

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 невизначеності. Сформулювати отримане завдання прийняття рішення в умовах невизначеності 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  $1000 \leq Q \leq 4000$ .

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:

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

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

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

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

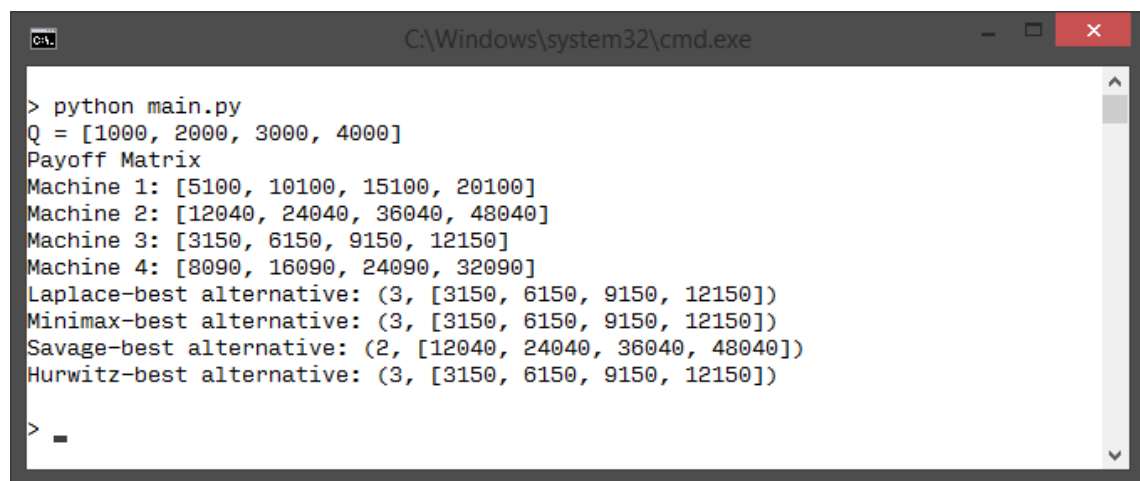
Неважко побачити, що при достатньо великих  $Q$ , тобто  $Q \gg K_i$ , змінною  $K_i$  можна знехтувати, і тоді функція ціни буде переважно залежати від  $c_i$ , а отже оптимальна альтернатива при розв'язку задачі буде  $\min_i \{c_i\}$ .

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).



```

C:\Windows\system32\cmd.exe
> python main.py
Q = [1000, 2000, 3000, 4000]
Payoff Matrix
Machine 1: [5100, 10100, 15100, 20100]
Machine 2: [12040, 24040, 36040, 48040]
Machine 3: [3150, 6150, 9150, 12150]
Machine 4: [8090, 16090, 24090, 32090]
Laplace-best alternative: (3, [3150, 6150, 9150, 12150])
Minimax-best alternative: (3, [3150, 6150, 9150, 12150])
Savage-best alternative: (2, [12040, 24040, 36040, 48040])
Hurwitz-best alternative: (3, [3150, 6150, 9150, 12150])
>

```

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 невизначеності, сформулювали отримане завдання прийняття рішення в умовах невизначеності та обрали оптимальну альтернативу.

#### А. ЛІСТИНГ КОДУ ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

---

Лістинг А.1: Файл main.py

---

```
1  import payoff_matrix as pm
2
3
4  MACHINE_COSTS = {
5      1: (100, 5),
6      2: ( 40, 12),
7      3: (150, 3),
8      4: ( 90, 8),
9  }
10
11
12  Q_STATES = [1000, 2000, 3000, 4000]
13
14
15  def cost_fn(fixed_cost, cost_per_unit, units):
16      return fixed_cost + cost_per_unit * units
17
18
19  def calc_cost_for_alternative(machine, units):
20      cost = cost_fn(*machine, units)
21      return cost
22
23
24  def get_laplace_alternative(costs_by_machine):
25      row_avg = (
26          sum(machine_cost_row) / len(machine_cost_row)
27          for machine_cost_row in costs_by_machine
28      )
29      res = max(row_avg)
30      return res
31
32
33  def main():
34      costs_by_machine = {}
35      for machine_name, machine_params in MACHINE_COSTS.items():
36          machine_costs = [
```

```

37         calc_cost_for_alternative(machine_params, q)
38         for q in Q_STATES
39     ]
40     costs_by_machine[machine_name] = machine_costs
41
42     print("Q = {}".format(Q_STATES))
43
44     pm1 = pm.PayoffMatrix(costs_by_machine)
45     pm1.print()
46     print(
47         "Laplace-best alternative:
48         ↪ {}".format(pm1.get_laplace_alternative())
49     )
50     print(
51         "Minimax-best alternative:
52         ↪ {}".format(pm1.get_minimax_alternative())
53     )
54     print(
55         "Savage-best alternative: {}".format(pm1.get_savage_alternative())
56     )
57     print(
58         "Hurwitz-best alternative:
59         ↪ {}".format(pm1.get_hurwitz_alternative())
60     )
61
62 if __name__ == '__main__':
63     main()

```

---

## Лістинг А.2: Файл payoff\_matrix.py

---

```

1  class PayoffMatrix(object):
2
3      def __init__(self, matrix, opt_fn=min, not_opt_fn=max):
4          self._raw_matrix = matrix
5          self.matrix = self.parse_raw_matrix(matrix)
6          self.opt_fn = opt_fn
7          self.not_opt_fn = min if opt_fn is max else max
8
9      @property
10     def rows(self):
11         for row in self._raw_matrix.items():
12             yield row
13
14     @staticmethod
15     def parse_raw_matrix(raw_matrix):

```

```

16         parsed = [row for row in row_matrix.values()]
17         return parsed
18
19     def print(self):
20         print("Payoff Matrix")
21         for machine, params in self._raw_matrix.items():
22             print(
23                 "Machine {}: {}".format(machine, params)
24             )
25
26
27     @staticmethod
28     def laplace_key(row):
29         """Key function for finding the Laplace criterion favourite."""
30         _, params = row
31         return sum(params) / len(params)
32
33     def get_laplace_alternative(self):
34         res = self.opt_fn(self.rows, key=self.laplace_key)
35         return res
36
37     def minimax_key(self, row):
38         """Key function for finding the minimax (maximin) criterion
39         ↪ favourite.
40         """
41         _, params = row
42         return self.not_opt_fn(params)
43
44     def get_minimax_alternative(self):
45         res = self.opt_fn(self.rows, key=self.minimax_key)
46         return res
47
48     @staticmethod
49     def transpose(matrix):
50         return list(map(list, zip(*matrix)))
51
52     def calc_regret_matrix(self):
53         col_opts = [self.opt_fn(row) for row in
54             ↪ self.transpose(self.matrix)]
55
56         regret_matrix = {}
57         for machine, params in self.rows:
58             regret_row = []
59             for el, col_opt in zip(params, col_opts):
60                 regret_row.append(el - col_opt)
61             regret_matrix[machine] = regret_row

```

```

61         res = PayoffMatrix(regret_matrix, opt_fn=max)
62         return res
63
64     def get_savage_alternative(self):
65         regret_matrix = self.calc_regret_matrix()
66         machine, params = regret_matrix.get_minimax_alternative()
67         res = self._raw_matrix[machine]
68         return (machine, res)
69
70     def hurwitz_key(self, row):
71         """Key function for finding the Hurwitz criterion favourite."""
72         _, params = row
73         left_term = self.hurwitz_alpha * self.opt_fn(params)
74         right_term = (1 - self.hurwitz_alpha) * self.not_opt_fn(params)
75         return left_term + right_term
76
77     def get_hurwitz_alternative(self, alpha=0.5):
78
79         def hurwitz_key(row):
80             """Inner key function for finding the best alternative
81             according to the Hurwitz criterion.
82
83             Used to account for variable alpha parameter.
84             """
85             _, params = row
86             left_term = alpha * self.opt_fn(params)
87             right_term = (1 - alpha) * self.not_opt_fn(params)
88             return left_term + right_term
89
90         res = self.opt_fn(self.rows, key=hurwitz_key)
91         return res

```

---