otml_project

November 30, 2023

[23]:
```python
import numpy as np
import pandas as pd
import math
import itertools

#Compute Generation Losses on Loss of Branches
n=6 #set number of branches here
gen_loss=np.random.randint(low=30,high=2000,size=(1,n))
gen_loss=gen_loss[0]
list1=[]
for i in range(1,len(gen_loss)+1):
    list1.append(i)
    print("Generation Loss on failiure of Branch",i,"-",gen_loss[i-1])
print("\n")

#n Choose 2 combinations of pairs of branches are possible, so we list them all
a=math.comb(n,2)
gen_pair_loss=np.zeros((a),dtype='i')
list_of_tuples=[]
list_of_indices=[]
i=0
for comb in itertools.combinations(list1, 2):
    list_of_tuples.append(comb)
    r=str(comb[0])+','+str(comb[1])
    list_of_indices.append(r)
    gen_pair_loss[i]=gen_loss[comb[0]-1]+gen_loss[comb[1]-1]
    i+=1
for i in range(0,len(gen_pair_loss)):
    print("Generation Losses for pair",list_of_tuples[i],gen_pair_loss[i])
print("\n")


#Creating Payoff Matrix
payoff=np.zeros((a,a),dtype='i')
for i in range(0,a):
    for j in range(0,a):
        q=gen_pair_loss[j]
```

```
        set1={list_of_tuples[j][0],list_of_tuples[j][1]}
        set2={list_of_tuples[i][0],list_of_tuples[i][1]}
        set1=set1.intersection(set2)
        if len(set1)>0:
            for k in set1:
                q=q-gen_loss[k-1]
        payoff[i][j]=q


#Conversion to DataFrame for easier processing
payoff_map=pd.
 ↪DataFrame(payoff,columns=list_of_tuples,index=list_of_tuples,dtype='f')


#Case-1: If Defender was static::defender choses each defence vector pair with␣
 ↪equal probability
#then our attacker must chose the column of payoff matrix that has highest sum␣
 ↪of values
game_value=payoff_map.sum(axis=0).values
for i in range(0,len(game_value)):
    game_value[i]/=math.comb(n,2)


#Case-1 solution
print("Best Attack Vector in Case-1 is: ",payoff_map.sum(axis=0).idxmax())
print("Best Probable Generation Loss is: ",round((payoff_map.
 ↪sum(axis=0)[payoff_map.sum(axis=0).idxmax()])/math.comb(n,2)))


#Case-2 Defender is intelligent
#Here defender chooses defence vector to protect his more valuable branch pairs␣
 ↪with a higher frequency.
probability_vector=[]
sum=gen_pair_loss.sum()
for i in range(0,len(gen_pair_loss)):
    probability_vector.append(1-((gen_pair_loss[i])/sum))
p_vector=np.array(probability_vector,dtype='f')
p_vector = (p_vector - np.min(p_vector)+np.min(p_vector)*0.1)/(np.
 ↪max(p_vector)-np.min(p_vector))
#print(p_vector)


#for every attack-vector column, the probable generation-loss is now the mean␣
 ↪of modified values. The modified values refer to
#the rows of payoff matrix multilplied by corresponding probability factor␣
 ↪p_vector.
```

```python
modified_payoff=payoff_map.mul(p_vector,axis=1)


#Case-2 solution
print("Best Attack Vector in Case-2 is: ",modified_payoff.sum(axis=0).idxmax())
print("Best Probable Generation Loss is: ",round((payoff_map.
 ↪sum(axis=0)[modified_payoff.sum(axis=0).idxmax()])/math.comb(n,2)))
```

```
Generation Loss on failiure of Branch 1 - 1617
Generation Loss on failiure of Branch 2 - 493
Generation Loss on failiure of Branch 3 - 328
Generation Loss on failiure of Branch 4 - 633
Generation Loss on failiure of Branch 5 - 363
Generation Loss on failiure of Branch 6 - 1972


Generation Losses for pair (1, 2) 2110
Generation Losses for pair (1, 3) 1945
Generation Losses for pair (1, 4) 2250
Generation Losses for pair (1, 5) 1980
Generation Losses for pair (1, 6) 3589
Generation Losses for pair (2, 3) 821
Generation Losses for pair (2, 4) 1126
Generation Losses for pair (2, 5) 856
Generation Losses for pair (2, 6) 2465
Generation Losses for pair (3, 4) 961
Generation Losses for pair (3, 5) 691
Generation Losses for pair (3, 6) 2300
Generation Losses for pair (4, 5) 996
Generation Losses for pair (4, 6) 2605
Generation Losses for pair (5, 6) 2335


Best Attack Vector in Case-1 is:  (1, 6)
Best Probable Generation Loss is:  2393
Best Attack Vector in Case-2 is:  (4, 6)
Best Probable Generation Loss is:  1737
```

[ ]:

[ ]: