

```

# Retrofitting problem ver-3

# Import PuLP modeler functions
from pulp import *

import numpy as np

# The prob variable is created to contain the problem data
prob = LpProblem("retrofit_problem",LpMinimize)

# Creates a list of the Asset Locations
Asset_Locations = ['NCJ', 'TVC', 'KCVL', 'ERM', 'ALLP']

# Creates a list of the Retrofit Locations
Retrofit_Locations = ['NCJ', 'TVC', 'KCVL', 'ERM', 'ALLP', 'TPJ', 'MAS']

# Creates a list of original capacities of Retrofit Locations per day
capacity_before_upgrade = [0.5, 1, 0.0, 0.5, 0, 1.0, 1.5]

# Creates a list of original capacities of Retrofit Locations after upgradation per day
capacity_after_upgrade = [1, 2, 0.5, 1, 0, 2, 3]

# For taking input on the number of months in which the retrofit is to be completed
n =int(input("Enter the number of months for completion: "))

# Depending on the number of months the capacities for facilities before upgrade is calculated
for i in range(len(capacity_before_upgrade)):
    capacity_before_upgrade[i]=capacity_before_upgrade[i]*n*30

# Depending on the number of months the capacities for facilities after upgrade is calculated
for i in range(len(capacity_after_upgrade)):
    capacity_after_upgrade[i]=capacity_after_upgrade[i]*n*30

# A list of the number of assets at each asset location is created

asset_nos = [357, 591, 69, 277, 98]

# A list of the cost of upgrading a facility is created

```

```
upgradation_cost = [400, 400, 500, 400, 400, 400, 500]
```

```
# Creates a list of costs of each transportation path
```

```
transport_costs = [ #Retrofit locations  
    #NCJ TVC KCVL ERM ALLP TPJ MAS  
    [0,0.39,0.13,1000,1000,1.0,1.85],# NCJ  
    [0.39,0,0.06,0.83,1000,1.23,2.19], # TVC  
    [0.13,0.06,0,0.77,1000,1.22, 2.18], # KCVL  
    [1000,0.83,0.77,0,1000, 1.22, 1.93], # ERM  
    [1000,0.57,0.56,0.27,1000,1.31,2.10], # ALLP  
    ]
```

```
#transport_costs=makeDict([range(0,5), range(0,7)], transport_costs, 0)
```

```
transport_costs=makeDict([range(len(Asset_Locations)), range(len(Retrofit_Locations))],  
transport_costs, 0)
```

```
# The problem variables are created
```

```
x =  
LpVariable.dicts("xij",(range(len(Asset_Locations)),range(len(Retrofit_Locations))),0,None,LpInteger)
```

```
y = LpVariable.dicts("yj",(range(len(Retrofit_Locations))), 0, 1, LpInteger )
```

```
# The objective function is added
```

```
prob += lpSum([transport_costs[i][j]*x[i][j]] for i in range (len(Asset_Locations)) for j in range  
(len(Retrofit_Locations))) + lpSum(upgradation_cost[j]*y[j] for j in range(len(Retrofit_Locations)))
```

```
for i in range(len(Asset_Locations)):  
    prob += lpSum ([x[i][j] for j in range (len(Retrofit_Locations))]) == asset_nos[i]
```

```
for j in range (len(Retrofit_Locations)):  
    prob += lpSum ([x[i][j] for i in range(len(Asset_Locations))]) <=capacity_before_upgrade[j] +  
y[j]*(capacity_after_upgrade[j]-capacity_before_upgrade[j])
```

```

# The problem data is written to an .lp file
prob.writeLP("retrofit_problem.lp")

# The problem is solved using PuLP's choice of Solver
prob.solve()

# The status of the solution is printed to the screen
print ("Status:", LpStatus[prob.status])

# Each of the variables is printed with it's resolved optimum value
for v in prob.variables():
    if v.varValue>0:
        print (v.name, "=", v.varValue)

# The optimised objective function value is printed to the screen
print (n, "Months")
print ("Total Cost of retrofitting =", value(prob.objective))
print ("Total variable cost= ", value(lpSum([transport_costs[i][j]*x[i][j]] for i in range
(len(Asset_Locations))
                                     for j in range (len(Retrofit_Locations)))))
print ("Total fixed cost= ", value(lpSum(upgradation_cost[j]*y[j] for j in
range(len(Retrofit_Locations)))))
print ("Number of facilities upgraded= ", (value(lpSum(y[j]for j in
range(len(Retrofit_Locations)))))

```

```

import matplotlib.pyplot as plt
import numpy as np
Months = np.array([5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24])
TCR = np.array([3769,2672,2137,1679,1304,1100,912,783,709,639,571, 505, 435, 365, 296,
230, 167, 104, 52, 35])
TVC=np.array([1169,1372,937,879,904,700,512,383, 309, 239,171, 505, 435, 365, 296, 230,
167,104, 52, 35])
TFC=np.array([2600,1300,1200,800,400,400,400,400,400,400,0,0,0,0,0,0,0,0,0,0])
FU=np.array([6,3,3,2,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0])
plt.plot(Months,TCR, 'ro--', label='Total costs' )

# 'r--'
# 'ro'
plt.xticks(np.arange(5, 25, 1))
plt.ylabel('Amount (lakhs)')
plt.xlabel("Time for completion (months)")

plt.plot(Months,TFC, 'g^--', label='Fixed costs')
plt.plot(Months,TVC, 'bs--', label='Variable costs')

plt.ylabel('Total fixed cost (lakhs)')
plt.xlabel("Time for completion (months)")
plt.legend(loc='upper right')
plt.show()

```

```

# Scheduling of coaches - Retrofitting problem
# Import PuLP modeler functions
import pulp
from pulp import *
import numpy as np

# The prob variable is created to contain the problem data
prob = pulp.LpProblem("scheduling_problem",LpMinimize)

# Creates a list of rakes
Rakes = ['R1', 'R2','R3','R4']

# Creates a list of coaches in each rake that need retrofit
Coaches = [8,7,8,14]

#Total days available
n=100

Time=[]
i=0
for i in range(n):
    Time.append(i+1)

# Discrete times available for starting of retrofit
#Time = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23]

# Creates lists of rake cycle times
Tr = [[1,7,13,19,25,31,37,43,49,55,61,67,73,79,85,91,97],
      [3,9,15,21,27,33,39,45,51,57,63,69,75,81,87,93,100],
      [4,10,16,22,28,34,40,46,52,58,64,70,76,82,88,94,100],

      [1,3,5,7,9,11,13,15,17,19,21,23,25,27,29,31,33,35,37,39,41,43,45,47,49,51,53,55,57,59,61,63,65,67,69,71,
      73,75,77,79,81,83,85,87,89,91,93,95,97,99]]

# The problem variables are created

```

```

x = pulp.LpVariable.dicts("xrt",(range(len(Rakes)),range(len(Time))),0,1,LpInteger)

y = pulp.LpVariable("y", cat="Integer")

# The objective function is added
prob += y

#prob += lpSum([Time[t]*x[r][t]] for t in range (len(Time)) for r in range (len(Coaches)))

for r in range(len(Rakes)):
    prob += pulp.lpSum ([x[r][t] for t in range (len(Time))]) == Coaches[r], "All_rakes"+str(r)

for t in range (len(Time)):
    prob += pulp.lpSum ([x[r][t] for r in range(len(Rakes))]) <= 1, "Capacity" + str(t)

#for r in range(len(Rakes)):
#    for t in range (len(Time) - 1):
#        prob += lpSum([x[r][t+1],x[r][t]]) <= 1

for t in range(len(Time) - 1):
    prob += pulp.lpSum([x[r1][t] for r1 in range (len(Rakes))] +\
        [x[r2][t + 1] for r2 in range(len(Rakes))]) <= 1, "Time_Constraint" + str(t)

for r in range(len(Rakes)):
    prob += pulp.lpSum([x[r][t] for t in range(len(Time)) if t not in Tr[r]]) == 0, "Rake_cycle" + str(r)

for r in range(len(Rakes)):
    for t in range (len(Time)):
        prob += y >= [Time[t]*x[r][t]] , "Maximum_time" + str(r) + "_" + str(t)

# The problem data is written to an .lp file
prob.writeLP("scheduling_problem.lp")

# The problem is solved using PuLP's choice of Solver
prob.solve()

# The status of the solution is printed to the screen
# print ("Status:", pulp.LpStatus[prob.status])

# Each of the variables is printed with it's resolved optimum value
for v in prob.variables():

```

```
if v.varValue>=1:  
    print (v.name, "=", v.varValue)
```