МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Севастопольский государственный университет»

Институт информационных технологий и управления в технических системах

кафедра Информационные системы

09.03.02 Информационные системы и технологии (уровень бакалавриата)

Лабораторная работа №2

по дисциплине: «Теория принятия решений»

ИССЛЕДОВАНИЕ ПРИМЕНЕНИЯ АППАРАТА ТЕОРИИ ПОЛЕЗНОСТИ ДЛЯ ОПИСАНИЯ БИНАРНЫХ ОТНОШЕНИЙ ПРИ ПРИНЯТИИ РЕШЕНИЙ

Вариант – 2

Выполнил

студент 3 курса группы ИС/б-33-о

Генералов Николай Николаевич

|  |  |
| --- | --- |
|  |  |

Проверил

доц. \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кротов К. В.

(должность) (подпись) (фамилия, инициалы)

г. Севастополь

2018 г.

# ЦЕЛЬ РАБОТЫ

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

# ЗАДАНИЕ

Используя метод, реализующий формирование классов эквивалентности *R(xi)* (*xi* ∈ *X*) и формирование множества *X*/~ неповторяющихся классов эквивалентности *kl*, выполнить разработку программы, которая:

* определяет значения функции полезности *U(kl*) для этих классов и значения функции *U(xi)* для решений *xi* ∈ *X*;
* определяет эффективные решений (*x\*i) = arg max U(xi).*

Вид матриц отношений предпочтения (A1) и эквивалентности (A2) следующий:

# ПОСТАНОВКА ЗАДАЧИ

* реализовать инициализацию матриц отношений строго предпочтения А1 и эквивалентности А2;
* реализовать процедуру, формирующую на основе матрицы отношения эквивалентности А2, классы эквивалентности R( xi ) ( xi X ∈ );
* реализовать процедуру, выполняющую сравнение полученных классов эквивалентности R(xi) (xi ∈ X), исключение повторяющихся классов, формирующую множество X /~ уникальных классов эквивалентности решений kl;
* реализовать процедуру, выполняющую упорядочивание классов эквивалентности kl с определением соответствующих им значений функции полезности U(kl) ;
* реализовать процедуру, которая выполняет инициализацию значений функции полезности элементов (решений) U( xi ) множества X, входящих в соответствующие классы эквивалентности kl, значениями функции полезности этих классов U( kl ). Разрабатываемая процедура также выполняет упорядочивание решений xi  ∈ X, с точки зрения значений их функции полезности и определяет решение x\*i ∈ X , для которого значение функции полезности является максимальным;
* реализовать вывод исходных данных, промежуточных и конечных результатов: матриц отношений А1 и А2, классов эквивалентности R( xi ) ( xi ∈ X), множества X /~ не повторяющихся ("уникальных") классов эквивалентности, полученных значений функции полезности U( kl ) для каждого класса kl , значений функции полезности для решений xi ∈ X, соответствующих этим классам, эффективных решений с максимальным значением функции полезности.

# ХОД РАБОТЫ

* 1. Аналитические расчеты.

На основании матрицы отношений эквивалентности для рассматриваемого множества X, сформируем множества эквивалентных решений:

*R(x1) = {x1, x3, x6};*

*R(x2) = {x2, x5};*

*R(x3) = {x1, x3, x6};*

*R(x4) = {x4, x7};*

*R(x5) = {x2, x5};*

*R(x6) = {x1, x3, x6};*

*R(x7) = {x4, x7};*

Для идентификации различных классов эквивалентности, составим множество уникальных классов эквивалентности *X/~ = {k1, k2, k3},* где:

*k1 →{x1, x3, x6};*

*k2 →{x2, x5};*

*k3 →{x4, x7};*

Исходя из матрицы отношений строгого предпочтения, можно утверждать, что *k3 ’ k2  ’ k1*.

Определим значения функции полезности *U(km):*

*U(k1)* = 0;

*U(k2)* = 2, так как выполняется отношение *x2 x1;*

*U(k3)* = 3, так как выполняется отношение *x4 x2;*

На основании значений функции полезности уникальных классов эквивалентности, определим значения функции полезности для каждого решения *xi* ∈ X:

*U(x1)* = 0;

*U(x2)* = 2;

*U(x3)* = 0;

*U(x4)* = 3;

*U(x5)* = 2;

*U(x6)* = 0;

*U(x7)* = 3;

Отсюда видно, что эффективными решениями являются x4 и x7.

1. РЕЗУЛЬТАТЫ ВЫПОЛНЕНИЯ ПРОГРАММЫ

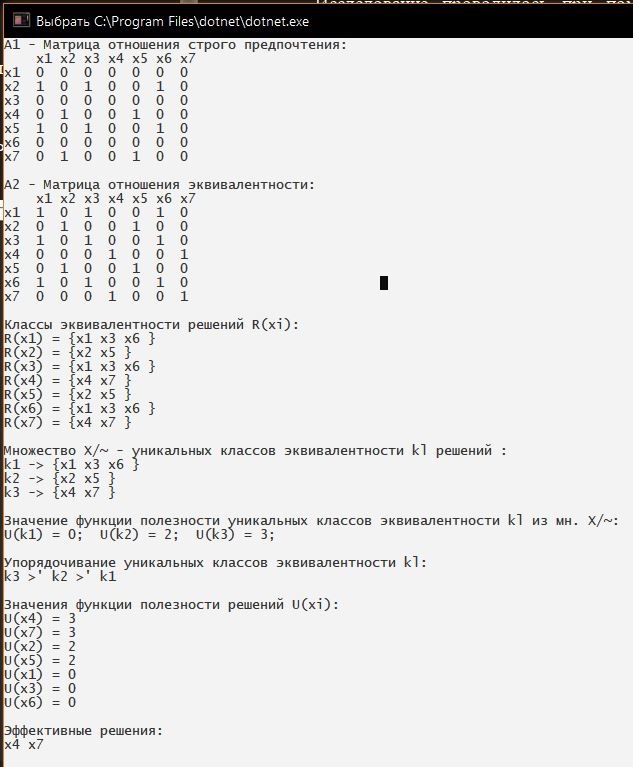


Рисунок 1 – Вывод результатов выполнения программы в консоль

# ВЫВОД

В ходе выполнения лабораторной работы исследовалось применение аппарата теории полезности при принятии решений по выбору альтернатив. Исследование проводилось при помощи метода, формирования классов эквивалентности и множества неповторяющихся классов эквивалентности, с определением значения функции полезности для этих классов. На основании значений классовых функций полезности, были найдены значения функций полезности для альтернатив принадлежащих этим классам и выбраны эффективные решения.

Так же в ходе лабораторной работы была написана программа, на языке C# осуществляющая, на основании загруженных из файла матриц отношений строго предпочтения и отношений эквивалентности:

* формирование классов эквивалентностей;
* формирование уникальных классов эквивалентности;
* расчет функции полезности для классов эквивалентности;
* упорядочивание классов эквивалентности;
* нахождение значений функции полезности для каждого из решений;
* выбор эффективных решений;

По результатам выполнения программы и аналитическим расчетам, можно сделать вывод, что программа составлена правильно и поставленная цель достигнута.

Приложение А

Текст программных модулей

«Program.cs»

using System;

namespace TPR\_lab\_2

{

class Program

{

static void Main(string[] args)

{

MatrixLoader LoadedMatrixA1 = new MatrixLoader("A1.txt");

MatrixLoader LoadedMatrixA2 = new MatrixLoader("A2.txt");

MyMatrix A1 = new MyMatrix();

MyMatrix A2 = new MyMatrix();

A1.Matrix = MyMatrix.ConvertToBool(LoadedMatrixA1.Load());

A2.Matrix = MyMatrix.ConvertToBool(LoadedMatrixA2.Load());

Console.WriteLine("A1 - Матрица отношения строго предпочтения:");

MyMatrix.PrintBoolMatrix(A1.Matrix, 'x');

Console.WriteLine();

Console.WriteLine("A2 - Матрица отношения эквивалентности:");

MyMatrix.PrintBoolMatrix(A2.Matrix, 'x');

Console.WriteLine();

UsefulValues main\_data = new UsefulValues(A1.Matrix, A2.Matrix);

Console.WriteLine("Классы эквивалентности решений R(xi):");

main\_data.PrintAllEqualsClasses();

Console.WriteLine();

Console.WriteLine("Множество X/~ - уникальных классов эквивалентности kl решений :");

main\_data.PrintMatrixClassGroups('x');

Console.WriteLine();

Console.WriteLine("Значение функции полезности уникальных классов эквивалентности kl из мн. X/~:");

main\_data.PrintClassGroupsUsefulCoeficients();

Console.WriteLine();

Console.WriteLine();

Console.WriteLine("Упорядочивание уникальных классов эквивалентности kl:");

main\_data.PrintComparableClassGroups();

Console.WriteLine();

Console.WriteLine();

Console.WriteLine("Значения функции полезности решений U(xi):");

main\_data.SortAndPrintElementsUsefulCoefficients();

Console.WriteLine();

Console.WriteLine("Эффективные решения:");

main\_data.PrintMostUsefulElements();

}

}

}

«MatrixLoader.cs»

using System;

using System.Collections.Generic;

using System.IO;

namespace TPR\_lab\_2

{

class MatrixLoader

{

public string FileName { get; private set; }

protected StreamReader \_fread { get; private set; }

public MatrixLoader(string \_filename)

{

FileName = \_filename;

try

{

\_fread = new StreamReader(new FileStream(FileName, FileMode.Open));

}

catch (FileNotFoundException err)

{

Console.WriteLine(err);

Environment.Exit(0);

}

}

public List<List<int>> Load()

{

try

{

List<List<int>> matrix = new List<List<int>>();

int matrixRowIndex = 0;

while (!\_fread.EndOfStream)

{

string tempString = \_fread.ReadLine();

matrix.Add(new List<int>());

string[] NumericStrings = tempString.Split(" ");

foreach (var num in NumericStrings)

{

int resultNum = 0;

if (Int32.TryParse(num, out resultNum))

{

matrix[matrixRowIndex].Add(resultNum);

}

}

++matrixRowIndex;

}

return matrix;

}

catch (IOException err)

{

Console.WriteLine(err);

return null;

}

}

~MatrixLoader()

{

if (\_fread != null)

{

\_fread.Close();

}

} }}

«MyMatrix.cs»

using System;

using System.Collections.Generic;

namespace TPR\_lab\_2

{

public class MyMatrix

{

public MyMatrix()

{

this.Matrix = null;

}

public MyMatrix(int[,] initMatrix)

{

Matrix = ConvertMatrix(initMatrix);

}

public List<List<bool>> Matrix { get; set; }

public static List<List<bool>> ConvertMatrix(int[,] matrix)

{

List<List<bool>> tempMatrix = new List<List<bool>>();

int rowCount = matrix.GetLength(0);

for (int i = 0; i < rowCount; ++i)

{

tempMatrix.Add(new List<bool>());

for (int j = 0; j < (matrix.Length / rowCount); ++j)

{ tempMatrix[i].Add(Convert.ToBoolean(matrix[i,j]));

}

}

return tempMatrix;

}

public static void PrintBoolMatrix(List<List<bool>> Matrix, char colRowChar = ' ')

{

Console.Write(" ");

for(int i = 0; i < Matrix[0].Count; ++i)

{

Console.Write($"{colRowChar}{i + 1} ");

}

Console.WriteLine();

for (int i = 0; i < Matrix.Count; ++i)

{

Console.Write($"{colRowChar}{i + 1} ");

for (int j = 0; j < Matrix[i].Count; ++j)

{ Console.Write($"{Convert.ToInt16(Matrix[i][j]), 2} ");

}

Console.WriteLine();

}

}

public static List<List<bool>> ConvertToBool(List<List<int>> IntMatrix)

{

List<List<bool>> resultMatrix = new List<List<bool>>();

for(int i = 0; i < IntMatrix.Count; ++i)

{

resultMatrix.Add(new List<bool>());

for (int j = 0; j < IntMatrix[i].Count; ++j)

{

resultMatrix[i].Add((IntMatrix[i][j] > 0) ? true : false);

}

}

return resultMatrix;

}

public static void FillSameValue(List<bool> vector, bool value)

{

for (int i = 0; i < vector.Count; ++i)

{

vector[i] = value;

} } }}

«UsefulValues.cs»

using System;

using System.Collections.Generic;

namespace TPR\_lab\_2

{

class UsefulValues

{

public UsefulValues(List<List<bool>> A1, List<List<bool>> A2)

{

MatrixA1 = A1;

MatrixA2 = A2;

EqualClassesMatrix = null;

ClassGroupUsefulCoeficientVector = null;

SortedClassGroupsIndexes = null;

ElementsUsefulCoefficients = null;

MostUsefulElementsIndexes = null;

Calculate();

}

public List<List<bool>> MatrixA1 { get; private set; }

public List<List<bool>> MatrixA2 { get; private set; }

public List<List<int>> EqualClassesMatrix { get; private set; }

public List<double> ClassGroupUsefulCoeficientVector { get; private set; }

private List<int> SortedClassGroupsIndexes { get; set; }

private List<double> ElementsUsefulCoefficients { get; set; }

private List<int> MostUsefulElementsIndexes { get; set; }

private void Calculate()

{

EqualClassesMatrix = SearchEqualsClasses();

ClassGroupUsefulCoeficientVector = CalculateClassGroupUsefulCoefficient();

SortedClassGroupsIndexes = CompareClassesVector();

ElementsUsefulCoefficients = CalculateElementsCoefficients();

MostUsefulElementsIndexes = SearchMostUsefulElements();

}

private List<List<int>> SearchEqualsClasses()

{

List<List<int>> EqualsClassMatrix = new List<List<int>>();

bool[] tempVector = new bool[this.MatrixA2.Count];

List<bool> answerVector = new List<bool>(tempVector);

MyMatrix.FillSameValue(answerVector, false);

int currentRowPosition = 0;

for (int i = 0; i < this.MatrixA2.Count; ++i)

{

if (answerVector[i] == false)

{

EqualsClassMatrix.Add(new List<int>());

for (int j = 0; j < this.MatrixA2.Count; ++j)

{

if (this.MatrixA2[i][j] == true)

{

EqualsClassMatrix[currentRowPosition].Add(j);

answerVector[i] = answerVector[j] = true;

}

}

++currentRowPosition;

}

}

return EqualsClassMatrix;

}

private double GetDoubleVal(double a, double b)

{

return (a + b) / 2.0 ;

}

private List<double> CalculateClassGroupUsefulCoefficient()

{

List<double> result = new List<double> { 0.0 };

for (int i = 1; i < EqualClassesMatrix.Count; ++i)

{

double coeficient = 0.0;

int IndexOfBiggestClass = -1;

int IndexOfSmallestClass = -1;

for (int j = 0; j < i; ++j)

{

if (MatrixA1[EqualClassesMatrix[i][0]][EqualClassesMatrix[j][0]] == true)

{

if (IndexOfSmallestClass == -1)

{

IndexOfSmallestClass = j;

} else

{

IndexOfSmallestClass = result[IndexOfSmallestClass] < result[j] ? IndexOfSmallestClass : j;

}

}

else if (MatrixA1[EqualClassesMatrix[j][0]][EqualClassesMatrix[i][0]] == true)

{

if (IndexOfBiggestClass == -1)

{

IndexOfBiggestClass = j;

}

else

{

IndexOfBiggestClass = result[IndexOfBiggestClass] > result[j] ? IndexOfSmallestClass : j;

}

}

}

if (IndexOfBiggestClass != -1 && IndexOfSmallestClass != -1)

{

coeficient = GetDoubleVal(result[IndexOfSmallestClass], result[IndexOfBiggestClass]);

}

else if (IndexOfSmallestClass == -1 && IndexOfBiggestClass != -1)

{

coeficient = -(double)(i + 1);

}

else

{

coeficient = (double)(i + 1);

}

result.Add(coeficient);

}

return result;

}

private List<int> CompareClassesVector()

{

List<int> result = new List<int>();

bool[] flagArray = new bool[ClassGroupUsefulCoeficientVector.Count];

List<bool> flagVector = new List<bool>(flagArray);

MyMatrix.FillSameValue(flagVector, false);

for (int i = 0; i < (ClassGroupUsefulCoeficientVector.Count); ++i)

{

int max = 0;

for(; max < flagVector.Count; ++max)

{

if (flagVector[max] == false)

{

break;

}

}

for (int j = 0; j < ClassGroupUsefulCoeficientVector.Count; ++j)

{

if (flagVector[j] == false)

{

if (ClassGroupUsefulCoeficientVector[max] < ClassGroupUsefulCoeficientVector[j])

{

max = j;

}

}

}

flagVector[max] = true;

result.Add(max);

}

return result;

}

private List<double> CalculateElementsCoefficients()

{

List<double> result = new List<double>();

for (int i = 0; i < MatrixA2.Count; ++i)

{

for (int j = 0; j < EqualClassesMatrix.Count; ++j)

{

if (EqualClassesMatrix[j].Contains(i))

{

result.Add(ClassGroupUsefulCoeficientVector[j]);

}

}

}

return result;

}

private List<int> SearchMostUsefulElements()

{

List<int> result = new List<int>();

for (int j = 0; j < ElementsUsefulCoefficients.Count; ++j)

{

if (ElementsUsefulrCoefficients[j] == ClassGroupUsefulCoeficientVector[SortedClassGroupsIndexes[0]])

{

result.Add(j);

}

}

return result;

}

public void PrintMatrixClassGroups(char colRowChar = ' ')

{

int counter = 0;

foreach (var i in EqualClassesMatrix)

{

Console.Write($"k{counter++ + 1} -> {{");

foreach (int j in i)

{

Console.Write($"{colRowChar}{j + 1} ");

}

Console.WriteLine("}");

}

}

public void PrintClassGroupsUsefulCoeficients()

{

for (int i = 0; i < ClassGroupUsefulCoeficientVector.Count; ++i)

{

Console.Write($"U(k{i + 1}) = {ClassGroupUsefulCoeficientVector[i]}; ");

}

}

public void PrintAllEqualsClasses()

{

for (int i = 0; i < MatrixA2.Count; ++i)

{

Console.Write($"R(x{i + 1}) = {{");

for (int j = 0; j < MatrixA2[i].Count; ++j)

{

if (MatrixA2[i][j] != false)

{

Console.Write($"x{j + 1} ");

}

}

Console.WriteLine("}");

}

}

public void PrintComparableClassGroups()

{

for(int i = 0; i < SortedClassGroupsIndexes.Count; ++i)

{

if (i != 0)

{

Console.Write(" >' ");

}

Console.Write($"k{SortedClassGroupsIndexes[i] + 1}");

}

}

public void SortAndPrintElementsUsefulCoefficients()

{

for (int i = 0; i < ClassGroupUsefulCoeficientVector.Count; ++i)

{

double groupCoeff = ClassGroupUsefulCoeficientVector[SortedClassGroupsIndexes[i]];

for (int j = 0; j < ElementsUsefulCoefficients.Count; ++j)

{

if (ElementsUsefulCoefficients[j] == groupCoeff)

{

Console.WriteLine($"U(x{j + 1}) = {groupCoeff}");

}

}

}

}

public void PrintMostUsefulElements()

{

for(int i = 0; i < MostUsefulElementsIndexes.Count; ++i)

{

Console.Write($"x{MostUsefulElementsIndexes[i] + 1} ");

}

}

}

}