Міністерство освіти і науки України
Національний авіаційний університет
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління

Лабораторна робота № 5
з дисципліни «Системи підтримки прийняття рішень»
на тему «Прийняття рішень в умовах невизначеності»
Варіант № 3

Виконав:
студент ФККПІ
групи СП-425
Клокун В. Д.
Перевірила:
Яковенко Л. В.

Київ 2020

1. Мета роботи

Ознайомитись з методом прийняття рішень в умовах невизначенності.Сформулювати отримане завдання прийняття рішення в умовах невизначенності та обрати оптимальну альтернативу.

2. Хід роботи

задача Один з N верстатів повинен бути обраний для виготовлення Q одиниць певної продукції. Мінімальна і максимальна потреба в продукції дорівнює Q^* і Q^{**} відповідно. Виробничі витрати TC_i на виготовлення Q одиниць продукції на верстаті i включають фіксовані витрати K_i і питомі витрати c_i на виробництво одиниці продукції і виражаються формулою $TC_i = K_i + c_i Q$. Завдання:

1. Вирішіть задачу за допомогою кожного з чотирьох критеріїв прийняття рішень в умовах невизначеності.
2. Вирішіть задачу при заданих даних (табл. 1), припускаючи, що $1000 \leqslant Q \leqslant 4000$.

Табл. 1: Вхідні дані

Верстат i	K_i, \$	C_i, \$
1	100	5
2	40	12
3	150	3
4	90	8

розв'язання Втрати від виробництва описуються функцією $TC_i = K_i + c_i Q$. Так як можлива кількість деталей може бути від Q^* до Q^{**}, опишемо множину можливих випадкових станів як $S = \{Q^*, \dots, Q^{**}\}$, а самі випадкові стани як $s_j \in S$. Тоді можна розв'язати задачу так:

$$\text{Критерій Лапласа:} \quad \min_i \left\{ \frac{1}{|S|} \sum_{s_j \in S} K_i + c_i Q_{s_j} \right\}$$

$$\text{Мінімаксний критерій:} \quad \min_i \left\{ \max_{s_j \in S} \left\{ K_i + c_i Q_{s_j} \right\} \right\}$$

$$\text{Критерій Севіджа:} \quad \min_i \left\{ \max_{s_j \in S} \left\{ K_i + c_i Q_{s_j} \right\} \right\}$$

$$\text{Критерій Гурвіца:} \quad \min_i \left\{ \alpha \min_{s_j \in S} \left\{ K_i + c_i Q_{s_j} \right\} + (1 - \alpha) \max_{s_j \in S} \left\{ K_i + c_i Q_{s_j} \right\} \right\}$$

Неважко побачити, що при достатньо великих Q, тобто $Q \gg K_i$, змінною K_i можна знехтувати, і тоді функція ціни буде переважно залежати від c_i, а отже оптимальна альтернатива при розв'язку задачі буде $\min_i\{c_i\}$.

Щоб розв'язати завдання на заданих даних, спочатку необхідно побудувати матрицю втрат на виробництво деталей. Для цього виділимо з множини можливих кількостей деталей Q 4 опорних випадки: $S = \{1000, 2000, 3000, 4000\}$. Далі на основі цих показників побудуємо матрицю втрат (табл. 2).

Табл. 2: Матриця втрат для $Q \in \{1000, 2000, 3000, 4000\}$

Верстат i	$TC(Q = 1000)$	$TC(Q = 2000)$	$TC(Q = 3000)$	$TC(Q = 4000)$
1	5100	10100	15100	20100
2	12040	24040	36040	48040
3	3150	6150	9150	12150
4	8090	16090	24090	32090

Розроблюємо програму, яка розв'яже задачу, оцінюючи значення кожного критерію. Вона складатиметься з модуля, який розв'язуватиме задачу (лістинг А.1) та модуля, який описує матрицю втрат (лістинг А.2). Запускаємо програму і спостерігаємо результат (рис. 1).

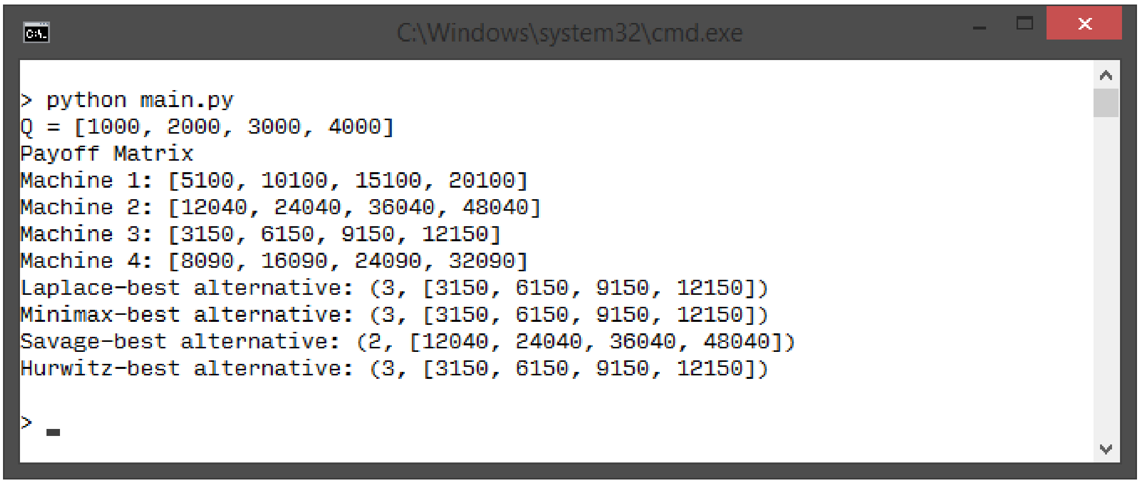

Рис. 1: Результат роботи розробленої програми

Видно, що розроблена реалізація розв'язала поставлену задачу і визначила оптимальний результат за чотирма критеріями. Судячи з їх значень, варто обрати верстат 2.

3. Висновок

Виконуючи дану лабораторну роботу, ми ознайомились з методом прийняття рішень в умовах ризику, сформулювали отримане завдання прийняття рішення в умовах ризику та обрали оптимальну альтернативу.

А. Лістинг коду програмної реалізації

Лістинг А.1: Файл `main.py`

```python
import payoff_matrix as pm


MACHINE_COSTS = {
    1: (100,  5),
    2: ( 40, 12),
    3: (150,  3),
    4: ( 90,  8),
}


Q_STATES = [1000, 2000, 3000, 4000]


def cost_fn(fixed_cost, cost_per_unit, units):
    return fixed_cost + cost_per_unit * units


def calc_cost_for_alternative(machine, units):
    cost = cost_fn(*machine, units)
    return cost


def get_laplace_alternative(costs_by_machine):
    row_avg = (
        sum(machine_cost_row) / len(machine_cost_row)
        for machine_cost_row in costs_by_machine
    )
    res = max(row_avg)
    return res


def main():
    costs_by_machine = {}
    for machine_name, machine_params in MACHINE_COSTS.items():
        machine_costs = [
```

```
            calc_cost_for_alternative(machine_params, q)
            for q in Q_STATES
        ]
        costs_by_machine[machine_name] = machine_costs

    print("Q = {}".format(Q_STATES))

    pm1 = pm.PayoffMatrix(costs_by_machine)
    pm1.print()
    print(
        "Laplace-best alternative:
         ↪  {}".format(pm1.get_laplace_alternative())
    )
    print(
        "Minimax-best alternative:
         ↪  {}".format(pm1.get_minimax_alternative())
    )
    print(
        "Savage-best alternative: {}".format(pm1.get_savage_alternative())
    )
    print(
        "Hurwitz-best alternative:
         ↪  {}".format(pm1.get_hurwitz_alternative())
    )


if __name__ == '__main__':
    main()
```

Лістинг А.2: Файл `payoff_matrix.py`

```
class PayoffMatrix(object):

    def __init__(self, matrix, opt_fn=min, not_opt_fn=max):
        self._raw_matrix = matrix
        self.matrix = self.parse_raw_matrix(matrix)
        self.opt_fn = opt_fn
        self.not_opt_fn = min if opt_fn is max else max

    @property
    def rows(self):
        for row in self._raw_matrix.items():
            yield row

    @staticmethod
    def parse_raw_matrix(raw_matrix):
```

```python
        parsed = [row for row in raw_matrix.values()]
        return parsed

    def print(self):
        print("Payoff Matrix")
        for machine, params in self._raw_matrix.items():
            print(
                "Machine {}: {}"
                .format(machine, params)
            )

    @staticmethod
    def laplace_key(row):
        """Key function for finding the Laplace criterion favourite."""
        _, params = row
        return sum(params) / len(params)

    def get_laplace_alternative(self):
        res = self.opt_fn(self.rows, key=self.laplace_key)
        return res

    def minimax_key(self, row):
        """Key function for finding the minimax (maximin) criterion favourite.
        """
        _, params = row
        return self.not_opt_fn(params)

    def get_minimax_alternative(self):
        res = self.opt_fn(self.rows, key=self.minimax_key)
        return res

    @staticmethod
    def transpose(matrix):
        return list(map(list, zip(*matrix)))

    def calc_regret_matrix(self):
        col_opts = [self.opt_fn(row) for row in self.transpose(self.matrix)]

        regret_matrix = {}
        for machine, params in self.rows:
            regret_row = []
            for el, col_opt in zip(params, col_opts):
                regret_row.append(el - col_opt)
            regret_matrix[machine] = regret_row

```

```python
        res = PayoffMatrix(regret_matrix, opt_fn=max)
        return res

    def get_savage_alternative(self):
        regret_matrix = self.calc_regret_matrix()
        machine, params = regret_matrix.get_minimax_alternative()
        res = self._raw_matrix[machine]
        return (machine, res)

    def hurwitz_key(self, row):
        """Key function for finding the Hurwitz criterion favourite."""
        _, params = row
        left_term = self.hurwitz_alpha * self.opt_fn(params)
        right_term = (1 - self.hurwitz_alpha) * self.not_opt_fn(params)
        return left_term + right_term

    def get_hurwitz_alternative(self, alpha=0.5):

        def hurwitz_key(row):
            """Inner key function for finding the best alternative
            according to the Hurwitz criterion.

            Used to account for variable alpha parameter.
            """
            _, params = row
            left_term = alpha * self.opt_fn(params)
            right_term = (1 - alpha) * self.not_opt_fn(params)
            return left_term + right_term

        res = self.opt_fn(self.rows, key=hurwitz_key)
        return res
```