Министерство образования и науки Российской Федерации Федеральное государственное автономное образовательное учреждение высшего профессионального образования «Севастопольский государственный университет»

ОТЧЕТ

о выполнении лабораторной работы № 1 по дисциплине

«Теория принятия решений»

Выполнил:

ст. гр. ИС/б-22-1-о

Гюнтер М. Ю.

Проверил:

Профессор кафедры

“Информационные системы”

Кротов К. В.

Севастополь,2025

# Цель работы: исследовать применение аппарата теории полезности при принятии решений по выбору альтернатив.

# Постановка задачи

Используя метод, реализующий формирование множеств и , а также их последующий анализ (с точки зрения (), (), , , выполнить для заданного вида матрицы отношения предпочтения определение значений функции полезности решений и определение по формируемым значениям функции полезности эффективных решений . Матрица отношения предпочтения имеет вид, приведенный на рисунке 2.1.

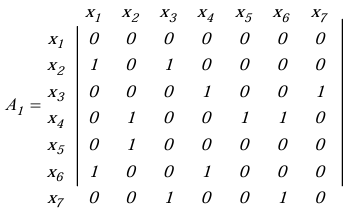


Рисунок 2.1 – Матрица отношения

2.1. Реализовать объявление и инициализацию матрицы отношений между решениями в соответствии с вариантом задания;

2.2. Реализовать процедуру определения для каждого рассматриваемого решения соответствующих ему множеств и , которые определяют для решения не худшие по отношению к нему решения (множество ) и не лучшие по отношению к нему решения (множество ); при определении множества необходимо выполнять просмотр (n+1)-го столбца матрицы отношений, при определении множества необходимо выполнять просмотр (n+1)-ой строки матрицы отношений, для рассматриваемого элемента выполнить вывод множеств , ;

2.3. Реализовать процедуру выполнения условий (), (), , ; тем самым определяется способ вычисления значений функции полезности для решения ; реализовать вывод информации о выполняющемся условии;

2.4. Реализовать процедуру вычисления значения функции полезности для текущего рассматриваемого решения ;

2.5. Реализовать процедуру управления процессом вычисления значений функции полезности для каждого элемента множества Х (решения множества Х); реализовать в рассматриваемой процедуре определение максимального значения функции полезности и соответствующего ему решения; выполнить вывод всех решений и соответствующих им значений функции полезности.

# Ход работы

## Вычисление значений функции полезности решений аналитическим путем

По заданному алгоритму определения значений функции полезности решений пошагово найдем значение функции полезности для каждого решения :

1. Решение , ;
2. Решение :

;

; ;

, , ;

1. Решение :

;

; ;

, , 1 – 1 = 0;

1. Решение :

;

; ;

, , = 0.5;

1. Решение :

;

; ;

, , = 0.75;

1. Решение :

;

; ;

, , = 0.5;

1. Решение :

;

; ;

, , = 0;

Итого получаем: , , , , , , . Отсюда максимальным решением является решение с наибольшим значением функции полезности .

## Формирование программного кода процедуры определения значений функции полезности решений

Была написана программа на языке Java, позволяющая определить значения функций полезности решений (листинг ‎3.1).

Листинг ‎3.1 – Программа, позволяющая определить значения функций полезности решений

import static java.lang.System.out;

import java.util.Arrays;

import java.util.Collections;

import java.util.ArrayList;

public class Lab2 {

public static int chooseXnPlus1(int[][] a, ArrayList<Integer> Xn) {

int card = a[0].length;

ArrayList<Integer> options = new ArrayList<>();

for (int x : Xn) {

for (int i = 0; i < card; i++) {

if (!Xn.contains(i) && !options.contains(i) && a[i][x] == 1) {

options.add(i);

}

}

for (int j = 0; j < card; j++) {

if (!Xn.contains(j) && !options.contains(j) && a[x][j] == 1) {

options.add(j);

}

}

}

if (!options.isEmpty()) return Collections.min(options);

return -1;

}

public static ArrayList<Integer> getXnPlus(int[][] a, ArrayList<Integer> Xn, int j) {

int card = a[0].length;

ArrayList<Integer> XnPlus = new ArrayList<>();

for (int i = 0; i < card; i++) {

if (a[i][j] == 1 && Xn.contains(i)) {

XnPlus.add(i);

}

}

return XnPlus;

}

public static ArrayList<Integer> getXnMinus(int[][] a, ArrayList<Integer> Xn, int i) {

int card = a[0].length;

ArrayList<Integer> XnMinus = new ArrayList<>();

for (int j = 0; j < card; j++) {

if (a[i][j] == 1 && Xn.contains(j)) {

XnMinus.add(j);

}

}

return XnMinus;

}

public static double computeU(ArrayList<Integer> XnPlus, ArrayList<Integer> XnMinus, double[] Us) {

if (XnPlus.isEmpty() && !XnMinus.isEmpty()) {

double xDoubleQt = -1;

for (int x : XnMinus) {

if (Us[x] > xDoubleQt)

xDoubleQt = Us[x];

}

double U = xDoubleQt + 1;

return U;

}

if (!XnPlus.isEmpty() && XnMinus.isEmpty()) {

double xQt = Us[XnPlus.get(0)];

for (int x : XnPlus) {

if (Us[x] < xQt)

xQt = Us[x];

}

double U = xQt - 1;

return U;

}

if (!XnPlus.isEmpty() && !XnMinus.isEmpty()) {

ArrayList<Integer> inter = new ArrayList<>(XnPlus);

inter.retainAll(XnMinus);

boolean isInter = !inter.isEmpty();

if (isInter) {

return Us[inter.get(0)];

} else {

double xDoubleQt = -1;

for (int x : XnMinus) {

if (Us[x] > xDoubleQt)

xDoubleQt = Us[x];

}

double xQt = Us[XnPlus.get(0)];

for (int x : XnPlus) {

if (Us[x] < xQt)

xQt = Us[x];

}

double U = (xQt + xDoubleQt) / 2;

return U;

}

}

return -1;

}

public static double[] findUs(int[][] a) {

int card = a[0].length;

double[] Us = new double[card];

ArrayList<Integer> Xn = new ArrayList<>();

ArrayList<Integer> XnPlus;

ArrayList<Integer> XnMinus;

// U(x1) = 0

Us[0] = 0;

// Xn = {x1}

Xn.add(0);

for (int xS = 1; xS < card; xS++) {

int xnPlus1 = chooseXnPlus1(a, Xn);

if (xnPlus1 == -1) {

out.println("Алгоритм не реализуется для заданной матрицы отношения.");

break;

}

XnPlus = getXnPlus(a, Xn, xnPlus1);

XnMinus = getXnMinus(a, Xn, xnPlus1);

Us[xnPlus1] = computeU(XnPlus, XnMinus, Us);

Xn.add(xnPlus1);

}

return Us;

}

public static void main(String[] args) {

int[][] a = new int[][] {

{ 0, 0, 0, 0, 0, 0, 0 },

{ 1, 0, 1, 0, 0, 0, 0 },

{ 0, 0, 0, 1, 0, 0, 1 },

{ 0, 1, 0, 0, 1, 1, 0 },

{ 0, 1, 0, 0, 0, 0, 0 },

{ 1, 0, 0, 1, 0, 0, 0 },

{ 0, 0, 1, 0, 0, 1, 0 },

};

double[] Us = findUs(a);

out.printf("Матрица отношения R: \n%s\n\n",

Arrays.deepToString(a).replace("], ", "]\n").replace("[[", "[").replace("]]", "]"));

out.printf("Полезность каждого решения U(xi): %s", Arrays.toString(Us));

}

}

Результаты работы программы приведены на рисунке ‎3.1.

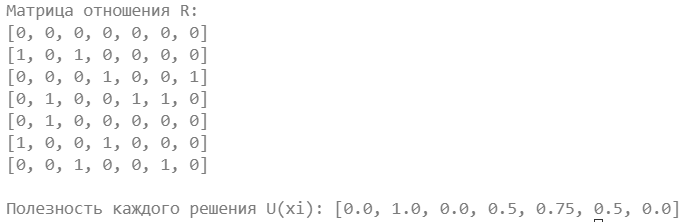


Рисунок ‎3.1 – Результат работы программы

Результаты работы программы говорят о том, что среди решений x1, x2, x3, x4, x5, x6, x7 максимальным решением является , что совпадает с полученным ранее аналитически результатом.

# Вывод:

В результате выполнения лабораторной работы было сформировано базовое представление о функции полезности.