"
                )

    # y less than x_k
    model.addConstrs((y_ijkl[i, j, k, l] <= x[k] for i in I for j in I for k in H for l in H if i != j), name="hub_upper")

    # y less than x_l
    model.addConstrs((y_ijkl[i, j, k, l] <= x[l] for i in
```

```
    I for j in I for k in H for l in H if i != j),
    name="hub_lower")

        # Display the model's details in the console
        #print(model)

        # Set model solve time limit
        model.setParam('TimeLimit', time*60)

        # Display the problem in LP format
        if write_down:
                model.write("heuristic_hub_location_model.lp")
                print("\n MODEL IS WRITTEN \n")
        else:
                print("\n Model is not stored \n")


        reduced_model = model.presolve()
        reduced_model.write("reduced_model.mps")
        return
```

This function aims to convert the problem into a model using MILP formulation. The model is written and solved using GUROBIPY library with academic license. At the end of this function the presolve subfunction is used. This is useful if termination during optimization occurs. In the next function seen below the model is solved.

```
def Solver(I, f, u, d, flows, hubcosts, r = 1000, cutoff = 69, alpha = 0.5, verbose_y = False, write_down = False, time = 1):
    hubs = []
    model = gp.read("reduced_model.mps")

    # Set model solve time limit
    model.setParam('TimeLimit', time*60)
     # Solve model
    model.optimize()

    # Display Results
    for v in model.getVars():
            if v.varName.startswith("x") and v.x > 0.5:
                    print(f"Hub selected at: {v.varName}")
                    index = int(v.varName.split("[")[1].strip("]"))
                    hubs.append(index)  # Collecting the index only
            elif v.varName.startswith("y") and v.x > 0.1 and verbose_y == 1:
                    print(f"y: {v}")

    for i in range(model.SolCount):
            model.Params.SolutionNumber = i
            model.write(f"{i}.sol")


    return hubs, model.objVal
```

Although this method is exact, the MIP solution has too many branches with large LP iterations because of the y_{ijkl} bounds named as "hub_ upper". This specific bound also creates a memory issue as the number of constraints is given by $I^2 \cdot H^2$ where I is the set of cities and H is the set of hub candidates.

Since the solution time and memory increases drastically as the size of H increases for practical purposes a size of 16 was used and solution time was limited to 1 hour. The following method of selecting *highly connected* nodes as candidate locations was used. In retrospect, this method should be used for sub areas or domain restrictions could be added such as: only 1 hub can exist in the following list of cities. But as geographical location information is not present this method was not used. It should also be mentioned that as this method introduces upper bounds for the binary variables cutting-planes would form which has been shown to increase the performance in this kind of problems.

An algorithm which can be seen in "HW2_q2b.ipynb" has been developed to prune edges with long distances, this value was chosen as 1000 as the resulting distribution of cities with degrees varied as seen in Figure 2 below.

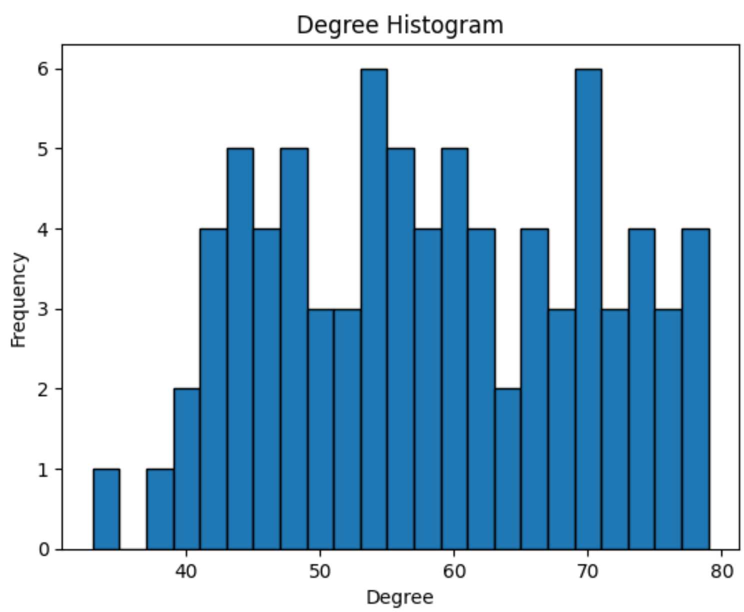
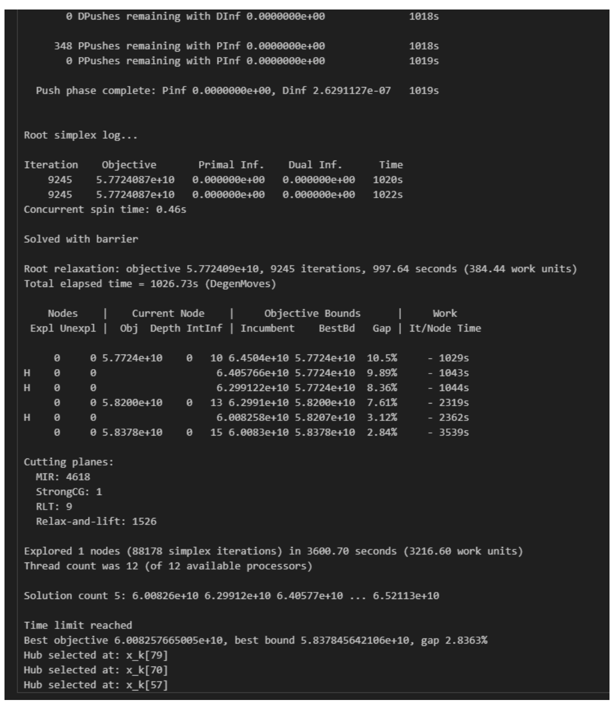


Figure 2: Histogram of city degrees, bin size: 2

It should be mentioned that a sensitivity analysis has not been done on this parameter. But as the next step selects an N number of highest degree nodes this selected set was seen to change rarely as the range parameter changed. This aimed to select a set of candidates which aren't on the edges of th network. When N was selected as a high number (~ 60), the set contained mostly of cities which aren't on Turkey's border. When N was selected as a low number (~ 10), the set contained mostly cities in the Central Anatolia Region which was a major drawback. Again, certain bounds such as maximum regional hub number or minimum hub distances would drastically improve the computation. But such heuristic approaches for specific boundaries is out of the scope of this work. By using the above-mentioned functions and method the problem was solved for 81 cities and 16 hubs. The optimal solution which can be seen below in the screen shot (plate codes are shown as 1 less) or the code folder is $[80, 71, 58]$ correlating to cities: Osmaniye, Kırıkkale, and Sivas. Although this solution creates a triangle and looks good the maximum cost boundary is very loose and a low cost city can be added which always creates a better solution. Possible reasons for Gurobi not finding a simply better solution is due to the bad selection of hub candidates, missing heuristic information about the graph, and low runtime.



```
      0 DPushes remaining with DInf 0.0000000e+00              1018s

    348 PPushes remaining with PInf 0.0000000e+00              1018s
      0 PPushes remaining with PInf 0.0000000e+00              1019s

  Push phase complete: Pinf 0.0000000e+00, Dinf 2.6291127e-07   1019s


Root simplex log...

Iteration    Objective       Primal Inf.    Dual Inf.      Time
    9245    5.7724087e+10   0.000000e+00   0.000000e+00   1020s
    9245    5.7724087e+10   0.000000e+00   0.000000e+00   1022s
Concurrent spin time: 0.46s

Solved with barrier

Root relaxation: objective 5.772409e+10, 9245 iterations, 997.64 seconds (384.44 work units)
Total elapsed time = 1026.73s (DegenMoves)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 5.7724e+10    0   10 6.4504e+10 5.7724e+10  10.5%     - 1029s
H    0     0                    6.405766e+10 5.7724e+10  9.89%     - 1043s
H    0     0                    6.299122e+10 5.7724e+10  8.36%     - 1044s
     0     0 5.8200e+10    0   13 6.2991e+10 5.8200e+10  7.61%     - 2319s
H    0     0                    6.008258e+10 5.8207e+10  3.12%     - 2362s
     0     0 5.8378e+10    0   15 6.0083e+10 5.8378e+10  2.84%     - 3539s

Cutting planes:
  MIR: 4618
  StrongCG: 1
  RLT: 9
  Relax-and-lift: 1526

Explored 1 nodes (88178 simplex iterations) in 3600.70 seconds (3216.60 work units)
Thread count was 12 (of 12 available processors)

Solution count 5: 6.00826e+10 6.29912e+10 6.40577e+10 ... 6.52113e+10

Time limit reached
Best objective 6.008257665005e+10, best bound 5.837845642106e+10, gap 2.8363%
Hub selected at: x_k[79]
Hub selected at: x_k[70]
Hub selected at: x_k[57]
```

Figure 3: Output of feasible solution

With ample solution time and a larger number of candidate hub locations a much better or even optimal solution can be found.