

Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Лабораторная работа № 4

«ИССЛЕДОВАНИЕ МЕТОДОВ РЕШЕНИЯ МНОГОКРИТЕРИАЛЬНЫХ
ЗАДАЧ ПРИНЯТИЯ РЕШЕНИЙ НА ОСНОВЕ ПОСТРОЕНИЯ
МНОЖЕСТВА ПАРЕТО»

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

Выполнил:

ст. гр. ИС-17-2о Горбенко К. Н.

Проверил

Кротов К.В.

Севастополь

2020

1 ЦЕЛЬ РАБОТЫ

Исследовать способы формирования множества Парето-оптимальных решений и определения эффективных решений в этом множестве.

2 ЗАДАНИЕ НА РАБОТУ

Вариант 2. Требуется для задаваемого множества X в виде: $X = \{x_i, i = \overline{1,10}\}$ выполнить определение эффективных решений двухкритериальной задачи выбора с использованием метода последовательных уступок. Значения критериев f_1 и f_2 для соответствующих решений x_i ($i = \overline{1,10}$) сведены в матрицу, представленную ниже.

| | f_1 | f_2 |
|----------|-------|-------|
| x_1 | 3 | 2 |
| x_2 | 5 | 6 |
| x_3 | 5 | 3 |
| x_4 | 8 | 4 |
| x_5 | 6 | 2 |
| x_6 | 3 | 8 |
| x_7 | 6 | 4 |
| x_8 | 2 | 5 |
| x_9 | 9 | 2 |
| x_{10} | 3 | 9 |

3 ХОД РАБОТЫ

```

Матрица критериев для решений:
x1 - (3,2)
x2 - (5,6)
x3 - (5,3)
x4 - (8,4)
x5 - (6,3)
x6 - (3,6)
x7 - (6,4)
x8 - (2,5)
x9 - (9,2)
x10 - (3,9)
Парето-граница состоит из решений P(X):
x2 - (5,6)
x4 - (8,4)
x9 - (9,2)
x10 - (3,9)
Убывание критерия K1
x9 - (9,2)
x4 - (8,4)
x2 - (5,6)
x10 - (3,9)
Убывание критерия K2
x10 - (3,9)
x2 - (5,6)
x4 - (8,4)
x9 - (9,2)
Эффективное решение :
x4 - (8,4)

```

Рисунок 1 – Результаты работы

Код программы:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Lab5
{
    public class K1Comparer : IComparer<Solution>
    {
        public int Compare(Solution x, Solution y)
        {
            return x.K1.CompareTo(y.K1);
        }
    }
    public class K2Comparer : IComparer<Solution>
    {
        public int Compare(Solution x, Solution y)
        {
            return x.K2.CompareTo(y.K2);
        }
    }
    public struct Solution
    {
        public int Index;
        public int K1, K2;

        public Solution(int Index, int k1, int k2)
        {
            this.Index = Index;
            this.K1 = k1;
            this.K2 = k2;
        }
    }
}

```

```

    }
    public override string ToString()
    {
        return String.Format("x{0} - ({1},{2})", Index, K1, K2);
    }
}
class Program
{
    static List<Solution> solutions = new List<Solution>();

    static void GetSolutions()
    {
        try
        {
            solutions.Add(new Solution(1, 3, 2));
            solutions.Add(new Solution(2, 5, 6));
            solutions.Add(new Solution(3, 5,3));
            solutions.Add(new Solution(4, 8,4));
            solutions.Add(new Solution(5, 6,3));
            solutions.Add(new Solution(6, 3,6));
            solutions.Add(new Solution(7, 6,4));
            solutions.Add(new Solution(8, 2,5));
            solutions.Add(new Solution(9,9,2 ));
            solutions.Add(new Solution(10,3,9));
        }
        catch
        {
        }
    }
}
static int CompareSolutions(Solution s1, Solution s2)
{
    int result = 1;
    if (s1.K1 >= s2.K1 && s1.K2 >= s2.K2)
    {
        if (s1.K1 > s2.K1 || s1.K2 > s2.K2)
        {
            result = 1;
        }
        else
            result = 0;
    }
    else
    {
        result = 0;
    }
    return result;
}
static List<Solution> Poisk(List<Solution> PX1, List<Solution> PX2)
{
    var list = new List<Solution>();
    Solution maxSolution;
    int delta1 = 0;
    int delta2 = 0;
    int i1 = 1, i2 = 1;
    while (true)
    {
        if (PX1[i1].K1 == PX2[i2].K1 && PX1[i1].K2 == PX2[i2].K2)
        {
            maxSolution = PX1[i1];
            list.Add(maxSolution);
            return list;
        }
        else
        {
            if (PX1[i1].K1 == PX2[i2 - 1].K1 && PX2[i2].K2 == PX1[i1 - 1].K2)
            {
                delta1 += PX1[i1].K1 - PX1[i1 - 1].K1;
            }
        }
    }
}

```

```

        delta2 += PX2[i2].K2 - PX2[i2 - 1].K2;
        if (delta1 > delta2)
        {
            maxSolution = PX1[i1];
            list.Add(maxSolution);
            return list;
        }
        if (delta1 < delta2)
        {
            maxSolution = PX2[i2];
            list.Add(maxSolution);
            return list;
        }
        if (delta1 == delta2)
        {
            maxSolution = PX1[i1];
            list.Add(maxSolution);
            return list;
        }
    }
    else
    {
        delta1 += PX1[i1].K1 - PX1[i1 - 1].K1;
        delta2 += PX2[i2].K2 - PX2[i2 - 1].K2;
        if (delta1 > delta2)
        {
            i2++;
        }
        else if (delta1 < delta2)
        {
            i1++;
        }
        else if (delta1 == delta2)
        {
            i1++;
            i2++;
        }
    }
}
}
return list;
}
static List<Solution> SortRegrK1(List<Solution> list)
{
    var array = list.ToList();
    array.Sort(new K1Comparer());
    array.Reverse();
    return array;
}
static List<Solution> SortRegrK2(List<Solution> list)
{
    var array = list.ToList();
    array.Sort(new K2Comparer());
    array.Reverse();
    return array;
}
static List<Solution> Pareto(List<Solution> list)
{
    var solutions = new List<Solution>();
    var n = list.Count;
    var matrix = new int[n, n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            if (i == j) continue;
            matrix[i, j] = CompareSolutions(list[i], list[j]);
        }
    }
}

```

```

    }
    for (int j = 0; j < n; j++)
    {
        var sum = 0;
        for (int i = 0; i < n; i++)
        {
            sum += matrix[i, j];
        }
        if (sum == 0)
        {
            solutions.Add(list[j]);
        }
    }
    return solutions;
}
static void Main(string[] args)
{
    GetSolutions();
    Console.WriteLine("Матрица критериев для решений:");
    foreach (var x in solutions)
    {
        Console.WriteLine(x);
    }
    var pareto = Pareto(solutions);
    Console.WriteLine("Парето-граница состоит из решений P(X):");
    foreach (var x in pareto)
    {
        Console.WriteLine(x);
    }
    Console.WriteLine("Убывание критерия K1");
    var k1max = SortRegrK1(pareto);
    foreach (var item in k1max)
    {
        Console.WriteLine(item);
    }

    Console.WriteLine("Убывание критерия K2");
    var k2max = SortRegrK2(pareto);
    foreach (var item in k2max)
    {
        Console.WriteLine(item);
    }

    Console.WriteLine("Эффективное решение :");
    var maxSolution = Poisk(k1max, k2max);
    foreach (var s in maxSolution)
    {
        Console.WriteLine(s);
    }
    Console.ReadLine();
}
}
}

```

ВЫВОДЫ

В ходе данной лабораторной работы был исследован метод последовательных уступок, позволяющий сформировать Парето-границу и выявить эффективное решение. Была написана программа, реализующая этот метод.

Эффективное решение $x_4(8,4)$.