Министерство образования и науки Российской Федерации
Севастопольский государственный университет
Институт информационных технологий

Кафедра ИС

ОТЧЕТ

по лабораторной работе №2

ИССЛЕДОВАНИЕ ПРИМЕНЕНИЯ АППАРАТА ТЕОРИИ ОДНОМЕРНОЙ ПОЛЕЗНОСТИ ДЛЯ РЕШЕНИЯ ЗАДАЧ ВЫБОРА АЛЬТЕРНАТИВ

Выполнил:
ст. гр. ИС/б-21-2-о
Мовенко К. М.
Проверил:
Кротов К.В.

Севастополь
2024

1. ЦЕЛЬ РАБОТЫ

Исследовать применение аппарата теории полезности при принятии решений по выбору альтернатив.

2. ЗАДАНИЕ

Вариант 3. Задана матрица отношения нестрогого предпочтения (Рисунок 1). Используя метод, реализующий формирование классов эквивалентности $R(x_i)$ $(x_i \in X)$, формирование множества $X/\sim$ неповторяющихся классов эквивалентности k_i, выполнить разработку программы, определяющей значения функции полезности $U(k_l)$ для этих классов и значения функции $U(x_i)$ для решений $x_i \in X$, с последующим определением эффективных решений, для которых $x_i^* = \arg \max\limits_{1 \le i \le N} U(x_i)$.

$A =$

	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
x_1	0	0	0	1	0	0	0	1	0	0
x_2	0	0	1	0	1	0	0	0	1	0
x_3	1	0	0	1	0	1	1	1	0	1
x_4	1	1	0	0	1	0	0	1	1	0
x_5	0	1	0	0	0	0	0	0	1	0
x_6	1	0	1	0	0	0	1	0	0	1
x_7	1	0	1	0	0	1	0	1	0	1
x_8	1	1	0	1	1	0	0	0	1	0
x_9	0	1	0	0	1	0	0	0	0	0
x_{10}	1	0	1	0	0	1	1	0	0	0

Рисунок 1 – Матрица отношений нестрогого порядка

3. ХОД РАБОТЫ

Была написана программа, последовательно выполняющая формирование классов эквивалентности и нахождение значений функции полезности для этих классов. Её правильность была подтверждена на ряде примеров.

Для исходной матрицы не выполняется условие нестрогого упорядочения, однако при $x_2 \not\succ x_3$ и заполнении диагонали единицами (рефлексивность эквивалентности) нахождение полезности возможно.

```
1 0 0 1 0 0 0 1 0 0
0 1 0 0 1 0 0 0 1 0
1 0 1 1 0 1 1 1 0 1
1 1 0 1 1 0 0 1 1 0
0 1 0 0 1 0 0 0 1 0
1 0 1 0 0 1 1 0 0 1
1 0 1 0 0 1 1 1 0 1
1 1 0 1 1 0 0 1 1 0
0 1 0 0 1 0 0 0 1 0
1 0 1 0 0 1 1 0 0 1
```

Рисунок 1 – Обновлённая матрица отношений нестрогого порядка

```
R(x1) = { x1, x4, x8,}
R(x2) = { x2, x5, x9,}
R(x3) = { x3, x6, x7, x10,}
R(x4) = { x1, x4, x8,}
R(x5) = { x2, x5, x9,}
R(x6) = { x3, x6, x7, x10,}
R(x7) = { x3, x6, x7, x10,}
R(x8) = { x1, x4, x8,}
R(x9) = { x2, x5, x9,}
R(x10) = { x3, x6, x7, x10,}
```

Рисунок 2 – Множества эквивалентных элементов

```
k1 -> { x1, x4, x8,}
k2 -> { x2, x5, x9,}
k3 -> { x3, x6, x7, x10,}
```

Рисунок 3 – Классы эквивалентности

```
0 1 0
0 0 0
1 0 0
```

Рисунок 4 – Матрица предпочтений для классов эквивалентности

```
U(k1) = 0
U(k2) = -1
U(k3) = 1
```

Рисунок 5 – Функция полезности для классов

```
U(x1) = 0
U(x2) = -1
U(x3) = 1
U(x4) = 0
U(x5) = -1
U(x6) = 1
U(x7) = 1
U(x8) = 0
U(x9) = -1
U(x10) = 1
```

Рисунок 6 – Функция полезности для решений

В данном примере наиболее эффективными решениями являются x_3, x_6, x_7, x_{10}.

4. ТЕКСТ ПРОГРАММЫ

```
import copy


# упорядочение по матрице отношений
def is_collatable(a, n):
    count = n

    while count > 0:
        excluded = []

        # определение исключаемых элементов
        for i in range(n):
            row_sum = sum(a[i])
            col_sum = sum([a[j][i] for j in range(n)])
            if row_sum == 0:
                if col_sum == 0:
                    return False
                excluded.append(i)

        if len(excluded) == 0 or len(excluded) == n:
            return False

        # обнуление отношений с исключаемыми элементами
        for q in range(len(excluded)):
            for x in range(n):
                a[x][excluded[q]] = 0

        # исключение элементов, добавленных в MaxR
        for q in range(len(excluded)):
            for x in range(n):
                a[excluded[q]][x] = 1

        count -= len(excluded)

    return True


# нахождение значений функции полезности
def findU(l_prev, h_prev, seen, m, iter):
    L = []
    H = []

    for l in range(0, len(K)):
        if B[m][l] == 1:
            if l not in seen:
                L.append(l)
                Uk[l] = Uk[m] - 1    #-iter
                iter += 1
                seen.append(l)
            else:
                if h_prev != -1:
                    Uk[m] = (Uk[l] + Uk[h_prev]) / 2

    for h in range(0, len(K)):
        if B[h][m] == 1:
            if h not in seen:
                H.append(h)
                Uk[h] = Uk[m] + 1    #iter
                iter += 1
                seen.append(h)
```

```
            else:
                if l_prev != -1:
                    Uk[m] = (Uk[h] + Uk[l_prev]) / 2

    for l in L:
        iter = findU(-1, m, seen, l, iter)

    for h in H:
        iter = findU(m, -1, seen, h, iter)

    return iter


n = 10  # число альтернатив

A = [
    [1, 0, 0, 1, 0, 0, 0, 1, 0, 0],
    [0, 1, 0, 0, 1, 0, 0, 0, 1, 0],
    [1, 0, 1, 1, 0, 1, 1, 1, 0, 1],
    [1, 1, 0, 1, 1, 0, 0, 1, 1, 0],
    [0, 1, 0, 0, 1, 0, 0, 0, 1, 0],
    [1, 0, 1, 0, 0, 1, 1, 0, 0, 1],
    [1, 0, 1, 0, 0, 1, 1, 1, 0, 1],
    [1, 1, 0, 1, 1, 0, 0, 1, 1, 0],
    [0, 1, 0, 0, 1, 0, 0, 0, 1, 0],
    [1, 0, 1, 0, 0, 1, 1, 0, 0, 1]
]

R = [[] for _ in range(n)]  # множества экв. элементов
K = []                      # классы эквивалентности
B = []                      # матрица отношений классов экв-ти
Uk = []                     # функция полезности для классов
Ux = [0] * n                # функция полезности для альтернатив


# ВЫЧИСЛЕНИЯ

# заполнение множеств экв. элементов
for i in range(n):
    for j in range(n):
        if A[i][j] == A[j][i] == 1:
            R[i].append(j)

# идентификация классов эквивалентности
K = list(sorted(set(tuple(k) for k in R)))

# заполнение матрицы отношений классов эквивалентности
B = [[0] * len(K) for _ in range(len(K))]
for l in range(len(K)):
    for h in range(len(K)):
        if l == h:
            continue
        summ = sum(A[i][j] for i in K[l] for j in K[h])
        if summ != 0:
            B[l][h] = 1

# проверка возможности упорядочить классы
if not is_collatable(copy.deepcopy(B), len(K)):
    print("ОШИБКА: невозможно упорядочить классы")
    exit()

# поиск значений U(k) для классов экв-ти
Uk = [0] * len(K)
findU(-1, -1, [0], 0, 1)
```

```
# установка значений U(x) для альтернатив
for l in range(len(Uk)):
    for i in K[l]:
        Ux[i] = Uk[l]


# ВЫВОД ДАННЫХ

print("Матрица нестрогого предпочтения:")
for row in A:
    print(*row)
print()

print("Множества эквивалентных элементов:")
for i in range(n):
    print("R(x{}) = {{".format(i + 1), end="")
    for j in R[i]:
        print(" x{}".format(j + 1), end=",")
    print("}")
print()

print("Классы эквивалентности X/~:")
for l in range(len(K)):
    print("k{} -> {{".format(l + 1), end="")
    for i in K[l]:
        print(" x{}".format(i + 1), end=",")
    print("}")
print()

print("Матрица строгого предпочтения классов эквивалентности:")
for row in B:
    print(*row)
print()

print("Полезность классов эквивалентности:")
for l in range(len(K)):
    print("U(k{}) = {}".format(l + 1, Uk[l]))
print()

print("Полезность альтернатив:")
for i in range(n):
    print("U(x{}) = {}".format(i + 1, Ux[i]))
print()

print("Эффективные решения:")
print("{", end="")
for i in range(n):
    if Ux[i] == max(Ux):
        print(" x{}".format(i + 1, Ux[i]), end=",")
print("}")
```

5. ВЫВОД

В ходе работы было исследовано применение аппарата теории полезности при принятии решений по выбору альтернатив.