Help Document for Research Experiemnts

GD Chen

May 2023

1 Multiple Linear Program (MLP)

Our code is an algorithm designed to solve the Stakelberg Security Game (SSG) problem, built upon the CVX package.

1.1 Single Linear Program

A linear program(LP) could be written as follows(standard form):

$$\begin{aligned} \min_{\mathbf{x}} \quad & \mathbf{c}^T\mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \leq \mathbf{b} \\ & \mathbf{x} \geq 0 \end{aligned} \tag{1}$$

When it comes to Stakelberg Security Game, SSG problem could be written as a LP problem under some constraints. Consider a SSG problem, there exists a special $\mathbf{A}$ and $\mathbf{b}$, every $\mathbf{x} \in \{A\mathbf{x} \leq \mathbf{b}, \mathbf{x} \geq 0\}$ will make attackers from specific types attack specific targets according to a constant corresponding relationship. In this case, the expected payoff function of defenders could be written as a linear function and we could compute optimal strategy $\mathbf{p}(\mathbf{x})$ in this region. Given target i, attacker type j, we could determine the region of $\mathbf{x}$ by determine the specific $\mathbf{A}$ and $\mathbf{b}$, denoted as $\mathbf{A}_j^i$ and $\mathbf{b}_j^i$. $\mathbf{A}_j^i$ and $\mathbf{b}_j^i$ could be determined by comparing the expected utility function of j attacking i with every expected utility function of attacker j. Expected utility function could be found in page 4 of [1].

1.2 Example

Table 1: Utility Matrix of defender

Protected Target	Attack 1	Attack 2
Target 1	1	-1
Target 2	-1	1

Table 2: Payoff Matrix of defender (target is not covered by defender)

Attacked Target	Target 1	Target 2
Defender	-1	-1

Table 3: Payoff Matrix of defender (target is covered by defender)

Attacked Target	Target 1	Target 2
Defender	1	1

Table 4: Payoff Matrix of attacker (target is not covered by defender)

Attacked Target	Attacker 1
Target 1	1
Target 2	1

Table 5: Payoff Matrix of attacker (target is covered by defender)

Attacked Target	Attacker 1
Target 1	0
Target 2	0

From the utility matrix above, we could get the decision boundary:

$$1 - p_1 = 1 - p_2$$

, where p_1, p_2 is the subvector of defender's strategy $\mathbf{p}$. Then we could write one SSG problem into two LP problems. Solve these LP problems, we could get the optimal strategy.

```

%Initialize utility matrix
U_defender_c=[1 1]
U_defender_u=[-1 -1]
U_attacker_u=[1 ;1]
U_attacker_c=[0 ;0]
n=size(U_defender_u)

%region 1, where target 1 is attacked by attackers
cvx_begin
    variable x(n)
    maximize(-(1-x(1))+x(1))
    subject to
        x(1)-x(2)<=0
        norm(x, 1) <= 1
        0 <= x <= 1
cvx_end

%region 2, where target 2 is attacked by attackers
cvx_begin
```

```
    variable y(n)

    maximize(-(1-y(2))+y(2))
    subject to
        y(2)-y(1)<=0
        norm(y, 1) <= 1
        0 <= y <= 1
cvx_end
```

We could solve LP problem and compute the optimal strategy in each region. The optimal solution in `region 1` is `(0.5, 0.5)` and the optimal solution in `region 2` is `(0.5, 0.5)`. The optimal strategy is `(0.5,0.5)`. By comparing the optimal strategy in each region, we could get the optimal solution for our SSG problem.

1.3 Algorithm Based On MLP

The expected payoff function of the defender is non-linear when there are more than one target and attacker type. By dividing the strategy space into smaller spaces, the payoff function could be written as a linear function when the best response of attackers in this region is constant. From this idea, we could divide strategy space into regions of this kind and compute their optimal strategies. By comparing these strategies, we could get the optimal strategy.
Hence, we designed an algorithm that could help us divide strategy space into regions corresponding to all n^k possible target-attacker corresponding relationships, where n is the number of targets and k is the number of attacker types.

1.3.1 Input Output

The function MLP is written as:

```
[optimal,z1,target] = MLP(U_defender_c,U_defender_u,U_attacker_c,
   U_attacker_u)
```

`U_defender_c(1×n)` is the utility matrix of defender when target attacked by attacker is under the protection of defender.
`U_defender_u(1×n)`is the utility matrix of defender when target attacked by attacker is not under the protection of defender.
`U_attacker_u(n×k)` is the utility matrix of attacker when target is not under the protection of defender.
`U_attacker_c(n×k)` is the utility matrix of attacker when target is under the protection of defender.
`optimal` is the optimal expected payoff of defender.
`z1` is the optimal strategy of defender.
`target` is the target to be attacked when the defender takes the optimal strategy.

1.3.2 Limit

1. Our algorithm is designed for SSG problem which has multiple attacker types. It could not be applied for SSG whose attacker type is only 1.
2. Our algorithm needs to solve n^k LP problems each call. Hence, when dimension is high, the runtime of MLP is pretty high.

1.4 Example

We will give an example from [1]. The utility matrix is as follows:

Table 6: Utility Matrix of defender

Protected Target	Attack 1	Attack 2
Target 1	0	-0.5
Target 2	-1	0

Table 7: Payoff Matrix of defender (target is not covered by defender)

Attacked Target	Target 1	Target 2
Defender	-1	-0.5

Table 8: Payoff Matrix of defender (target is covered by defender)

Attacked Target	Target 1	Target 2
Defender	0	0

Table 9: Payoff Matrix of attacker (target is not covered by defender)

Attacked Target	Attacker 1	Attacker 2
Target 1	0.5	1
Target 2	1	0.5

Table 10: Payoff Matrix of attacker (target is covered by defender)

Attacked Target	Attacker 1	Attacker 2
Target 1	0	0
Target 2	0	0

Decision Boundary:

$$0.5(1 - p_1) = 1 - p_2$$

$$0.5(1 - p_2) = 1 - p_1$$

The graph below shows the strategy space divided by two different attackers. Our algorithm will compute optimal strategy for each region and choose an optimal strategy.
According to 4, the region could be divided by hyperplane: $0.5(1 - p_1) = 1 - p_2$ and $0.5(1 - p_2) = 1 - p_1$. We could represent best-response region of our example in the graph below. The upper red

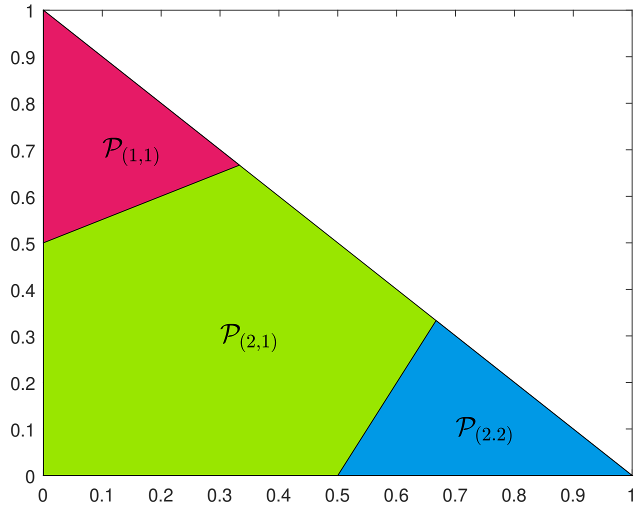


Figure 1: Best Response Region Example

region is $\mathcal{P}_1^1$, means target to be attacked in this region for attacker 1 and 2 is target 1 and 1. The green region in the middle is $\mathcal{P}_2^1$, means target to be attacked in this region for attacker 1 and 2 is target 2 and 1. and the region in the bottom is $\mathcal{P}_2^2$, means target to be attacked in this region for attacker 1 and 2 is target 2 and 2.

```
U2_u=[0.5 1;1 0.5];
U2_c = [0 0; 0 0];
U1_c=[0 0];
U1_u=[-0.5 -1];
[optimal,z1,target] = MLP(U1_c,U1_u,U2_c,U2_u)
```

Then we get the optimal strategy (0.3333,0.6667).

2 Mixed Integer Linear Programming

When it comes to high dimension SSG problem, time cost of solving MLP is pretty high (we have to consider n^k target-attacker corresponding relationships). By the DOBSS algorithm proposed in [3], the optimization problem could be written as a mixed integer linear programming problem. In the [3], the SSG optimization problem could be written as a Mixed-Integer Quadratic Program, it could be represented as follows:

$$\begin{aligned}
\min_{\mathbf{x},\mathbf{q},a} \quad & \sum_{l\in[K]}\sum_{i\in[L]}\sum_{j\in[L]} p^l R^l_{ij} x_i q^l_j \\
\text{s.t.} \quad & \sum_{i\in[L]} x_i = 0 \\
& \sum_{j\in[L]} q^l_j = 1 \\
& 0 \le (a^l - \sum_{i\in[L]} C^l_{ij} x_i) \le (1-q^l_j)M \\
& x_i \in [0,1] \\
& q^l_j \in \{0,1\} \\
& a \in \mathcal{R},
\end{aligned} \tag{2}$$

where $\mathbf{x}$ is the mixed strategy of defender, $[K]$ is the set of attacker types, $[L]$ is the set of defender types, R^l_{ij} is the payoff of defender if target j is attacked by attacker l and defender protects target i. The probability of the l type attacker is $p^l = P_{\alpha_l}$. If $q^l_j = 1$, the target attacked by attacker j is i. To solve this problem by MIQP, the matrix must be positive definite. The matrix does not satisfy such conditions sometimes. Through variables $z^l_{ij} = x_i q^l_j$, we could transform MIQP problem into a MILP problem[3]:

$$\begin{aligned}
\min_{\mathbf{z},a} \quad & \sum_{l\in[K]}\sum_{i\in[L]}\sum_{j\in[L]} p^l R^l_{ij} z^l_{ij} \\
\text{s.t.} \quad & \sum_{i\in[L]}\sum_{j\in[L]} z^l_{ij} = 1 \\
& \sum_{j\in[L]} z^l_{ij} \le 1 \\
& q^l_j \le \sum_{i\in[L]} z^l_{ij} \le 1 \\
& \sum_{j\in[L]} q^l_j = 1 \\
& 0 \le (a^l - \sum_{i\in[L]} C^l_{ij} (\sum_{h\in[L]} z^l_{ih})) \le (1-q^l_j)M \\
& z^l_{ij} \in [0,1] \\
& q^l_j \in \{0,1\} \\
& a \in \mathcal{R}.
\end{aligned} \tag{3}$$

Utility matrix R and C could be represented as follows: $R_{ii} = c^i_d$, $R_{ij} = u^j_d$ $(i \ne j)$, $C^l_{ii} = c^i_l$, $C^l_{ij} = u^j_l$ $(i \ne j)$. Where c^i_d and u^i_d are defender's payoff values when target i (covered and not covered) is attacked, c_j^i, u_j^i is payoff of attacker i under the condition that target j is covered, not covered by the defender.

2.1 MILP

In this section, we developed a code to transform the SSG problem into an MILP problem and solved it using cvx and integer program solvers like Gurobi. Under the condition that possibility of every attacker type is same, we wrote a function code to solve SSG by solving MILP.. The algorithm based on MILP is written as:

```
function [cvx_optval,z,a] = MILP(U_defender_c,U_defender_u,U_attacker_u
    ,U_attacker_c)
```

2.1.1 Input Output

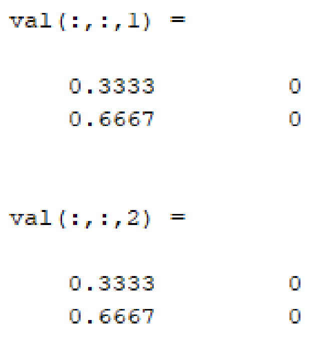
`U_defender_c(1×n)` is the utility matrix of defender when target attacked by attacker is under the protection of defender.
`U_defender_u(1×n)` is the utility matrix of defender when target attacked by attacker is not under the protection of defender.
`U_attacker_u(n×k)` is the utility matrix of attacker when target is not under the protection of defender.
`U_attacker_c(n×k)` is the utility matrix of attacker when target is under the protection of defender.
`cvx_optval` is the abs value of the optimal expected utility.
`z(n×n×k)` is the variable $\mathbf{z}$ in DOBSS algorithm, which we have mentioned above in page 6. Since $\mathbf{z} = \mathbf{xq}$, the first n of `z(n×n×k)` means dimension of mixed strategy of defender, it corresponds to $\mathbf{x}$ in the definition of $\mathbf{z}$ above, the second n and k means the dimension of $\mathbf{q}$, they correspond to optimal target to attack and attacker type in this time round step.

2.1.2 Example

```
U_attacker_c=[0 0; 0 0];
U_attacker_u=[0.5 1;1 0.5];
U_defender_c=[0 0];
U_defender_u=[-0.5 -1];
[cvx_optval,z,a]=MILP(U_defender_c,U_defender_u,U_attacker_u,
    U_attacker_c)
```

. 0- Output is as follows:
`cvx_optval`=0.3333
`a`=[0.3333; 0.6667]
`z` is as follows:

We will explain outputs above to help people understand it easier:
`a(1×k)` is the array of attacker's optimal utility value. The optimal mixed strategy for defender is `[0.3333,0.6667]`
The first dimension of z corresponds to the strategy of defender. $z_j^l = xq_j^l$ corresponds to `z(:,j,l)`



```
val(:,:,1) =

    0.3333         0
    0.6667         0


val(:,:,2) =

    0.3333         0
    0.6667         0
```

Figure 2: z

in the codes. It means that:
1.If target j is the optimal target to attack for attacker l, `z(:.j,l)` is the strategy of defender. (Since $q_j^l = 1$, $z_j^l = \mathbf{x}q_j^l = \mathbf{x}$)
2.If target j is not the optimal target to attack for attacker l, `z(:,j,l)` is zero. (Since $q_j^l = 0$, $z_j^l = \mathbf{x}q_j^l = \mathbf{0}$).
Apply the rules above to the example above, we have:
`z(:,1,1)=[0.3333;0.6667]` means optimal target to attack for attacker 1 is target 1 and the optimal strategy of defender is [0.3333;0.6667].
`z(:,2,1)`=$\mathbf{0}$, means optimal target to attack for attacker 1 is not target 2.
`z(:,1,2)`=[0.3333;0.6667] means optimal target to attack for attacker 2 is target 1 and the optimal strategy of defender is [0.3333;0.6667].
`z(:,2,2)`=$\mathbf{0}$ means optimal target to attack for attacker 2 is not target 2.
From the value of $\mathbf{z}$ above, $q_1^1 = 1$, $q_2^1 = 0$, $q_1^2 = 1$, $q_2^2 = 0$, $\mathbf{x} = [0.3333; 0.6667]$.

2.2 MILP by YALMIP and Gurobi Solver

To solve MILP problem more efficiently, we applied YALMIP and Gurobi solver to improve our algorithm.

Hint

After downloading Gurobi solver, it is necessary to apply for and activate Gurobi's license. A free academic license can be obtained by applying in the school through Sustech server or connecting to the Sustech VPN if you are not in Sustech.

```
function [optobj,optz,optq] = MILP_yalmip_gurobi(U_defender_c,
    U_defender_u,U_attacker_u,U_attacker_c,OfflineFrequency)
```

2.2.1 Input Output

`U_defender_c(1×n)` is the utility matrix of defender when target attacked by attacker is under the protection of defender.
`U_defender_u(1×n)` is the utility matrix of defender when target attacked by attacker is not under the protection of defender.

`U_attacker_u(n× k)` is the utility matrix of attacker when target is not under the protection of defender.
`U_attacker_c(n× k)` is the utility matrix of attacker when target is under the protection of defender.
`OfflineFrequency(1× k)` is the frequency of attackers of different attacker types.
`optobj` is the abs value of the optimal expected utility.
`optz(n×n×k)` is the optimal variable **z** in DOBSS algorithm.
`optq(n×k)` is optimal **q** in DOBSS algorithm.

3 Online Learning

3.1 Problem Formulation

Consider an attacker sequence from $[K]$, where the attacker will attack targets according to the attacker sequence. In repeated SSG, the attacker's sequence is unknown to the defender, the defender can utilize information from previous rounds to formulate strategies for the current round. Therefore, at every time step t we can try to conduct online combinatorial optimization to compute an optimal mixed strategy.

3.2 Follow the Leader

Follow the Leader (FTL) algorithm is a simple and effective online learning algorithm, which selects the action that has achieved the best result at each time step. Specifically, FTL algorithm selects an action based on the current data and model parameters, updates the model parameters, and selects the next action based on the updated parameters. Applying FTL, our algorithm is as follows:

1. Conduct online learning at each round. In each round t, we know randomly generated $\mathbf{c}_0(t)$, and $c_1, \ldots, c_{t-1}$ in last $t-1$ rounds. We will "predict" $\mathbf{p}_t$ in round t according to the known information by solving problem below:

$$\textbf{maximize} \sum_{i=1}^{t-1} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}) \rangle$$

   We have $\mathbf{p}_t = \arg\max_{\mathbf{p}\in\mathbb{R}^L} (\sum_{i=1}^{t-1} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}) \rangle)$.

2. Determine best strategy at round t. Having known the sequence of attacker types in last t rounds, the best strategy in round t could be determined. Define $\mathbf{p}_t^*$ as the best strategy subject last t rounds of games, $\mathbf{p}_t^* = \arg\max_{\mathbf{p}\in\mathbb{R}^L} \sum\limits_{i=1}^{t} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}) \rangle$.

3. Compute regret at round t. As the best strategy and "predicted" best strategy are determined, the regret at round t could be written as follows: $\text{Regret}(t) = \sum_{i=1}^{t} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}_t^*) \rangle - \sum\limits_{i=1}^{t} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}_i) \rangle$. $\mathbf{p}_t^*$and $\mathbf{p}_t$ can be computed by solving multiple linear programs or a single mixed-integer linear program.

T is the time range, R, C is the utility matrix.

```
Algorithm 1 Follow the Leader
Require: $T, R, C$
1: $\mathbf{p}_T \leftarrow \arg\max \sum_{i=1}^{t-1} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}) \rangle$    ▷ Mixed Integer Linear Program
2: Regret$(T) \leftarrow \sum_{i=1}^{T} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}_T^*) \rangle - \sum_{i=1}^{T} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}_i) \rangle$
Ensure: $\mathbf{p}_T$
```

3.3 Follow the Perturbed Leader

In this section, we will give a detailed introduction of our algorithms which comes from Follow the Perturbed Leader. Firstly, we initialize $\mathbf{c}_0(t)$ at round t. Then we compute the best strategy $\mathbf{p}_t$ at time step t by solving mixed integer programming. After T rounds of games, optimal strategy $\mathbf{p}_T$ could be computed and the optimal strategy in hindsight $\mathbf{p}_t^*$ could also be computed by solving MILP. The cumulative regret at round T could be determined.

1. At the beginning of each round, conduct initialization of c_0. At round t of game, we initialize $\mathbf{c}_0(t) = (c_{0,1}(t), \ldots, c_{0,L}(t))$, where $c_{0,l}(t) \sim \mathcal{U}[0, 2/\epsilon(t)]$ and $\epsilon(t) = \sqrt{2/t}$.

2. Conduct online learning at each round. In each round t, we know randomly generated $\mathbf{c}_0(t)$, and $c_1, \ldots, c_{t-1}$ in last $t-1$ rounds. We will "predict" $\mathbf{p}_t$ in round t according to the known information by solving problem below:

$$\textbf{maximize} \sum_{i=1}^{t-1} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}) \rangle + \langle \mathbf{c}_0, \mathbf{f}(\mathbf{p}) \rangle.$$

   We have $\mathbf{p}_t = \arg\max_{\mathbf{p}\in\mathbb{R}^L} (\sum_{i=1}^{t-1} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}) \rangle + \langle \mathbf{c}_0, \mathbf{f}(\mathbf{p}) \rangle)$.

3. Determine best strategy at round t. Having known the sequence of attacker types in last t rounds, the best strategy in round t could be determined. Define $\mathbf{p}_t^*$ as the best strategy subject last t rounds of games, $\mathbf{p}_t^* = \arg\max_{\mathbf{p}\in\mathbb{R}^L} \sum\limits_{i=1}^{t} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}) \rangle$.

4. Compute regret at round t. As the best strategy and "predicted" best strategy are determined, the regret at round t could be written as follows: Regret$(t) = \sum_{i=1}^{t} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}_t^*) \rangle - \sum\limits_{i=1}^{t} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}_i) \rangle$. $\mathbf{p}_t^*$and $\mathbf{p}_t$ can be computed by solving multiple linear programs or a single mixed-integer linear program.

```
Algorithm 2 Follow the Perturbed Leader
Require: $T, R, C$
1: $c_{0,i}(T) \leftarrow \mathcal{U}[0, 2/\epsilon(T)]$
2: $\mathbf{c}_0(T) \leftarrow (c_{0,1}(T), \ldots, c_{0,L}(T))$
3: $\mathbf{p}_T^* \leftarrow \arg\max \sum_{i=1}^{t-1} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}) \rangle + \langle \mathbf{c}_0, \mathbf{f}(\mathbf{p}) \rangle$    ▷ Mixed Integer Linear Program
4: Regret$(T) \leftarrow \sum_{i=1}^{T} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}_T^*) \rangle - \sum_{i=1}^{T} \langle \mathbf{c}_i, \mathbf{f}(\mathbf{p}_i) \rangle$
Ensure: $\mathbf{p}_T$
```

4 Convergence Test

4.1 Sublinear Property and Test Output

We designed a repeated SSG problem with dimension of 6×5. Utility matrix and attacker are generated randomly. We applied FTL and FTPL to solve this repeated SSG problem for 100 time steps. We compared the regret of FTL and FTPL with our theoretical upper bound $2\sqrt{t}$ (come from *Theorem 3.3* in [2]) and confirmed the sublinear property of the regret in this random-generated example. Then we compared the regret between two different online learning algorithms. Here is the output of regret of FTPL and theoretical bound $2\sqrt{t}$:

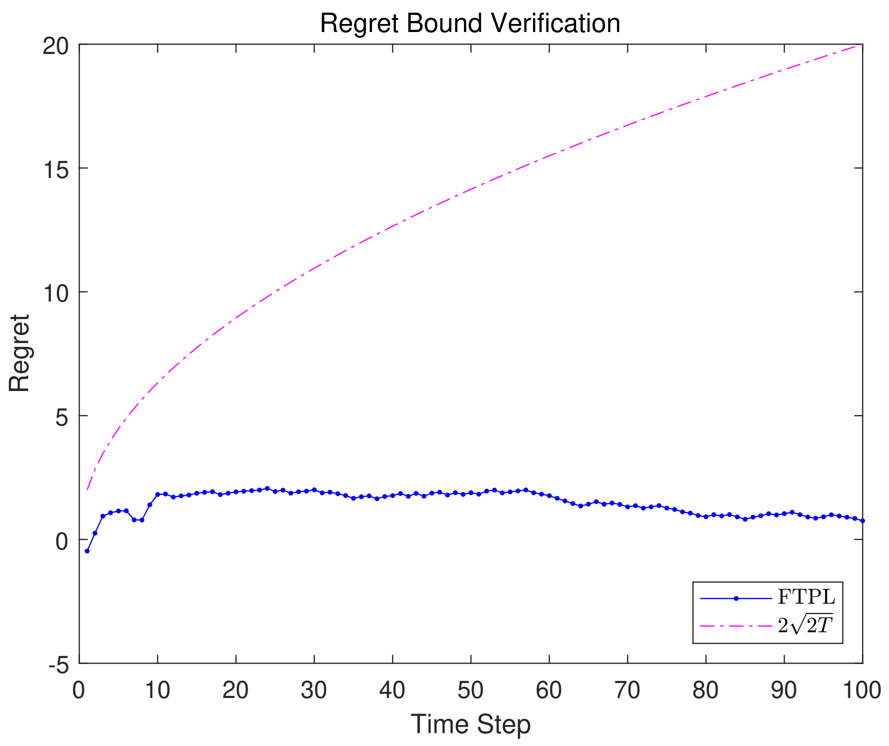


Figure 3:

4.2 Perform Simulation

The code is attached in the file `Sublinear Property Test.m`. In this file, we could compute regret of FTPL and compare it with our theoretical regret bound. The step is as follows:

4.2.1 Initialization

In this step, we will initialize our SSG problem with dimension 6×5, including generating utility matrix randomly, generating attacker sequence randomly, and generating $\mathbf{c}_0$ sequence.

```
n=6;
```

```
k=5;
U1u_Array=zeros(1,n);
U1c_Array=zeros(1,n);
U2u_Array=zeros(n,k);
U2c_Array=zeros(n,k);
U2_u=round(rand(n,k),2);
U2_c = zeros(n,k);
U1_u=round(unifrnd(-1,0,[1,n]),2);
U1_c=zeros(1,n);
for inx=1:n
    for iny=1:k
    U2_c(inx,iny)=unifrnd(-U2_u(inx,iny),U2_u(inx,iny));
    end
    U1_c(inx)=unifrnd(0,-U1_u(inx));
end
U1_c=round(U1_c,2);
U2_c=round(U2_c,2);

t=1000;
Strategy_Table=zeros(n,t,2);
Strategy_Table1=zeros(n,n,k,t,2);
AttackerSequence=randi(k,1,t);
SequecneFrequency=zeros(k,t);

for i=1:t
    for j=1:k
        for m=1:i
            if AttackerSequence(m)==j
                SequecneFrequency(j,i)=SequecneFrequency(j,i)+1/i;
            end
        end
    end
end

C0_sequence=zeros(n,t);
for i=2:t
    C0_sequence(:,i)=rand(n,1)*(sqrt(2*i)/(i-1));
end
C0_sequence(:,1)=rand(n,1);
```

4.2.2 Online Learning

Limited by time cost, we only computed first 100 steps and saved their optimal strategies. The code is as follows:

```
for i=1:100
    [optimal,z11,target] = MILP_yalmip_gurobi_C0(U1_c,U1_u,U2_u,U2_c,
       SequecneFrequency(:,i),C0_sequence(:,i));
    Strategy_Table(:,i,2)=sum(z11(:,:,1),2);
    Strategy_Table1(:,:,:,i,2)=z11;
end
```

4.2.3 Compute Expected Utility

In this step, we will compute the expected utility of FTPL in each round and save its value in `Regret table(:,2)`. We will also compute the optimal offline expected utility in each round and save it in `Regret table(:,1)`. The code is as follows:

```
[R,C] = PayoffMatrix(-U1_c, -U1_u,U2_c,U2_u);
Regret_table=zeros(t,2);
Regret_table(1,2)=1;

for h=1:1
    for i=1:n
    for j=1:n
        for l=1:k
Regret_table(h,1)=Regret_table(h,1)+h*SequecneFrequency(l,h)*(R(i,j))*
   Strategy_Table1(i,j,l,h,1);

        end
    end
end

end
for h=2:t
    for i=1:n
    for j=1:n
        for l=1:k
Regret_table(h,1)=Regret_table(h,1)+h*SequecneFrequency(l,h)*(R(i,j))*
   Strategy_Table1(i,j,l,h,1);
Regret_table(h,2)=Regret_table(h,2)+SequecneFrequency(l,h)*(R(i,j))*
   Strategy_Table1(i,j,l,h-1,2);
        end
    end
end
Regret_table(h,2)=Regret_table(h-1,2)+Regret_table(h,2);
end
```

4.2.4 Compute Regret

In this step, we will compute the regret of FTPL.

```
Regret=zeros(100,1);
for i=1:100
    Regret(i)=Regret_table(i,2)-Regret_table(i,1)
end
```

4.2.5 Regret Upper Bound

```
x=[1:100];
bound=2*sqrt(x);
```

4.2.6 FTPL vs Upper Bound

```
plot(x,Regret,'b.-', x,bound,'m-.');
title('Regret Bound Verification')
xlabel('Time Step')
ylabel('Regret')
legend('FTPL','$2\sqrt{2T}$','interpreter','latex','Location','
    southeast')
```

5 Robustness comparison

In some special cases, FTL has bad performance. With the idea of [3], we designed a special example to compare the robustness of FTL and FTPL in pages 13-14 in this thesis. The output is as follows:

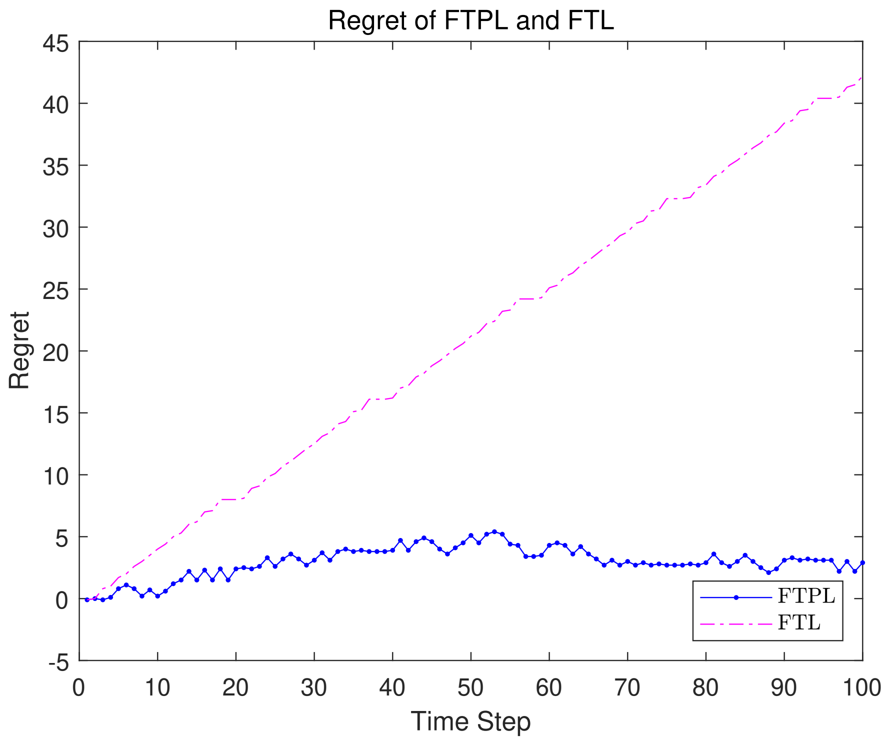


Figure 4:

The cumulative regret of FTL is linear, and the cumulative regret of FTPL is sublinear.
To realize this illustration through programming, we wrote a program of this experiment contained in the file `FTPL vs FTL.m`. The program could be divided into followed steps:

5.1 Initialization

In this step, we initialized the utility matrix and other parameters in our example. The code is as follows:

```
 U2_u=[1 0 ; 0 1];
 U2_c = [0.5 0;  0 0.5];
 U1_c=[0.9 1 ];
 U1_u=[0 0];
 U_attacker_c=U2_c;
 U_attacker_u=U2_u;
 U_defender_c=U1_c;
 U_defender_u=U1_u;

n=size(U_defender_c,2);
k=size(U_attacker_u,2);
t=100;
Strategy_Table=zeros(n,t,2);
Strategy_Table1=zeros(n,n,k,t,2);
```

5.2 Generate Attacker Sequence

By thesis, when the attacker attacks in a specific sequence, FTL will receieve linear regret. We will generate this specific attacker sequence in this section. The code is as follows:

```
AttackerSequence=randi(2,1,t);
for i=1:500
    AttackerSequence(i)=2-mod(mod(i,19),2);
    if mod(i,19)==0
        AttackerSequence(i)=1;
    end
end
```

5.3 Determine Optimal Target To Attack

Since we have known the optimal target to attack in each round when we design the attacker sequence. So we directly assign value to variable `Attackedlist`. `Attackedlist(i,j)=1` means target i is attacked in round j. The code is as follows:

```
Attackedlist=zeros(k,t);
for i=1:100
    if AttackerSequence(i)==1
        Attackedlist(1,i)=1;
    end

    if AttackerSequence(i)==2
        Attackedlist(2,i)=1;
    end
end
```

5.4 Frequency of Attacker Type

When the probability of attacker types is not the same, we will apply function `MILP_yalmip_gurobi` and `MILP_yalmip_gurobi_C0` to solve SSGs in FTL and FTPL. The code is as follows:

```
SequecneFrequency=zeros(k,t);
for i=1:t
    for j=1:n
        for l=1:i
            if AttackerSequence(l)==j
                SequecneFrequency(j,i)=SequecneFrequency(j,i)+1/i;
            end
        end
    end
end
```

5.5 Generate C0 for FTPL

By Follow the Perturbed Leader, we need to initialize $\mathbf{c}_0(t)$ in each round. The code in this step is to generate $\mathbf{c}_0$ sequence:

```
C0_sequence=zeros(n,t);
for i=2:t
    C0_sequence(:,i)=rand(n,1)*(sqrt(2*i)/(i-1));
end
C0_sequence(:,1)=rand(n,1);
```

5.6 Solution of FTL and FTPL

In this section, we solved SSGs at each time step in online learning. We separately saved optimal strategy of FTL and FTPL at time step i in variable `Strategy_Table(:,i,1)` and `Strategy_Table(:,i,2)`. To compute expected utility at each round, we saved variable z of FTL and FTPL at time step i in `Strategy_Table1(:,:,:,i,1)` and `Strategy_Table1(:,:,:,i,2)`.

```
for i=1:100
    [optimal,z11,target] = MILP_yalmip_gurobi(U1_c,U1_u,U2_u,U2_c,
       SequecneFrequency(:,i));


    Strategy_Table(:,i,1)=sum(z11(:,:,1),2);
    Strategy_Table1(:,:,:,i,1)=z11;

end

for i=1:100
```

```
    [optimal,z11,target] = MILP_yalmip_gurobi_C0(U1_c,U1_u,U2_u,U2_c,
       SequecneFrequency(:,i),C0_sequence(:,i));

    Strategy_Table(:,i,2)=sum(z11(:,:,1),2);
    Strategy_Table1(:,:,:,i,2)=z11;

end
```

5.7 Utility of FTL, FTPL and Offline Optimal Strategy

In this section, we will apply the optimal strategy of FTL and FTPL to compute the expected utility of strategy provided by online learning. And we will compute the expected utility to help us get regret of FTL and FTPL in each round. Follows are variable descriptions of our code:
1. `Expected_Utility_table_FTPL(h)` corresponds to expected utility of optimal strategy computed by FTPL in time step h according to information from last h-1 time steps.
2. `Expected_Utility_table_FTL(h)` corresponds to expected utility of optimal strategy computed by FTL in time step h according to information from last h-1 time steps.
3. `Optimal_Offline_Utility_Sum (h)` corresponds to expected utility of offline optimal strategy in time step h according to information from h time steps.

```
[R,C] = PayoffMatrix(U1_c, U1_u,U2_c,U2_u);
Expected_Utility_table_FTPL=zeros(100,1);
Expected_Utility_table_FTL=zeros(100,1);
Optimal_Offline_Utility_Sum=zeros(100,1);
Expected_Utility_table_FTPL(1,1)=-randi(1);
for h=2:100
    for i=1:n
    for j=1:n
        for l=1:k
Expected_Utility_table_FTPL(h)=Expected_Utility_table_FTPL(h)+
   Attackedlist(i,h)*(R(j,i))*Strategy_Table1(i,j,l,h-1,2);
Optimal_Offline_Utility_Sum(h)=Optimal_Offline_Utility_Sum(h)+h*
   SequecneFrequency(l,h)*(R(i,j))*Strategy_Table1(i,j,l,h,1);
        end
    end
    end
Expected_Utility_table_FTPL(h)=Expected_Utility_table_FTPL(h-1)+
   Expected_Utility_table_FTPL(h);
end
Expected_Utility_table_FTL=zeros(100,1);
for h=2:100
    for i=1:n
    for j=1:n
        for l=1:k
Expected_Utility_table_FTL(h)=Expected_Utility_table_FTL(h)+
   Attackedlist(i,h)*(R(j,i))*Strategy_Table1(i,j,l,h,1);
        end
    end
    end
Expected_Utility_table_FTL(h)=Expected_Utility_table_FTL(h-1)+
   Expected_Utility_table_FTL(h);
end
```

5.8 Regret of FTL and FTPL

In this section, we will compute the regret of FTL and FTPL and save them in `Regret_FTL` and `Regret_FTPL` separately.

```
Regret_FTL=zeros(100,1);
for i=1:100
Regret_FTL(i)=Optimal_Offline_Utility_Sum(i)-Expected_Utility_table_FTL
   (i)
end
Regret_FTPL=zeros(100,1);
for i=1:100
Regret_FTPL(i)=Optimal_Offline_Utility_Sum(i)-
   Expected_Utility_table_FTPL(i)
end
```

5.9 FTL vs FTPL

We plotted the regret of FTL and FTPL.

```
plot(x,Regret_FTPL,'b.-', x,Regret_FTL,'m-.');
title('Regret of FTPL and FTL')
xlabel('Time Step')
ylabel('Regret')
legend('FTPL','FTL','interpreter','latex','Location','southeast')
```

References

[1] Maria-Florina Balcan, Avrim Blum, Nika Haghtalab, and Ariel D Procaccia. Commitment without regrets: Online learning in stackelberg security games. In *Proceedings of the sixteenth ACM conference on economics and computation*, pages 61–78, 2015.

[2] Adam Kalai and Santosh Vempala. Efficient algorithms for online decision problems. *Journal of Computer and System Sciences*, 71(3):291–307, 2005.

[3] Praveen Paruchuri, Jonathan P Pearce, Janusz Marecki, Milind Tambe, Fernando Ordonez, and Sarit Kraus. Playing games for security: An efficient exact algorithm for solving bayesian stackelberg games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pages 895–902, 2008.