Министерство образования и науки Российской Федерации

Севастопольский государственный университет

Кафедра ИС

Лабораторная работа №5

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

Выполнил:

ст. гр. И-31д

Малиновский А.А.

Проверил:

Токарев А.И.

Севастополь

2014

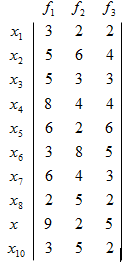
1. ЦЕЛЬ РАБОТЫ

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

1. ЗАДАНИЕ НА РАБОТУ
   1. Для первого и третьего вариантов в соответствии с заданием необходимо реализовать следующий порядок действий для выполнения лабораторной работы:
2. разработать процедуру определения на основе задаваемого множества решений Х и соответствующих им значений критериев и множества Р(Х), представляющего собой Парето-границу Х;
3. разработать процедуру определения координат идеальной точки (точки утопии);
4. разработать процедуру расчета расстояния до точки утопии для координат текущего рассматриваемого решения;
5. разработать процедуру определения эффективного решения, расстояние до которого от идеальной точки является минимальным.
6. ВАРИАНТ ЗАДАНИЯ

Вариант 3. Требуется для задаваемого множества Х в виде:  выполнить определение эффективных решений трехкритериальной задачи выбора с использованием метода идеальной точки. Значения критериев ,  и  для соответствующих решений  () сведены в матрицу, представленную ниже.

Таблица 1. Значения критериев и решений



1. ТЕКСТ ПРОГРАММЫ

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace ТПР5

{

public partial class Solution : Form

{

public Solution()

{

InitializeComponent();

dataGridView.Columns.Add("f1","f1");

dataGridView.Columns.Add("f2", "f2");

dataGridView.Columns.Add("f3", "f3");

dataGridView.Rows.Add(3, 2, 2);

dataGridView.Rows[0].HeaderCell.Value = "x1";

dataGridView.Rows.Add(5, 6, 4);

dataGridView.Rows[1].HeaderCell.Value = "x2";

dataGridView.Rows.Add(5, 3, 3);

dataGridView.Rows[2].HeaderCell.Value = "x3";

dataGridView.Rows.Add(8, 4, 4);

dataGridView.Rows[3].HeaderCell.Value = "x4";

dataGridView.Rows.Add(6, 2, 6);

dataGridView.Rows[4].HeaderCell.Value = "x5";

dataGridView.Rows.Add(3, 8, 5);

dataGridView.Rows[5].HeaderCell.Value = "x6";

dataGridView.Rows.Add(6, 4, 3);

dataGridView.Rows[6].HeaderCell.Value = "x7";

dataGridView.Rows.Add(2, 5, 2);

dataGridView.Rows[7].HeaderCell.Value = "x8";

dataGridView.Rows.Add(9, 2, 5);

dataGridView.Rows[8].HeaderCell.Value = "x9";

dataGridView.Rows.Add(3, 5, 2);

dataGridView.Rows[9].HeaderCell.Value = "x10";

List<List<double>> ValueOfTheSolutionAndCriteria = ReadingFromDataGrid(dataGridView);

List<double> idealPointValues = CalculatinTheValueOfIdealPoints(ValueOfTheSolutionAndCriteria, comboBoxSearchDirection.Text);

FillingLinesList(idealPointValues, dataGridViewUtopiaPointValue, "f");

SortedList<int, List<double>> matrixOfIncomparableSolutions;

if (comboBoxSearchDirection.Text == "Максимум")

{

matrixOfIncomparableSolutions = CalculationOfMatrixSetsOfIncomparableSolutions\_XMax(ValueOfTheSolutionAndCriteria);

}

else

{

matrixOfIncomparableSolutions = CalculationOfMatrixSetsOfIncomparableSolutions\_XMin(ValueOfTheSolutionAndCriteria);

}

FillingMatrixSortedList(matrixOfIncomparableSolutions, dataGridViewUnmatchedSolutions, "f", "x");

SortedList<int,double> DistanceToTheIdealPoint = CalculatingTheDistanceToTheIdealPoint(idealPointValues, matrixOfIncomparableSolutions);

FillingColumnSortedList(DistanceToTheIdealPoint, dataGridViewDistanceToTheIdealPoint, "x");

dataGridViewDistanceToTheIdealPoint.Rows[DistanceToTheIdealPoint.IndexOfValue(DistanceToTheIdealPoint.Values.Min())].Cells[0].Style.BackColor = Color.Coral;

}

List<List<double>> ReadingFromDataGrid(DataGridView dataGridView)

{

List<List<double>> more\_one\_matrix = new List<List<double>>();

for (int i = 0; i < dataGridView.Rows.Count - 1; i++)

{

more\_one\_matrix.Add(new List<double>());

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

{

try

{

more\_one\_matrix[i].Add(double.Parse(dataGridView.Rows[i].Cells[j].Value.ToString()));

}

catch (Exception e)

{

MessageBox.Show(e.Message);

return null;

}

}

}

return more\_one\_matrix;

}

void FillingMatrixSortedList(SortedList<int, List<double>> fill\_matrix, DataGridView dataGridView, string nameColumns, string nameRows)

{

for (int i = 0; i < fill\_matrix[fill\_matrix.Keys[0]].Count; i++)

{

dataGridView.Columns.Add(nameColumns + (i + 1), nameColumns + (i + 1));

}

int k = 0;

foreach (int i in fill\_matrix.Keys)

{

dataGridView.Rows.Add();

dataGridView.Rows[k].HeaderCell.Value = nameRows + (i + 1);

for (int j = 0; j < fill\_matrix[i].Count; j++)

{

dataGridView.Rows[k].Cells[j].Value = fill\_matrix[i][j] + "";

}

k++;

}

}

void FillingLinesList(List<double> fill\_vector, DataGridView dataGridView, string nameColumns)

{

for (int i = 0; i < fill\_vector.Count; i++)

{

dataGridView.Columns.Add(nameColumns + (i + 1), nameColumns + (i + 1));

}

dataGridView.Rows.Add();

for (int i = 0; i < fill\_vector.Count; i++)

{

dataGridView.Rows[0].Cells[i].Value = fill\_vector[i] + "";

}

}

void FillingColumnSortedList(SortedList<int,double> key\_vector, DataGridView dataGridView, string nameRows)

{

dataGridView.Columns.Add("Растояние", "Растояние");

int k = 0;

foreach (int i in key\_vector.Keys)

{

dataGridView.Rows.Add(key\_vector[i]);

dataGridView.Rows[k].HeaderCell.Value = nameRows + (i + 1);

k++;

}

}

List<double> CalculatinTheValueOfIdealPoints(List<List<double>> valueOfTheSolutionAndCriteria, string theSearchDirection)

{

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

if (theSearchDirection == "Максимум")

{

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

{

idealPointValues.Add(0);

}

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

{

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

{

if (valueOfTheSolutionAndCriteria[i][j] > idealPointValues[j])

{

idealPointValues[j] = valueOfTheSolutionAndCriteria[i][j];

}

}

}

}

else

{

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

{

idealPointValues.Add(double.MaxValue);

}

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

{

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

{

if (valueOfTheSolutionAndCriteria[i][j] < idealPointValues[j])

{

idealPointValues[j] = valueOfTheSolutionAndCriteria[i][j];

}

}

}

}

return idealPointValues;

}

public SortedList<int, List<double>> CalculationOfMatrixSetsOfIncomparableSolutions\_XMax(

List<List<double>> matrixOfScalarRatings)

{

SortedList<int, List<double>> matrixSetsOfIncomparableSolutionsX = new SortedList<int, List<double>>();

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

{

matrixSetsOfIncomparableSolutionsX.Add(i, matrixOfScalarRatings[i]);

}

for (int k = 0; k < matrixOfScalarRatings.Count; k++)

{

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

{

bool rankGreaterThanOrEqualRatingsK = true;

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

{

if (matrixOfScalarRatings[k][j] < matrixOfScalarRatings[i][j] && i != k)

{

rankGreaterThanOrEqualRatingsK = false;

}

}

bool oneEstimatesMoreThanOneIJRatings = false;

if (rankGreaterThanOrEqualRatingsK == true)

{

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

{

if (matrixOfScalarRatings[k][j] > matrixOfScalarRatings[i][j] && i != k)

{

oneEstimatesMoreThanOneIJRatings = true;

break;

}

}

}

if (rankGreaterThanOrEqualRatingsK == true && oneEstimatesMoreThanOneIJRatings == true)

{

matrixSetsOfIncomparableSolutionsX.Remove(i);

}

}

}

return matrixSetsOfIncomparableSolutionsX;

}

public SortedList<int, List<double>> CalculationOfMatrixSetsOfIncomparableSolutions\_XMin(

List<List<double>> matrixOfScalaRatings)

{

SortedList<int, List<double>> matrixSetsOfIncomparableSolutionsX = new SortedList<int, List<double>>();

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

{

matrixSetsOfIncomparableSolutionsX.Add(i, matrixOfScalaRatings[i]);

}

for (int k = 0; k < matrixOfScalaRatings.Count; k++)

{

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

{

bool jRankGreaterThanOrEqualiRatings = true;

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

{

if (matrixOfScalaRatings[k][j] > matrixOfScalaRatings[i][j] && i != k)

{

jRankGreaterThanOrEqualiRatings = false;

}

}

bool oneJKEstimatesMoreThanOneJIRatings = false;

if (jRankGreaterThanOrEqualiRatings == true)

{

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

{

if (matrixOfScalaRatings[k][j] < matrixOfScalaRatings[i][j] && i != k)

{

oneJKEstimatesMoreThanOneJIRatings = true;

break;

}

}

}

if (jRankGreaterThanOrEqualiRatings == true && oneJKEstimatesMoreThanOneJIRatings == true)

{

matrixSetsOfIncomparableSolutionsX.Remove(i);

}

}

}

return matrixSetsOfIncomparableSolutionsX;

}

public SortedList<int,double> CalculatingTheDistanceToTheIdealPoint(List<double> idealPointValues, SortedList<int, List<double>> theSetOfIncomparableSolutions)

{

SortedList<int,double> DistanceToTheIdealPoint = new SortedList<int,double>();

foreach (int i in theSetOfIncomparableSolutions.Keys)

{

DistanceToTheIdealPoint.Add(i,0);

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

{

DistanceToTheIdealPoint[i] += Math.Pow(idealPointValues[j] - theSetOfIncomparableSolutions[i][j], 2);

}

DistanceToTheIdealPoint[i] = Math.Sqrt(DistanceToTheIdealPoint[i]);

}

return DistanceToTheIdealPoint;

}

private void buttonДобавитьКритерий\_Click(object sender, EventArgs e)

{

dataGridView.Columns.Add("f" + dataGridView.Columns.Count + "", "f" + dataGridView.Columns.Count + "");

}

private void buttonВычислить\_Click(object sender, EventArgs e)

{

dataGridViewUtopiaPointValue.Columns.Clear();

dataGridViewUtopiaPointValue.Rows.Clear();

dataGridViewUnmatchedSolutions.Columns.Clear();

dataGridViewUnmatchedSolutions.Rows.Clear();

dataGridViewDistanceToTheIdealPoint.Columns.Clear();

dataGridViewDistanceToTheIdealPoint.Rows.Clear();

List<List<double>> ValueOfTheSolutionAndCriteria=ReadingFromDataGrid(dataGridView);

if (ValueOfTheSolutionAndCriteria != null&&ValueOfTheSolutionAndCriteria.Count!=0)

{

List<List<double>> valueOfTheSolutionAndCriteria = ReadingFromDataGrid(dataGridView);

List<double> idealPointValues = CalculatinTheValueOfIdealPoints(valueOfTheSolutionAndCriteria, comboBoxSearchDirection.Text);

FillingLinesList(idealPointValues, dataGridViewUtopiaPointValue, "f");

SortedList<int, List<double>> matrixOfIncomparableSolutions;

if (comboBoxSearchDirection.Text == "Максимум")

{

matrixOfIncomparableSolutions = CalculationOfMatrixSetsOfIncomparableSolutions\_XMax(valueOfTheSolutionAndCriteria);

}

else

{

matrixOfIncomparableSolutions = CalculationOfMatrixSetsOfIncomparableSolutions\_XMin(valueOfTheSolutionAndCriteria);

}

FillingMatrixSortedList(matrixOfIncomparableSolutions, dataGridViewUnmatchedSolutions, "f", "x");

SortedList<int, double> DistanceToTheIdealPoint = CalculatingTheDistanceToTheIdealPoint(idealPointValues, matrixOfIncomparableSolutions);

FillingColumnSortedList(DistanceToTheIdealPoint, dataGridViewDistanceToTheIdealPoint, "x");

dataGridViewDistanceToTheIdealPoint.Rows[DistanceToTheIdealPoint.IndexOfValue(DistanceToTheIdealPoint.Values.Min())].Cells[0].Style.BackColor = Color.DarkOrange;

}

}

private void buttonУдалитьКритерий\_Click(object sender, EventArgs e)

{

if (dataGridView.Columns.Count > 2)

{

dataGridView.Columns.Remove("f" + dataGridView.Columns.Count + "");

}

}

private void dataGridView\_RowsAdded(object sender, DataGridViewRowsAddedEventArgs e)

{

dataGridView.Rows[dataGridView.Rows.Count - 1].HeaderCell.Value = "x" + dataGridView.Rows.Count;

}

private void buttonВыход\_Click(object sender, EventArgs e)

{

Dispose();

}

}

}

1. ТЕСТОВЫЕ ПРИМЕРЫ

На рисунках 1-2 изображены тестовые примеры работы программы.

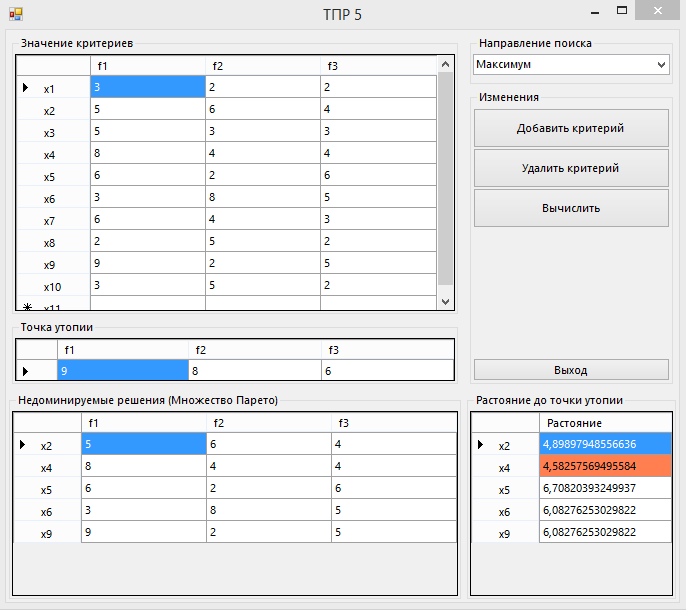


Рисунок 1 – Главное меню, задание значений и решение на максимум

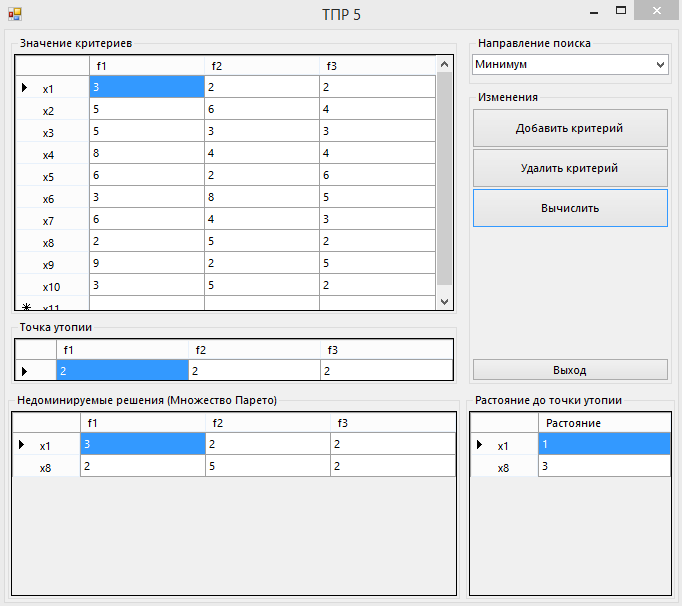


Рисунок 2 – Главное меню, задание значений и решение на минимум

ВЫВОДЫ

В данной лабораторной работе были исследованы способы формирования множества Парето-оптимальных решений и определения эффективных решений в этом множестве.