1 Index Use

$$i \in I = 1, 2, ..., n \text{ aircrafts}$$
$$m \in 1, 2, ..., n \text{ mission m}$$
$$a \in 1, ..., q \text{ node index}$$
$$b \in 1, ..., q \text{ node index}$$

2 Given Data

$$v_i \in \mathbb{Z}^n \text{ value of selecting aircraft i}$$
$$W \in \mathbb{N} \text{ budget}$$
$$variableCost_i \text{ cost per km for flying aircraft i}$$
$$q \text{ number of targets (nodes)}$$

3 Decision Variables

$$x_i = \begin{cases} 1 \text{ if aircraft i is chosen,} \\ 0 \text{ otherwise} \end{cases}$$
$$p_{a,b,i} \in \mathbb{R}^{q \times q \times n}$$
$$\text{where } p_{a,b,i} = \begin{cases} 1 \text{ if path from node a to b is taken for aircraft i ,} \\ 0 \text{ otherwise} \end{cases}$$

4 Formulation

$$maximize \sum_{i=1}^{n} v_i x_i \quad (1)$$
$$\textbf{s.t.} \ \sum_{i=1}^{n} w_i x_i \leq W \quad (2)$$
where w_i is defined by:

4.1 Subproblem

$$w_i = variableCost_i \cdot min \sum_{a=1}^{q} \sum_{b \neq a, b=1}^{q} c_{a,b} p_{a,b,i} \quad (3)$$
$$\textbf{s.t.} \ u_a \in \mathbb{Z} = 2, ...q \quad (4)$$
$$\sum_{a=1, a \neq b}^{q} p_{a,b} = 1 \ \forall b \quad (5)$$
$$\sum_{b=1, b \neq a}^{q} p_{a,b} = 1 \ \forall a \quad (6)$$
$$u_a - u_b + q p_{a,b,i} \leq q - 1 \ \ \forall i \text{ s.t. } 2 \leq a \neq b \leq q \quad (7)$$
$$1 \leq u_a \leq q - 1 \quad (8)$$

5 Discussion

This is a hybrid problem which selects a subset of aircraft for an airbase. Each for each aircraft i at the airbase, there are is a "mission" which is represented by a graph; nodes realistically represent targets. In addition, each aircraft i has an associated US dollar cost per kilometer flown, $variableCost_i$, and a value for completing a mission, v_i.

Objective (1) and constraint (2) define a binary knapsack for selecting aircraft to maximize mission value. However, to calculate the cost of selecting an aircraft, a shortest path traveling salesman subproblem is solved and the distance traveled is multiplied by the variable cost for that aircraft. In the subproblem, u_a in constraint (2) is a dummy variable use to represent tour ordering. Constraint (7) ensures that each target is arrived at from only one other target. Constraint (8) ensures that each target has a departure to exactly one other target.

5.1 Scalability

This solution is not likely to scale well. The knapsack master problem is $O(nW)$ and the TSP subproblem is $O(n^3 * 2^n)$. This problem might start becoming infeasible for home computers at around n = 75.

5.2 Complexity

This was fairly complex to compute. My initial idea was to have the TSP be a multiple TSP where multiple aircraft would work together to solve the TSP using particle swarm optimization. Given the time constraints, I chose not to go this path.

5.3 Sensitivity

There is some buffer room here. Small changes aircraft value relative to each other are unlikely to have significant impact on the final solution. However, small changes in the variable cost could have huge impact.

5.4 Convergence

My solution is guaranteed to be optimal given the choice of DP and MIP.

6 Pseudocode

```
import numpy as np
import pyomo.environ as pyEnv

    # First, lets create a function that solves the DP 0/1 knapsack problem
    def dp_ks(W, cost_list, val):
        # W = budget
        # cost_list is the cost per km of each aircraft
        # val is an array of values

        # we will implement this subproblem afterwards to get the weights
        # of each aircraft
        wt = get_tsp_costs_for_each_aircraft(cost_list)

        n = len(val)

        #initialize this 2d array to all zeros
        table = array[W+1]{n+1]

        for i in range(n + 1):
```

```
                for j in range(W + 1):
                    if i == 0 or j == 0:
                        # set first row and col to zero
                        table[i][j] = 0
                        elif wt[i-1] <= j: # make sure we do not exceed budget
                            # if choosing the aircraft increases value, add it
                            table[i][j] = max(val[i-1] + table[i-1][j-wt[i-1]],
table[i-1][j])
                        else:
                            # else update the table to its previous value
                            table[i][j] = table[i-1][j]

            #residual table for printing selected items in knapsack
            res = table[n][W]

            for i in range(n):
                if res <= 0:
                        break;

                #if item comes from val[i-1] + table[i-1][w-wt[i-1]]
                #then it will be included
                if res == table[i - 1][w]:
                    continue
                else:
                    # item i-1 is included
                    print(item_names[i-1])
                    # deduct value since weight is included
                    res = res - val[i - 1]
                        w = w - wt[i - 1]

            # trivially generates random matrices with zero on the diagonal
            # to represent target graphs for each of the n aircraft
            def generate_n_distance_matrices(n, num_nodes):
                return list_of_n_matrices

            def get_tsp_costs_for_each_aircraft(ac_costs_per_km):
                """
                returns a sum of km traveled for the shortest TSP path for each
                aircraft-specific mission
                for this function we are going to use the pyomo
                package with the CPLEX MIP solver
                """

                cost_matrices= generate_n_distance_matrices(n)
                distances_foreach_mission = []

                # calculate TSP for each aircraft
                for cost_matrix in cost_matrices:
                    n = len(cost_matrix)

                    model = pyEnv.ConcreteModel()

                    #Indexes for the targets
                    model.M = pyEnv.RangeSet(n)
```

```
model.N = pyEnv.RangeSet(n)

#dummy variable U index
model.U = pyEnv.RangeSet(2,n)
#Decision variables xij
model.x=pyEnv.Var(model.N,model.M, within=pyEnv.Binary)

#Dummy variable ui
model.u=pyEnv.Var(model.N, within=pyEnv.NonNegativeIntegers,bounds=

#Cost Matrix cij
model.c = pyEnv.Param(model.N, model.M,initialize=lambda model, i,

# objective function: see line (3) in \textbf{Subproblem}
def obj_func(model):
    return sum(model.x[i,j] * model.c[i,j] for i in model.N for j i

model.objective = pyEnv.Objective(rule=obj_func,sense=pyEnv.minimi


# see subproblem (5) and (6)
def rule_const1(model,M):
    return sum(model.x[i,M] for i in model.N if i!=M ) == 1
model.const1 = pyEnv.Constraint(model.M,rule=rule_const1)
def rule_const2(model,N):
    return sum(model.x[N,j] for j in model.M if j!=N) == 1
model.rest2 = pyEnv.Constraint(model.N,rule=rule_const2)

# see subproblem (8)
def rule_const3(model,i,j):
    return model.u[i] - model.u[j] + model.x[i,j] * n <= n-1
model.rest3 = pyEnv.Constraint(model.U,model.N,rule=rule_const3)

solver = pyEnv.SolverFactory('cplex')

distances = []
#get edges_selected from model
for i,j in edges_selected:
    distances.append(cost_matrix[i][j])

# this will be a list of arrays
distances_foreach_mission.append(distances)

# length of shortest path for each aircraft
tsp_costs_per_ac = []


for i in range(len(distances_foreach_mission)):
    dot_product = dot_product(ac_costs_per_km[i],
                              distances_foreach_mission[i])
    tsp_costs_per_ac.append(dot_product)

# the total cost for each aircraft
sums = [np.sum(arr) for arr in tsp_costs_per_ac]
```

```
return sums
```