МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по учебной практике

ТЕМА: ГЕНЕТИЧЕСКИЕ АЛГОРИТМЫ И PSA

Студенты Кроткина З.Э.,
Ларукова А.А.,
Романова К..

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2024

Цель работы.

Изучение работы генетических алгоритмов на примере решения задачи о назначениях.

Задание.

Задача о назначениях

Пусть имеется N работ и N кандидатов на выполнение этих работ, причем назначение j-й работы i-му кандидату требует затрат $Cij > 0$.

Необходимо назначить каждому кандидату по работе, чтобы минимизировать суммарные затраты. Причем каждый кандидат может быть назначен на одну работу, а каждая работа может выполняться только одним кандидатом.

Выполнение работы.

Графика

Класс MainWindow

Этот класс является основным окном приложения и наследует от QMainWindow.

Публичные методы и поля:

MainWindow(QWidget *parent = nullptr): Конструктор класса, инициализирующий главное окно приложения.

~MainWindow(): Деструктор класса, освобождающий ресурсы.

bool choice: Логическая переменная для выбора состояния.

Приватные методы и поля:

*Ui::MainWindow ui: Указатель на интерфейс пользователя.

QPalette pl: Объект для управления цветовой палитрой приложения.

*QWidget stripe: Виджет для отображения полоски сверху окна.

*QLabel textTitle: Метка для отображения заголовка на полоске.

*QWidget menu: Виджет для бокового меню.

*QLabel textMenu: Метка для текста в боковом меню.

*QPushButton matrixButton: Кнопка для выбора матрицы.

*QPushButton fileButton: Кнопка для выбора файла.

*QPushButton randomButton: Кнопка для случайной генерации данных.

*QPushButton DaleeButtom: Кнопка "Далее" для подтверждения ввода данных.

QString fileName: Переменная для хранения имени файла.

*QLineEdit line: Поле ввода для имени файла.

QString valMatrix: Переменная для хранения значений матрицы.

int sizeMatrix: Переменная для хранения размера матрицы.

*QSpinBox countSpinBox: Поле для ввода количества работников.

*QLabel countLabel: Метка для поля ввода количества работников.

*QTableWidget matrixTable: Таблица для ввода данных матрицы.

bool allFilled: Переменная для проверки заполненности всех ячеек матрицы.

Описание методов:

SideMenu(): Метод для создания бокового меню.

PushButtonMenu(int x, int y, const QString &text): Метод для создания кнопки в боковом меню.

DataEntryButton(): Метод для создания кнопки "Далее".

Matrix(): Метод для инициализации ввода матрицы.

createMatrix(): Метод для создания и отображения таблицы матрицы.

checkAllCellsFilled(): Метод для проверки заполненности всех ячеек матрицы.

handleFilledMatrix(): Метод для обработки данных после заполнения матрицы.

File(): Метод для инициализации ввода данных из файла.

ReadLine(): Метод для чтения имени файла.

Random(): Метод для случайной генерации данных.

getSize(): Метод для получения размера матрицы.

clearMenuExceptButtons(): Метод для очистки меню от виджетов, кроме кнопок "Матрица", "Из файла" и "Случайная генерация".

Файл mainwindow.cpp

Этот файл содержит реализацию методов класса MainWindow, определенных в mainwindow.h.

Конструктор и деструктор

MainWindow(QWidget *parent): Конструктор инициализирует главное окно, устанавливает размеры и перемещает его на экран. Также он создает и настраивает виджеты полоски сверху и бокового меню.

~MainWindow(): Деструктор освобождает ресурсы, используемые главным окном.

Методы:

void SideMenu(): Создает боковое меню и настраивает его внешний вид.

QPushButton* PushButtonMenu(int x, int y, const QString &text): Создает и возвращает кнопку с заданными координатами и текстом.

QPushButton* DataEntryButton(): Создает и возвращает кнопку "Далее", если состояние выбора (choice) истинно.

void Matrix(): Инициализирует процесс ввода матрицы, создает соответствующие виджеты.

void createMatrix(): Создает таблицу матрицы на основе введенного пользователем значения.

void checkAllCellsFilled(): Проверяет, заполнены ли все ячейки таблицы матрицы, и вызывает метод обработки заполненной матрицы.

void handleFilledMatrix(): Обрабатывает заполненные данные матрицы.

void File(): Инициализирует процесс ввода данных из файла, создает соответствующие виджеты.

void ReadLine(): Читает и сохраняет введенное пользователем имя файла.

void Random(): Инициализирует процесс случайной генерации данных, создает соответствующие виджеты.

void getSize(): Получает и сохраняет размер матрицы.

void clearMenuExceptButtons(): Очищает меню от всех виджетов, кроме кнопок "Матрица", "Из файла" и "Случайная генерация".

void clear(QWidget parent):*Скрывает все дочерние виджеты указанного родительского виджета, кроме заранее определенных виджетов (setting, graph, view, matrixButton, fileButton, randomButton, textMenu).

void SolutionMenu(): Создает два новых виджета (other и solution) и настраивает их размеры и позиции. Включает в себя создание дополнительных виджетов для отображения настроек и графиков.

void SettingMenu(): Настраивает виджеты, относящиеся к настройкам. Включает создание и настройку меток, полей ввода и кнопок. Устанавливает стили и шрифты для элементов интерфейса. Кнопка "Далее" связывается с методом ReadVer.
QPushButton NextButton():* Создает и настраивает кнопку ">>" в виджете настроек. Устанавливает шрифт, цвет и размер кнопки, делает ее видимой.

void ReadVer(): Считывает данные из поля ввода (verLine). Если поле не пустое, скрывает кнопку "Далее" и вызывает метод Graph для отображения графика.

void Graph():Отображает графическое представление решений. Создает метки, виджет графического представления (QGraphicsView) и сцену (QGraphicsScene). Добавляет линии и вершины, представляющие граф, и кнопки для дальнейших действий.

void Plot():Загружает изображение из указанного пути и отображает его в интерфейсе. Создает метку (QLabel) и загружает изображение, масштабируя его до нужного размера.

QPushButton PushButtonSolution(QWidget parent, const QString &text, int x, int y, int width, int height): Создает и настраивает кнопки с заданными параметрами (текст, позиция, размер) и добавляет их в родительский виджет. Устанавливает шрифт, цвет и делает кнопку видимой.

int getSizeMatrix(): Возвращает размер матрицы (sizeMatrix), если он больше нуля. В противном случае возвращает ноль.

QString getValMatrix(): Метод, который должен возвращать значения матрицы. В текущей реализации возвращает ноль.

void closeMatrix():Закрывает таблицу матрицы (matrixTable). Используется для завершения работы с таблицей и освобождения ресурсов.

Файл main.cpp

Этот файл содержит функцию main, которая запускает приложение.

Функция main

int main(int argc, char *argv[]): Создает экземпляр приложения QApplication и экземпляр главного окна MainWindow, затем отображает главное окно и запускает основной цикл обработки событий.

Описание функций алгоритма

Класс Chromosome: std::vector<int> _workers - вектор, хранящий назначенную рабочему задачу, int _cost - затраты, int _size - размер.

Класс матрицы стоимостей CostMatrix: std::vector<std::vector<int>> _costArray - матрица стоимостей, int _n - размер.

Класс популяции Population: std::vector<Chromosome> _chromosomes - вектор, хранящий объекты типа Chromosome, long _bestChromosomeCost - затраты на лучшую хромосому, int _bestChromosomeIndex - индекс лучшей хромосомы, Chromosome _bestChromosome - лучшая хромосома, bool _maximise - переменная для минимизации.

void Print(int iteration) - функция вывода в терминал информации.

Chromosome Crossover(Chromosome &chr, std::mt19937 &rnd) - функция скрещивания

void Mutation(std::mt19937 &rnd) - функция мутации

void Copy(Chromosome &chr) - копирование обьекта

void Assign(int worker, int task) - функция назначает каждому рабочему задачу

void SetCost(int agent, int task, int cost) - функция присваивает стоимости

int GetChromosomeCost(Chromosome &chromosome, bool maximise) - функция получения стоимости в хромосоме

void CreateArbitraryPopulation(std::mt19937 &rnd, int populationSize, int taskSize) - создаем произвольную популяцию

void Evaluate(CostMatrix &costMatrix, int iteration) - функция оценки

void ApplyCrossover(std::mt19937 &rnd, int taskSize) - функция применения скрещивания

void Crossover(int parentIndex1, int parentIndex2, std::mt19937 &rnd, int taskSize) - ф-я скрещивания, в кач-ве аргументов принимает 2 родителей, случайное число и размер задачи

void Mutate(std::mt19937 &rnd) - функция мутации, в кач-ве аргумента принимает случайное число

bool IsBetter(long cost1, long cost2) - ф-я для определения лучшей стоимости

void Selection(std::mt19937 &rnd) - функция отбора

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД

Файл mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H
#include <iostream>
#include <QtWidgets>
#include <vector>
#include "alg.cpp"

QT_BEGIN_NAMESPACE
namespace Ui {
class MainWindow;
}
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();
    bool choice = false ;



private:
    //общее
    Ui::MainWindow *ui;
    QPalette pl;
    QFont font = QFont("Cascadia Code");
    QWidget *stripe;
    QLabel *textTitle;
    QWidget *menu;
    QWidget *setting;
    QWidget *solution;
    QWidget *graph;
    QLabel *textMenu;
```

```
QLabel *textSetting;
QPushButton *matrixButton;
QPushButton *fileButton;
QPushButton *randomButton;
QPushButton *DaleeButtom;
QPushButton *nextButton;
QLabel *textError;

//ответ
std::vector<int> best {};
std::vector<int> good1 {};
std::vector<int> good2 {};

//граф
QGraphicsView *view;
QGraphicsScene *scene;

//файл
QString fileName = nullptr;
QLineEdit*line = nullptr;

//матрица
int sizeMatrix;
CostMatrix matrix=CostMatrix(getSizeMatrix());
QSpinBox* countSpinBox;
QLabel *countLabel;
QTableWidget *matrixTable;
bool allFilled = false;

//общее
void SideMenu();
QPushButton* PushButtonMenu( int x, int y, const QString &text);
QPushButton* DataEntryButton();



void Matrix();
void closeMatrix();
void createMatrix();
```

```
        void checkAllCellsFilled();
        void handleFilledMatrix();


        void File();
        void ReadLine();

        void Random();
        void getSizeRand();

        void clearSideMenu();
        void clear(QWidget* parent);

        //решение
        void SettingMenu();
        void SolutionMenu();
        void Graph();
        void Solution();


        QPushButton*  PushButtonSolution(QWidget*  parent,  const  QString
&text, int x, int y, int width, int height);
        int iteration = 0;
        QWidget *other;
        //График
        void Plot();

        //считывание настроек
        QString verData = nullptr;
        QLineEdit* verLine = nullptr;
        QPushButton* NextButton();
        void ReadVer();



        //геттеры сеттеры
        int getSizeMatrix();
        QString* getValMatrix();
```

```
    QWidget *plashca;


};
#endif // MAINWINDOW_H
```

`Файл mainwindow.cpp

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include <iostream>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)

{
    ui->setupUi(this);
    setFixedSize(1400, 750);
    move(100, 20);
    pl.setColor(QPalette::Window, QColor(244,244,244));
    setPalette(pl);

    //полоска сверху
    stripe = new QWidget(this);
    stripe->setGeometry(0,0,1400,150);
    pl.setColor(QPalette::Window, QColor(44,103,115));
    stripe->setPalette(pl);
    stripe->setAutoFillBackground(true);
    stripe->show();

    //текст на полоске
    textTitle = new QLabel("Назначения", stripe);
    font.setPointSize(74);
    textTitle->setFont(font);
    textTitle->setStyleSheet("color: rgb(244, 244, 244);");
```

```
        textTitle->setAlignment(Qt::AlignVCenter);
        QHBoxLayout *stripeLayout = new QHBoxLayout(stripe);
        stripeLayout->addWidget(textTitle);
        stripeLayout->setAlignment(Qt::AlignCenter);


        //боковое меню
        SideMenu();
        //view = new QGraphicsView(graph);
        //scene = new QGraphicsScene(graph);
        //решение



        matrixButton = PushButtonMenu( 35, 80, "Матрица");
        fileButton = PushButtonMenu( 35, 160, "Из файла");
        randomButton = PushButtonMenu( 35, 240, "Случайная \nгенерация");

        SolutionMenu();



        connect(matrixButton, &QPushButton::clicked, this,
&MainWindow::Matrix);
        connect(fileButton, &QPushButton::clicked, this,
&MainWindow::File);
        connect(randomButton, &QPushButton::clicked, this,
&MainWindow::Random);


        plashca = new QWidget(solution);
        plashca->setGeometry(450, 50, 650,550);
        plashca->setStyleSheet("background-color: rgb(244, 244, 244)");



    }
```

```
MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::SideMenu(){
    menu = new QWidget(this);
    pl.setColor(QPalette::Window, QColor(44,154,176));
    menu->setPalette(pl);
    menu->setAutoFillBackground(true);
    menu->setGeometry(0, 150, 300,600);


    textMenu = new QLabel("Способ ввода \nданных:", menu);
    font.setPointSize(21);
    textMenu->setFont(font);
    textMenu->setStyleSheet("color: rgb(244, 244, 244);");
    textMenu->setAlignment(Qt::AlignLeft | Qt::AlignTop);
    QHBoxLayout *stripeLayout = new QHBoxLayout(menu);
    stripeLayout->addWidget(textMenu);
    stripeLayout->setAlignment(textMenu, Qt::AlignLeft |
Qt::AlignTop);


}


QPushButton* MainWindow::PushButtonMenu( int x, int y, const QString
&text)
{
    QPushButton *button = new QPushButton(text, menu);
    font.setPointSize(20);
    button->setFont(font);

    button->setStyleSheet("color: rgb(244, 244, 244);");
    pl.setColor(QPalette::Button, QColor(44,154,176));
    button->setPalette(pl);
    button->setGeometry(x, y, 200, 70);
    return button;
```

```
    }


    void MainWindow::Matrix(){



        clearSideMenu();
        clear(setting);
        clear(other);
        countLabel = new QLabel("Количество \nработников:", menu);
        countSpinBox = new QSpinBox(menu);
        countSpinBox->setMinimum(2);
        countSpinBox->setMaximum(10);

        font.setPointSize(15);
        countLabel->setFont(font);
        countLabel->setStyleSheet("color: rgb(244, 244, 244);");
        countLabel->setGeometry(35, 350, 150, 80);
        countLabel->show();
        countSpinBox->setGeometry(200, 370, 50,40);
        countSpinBox->show();


        choice = true;
        DaleeButtom = this->DataEntryButton();
        connect(DaleeButtom, &QPushButton::clicked, this,
&MainWindow::createMatrix);

    }

    void MainWindow::createMatrix()
    {
        clearSideMenu();

        if (countSpinBox->value() < 7){
            matrixTable = new QTableWidget(countSpinBox->value(),
countSpinBox->value(), menu);
```

```
            matrixTable->setGeometry(10, 350, 280, 200);
            for (int i = 0; i < countSpinBox->value(); ++i) {
                matrixTable->setColumnWidth(i,
260/countSpinBox->value());
                matrixTable->setRowHeight(i, 175/countSpinBox->value());
            }
        }
        else {
            matrixTable = new QTableWidget(countSpinBox->value(),
countSpinBox->value());
            matrixTable->setGeometry(550,
300,countSpinBox->value()*40+30, countSpinBox->value()*30+35);
            for (int i = 0; i < countSpinBox->value(); ++i) {
                matrixTable->setColumnWidth(i, 40);
                matrixTable->setRowHeight(i, 30);
            }
        }
        //matrixTable->setStyleSheet("color: rgb(44,103,115);"
        //                           "background-color: rgb(244, 244,
244);");
        matrixTable->show();
        DaleeButtom = this->DataEntryButton();
        connect(DaleeButtom, &QPushButton::clicked, this,
&MainWindow::closeMatrix);
        connect(DaleeButtom, &QPushButton::clicked, this,
&MainWindow::checkAllCellsFilled);




    }


    void MainWindow::checkAllCellsFilled()
    {

        bool allFilled = true;

        for (int i = 0; i < matrixTable->rowCount(); ++i) {
```

```
            for (int j = 0; j < matrixTable->columnCount(); ++j) {
                QTableWidgetItem *item = matrixTable->item(i, j);
                if (!item ||item->text().isEmpty()) {
                    allFilled = false;
                    break;
                }
            }
            if (!allFilled) {
                QLabel *textError = new QLabel("Ошибка в
заполнении\nВыберите способ заново",menu);
                textError->setStyleSheet("color: red;");
                font.setPointSize(12);
                textError->setFont(font);
                textError->setGeometry(10,305, 280, 50);
                textError->show();
                break;
            }
        }

        if (allFilled){
            handleFilledMatrix();
        }



    }

    void MainWindow::handleFilledMatrix()
    {

        sizeMatrix = matrixTable->rowCount();
        for (int i = 0; i < matrixTable->rowCount(); ++i) {
            for (int j = 0; j < matrixTable->rowCount(); ++j) {
                QTableWidgetItem *item = matrixTable->item(i, j);
                if (item) {
                    matrix.SetCost(i,j,item->text().toInt());
                }
            }
        }
        textError = new QLabel("Матрица заполнена",menu);
```

```
        textError->setStyleSheet("color: rgb(244, 244, 244);");
        font.setPointSize(12);
        textError->setFont(font);
        textError->setGeometry(10,305, 280, 50);
        textError->show();

        SettingMenu();


    }

    void MainWindow::File(){

        clearSideMenu();
        clear(setting);
        clear(other);
        QLabel *text = new QLabel("Название:", menu);
        font.setPointSize(15);
        text->setStyleSheet("color: rgb(244, 244, 244);");
        text->setFont(font);
        text->move(35, 370);
        text->show();

        line = new QLineEdit(menu);
        line->setGeometry(35, 400, 200, 30);
        line->show();
        this->choice = true;
        DaleeButtom = this->DataEntryButton();
        connect(DaleeButtom, &QPushButton::clicked, this,
&MainWindow::ReadLine);



    }
    void MainWindow::ReadLine(){
        fileName = line->text();
        if (fileName != nullptr){
        textError = new QLabel("Файл считан",menu);
        textError->setStyleSheet("color: rgb(244, 244, 244);");
```

```
        font.setPointSize(12);
        textError->setFont(font);
        textError->setGeometry(10,305, 280, 50);
        textError->show();
        SettingMenu();
        }
    }


    void MainWindow::Random(){
        clearSideMenu();
        clear(setting);
        clear(other);
        countLabel = new QLabel("Количество \nработников:", menu);
        countSpinBox = new QSpinBox(menu);
        countSpinBox->setMinimum(2);
        countSpinBox->setMaximum(10);

        font.setPointSize(15);
        countLabel->setFont(font);
        countLabel->setStyleSheet("color: rgb(244, 244, 244);");
        countLabel->setGeometry(35, 350, 150, 80);
        countLabel->show();
        countSpinBox->setGeometry(200, 370, 50,40);
        countSpinBox->show();


        choice = true;
        DaleeButtom = this->DataEntryButton();
        connect(DaleeButtom, &QPushButton::clicked, this,
&MainWindow::getSizeRand);
    }

    void MainWindow::getSizeRand(){
        sizeMatrix = countSpinBox->value();
        textError = new QLabel("Матрица сгенерирована",menu);
        textError->setStyleSheet("color: rgb(244, 244, 244);");
        font.setPointSize(12);
        textError->setFont(font);
```

```
        textError->setGeometry(35,305, 280, 50);
        textError->show();
        SettingMenu();

    }

    QPushButton* MainWindow::DataEntryButton(){
        if (choice){
            QPushButton *button = new QPushButton("Далее", menu);
            font.setPointSize(17);
            button->setFont(font);
            button->setStyleSheet("color: rgb(244, 244, 244);");
            button->setGeometry(200, 550, 90, 40);
            pl.setColor(QPalette::Button, QColor(44,103,115));
            button->setPalette(pl);
            button->show();
            return button;

        }
        return nullptr;
    }

    void MainWindow::clearSideMenu()
    {

        QList<QWidget*> children = menu->findChildren<QWidget*>();

        for (QWidget *child : children) {

            if (child != matrixButton && child != fileButton && child !=
randomButton && child != textMenu
                ) {
                child->hide();
                child->deleteLater();
            }
        }

        fileName = nullptr;
      // matrix = nullptr;
```

```
        sizeMatrix = 0 ;
        allFilled = false;
        verData = nullptr;
    }

    void MainWindow::clear(QWidget* parent)
    {

        QList<QWidget*> children = parent->findChildren<QWidget*>();

        for (QWidget *child : children) {
            if (child != setting && child != graph && child != view
&&child != matrixButton && child != fileButton && child != randomButton &&
child != textMenu ) {
                child->hide();
            }
        }
        setting->setStyleSheet("background-color: rgb(244, 244, 244)");
        plashca->show();

    }


    void MainWindow::SolutionMenu(){
        other = new QWidget(this);
        other->setGeometry(300, 150, 1150,600);
        solution = new QWidget(this);
        solution->setGeometry(300, 150, 1150,600);
        setting = new QWidget(solution);
        setting->setGeometry(0, 0, 300,150);

        graph = new QWidget(solution);
        graph->setGeometry(450, 50, 650,500);



    }

    void MainWindow::SettingMenu(){
```

```
        DaleeButtom->setEnabled(false);
        textSetting = new QLabel("Вероятность мутации:", setting);
        QLabel * percent = new QLabel("%", setting);
        verLine = new QLineEdit(setting);

        setting->setStyleSheet("background-color: rgb(140,178,188);");


        font.setPointSize(14);
        textSetting->setFont(font);
        textSetting->setStyleSheet("color: rgb(24, 24, 24);");
        textSetting->setGeometry(0,10, 220,25);
        textSetting->show();

        percent->setFont(font);
        percent->setStyleSheet("color: rgb(24, 24, 24);");
        percent->setGeometry(280,15, 25,25);
        percent->show();


        verLine->setGeometry(225, 10, 50, 30);
        verLine->setStyleSheet("color: rgb(224, 224, 224);"
                               "background-color: rgb(45, 45, 45)");


        verLine->show();
        nextButton = NextButton();
        connect(nextButton, &QPushButton::clicked, this,
&MainWindow::ReadVer);


    }

    //setiingMenu
    QPushButton* MainWindow::NextButton(){

        QPushButton *button = new QPushButton(">>", setting);
        font.setPointSize(20);
        button->setFont(font);
```

```
        button->setStyleSheet("color: rgb(24, 24, 24);"
                              "background-color: rgb(140,178,188);");
        button->setGeometry(250,110, 50,40);
        button->show();
        return button;


    }

    void MainWindow::ReadVer(){
        verData = verLine->text();
        if (!verData.isEmpty()) {
            //clear(other);
            nextButton->hide();
            Graph();


        }

    }

    //graph



    void MainWindow::Graph(){

        QLabel *answer = new QLabel("Решение:",other);
        font.setPointSize(20);
        answer->setFont(font);
        answer->setGeometry(500,25, 280, 25);
        answer->setStyleSheet("color: rgb(24, 24, 24);");
        answer->show();

        QLabel *iter = new QLabel("Итерация:
"+QString::number(iteration),other);
        //setting->setStyleSheet("background-color: rgb(140,178,188);");
        font.setPointSize(20);
        iter->setFont(font);
```

```
iter->setGeometry(20,250, 200, 25);
iter->setStyleSheet("color: rgb(24, 24, 24);");
iter->show();

iteration++;
int numVertices = best.size();
plashca->hide();


view = new QGraphicsView(graph);
scene = new QGraphicsScene(graph);
view->setGeometry(450, 50, 650, 550);
view->setScene(scene);


QVBoxLayout *layout = new QVBoxLayout(graph);
layout->addWidget(view);
setLayout(layout);


const int radius = 30;
const int spacing = 550/numVertices;


for (int i = 0; i < numVertices; ++i) {

    int x1 = 50 + radius / 2;
    int y1 = 50 + i * spacing + radius / 2;

    int x2 = 250 + radius / 2;
    int y2 = 50 + (best[i]-1) * spacing + radius / 2;
    scene->addLine(x1, y1, x2, y2, QPen(QColor(33, 75, 86), 5));

}
for (int i = 0; i < numVertices; ++i) {

    int x1 = 50 + radius / 2;
    int y1 = 50 + i * spacing + radius / 2;
```

```
            int x2 = 250 + radius / 2;
            int y2 = 50 + (good1[i]-1) * spacing + radius / 2;
            scene->addLine(x1, y1, x2, y2, QPen(QColor(82, 122, 132), 5,
Qt::DashLine));

        }
        for (int i = 0; i < numVertices; ++i) {

            int x1 = 50 + radius / 2;
            int y1 = 50 + i * spacing + radius / 2;

            int x2 = 250 + radius / 2;
            int y2 = 50 + (good2[i]-1) * spacing + radius / 2;
            scene->addLine(x1, y1, x2, y2, QPen(QColor(140, 178, 188), 5,
Qt::DotLine));

        }

        // Создаем вершины множества U
        for (int i = 0; i < numVertices; ++i) {
            int x = 50;
            int y = 50 + i * spacing;

            QGraphicsEllipseItem *vertex = scene->addEllipse(x, y,
radius, radius, QPen(QColor(34, 88, 101), 2), QBrush(QColor(34, 88, 101),
Qt::SolidPattern));
            font.setPointSize(17);
            QGraphicsTextItem *name = scene->addText(QString::number(i +
1), font);

            name->setDefaultTextColor(QColor(224, 224, 224));
            name->setPos(x + radius / 8, y - radius / 8);
        }

        // Создаем вершины множества V
        for (int i = 0; i < numVertices; ++i) {
            int x = 250;
            int y = 50 + i * spacing;
```

```
            QGraphicsEllipseItem *vertex = scene->addEllipse(x, y,
radius, radius, QPen(QColor(34, 88, 101), 2), QBrush(QColor(34, 88, 101),
Qt::SolidPattern));
            QGraphicsTextItem *name = scene->addText(QString::number(i +
1), font);
            name->setDefaultTextColor(QColor(224, 224, 224));
            font.setPointSize(17);
            name->setPos(x + radius / 8, y - radius / 8);
        }
        Plot();

        QPushButton *finishButton = PushButtonSolution(other, "Перейти в
конец", 10, 545, 250, 45);
        QPushButton *continueButton = PushButtonSolution(other, "Далее",
1000, 545, 90, 30);


    }

    void MainWindow::Plot(){

        QLabel *imageLabel = new QLabel(other);
        imageLabel->setGeometry(20, 290, 500, 250);
        QPixmap pixmap;
        pixmap.load("D:/Genetic/untitled1/images/img.png");
        pixmap = pixmap.scaled(imageLabel->size(), Qt::KeepAspectRatio,
Qt::SmoothTransformation);
        imageLabel->setPixmap(pixmap);
        imageLabel->show();
    }

    QPushButton* MainWindow::PushButtonSolution(QWidget* parent, const
QString &text, int x, int y, int width, int height){

        QPushButton *button = new QPushButton(text, parent);
        font.setPointSize(17);
        button->setFont(font);
        button->setStyleSheet("color: rgb(24, 24, 24);");
        pl.setColor(QPalette::Button, QColor(244,244,244));
```

```
    button->setPalette(pl);
    button->setGeometry(x, y, width, height);
    button->show();
    return button;
}


int MainWindow::getSizeMatrix(){
    if (sizeMatrix >0 ){
        return sizeMatrix;
    }
    return 0;

}
QString* MainWindow::getValMatrix(){
  //  if (matrix.GetCost(i,j)!=nullptr ){
       // return matrix;
    //}
    return 0;
}

void MainWindow::closeMatrix(){
    matrixTable->close();
}

void MainWindow::Solution(){
    std::mt19937 rnd(std::random_device{}());
    int popSize=100;
    bool maximise=false;
    Population _population(rnd,popSize,sizeMatrix,maximise);
    int iteration=1;
    _population.Evaluate(matrix,iteration,best,good1,good2);

    while (iteration<100){
        _population.StoreBestSolution(sizeMatrix);
        _population.Mutate(rnd,verData.toInt());
        _population.ApplyCrossover(rnd,sizeMatrix);
        _population.SeedBestSolution(rnd);
        _population.Evaluate(matrix,iteration,best,good1,good2);
```

```
        _population.Selection(rnd);
        iteration++;
    }
}
```

Файл main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```