****

**Министерство науки и высшего образования Российской Федерации**

**МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Н. Э. БАУМАНА**

Факультет: «Информатика и системы управления»

Кафедра: «Информационная безопасность (ИУ8)» / ГУИМЦ

**ТЕОРИЯ СИСТЕМ И СИСТЕМНЫЙ АНАЛИЗ**

**Лабораторная работа №5 на тему:**

«Решение задачи многокритериальной оптимизации»

Вариант 4

**Преподаватель:** Коннова Н. С.

**Студент:** Месяцева Н. В.

**Группа**: ИУ8Ц-51

Москва

2019

# **Цель работы**

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

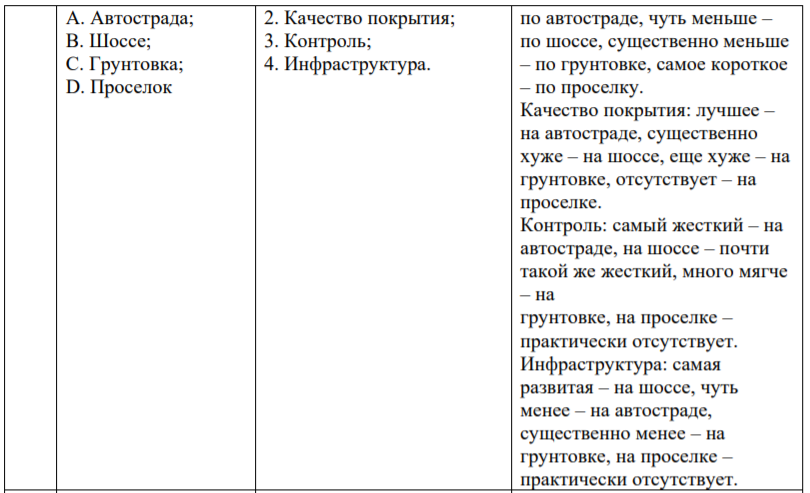
# **Постановка задачи**

Выбрать лучшую из альтернатив решения предложенной задачи по варианту из табл. 5.1 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.



# **Ход работы**

**1) Метод замены критериев ограничениями**

Составим матрицу A оценок для альтернатив:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дороги\Критерии | 1.Расстояние | 2. Качество покрытия | 3. Контроль | 4. Инфраструктура |
| Автострада | 2 | 8 | 1 | 7 |
| Шоссе | 4 | 5 | 2 | 8 |
| Грунтовка | 6 | 3 | 6 | 3 |
| Просёлок | 8 | 1 | 9 | 1 |

Выберем в качестве главного критерия качество покрытия (критерий 2).

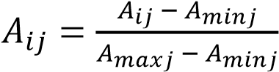
Установим минимально допустимые уровни для остальных критериев:

Расстояние: не менее 0.6 \* .

Контроль: не менее 0.1 \*

Инфраструктура: не менее 0.2 \* .

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

 , где  и  – минимальное и максимальное значение в столбце

соответственно.

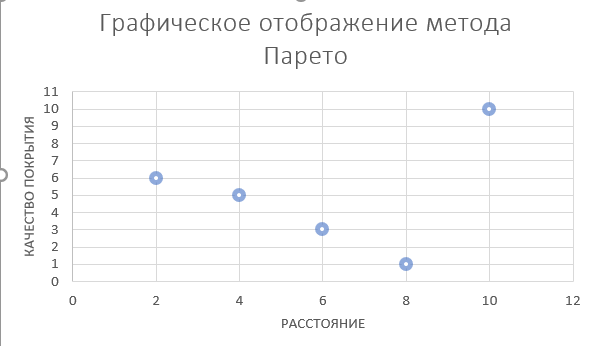
Получим следующую матрицу:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дороги\Критерии | 1.Расстояние | 2. Качество покрытия | 3. Контроль | 4. Инфраструктура |
| Автострада | 0 | 8 | 0 | 0.857143 |
| Шоссе | 0.33333 | 5 | 0.125 | 1 |
| Грунтовка | 0.666667 | 3 | 0.625 | 0.285714 |
| Просёлок | 1 | 1 | 1 | 0 |

Здесь оптимальный вариант – Грунтовка.

**2) Метод Парето**

Выберем в качестве главных критериев для данного метода расстояние и качество покрытия. Сформируем множество Парето на графике. Оба критерия максимизируются, поэтому точка утопии находится в правом верхнем углу графика (*координаты (10;10)*).



Оценка: (2; 8): Автострада

Оценка: (4; 5): Шоссе

Оценка: (6; 3): Грунтовка

Оценка: (8; 1): Проселок

С помощью Евклидова расстояния выяснено, что наилучшим (оптимальным) выбором будет Шоссе.

**3) Взвешивание и объединение критериев**

На основе существующей матрицы критериев нормализуем её:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Дороги\Критерии | 1.Расстояние | 2. Качество покрытия | 3. Контроль | 4. Инфраструктура |
| Автострада | 0.1 | 0.470588 | 0.0555556 | 0.368421 |
| Шоссе | 0.2 | 0.294118 | 0.111111 | 0.421053 |
| Грунтовка | 0.3 | 0.176471 | 0.333333 | 0.157895 |
| Просёлок | 0.4 | 0.0588235 | 0.5 | 0.0526316 |

Составим экспертную оценку критериев (по методу попарного сравнения):

y12 = 0.5 => одинаково важны приоритеты дистанции и покрытия => y21 = 0.5

y23 = 1 => отдаю приоритет покрытию дороги, нежели контролю, а значит y32 = 0 , т.е. мне не важен контроль по сравнению с качеством покрытия y34 = 0 => отдаю приоритет инфраструктуре, т.е. y43 = 1; y13 = 1; y24 = 1; y14 = 0.

Составляем формулу, чтобы получить вектор критерий для матрицы: критерий между собой не рассматривается, т.е. y11, y22, y33 не входят в оценку

a1 = y12 + y13 + y14 = 0.5 + 1 + 0.5 = 2

a2 = y21 + y23 + y24 = 0.5 + 1 + 1 = 2.5

a3 = y31 + y32 + y34 = 0 + 0 + 0 = 0

a4 = y41 + y42 + y43 = 0.5 + 0 + 1 = 1.5

Получим вектор-столбец (2, 2.5, 0, 1.5) => сумма всех элементов 6

Нормализуем, поделив на сумму всех элементов, и получим:

(0.333, 0.416, 0, 0.25)

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

(0.32117, 0.294216, 0.212785, 0.170828)

Как видно из полученной интегральной оценки, наиболее приемлемой является альтернатива Автострада.

1. **Метод анализа иерархий**

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

*Расстояние*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Автострада*** | ***Шоссе*** | ***Грунтовка*** | ***Просёлок*** | ***Сумма по строке*** | ***Нормированная сумма по строке*** |
| Автострада | 1 | 0.333 | 0.2 | 0.142 | 3.4 | 1.102 |
| Шоссе | 3 | 1 | 0.333 | 0.2 | 1.785 | 0.054 |
| Грунтовка | 5 | 3 | 1 | 0.333 | 14 | 0.421 |
| Просёлок | 7 | 5 | 3 | 1 | 14 | 0.421 |

*Качество покрытия*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Автострада*** | ***Шоссе*** | ***Грунтовка*** | ***Просёлок*** | ***Сумма по строке*** | ***Нормированная сумма по строке*** |
| Автострада | 1 | 5 | 7 | 9 | 3.4 | 1.102 |
| Шоссе | 0.2 | 1 | 3 | 5 | 1.785 | 0.054 |
| Грунтовка | 0.142 | 0.333 | 1 | 3 | 14 | 0.421 |
| Просёлок | 0.111 | 0.2 | 0.333 | 1 | 14 | 0.421 |

*Контроль*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Автострада*** | ***Шоссе*** | ***Грунтовка*** | ***Просёлок*** | ***Сумма по строке*** | ***Нормированная сумма по строке*** |
| Автострада | 1 | 0.333 | 0.142 | 0.111 | 3.4 | 1.102 |
| Шоссе | 3 | 1 | 0.2 | 0.111 | 1.785 | 0.054 |
| Грунтовка | 7 | 5 | 1 | 0.2 | 14 | 0.421 |
| Просёлок | 9 | 9 | 5 | 1 | 14 | 0.421 |

*Инфраструктура*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***Автострада*** | ***Шоссе*** | ***Грунтовка*** | ***Просёлок*** | ***Сумма по строке*** | ***Нормированная сумма по строке*** |
| Автострада | 1 | 0.333 | 5 | 7 | 3.4 | 1.102 |
| Шоссе | 3 | 1 | 7 | 9 | 1.785 | 0.054 |
| Грунтовка | 0.2 | 0.142 | 1 | 3 | 14 | 0.421 |
| Просёлок | 0.142 | 0.111 | 0.333 | 1 | 14 | 0.421 |

*Оценка приоритетов критериев*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | ***1*** | ***2*** | ***3*** | ***4*** | ***Сумма по строке*** | ***Нормированная сумма по строке*** |
| ***1*** | 1 | 0.333 | 5 | 3 | 16 | 0.507 |
| ***2*** | 3 | 1 | 7 | 5 | 9.333 | 0.295 |
| ***3*** | 0.2 | 0.142 | 1 | 0.333 | 12.2 | 0.144 |
| ***4*** | 0.333 | 0.2 | 3 | 1 | 1.676 | 0.053 |

Составим матрицу (i – альтернатива, j - критерий) и умножим на столбец оценки приоритетов:

0.0531055 0.143718 0.295901 0.507276 0. 295901 0.177237

0.589512 0.246523 0.119912 0.0440526 × 0. 507276 = 0.312191

0.0368 0.10003 0.306286 0.556883 0. 0531055 0.157932

0.339599 0.509411 0.110593 0.0403963 0. 143718 0.370579

Оценив полученный вектор, можем сделать вывод, что оптимальным вариантом является Просёлок.

# **Вывод**

При решении многокритериальной задачи различными методами было обнаружено, что во всех методах наилучшие выборы дорог отличаются. Это говорит о том, что нам стоит поиграть с параметрами и экспертным выбором собственных критериев.

# **Листинг**

*Программа написана на языке С++ в Visual Studio 2017.*

*Файл HeaderFile.hpp*

#ifndef INCLUDE\_HEADER\_HPP\_

#define INCLUDE\_HEADER\_HPP\_

#include <iostream>

#include <map>

#include <vector>

#include <string>

#include <numeric>

**using** **namespace** std;

// структура "Дороги"

**struct** Rating

{

string road\_name;

vector<**double**> criteries;

Rating();

//Для таблицы с оценками

Rating(**const** string &name, **const** **double** &distance, **const** **double** &roadQuality, **const** **double** &control, **const** **double** &infrastructure);

};

// Конструкторы

Rating::Rating()

{

road\_name = "";

criteries = { **0**, **0**, **0**, **0** };

}

Rating::Rating(**const** string &name, **const** **double** &distance, **const** **double** &roadQuality, **const** **double** &control, **const** **double** &infrastructure)

{

road\_name = name;

criteries = { distance, roadQuality, control, infrastructure };

}

// Вывод таблицы

**void** OutputTable(**const** vector<Rating>& vectorRatings);

// Методы внутри структуры Rating: 1. Замена критериев

string **ChangeCriterionMethod**(**const** vector<Rating>& Ratings, **const** **int**& main\_criterion, **const** vector<**double**>& part);

// 2. Метод Парето

**double** **EvklidovoDistance**(**const** **double**& x\_max, **const** **double**& y\_max, **const** **double**& x\_from\_main\_critery, **const** **double**& y\_from\_second\_critery);

Rating **ParetoMethod**(**const** vector<Rating>& Ratings, **const** pair<**int**, **int**>& pareto\_pair);

// 3. Метод объединения и взвешивания

string **WeightAndCombMethod**(**const** vector<Rating> Ratings, **const** vector<**double**>& normalizeVector);

// 4. Метод анализа иерархии

string **Compare**(**const** vector<Rating>& Ratings, **const** vector<**double**>& PriorityVect);

#endif // INCLUDE\_HEADER\_HPP\_

*Файл Lab05.cpp*

#include "pch.h"

// Вызов созданного нами заголовочного файла

#include "HeaderFile.hpp"

#include <cmath>

// Скопировано из заголовочного файла

#include <iostream>

#include <map>

#include <vector>

#include <string>

#include <numeric>

**using** **namespace** std;

**int** **main**()

{

setlocale(LC\_ALL, "Russian");

// Создание таблицы и ее вывод

vector<Rating> Ratings(**4**);

Ratings[**0**] = Rating("Автострада", **2**, **8**, **1**, **7**);

Ratings[**1**] = Rating("Шоссе ", **4**, **5**, **2**, **8**);

Ratings[**2**] = Rating("Грунтовка ", **6**, **3**, **6**, **3**);

Ratings[**3**] = Rating("Проселок ", **8**, **1**, **9**, **1**);

cout << "**\t\t** Таблица оценок";

OutputTable(Ratings);

//// 1. Метод замены критериев

**int** main\_criterion = **1**; // Индекс = 1 => главный критерий - "Качество дороги" (2 по счету в массиве

/\*string criterion;

if (main\_criterion == 1) criterion = "Качество дороги";\*/

cout << "1. Замена критериев ограничениями: главный критерий - Качество дороги" << endl;

vector<**double**> part = { **0.6**, **1**, **0.1**, **0.2** };

string max\_ch = ChangeCriterionMethod(Ratings, main\_criterion, part);

cout << " Замена критериев ограничениями: наилучший выбор - " << max\_ch << "**\n**";

// 2. Метод Парето

pair<**int**, **int**> main\_pair = { **0**, **1** };

Rating best\_choice = ParetoMethod(Ratings, main\_pair);

cout << "**\n** Метод Парето - выбор: " << best\_choice.road\_name << "**\n**";

// 3. Метод взвешивания и объединения критериев

cout << "**\n** 3. Метод взвешивания попарных сравнений с последующим объединением";

vector<**double**> normalizeVector = { **0.333**, **0.416**, **0**, **0.25** };

string result = WeightAndCombMethod(Ratings, normalizeVector);

cout << "**\n** Метод взвешивания и объединения критериев: Выбор - " << result << "**\n**";

// 4. Метод анализа иерархий

cout << endl << "4. Метод анализа иерархий" << endl;

vector<**double**> PriorityVect = { **6**, **8**, **2**, **4** }; // на основе экспертного решения

result = Compare(Ratings, PriorityVect);

cout << "**\n** 4. Метод анализа иерархий: Выбор - " << result << "**\n**";

system("pause");

}

// Вывод таблицы

**void** **OutputTable**(**const** vector<Rating>& vectorRatings)

{

cout << endl << "------------------------------------------------**\n**";

cout << "| Наименование | Дист. | Покр. | Конт. | Инфр. |" << endl;

cout << "|--------------|-------|-------|-------|-------|";

**for** (**size\_t** j = **0**; j < vectorRatings.size(); j++)

{

cout << "**\n**";

cout << "| " << vectorRatings[j].road\_name << " | ";

**for** (**size\_t** i = **0**; i < vectorRatings[j].criteries.size(); i++)

{

cout << vectorRatings[j].criteries[i] << " | ";

}

}

cout << endl << "------------------------------------------------**\n\n**";

}

// 1. Замена критериев

string **ChangeCriterionMethod**(**const** vector<Rating>& Ratings, **const** **int**& main\_criterion, **const** vector<**double**>& part)

{

vector<**double**> max\_values = Ratings[**0**].criteries;

vector<**double**> min\_values = Ratings[**0**].criteries;

// Ищем максимальное и минимальное значение в каждом критерии

**for** (**int** j = **0**; j < Ratings[**0**].criteries.size(); j++)

**for** (**int** i = **1**; i < Ratings.size(); i++)

**if** (max\_values[j] < Ratings[i].criteries[j])

max\_values[j] = Ratings[i].criteries[j];

**else**

min\_values[j] = Ratings[i].criteries[j];

// A[i][j] = ( A[i][j] - A[j]min ) / ( A[j]max - A[j]min )

// Меняем вид матрицы нормировка матрицы (кроме столбца главного критерия)

// Для этого создаем новый вектор-матрица и выведем в консоль новую матрицу

vector<Rating> NormRatings(**4**);

**for** (**int** i = **0**; i < Ratings.size(); i++) {

NormRatings[i].road\_name = Ratings[i].road\_name;

**for** (**int** j = **0**; j < Ratings[i].criteries.size(); j++)

**if** (j != main\_criterion)

NormRatings[i].criteries[j] = (Ratings[i].criteries[j] - min\_values[j]) / (max\_values[j] - min\_values[j]);

**else**

NormRatings[i].criteries[j] = Ratings[i].criteries[j];

}

OutputTable(NormRatings);

vector<**bool**> BooleanList(**4**);

**for** (**int** i = **0**; i < Ratings.size(); i++) {

**bool** FalseOrTrue = true;

**for** (**int** j = **0**; j < Ratings[i].criteries.size(); j++)

**if** (j != main\_criterion)

FalseOrTrue = FalseOrTrue & (Ratings[i].criteries[j] >= max\_values[j] \* part[j]);

BooleanList[i] = FalseOrTrue;

}

// Если несколько дорог соответствует критериям, то на основе главного критерия выясняем, какая дорога станет наилучшим вариантом для передвижения

**double** maxV = min\_values[main\_criterion];

string maxRoadName = Ratings[**0**].road\_name;

**for** (**int** i = **0**; i < Ratings.size(); i++)

**if** (BooleanList[i])

**if** (Ratings[i].criteries[main\_criterion] > maxV) {

maxV = Ratings[i].criteries[main\_criterion];

maxRoadName = Ratings[i].road\_name;

}

**return** maxRoadName;

}

// 2. Метод Парето + Евклидово расстояние (Вариант 4) - Формула по Евклидову расстоянию взято из лабы 6

**double** **EvklidovoDistance**(**const** **double**& x\_max, **const** **double**& y\_max, **const** **double**& x\_from\_main\_critery, **const** **double**& y\_from\_second\_critery)

{

**return** pow(x\_max - x\_from\_main\_critery, **2**) + pow(y\_max - y\_from\_second\_critery, **2**);

}

Rating **ParetoMethod**(**const** vector<Rating>& Ratings, **const** pair<**int**, **int**>& main\_pair)

{

// Максимально возможные баллы = точка утопии

**double** x\_max = **10**;

**double** y\_max = **10**;

// Предположим, что первая дорога - "выбранная". Проверим через цикл

Rating best\_choice = Ratings[**0**];

**double** min\_dis = pow(x\_max - Ratings[**0**].criteries[main\_pair.first], **2**) + pow(y\_max - Ratings[**0**].criteries[main\_pair.second], **2**);

**double** cicle\_distance; // Вычисляем евклидово расстояние в течение цикла

cout << "2. Метод Парето" << endl;

**for** (**int** i = **0**; i < Ratings.size(); i++)

{

cout << "**\n** Оценка: (";

cout << Ratings[i].criteries[main\_pair.first] << "; ";

cout << Ratings[i].criteries[main\_pair.second] << "): " << Ratings[i].road\_name;

// Меняем или оставляем приоритет по дороге

cicle\_distance = EvklidovoDistance(x\_max, y\_max, Ratings[i].criteries[main\_pair.first], Ratings[i].criteries[main\_pair.second]);

**if** (cicle\_distance < min\_dis)

{

best\_choice = Ratings[i];

min\_dis = cicle\_distance;

}

}

**return** best\_choice;

}

// 3. Взвешивание и объединение критериев

string **WeightAndCombMethod**(vector<Rating> Ratings, **const** vector<**double**>& normalizeVector)

{

vector<**double**> sumColumn = Ratings[**0**].criteries;

// Суммируем каждый столбец, чтобы потом провести нормировку матрицы

**for** (**int** j = **0**; j < Ratings[**0**].criteries.size(); j++){

sumColumn[j] = **0**;

**for** (**int** i = **0**; i < Ratings.size(); i++)

sumColumn[j] += Ratings[i].criteries[j];

}

// Создаем новый вектор-матрицу, куда занесем нормализованную матрицу: каждый элемент поделим на сумму каждого столбца + выведем в консоль новую матрицу

// A[i][j = A[i][j] / sumColumn[j] - элемент исходной матрицы на сумму каждого столбца

vector<Rating> NormRatingsAfterSumColumn(**4**);

**for** (**int** i = **0**; i < Ratings.size(); i++) {

NormRatingsAfterSumColumn[i].road\_name = Ratings[i].road\_name;

**for** (**int** j = **0**; j < Ratings[i].criteries.size(); j++)

NormRatingsAfterSumColumn[i].criteries[j] = Ratings[i].criteries[j] / sumColumn[j];

}

OutputTable(NormRatingsAfterSumColumn); // Используемая функция

// Перемножаем 2 матрицы 4 х 4 на 4 х 1, получим вектор 4 х 1. По наибольшему значению определим наилучший выбор

vector<**double**> ResultVector(**4**);

cout << "**\n** Получившийся вектор объединенных критерий: ( ";

**for** (**int** i = **0**; i < NormRatingsAfterSumColumn.size(); i++) {

ResultVector[i] = **0**;

**for** (**int** j = **0**; j < NormRatingsAfterSumColumn[i].criteries.size(); j++)

ResultVector[i] += NormRatingsAfterSumColumn[i].criteries[j] \* normalizeVector[j];

cout << ResultVector[i] << " ";

}

cout << ")" << endl;

// Ищем максимальное значение по массиву - опеределяем индекс

**double** max\_vector\_value = ResultVector[**0**];

**int** index = **0**;

**for** (**int** i = **0**; i < ResultVector.size(); i++)

{

**if** (max\_vector\_value < ResultVector[i])

{

max\_vector\_value = ResultVector[i];

index = i;

}

}

// На основе известного индекса, под которым хранится максимальное значение, запишем в ответ название дороги

string resultMethod = "";

**switch** (index)

{

**case** **0**:

resultMethod = Ratings[**0**].road\_name;

**break**;

**case** **1**:

resultMethod = Ratings[**1**].road\_name;

**break**;

**case** **2**:

resultMethod = Ratings[**2**].road\_name;

**break**;

**case** **3**:

resultMethod = Ratings[**3**].road\_name;

**break**;

**default:**

**break**;

}

**return** resultMethod;

}

// 4. Метод анализа иерархии

vector<**double**> BuildNormVect(**const** vector<**double**>& vect)

{

// Строю матрицу.

vector<vector<**double**>> Matr = { {**0**, **0**, **0**, **0**}, {**0**, **0**, **0**, **0**}, {**0**, **0**, **0**, **0**}, {**0**, **0**, **0**, **0**} };

**for** (**int** i = **0**; i < vect.size(); i++)

**for** (**int** j = **0**; j < vect.size(); j++)

**if** (i == j)

Matr[i][j] = **1**;

**else**

**if** ((vect[i] - vect[j] == **1**) || (vect[i] - vect[j] == **2**))

Matr[i][j] = **3**;

**else**

**if** ((vect[i] - vect[j] == **3**) || (vect[i] - vect[j] == **4**))

Matr[i][j] = **5**;

**else**

**if** ((vect[i] - vect[j] == **5**) || (vect[i] - vect[j] == **6**))

Matr[i][j] = **7**;

**else**

**if** ((vect[i] - vect[j] == **7**) || (vect[i] - vect[j] == **8**))

Matr[i][j] = **9**;

**else**

**if** ((vect[i] - vect[j] == -**1**) || (vect[i] - vect[j] == -**2**))

Matr[i][j] = **0.333**;

**else**

**if** ((vect[i] - vect[j] == -**3**) || (vect[i] - vect[j] == -**4**))

Matr[i][j] = **0.2**;

**else**

**if** ((vect[i] - vect[j] == -**5**) || (vect[i] - vect[j] == -**6**))

Matr[i][j] = **0.142**;

**else**

**if** ((vect[i] - vect[j] == -**7**) || (vect[i] - vect[j] == -**8**))

Matr[i][j] = **0.111**;

// Получу в векторе сумму по строкам.

vector<**double**> vectRes(vect.size());

**for** (**int** i = **0**; i < vect.size(); i++) {

**double** s = **0.0**;

**for** (**int** j = **0**; j < vect.size(); j++) {

s += Matr[i][j];

cout << Matr[i][j] << " ";

}

vectRes[i] = s;

cout << " " << endl;

}

// Нормализую вектор.

vector<**double**> vectResult = vectRes;

**double** sum = accumulate(vectResult.begin(), vectResult.end(), **0.0**);

**for** (**int** i = **0**; i < vectResult.size(); i++)

{

vectResult[i] /= sum;

}

**return** vectResult;

}

string Compare(**const** vector<Rating>& Ratings, **const** vector<**double**>& PriorityVect)

{

vector<vector<**double**>> MatrNormVect(Ratings[**0**].criteries.size());

vector<**double**> Vect(Ratings.size());

// Цикл по альтернативам.

**for** (**int** k = **0**; k < Ratings[**0**].criteries.size(); k++) {

// Строю вектор по k-му критерию.

**if** (k == **0**) cout << "Матрица по критерию 'Дистанция'" << endl;

**else** **if** (k == **1**) cout << "Матрица по критерию 'Покрытие'" << endl;

**else** **if** (k == **2**) cout << "Матрица по критерию 'Контроль'" << endl;

**else** **if** (k == **3**) cout << "Матрица по критерию 'Инфраструктура'" << endl;

**for** (**int** i = **0**; i < Ratings.size(); i++)

Vect[i] = Ratings[i].criteries[k];

// Строю матрицу и нормализованный вектор по k-му критерию.

MatrNormVect[k] = BuildNormVect(Vect);

}

// Вывод нормализованной матрицы

cout << "Вывод нормализованной матрицы" << endl;

**for** (**int** k = **0**; k < MatrNormVect.size(); k++) {

**for** (**int** j = **0**; j < MatrNormVect.size(); j++)

cout << MatrNormVect[k][j] << " ";

cout << endl;

}

// Оценка приоритетов критериев + вывод нормализованного вектора

cout << "Матрица оценки приоритетов критериев" << endl;

Vect = BuildNormVect(PriorityVect);

**for** (**int** i = **0**; i < Vect.size(); i++)

cout << Vect[i] << " ";

cout << endl << endl;

cout << "Полученный результат - вектор" << endl << endl;

vector<**double**> ResultVector(**4**);

// Умножение нормализованной матрицы на нормализованный вектор.

**for** (**int** i = **0**; i < Ratings.size(); i++) {

ResultVector[i] = **0**;

**for** (**int** j = **0**; j < Ratings[i].criteries.size(); j++)

ResultVector[i] += MatrNormVect[i][j] \* Vect[j];

cout << ResultVector[i] << " ";

}

// Поиск максимального элемента вектора.

**double** max = **0.0**;

**int** index = **0**;

**for** (**int** i = **0**; i < ResultVector.size(); i++)

**if** (ResultVector[i] > max) {

max = ResultVector[i];

index = i;

}

string resultMethod = Ratings[index].road\_name;

**return** resultMethod;

}