A Study of Linear Programming and Reinforcement Learning for One-Shot Game in Smart Grid Security

A Report

submitted by

Anshu Saini, Kathiravan, Dhivya Dharshan, Abishek Chakravarthy

CS22B2051, CS22B2052, CS22B2053, CS22B2054

B. Tech. Computer Science and Engineering with major in AI

Department of Computer Science

INDIAN INSTITUTE OF INFORMATION TECHNOLOGY, DESIGN AND MANUFACTURING, KANCHEEPURAM

CHENNAI-600127

9 December 2023

Contents

0 Abstract:-

Smart Electric Power Grid Security: A Game Theoretic Approach

Smart electric power grid systems are interconnected between various regions to meet the industrial and domestic energy demands of the modern age. It is of utmost importance that we minimize the risk of cyber attacks in these important power transmission systems. This report explores different approaches to defend against the cyber attacks that the power grid is vulnerable to, by simulating "*one-shot games*" using Game Theory, wherein the attacker and the defender choose their actions simultaneously. The proposed game is demonstrated on two power system models - a 6-bus system and the IEEE 30-bus system. We discuss two possible variations of the one-shot game, *Single-line attack* and *Multi-line attack*.

Single-line Attack

We model this scenario using *Reinforcement Learning* (RL). The RL algorithm learns the optimal attack strategy for the attacker based on the defender's chosen action. This helps defenders anticipate and counter potential attacks.

Multi-line Attack

Here, we employ *Linear Programming* (LP) to solve the game. The LP solution provides the attacker's probability of choosing the best attack strategy for different defender actions. This allows defenders to prioritize hardening critical components.

We measure success based on the attacker's ability to disrupt power generation. Both defender actions and their effectiveness are factored into the attacker's payoff (gain or loss). In essence, this report offers a strategic framework for both sides in the smart grid security game. Defenders can leverage *Linear Programming* and *Reinforcement Learning* to predict and counter potential attacks, while attackers can refine their strategies based on defender actions.

1 Introduction to the Problem:-

Modern power grids are vast and interconnected, encircling diverse components across regions and nations. They efficiently deliver electricity through large-scale generation, flexible transmission and distribution, and distributed energy sources to meet industrial and domestic energy demands. However, this modernization exposes the grid to serious vulnerabilities, especially cyber-attacks. Events like transmission line outages, generator failures, and load variations pose challenges, requiring robust monitoring and control.

The integration of traditional power grids with cyber operations introduces new threats, including cyber attacks on operating stations, false data injections, and malware injections. Cascading failures, cyber-physical security, and human factors become crucial research areas for securing modern smart grids.

To address these challenges, *game theory* has emerged as a valuable analytical tool for understanding complex interactions in the smart power system. Game theory enables the analysis of strategic interactions between rational and intelligent players, such as attackers and defenders, using mathematical rules and frameworks. System operators constantly monitor and manage the system's health, taking corrective actions during disruptions to restore normal operation. Game theory sheds light on *attacker-defender dynamics*, revealing potential weaknesses.

Power system operators and protection schemes vigilantly monitor key parameters like voltage, power, and frequency. Incidents such as the northeast power grid blackout, the cyber attack on Ukraine's power grid, and Stuxnet's attack on Iran's nuclear program highlight the urgent need for enhanced security. Understanding attacker strategies and system vulnerabilities is essential for achieving this.

Existing research leverages game theory to grasp attacker-defender interactions, thereby strengthening protection schemes by identifying and patching vulnerabilities. Static games analyze single actions, while dynamic games consider multiple interdependent actions.

This report proposes solutions for a one-shot game in the smart electric power grid, considering single and multiple transmission line outages in the attacker's action set. The proposed linear programming-based approach provides multiple solutions for different attack-defense action sets with probabilities for execution using multi-line-switching attacks. A *pay-off matrix*, considering generation loss due to line switching, is formulated to solve the game. Reinforcement learning offers an optimal attack action from the attacker's perspective using single-line switching attacks. This approach reveals that the defender's actions can be adapted based on the attacker's history, strengthening power system security by identifying and protecting critical elements, thereby reducing generation and financial losses.

2 Problem Formulation & Implementation:-

In this section, we present the formulation and resolution of the game between the attacker and defender through two distinct methods: linear programming for multi-line switching attacks and reinforcement learning for single-line switching attacks. The problem formulation for a *two-player zero-sum* game is demonstrated using a game matrix for the linear programming-based solution. The calculation of generation loss is briefly explained, employing load ranking of buses to identify critical buses and associated transmission lines in a typical power system.

The proposed game solution is demonstrated using the following approach: Both the linear programming solution and reinforcement learning solution is showcased using the *IEEE 6-bus system*.

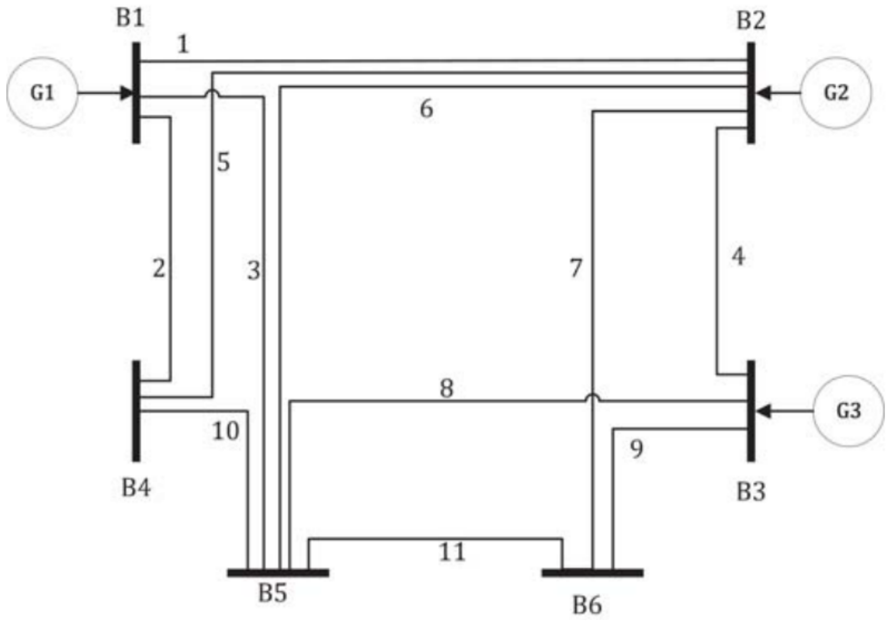


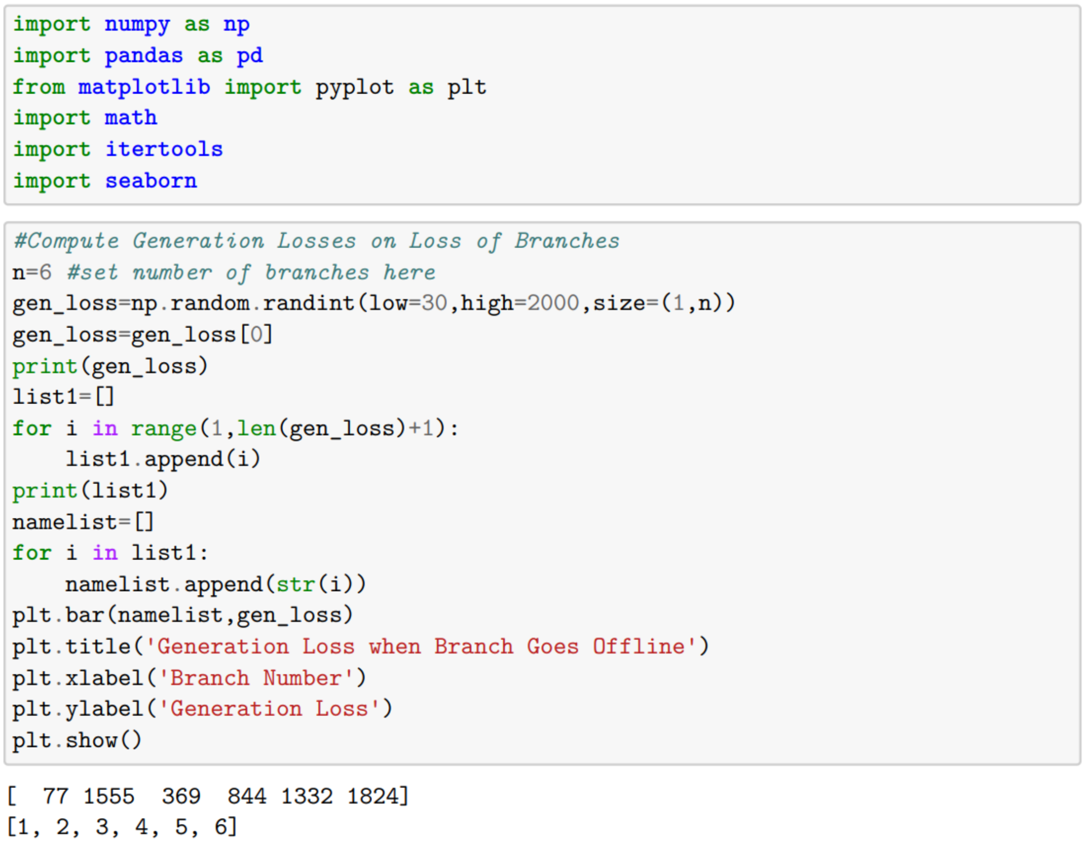
Figure 1: 6-bus system

The above figure illustrates the one-line diagram of the 6-bus system. The topological information is utilized to create attack and defence vectors and assess the attack-defence strategy.

Calculation of Generation loss:-

In this report, we generate random values for generation losses due to the failure of different data transmission lines in the grid system to improve our understanding about the one-shot game and for data visualisation purposes.

The calculation of generation loss involves giving the system the indices of the target branches. We then use random library to generate the values for the generation loss. To address simultaneous attack strategies, where multiple targets are attacked simultaneously, the report adopts this approach for linear programming-based game solution. In contrast, for the reinforcement learning-based solution, single-line-switching attacks are considered one at a time.



```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import math
import itertools
import seaborn
```

```python
#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]
print(gen_loss)
list1=[]
for i in range(1,len(gen_loss)+1):
    list1.append(i)
print(list1)
namelist=[]
for i in list1:
    namelist.append(str(i))
plt.bar(namelist,gen_loss)
plt.title('Generation Loss when Branch Goes Offline')
plt.xlabel('Branch Number')
plt.ylabel('Generation Loss')
plt.show()
```

```
[  77 1555  369  844 1332 1824]
[1, 2, 3, 4, 5, 6]
```

Figure 2: Computing Generation losses

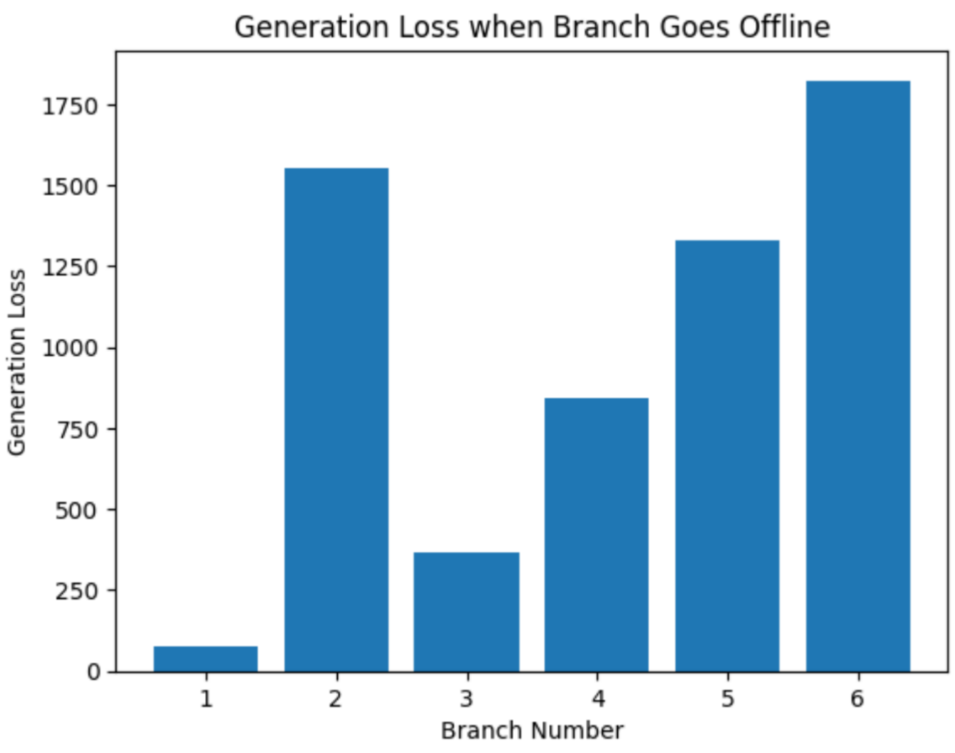


Figure 3: Generation losses

```
a=math.comb(n,2)
print(a)
gen_pair_loss=np.zeros((a),dtype='i')
print(gen_pair_loss)
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
print(gen_pair_loss)
plt.bar(list_of_indices,gen_pair_loss,color='green')
plt.xlabel("Pairs of Branches")
plt.ylabel("Generation Loss")
plt.title("Generation Loss when Pairs of Branches Go Offline")
```

Figure 4: Computing Generation loss for pairs

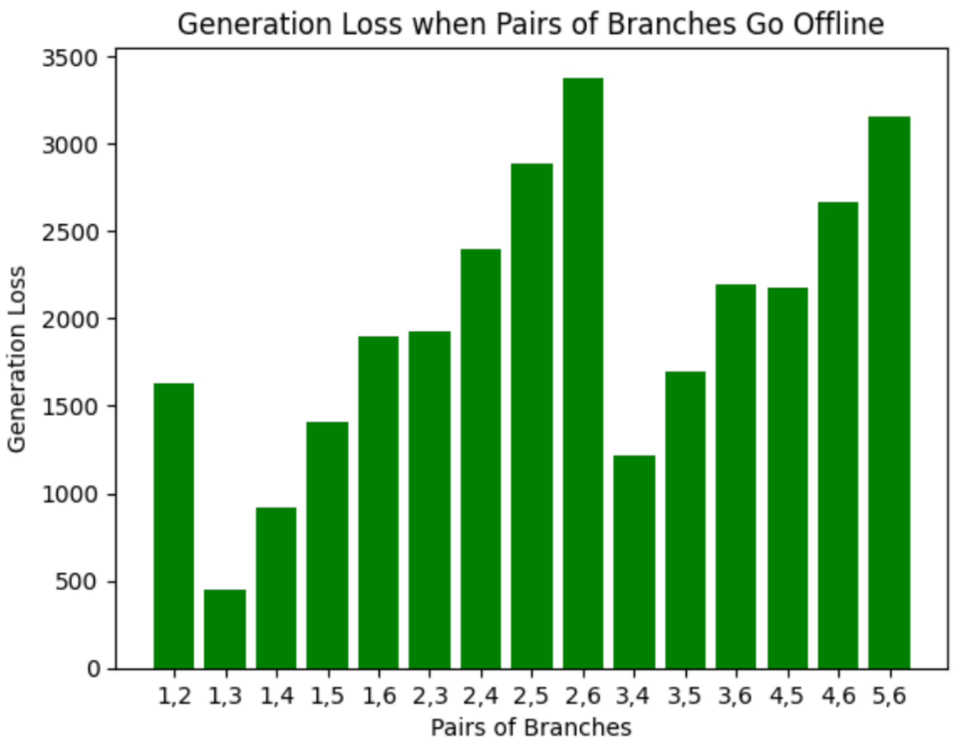


Figure 5: Generation loss for pairs

Generating Attack Matrix:-

The one-shot game requires an *attack matrix*, which includes branches associated with the buses in a typical power system. Each row corresponds to a bus, and columns represent branches connected to these buses. After calculating generation loss, target buses are chosen based on their criticality, determined by the loss incurred from switching the lines connected to these buses. More significant generation loss implies greater criticality.

After calculating generation loss for each target against defence sets, a pay-off matrix is constructed. The figures below illustrate the pay-off matrix, where columns represent pay-offs for the attacker's strategies, and rows represent the defender's strategies. Element (i, j) in the matrix signifies the pay-off for attack action j in response to defence action i.

The pay-off matrix reveals that if target z1z2 is simultaneously defended and attacked, the attack fails, causing no generation loss. The diagonal elements of the matrix are zero because these attacks are successfully defended by the defender. Solutions to the game matrix can be found through various methods, such as the minimax theorem, linear programming, etc.

```python
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
print(payoff)
```

Figure 6: Computing Payoff matrix

```python
payoff_map=pd.
 ↪DataFrame(payoff,columns=list_of_tuples,index=list_of_tuples,dtype='f')
seaborn.heatmap(payoff_map,cmap=seaborn.cm.rocket_r)
plt.title("Payoff Matrix - HeatMap of Generation Losses")
plt.xlabel("Pair of Branches Chosen By Attacker")
plt.ylabel("Pair of Branches Chosen By Defender")
plt.show()
```

Figure 7: Heatmap of Payoff matrix (Code)

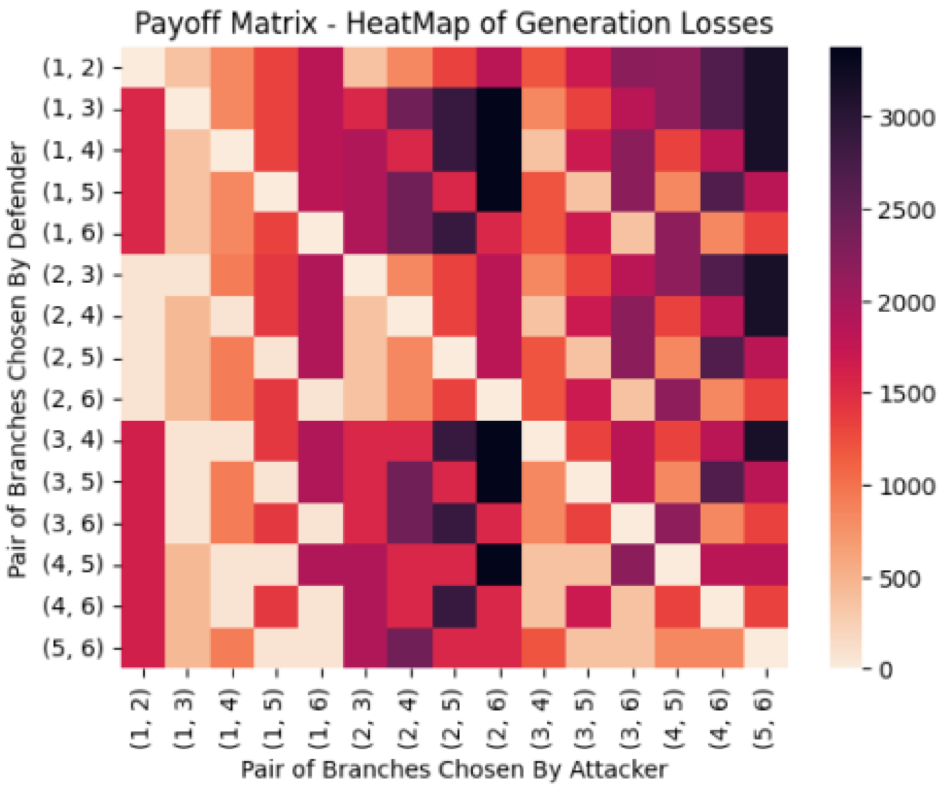


Figure 8: Heatmap of Payoff matrix

3 Attacker Defender Interaction:-

To safeguard a set of transmission lines (i.e., specific targets), the defender engages in a strategic game against the attacker. Notably, the defender cannot protect all elements simultaneously, and likewise, the attacker cannot attack all elements simultaneously.

We analyze two versions of the one-shot game - *Two player Zero sum Game* using *Linear programming* and *Two player Zero sum Game* using *Q-Learning.*

Two Player Zero Sum Game - Linear Programming Approach:-

We design an interactive decision-making game modeling a strategic game. Given a matrix $A_{m\times n} = \{a_{ij} : i = 1, \dots, m; j = 1, \dots, n\}$ which here represents the payoff matrix, a pair of strategies $\{row : i^*, column : j^*\}$, adopted by the players (attacker and defender) constitutes a saddle point equilibrium if the following condition for all i, j is satisfied.

$$a_{i^*j} \leq a_{i^*j^*} \leq a_{ij^*} \tag{1}$$

These strategies, known as saddle point strategies, form a saddle point equilibrium. If a two-person zero-sum game has a single saddle point, the associated value is termed the value of the game. If there's no saddle point in pure strategies, mixed strategies are used.

For a given payoff matrix $A_{m\times n} = \{a_{ij} : i = 1, \dots, m; j = 1, \dots, n\}$, the average value of the game, considering mixed strategies, can be expressed as:

$$J(y, w) = \sum_{i=1}^{m}\sum_{j=1}^{n} y_i a_{ij} w_j = y^T A w \tag{2}$$

Here, y and w are probability distribution vectors defined as

$$y = (y_1, \dots, y_m)^T \tag{3}$$

$$w = (w_1, \dots, w_n)^T \tag{4}$$

The defender aims to minimize $J(y, w)$ by selecting an optimal probability distribution vector $y \in Y$, while the attacker seeks to maximize the same quantity with an appropriate $w \in W$. The sets Y and W are defined as:

$$Y = \{y \in R^m \mid y \geq 0, \sum_{i=1}^{m} y_i = 1\} \tag{5}$$

$$W = \{w \in R^n \mid w \geq 0, \sum_{j=1}^{n} w_j = 1\} \tag{6}$$

A vector w^* is a mixed strategy for the defender if the condition for all $\forall y \in Y$ is satisfied:

$$V_m(A) = \max_{w \in W} y^{*T} A w \leq \max_{w \in W} y^T A w, \quad y \in Y \tag{7}$$

Similarly, the attacker's average security level can be formulated:

$$V_m(A) = \min_{y \in Y} y^T A w \geq \min_{y \in Y} y^T A w, \quad w \in W \tag{8}$$

These inequalities can be alternatively expressed as:

$$V_m(A) = \min_{y} \max_{w} y^T A w \tag{9}$$

$$V_m(A) = \max_{w} \min_{y} y^T A w \tag{10}$$

Here is where duality concept from LPP comes into the picture and this LPP is solved to get the probable attack and defense vectors for the attacker and defender respectively.

Two Player Zero Sum Game - Q-Learning Approach:-

In Q-learning, the agents are the attacker and defender, and the environment is the power system. The reward is the feedback from the environment for attacker-defender actions. Optimal strategies can be found through mixed strategies maximizing expected long-term rewards. The value function $V_a(s)$ and $V_d(s)$ are formulated for the attacker and defender, respectively.

The problem is solved through value iteration, and from the attacker's perspective, the problem becomes:

$$V_A(s) = \max_{\pi_A(s)} \min_{\pi_D(s)} \sum_{a \in M_A(s)} \sum_{d \in M_D(s)} \pi_A(s) Q_A(a,d,s) \pi_D(s) \tag{11}$$

Here $Q_A(a,d,s)$ represents the quality of state s; $\pi_A(s)$ and $\pi_D(s)$ represent probability distributions over the strategies of the attacker and defender at a particular state s.

In this context, it is assumed that the defender's action is fixed throughout the game. The game is solved using value iteration, and equations are established to determine optimal strategies and values for each player. Here, the defender's action is assumed to be fixed throughout the game, initially determined randomly.

4 Simulation Results:-

Simulations using Linear Programming Approach:-

In this section, we will examine the consequences of an attack on the IEEE 30 bus system. Under the given assumptions, the attacker can simultaneously target two of the data buses. Consequently, the final targets encompass transmission lines connected to two buses simultaneously. The defender also possesses the capability to defend against two targets concurrently. It is presumed that if the attacker selects the strategy $\{z_i,\, z_j\}$ (attacking targets i and j) and the defender opts for the strategy $\{z_j,\, z_k\}$ (defending targets j and k), the attack will succeed, resulting in generation loss solely for the switching lines connected to z_i. The payoff matrix which was constructed earlier represented the outcomes of various attack and defense strategies, considering the actions of both players.

Upon analysis of the IEEE 30 bus system results, there is no single saddle point for a solution in equilibrium and the problem shifts towards determining the proportion of times that the attacker and the defender employ their respective strategies.

The code and the data visualization that we have done for this approach indicates that the best combination for the attacker is (2,5) with a generation loss of 1925 MW and the best combination for the defender is (1,3).

Simulations using Reinforcement Learning Approach:-

Reinforcement learning proves effective in solving the one-shot game based on the formulas outlined in Section III-B, and the subsequent simulation results are presented and elucidated here. The benchmark for this analysis is a 6-bus system.

Generation loss serves as the reward R(a, d, s) in solving the two-person zero-sum game using reinforcement learning.

In our case, the best possible attack strategy for the attacker is (2,6) with a generation loss of 2253 MW, and the best possible defence strategy for the defender is (1,3).

Code and Visualization of Results for both approaches:-

i Defender is Static:-

```
#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)
plt.xlabel("Attack Vector")
plt.ylabel("Probable Generation Loss")
plt.title("Avg Game Value for Each Attack Vector")
plt.bar(list_of_indices,game_value,color='#000000')
plt.show()
```

Figure 9: Computing Game Value, when Defender is Static (Code)

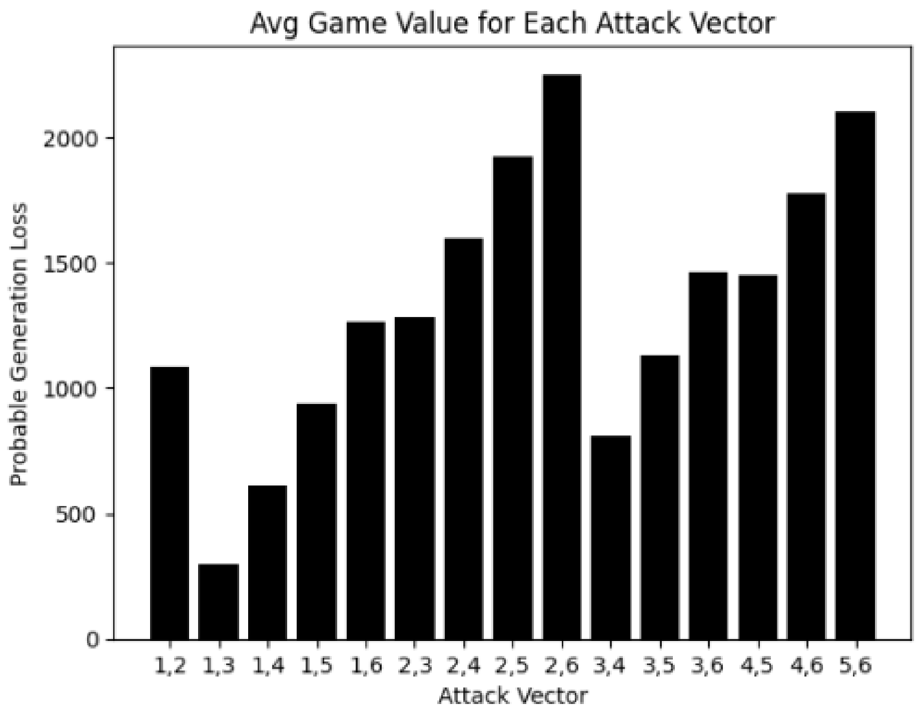


Figure 10: Game Value, when Defender is Static (Visualization)

```
#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)))
```

```
Best Attack Vector in Case-1 is:  (2, 6)
Best Probable Generation Loss is:  2253
```

Figure 11: Solution, when Defender is Static

ii Defender is Dynamic:-

```
#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)
```

```
[1.5034441 1.9078078 1.7458576 1.5794749 1.4117289 1.4038874 1.241937

 1.0755545 0.9078078 1.6463008 1.4799182 1.3121722 1.3179679 1.150222
 0.9838393]
```

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

```
gr=modified_payoff.sum(axis=0).values
plt.xlabel("Attack Vector")
plt.ylabel("Probable Generation Loss")
plt.title("Game Value for Each Attack Vector")
plt.bar(list_of_indices,gr,color='#000000')
plt.show()
```

Figure 12: Computing Game Value, when Defender is Dynamic (Code)

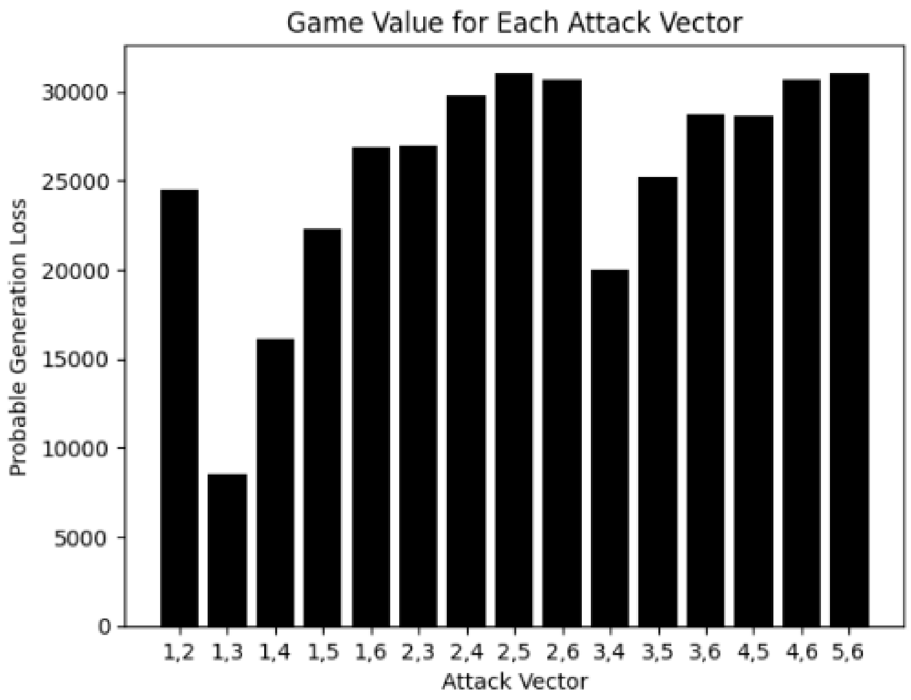


Figure 13: Game Value, when Defender is Dynamic (Visualization)

```
#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)))
```

```
Best Attack Vector in Case-2 is:  (2, 5)
Best Probable Generation Loss is:  1925
```

Figure 14: Solution, when Defender is Dynamic

5 Conclusion:-

In this report, we have introduced two approaches to address the smart grid security problem within a game framework. Firstly, we applied a linear programming algorithm for a scenario involving multi-line-switching attacks. In this case, we performed pre-calculation of generation loss based on the system model. Secondly, we employed reinforcement learning for a scenario involving single-line-switching attacks. Notably, this approach does not require pre-calculation and operates in an online and data-driven manner.

In the first method utilizing linear programming, the mixed strategy of both the attacker and defender is determined to identify optimal actions and their respective probabilities. This approach provides insights into the strategic choices of both parties. Conversely, the reinforcement learning-based solution for the one-shot game operates without the need for pre-calculation. It learns the defender's action policy from historical data of the attacker's actions. This method sheds light on the optimal attack actions in the presence of a static defender's strategy for enhancing smart grid security.

We have also coded all the algorithms in the report and we have visualized the payoff matrix and the results that we have inferred.

6 References:-

1. "Linear Programming lecture slides."
   aclanthology.org, www.aclanthology.org/L18-1334.pdf

2. "LPP reference paper",
   uomustansiriyah.edu,
   www.uomustansiriyah.edu.iq/media/lectures/5/5_2018_01_14!04_31_43_PM.pdf

3. "Reinforcement Learning Wikipedia Page",
   https://en.wikipedia.org/wiki/Reinforcement_learning