**Take Home Exam-Report**

I started with prior GIPSolver code. And I coded an anumerative solver to find optimal solution of my problem sets:

*--------------------------------------------------------------------------------------------------------------------------------------*

*def optimal\_knapsack\_solver(A,b,c):*

*#Create a list of all possible solutions*

*import itertools*

*lst = list(map(list, itertools.product([0, 1], repeat=len(c))))*

*# Multiply constraints with all possible solutions*

*std = np.dot(A,np.transpose(lst))*

*#Calculate the differences between calculated solutions and constraints*

*diff = np.transpose(std) - np.transpose(b)*

*#Calculate and append objective functions value and find optimal solution*

*sol\_list = []*

*for i in range(0,len(diff)):*

*if ((diff[i][0] > 0) | (diff[i][1] > 0)) :*

*sol\_list.append(0)*

*else:*

*sol\_list.append(sum(lst[i]\*c))*

*optimal\_solution, optimal\_value = lst[sol\_list.index(max(sol\_list))] , max(sol\_list)*

*return optimal\_value, optimal\_solution*

*--------------------------------------------------------------------------------------------------------------------------------------*

As mentioned in assignment; I used 4 factors;

* Size
* Time
* Ratio
* Probability

I created 3 levels for each factor:

*size\_opt = [10,50,100]*

*time\_opt = [20,100,200]*

*ratio\_opt = [0.5,0.7,0.9]*

*prob\_opt = [0.2,0.5,0.8]*

These are my **inner arrays.**

As **outer array**, I implemented 5 different trials with 5 different problems.

Each problem has 2 constraints with 10 dimensions and aim is to find optimal solution. Each problem created randomly with the code:

*--------------------------------------------------------------------------------------------------------------------------------------*

c = np.array([random.randint(0,4) for i in range(10)])

A = np.array([[random.randint(0,4) for i in range(10)],[random.randint(0,4) for i in range(10)]])

b = [random.randint(13,15) for i in range(2)]

*--------------------------------------------------------------------------------------------------------------------------------------*

I implemented Taguchian methodology network which is given already. I copied it to python:

*--------------------------------------------------------------------------------------------------------------------------------------*

#Given Taguchian\_OA

*Taguchian\_OA = [[0,0,0,0],*

*[0,1,1,1],*

*[0,2,2,2],*

*[1,0,1,2],*

*[1,1,2,0],*

*[1,2,0,1],*

*[2,0,2,1],*

*[2,1,0,2],*

*[2,2,1,0]]*

*(Python index starts from 0 so 0 means 1 in Taguchian table)*

*--------------------------------------------------------------------------------------------------------------------------------------*

*After that, I calculated optimality gap of every factor combination with every trial of problems.*

*The optimality gap formula that I used is:*

*round(((optimal\_value-score) / optimal\_value)\*100,0)*

*After Calculating every cell’s optimality gap my Taguchian table looks like:*

*--------------------------------------------------------------------------------------------------------------------------------------*



*Experiment numbers on index, initial 4 columns are factor combinations (inner array), last 5 columns are different problem sets which are trial numbers (outer array).*

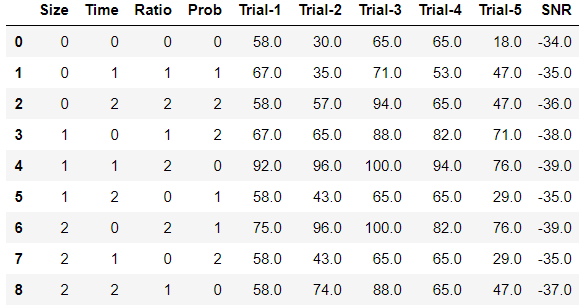
*--------------------------------------------------------------------------------------------------------------------------------------*

*The next step must be SNR calculation.The aim is to minimize optimality gap. So I used minizing Signal to noise ratio formula:*

**

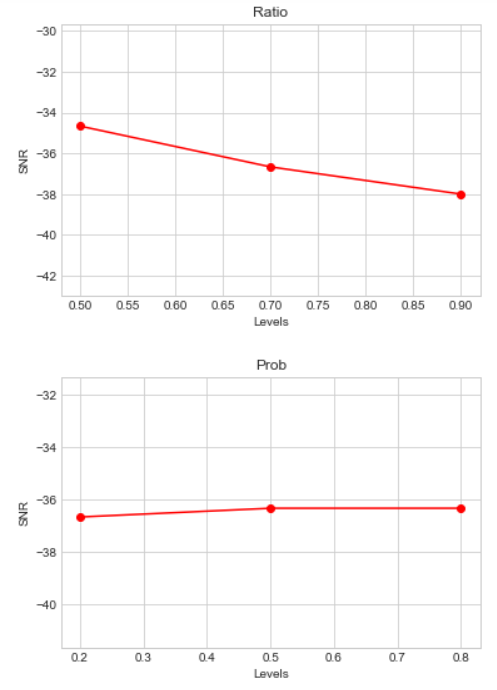
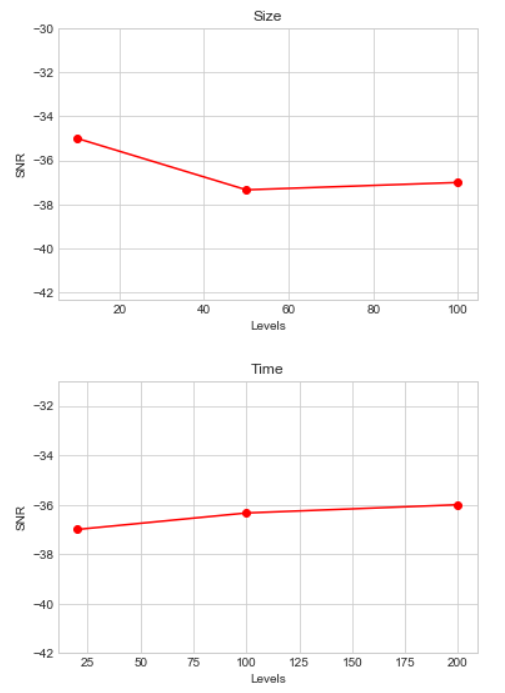
*The outcome is:*

*--------------------------------------------------------------------------------------------------------------------------------------*



*--------------------------------------------------------------------------------------------------------------------------------------*

*After I calculated SNR of every experiment, the next step was to find average SNR for each level of each factor and the summary graphs are:*



*After I choose optimal of every SNR value. My robust parameter set was :*

Size, Time, Ratio, Probability : (10, 200, 0.5, 0.8)

*--------------------------------------------------------------------------------------------------------------------------------------*

***Finally:***

*I set my params and my final optimality gap with choosen ones is: 15.0*