

Прізвище: Долінський

Ім'я: Олег

Група: КН-406

Варіант: 8



Кафедра: САПР

Дисципліна: Теорія прийняття рішень

Перевірив: Кривий Р.З.

ЗВІТ

до лабораторної роботи №5
на тему "Теорія ігор. Матричні ігри"

Мета роботи: визначити основні поняття теорії ігор, властивості змішаних стратегій. Вивчити метод вирішення матричних ігор у змішаних стратегіях за допомогою введення до подвійних завдань лінійного програмування.

Індивідуальне завдання:

У грі беруть участь два гравці: А і В. У розпорядженні кожного гравця є кінцеве безліч варіантів вибору - стратегій. Нехай - безліч стратегій гравця А, - безліч стратегій гравця В. З кожною парою стратегій пов'язаний платіж, який один з гравців виплачує іншому. Тобто, коли гравець А вибирає стратегію (свою i -ю стратегію), а гравець В - стратегію, то результатом такого вибору стає платіж. Оскільки стратегій кінцеве число, то платежі утворюють матрицю розмірності $n \times m$, звану матрицею платежів (або матрицею гри). Рядки цієї матриці відповідають стратегіям гравця А, а стовпці - стратегіям гравця В.

8.					
	15	8	15	16	4
	11	13	15	10	8
	14	16	10	12	14
	13	15	9	16	15
	11	14	16	10	6

Результати виконання індивідуального завдання:

1. Пошук сідлової точки.

	Стратегія В					
Стратегія А	B_1	B_2	B_3	B_4	B_5	Мінімум
A_1	15	8	15	16	4	4
A_2	11	13	15	10	8	8
A_3	14	16	10	12	14	10
A_4	13	15	9	16	15	9
A_5	11	14	16	10	6	6

	Стратегія В					
Стратегія А	В ₁	В ₂	В ₃	В ₄	В ₅	Мінімум
А ₁	15	8	15	16	4	4
А ₂	11	13	15	10	8	8
А ₃	14	16	10	12	14	10
А ₄	13	15	9	16	15	9
А ₅	11	14	16	10	6	6
Максимум	15	16	16	16	15	

Порівняємо нижню і верхню ціни гри, в даній задачі вони розрізняються, тобто $\alpha \neq \beta$, платіжна матриця не містить сідлової точки. Це означає, що гра не має рішення в чистих мінімаксії стратегіях, але вона завжди має рішення в змішаних стратегіях.

2. Спрощення платіжної матриці.

	Стратегія В			
Стратегія А	В ₁	В ₂	В ₃	В ₄
А ₁	15	15	16	4
А ₂	11	15	10	8
А ₃	14	10	12	14
А ₄	13	9	16	15
А ₅	11	16	10	10

3. Змішана стратегія.

Змішана стратегія гравця А:

$$15y_1 + 15y_2 + 16y_3 + 4y_4 \leq 1$$

$$11y_1 + 15y_2 + 10y_3 + 8y_4 \leq 1$$

$$14y_1 + 10y_2 + 12y_3 + 14y_4 \leq 1$$

$$13y_1 + 9y_2 + 16y_3 + 15y_4 \leq 1$$

$$11y_1 + 16y_2 + 10y_3 + 6y_4 \leq 1$$

$$F(x) = y_1 + y_2 + y_3 + y_4 \rightarrow \max$$

Змішана стратегія гравця В:

$$15x_1 + 11x_2 + 14x_3 + 13x_4 + 11x_5 \geq 1$$

$$15x_1 + 15x_2 + 10x_3 + 9x_4 + 16x_5 \geq 1$$

$$16x_1 + 10x_2 + 12x_3 + 16x_4 + 10x_5 \geq 1$$

$$4x_1 + 8x_2 + 14x_3 + 15x_4 + 6x_5 \geq 1$$

$$F(x) = x_1 + x_2 + x_3 + x_4 + x_5 \rightarrow \min$$

4. Пошук оптимального плану.

За допомогою онлайн-калькулятора симплекс-таблиця була зведена до:

Базис	В (вільний член)	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8	y_9
y_5	3/22	114/11	0	0	0	1	-19/11	69/22	-25/11	0
y_3	1/484	-47/121	0	1	0	0	6/121	-153/484	65/242	0
y_2	1/22	5/11	1	0	0	0	1/11	1/22	-1/11	0
y_4	9/242	122/121	0	0	1	0	-13/121	75/242	-20/121	0
y_9	7/242	189/121	0	0	0	0	-158/121	139/242	-29/121	1
Z(Y4)	41/484	9/121	0	0	0	0	4/121	19/484	3/242	0

Оптимальний план можна записати так:

$$x_1 = 0, x_2 = 4/121, x_3 = 19/484, x_4 = 3/242, x_5 = 0$$

$$y_1 = 0, y_2 = 1/22, y_3 = 1/484, y_4 = 9/242$$

$$Z(Y) = 41/484$$

$P = (0; 16/41; 19/41; 6/41)$ – оптимальна стратегія гравця А

$Q = (1, 0, 22/41; 1/41)$ – оптимальна стратегія гравця В

Код програми:

```
const getLowerGameValue = matrix => {
  return Math.max(...(matrix.reduce((acc, row) => {
    acc.push(row.reduce((acc, col) => {
      if (acc === 0 || acc > col) {
        acc = col;
      }
      return acc;
    }, 0))
    return acc;
  }, [])));
}

const getUpperGameValue = matrix => {
  const max = [];
  for (let i = 0; i < matrix.length; i++) {
    let temp = 0;
    for (let j = 0; j < matrix.length; j++) {
      if (temp === 0 || temp < matrix[j][i]) {
        temp = matrix[j][i];
      }
    }
    max.push(temp);
  }
}
```

```

    return Math.min(...max);
}
const matrixSimplification = matrix => {
    let newMatrix = matrix;

    for (let i = 0; i < newMatrix.length; i++) {
        let iRowToDelete = true;
        let jRowToDelete = true;
        let iIndex = -1;
        let jIndex = -1;

        for (let j = i + 1; j < newMatrix.length; j++) {
            iRowToDelete = true;
            jRowToDelete = true;
            iIndex = -1;
            jIndex = -1;

            for (let col = 0; col < matrix[0].length; col++) {
                if (matrix[i][col] > matrix[j][col]) {
                    if (jRowToDelete) jIndex = j;
                    iRowToDelete = false;
                } else if (matrix[i][col] < matrix[j][col]) {
                    if (iRowToDelete) iIndex = i;
                    jRowToDelete = false;
                }
            }
        }
        if (iRowToDelete) {
            newMatrix = deleteRow(matrix, iIndex);
            break;
        }
        if (jRowToDelete) {
            newMatrix = deleteRow(matrix, jIndex);
            break;
        }
    }

    if (newMatrix !== newMatrix[0].length) {
        for (let i = 0; i < newMatrix[0].length; i++) {
            let iColToDelete = true;
            let jColToDelete = true;
            let iIndex = -1;
            let jIndex = -1;

            for (let j = i + 1; j < newMatrix[0].length; j++) {
                iColToDelete = true;
                jColToDelete = true;
                iIndex = -1;
                jIndex = -1;

                for (let col = 0; col < newMatrix.length; col++) {
                    if (newMatrix[col][i] < newMatrix[col][j]) {
                        if (jColToDelete) jIndex = j;
                        iColToDelete = false;
                    } else if (newMatrix[col][i] > newMatrix[col][j]) {
                        if (iColToDelete) iIndex = i;
                    }
                }
            }
        }
    }
}

```

```

        jColToDelete = false;
    }
}
}
if (iColToDelete) {
    newMatrix = deleteCol(matrix, iIndex);
    break;
}
if (jColToDelete) {
    newMatrix = deleteCol(matrix, jIndex);
    break;
}
}
}

return newMatrix;
}

const simplexMethod = matrix => {
    const constraintsA = matrix.reduce((acc, row) => {
        let limitation = '';

        row.forEach((col, index) => {
            if (row.length - 1 !== index) {
                limitation += `${col}x${index + 1} + `
            } else {
                limitation += `${col}x${index + 1} `
            }
        })

        limitation += '<= 1';
        acc.push(limitation);
        return acc;
    }, []);

    const constraintsB = matrix.reduce((acc, row, i) => {
        let limitation = '';

        row.forEach((col, j) => {
            if (row.length - 1 !== j) {
                limitation += `${matrix[j][i]}x${j + 1} + `
            } else {
                limitation += `${matrix[j][i]}x${j + 1} `
            }
        })

        limitation += '>= 1';
        acc.push(limitation);
        return acc;
    }, []);

    const playerA = {
        type: "maximize",
        objective : "x1 + x2 + x3 + x4",
        constraints : constraintsA,
    };

    const playerB = {

```

```

    type: "minimize",
    objective : "x1 + x2 + x3 + x4",
    constraints : constraintsB,
  };

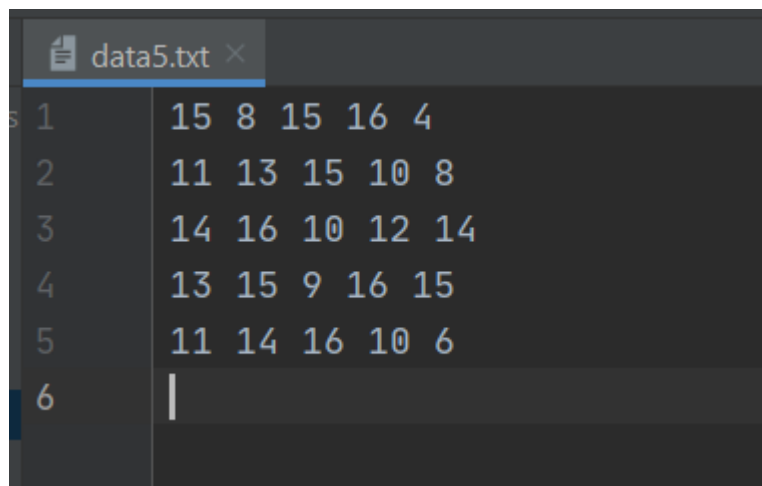
  return [
    YASMIJ.solve(playerA),
    YASMIJ.solve(playerB),
  ];
}

const getOptimalStrategy = (data, label) => {
  const { x1, x2, x3, x4, z } = data.result;

  return {
    [`${label}1`]: 1 / z * x1,
    [`${label}2`]: 1 / z * x2,
    [`${label}3`]: 1 / z * x3,
    [`${label}4`]: 1 / z * x4,
  }
}

```

Результати виконання програми:



s 1	15	8	15	16	4
2	11	13	15	10	8
3	14	16	10	12	14
4	13	15	9	16	15
5	11	14	16	10	6
6					

Рис. 1. Файл data5.txt

```

{
  p1: 0,
  p2: 0.5365853658536586,
  p3: 0.024390243902439067,
  p4: 0.4390243902439022
}
{ q1: 1, q2: 0, q3: 0, q4: 0 }

```

Висновок:

Виконуючи дану лабораторну роботу, я визначив основні поняття теорії ігор, властивості змішаних стратегій, вивчив метод вирішення матричних ігор у змішаних стратегіях за допомогою введення до подвійних завдань лінійного програмування.