Task list:

1. **Multi-Skilled Traveling Salesman Problem" (MSTSP). Check my mSkilledmTSP.py and plot_routes.py scripts. I used Anaconda -> Spyder for coding them.**
    a. **Check Multi-Skilled Traveling Salesman Problem formulation.**
    b. **Import distances and skills from spreadsheets. Will provide the spreadsheets later.**

    ```
    # Generate random distance matrix (symmetric with zeros on the diagonal)
    np.random.seed(42)
    distances = np.random.randint(1, 20, size=(n, n))
    np.fill_diagonal(distances, 0)
    distances = (distances + distances.T) // 2  # Make it symmetric

    # Skill constraints: which salesmen can visit which cities (excluding depot)
    skills = {
      0: [1, 2, 3],  # Salesman 0 can visit cities 1, 2, 3
      1: [3, 4],     # Salesman 1 can visit cities 3, 4
    }
    ```

    c. **Not sure about the best way to generate coordinates for visualizing the routes. Fig.1 below is from my script and Fig. 2 from literature.**

    ```
    coordinates = {
        0: (0, 0),  # Depot
        1: (2, 1),
        2: (1, 3),
        3: (3, 2),
        4: (4, 4),
      }
    ```

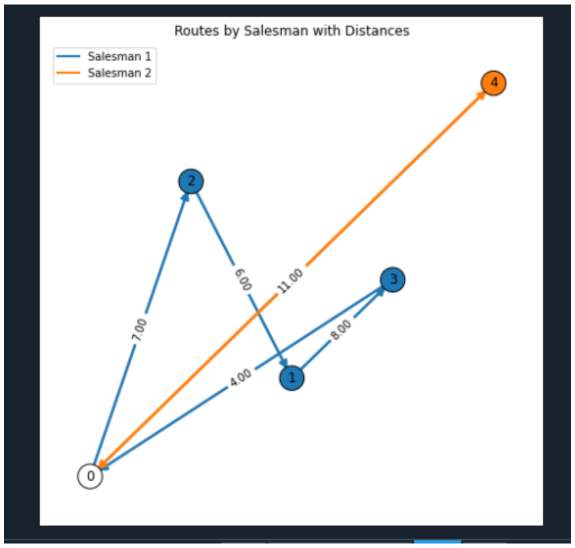


Fig. 1

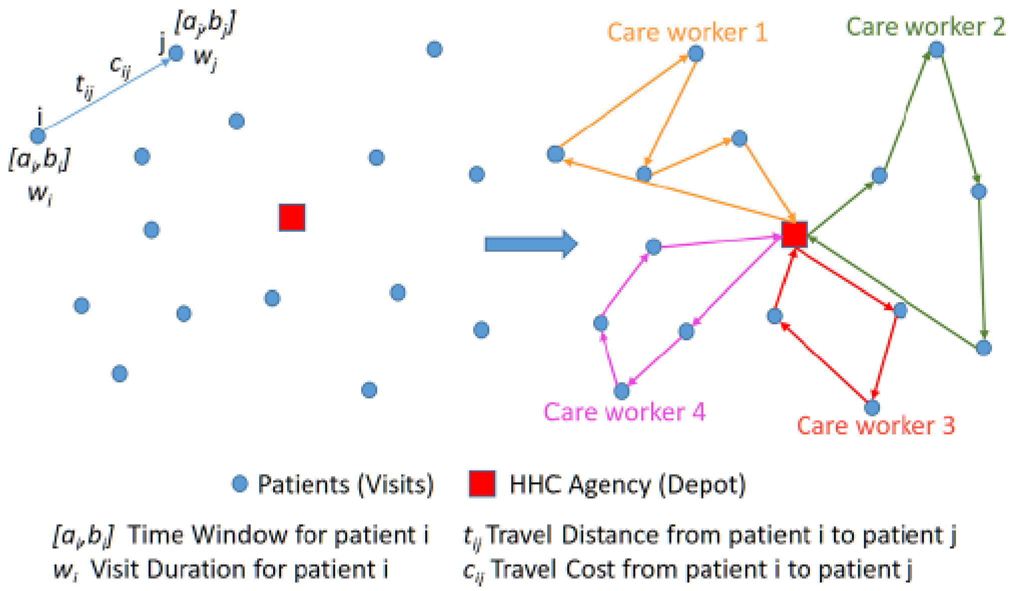

$[a_j,b_j]$
j
w_j
c_{ij}
t_{ij}
i
$[a_i,b_i]$
w_i
Care worker 1
Care worker 2
Care worker 4
Care worker 3
● Patients (Visits)
🟥 HHC Agency (Depot)
$[a_i,b_i]$ Time Window for patient i
w_i Visit Duration for patient i
t_{ij} Travel Distance from patient i to patient j
c_{ij} Travel Cost from patient i to patient j

Fig. 2

2. **Extend mSkilledmTSP.py and plot_routes.py scripts to Multi-Skilled *Multi-Depot* Traveling Salesman Problem" (MSMDTSP). Workload estimate?**

3. **PyEPO Modification Upwork.py is for running the original PyEPO lib. It compares the results from 2S (predict and then optimize) and SPO (smart predict and optimize). The former uses prediction to train a neural network while the latter decision error.**

a. **optmodel = pyepo.model.grb.tspMTZModel(num_node) – line 49. This calls PyEPO-main\pkg\pyepo\model\grb. Single TSP.py.**

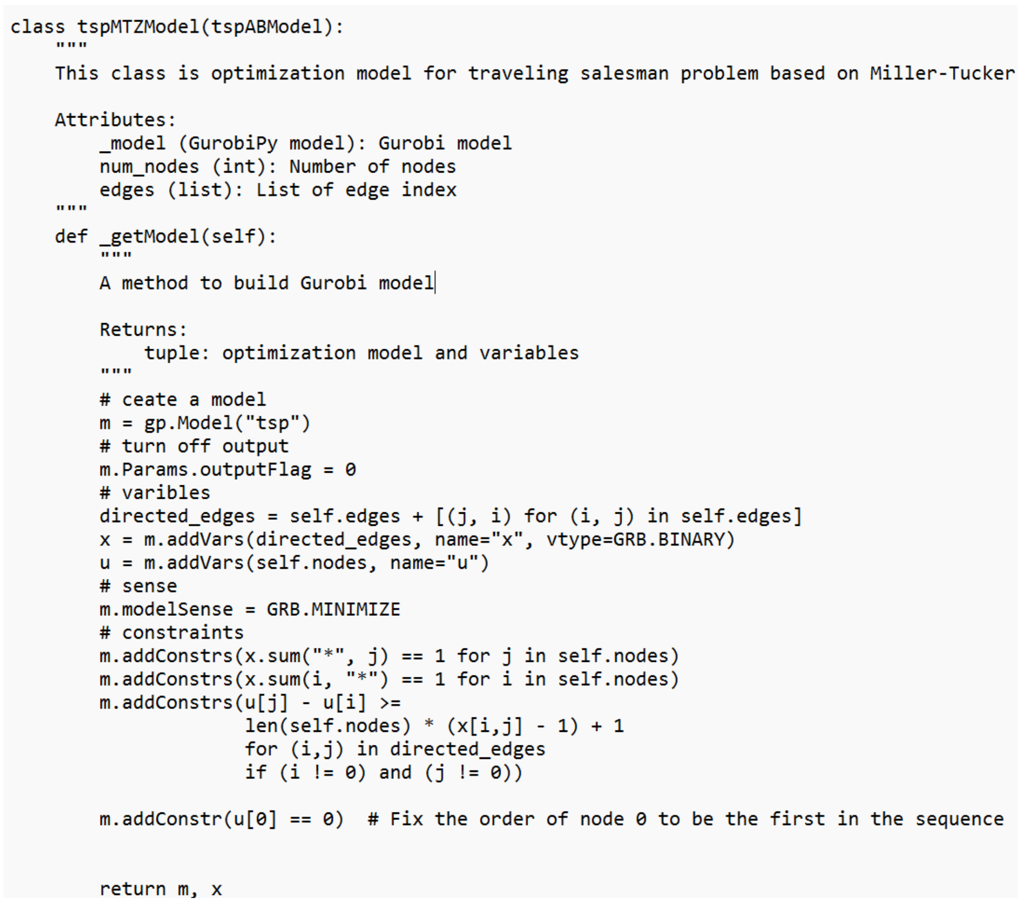
   Build a new class within grb.Single TSP.py based on mSkilledmTSP.py. So, I can build optmodel as follows.

   optmodel = pyepo.model.grb.MStspMTZModel(num_node)

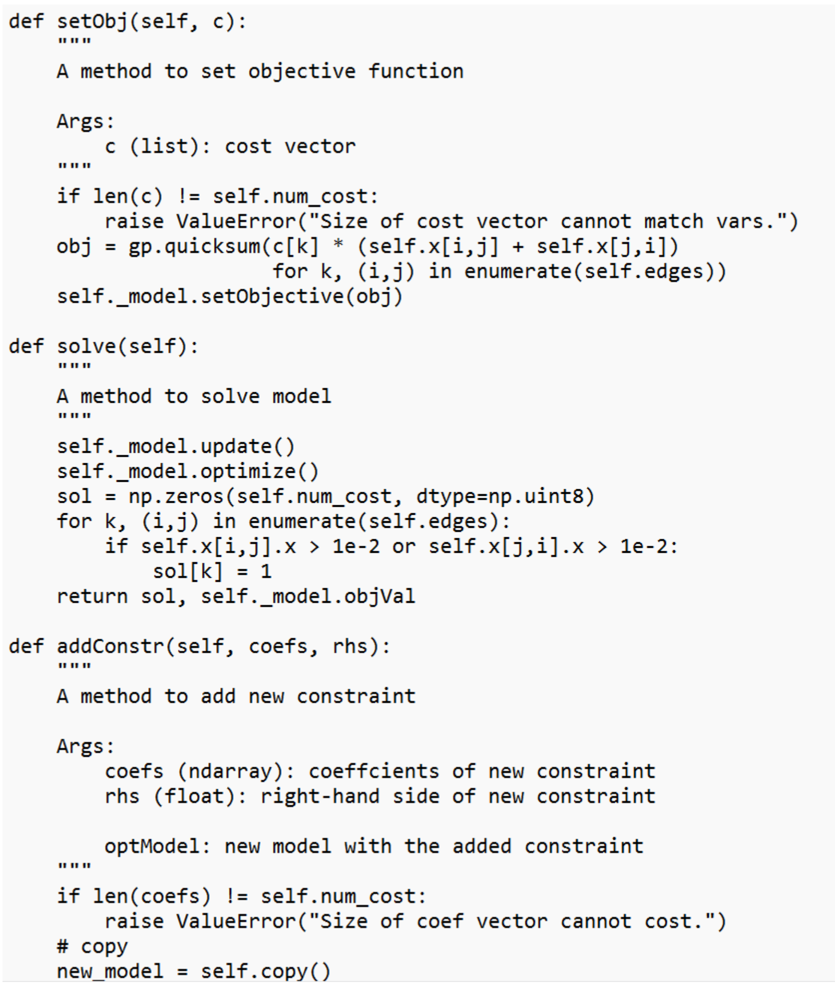
   # optmodel.setObj(c[0]) # set objective function

   # sol, obj = optmodel.solve()

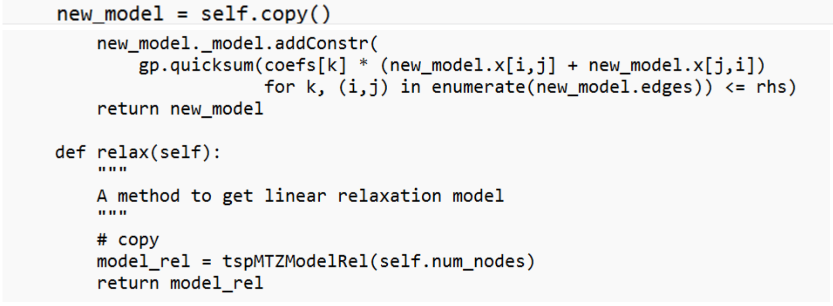
   Current single travelling salesman class:



```
class tspMTZModel(tspABModel):
    """
    This class is optimization model for traveling salesman problem based on Miller-Tucker

    Attributes:
        _model (GurobiPy model): Gurobi model
        num_nodes (int): Number of nodes
        edges (list): List of edge index
    """
    def _getModel(self):
        """
        A method to build Gurobi model

        Returns:
            tuple: optimization model and variables
        """
        # ceate a model
        m = gp.Model("tsp")
        # turn off output
        m.Params.outputFlag = 0
        # varibles
        directed_edges = self.edges + [(j, i) for (i, j) in self.edges]
        x = m.addVars(directed_edges, name="x", vtype=GRB.BINARY)
        u = m.addVars(self.nodes, name="u")
        # sense
        m.modelSense = GRB.MINIMIZE
        # constraints
        m.addConstrs(x.sum("*", j) == 1 for j in self.nodes)
        m.addConstrs(x.sum(i, "*") == 1 for i in self.nodes)
        m.addConstrs(u[j] - u[i] >=
                     len(self.nodes) * (x[i,j] - 1) + 1
                     for (i,j) in directed_edges
                     if (i != 0) and (j != 0))

        m.addConstr(u[0] == 0)  # Fix the order of node 0 to be the first in the sequence


        return m, x
```



```
def setObj(self, c):
    """
    A method to set objective function

    Args:
        c (list): cost vector
    """
    if len(c) != self.num_cost:
        raise ValueError("Size of cost vector cannot match vars.")
    obj = gp.quicksum(c[k] * (self.x[i,j] + self.x[j,i])
                      for k, (i,j) in enumerate(self.edges))
    self._model.setObjective(obj)

def solve(self):
    """
    A method to solve model
    """
    self._model.update()
    self._model.optimize()
    sol = np.zeros(self.num_cost, dtype=np.uint8)
    for k, (i,j) in enumerate(self.edges):
        if self.x[i,j].x > 1e-2 or self.x[j,i].x > 1e-2:
            sol[k] = 1
    return sol, self._model.objVal

def addConstr(self, coefs, rhs):
    """
    A method to add new constraint

    Args:
        coefs (ndarray): coeffcients of new constraint
        rhs (float): right-hand side of new constraint

        optModel: new model with the added constraint
    """
    if len(coefs) != self.num_cost:
        raise ValueError("Size of coef vector cannot cost.")
    # copy
    new_model = self.copy()
```



```
        new_model._model.addConstr(
            gp.quicksum(coefs[k] * (new_model.x[i,j] + new_model.x[j,i])
                        for k, (i,j) in enumerate(new_model.edges)) <= rhs)
        return new_model

    def relax(self):
        """
        A method to get linear relaxation model
        """
        # copy
        model_rel = tspMTZModelRel(self.num_nodes)
        return model_rel
```

b. **Note that in our MSTSP case, predicted c is travel speed, not travel time. So, the travel time is dist/c.**

c. **optmodel appears in multiple places in the script. Should be all good if MStspMTZModel follows the same data structure used in tspMTZModel.**

d. **Reduce computational burden; training by solving LP relaxation; validation by solving the true binary integer model.**

   e. **How to handle plot_routes.py? Toggle on/off?**

4. **General questions**
   a. **The best way to call a revised package.**
   b. **The best way to set up github with revised lib.**