Homework #3 Jae Yul Woo (7466887196)

1 (a) Formulation of an LP which maximizes revenue corresponding to deterministic demands and identify the appropriate dual prices to use for comparisons with bids for each leg.

<Formulation with Pyomo>

```
from pyutilib.services import register_executable, registered_executable
register_executable(name='glpk')
import numpy as np
from pyomo.environ import *
import pyomo.environ as pyo
from pyomo.opt import SolverFactory

model = ConcreteModel()
opt = SolverFactory('glpk')
X = ['x1','x2','x3','x4','x5']
model.x = Var(X,within=NonNegativeIntegers)

v = {1:280,2:300,3:190,4:220,5:140}
c = [[0,1,1,0,0,1],
     [1,0,1,1,0,0],
     [0,1,1,0,0,1],
     [1,0,1,1,0,0],
     [0,0,1,0,1,0]]
capa = {1:100,2:100,3:300,4:100,5:100,6:100}; d_mean = {1:50,2:30,3:200,4:100,5:160}

# Objective
model.obj = Objective(expr= model.x['x1']*v[1]+model.x['x2']*v[2]+model.x['x3']*v[3
+model.x['x4']*v[4]+model.x['x5']*v[5],sense=maximize)

# Capacity Constraint
c1 =np.transpose(c)[0]
model.c1 = Constraint(expr= model.x['x1']*c1[0]+model.x['x2']*c1[1]+model.x['x3']*c
+model.x['x4']*c1[3]+model.x['x5']*c1[4] <= capa[1])

c2 =np.transpose(c)[1]
model.c2 = Constraint(expr= model.x['x1']*c2[0]+model.x['x2']*c2[1]+model.x['x3']*c
+model.x['x4']*c2[3]+model.x['x5']*c2[4] <= capa[2])

c3 =np.transpose(c)[2]
model.c3 = Constraint(expr=
model.x['x1']*c3[0]+model.x['x2']*c3[1]+model.x['x3']*c+model.x['x4']*c3[3]+model.x['x5']*c3[4] <= capa[3])

c4 =np.transpose(c)[3]
model.c4 = Constraint(expr=
model.x['x1']*c4[0]+model.x['x2']*c4[1]+model.x['x3']*c+model.x['x4']*c4[3]+model.x['x5']*c4[4] <= capa[4])

c5 =np.transpose(c)[4]
model.c5 = Constraint(expr=
model.x['x1']*c5[0]+model.x['x2']*c5[1]+model.x['x3']*c+model.x['x4']*c5[3]+model.x['x5']*c5[4] <= capa[5])

c6 =np.transpose(c)[5]
model.c6 = Constraint(expr=
model.x['x1']*c6[0]+model.x['x2']*c6[1]+model.x['x3']*c+model.x['x4']*c6[3]+model.x['x5']*c6[4] <= capa[6])
# Remaining Demand Constraint (when D is average D)
upper = {'x1':50, 'x2':30, 'x3':200, 'x4':100, 'x5':160}
model.d1 = Constraint(expr = model.x['x1'] <= upper['x1'])
model.d2 = Constraint(expr = model.x['x2'] <= upper['x2'])
```

```
model.d3 = Constraint(expr = model.x['x3'] <= upper['x3'])
model.d4 = Constraint(expr = model.x['x4'] <= upper['x4'])
model.d5 = Constraint(expr = model.x['x5'] <= upper['x5'])
model.dual = pyo.Suffix(direction=pyo.Suffix.IMPORT)
# model.pprint()

results = opt.solve(model)
model.display()
```

Output

```
Variables:
  x : Size=5, Index=x_index
    Key : Lower : Value : Upper : Fixed : Stale : Domain
     x1 :     0 :  50.0 :  None : False : False : NonNegativeIntegers
     x2 :     0 :  30.0 :  None : False : False : NonNegativeIntegers
     x3 :     0 :  50.0 :  None : False : False : NonNegativeIntegers
     x4 :     0 :  70.0 :  None : False : False : NonNegativeIntegers
     x5 :     0 : 100.0 :  None : False : False :
NonNegativeIntegers

 Objectives:
  obj : Size=1, Index=None, Active=True
    Key  : Active : Value
    None :   True : 61900.0

 Constraints:
  c1 : Size=1
    Key  : Lower : Body  : Upper
    None :  None : 100.0 : 100.0
  c2 : Size=1
    Key  : Lower : Body  : Upper
    None :  None : 100.0 : 100.0
  c3 : Size=1
    Key  : Lower : Body  : Upper
    None :  None : 300.0 : 300.0
  c4 : Size=1
    Key  : Lower : Body  : Upper
    None :  None : 100.0 : 100.0
  c5 : Size=1
    Key  : Lower : Body  : Upper
    None :  None : 100.0 : 100.0
  c6 : Size=1
    Key  : Lower : Body  : Upper
    None :  None : 100.0 : 100.0
  d1 : Size=1
    Key  : Lower : Body : Upper
    None :  None : 50.0 :  50.0
  d2 : Size=1
    Key  : Lower : Body : Upper
    None :  None : 30.0 :  30.0
  d3 : Size=1
    Key  : Lower : Body : Upper
    None :  None : 50.0 : 200.0
  d4 : Size=1
    Key  : Lower : Body : Upper
    None :  None : 70.0 : 100.0
  d5 : Size=1
    Key  : Lower : Body  : Upper
    None :  None : 100.0 : 160.0
```

I could get the primal solution for X (the numer of demands to maximize the profit for each itinerary). Itinerary 1 : 50, Itinerary 2 : 30, Itinerary 3 : 50, Itinerary 4 : 70, Itinerary 5 : 100

<Formulation with AMPL>

```
var x1 integer;
var x2 integer;
var x3 integer;
var x4 integer;
var x5 integer;

maximize Profit:
280*x1+300*x2+190*x3+220*x4+140*x5;
subject to c1: x2+x4<=100;
subject to c2: x1+x3<=100;
subject to c3: x1+x2+x3+x4+x5<=300;
subject to c4: x2+x4<=100;
subject to c5: x5<=100;
subject to c6: x1+x3<=100;
subject to d1: x1<=50;
subject to d2: x2<=30;
subject to d3: x3<=200;
subject to d4: x4<=100;
subject to d5: x5<=160;
```

Output

```
ampl: reset;
ampl: model hw3.1.mod;
ampl: solve;
MINOS 5.51: ignoring integrality of 5 variables
MINOS 5.51: optimal solution found.
5 iterations, objective 61900

ampl: display x1,x2,x3,x4,x5;
x1 = 50
x2 = 30
x3 = 50
x4 = 70
x5 = 100
```

```
ampl: display c1,c2,c3,c4,c5,c6;
c1 = 80
c2 = 50
c3 = 140
c4 = 0
c5 = 0
c6 = 0
```

```
ampl: display d1,d2,d3,d4,d5;
d1 = 90
d2 = 80
d3 = 0
d4 = 0
d5 = 0
```

I derived the same primal solution with AMPL solver and the dual price for each leg.

Leg j	Link	Dual Price
1	C - A	80
2	D - A	50
3	A - B	140
4	B - E	0
5	B - F	0
6	B - G	0

1 (b) Suppose you are going to use the dual prices obtained in 1(a) to define a threshold for a passenger traveling from C-A-B-E. What threshold price would you use to accept/reject a bid.

The sum of dual prices for itinerary 2 (C-A-B-E) is 220. Hence, we could get the $220 value from the itinerary 2. The threshold price for accepting/rejecting the bid for the travel C-A-B-E is **$220**.

1 (c) Assume that you are deciding the pricing plan about 60 days before departure. Formulate the PNLP formulation presented in the paper by a linear program. Use the software to calculate the dual multipliers associated each leg, using the PNLP formulation.

I used PNLP for the part (c) and (d). {x_i} is the number of each itinerary we want to maximize our revenue, {d_i} is the real demand observed after the formulation. The objective for this Stochastic Programming maximizing the revenue with respect to realized demand which is Min{x_i, d_i}. The first stage constraint is the capacity constraint for each leg. The second stage constraint is the Min{x_i, d_i} = t_i which is equivalent to another LP:

Max t_i

subject to t_i <= x_i; t_i <= d_i

First stage variables are {x_i} and second stage variables are {t_i} where {d_i} is the stochastic RHS.

The Pyomo code is below.

```
import numpy as np
import pandas as pd
import itertools
from pyomo.environ import *
from pyomo.core import *
from pyomo.pysp.annotations import (PySP_ConstraintStageAnnotation, StochasticConstraintBoundsAnnotation,
                                    StochasticConstraintBodyAnnotation)
```

```
model = ConcreteModel()

v = {1:280,2:300,3:190,4:220,5:140}
c = [[0,1,1,0,0,1],
     [1,0,1,1,0,0],
     [0,1,1,0,0,1],
     [1,0,1,1,0,0],
     [0,0,1,0,1,0]]
capa = {1:100,2:100,3:300,4:100,5:100,6:100}
d_mean = {1:50,2:30,3:200,4:100,5:160}

# Possible Demands
d1_rhs_table = [25,50,75]
d2_rhs_table = [15,30,45]
d3_rhs_table = [100,200,300]
d4_rhs_table = [50,100,150]
d5_rhs_table = [80,160,240]

model.constraint_stage = PySP_ConstraintStageAnnotation()
model.stoch_rhs = StochasticConstraintBoundsAnnotation()
num_scenarios = len(d1_rhs_table) * len(d2_rhs_table) * len(d3_rhs_table) * len(d4_rhs_table) *
len(d5_rhs_table)
scenario_data = dict(('Scenario'+str(i), (d1val, d22val, d3val, d4val, d5val))
                     for i, (d1val, d22val, d3val, d4val, d5val) in
                     enumerate (itertools.product(d1_rhs_table, d2_rhs_table, d3_rhs_table, d4_rhs_table,
d5_rhs_table),1))

# Declare Variables
model.x1 = Var(within=NonNegativeIntegers)
model.x2 = Var(within=NonNegativeIntegers)
model.x3 = Var(within=NonNegativeIntegers)
model.x4 = Var(within=NonNegativeIntegers)
model.x5 = Var(within=NonNegativeIntegers)

model.t1 = Var(within=NonNegativeIntegers)
model.t2 = Var(within=NonNegativeIntegers)
model.t3 = Var(within=NonNegativeIntegers)
model.t4 = Var(within=NonNegativeIntegers)
model.t5 = Var(within=NonNegativeIntegers)

model.d1_rhs = Param(mutable=True, initialize=0.0)
model.d2_rhs = Param(mutable=True, initialize=0.0)
model.d3_rhs = Param(mutable=True, initialize=0.0)
model.d4_rhs = Param(mutable=True, initialize=0.0)
model.d5_rhs = Param(mutable=True, initialize=0.0)

# Objective
model.FirstStageCost = Expression(initialize=0)
model.SecondStageCost = Expression(initialize=model.t1*v[1]+model.t2*v[2]+model.t3*v[3]
                      +model.t4*v[4]+model.t5*v[5])

# Capacity Constraints
c1 =np.transpose(c)[0]
model.c1 = Constraint(expr= model.x1*c1[0]+model.x2*c1[1]+model.x3*c1[2]
                      +model.x4*c1[3]+model.x5*c1[4] <= capa[1])
model.constraint_stage.declare(model.c1,1)
c2 =np.transpose(c)[1]
model.c2 = Constraint(expr= model.x1*c2[0]+model.x2*c2[1]+model.x3*c2[2]
                      +model.x4*c2[3]+model.x5*c2[4] <= capa[2])
```

```
model.constraint_stage.declare(model.c2,1)
c3 =np.transpose(c)[2]
model.c3 = Constraint(expr= model.x1*c3[0]+model.x2*c3[1]+model.x3*c3[2]
                      +model.x4*c3[3]+model.x5*c3[4] <= capa[3])
model.constraint_stage.declare(model.c3,1)
c4 =np.transpose(c)[3]
model.c4 = Constraint(expr= model.x1*c4[0]+model.x2*c4[1]+model.x3*c4[2]
                      +model.x4*c4[3]+model.x5*c4[4] <= capa[4])
model.constraint_stage.declare(model.c4,1)
c5 =np.transpose(c)[4]
model.c5 = Constraint(expr= model.x1*c5[0]+model.x2*c5[1]+model.x3*c5[2]
                      +model.x4*c5[3]+model.x5*c5[4] <= capa[5])
model.constraint_stage.declare(model.c5,1)
c6 =np.transpose(c)[5]
model.c6 = Constraint(expr= model.x1*c6[0]+model.x2*c6[1]+model.x3*c6[2]
                      +model.x4*c6[3]+model.x5*c6[4] <= capa[6])
model.constraint_stage.declare(model.c6,1)

# Second Stage Min X,D Constraint (when D is stochastic)
model.d11 = Constraint(expr=model.t1 <= model.d1_rhs)
model.constraint_stage.declare(model.d11,2)
model.stoch_rhs.declare(model.d11)

model.d12 = Constraint(expr=model.t1 - model.x1 <= 0)
model.constraint_stage.declare(model.d12,2)

model.d21 = Constraint(expr=model.t2 <= model.d2_rhs)
model.constraint_stage.declare(model.d21,2)
model.stoch_rhs.declare(model.d21)

model.d22 = Constraint(expr=model.t2 - model.x2 <= 0)
model.constraint_stage.declare(model.d22,2)

model.d31 = Constraint(expr=model.t3 <= model.d3_rhs)
model.constraint_stage.declare(model.d31,2)
model.stoch_rhs.declare(model.d31)

model.d32 = Constraint(expr=model.t3 - model.x3 <= 0)
model.constraint_stage.declare(model.d32,2)

model.d41 = Constraint(expr=model.t4 <= model.d4_rhs)
model.constraint_stage.declare(model.d41,2)
model.stoch_rhs.declare(model.d41)

model.d42 = Constraint(expr=model.t4 - model.x4 <= 0)
model.constraint_stage.declare(model.d42,2)

model.d51 = Constraint(expr=model.t5 <= model.d5_rhs)
model.constraint_stage.declare(model.d51,2)
model.stoch_rhs.declare(model.d51)

model.d52 = Constraint(expr=model.t5 - model.x5 <= 0)
model.constraint_stage.declare(model.d52,2)

# Final Objective
model.obj = Objective(expr=model.FirstStageCost+model.SecondStageCost, sense=maximize)

def pysp_scenario_tree_model_callback():
    from pyomo.pysp.scenariotree.tree_structure_model import \
```

```
        CreateConcreteTwoStageScenarioTreeModel

    st_model = CreateConcreteTwoStageScenarioTreeModel(num_scenarios)

    first_stage = st_model.Stages.first()
    second_stage = st_model.Stages.last()

    # First Stage
    st_model.StageCost[first_stage] = 'FirstStageCost'
    st_model.StageVariables[first_stage].add('x1')
    st_model.StageVariables[first_stage].add('x2')
    st_model.StageVariables[first_stage].add('x3')
    st_model.StageVariables[first_stage].add('x4')
    st_model.StageVariables[first_stage].add('x5')
    # Second Stage
    st_model.StageCost[second_stage] = 'SecondStageCost'
    st_model.StageVariables[second_stage].add('t1')
    st_model.StageVariables[second_stage].add('t2')
    st_model.StageVariables[second_stage].add('t3')
    st_model.StageVariables[second_stage].add('t4')
    st_model.StageVariables[second_stage].add('t5')
    return st_model

def pysp_instance_creation_callback(scenario_name, node_names):

    #
    # Clone a new instance and update the stochastic
    # parameters from the sampled scenario
    #

    instance = model.clone()

    d1_rhs_val, d2_rhs_val, d3_rhs_val, d4_rhs_val, d5_rhs_val  = scenario_data[scenario_name]
    instance.d1_rhs.value = d1_rhs_val
    instance.d2_rhs.value = d2_rhs_val
    instance.d3_rhs.value = d3_rhs_val
    instance.d4_rhs.value = d4_rhs_val
    instance.d5_rhs.value = d5_rhs_val

    return instance

# python -m pyomo.pysp.convert.smps -m itinerary_network.py --basename Itinerary \--output-directory
sdinput/Itinerary --symbolic-solver-labels
```

output

```
Reading problems from ./sdinput/Itinerary/Itinerary.cor
the filename is : ./sdinput/Itinerary/Itinerary.cor
Specified objective sense: MAXIMIZE
Selected objective  name:  obj
Selected RHS       name:  RHS
Selected bound      name:  BOUND
finished loading core file via external Solver.
25.000000
30.000000
200.000000
100.000000
160.000000
Vector x_k from the mean problem :: 1.00000000000000000000 -0.00000000000000000000 -0.00000000000000000000
-0.00000000000000000000 -0.00000000000000000000 -0.00000000000000000000 0.00000000000000000000
0.00000000000000000000 0.00000000000000000000 0.00000000000000000000 0.00000000000000000000
1.00000000000000000000
```

```
Objective from the mean problem : 0.000000


num_rv = 5;
num_cipher = 1;

-------------------- Replication No.0 --------------------
************Max_pi***************: 0.000000


-------------------- Replication No.1 --------------------
************Max_pi***************: 0.000000


-------------------- Replication No.2 --------------------


 Begin evaluation of average solution


Final Estimate
 obs:100 mean:0.000000
 0.95 C.I.: [0.000000 , 0.000000]

************Max_pi***************: 0.000000

Ending SD...

Total running time: 5545.827000 ms
```

I derived the same primal solution with AMPL solver and the dual price for each leg.

Leg j	Link	Dual Price
1	C - A	0
2	D - A	0
3	A - B	0
4	B - E	0
5	B - F	0
6	B - G	0

1 (d) Repeat the exercise of 1 (b) by using prices from 1 (c).

The sum of dual prices for itinerary 2 (C-A-B-E) is 0. Hence, we could get the $0 value from the itinerary 2. The threshold price for accepting/rejecting the bid for the travel C-A-B-E is **$0**