Matrix games - Optimal Strategies

Nguyen Huu Hoan

K65 Mathematics - Hanoi University of Science

June 7, 2022

Contents

1. Matrix games

A **matrix game** is a *two-person game* defined as follows:
First, each player independently selects an action from a finite set of choices. (The two sets of choices need not be the same).
Then both reveal to each other their choice.
If we let i and j denote the first and the second player's choice respectively, then the rules of the game stipulate that the first player will pay the second player a_{ij} dollars.
The array of possible payments/the game's rules: $A = [a_{ij}]$ is known by both players before the game starts.
We will refer to the first player as the *row player* and the second player as the *column player*.
Since the players has only a finite number of actions to choose, we can enumerate actions of the first player as integers from 1 to m and similarly, actions of the second player as integers from 1 to n.
Then, the above array is a $m \times n$ matrix: $A = [a_{ij}]_{m \times n}$, we shall call it the *rule-matrix*.

Example 1: Paper - Scissors - Rock.

This is a two-person game in which on the count of three each player declares either Paper, Scissors, or Rock.

If both players declare the same object, then the round is a draw.

Otherwise, the winner/loser is decided by the following:

- Paper loses to Scissors.
- Scissors loses to Rock.
- Rock loses to Paper.

Then, if we enumerate the actions of declaring Paper, Scissors, Rock as $1, 2, 3$, respectively, then the payoff matrix is:

$$\begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 1 & -1 \\ -1 & 0 & 1 \\ 1 & -1 & 0 \end{bmatrix}$$

With this game, neither player has a deterministic winning strategy.
Instead, they should randomize their choices in each round.
In fact, due to symmetry, we can guess that both players should make each of the three choices with equal probability in each round, and that this game is *fair*.

Example 2: Paper - Scissors - Rock*

$$\begin{bmatrix} 0 & 1 & -2 \\ -3 & 0 & 4 \\ 5 & -6 & 0 \end{bmatrix}$$

How should each player play?
How can we determine the *fairness* of this game?

Consider the row player. By a *randomized strategy*, we mean that, at each round of the game, it appears (from the column player's viewpoint) that the row player is making their choices at random according to some fixed probability distribution. Let y_i denote the probability that the row player selects action i. The vector y composed of these probability satisfy: $y \geq 0$ and $\mathbb{1}^T y = 1$.
Similarly, let x_j denote the probability that the column player selects action j. The vector x composed of these probability satisfy: $x \geq 0$ and $\mathbb{1}^T x = 1$.
With y and x as *randomized strategy* for the row and column player respectively, we compute the expected result for each round as follows. The set of possible outcomes is the set of pairs (i, j), with $i \in \{1, ..., m\}$, $j \in \{1, ..., n\}$, for outcome (i, j) the payoff is a_{ij}, and this outcome has probability of $y_i x_j$. Therefore, the expected result is:

$$\sum_{i,j} y_i a_{ij} x_j = y^T A x$$

2. Optimal strategies

Suppose we have a matrix game with the corresponding rule-matrix A, and also that the column player uses strategy x. Then the row player's best strategy is y^* that achieves the following minimum:

$$(1) \qquad \begin{aligned} \min \quad & y^T A x \\ \text{s.t} \quad & \mathbb{1}^T y = 1 \\ & y \geq 0 \end{aligned}$$

We know that (1) has a basic optimal solution (simplex method).

Example 3: Take the rule-matrix A of Example 2 and assume $x = \begin{bmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{bmatrix}^T$. Then $Ax = \begin{bmatrix} -\frac{1}{3} & \frac{1}{3} & -\frac{1}{3} \end{bmatrix}^T$ and so the best choice for the row player is to select either $i = 1$ or $i = 3$ or any combination of the two. That is, a basic optimal solution is $y^* = [1 \quad 0 \quad 0]^T$ (it is not unique).

2. Optimal strategies

Since for any given x the row player *will* adopt y to achieve the minimum in (1), it is natural that the column player should use a strategy x^* that attains the following maximum:

$$\begin{aligned} &\max_x \ \min_y \ y^T Ax \\ (2) \qquad &\text{s.t} \quad \mathbb{1}^T x = 1 \\ &\qquad\quad x \geq 0 \end{aligned}$$

We can reformulate (2) as a linear programming problem. Indeed, notice that: $\min_y \ y^T Ax = \min_i \ e_i^T Ax$. Hence, we can rewrite (2) as:

$$\begin{aligned} &\max \ \min_i \ e_i^T Ax \\ &\text{s.t} \quad \mathbb{1}^T x = 1 \quad \text{and} \quad x \geq 0 \end{aligned}$$

Here we let v be a new variable representing a lower bound on the $e_i^T Ax$, then we turn the above problem into a linear program:

$$\begin{aligned} &\max \ v \\ (3) \qquad &\text{s.t} \quad v\mathbb{1} - Ax \leq 0 \\ &\qquad\quad \mathbb{1}^T x = 1 \\ &\qquad\quad x \geq 0 \end{aligned}$$

2. Optimal strategies

Now, by symmetry, the row player wants to use a strategy y^* that achieve the following minimum:

$$\begin{aligned} \min_{y} \quad & \max_{x} \; y^T A x \\ \text{s.t} \quad & \mathbb{1}^T y = 1 \\ & y \geq 0 \end{aligned}$$

which similarly, can be reformulated as the following linear program:

$$\begin{aligned} \min \quad & u \\ \text{s.t} \quad & u\mathbb{1} - A^T y \geq 0 \\ & \mathbb{1}^T y = 1 \\ & y \geq 0 \end{aligned} \tag{4}$$

3. The Minimax Theorem

So far, we have reduced the task of finding optimal strategies x^* and y^* for the two players to the solution of the linear programs (3) and (4), which we know how to solve (e.g: simplex method).

Intuitively, we see that in a matrix game, the minimum expected *loss* of the row player should equal the maximum expected *gain* of the column player.

This is proved in the next theorem called **The Minimax Theorem**.

Theorem (The Minimax Theorem)

There exist vectors x^* and y^* for which

$$\max_x \ {y^*}^T A x = \min_y \ y^T A x^*$$

Theorem (The Minimax Theorem)

There exist vectors x^* and y^* for which

$$\max_x \ {y^*}^T A x = \min_y \ y^T A x^*$$

Proof.

Consider linear programs (3),(4). We see that both has a basic feasible solution, therefore, via simplex method, we can find basic optimal solutions $[x^* \quad v^*]^T$ and $[y^* \quad u^*]^T$ for (3) and (4).

We also notice that (3) and (4) are dual to each other. Hence, due to the **Strong duality theorem**, we have $v^* = u^*$.

Furthermore,

$$v^* = \min_i \ e_i^T A x^* = \min_y \ y^T A x^*;$$

$$u^* = \max_j \ e_j^T A^T y^* = \max_x \ x^T A^T y^* = \max_x \ {y^*}^T A x.$$

Hence, we conclude the proof. □

The common optimal value $\xi = v^* = u^*$ of (3) and (4) is called the *value* of the game.

From the **Minimax Theorem**, we see that the row player can assure not losing more than ξ dollars per round on average by using strategy y^*, and the column player can assure not winning less than ξ dollars per round on average by using strategy x^*.

In a way, the value ξ can be consider as the *fair-o-meter* for the matrix game. Obviously, when $\xi = 0$, the matrix game is fair.

Example 2:(continued)

Here, the corresponding (3) is:

$$\begin{aligned} \max\ & v \\ \text{s.t}\quad & -x_2 + 2x_3 + v \le 0 \\ & 3x_1 - 4x_3 + v \le 0 \\ & -5x_1 + 6x_2 + v \le 0 \\ & x_1 + x_2 + x_3 = 1 \\ & x_1, x_2, x_3 \ge 0 \end{aligned}$$

Optimal value:

$$\xi = -\frac{16}{102} = -0.15686275$$

Optimal strategy:

$$x^* = \begin{bmatrix} \frac{40}{102} & \frac{36}{102} & \frac{26}{102} \end{bmatrix}^T$$

From this, we can conclude that this game is more favorable for the row player.

4. The Simplified Poker Game

Simplified poker involves two players, A and B, and a deck having three cards $0, 1$ and 2.
At the start of a round, each player bet add \$1 to the "piggy" and is dealt one card from the deck.
A bidding session follows in which each player in turn, starting with A, either *bets* and adds \$1 to the piggy or *passes*. This session ends when:

a bet is followed by a bet,
a pass is followed by a pass, or
a bet is followed by a pass.

In the first two cases, the winner of the round is decided by comparing cards, and the piggy goes to the player with the higher card. In the third case, bet followed by pass, the player who bet wins the round.

4. The Simplified Poker Game

We list the possbile scenarios as follow:

A passes,	B passes:		\$1 to holder of higher card
A passes,	B bets,	A passes:	\$1 to B
A passes,	B bets,	A bets:	\$2 to holder of higher card
A bets,	B passes:		\$1 to A
A bets,	B bets:		\$2 to holder of higher card

After being dealt a card, player A will have 3 lines:

0. Pass. If B bets, pass again.
1. Pass. If B bets, bet.
2. Bet.

and player B will have 4 lines:

0. Pass no matter what.
1. If A passes, pass, but if A bets, bet.
2. If A passes, bet, but if A bets, pass.
3. Bet no matter what.

4. The Simplified Poker Game

A pure strategy is a statement of what line of betting a player intends to follow for each possible card that the player is dealt. We can denote them by (y_0, y_1, y_2), where y_i is the line that the player will use when being dealt card i.
Given a pure strategy for both players, one can compute the expected payment per round from A to B. For example, A chooses $(2, 0, 1)$ and B chooses $(2, 1, 3)$, we analyze as follows:

A	B	betting session			A to B
0	1	A bets,	B bets		2
0	2	A bets,	B bets		2
1	0	A passes,	B bets,	A passes	1
1	2	A passes,	B bets,	A passes	1
2	0	A passes,	B bets,	A bets	-2
2	1	A passes,	B passes		-1

Since the probability for each possibility is $1/6$, we conclude that the expected payment per round from A to B is 0.5 .

4. The Simplified Poker Game

We will now code our way to find the *fair-o-meter* of this game, and the optimal strategies for both players.

```
217  #PokerThingy
218  #paymentMatrix
219  paymentMatrix = np.zeros((6, 12), dtype = np.int64)
220  paymentMatrix[0, :] = np.array([1,1,1,1,1,1,2,2,-1,2,-1,2])
221  paymentMatrix[1, :] = np.array([1,1,1,1,1,1,2,2,-1,2,-1,2])
222  paymentMatrix[2, :] = np.array([-1,-1,1,1,-1,-1,-2,-2,-1,-2,-1,-2])
223  paymentMatrix[3, :] = np.array([1,1,1,1,1,1,2,2,-1,2,-1,2])
224  paymentMatrix[4, :] = np.array([-1,-1,1,1,-1,-1,-2,-2,-1,-2,-1,-2])
225  paymentMatrix[5, :] = np.array([-1,-1,1,1,-1,-1,-2,-2,-1,-2,-1,-2])
226
227  cardDealt = [(0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1)]
228
229  def linesChosen(x, y):
230      return 4*x + y
231
232  def betSessionResult(a, x, y):
233      return paymentMatrix[a, linesChosen(x, y)]
234
235  def expectedPayment(u, v):
236      result = 0
237      for k in range(6):
238          cards = cardDealt[k]
239          result += betSessionResult(k, u[cards[0]], v[cards[1]])
240      return result/6
241
242  expectedPaymentMatrix = np.zeros((27, 64))
243  for i in range(27):
244      for j in range(64):
245          u = Lnarize(i, 3, 3)
246          v = Lnarize(j, 3, 4)
247          expectedPaymentMatrix[i, j] = expectedPayment(u, v)
```

4. The Simplified Poker Game

```
pokerModelB = gp.Model()

x = pokerModelB.addMVar(shape = (64, 1), lb = 0, ub = 1, vtype = GRB.CONTINUOUS)
v = pokerModelB.addVar(lb = -GRB.INFINITY, ub = GRB.INFINITY, vtype = GRB.CONTINUOUS)

pokerModelB.addConstrs(-sum(expectedPaymentMatrix[i, j] * x[j, 0] for j in range(64)) + v <= 0 for i in range(27))
pokerModelB.addConstr(x.sum() == 1)

pokerModelB.setObjective(v, sense = GRB.MAXIMIZE)
pokerModelB.optimize()

print('-'*50)
print(f'optimal value: {pokerModelB.objval}')
stratArrayB = x.x
rowIndices, columnIndices = stratArrayB.nonzero()
usedStratB = [Lnarize(rowIndices[i], 3, 4) for i in range(len(rowIndices))]
print(f'strategy: {usedStratB}')
print(f'..... {x[rowIndices, 0].x}')
```

Let Gurobi solve our linear program, we obtain:

Optimal value: $5/9$;

Optimal strategy for player B: $[(0,0,3),(0,1,3),(2,0,3)]$.

$$[\ 1/3\ ,\ 1/3\ ,\ 1/3\ ]$$

We can described player B's optimal strategy as:

when holding 0, mix lines 0 and 2 in $2:1$ proportion;

when holding 1, mix lines 0 and 1 in $2:1$ proportion;

when holding 2, use line 3.

4. The Simplified Poker Game

```
#pokerModelA
pokerModelA = gp.Model()

y = pokerModelA.addMVar(shape = (27, 1), lb = 0, ub = 1, vtype = GRB.CONTINUOUS)
u = pokerModelA.addVar(lb = -GRB.INFINITY, ub = GRB.INFINITY, vtype = GRB.CONTINUOUS)

pokerModelA.addConstrs(-sum(expectedPaymentMatrix[i, j] * y[i, 0] for i in range(27)) + u >= 0 for j in range(64))
pokerModelA.addConstr(y.sum() == 1)

pokerModelA.setObjective(u, sense = GRB.MINIMIZE)

pokerModelA.optimize()

print('-'*50)
print(f'optimal value: {pokerModelA.objval}')
stratArrayA = y.x
rowIndices, columnIndices = stratArrayA.nonzero()
usedStratA = [Lnarize(rowIndices[i], 3, 3) for i in range(len(rowIndices))]
print(f'strategy: {usedStratA}')
print(f'..... {y[rowIndices, 0].x}')
```

Optimal value: $5/9$;

Optimal strategy for player A: $[(0,0,1),(0,1,2),(2,1,1)]$.

$$[\ 5/9\ ,\ 1/3\ ,\ 1/9\ ]$$

We can described player A's optimal strategy as:

when holding 0, mix lines 0 and 2 in $8:1$ proportion;

when holding 1, mix lines 0 and 1 in $5:4$ proportion;

when holding 2, mix lines 1 and 2 in $2:1$ proportion.

We also computed the *fair-o-meter* of this game: $\xi = 5/9$.

Hence, we conclude that this game is more favorable for player B.

5. The Find-the-square Game

The game involves 2 sides with 3 players: side W consists of player A, and side P consists of players B and C.
There is a board of size $1 \times n$ and n identical two-side coins.
At the start of a round, A choose a square on the board and set n coins on n squares, each square has a coin, *Heads*(H) or *Tails*(T) as she wishes.
Then A let only B knows about the chosen square and the coins status.
Then B has to turn only one coin from H to T or T to H.
Then C has to find which square was chosen. If C finds it, B and C wins. If C does not, A wins.
Questions:
1) Assume A choose the square and the coins status with the same probability for all possible choices, what is the best probability for B and C to win with a strategy?
2) What is the *fair-o-meter* of this game? What are the optimal strategies for both sides?

5. The Find-the-square Game

First, we will model this game:

We denote the n squares by $0, 1, ..., n-1$. A coin status is either 0(H) or 1(T). Hence, a status for n coins is denoted by a $1 \times n$ array of $0s$ and $1s$. For example: HHT $\leftrightarrow [0, 0, 1]$.

We describe a pure strategy for B and C as follows.

For the action of turn a coin status from H to T or from T to H of player B, we consider the graph $G(V, E)$, where each vertex of V is a status of n coins, and two vertices v_1 and v_2 are adjacent if v_1, v_2 differ from one another in only one position.

A pure strategy for B and C is the same as a way of coloring n colors $\{0, 1, ..., n-1\}$ for the graph $G(V, E)$. From the n coins status - vertex $v \in V$ received from A, B tries to tell C which square to choose by turn the status of one in n coins - move to a vertex $v' \in V$ that is adjacent to v. The color i of v' tells C to choose square i.

5. The Find-the-square Game

We will now tackle *Question 1* for some small n, which can now be rephrased as: Each vertex $v \in V$ have a number r_v indicates how many different colors are used for the vertices adjacent to v. What is the maximum of $\sum_{v \in V} r_v$?

We will model this to a LP as follows.
Each vertex $v \in V$ corresponds to a binary array of length n, which can be considered as a binary number. Hence, we can enumerate v by this number but in decimal system.
To find the vertices adjacent to v, we simply take the binary array correspond to v, and for position i in this array, turn it from 0 to 1 or from 1 to 0 to obtain the binary array correspond to one of the adjacent vertices to v.

Example: $n = 3$
vertex $3 \leftrightarrow$ vertex $[0, 1, 1] \rightarrow \{[0, 1, 0], [0, 0, 1], [1, 1, 1]\} \leftrightarrow \{2, 1, 7\}$

Therefore, we have created a mapping f that take $v \in V$ to $f(v)$ the set of adjacent vertices of v.

5. The Find-the-square Game

Color of vertex i:

Variables: $p_{ij} = \begin{cases} 1, \text{ if vertex } i \text{ has color } j \\ 0, \text{ if otherwise} \end{cases}, \quad \begin{matrix} i = \overline{0, 2^n - 1} \\ j = \overline{0, n-1} \end{matrix}$

Constrains: $\sum_{j=0}^{n-1} p_{ij} = 1, \qquad i = \overline{0, 2^n - 1}$

In the vertices adjacent to vertex i:

Variables: q_{ij}: number of j colored vertices adjacent to vertex i

Constrains: $q_{ij} = \sum_{k \in f(i)} p_{kj}, \qquad i = \overline{0, 2^n - 1},\ j = \overline{0, n-1}$

Variables: $r_{ij} = \begin{cases} 1, \text{ if } q_{ij} \geq 1 \\ 0, \text{ if } q_{ij} = 0 \end{cases}, \qquad i = \overline{0, 2^n - 1}, j = \overline{0, n-1}$

Constrains: $\begin{cases} r_{ij} - q_{ij} \leq 0 \\ (n-1)\, r_{ij} - q_{ij} \geq 0 \end{cases}, \quad i = \overline{0, 2^n - 1}, j = \overline{0, n-1}$

Objective: max $\sum_{i,j} r_{ij}$

5. The Find-the-square Game

The code for the mapping f:

```python
#binarize a number
def binarize(n, m):
    a = n
    result = []
    for i in range(m):
        if a < 2**(m-1-i):
            result.append(0)
        else:
            result.append(1)
            a = a - 2**(m-1-i)
    return result
```

```python
#edges of the graph
def adjacentVerticesOf(n, m):
    binary = binarize(n, m)
    result = []
    for i in range(m):
        if binary[i] == 0:
            result.append(n + 2**(m-1-i))
        else:
            result.append(n - 2**(m-1-i))
    return result
```

Model for Gurobi:

```python
n = 5
model2 = gp.Model("test2")
pvars = model2.addMVar(shape = (2**n, n), vtype = GRB.BINARY)
qvars = model2.addMVar(shape = (2**n, n), vtype = GRB.INTEGER)
rvars = model2.addMVar(shape = (2**n, n), vtype = GRB.BINARY)
model2.addConstrs(pvars[i, :].sum() == 1 for i in range(2**n))
model2.addConstrs(qvars[i, j] == sum(pvars[k, j] for k in adjacentVerticesOf(i, n))
                  for i in range(2**n) for j in range(n))
model2.addConstrs(rvars[i, j] - qvars[i, j] <= 0 for i in range(2**n)
                  for j in range(n))
model2.addConstrs((n-1)*rvars[i, j] - qvars[i, j] >= 0 for i in range(2**n)
                  for j in range(n))
model2.setObjective(rvars.sum(), sense = GRB.MAXIMIZE)
model2.Params.DisplayInterval = 300
model2.optimize()
```

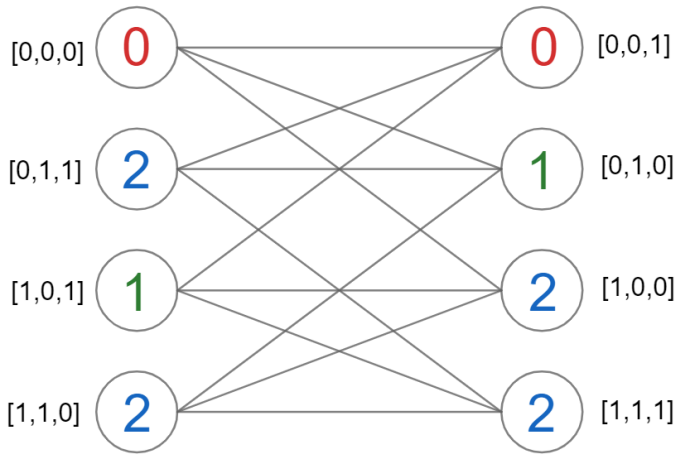
5. The Find-the-square Game

Run the model for $n = 2, 3, 4, 5, 6, 7, 8$, we obtain:

n	opt.value	best found	runtime	probability
2	8	8	0.010	1.000
3	20	20	0.045	0.833
4	64	64	0.024	1.000
5	136	136	0.743	0.850
6	336	336	1142.927	0.875
7	unknown	808	37053	0.902
8	2048	2048	1.675	1.000

From p_{ij}, we can find the optimal strategy. For $n = 3$:

```
[[1. 0. 0.]
 [1. 0. 0.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]
 [0. 1. 0.]
 [0. 0. 1.]
 [0. 0. 1.]]
```



5. The Find-the-square Game

We continue with *Question 2* for $n = 3$. For this, assume:

If A wins, A get 2\$ from the side of B and C.

If B and C wins, A pay 1\$ to the side of B and C.

We need to model the game to LP (3), (4). This is done as follows.

The choices A has are pairs (y, z), where y is the vertex A chooses and z is the square A chooses.

The choices B and C have are ways of coloring the graph $G(V, E)$ using $n = 3$ colors. Each way can be represented by an array of length $2^n = 8$ with each element is in $\{0, 1, ..., n-1\} = \{0, 1, 2\}$.

We need to calculate the *rule-matrix* M for this game. To do this, we first consider:

```
def Lnarize(n, m, l):
    a = n
    result = []
    for i in range(m):
        x = a // l**(m-1-i)
        result.append(x)
        a = a - x * l**(m-1-i)
    return result
```

```
def checkWhoWins(n, x, y, z):
    a = Lnarize(x, 2**n, n)
    B = adjacentVerticesOf(y, n)
    for i in range(n):
        if z == a[(B[i])]:
            return 1
    return -2
```

5. The Find-the-square Game

With these support function, we can calculate the *rule-matrix* M:

```
n = 3
M = np.zeros((n* (2**n), n**(2**n)), dtype = np.int64)
for i in range(n * (2**n)):
    for j in range(n**(2**n)):
        M[i, j] = checkWhoWins(n, j, i - (2**n) * (i//(2**n)), i//(2**n))
```

Now we are ready to model the game to LP (3):

```
testModel = gp.Model()

x = testModel.addMVar(shape = (n**(2**n), 1), lb = 0, ub = 1, vtype = GRB.CONTINUOUS)
v = testModel.addVar(lb = -GRB.INFINITY, ub = GRB.INFINITY, vtype = GRB.CONTINUOUS)

testModel.addConstrs(-sum(M[i, j] * x[j, 0] for j in range(n**(2**n))) + v <= 0
                     for i in range(n*(2**n)))
testModel.addConstr(x.sum() == 1)

testModel.setObjective(v, sense = GRB.MAXIMIZE)
testModel.optimize()
```

Run this model, we obtain *fair-o-meter*: $\xi = 0.5$.
Hence, we conclude that this game is more favorable for B and C. Here, we can find the optimal strategy for B and C. The optimal strategy for A can be found via LP (4). These strategy are:

5. The Find-the-square Game

pure strategy	percentage
$[0,0,0,2,2,1,2,1]$	0.054
$[0,0,1,0,1,1,2,2]$	0.032
$[0,0,1,1,2,2,2,2]$	0.095
$[0,0,1,2,2,1,0,0]$	0.043
$[0,0,2,2,1,2,1,0]$	0.018
$[0,2,0,1,1,2,0,1]$	0.072
$[0,2,2,0,1,2,1,0]$	0.065
$[1,0,0,2,2,1,0,1]$	0.069
$[1,0,2,0,1,2,1,1]$	0.013
$[1,0,2,2,2,0,1,1]$	0.027
$[1,1,0,0,0,2,2,2]$	0.002
$[1,1,0,0,1,1,2,2]$	0.088
$[1,1,0,2,2,2,0,2]$	0.007
$[1,1,1,2,0,1,0,2]$	0.027
$[1,1,2,2,0,0,2,2]$	0.039
$[1,1,2,2,1,0,2,0]$	0.101
$[1,2,0,0,1,1,2,0]$	0.047
$[1,2,2,0,0,2,0,1]$	0.017
$[2,0,0,0,1,1,2,2]$	0.017
$[2,1,2,0,0,0,1,2]$	0.084
$[2,1,2,1,0,1,0,0]$	0.035
$[2,2,0,0,1,1,0,0]$	0.020
$[2,2,1,1,0,1,0,0]$	0.028

pure strategy	percentage
$(1,0)$	$1/12$
$(2,0)$	$1/12$
$(4,0)$	$1/12$
$(7,0)$	$1/12$
$(1,1)$	$1/12$
$(2,1)$	$1/12$
$(4,1)$	$1/12$
$(7,1)$	$1/12$
$(1,2)$	$1/12$
$(2,2)$	$1/12$
$(4,2)$	$1/12$
$(7,2)$	$1/12$

References

[1]: Robert J.Vanderbei, *Linear Programming: Foundations and Extensions - Second Edition*, https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.111.1824&rep=rep1&type=pdf.

[2]: "The impossible chessboard puzzle", *YouTube*, uploaded by 3Blue1Brown, 6 Jul. 2020, https://www.youtube.com/watch?v=wTJI_WuZSwE&t=249s.