Министерство образования Российской Федерации

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н.Э. БАУМАНА

Факультет: Информатика и системы управления Кафедра: Информационная безопасность (ИУ8)

Методы Оптимизации

Лабораторная работа No 3 на тему:

«Решение задачи многокритериальной оптимизации»

Вариант 14 (4)

Преподаватель:

Коннова Н.С.

Студент:

Кузьмина К.А.

Группа:

ИУ8-34

Москва 2023

Цель работы

Изучить постановку задачи многокритериальной оптимизации (МКО); овладеть навыками решения задач МКО с помощью различных методов, выполнить сравнительный анализ результатов, полученных при помощи разных методов.

Постановка задачи

Выбрать лучшую из альтернатив решения предложенной задачи по варианту из табл. 6.1 с точки зрения указанных критериев следующими методами:

1) заменой критериев ограничениями;

2) формированием и сужением множества Парето;

3) методом взвешивания и объединения критериев;

4) методом анализа иерархий.

4	Выбор дороги: A. Автострада; B. Шоссе; C. Грунтовка; D. Проселок	1. Расстояние; 2. Качество покрытия; 3. Контроль; 4. Инфраструктура.	Расстояние: самое большое – по автостраде, чуть меньше – по шоссе, существенно меньше – по грунтовке, самое короткое – по проселку. Качество покрытия: лучшее – на автостраде, существенно хуже – на шоссе, еще хуже – на грунтовке, отсутствует – на проселке. Контроль: самый жесткий – на автостраде, на шоссе – почти такой же жесткий, много мягче – на грунтовке, на проселке – практически отсутствует. Инфраструктура: самая развитая – на шоссе, чуть менее – на автостраде, существенно менее – на грунтовке, на проселке – практически отсутствует.

Ход работы

1) Метод замени критериев ограничениями

1. Составляем вектор весов критериев (с нашей точки зрения), используя шкалу 1÷10.

Расстояние	Качество покрытия	Контроль	Инфраструктура
8	7	9	6

Нормализовав, получим 0,26; 0,24; 0,28; 0,22 ;

	Расстояние	Качество покрытия	Контроль	Инфраструктура
Автострада- A	1	7	1	8
Шоссе- B	3	4	2	10
Грунтовка- C	6	3	7	4
Проселок- D	9	1	9	1

Выберем в качестве главного критерия (критерий 1).

Установим Минимально допустимые уровни для оставшихся критериев

0,1 Amax2

0,8Amax3

0,1Amax4

Проведём нормировку матрицы (кроме столбца главного критерия) по формуле:

Проведём нормировку матрицы (кроме столбца главного критерия) по формуле:

$$A_{ij} = \frac{A_{ij} - A_{minj}}{A_{maxj} - A_{minj}}$$

где A_{minj} и A_{maxj} – минимальное и максимальное значение в столбце соответственно.

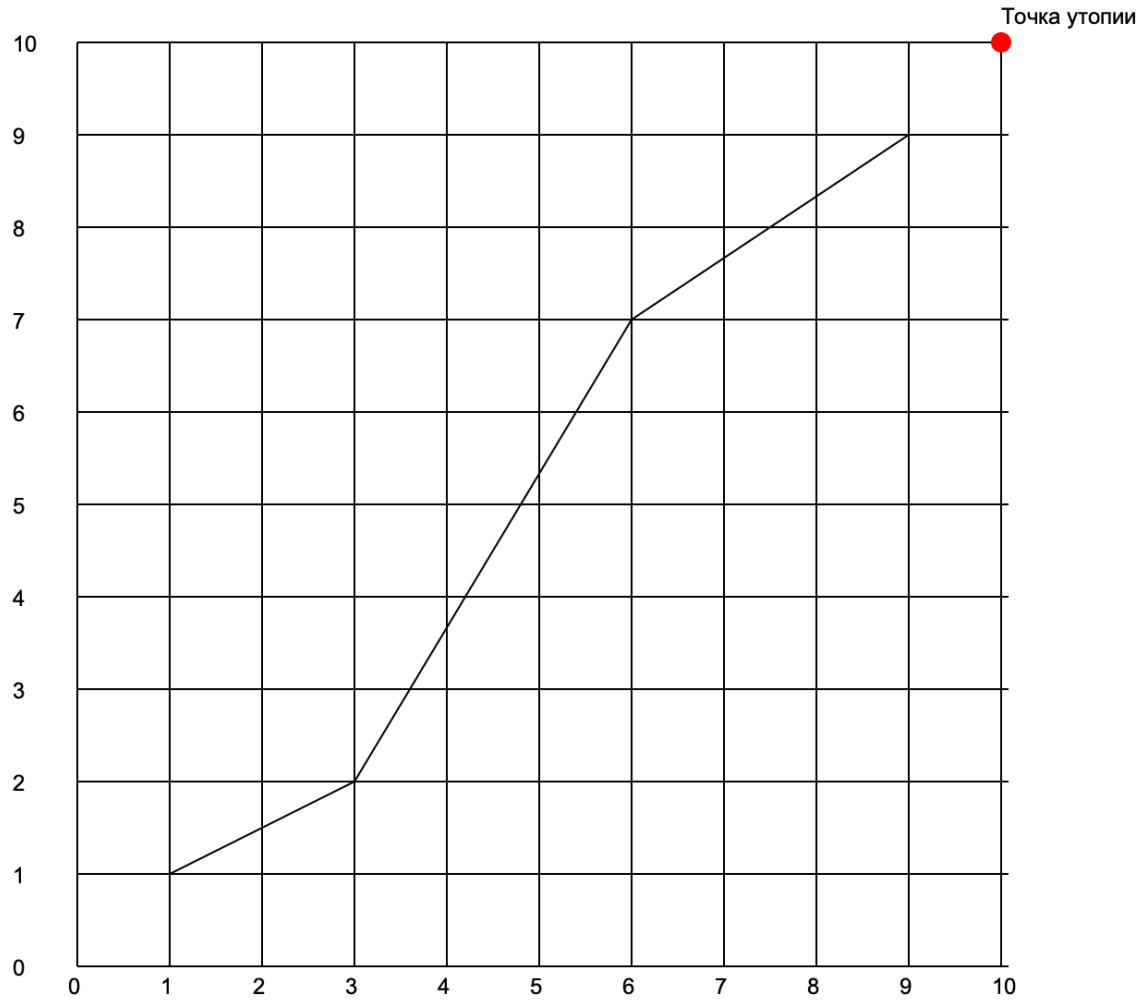
Нормированная матрица A:

	1	2	3	4
A	0	1	0	0,7
B	0,25	0,5	0,125	1
C	0,625	0,3	0,75	0,3
D	1	0	1	0

Проверим удовлетворение минимальным критериям. Удовлетворяют Альтернативы B, C, D. Из них выберем наиболее оптимальный вариант – B(Шоссе).

2) Формирование и сужения множества Парето

Выберем в качестве главных критериев для данного метода Качество лечения и уровень сервиса. Качество лечения – по оси x, уровень сервиса – по y. Сформируем множество Парето графическим методом (см. рис. 1).



Исходя из графика можно сказать, что Манхеттенское расстояние до точки утопии минимально для варианта D (Проселок). А значит, альтернатива D оптимальна

3) Взвешивание и объединение критериев

Составим матрицу рейтингов альтернатив по критериям, используя шкалу 1÷10:

	1	7	1	8
A	3	4	2	10
B	6	6	3	5
C	6	3	7	4
D	9	1	7	1

Нормализуем её:

	1	2	3	4
A	0,053	0,47	0,053	0,35
B	0,158	0,27	0,11	0,44
C	0,316	0,2	0,37	0,174
D	0,474	0,67	0,474	0,044

Составим экспертную оценку критериев (по методу попарного сравнения):

$\gamma_{12}=0,5$; $\gamma_{13}=1$; $\gamma_{14}=0,5$

$\gamma_{34}=0,5$; $\gamma_{23}=0,5$; $\gamma_{24}=0$

Получить вектор весов критериев

a1 =1+0,5+1=2,5

a2=0+0+1=1

a3=1+1+0,5=2,5

a4=0+0+0=0

Нормируем а 0,42 0,17 0,42 0

Умножим нормализованную матрицу на нормализованный вектор весов критериев и получим значения объединенного критерия для всех альтернатив:

0,053	0,47	0,053	0,35		0,47		0,126
0,158	0,27	0,11	0,44	*	0,167	=	0,165
0,316	0,2	0,37	0,174		0,417		0,336
0,474	0,67	0,474	0,044		0		0,532

Как видно из полученной интегральной оценки, наиболее приемлемой является альтернатива D –Проселок

4) Метод анализа иерархий:

Для каждого из критериев составим и нормализуем матрицу попарного сравнения альтернатив:

•Расстояние

	A	B	C	D	Сумма по строке	Нормированная сумма по строке
A	1	1/3	1/6	1/9	1,61	0,053
B	3/1	1	3/6	3/9	4,83	0,158
C	6/1	6/3	1	6/9	9,6	0,314
D	9/1	9/3	9/6	1	14,5	0,475

• Качество покрытия

	A	B	C	D	Сумма по строке	Нормированная сумма по строке
A	1	7/4	7/3	7/1	12,08	0,478
B	4/7	1	4/3	4/1	6,9	0,27
C	3/7	3/4	1	3/1	5,2	0,2
D	1/7	1/4	1/3	1	1,73	0,067

• Контроль

	A	B	C	D	Сумма по строке	Нормированная сумма по строке
A	1	1/2	1/7	1/9	1,75	0,0531
B	2/1	1	2/7	2/9	3,5	0,105
C	7/1	7/2	1	7/9	12,27	0,368
D	9/1	9/2	9/7	1	15,786	0,3474

• Инфраструктура

	A	B	C	D	Сумма по строке	Нормированная сумма по строке
A	1	8/10	8/4	8/1	4,8	0,178
B	10/8	1	10/4	10/1	14,75	0,548
C	4/8	4/10	1	4/1	5,9	0,219
D	1/8	1/10	1/4	1	1,475	0,055

• Оценка приоритетов критериев

	A	B	C	D	Сумма по строке	Нормированная сумма по строке
A	1	8/7	8/9	8/6	4,365	0,267
B	7/8	1	7/9	7/6	3,82	0,233
C	9/8	9/7	1	9/6	4,91	0,3
D	6/8	6/7	6/9	1	3,27	0,199

Составим матрицу критериев, умножим на столбец приоритетов

0,053 0,47 0,053 0,178 0,267 0,295

0,158 0,27 0,105 0,548 * 0,233 = 0,246

0,314 0,2 0,368 0,219 0,3 0,284

0,475 0,067 0,474 0,055 0,199 0,296

Оценив полученный вектор, можем сделать вывод, что оптимальным вариантом является D - Проселок

Вывод

В ходе выполнения работы были изучены и разобраны различные методы решения многокритериальных задач, а именно метод замены критериев ограничениями, метод Парето, метод взвешивания и метод анализа иерархий. Также в ходе работы в различных методах были получены различные оптимальные решения, это связано с тем, что в в первом методе для наглядности использовались отличный главный критерий (в последних трех методах оптимальные решения оказались сравнимо одинаковы)

Приложение А

Lbpomo3

```
namespace lb3
{
   class ChangeMethod
   {
      private ChangeMethod()
      {
      }

      public static string RunChangeMethod(double[][] A, int[] weight, double[] minimalValue, string[] alternative)
      {
         double[] normalizedWeight = NormalizeWeight(weight);
         Console.WriteLine("Нормализованный вектор весов: " + string.Join(", ", normalizedWeight));
         int columns = A[0].Length;

         // Поиск индекса главного критерия
         int index = Array.IndexOf(minimalValue, 1);

         // Поиск максимума и минимума столбцов
         double[] maxFound = new double[columns];
         double[] minFound = new double[columns];
         for (int j = 0; j < columns; j++)
         {
            maxFound[j] = FoundMax(A, j);
            minFound[j] = FoundMin(A, j);
```

```
}

Console.WriteLine($"Максимальные и минимальные элементы столбцов:\nМаксимумы: {string.Join(", ", maxFound)}\nМинимумы: {string.Join(", ", minFound)}");

// Нормировка матрицы
Console.WriteLine("Нормированная матрица A:");
for (int i = 0; i < A.Length; i++)
{
    for (int j = 0; j < columns; j++)
    {
        if (j != index)
        {
            A[i][j] = (A[i][j] - minFound[j]) / (maxFound[j] - minFound[j]);
        }
    }
}

foreach (var row in A)
{
    Console.WriteLine(string.Join(", ", row));
}

// Повторно ищем максимумы и минимумы столбцов, чтобы провести проверку на удовлетворение условий минимальности
for (int j = 0; j < columns; j++)
{
    maxFound[j] = FoundMax(A, j);
}
```

```
for (int j = 0; j < columns; j++)
{
    minFound[j] = FoundMin(A, j);
}

List<int> indexes = new List<int>();
// Проверим минимальное значение для условий
for (int i = 0; i < A.Length; i++)
{
    for (int j = 0; j < A[i].Length; j++)
    {
        if (A[i][j] == maxFound[j] && A[i][j] >= maxFound[j] * minimalValue[j])
        {
            indexes.Add(i);
            break;
        }
    }
}

// Теперь в indexes есть индексы строк, которые удовлетворяют минимальным критериям
// Дальше нужно посчитать значение на нормализованный вес и отдать максимум
Dictionary<int, double> values = new Dictionary<int, double>();
foreach (int i in indexes)
{
    double val = 0;
    bool hasZero = false; // Флаг для проверки наличия нулевых элементов
    for (int j = 0; j < columns; j++)
```

```
    {
        if (j != index)
        {
            if (A[i][j] == 0)
            {
                hasZero = true; // Если есть нулевой элемент, устанавливаем флаг
                break;
            }
            val += A[i][j] * normalizedWeight[j];
        }
    }

    if (!hasZero)
    {
        values.Add(i, val); // Добавляем в словарь только если нет нулевых элементов
    }
}

// Найти максимальное значение и соответствующий индекс
double maxVal = double.NegativeInfinity;
int maxIndex = -1;
foreach (var entry in values)
{
    if (entry.Value > maxVal)
    {
        maxVal = entry.Value;
        maxIndex = entry.Key;
    }
```

```
    }

    return alternative[maxIndex];
}

public static double FoundMax(double[][] A, int j)
{
    double max = double.NegativeInfinity;
    foreach (var row in A)
    {
        if (row[j] > max)
        {
            max = row[j];
        }
    }
    return max;
}

public static double FoundMin(double[][] A, int j)
{
    double min = double.PositiveInfinity;
    foreach (var row in A)
    {
        if (row[j] < min)
        {
            min = row[j];
        }
    }
```

```
    return min;
}

// Функция нормализации вектора весов
public static double[] NormalizeWeight(int[] weight)
{
    double sum = weight.Sum();
    // Создать массив для нормализованных весов
    double[] normalizedWeights = new double[weight.Length];
    // Нормализовать веса
    for (int i = 0; i < weight.Length; i++)
    {
        normalizedWeights[i] = (double)weight[i] / sum;
    }
    return normalizedWeights;
}

static void Main(string[] args)
{
    // Матрица оценок для альтернатив
    double[][] A = {
        new double[] { 1, 7, 1, 8 },
        new double[] { 3, 4, 2, 10 },
        new double[] { 6, 3, 7, 4 },
        new double[] { 9, 1, 9, 1 },
    };
    // Вектор весов
    int[] w = { 8, 7, 9, 6 };
```

```
            // Допустимые уровни для критериев
            double[] a = { 1.0, 0.1, 0.8, 0.1 };
            string[] alternative = { "Автострада", "Шоссе", "Грунтовка", "Проселок" };
            string result = RunChangeMethod(A, w, a, alternative);
            string GREEN = "\u001B[32m";
            string RESET = "\u001B[0m";
            Console.WriteLine(GREEN + "Результат работы программы." + RESET);
            Console.WriteLine("Лучший выбор: " + result);
            Console.ReadLine();
        }
    }
}
```

Lbpomo

```
using Gtk;
using Cairo;

namespace ParetoMethodCSharp
{
    public class ParetoMethod
    {
        public class Point
        {
            public double X { get; set; }
            public double Y { get; set; }

            public Point(double x, double y)
            {
                X = x;
                Y = y;
            }

            public override string ToString()
            {
                return $"({X};{Y})";
            }
        }

        private static bool resultDisplayed = false; // Move the variable here

        private static double ManhattanLength(Point p1, Point p2)
```

```
    {
        return Math.Abs(p1.X - p2.X) + Math.Abs(p1.Y - p2.Y);
    }

    private static void DisplayParetoGraph(List<Point> setPareto, Point utopiaPoint)
    {
        Application.Init();

        double minX = 0;
        double minY = 0;
        double maxX = setPareto.Max(point => point.X);
        double maxY = setPareto.Max(point => point.Y);

        minX = Math.Min(minX, utopiaPoint.X);
        minY = Math.Min(minY, utopiaPoint.Y);
        maxX = Math.Max(maxX, utopiaPoint.X);
        maxY = Math.Max(maxY, utopiaPoint.Y);

        double scale = 50;
        int padding = 100;
        int windowWidth = (int)((maxX - minX) * scale) + 2 * padding;
        int windowHeight = (int)((maxY - minY) * scale) + 2 * padding;

        Window window = new Window("Множество Парето");
        window.Resize(windowWidth, windowHeight);

        DrawingArea area = new DrawingArea();
        area.Drawn += (o, args) => OnDrawEvent(area, setPareto, utopiaPoint);

        window.Add(area);
        window.ShowAll();

        window.Destroyed += (sender, e) => Application.Quit();

        Application.Run();
    }


    private static void OnDrawEvent(DrawingArea area, List<Point> setPareto, Point
utopiaPoint)
    {
        Cairo.Context cr = Gdk.CairoHelper.Create(area.GdkWindow);

        double minX = 0;
        double minY = 0;
        double maxX = setPareto.Max(point => point.X);
        double maxY = setPareto.Max(point => point.Y);

        minX = Math.Min(minX, utopiaPoint.X);
        minY = Math.Min(minY, utopiaPoint.Y);
        maxX = Math.Max(maxX, utopiaPoint.X);
        maxY = Math.Max(maxY, utopiaPoint.Y);
```

```
double scale = 50;
int padding = 100;

cr.SetSourceRGB(1, 1, 1);
cr.Rectangle(0, 0, area.Allocation.Width, area.Allocation.Height);
cr.Fill();

cr.SetSourceRGB(0, 0, 0);
cr.LineWidth = 1;

for (int y = (int)Math.Ceiling(minY); y <= (int)Math.Floor(maxY); y++)
{
    int yPos = (int)((maxY - y) * scale) + padding;
    cr.MoveTo(padding, yPos);
    cr.LineTo(area.Allocation.Width - padding, yPos);
    cr.Stroke();

    cr.SelectFontFace("Arial", FontSlant.Normal, FontWeight.Normal);
    cr.SetFontSize(12);
    cr.MoveTo(padding - 35, yPos + 5);
    cr.ShowText(y.ToString());
}

for (int x = (int)Math.Ceiling(minX); x <= (int)Math.Floor(maxX); x++)
{
    int xPos = (int)((x - minX) * scale) + padding;
    cr.MoveTo(xPos, padding);
    cr.LineTo(xPos, area.Allocation.Height - padding);
    cr.Stroke();

    cr.SelectFontFace("Arial", FontSlant.Normal, FontWeight.Normal);
    cr.SetFontSize(12);
    cr.MoveTo(xPos - 5, area.Allocation.Height - padding + 15);
    cr.ShowText(x.ToString());
}

cr.SetSourceRGB(0, 0, 0);
cr.LineWidth = 1;

setPareto = setPareto.OrderBy(p => p.X).ToList();

for (int i = 0; i < setPareto.Count - 1; i++)
{
    int x1 = (int)((setPareto[i].X - minX) * scale) + padding;
    int y1 = (int)((maxY - setPareto[i].Y) * scale) + padding;

    int x2 = (int)((setPareto[i + 1].X - minX) * scale) + padding;
    int y2 = (int)((maxY - setPareto[i + 1].Y) * scale) + padding;

    cr.MoveTo(x1, y1);
    cr.LineTo(x2, y2);
```

```
            cr.Stroke();
        }

        Point bestPoint = setPareto[0];
        foreach (Point point in setPareto)
        {
            if (ManhattanLength(utopiaPoint, point) < ManhattanLength(utopiaPoint,
bestPoint))
            {
                bestPoint = point;
            }
        }

        int utopiaX = (int)((utopiaPoint.X - minX) * scale) + padding;
        int utopiaY = (int)((maxY - utopiaPoint.Y) * scale) + padding;

        cr.SetSourceRGB(1, 0, 0);
        cr.Arc(utopiaX, utopiaY, 5.5, 0, 2 * Math.PI);
        cr.Fill();

        cr.SetSourceRGB(0, 0, 0);
        cr.SelectFontFace("Arial", FontSlant.Normal, FontWeight.Normal);
        cr.SetFontSize(12);
        cr.MoveTo(utopiaX, utopiaY - 10);
        cr.ShowText("Точка утопии");

        cr.Dispose();

        if (!resultDisplayed)
        {
            string[] alternatives = { "Автострада", "Шоссе", "Грунтовка", "Проселок" };
            Console.WriteLine($"Оптимальный результат:
{alternatives[Array.IndexOf(setPareto.ToArray(), bestPoint)]}");
            resultDisplayed = true;
        }
    }




    private static string RunPareto(double[][] A, string[] alternative, int ind1, int ind2)
    {
        Point utopiaPoint = new Point(10.0, 10.0);
        List<Point> setPareto = A.Select(row => new Point(row[ind1], row[ind2])).ToList();

        Console.Write("Множество Парето: ");
        foreach (Point p in setPareto)
        {
            Console.Write(p.ToString() + " ");
        }
        Console.WriteLine();
```

```
            DisplayParetoGraph(setPareto, utopiaPoint);

            Point bestPoint = setPareto[0];
            foreach (Point point in setPareto)
            {
                if (ManhattanLength(utopiaPoint, point) < ManhattanLength(utopiaPoint,
bestPoint))
                {
                    bestPoint = point;
                }
            }

            int index = -1;
            double minDistance = double.MaxValue;

            for (int i = 0; i < A.Length; i++)
            {
                double distance = ManhattanLength(utopiaPoint, new Point(A[i][ind1], A[i]
[ind2]));
                if (distance < minDistance)
                {
                    minDistance = distance;
                    index = i;
                }
            }

            if (index != -1)
            {
                Console.WriteLine($"Оптимальный результат: {alternative[index]}");
                return alternative[index];
            }
            else
            {
                Console.WriteLine("Оптимальный результат не найден");
                return null;
            }
        }

        static void Main()
        {
            string[] alternative = { "Автострада", "Шоссе", "Грунтовка", "Проселок" };
            double[][] A =
            {
                new double[] { 1, 1 },
                new double[] { 3, 2 },
                new double[] { 6, 7 },
                new double[] { 9, 9 }
            };

            Console.WriteLine(RunPareto(A, alternative, 0, 1));
        }
    }
```

```
}
```

Lbpomo2

```
namespace CriteriaCombination
{
    class Program
    {
        static void Main(string[] args)
        {
            // Карта для γ
            Dictionary<string, double> markCriteria = new Dictionary<string, double>
            {
                {"12", 0.5},
                {"13", 1.0},
                {"14", 1.0},
                {"21", 0.5},
                {"23", 1.0},
                {"24", 1.0},
                {"31", 0.0},
                {"32", 0.0},
                {"34", 1.0},
                {"41", 0.0},
                {"42", 0.0},
                {"43", 0.0}
            };

            // Рейтинг альтернатив по критериям
            double[][] A =
            {
                new double[] {1, 7, 1, 8},
                new double[] {3, 4, 2, 10},
                new double[] {6, 3, 7, 4},
                new double[] {9, 1, 9, 1}
            };

            string[] alternative = { "Автострада", "Шоссе", "Грунтовка", "Проселок" };

            string result = RunCriteriaCombination(A, markCriteria, alternative);

            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("Результат работы программы.");
            Console.WriteLine("Лучший выбор: " + result);
            Console.ResetColor();
            Console.ReadLine();
        }

        public static string RunCriteriaCombination(double[][] A, Dictionary<string, double>
markCriteria, string[] alternative)
        {
            // Нормализуем матрицу
            double[][] normalizeMatrix = NormalizeMatrix(A);
            Console.WriteLine("Нормализованная матрица A:");
```

```
            foreach (var row in normalizeMatrix)
            {
                Console.WriteLine(string.Join(", ", row));
            }

            int columns = A[0].Length;

            // Получаем αi для всего
            List<double> weight = GetWeight(columns, markCriteria);
            Console.WriteLine("Вектор α: " + string.Join(", ", weight));

            // Нормализуем этот вектор
            List<double> normalizeWeight = NormalizeWeight(weight);
            Console.WriteLine("Нормализованный вектор α: " + string.Join(", ",
normalizeWeight));

            // Перемножаем полученные нормализованные матрицы
            double[] multiplyResult = MultiplyMatrixAndWeight(normalizeMatrix,
normalizeWeight);
            Console.WriteLine("Произведение нормализованных матриц: " + string.Join(",
", multiplyResult));

            // Ищем максимальный индекс
            int indexMax = FindMaxIndex(multiplyResult);
            Console.WriteLine("Индекс максимального элемента: " + indexMax);

            return alternative[indexMax];
        }

        public static int FindMaxIndex(double[] array)
        {
            double max = double.NegativeInfinity;
            int maxIndex = -1;

            for (int i = 0; i < array.Length; i++)
            {
                if (array[i] > max)
                {
                    max = array[i];
                    maxIndex = i;
                }
            }

            return maxIndex;
        }

        public static double[] MultiplyMatrixAndWeight(double[][] matrix, List<double>
weight)
        {
            int rows = matrix.Length;
            int cols = matrix[0].Length;
```

```
            if (cols != weight.Count)
            {
                throw new ArgumentException("Несоответствие размеров матрицы и веса");
            }

            double[] result = new double[rows];

            for (int i = 0; i < rows; i++)
            {
                double sum = 0.0;
                for (int j = 0; j < cols; j++)
                {
                    sum += matrix[i][j] * weight[j];
                }
                result[i] = sum;
            }

            return result;
        }

        public static List<double> NormalizeWeight(List<double> weight)
        {
            List<double> normalize = new List<double>();
            double sum = weight.Sum();

            foreach (double w in weight)
            {
                normalize.Add(w / sum);
            }

            return normalize;
        }

        public static List<double> GetWeight(int columns, Dictionary<string, double>
markCriteria)
        {
            List<double> weight = new List<double>();

            for (int i = 0; i < columns; i++)
            {
                double a = 0;
                for (int j = 0; j < columns; j++)
                {
                    if (i != j)
                    {
                        string key = (i + 1) + "" + (j + 1);
                        a += markCriteria[key];
                    }
                }
                weight.Add(a);
            }
```

```
            return weight;
        }

        public static double[][] NormalizeMatrix(double[][] A)
        {
            int columns = A[0].Length;

            for (int j = 0; j < columns; j++)
            {
                double sum = SumColumn(A, j);
                for (int i = 0; i < A.Length; i++)
                {
                    A[i][j] /= sum;
                }
            }

            return A;
        }


        public static double SumColumn(double[][] A, int j)
        {
            double sum = 0;
            foreach (var row in A)
            {
                sum += row[j];
            }

            return sum;



        }
    }
}
```

Lbpomo1

```
namespace org.lb.lb3
{
    public class AnalyticHierarchyProcess
    {
        public static string RunAnalyticHierarchyProcess(string[] alternative,
                                            double[][] A,
                                            double[][] B,
                                            double[][] C,
                                            double[][] D,
                                            double[][] critery)
        {
            List<List<double>> Apoln = GetPoln(A);
            List<List<double>> Bpoln = GetPoln(B);
```

```
        List<List<double>> Cpoln = GetPoln(C);
        List<List<double>> Dpoln = GetPoln(D);
        List<List<double>> critPoln = GetPoln(critery);
        Console.WriteLine("Дополненная матрица A:");
        PrintMatrix(Apoln);
        Console.WriteLine("Дополненная матрица B:");
        PrintMatrix(Bpoln);
        Console.WriteLine("Дополненная матрица C:");
        PrintMatrix(Cpoln);
        Console.WriteLine("Дополненная матрица D:");
        PrintMatrix(Dpoln);
        Console.WriteLine("Дополненная матрица критериев:");
        PrintMatrix(critPoln);

        Console.WriteLine("Матрица из столбцов нормализованных сумм: ");
        List<List<double>> matrix = GetMatrix(Apoln, Bpoln, Cpoln, Dpoln);
        PrintMatrix(matrix);

        Console.WriteLine("Матрица нормализованных сумм критериев: ");
        List<double> critMatrix = new List<double>();
        foreach (List<double> doubles in critPoln)
        {
            critMatrix.Add(doubles[doubles.Count - 1]);
        }
        Console.WriteLine(string.Join(", ", critMatrix));

        List<double> resultVector = MultiplyMatrixAndVector(matrix, critMatrix);
        Console.WriteLine("Результат перемножения матриц: ");
        Console.WriteLine(string.Join(", ", resultVector));

        return alternative[FindMaxIndex(resultVector)];
    }

    public static int FindMaxIndex(List<double> vector)
    {
        double max = double.NegativeInfinity;
        int maxIndex = -1;
        for (int i = 0; i < vector.Count; i++)
        {
            if (vector[i] > max)
            {
                max = vector[i];
                maxIndex = i;
            }
        }
        return maxIndex;
    }

    public static List<double> MultiplyMatrixAndVector(List<List<double>> matrix,
List<double> vector)
    {
        int numColsA = matrix[0].Count;
```

```
            int numRowsB = vector.Count;
            if (numColsA != numRowsB)
            {
                throw new ArgumentException("Несоответствие размеров матрицы и
вектора");
            }
            List<double> result = new List<double>();
            foreach (List<double> doubles in matrix)
            {
                double sum = 0.0;
                for (int j = 0; j < numColsA; j++)
                {
                    sum += doubles[j] * vector[j];
                }
                result.Add(sum);
            }
            return result;
        }

        public static List<List<double>> GetMatrix(List<List<double>> A,
                                            List<List<double>> B,
                                            List<List<double>> C,
                                            List<List<double>> D)
        {
            List<List<double>> res = new List<List<double>>();
            for (int i = 0; i < A.Count; i++)
            {
                List<double> resLine = new List<double>();
                resLine.Add(A[i][A[i].Count - 1]);
                resLine.Add(B[i][B[i].Count - 1]);
                resLine.Add(C[i][C[i].Count - 1]);
                resLine.Add(D[i][D[i].Count - 1]);
                res.Add(resLine);
            }
            return res;
        }

        public static List<List<double>> GetPoln(double[][] matrix)
        {
            List<List<double>> res = new List<List<double>>();
            foreach (double[] row in matrix)
            {
                double sum = 0;
                List<double> resLine = new List<double>();
                foreach (double value in row)
                {
                    resLine.Add(value);
                    sum += value;
                }
                resLine.Add(sum);
                res.Add(resLine);
            }
```

```
    foreach (List<double> row in res)
    {
        double normalizedSum = row[row.Count - 1] / SumColumn(res, row.Count - 1);
        row.Add(normalizedSum);
    }
    return res;
}

public static double SumColumn(List<List<double>> matrix, int columnIndex)
{
    double sum = 0;
    foreach (List<double> row in matrix)
    {
        sum += row[columnIndex];
    }
    return sum;
}

public static void PrintMatrix(List<List<double>> matrix)
{
    foreach (List<double> row in matrix)
    {
        Console.WriteLine(string.Join(", ", row));
    }
}

public static void Main(string[] args)
{

        double[][] A = {
        new double[] {1, 1.0/3, 1.0/6, 1.0/9},
        new double[] {1, 7.0/4, 7.0/3, 7.0/1},
        new double[] {1, 1.0/2, 1.0/7, 1.0/9},
        new double[] {1, 8.0/10, 8.0/4, 8/1}
    };

        double[][] B = {
        new double[] {3.0/1, 1, 3.0/6, 3.0/9},
        new double[] {4.0/7, 1, 4.0/3, 4.0/1},
        new double[] {2.0/1, 1, 2.0/7, 2.0/9},
        new double[] {10.0/8, 1, 10.0/4, 10}
    };

        double[][] D = {
        new double[] {9.0/1, 9.0/3, 9.0/6, 1},
        new double[] {1.0/7, 1.0/4, 1.0/3, 1},
        new double[] {9/1, 9.0/2, 9.0/7, 1},
        new double[] {1.0/8,1.0/10,1.0/4,1}
    };

        double[][] C = {
```

```
            new double[] {6.0/1, 6.0/3, 1, 6.0/9},
            new double[] {3.0/7, 3.0/4, 1, 3/1},
            new double[] {7.0/1, 7.0/2, 1, 7.0/9},
            new double[] {4.0/8, 4.0/10, 1, 4}
        };

            double[][] criteiry = {
            new double[] {1, 8.0/7, 8.0/9, 8.0/6},
            new double[] {7.0/8, 1, 7.0/9, 7.0/6},
            new double[] {9.0/8, 9.0/7, 1, 9.0/6},
            new double[] {6.0/8, 6.0/7, 6.0/9, 1}
        };


            string[] alternative = { "Автострада", "Шоссе", "Грунтовка", "Проселок" };
            string result = RunAnalyticHierarchyProcess(alternative, A, B, C, D, criteiry);
            string GREEN = "\u001B[32m";
            string RESET = "\u001B[0m";
            Console.WriteLine(GREEN + "Результат работы программы." + RESET);
            Console.WriteLine("Лучший выбор: " + result);
            Console.ReadLine();
        }
    }
}
```

Приложение Б

Ссылка на GitHub репозиторий с представленными проектами решения лабораторной работы - https://github.com/yourfavoriteself/lbpomo