

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической кибернетики и компьютерных наук

**ГЕНЕТИЧЕСКИЙ АЛГОРИТМ ДЛЯ ПОИСКА ЦЕНТРАЛЬНЫХ
ВЕРШИН В ГРАФАХ**

БАКАЛАВРСКАЯ РАБОТА

студента 4 курса 411 группы
направления 02.03.02 — Фундаментальная информатика и информационные
технологии
факультета КНиИТ
Власова Андрея Александровича

Научный руководитель
доцент, к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н., доцент _____ А. С. Иванов

СОДЕРЖАНИЕ

| | |
|---|-----------|
| ВВЕДЕНИЕ | 4 |
| 1 Описание задачи и алгоритмы для ее решения | 5 |
| 1.1 Тривиальный алгоритм | 5 |
| 1.2 Алгоритмы с улучшенной асимптотикой | 6 |
| 1.3 Алгоритмы, использующие матричное умножение | 6 |
| 1.4 Генетические алгоритмы | 6 |
| 1.4.1 Представление решения и начальная популяция | 8 |
| 1.4.2 Оператор скрещивания | 8 |
| 1.4.3 Оператор мутации | 8 |
| 1.4.4 Оператор естественного отбора | 8 |
| 2 Разработка генетического алгоритма и его исследование | 11 |
| 2.1 Старт алгоритма | 12 |
| 2.2 Естественный отбор | 12 |
| 2.3 Этап скрещивания | 13 |
| 2.4 Этап мутации | 13 |
| 2.5 Исследование алгоритма | 15 |
| 2.6 Модели случайных графов | 19 |
| 2.6.1 Модель случайного графа Эрдеша-Рены | 19 |
| 2.6.2 Модель случайного графа Барабаши-Альберт | 20 |
| 2.6.3 Геометрический случайный граф | 21 |
| 2.7 Результаты вычислительных экспериментов | 21 |
| 2.8 Исследование параметров генетического алгоритма | 23 |
| 3 Описание приложения, созданного для работы с генетическим алгоритмом | 33 |
| 3.1 Описание технологий и архитектуры приложения | 36 |
| 3.2 Структура базы данных | 37 |
| 3.3 Уровень доступа к данным | 38 |
| 3.4 Уровень бизнес-логики | 39 |
| 3.5 Уровень представления | 40 |
| 3.5.1 Контроллеры | 41 |
| 3.5.2 Модели данных и валидация | 42 |
| 3.5.3 Аутентификация | 44 |
| 3.6 Хеширование паролей | 44 |

| | | |
|--|--|----|
| 3.7 | Внедрение зависимостей | 45 |
| ЗАКЛЮЧЕНИЕ | | 47 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ | | 48 |
| Приложение А | Код генетического алгоритма..... | 51 |
| Приложение Б | Код классов для работы с графами | 56 |
| Приложение В | Код классов для экспериментов с алгоритмом | 65 |
| Приложение Г | Уровень доступа данных | 70 |
| Приложение Д | Уровень бизнес-логики..... | 76 |
| Приложение Е | Уровень пользовательского интерфейса | 83 |
| Приложение Ж | CD-диск с отчетом о выполненной работе..... | 91 |

ВВЕДЕНИЕ

В современных компьютерных науках одно из центральных мест занимает математическая модель графа. Графовая модель позволяет описывать целый ряд систем и явлений, которые встречаются в различных предметных областях. Изучение структуры того или иного графа, или же выявление его свойств и особенностей может приносить невероятную практическую пользу. В связи с этим работы, посвященные этой теме, вызывают наибольший интерес как со стороны исследователей теоретиков, так и со стороны практиков.

Одной из важных характеристик любого графа можно назвать радиус графа и расположение его центральных вершин. Получив данные параметры, можно судить об общей структуре графа или же о его особенностях. Современные графовые модели могут насчитывать десятки сотен тысяч вершин, поэтому для эффективного решения задач зачастую бывает недостаточно использовать классические алгоритмы, а необходимо реализовать некоторый эвристический подход, одним из которых является генетический алгоритм.

Основной целью работы было следующее — разработать и исследовать генетический алгоритм, способный решать задачу поиска центральных вершин. Для достижения этой цели были поставлены следующие задачи:

- реализовать идеи генетических алгоритмов с учетом рассматриваемой задачи,
- реализовать существующие алгоритмы для поиска центральных вершин,
- создать программное приложение, позволяющее проводить запуски алгоритма с различными параметрами,
- исследовать параметры алгоритма и выявить его слабые и сильные стороны.

1 Описание задачи и алгоритмы для ее решения

Для описания задачи сначала необходимо дать формальное определение понятию граф. Граф — это упорядоченная пара множеств (V, E) , где V — множество вершин графа, а E — множество упорядоченных и неупорядоченных пар вершин — дуг или ребер. В случае, когда вершины в парах упорядочены, говорят, что граф является ориентированным, иначе — неориентированным.

В случае неориентированного графа также используется понятие связности графа — граф является связным, если между любой парой вершин существует по крайней мере один путь. Кроме этого графы можно разделить на взвешенные или невзвешенные — в случае взвешенного графа каждое ребро имеет некоторый вес — положительное или отрицательное число, в случае невзвешенного графа каждое ребро имеет вес равный единице.

В данной работе рассматривается задача поиска центральных вершин в графах. Для начала можно дать определение эксцентриситета вершины. Эксцентриситетом вершины называется максимальное из расстояний от этой вершины до всех остальных вершин в графе. С использованием этого определения можно сказать, что центральными вершинами в графе называются вершины с минимальным значением эксцентриситета, при чем само значение этого минимального эксцентриситета представляет собой радиус графа.

Радиус графа и эксцентриситет вершины являются одними из базовых параметров, которые наиболее часто встречаются в прикладных задачах, либо необходимы при исследовании свойств графа [1, 2]. Этим объясняется довольно большое количество работ, в которых изучается данная задача.

1.1 Тривиальный алгоритм

Легко заметить, что задача поиска центральных вершин может быть с легкостью решена при использовании алгоритма обхода в ширину. Для того, чтобы найти вершину с минимальным эксцентриситетом необходимо запустить обход в ширину из каждой вершины графа, после чего станут известны все длины путей между всеми вершинами в графе. Алгоритм поиска в ширину имеет асимптотическое время работы равное $O(n + m)$ [3]. При этом, если запустить его из каждой вершины, то время работы всего алгоритма будет равным $O(n^2 + nm)$.

1.2 Алгоритмы с улучшенной ассимптотикой

Кроме этого данная задача была хорошо изучена различными исследователями [4–6], которые предлагали несколько подходов, для решения подобных задач. Например, в работе [7] предлагается алгоритм, использующий эвристику разделения вершин на два множества — множество вершин с высокой степенью и множество вершин с низкой степенью. После такого разделения для множества с вершинами с высокой степенью строится доминирующее множество, после чего из этого множества запускаются обходы в ширину. Кроме этого алгоритм поиска в ширину запускается внутри множества вершин с низкой степенью. Алгоритм обхода проходит не по всем вершинам в графе, а лишь по вершинам внутри множеств, что позволяет быстрее найти центральные вершины, а кроме этого радиус графа.

1.3 Алгоритмы, использующие матричное умножение

В работе [8] приводится алгоритм, который использует в своей основе матричное представление графов. Данный алгоритм оперирует в своей работе матрицами и самыми ресурсоемкими задачами в этом алгоритме являются процессы матричного перемножения, поэтому целиком и полностью ассимптотика этого подхода равна времени выполнения матричного умножения. Тривиальный алгоритм имеет ассимптотику $O(n^3)$, но при этом существует алгоритм быстрого матричного умножения [9], способный решить эту задачу за время $O(n^{2.81})$.

1.4 Генетические алгоритмы

Под генетическими алгоритмами принято подразумевать вероятностно-эвристические алгоритмы, которые применяются для решения задач оптимизации. Сфера применения генетических алгоритмов достаточно широка, с одной стороны данные алгоритмы могут быть применены при решении задач оптимизации, в которых недостаточно накопленных математических и алгоритмических знаний ввиду уникальности задачи или ее мало изученности. Кроме этого генетические алгоритмы могут быть применены при решении задач, для которых не существует эффективных алгоритмов решения — задачи из NP-класса, а также подобные алгоритмы могут найти применение при попытках уменьшить временные затраты на решение хорошо изученной задачи.

В основе любого генетического алгоритма лежит моделирование эволюционного развития живых организмов за счет таких факторов, как естественный отбор, мутация и скрещивание. Процесс скрещивания или кроссинговера впервые начал изучаться в XIX веке ученым-ботаником Г. Менделем. В результате его исследований было установлено, что в набор генов живых организмов передаются гены его родителей причем в скомбинированном виде. Этот факт во многом объясняет с одной стороны все многообразие живых существ, а с другой явление передачи полезных свойств через поколения.

В дальнейшем с развитием науки были сделаны ряд открытий, связанных с таким явлением, как мутация генов. Под мутацией понимается изменение генетических участков организма. Чаще всего эти изменения происходят под воздействием внешних факторов или внутренних. Такие мутации чаще всего приводят к негативным последствиям, но вместе с тем у живого организма появляется небольшая возможность получить новые внешние свойства, которые будут выгодно выделять его среди других организмов и переведут на новый виток эволюции.

В роли явления, которое отвечает за селекцию и выявление какие особи являются наиболее приспособленными к окружающей действительности, выступает естественный отбор. Отмеченный в работах Ч. Дарвина как один из ключевых процессов, который обеспечивает эволюционное развитие, естественный отбор сохраняет организмы с наиболее высоким уровнем приспособленности к окружающей среде и удаляет особи из популяции, которые недостаточно хорошо приспособлены.

При рассмотрении природы, которая окружает организм, как некоторой сложно организованной системы легко заметить, что в процессе эволюции живые существа под воздействием описанных факторов способны с легкостью решать некоторую оптимизационную задачу, находя в окружающих условиях оптимальные положения и состояния. В связи с этим была предложена идея генетических алгоритмов — смоделировать описанные три процесса и на их основе запустить оптимизационный поиск.

Каждый генетический алгоритм [10, 11] представляет собой итерационное применение операторов мутации, скрещивания и естественного отбора. При этом все эти операторы применяются к основной единице эволюции — популяции. В ходе такого итерационного применения этих операторов попу-

ляция должна найти некоторое оптимальное решение, при этом отнюдь не гарантируется, что это решение будет верным или же найденный оптимум будет являться глобальным.

1.4.1 Представление решения и начальная популяция

Первым этапом в реализации генетического алгоритма является выбор способа кодирования решения. Закодированные возможные решения будут представлять собой особи в популяции. Способ должен быть выбран таким образом, чтобы у операторов мутации и скрещивания была возможность с легкостью изменять каждую особь. Чаще всего при поиске оптимума некоторой вещественной функции каждая особь — это набор битов, которые кодируют вещественное число. Но данный подход не всегда может быть применен, поэтому способ кодирования решения выбирается чаще всего из постановки решаемой задачи. Одним из параметров генетического алгоритма является размер популяции N .

1.4.2 Оператор скрещивания

Данный оператор занимается выбором особей для скрещивания и самим процессом скрещивания. Задача этого оператора скомбинировать гены двух особей и создать на их основе новую особь для перехода в следующее поколение. При этом процесс скрещивания происходит не всегда, а с вероятностью заданной в виде параметра p_c .

1.4.3 Оператор мутации

Оператор просматривает каждую особь в популяции и некоторым образом ее изменяет, при этом работает так же с некоторой вероятностью заданной через параметр p_m .

1.4.4 Оператор естественного отбора

При естественном отборе важна функция для оценки приспособленности каждой особи в популяции. Чаще всего в качестве такой функции выступает функция, оптимальное значение которой ищется. Для начала оператор вычисляет приспособленность каждого организма в популяции, после чего формируется для каждой особи вероятность ее попадания в следующее поколение. Эта вероятность тем выше, чем наиболее оптимальное решение представляет

собой рассматриваемый элемент. Выбор элементов популяции осуществляется при помощи так называемого колеса рулетки — каждой особи ставится в соответствие сектор в зависимости от уровня вероятности, после чего происходит генерация псевдослучайного числа, и в зависимости от того в какой сектор попало число, тот элемент и переходит в следующее поколение. Очевидно, что в следующем поколении окажется большинство особей с высоким уровнем приспособленности.

Среди недостатков, которыми обладает данный подход, можно выделить неуниверсальность генетических алгоритмов. Каждая задача требует уникальной разработки и адаптации всех описанных этапов под решаемую задачу. Кроме этого успешность работы алгоритма зависит от значений параметров p_c , p_m и N .

Таким образом основными вопросами, которые стоят перед разработчиком, является реализация процессов скрещивания, мутации, естественного отбора и выбор критерия остановки алгоритма. Кроме этого необходимо исследовать параметры p_c , p_m и N , которые существенным образом влияют на работу генетического алгоритма.

Генетические алгоритмы также применялись для поиска центральных вершин. Например, в работе [12], приводится генетический алгоритм, позволяющий решать различного рода задачи из теории графов.

В основе построения данного алгоритма лежат классические этапы генетического подхода, при этом решение кодируется набором вершин, заданной мощности. В этой же работе рассматривается применение генетического алгоритма для решения целого ряда задач, одна из которых проблема нахождения k -центральных вершин, которая заключается в том, что необходимо найти ровно k центральных вершин, сумма эксцентрикитетов которых была бы минимальна. Несложно заметить, что задача будет совпадать с проблемой поиска центральной вершины, если в качестве k выбрать 1.

В описываемом алгоритме оператор скрещивания имеет следующую реализацию. Рассматриваются две особи (два множества), которые должны перейти в следующее поколение. Далее находится разность второго множества с первым, которая называется первым вектором обмена, также находится разность первого со вторым, которая называется вторым вектором обмена, после чего генерируется случайная позиция, относительно которой будет производится

скрещивание. После проделанных операций происходит обмен элементами относительно найденной позиции каждого множества с соответствующим ему вектором обмена. За счет такого подхода на каждой итерации алгоритма в множестве, описывающем конкретную особь, не существует повторяющихся вершин, при этом размер этого множества остается неизменным.

В качестве оператора мутации используется следующий подход: для каждой вершины находится набор ее соседей, после чего из этого множества случайным образом выбирается 4 вершины, которых нет в множестве особи. Для каждой из четырех вершин находится ее эксцентриситет, после чего с вероятностью заданной для оператора мутации вершина-сосед заменяет вершину в популяции. Данный подход был назван авторами N4N эвристикой, поэтому далее данный алгоритм будет называться «N4N алгоритм». В естественном отборе в качестве целевой функции ставиться значение эксцентриситета вершины.

Данный подход в реализации генетического алгоритма позволяет решать не только задачу поиска центральных вершин, но и еще ряд проблем связанных с теорией графов, однако ввиду общности способа представления решения и реализации оператора скрещивания, может иметь свои недостатки, которые могут проявиться в конкретной задаче.

2 Разработка генетического алгоритма и его исследование

Так как генетические алгоритмы являются методом решения оптимизационных задач, в задаче поиска центральных вершин необходимо выделить функцию, оптимальное значение которой требуется оптимизировать. Легко заметить, что если рассмотреть граф $G = (V, E)$ и функцию на нем $F : V \mapsto \mathbb{R}$, которая определяет эксцентризитет каждой вершины, то получается, что для нахождения центральных вершин необходимо найти минимальное значение этой функции с помощью генетического алгоритма.

Генетический алгоритм должен содержать адаптированные под решаемую задачу этапы скрещивания, мутации и естественного отбора. Основная идея алгоритма может быть выражена следующим образом. Пусть существует некоторое абстрактное или реальное изображение графа, причем в центре этого изображения находятся центральные вершины графа. Тогда, если популяция генетического алгоритма представляет собой набор вершин, то этот набор может быть представлен в виде некоторого шара, внутри которого находится центральные вершины графа (см. рисунок 1).

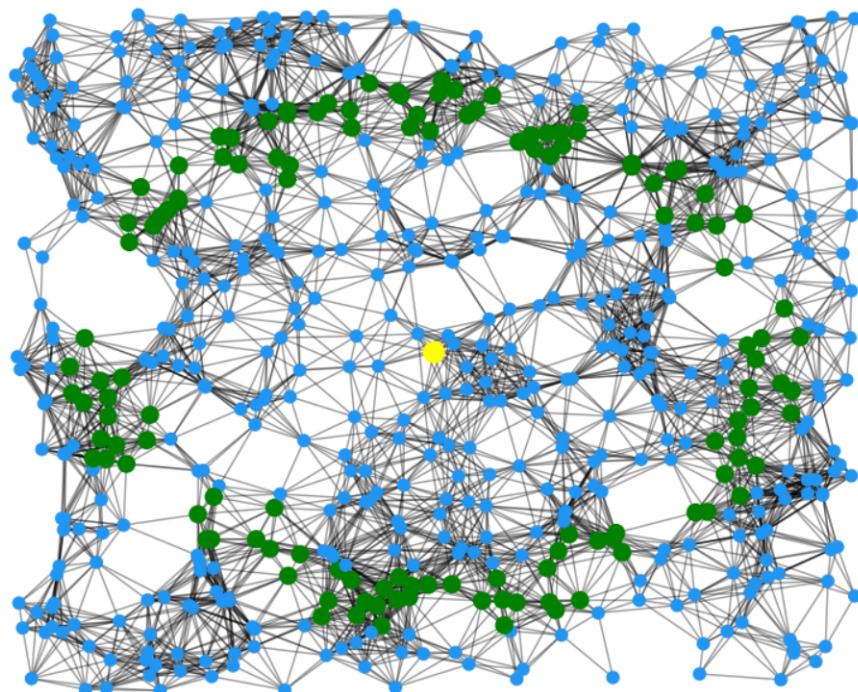


Рисунок 1 – Начальное состояние алгоритма (желтый цвет – центральная вершина, зеленый цвет – начальная популяция)

Из этой идеи следует, что для того чтобы найти центральные вершины необходимо, чтобы эта абстрактная сфера сжималась к центру. Именно этот

смысл заложен в работу оператора скрещивания. В добавок к этому алгоритм должен сжимать эту «сферу» к глобальному центру, не сбиваясь в локальные оптимальные значение, за это отвечает оператор мутации. Естественный отбор соответственно занимается выбором вершин с оптимальными эксцентричеситетами. Если реализовать все эти три шага и запустить их, то в идеальном варианте после нескольких итераций популяция алгоритма должна находиться в состоянии, которое показано на рисунке 2.

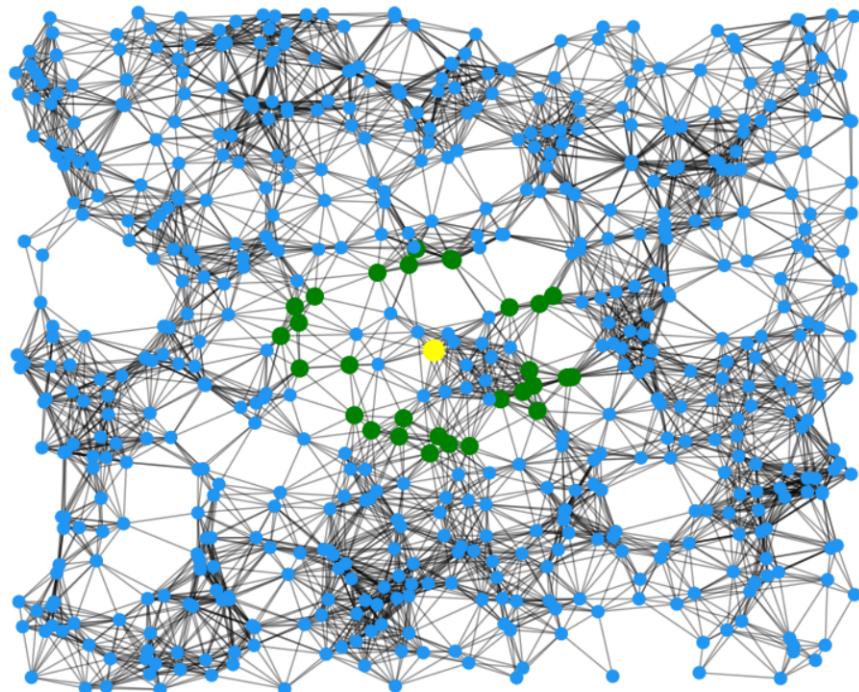


Рисунок 2 – Состояние алгоритма после нескольких итераций (желтый цвет – центральная вершина, зеленый цвет – начальная популяция)

2.1 Старт алгоритма

Для начала алгоритма необходимо сгенерировать начальную популяцию с размером N , которая и будет эволюционировать. В предлагаемом алгоритме в качестве начальной популяции генерируется случайный набор уникальных вершин, причем вероятность попадания в начальную популяцию одинакова для всех вершин.

2.2 Естественный отбор

Как уже говорилось ранее, естественный отбор занимается выбором оптимальных вершин для продолжения работы алгоритма. Для каждой вершины находится ее эксцентричеситет при помощи обхода в ширину (см. рисунок 3),

после чего методом колеса рулетки, при этом приоритет отдается вершинам с меньшим эксцентризитетом.

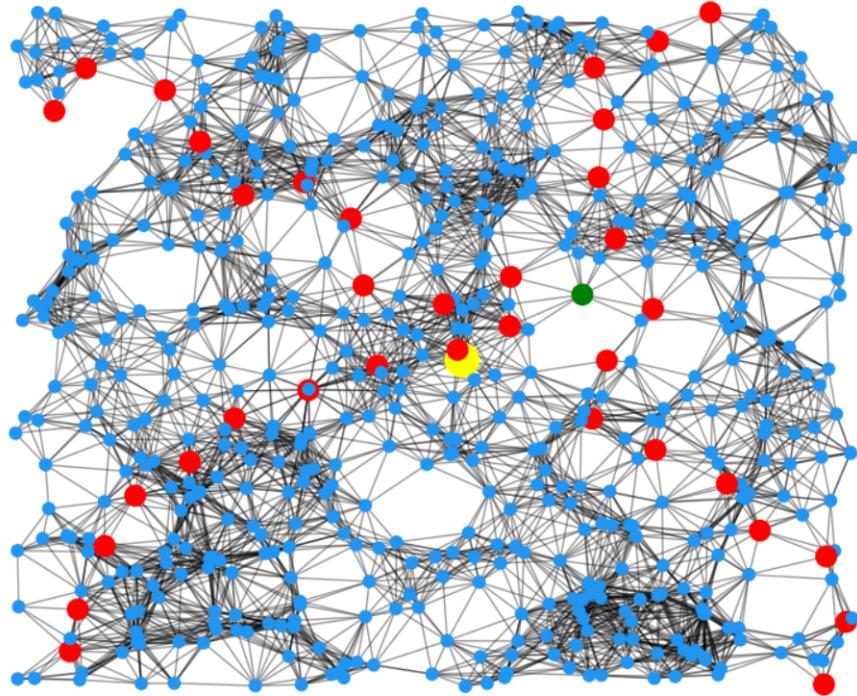


Рисунок 3 – Процесс поиска самой удаленной вершины (желтый цвет – центральная вершина, зеленый цвет – начальная популяция, красный – вершины на пути в самые удаленные точки графа)

2.3 Этап скрещивания

Основной смысл разработанного алгоритма кроется в операторе скрещивания. Так как необходимо, чтобы «сфера», описываемая популяцией, с каждой итерацией сжималась, то скрещивание реализовано следующим образом: в качестве родителей выбираются две вершины из популяции, после чего между ними находится кратчайший путь, после чего из этого пути выбирается одна вершина в качестве потомка (см. рисунок 4). Именно такая реализация данного этапа позволяет алгоритму сходиться к центральным вершинам. При этом процесс скрещивания происходит с вероятностью p_c .

2.4 Этап мутации

Для того, чтобы у алгоритма была возможность выхода из локальных оптимальных значений существует этап мутации, который имеет следующую реализацию: перебираются все вершины в популяции и для каждой вершины

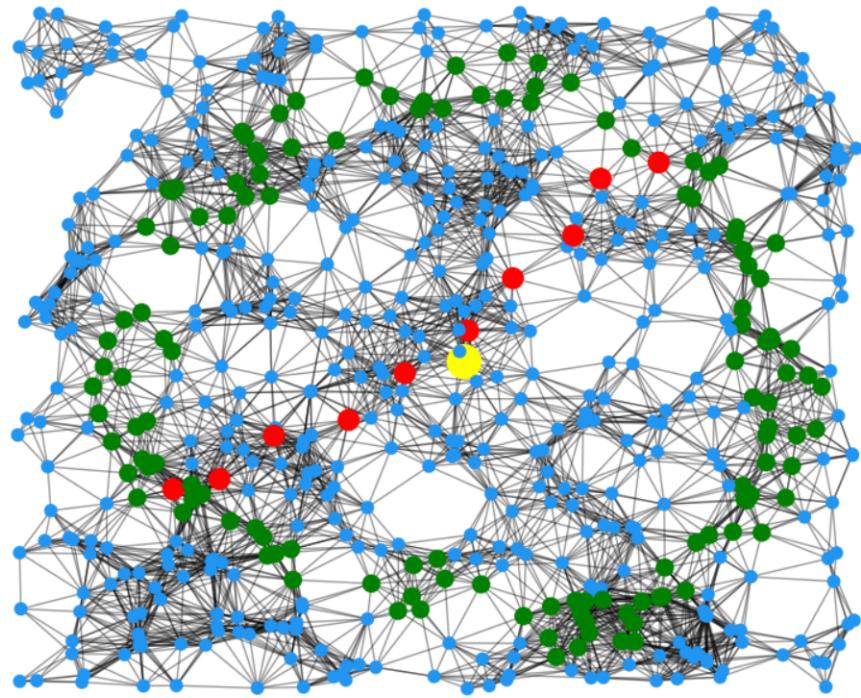


Рисунок 4 – Процесс скрещивания (желтый цвет – центральная вершина, зеленый цвет – начальная популяция, красный – кратчайший путь между вершинами популяции)

находятся ее соседи — вершины смежные с ней, после чего с вероятностью p_m вершина заменяется на одного из своих соседей (см. рисунок 5)

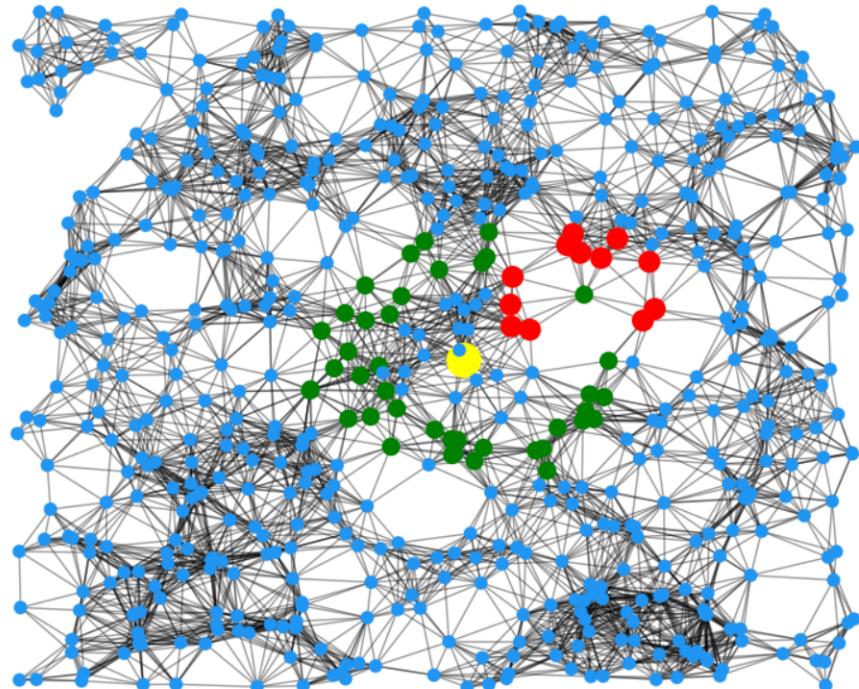


Рисунок 5 – Процесс мутации (желтый цвет – центральная вершина, зеленый цвет – начальная популяция, красный – соседние вершины одной из вершин в популяции)

Все описанные этапы представлены в виде псевдокода 1.

При работе каждого из этапов в оперативной памяти поддерживается матрица $n \times n$, где n — число вершин в графе, хранящая найденные расстояния между вершинами, а кроме этого такая же матрица, внутри которой лежат кратчайшие пути между вершинами. Такое поддержание матриц позволяет многократно не пересчитывать расстояния между вершинами, если этого требует алгоритм. Визуальную работу алгоритма можно увидеть на рисунке 7.

2.5 Исследование алгоритма

В качестве языка программирования для исследования алгоритма был выбран язык программирования C++. Кроме этого в качестве среды разработки, в которой производилась реализация алгоритма, была выбрана Visual Studio 2017, а вычислительная машина, на которой были выполнены все испытания обладает оперативной памятью с размером 6.0 GB и процессором AMD A8-7410 с частотой 2.20 GHz.

Кроме этого для запусков тестов алгоритма необходимы наборы графов с разными свойствами. Для создания графов использовалась Python библиотека NetworkX, которая предоставляет возможности для генерации данных на основе различных моделей случайных графов.

За основную работу алгоритма отвечает класс `GeneticAlgorithm`, в котором реализованы основные методы генетического алгоритма:

- `makeSelection` — метод, отвечающий за реализацию естественного отбора,
- `crossing` — метод, отвечающий за реализацию скрещивания,
- `mutation` — метод, отвечающий за реализацию мутации,
- `getBestResult` — метод, который запускает генетический алгоритм, измеряет время его работы и возвращает найденные оптимальные значения.

Полный код класса можно увидеть в приложении А.

При всем этом для хранения найденных путей и кратчайших расстояний реализован класс `Graph`, внутри которого находятся методы, отвечающие за запуск обхода в ширину и получение найденных расстояний и кратчайших путей. Основные методы этого класса:

- `bfsFromVertex(int x)` — метод, запускающий алгоритм обхода в ширину из вершины, переданной в качестве параметра

```

begin
    Data: Graph  $G$ 
    Result: Vertex  $c$ , which is center of graph  $G$ 
    for  $i := 1$  to  $populationSize$  do
        |  $population[i] \leftarrow$  random vertex  $\in G$ ;
    end
    for  $i \leftarrow 1$  to  $iterationNumber$  do
        |  $Crossover()$ 
        |  $Mutation()$ 
        |  $Selection()$ 
    end
     $c.eccentricity \leftarrow \infty$ 
    for  $v \in population$  do
        | if  $v.eccentricity < c.eccentricity$  then
        | |  $c \leftarrow v$ 
        | end
    end
end

Function  $Mutation()$ 
    Data: Population on current algorithm step
    Result: Population after applying a mutation operator to each individual
    for  $i \leftarrow 1$  to  $populationSize$  do
        | if  $randomValue < mutationProbability$  then
        | |  $neighbours \leftarrow population[i].getNeighbours$ 
        | |  $population[i] \leftarrow$  random vertex from  $neighbours$ 
        | end
    end
end

Function  $Crossover()$ 
    Data: Population on current algorithm step
    Result: Population after crossover
    for  $i \leftarrow 1$  to  $populationSize$  do
        | if  $randomValue < crossPopulation$  then
        | |  $u \leftarrow$  random vertex from  $popualtion$ 
        | |  $v \leftarrow$  random vertex from  $population$ 
        | |  $path \leftarrow G.pathBetween(u, v)$ 
        | end
    end
end

Function  $Selection()$ 
    Data: Population on current algorithm step
    Result: Population for next algorithm step
    for  $i \leftarrow 1$  to  $populationSize$  do
        |  $eccentricity[i] \leftarrow population[i].eccentricity$ 
    end
    for  $i \leftarrow 1$  to  $populationSize$  do
        |  $probability[i] \leftarrow$  number from  $[0, 1]$ 
    end
    for  $i \leftarrow 1$  to  $populationNumber$  do
        |  $nextPopulation[i] \leftarrow v$  from  $population$  with using probability
    end
end

```

Algorithm 1: Псевдокод основных этапов предлагаемого алгоритма

- `getPath(int x, int y)` — метод, возвращающий расстояние между двумя вершинами в графе,
- `getEccentricity(int x)` — метод, возвращающий эксцентризитет вершины,
- `getNeighbour(int x)` — метод, получающий соседей заданной вершины.

Также с кодом этого класса можно ознакомиться в приложении Б. Сгенерированные графы, на которых тестируется алгоритм, хранятся в текстовых файлах. В текстовых файлах находятся ребра графа, которые записаны на каждой строке, при этом на первой строке записано число вершин в графе и число ребер. Для того, чтобы прочитать текстовый график была реализована следующая функция чтения:

```

1 vector<pair<int, int>> readFromFileNM(string file, int& N, int& M) {
2     ifstream in(file);
3     vector<pair<int, int>> e;
4     int n, m;
5     in >> n >> m;
6     for (int i = 0; i < m; i++) {
7         int x, y;
8         in >> x >> y;
9         e.push_back(make_pair(x, y));
10    }
11    N = n;
12    M = m;
13    return e;
14 }
```

методу на вход передаются такие параметры, как имя файла, ссылка на переменную для числа вершин в графике, а также ссылка на число ребер в графике. Метод возвращает вектор пар, где каждая пара описывает ребро графа.

За связывание файлов с графиками с их найденными ранее радиусами отвечает класс `Repository`, внутри которого закодированы имена файлов и дополнительная информация об этих файлах. Для того, чтобы получить график существует публичный метод `getGraph`, который принимает строку, описывающую какой тип графа запрошен, после чего открывается файл, график считывается и возвращается из метода. Со всеми этими действиями можно ознакомиться в приложении Б.

Для того, чтобы данные, возвращаемые из методов или передаваемые в качестве параметров были соединены в единую структуру были созданы следующие сущности. Класс GraphDescription:

```
1 class GraphDescription {
2 public:
3     int n, m;
4     edges e;
5     int realR;
6     GraphDescription() {}
7
8     GraphDescription(int n, int m, edges e, int realR) {
9         this->n = n;
10        this->m = m;
11        this->e = e;
12        this->realR = realR;
13    }
14};
```

используется для хранения графа в нем содержаться поля, отвечающие за размеры графа и его радиус.

Кроме этого для того, чтобы хранить результаты вычислительных экспериментов существует класс GaTestResult:

```
1 class GaTestResult {
2 public:
3     GaTestResult() {}
4
5     GaTestResult(double popSize, double pm,
6                  double pc, double functionValue) {
7         this->pm = pm;
8         this->pc = pc;
9         this->popSize = popSize;
10        this->functionValue = functionValue;
11    }
12
13    double pc, pm;
14    int popSize;
15    double functionValue;
16};
```

В объектах данного класса хранятся значения вычисленной функции,

например времени, а кроме этого значения параметров, на которых были получены эти результаты.

Для того, чтобы облегчить работу с запусками алгоритма с разными параметрами и легко проводить различного рода тесты создан класс `Experiment`. Данный класс включает реализацию всех необходимых методов для исследования параметров алгоритма:

- `calculateTimeErrorValue` — private метод, который запускает генетический алгоритм, и возвращает среднее время работы и процент ошибок,
- `testWithChangebleProb` — метод который принимает размер популяции и фиксированный параметр, а так же флаг показывающий какой из параметров будет изменяться для исследования,
- `oneDimentionFixedGATest` — запуск теста с изменяющимся параметром,
- `simpleTimeErrorTest` — простой запуск алгоритма, с переданными ему вероятностью скрещивания, мутации и размеров популяции,
- `simpleNANTest` — запуск алгоритма $N4N$,
- `rmpcGaTest` — метод для запуска теста с перебором всех значений p_c и p_m ,
- `nGATest` — метод для запуска теста с перебором параметра N .

Полный код класса представлен в приложении **B**.

2.6 Модели случайных графов

Для исследования алгоритма применялись следующие модели случайных графов: модель Эрдеша-Ренъи, модель Барабаши-Альберт и модель случайного геометрического графа. Все они в той или иной степени частично описывают реальные графы.

2.6.1 Модель случайного графа Эрдеша-Ренъи

Данная модель представлена в работе [13] и является одной из самых простых и базовых моделей случайного графа. Изначально для создания графа задаются два параметра n — число вершин в графе и p — вероятность проведения ребра. Далее для построения графа рассматриваются все пары вершин, и между ними проводится неориентированное ребро с вероятностью p . В данной модели ввиду ее простоты формулировки довольно легко выводятся формулы, зависящие от параметра p и n , которые описывают основные характеристики графов, такие как связность, размер максимальной клики, распределение

степени вершин и т.д. Поэтому, если удается установить, что граф в рассматриваемой задаче близок по свойствам с графом Эрдеша-Ренъи, то можно без труда получить много информации об изучаемом графе.

2.6.2 Модель случайного графа Барабаши-Альберт

Данная модель случайного графа была предложена в работе [14]. На данный момент описанная модель позволяет создавать так называемые безмасштабные сети — графы, в которых распределение степеней подчиняется степенному закону. Исследования данного подхода показали, что графовые модели, которые описывают взаимосвязи внутри различных самоорганизующихся систем, совпадают с моделью Барабаши-Альберт. К таким сетям относятся ряд графов социальных сетей, сеть Интернет, ряд графовых моделей в природных сетях.

В ходе построения случайного графа поддерживаются два основных принципа, которые главным образом характеризуют безмасштабные сети. Первый из них принцип расширения сети, второй — предпочтительное прикрепление. Первый принцип описывается тем фактом, что в существующую сеть могут постоянно добавляться новые узлы, при этом не нарушая его свойств из-за второго принципа. Принцип предпочтительного прикрепления заключается в том, что добавляемый узел случайным образом прикрепляется ребрами к вершинам, которые уже существуют в графе, при этом предпочтение отдается вершинам с большей степенью, формально вероятность проведения ребра к i -му узлу описывается следующий формулой:

$$p_i = \frac{k_i}{\sum_j k_j},$$

где k — степень узла, j пробегает все вершины в графе.

Для создания графа задаются два параметра n — число вершин в создаваемом графе, и t — число ребер, которые проводятся из каждой новой добавляемой вершины в граф. Алгоритм создания достаточно прост, в качестве начального графа берется связный граф с числом вершин большим или равным t , после чего добавляются новые вершины до необходимого количества, при этом каждая новая вершина соединяется с t вершинами случайным образом с вероятностным распределением, описанным формулой выше.

2.6.3 Геометрический случайный граф

Данная модель случайного графа [15] описывает основные свойства, которые возникают в графовых моделях компьютерных сетей и сетей, имеющих явную географическую интерпретацию. Для построения графа выбирается размерность пространства, в котором будет создаваться граф, наиболее часто выбирается двумерное пространство, на котором случайным образом генерируется набор геометрических точек равных по количеству числу узлов в создаваемом графе, после чего для каждой пары точек рассчитывается евклидово расстояние между ними и проводится ребро в том случае, если расстояние меньше параметра r , который задается заранее.

Данная модель случайного графа имеет свои отличительные характеристики, которые не совпадают с моделями Эрдеша-Ренни и Барабаши-Альберт.

2.7 Результаты вычислительных экспериментов

Для выявления сильных и слабых сторон предложенного алгоритма его сравнение проводилось с рядом точных алгоритмов, а также с алгоритмом «N4N». В качестве тестовых графов были выбраны графовые модели описанные ранее. Для модели Эрдеша-Ренни в качестве параметра p было выбрано значение 5%, для модели Барабаши-Альберт $m = 2$, а для геометрического случайного графа $r = 0.1$.

Для сравнения по временным результатам были реализованы тривиальный алгоритм и алгоритм с улучшенной асимптотикой. Эти алгоритмы и генетический алгоритм описанный в данной работе запускались на трех моделях случайных графов, с количеством вершин 500, 1000, 1500, 2000, 2500, 5000. Исходя из практических экспериментов в качестве размера популяции выбрано значение 50, для оператора скрещивания 0.7, для оператора мутации 0.1, кроме этого число итераций ограничено числом 20. Результаты временных измерений приведены на графиках 7. Из полученных результатов видно, что созданный генетический алгоритм дает выигрыш по времени в несколько раз по сравнению с точными алгоритмами.

Также так как эвристический алгоритм не гарантирует получение точного ответа, а допускает некий процент ошибки, то для изучения точности алгоритма были проведены тесты позволяющие выявить процент неправильных ответов. Для этого алгоритм запускался на все тех же графах, при этом

так как размерности графа, позволяют за приемлемые временные затраты с помощью точного алгоритма найти центральные вершины, то зная эту информацию можно говорить о проценте неправильных ответов. Для сравнения брался алгоритм «N4N», после чего оба алгоритма запускались 100 раз, что позволило подсчитать процент ошибки. Результаты вычислительных экспериментов приведены в таблицах 1, 2 и 3.

Таблица 1 – Время работы алгоритма и процент неверных ответов на модели случайного графа Эрдоша-Ренъи. GA – генетический алгоритм, N4NA – N4N алгоритм, AA – алгоритм с улучшенной ассимптотикой

| Размеры графа | | Время, сек. | | | Ошибка, % | |
|---------------|---------|-------------|------|-------|-----------|------|
| $ V $ | $ E $ | GA | N4NA | AA | GA | N4NA |
| 500 | 3572 | 0.11 | 0.39 | 0.07 | 38.0 | 40.0 |
| 1000 | 14202 | 0.31 | 0.77 | 0.56 | 21.0 | 50.0 |
| 1500 | 31861 | 0.71 | 1.44 | 1.13 | 13.0 | 62.0 |
| 2000 | 57438 | 1.20 | 1.92 | 1.70 | 8.0 | 48.0 |
| 2500 | 90268 | 1.76 | 2.68 | 3.20 | 4.0 | 30.0 |
| 5000 | 358553 | 4.48 | 8.61 | 18.45 | 0.0 | 0.0 |
| 10000 | 1439255 | 13.54 | 26.0 | 41.47 | 0.0 | 0.0 |

Таблица 2 – Время работы алгоритма и процент неверных ответов на модели случайного графа Барабаши-Альберт. GA – генетический алгоритм, N4NA – N4N алгоритм, AA – алгоритм с улучшенной ассимптотикой

| Размеры графа | | Время, сек. | | | Ошибка, % | |
|---------------|-------|-------------|------|-------|-----------|------|
| $ V $ | $ E $ | GA | N4NA | AA | GA | N4NA |
| 500 | 996 | 0.07 | 0.29 | 0.08 | 16.0 | 0.0 |
| 1000 | 1996 | 0.18 | 0.68 | 0.36 | 12.0 | 0.0 |
| 1500 | 2996 | 0.37 | 1.24 | 0.98 | 4.0 | 0.0 |
| 2000 | 3996 | 0.55 | 1.67 | 1.68 | 1.0 | 0.0 |
| 2500 | 4996 | 0.69 | 2.18 | 3.22 | 0.0 | 0.0 |
| 5000 | 9996 | 1.84 | 8.28 | 14.17 | 0.0 | 0.0 |
| 10000 | 19996 | 3.90 | 15.8 | 24.56 | 0.0 | 0.0 |

Из полученных результатов видно, что созданный алгоритм не уступает существующему эвристическому алгоритму. Предложенный алгоритм превосходит второй генетический алгоритм в несколько раз. Это во многом объясняется тем, что в алгоритме «N4N» используется множество для описания одной особи в популяции, а также при процессе мутации для вершин, которые претендуют на изменение особи, находится эксцентриситет, всех этих процессов нет в созданном алгоритме, поэтому его время работы меньше. При этом

Таблица 3 – Время работы алгоритма и процент неверных ответов на модели случайного геометрического графа. GA – генетический алгоритм, N4NA – N4N алгоритм, AA – алгоритм с улучшенной асимптотикой

| Размеры графа | | Время, сек. | | | Ошибка, % | |
|---------------|---------|-------------|------|-------|-----------|------|
| $ V $ | $ E $ | GA | N4NA | AA | GA | N4NA |
| 500 | 3572 | 0.11 | 0.39 | 0.07 | 38.0 | 40.0 |
| 1000 | 14202 | 0.31 | 0.77 | 0.56 | 21.0 | 50.0 |
| 1500 | 31861 | 0.71 | 1.44 | 1.13 | 13.0 | 62.0 |
| 2000 | 57438 | 1.20 | 1.92 | 1.70 | 8.0 | 48.0 |
| 2500 | 90268 | 1.76 | 2.68 | 3.20 | 4.0 | 30.0 |
| 5000 | 358553 | 4.48 | 8.61 | 18.45 | 0.0 | 0.0 |
| 10000 | 1439255 | 13.54 | 26.0 | 41.47 | 0.0 | 0.0 |

видно, что алгоритм «N4N» на некоторых моделях случайных графов имеет меньший процент ошибки по сравнению с предложенным алгоритмом, однако этот процент нивелируется с увеличением размерности графа.

2.8 Исследование параметров генетического алгоритма

В приведенных ранее результатах в качестве значений параметров генетического алгоритма были выбраны значения $N = 50$, $p_c = 0.7$ и $p_m = 0.1$. Однако эти значения выбирались в качестве тестовых для того, чтобы проверить жизнеспособность идей, заложенных в алгоритм. При этом эти параметры ключевым образом влияют как на точность алгоритма, так и на время его выполнения. В связи с этим был также проведен еще ряд экспериментов, в которых исследовались эти параметры. Для начала были произведены запуски, в которых перебирались значения для p_m и p_c от 0 до 1. Для каждого значения этих параметров измерялось время работы алгоритма и его процент ошибок. При этом значение размера популяции было равно 20. Результаты этих экспериментов представлены в таблицах 4 и 5.

Из этих таблиц видно, что время работы алгоритма растет по мере увеличения как параметра p_m , так и параметра p_c . Вместе с тем видно, что и процент неверных ответов падает по мере увеличения тех же параметров. Очевидно, что необходимо найти некоторые оптимальные значения для того, чтобы процент неверных ответов был невелик и одновременно с этим необходимо, чтобы время работы было минимальным. Для того чтобы соединить воедино два этих фактора была введена функция:

Таблица 4 – Время работы (сек.) алгоритма на представителе модели геометрического случайного графа $|V| = 2500$, $|E| = 90268$, $N = 20$

| pm/pc | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|-------|------|------|------|------|------|------|------|------|------|------|------|
| 0.0 | 0.06 | 0.11 | 0.13 | 0.15 | 0.17 | 0.19 | 0.20 | 0.29 | 0.28 | 0.27 | 0.31 |
| 0.1 | 0.18 | 0.26 | 0.27 | 0.33 | 0.32 | 0.36 | 0.36 | 0.41 | 0.40 | 0.40 | 0.40 |
| 0.2 | 0.29 | 0.35 | 0.52 | 0.55 | 0.52 | 0.54 | 0.49 | 0.54 | 0.56 | 0.54 | 0.50 |
| 0.3 | 0.45 | 0.50 | 0.53 | 0.57 | 0.60 | 0.54 | 0.60 | 0.58 | 0.65 | 0.68 | 0.66 |
| 0.4 | 0.51 | 0.53 | 0.64 | 0.71 | 0.69 | 0.62 | 0.64 | 0.66 | 0.70 | 0.75 | 0.78 |
| 0.5 | 0.62 | 0.67 | 0.74 | 0.79 | 0.78 | 0.84 | 0.87 | 0.82 | 0.80 | 0.82 | 0.93 |
| 0.6 | 0.60 | 0.77 | 0.77 | 0.96 | 0.87 | 0.86 | 0.88 | 0.93 | 1.01 | 1.04 | 1.14 |
| 0.7 | 0.86 | 0.96 | 1.04 | 1.09 | 1.13 | 1.09 | 1.00 | 1.15 | 1.03 | 1.32 | 1.00 |
| 0.8 | 0.76 | 0.85 | 0.89 | 0.93 | 0.98 | 0.96 | 1.02 | 1.19 | 1.05 | 1.08 | 1.13 |
| 0.9 | 0.82 | 0.91 | 1.01 | 1.01 | 1.23 | 1.29 | 1.30 | 1.31 | 1.18 | 1.20 | 1.10 |
| 1.0 | 0.96 | 1.08 | 1.03 | 1.22 | 1.25 | 1.30 | 1.30 | 1.26 | 1.18 | 1.32 | 1.50 |

Таблица 5 – Процент ошибок (%) алгоритма на представителе модели геометрического случайного графа $|V| = 2500$ $|E| = 90268$ $N = 20$

| pm/pc | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.0 | 81 | 54 | 32 | 52 | 42 | 52 | 59 | 53 | 62 | 60 | 61 |
| 0.1 | 48 | 35 | 19 | 28 | 33 | 34 | 28 | 32 | 23 | 39 | 31 |
| 0.2 | 46 | 27 | 29 | 20 | 21 | 25 | 21 | 25 | 26 | 21 | 17 |
| 0.3 | 44 | 23 | 18 | 17 | 20 | 18 | 15 | 14 | 15 | 17 | 21 |
| 0.4 | 32 | 28 | 19 | 22 | 9 | 16 | 21 | 16 | 18 | 30 | 16 |
| 0.5 | 32 | 16 | 14 | 19 | 20 | 16 | 22 | 18 | 14 | 14 | 9 |
| 0.6 | 30 | 18 | 12 | 21 | 11 | 12 | 15 | 9 | 12 | 20 | 18 |
| 0.7 | 32 | 17 | 20 | 14 | 13 | 10 | 19 | 12 | 22 | 8 | 12 |
| 0.8 | 33 | 22 | 20 | 14 | 21 | 15 | 21 | 16 | 15 | 14 | 19 |
| 0.9 | 24 | 18 | 24 | 15 | 23 | 17 | 19 | 14 | 20 | 11 | 14 |
| 1.0 | 32 | 16 | 13 | 14 | 12 | 15 | 16 | 14 | 7 | 13 | 9 |

$$F(pm, pc) = \alpha time(pm, pc) + \beta error(pm, pc), \quad (1)$$

которая учитывает время работы и процент ошибок, причем параметр α отвечает за уровень значимости временных затрат, а параметр β отвечает за значимость процента ошибок. Если подставить все полученные данные в эту функцию, то получится результат, который представлен в таблице 6.

Из этой таблицы видно, что оптимальных значений функция F достигает в центре таблицы. Например, в эксперименте, который описывает таблица 6 оптимальное значение достигается при $p_m = 0.4$ и $p_c = 0.4$. При фиксированных значениях параметра $p_m = 0.4$ и $p_m = 0.6$ были построены графики 8 и 9, на которых отражено нормированное время работы и нормированный процент ошибки. Из этих графиков видно, что при увеличении параметра p_m значительно увеличивается и время работы алгоритма и уменьшается процент

Таблица 6 – Значения функции F , $\alpha = 0.3$, $\beta = 0.7$, $|V| = 2500$, $|E| = 90268$, $N = 20$

| pm/pc | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|-------|------|------|------|------|------|------|------|------|------|------|------|
| 0.0 | 0.71 | 0.49 | 0.30 | 0.48 | 0.40 | 0.49 | 0.55 | 0.52 | 0.59 | 0.57 | 0.59 |
| 0.1 | 0.45 | 0.36 | 0.22 | 0.31 | 0.35 | 0.37 | 0.32 | 0.36 | 0.28 | 0.42 | 0.35 |
| 0.2 | 0.46 | 0.31 | 0.36 | 0.28 | 0.29 | 0.32 | 0.28 | 0.32 | 0.34 | 0.29 | 0.25 |
| 0.3 | 0.47 | 0.30 | 0.26 | 0.26 | 0.29 | 0.27 | 0.25 | 0.24 | 0.26 | 0.28 | 0.32 |
| 0.4 | 0.38 | 0.35 | 0.29 | 0.33 | 0.22 | 0.26 | 0.31 | 0.27 | 0.30 | 0.41 | 0.30 |
| 0.5 | 0.40 | 0.27 | 0.27 | 0.32 | 0.33 | 0.31 | 0.37 | 0.32 | 0.28 | 0.29 | 0.26 |
| 0.6 | 0.38 | 0.31 | 0.26 | 0.37 | 0.27 | 0.28 | 0.31 | 0.26 | 0.31 | 0.38 | 0.38 |
| 0.7 | 0.45 | 0.34 | 0.38 | 0.34 | 0.34 | 0.30 | 0.36 | 0.33 | 0.40 | 0.33 | 0.30 |
| 0.8 | 0.44 | 0.36 | 0.35 | 0.31 | 0.38 | 0.32 | 0.39 | 0.38 | 0.34 | 0.34 | 0.39 |
| 0.9 | 0.37 | 0.34 | 0.41 | 0.33 | 0.44 | 0.40 | 0.42 | 0.38 | 0.41 | 0.34 | 0.34 |
| 1.0 | 0.47 | 0.35 | 0.32 | 0.36 | 0.35 | 0.39 | 0.40 | 0.37 | 0.30 | 0.38 | 0.38 |

неверных ответов.

Кроме этого важным параметром для генетического алгоритма является размер популяции N и в связи с этим был проведен эксперимент, в котором измерялся процент ошибок и время работы при фиксированных значениях $p_m = 0.4$ и $p_c = 0.4$ [10](#) и [11](#).

На этих графиках видно, что время работы алгоритма имеет практически линейную зависимость, а также процент ошибок экспоненциально уменьшается в зависимости от числа N и после значения 30, становится близким к нулю.

Очевидно, что в зависимости от того каким образом будут изменяться размеры графов, будет зависеть и набор оптимальных значений параметров. В связи с этим были проведены вычислительные эксперименты, на других графе другой размерности. Результаты этих экспериментов представлены в таблицах [7](#), [8](#), [9](#).

Таблица 7 – Время работы алгоритма (сек.) на представителе модели геометрического случайного графа $|V| = 5000$, $|E| = 358553$, $N = 20$

| pm/pc | 0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|-------|------|------|------|------|------|------|------|------|------|------|------|
| 0.00 | 0.07 | 0.13 | 0.17 | 0.21 | 0.33 | 0.30 | 0.39 | 0.38 | 0.39 | 0.32 | 0.36 |
| 0.10 | 0.22 | 0.31 | 0.36 | 0.39 | 0.42 | 0.44 | 0.47 | 0.53 | 0.59 | 0.55 | 0.57 |
| 0.20 | 0.44 | 0.54 | 0.60 | 0.64 | 0.76 | 0.73 | 0.64 | 0.75 | 0.67 | 0.69 | 0.68 |
| 0.30 | 0.52 | 0.77 | 0.80 | 0.83 | 0.84 | 0.88 | 0.80 | 0.83 | 0.85 | 0.85 | 0.86 |
| 0.40 | 0.61 | 0.74 | 0.83 | 1.17 | 1.02 | 1.12 | 1.07 | 1.01 | 1.13 | 1.20 | 1.34 |
| 0.50 | 0.88 | 1.07 | 0.96 | 0.98 | 1.03 | 1.06 | 1.08 | 1.10 | 1.11 | 1.13 | 1.29 |
| 0.60 | 0.82 | 0.96 | 1.03 | 1.16 | 1.28 | 1.25 | 1.30 | 1.31 | 1.29 | 1.39 | 1.46 |
| 0.70 | 1.17 | 1.14 | 1.25 | 1.46 | 1.36 | 1.44 | 1.58 | 1.40 | 1.46 | 1.44 | 1.56 |
| 0.80 | 1.05 | 1.18 | 1.27 | 1.36 | 1.41 | 1.43 | 1.49 | 1.51 | 1.56 | 1.54 | 1.57 |
| 0.90 | 1.17 | 1.31 | 1.42 | 1.66 | 1.55 | 1.58 | 1.61 | 1.66 | 1.68 | 1.74 | 1.63 |
| 1.00 | 1.19 | 1.41 | 1.52 | 1.73 | 1.69 | 1.69 | 1.75 | 1.72 | 1.79 | 1.86 | 1.89 |

Таблица 8 – Процент ошибок (%) на представителе модели геометрического случайного графа $|V| = 5000$, $|E| = 358553$, $N = 20$

| pm/pc | 0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|-------|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 0.00 | 81 | 53 | 41 | 45 | 47 | 40 | 53 | 49 | 53 | 54 | 51 |
| 0.10 | 70 | 39 | 31 | 28 | 33 | 34 | 21 | 25 | 26 | 25 | 24 |
| 0.20 | 45 | 22 | 30 | 14 | 27 | 21 | 20 | 18 | 27 | 22 | 21 |
| 0.30 | 52 | 27 | 23 | 19 | 17 | 15 | 19 | 21 | 21 | 23 | 22 |
| 0.40 | 47 | 25 | 18 | 25 | 18 | 14 | 23 | 19 | 19 | 15 | 16 |
| 0.50 | 36 | 22 | 19 | 23 | 18 | 20 | 16 | 15 | 18 | 17 | 16 |
| 0.60 | 41 | 16 | 27 | 16 | 18 | 12 | 18 | 15 | 20 | 9 | 10 |
| 0.70 | 45 | 23 | 21 | 18 | 17 | 20 | 16 | 23 | 12 | 18 | 16 |
| 0.80 | 39 | 22 | 20 | 22 | 17 | 16 | 16 | 15 | 13 | 19 | 17 |
| 0.90 | 35 | 25 | 17 | 20 | 19 | 20 | 19 | 22 | 11 | 20 | 17 |
| 1.00 | 33 | 19 | 20 | 18 | 17 | 16 | 18 | 21 | 10 | 11 | 17 |

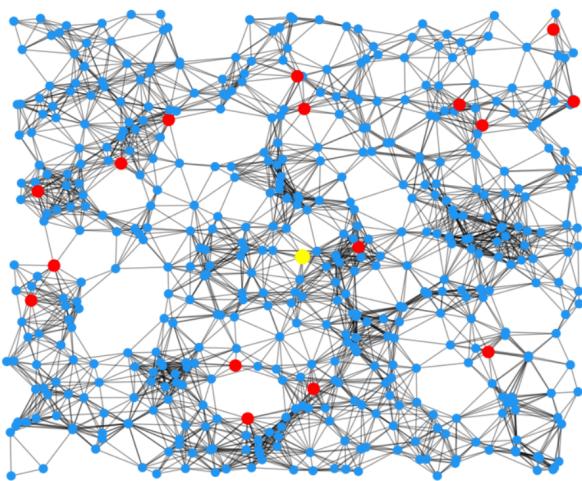
Таблица 9 – Значения функции F , $\alpha = 0.3$, $\beta = 0.7$, $|V| = 5000$, $|E| = 358553$, $N = 20$

| pm/pc | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|-------|------|------|------|------|------|------|------|------|------|------|------|
| 0.00 | 0.71 | 0.48 | 0.38 | 0.42 | 0.46 | 0.39 | 0.52 | 0.48 | 0.52 | 0.52 | 0.50 |
| 0.10 | 0.64 | 0.39 | 0.33 | 0.30 | 0.35 | 0.36 | 0.26 | 0.30 | 0.32 | 0.30 | 0.30 |
| 0.20 | 0.46 | 0.28 | 0.35 | 0.22 | 0.35 | 0.30 | 0.27 | 0.27 | 0.34 | 0.30 | 0.29 |
| 0.30 | 0.53 | 0.36 | 0.33 | 0.30 | 0.28 | 0.27 | 0.29 | 0.31 | 0.32 | 0.33 | 0.33 |
| 0.40 | 0.50 | 0.33 | 0.29 | 0.40 | 0.32 | 0.30 | 0.37 | 0.32 | 0.34 | 0.32 | 0.35 |
| 0.50 | 0.45 | 0.36 | 0.32 | 0.35 | 0.32 | 0.34 | 0.31 | 0.30 | 0.33 | 0.33 | 0.34 |
| 0.60 | 0.48 | 0.29 | 0.40 | 0.32 | 0.36 | 0.30 | 0.36 | 0.34 | 0.38 | 0.30 | 0.32 |
| 0.70 | 0.57 | 0.38 | 0.38 | 0.39 | 0.36 | 0.40 | 0.39 | 0.42 | 0.34 | 0.38 | 0.39 |
| 0.80 | 0.50 | 0.38 | 0.37 | 0.41 | 0.37 | 0.37 | 0.37 | 0.37 | 0.36 | 0.41 | 0.40 |
| 0.90 | 0.49 | 0.42 | 0.37 | 0.44 | 0.41 | 0.42 | 0.42 | 0.45 | 0.36 | 0.45 | 0.41 |
| 1.00 | 0.47 | 0.39 | 0.41 | 0.43 | 0.42 | 0.41 | 0.43 | 0.45 | 0.37 | 0.39 | 0.45 |

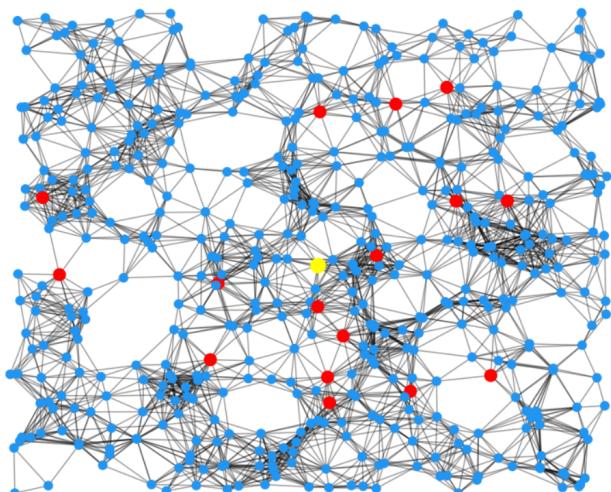
Из этих таблиц видно, что тенденция роста времени работы сверху вниз и слева направо сохранена, что можно сказать и про такую же тенденцию к снижению ошибок. Вместе с тем из таблицы 9 видно, что оптимальные значения параметров равны $p_m = 0.2$ и $p_c = 0.3$. На этом же графе был произведен замер времени работы и процента ошибок в зависимости от размеров популяции. Результаты этих измерений представлены на графиках 12 и 13.

Из этих графиков точно так же видно, что время работы растет линейно, а процент неверных ответов падает экспоненциально. Кроме этого для некоторых значений N было найдено минимальное значение функции F при различных значениях α и β (см. рисунки 14 и 15).

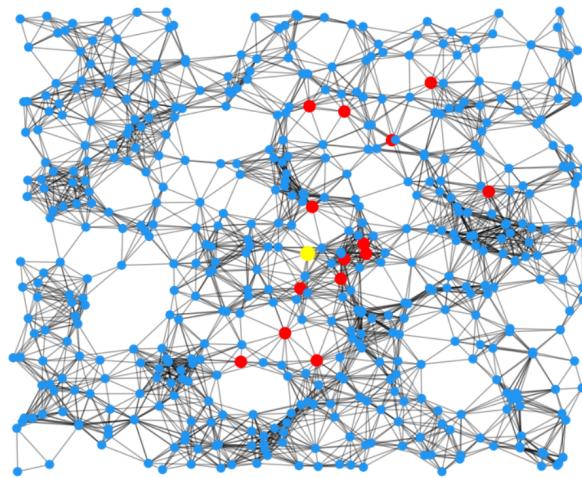
Как видно из графиков, при больших значениях параметра α , отражающего значимость временных затрат значения функции F возрастают с увеличением размеров популяции, а при равных значениях α и β такого большого изменения функции не происходит.



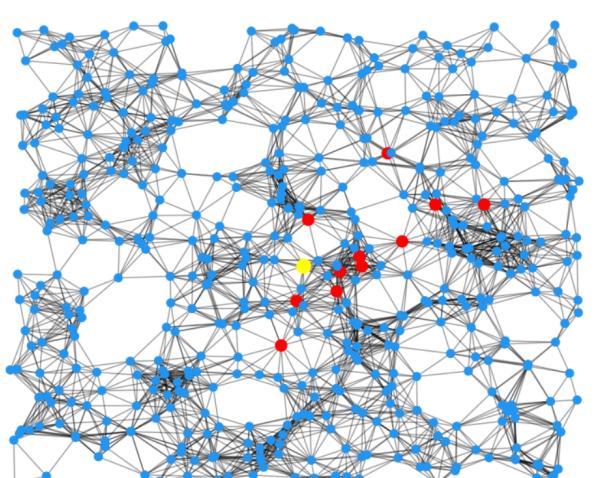
Начальная популяция



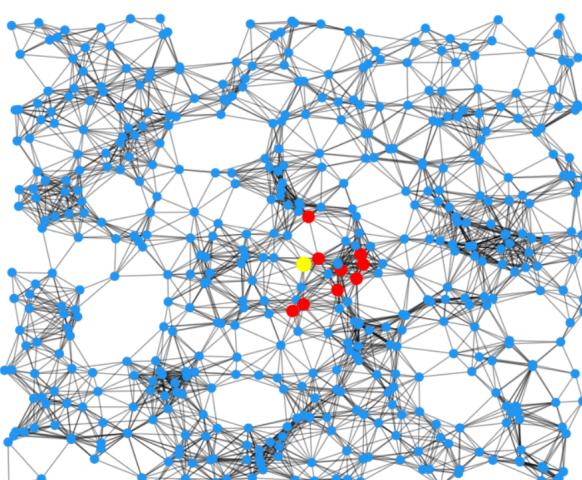
Популяция после двух шагов



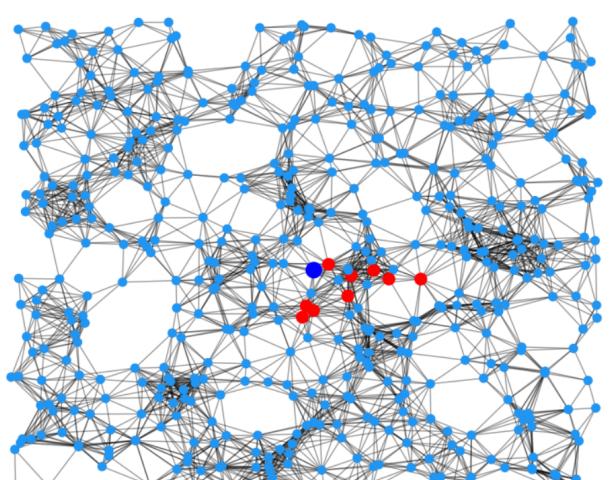
Популяция после четырех шагов



Популяция после шести шагов



Популяция после восьми шагов



Популяция после десяти шагов

Рисунок 6 – Шаги работы алгоритма (желтый цвет – центральная вершина, красный – популяция, темно синий – особь, представляющая правильный ответ)

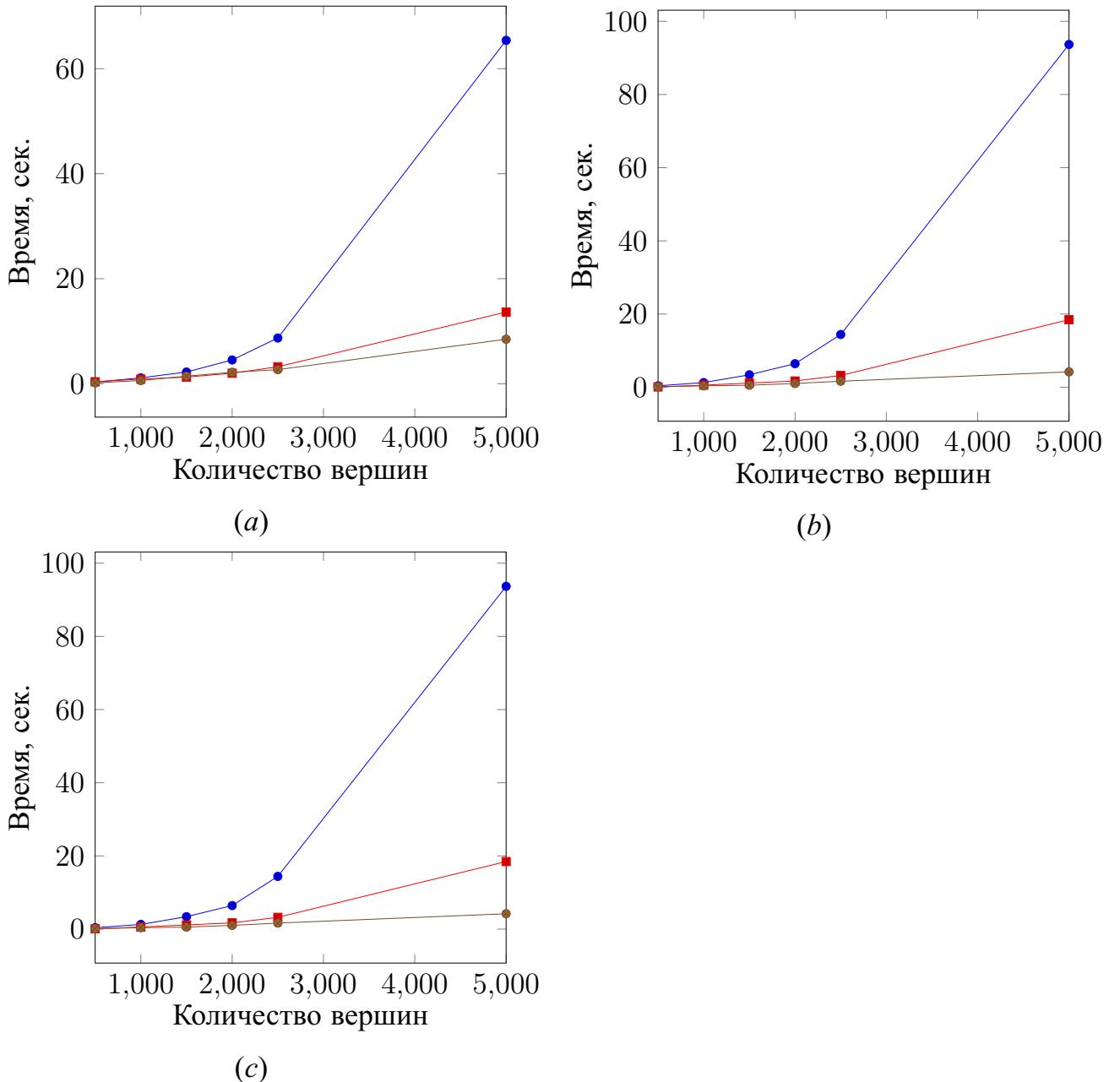


Рисунок 7 – Графики зависимости времени работы от размеров графа (синий цвет – тривиальный $O(nm + n^2)$ алгоритм, красный – алгоритм с улучшенной асимптотикой $O(m\sqrt{n})$, коричневый – генетический алгоритм): (a) – модель Эрдеша-Ренъи $p = 1\%$, (b) – модель Барабаши-Альберта $m = 2$, (c) – геометрический случайный граф $r = 0.1$

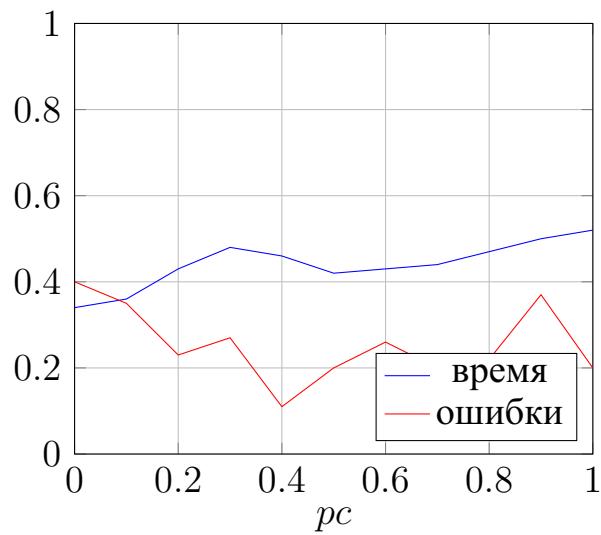


Рисунок 8 – Нормализованные значения времени выполнения алгоритма и процента ошибок с параметрами $pm = 0.4$, $|V| = 2500$, $|E| = 90268$, $N = 20$

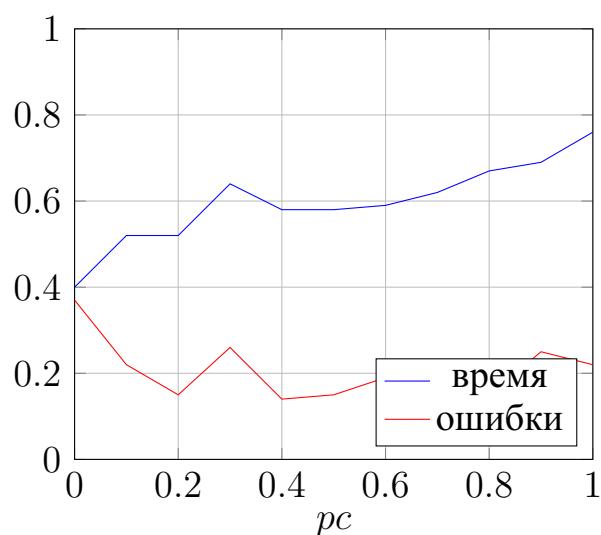


Рисунок 9 – Нормализованные значения времени выполнения алгоритма и процента ошибок с параметрами $pm = 0.6$, $|V| = 2500$, $|E| = 90268$, $N = 20$

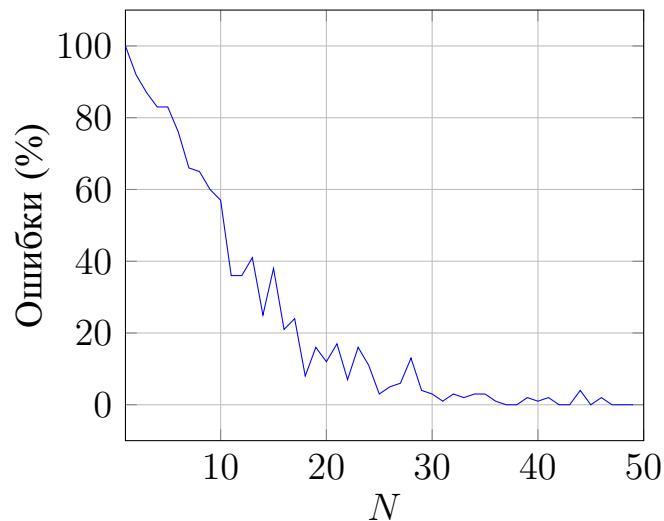


Рисунок 10 – Зависимость процента ошибок (%) от размеров популяции N , с параметрами $p_m = 0.4, p_c = 0.4$

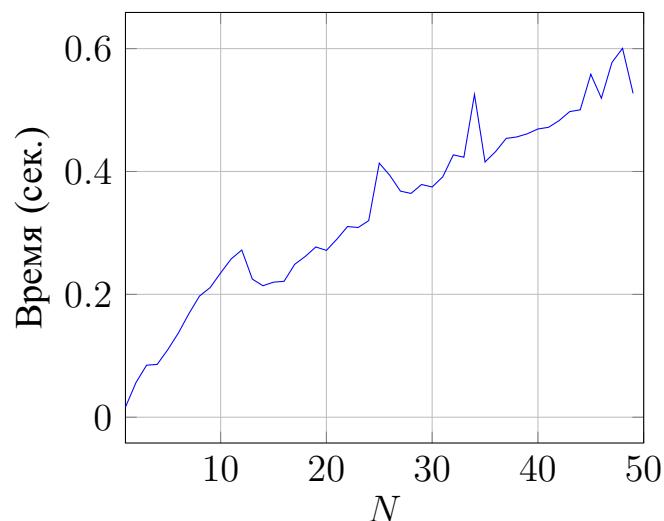


Рисунок 11 – Зависимость времени работы (сек.) от размеров популяции N , с параметрами $p_m = 0.4, p_c = 0.4$

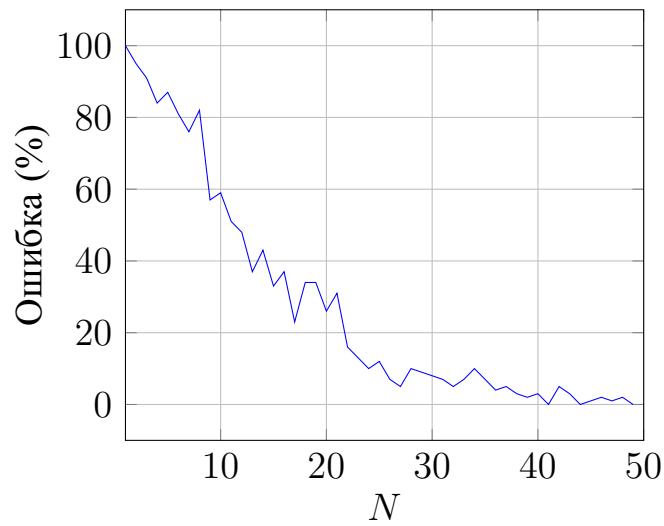


Рисунок 12 – Зависимость процента ошибок (%) от размеров популяции N , с параметрами $p_m = 0.2, p_c = 0.3, |V| = 5000, |E| = 358553$

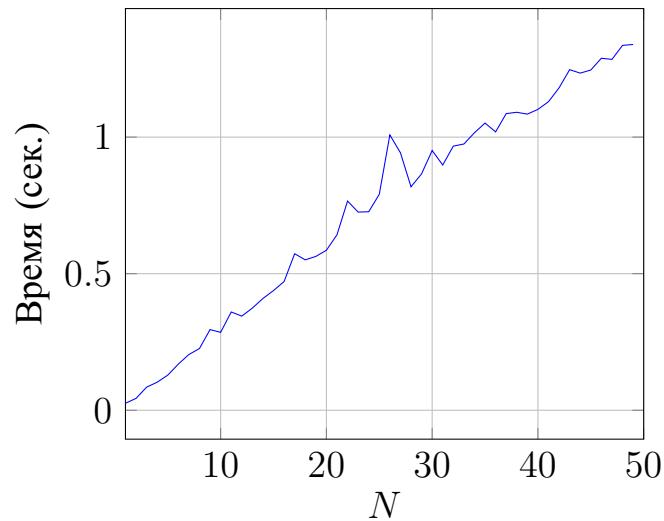


Рисунок 13 – Зависимость времени работы (сек.) от размеров популяции N , с параметрами $p_m = 0.2, p_c = 0.3, |V| = 5000, |E| = 358553$

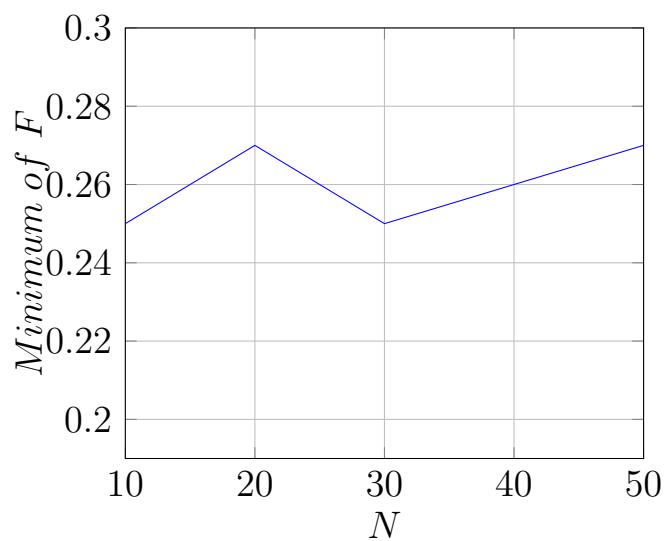


Рисунок 14 – Зависимость минимума функции F с параметрами ($\alpha = 0.5, \beta = 0.5$) с N

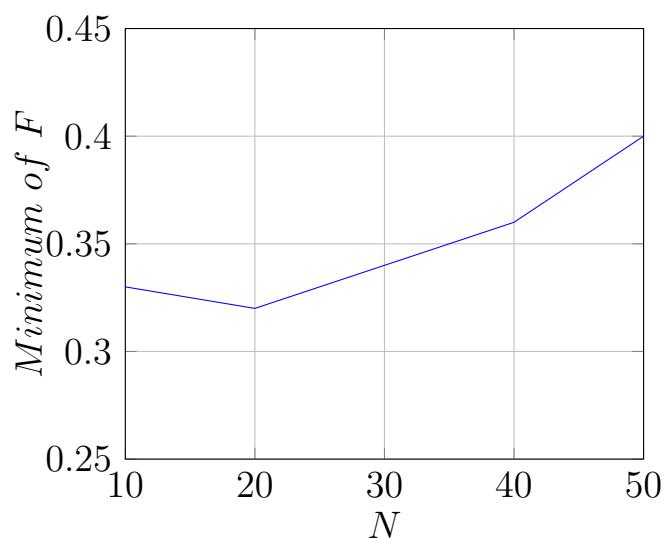


Рисунок 15 – Зависимость минимума функции F с параметрами $\alpha = 0.7\beta = 0.3$ от размера популяции N

3 Описание приложения, созданного для работы с генетическим алгоритмом

Кроме этого для работы с генетическим алгоритмом создано отдельное приложение. Приложение представляет собой веб-сайт, через который предоставляется возможность для работы с генетическим алгоритмом. С одной стороны у пользователя есть доступ к исследованию алгоритма и его запуску с различными параметрами, а с другой пользователь может заняться исследованием собственного графа и запустить алгоритм на нем.

При входе в приложение пользователь попадает на главную страницу см. рисунок 16.



Рисунок 16 – Главная страница

На главной странице пользователь видит небольшое описание алгоритма и, главным образом, навигационную панель, которая находится наверху страницы. На этой навигационной панели находятся ссылки на страницу с формой для поиска центральных вершин, страницу с формой для запуска алгоритма с различными параметрами и на различных графах, а так же ссылка для входа зарегистрированных пользователей.

После перехода на страницу входа (см. рисунок 17) пользователь может ввести свои логин и пароль и, после успешной аутентификации он будет перенаправлен на главную страницу, при этом в навигационной панели появятся ссылки на страницу добавления новых пользователей, новых графов, а так же будет отображаться его логин (см. рисунок 18)

При переходе на страницу, с которой возможно загрузить граф для поиска радиуса, пользователю показывается форма, через которую загружается граф (см. рисунок 19).



Вход в аккаунт

Логин

Пароль

Рисунок 17 – Страница входа

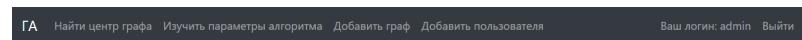


Рисунок 18 – Навигационная панель после входа в систему

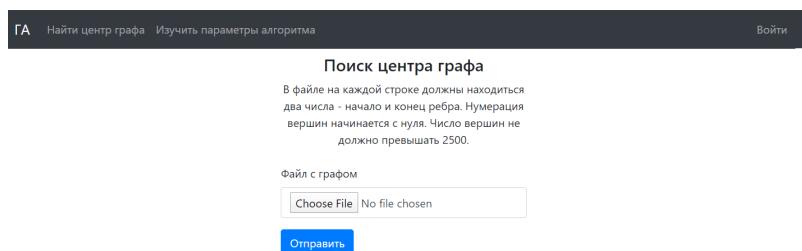


Рисунок 19 – Форма для загрузки графа с целью поиска радиуса

Похожая страница показывается пользователю при загрузке нового графа на сервер (см. рисунок 20).

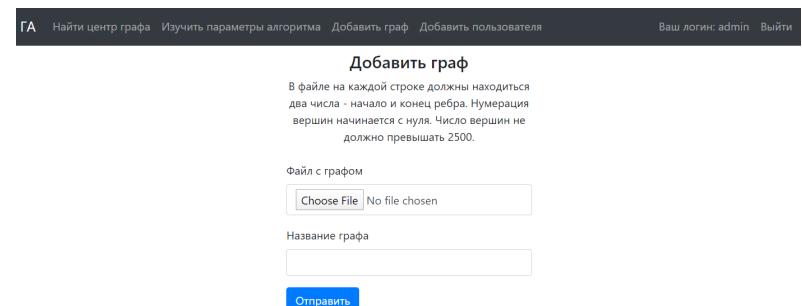


Рисунок 20 – Страница для добавления графа

Если пользователь не загрузит граф или загрузит файл в неверном формате, то получит страницу с сообщением об ошибке (см. рисунок 21, 22).



Рисунок 21 – Страница с сообщением об ошибке

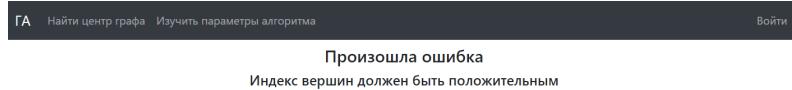


Рисунок 22 – Страница с сообщением об ошибке

При верно загруженном файле пользователю открывается страница, на которой отображаются результаты запуска алгоритма: время его работы, полученный радиус графа и возможные центральные вершины (см. рисунок 23). Кроме этого у пользователя есть возможность запускать генетический



Рисунок 23 – Страница с результатами поиска центральных вершин

алгоритм с различными параметрами (см. рисунок 24). На форме находятся

Рисунок 24 – Форма для исследования параметров алгоритма

несколько ползунков, через которые можно выставить основные параметры генетического алгоритма, а также там находится выпадающий список, в котором выбирается один из сохраненных графов. После того как пользователь выбрал соответствующие параметры и отправил форму на сервер с выбранными параметрами, запускается алгоритм, причем несколько раз, для того, чтобы получить эмпирическую оценку времени работы и процента неверных ответов. Именно эти результаты показываются на форме, которая открывается пользователю после работы алгоритма (см. рисунок 25). Каждому зарегистрированному пользователю предоставляется возможность для добавления новых пользователей через соответствующую форму (см. рисунок 26). К данным, которые вводит пользователь, выдвигаются следующие требования: длина пароля и логина должна быть от 5 до 50 символов, и пользователь с таким же



Рисунок 25 – Форма с результатами исследования параметров алгоритма

логином не должен уже существовать в базе данных. Если эти условия не будут выполнены, пользователь получит соответствующее сообщение об ошибке (см. рисунок 27).

Рисунок 26 – Форма для регистрации нового пользователя

Рисунок 27 – Форма с ошибкой при неверных данных для регистрации пользователя

3.1 Описание технологий и архитектуры приложения

В качестве языка программирования для создания проекта был выбран объектно-ориентированный язык C#. Вместе с тем, для создания клиент-серверного приложения был использован фреймворк ASP .NET MVC 5 [16, 17], который позволяет создавать веб-приложения с использованием архитектуры MVC. Кроме этого в качестве системы объектно-реляционного отображения используется технология Entity Framework 6 [18]. При этом приложение разделено на три слоя абстракции – уровень доступа к данным, уровень бизнес-логики и уровень визуального представления.

3.2 Структура базы данных

Для создания базы данных используется технология Entity Framework 6, который позволяет использовать подход Code-first, согласно которому были созданы классы Graph, GraphInfo, Edge, User, описывающие модели данных:

```
1 public class GraphInfo
2 {
3     public int Id { get; set; }
4     public int N { get; set; }
5     public int M { get; set; }
6     public string Name { get; set; }
7     public int R { get; set; }
8 }

1 public class Graph : GraphInfo
2 {
3     public ICollection<Edge> Edges { get; set; }
4     public Graph()
5     {
6         Edges = new List<Edge>();
7     }
8 }

1 public class Edge
2 {
3     public int Id { get; set; }
4     public int V1 { get; set; }
5     public int V2 { get; set; }
6 }
```

После чего фреймворк на основании созданных моделей создал структуру базы данных и связи между таблицами.

Для хранения графов в базе данных были созданы следующие таблицы: Graphs и Edges. Таблица Graphs содержит следующие поля:

- поле Id (типа данных INT) — уникальный идентификатор, внутренний ключ,
- поле N (типа данных INT) — количество вершин в графе,
- поле M (типа данных INT) — количество ребер в графе,
- поле Name (типа данных NVARCHAR) — название графа,
- поле R (типа данных INT) — радиус графа.

Таблица Edges состоит из следующих полей:

- поле Id (типа данных INT) — уникальный идентификатор, внутренний ключ,
- поле V1 (типа данных INT) — одна из вершин, которые соединяет ребро,
- поле V2 (типа данных INT) — вторая из вершин, которые соединяет ребро,
- поле Graph_Id (типа данных INT) — Id графа, которому принадлежит ребро, внешний ключ.

Кроме этого для хранения зарегистрированных пользователей существует таблица Users:

- поле Id (типа данных INT) — уникальный идентификатор, внутренний ключ,
- поле Login (типа данных NVARCHAR) — логин пользователя,
- поле Password (типа данных VARBINARY) — захешированный пароль пользователя.

3.3 Уровень доступа к данным

Для гибкой и стандартизированной работы с базой данных был создан ряд интерфейсов, в которые были вынесены основные методы для доступа к данным и их изменениям. Классы, реализующие эти интерфейсы представляют собой уровень доступа к данным, при этом использование интерфейсов позволяет с легкостью изменять реализацию этих классов, а также упрощает процесс тестирования.

Далее приводится код основных интерфейсов, которые используются при работе с данными:

```
1 public interface IGraphDao
2 {
3     IEnumerable<GraphInfo> GetAllGraphInfo();
4     Graph GetById(int id);
5     Graph Add(Graph graph);
6 }

1 public interface IUserDao
2 {
3     User GetById(int id);
4     User GetByName(string name);
5     User Add(User user);
6 }
```

Вместе с тем для использования технологии Entity Framework созданы классы `GraphContext` и `UserContext`, которые наследуются от класса `System.Data.Entity.DbContext`, что позволяет получить возможность для легкого доступа к базе данных без написания SQL запросов. Классы `GraphContext` и `UserContext` содержат в себе поля типа `DbSet<Graph>` и `DbSet<User>`, через которые происходит добавление, чтение или изменение данных в базе данных. Кроме этого в этих классах описан статический конструктор, внутри которого указан способ начальной инициализации базы данных, за счет классов `UserContextInitializer` и `GraphContextInitializer`. Эти классы наследуются от класса `CreateDatabaseIfNotExists`, что позволяет фреймворку выполнить начальное заполнение данными, если база данных еще не существует, за счет кода, который описан в переопределенном методе `Seed`. Внутри метода, описанного в классе `GraphContextInitializer`, происходит чтение нескольких созданных графов из текстовых файлов, в методе с этим же именем в классе `UserContextInitializer` добавляется пользователь с логином `admin` и паролем `admin`. Полный код представлен в приложении Г.

3.4 Уровень бизнес-логики

Уровень бизнес-логики представляет собой похожую структуру, как и уровень доступа данных — так же созданы ряд интерфейсов и классы, которые их реализуют. При этом многие из этих интерфейсов похожи на те, которые описаны в уровне доступа к данным, однако именно на этом уровне происходит запуск генетического алгоритма с различными параметрами и анализ загруженных графов. С определением интерфейсов и классов, реализующих бизнес-логику приложения можно ознакомиться в приложении Д. Классы `GraphBL` и `UserBL`, реализуют интерфейсы `IGraphBL` и `IUserBL`. Они содержат ссылки на объекты, реализующие интерфейсы `IGraphDao` и `IUserDao`. При добавлении нового пользователя происходит проверка на существование записи с таким же логином, а кроме этого происходит хеширование пароля.

При добавлении нового графа в базу данных в классе, который отвечает за работу с моделью графа, происходит проверка графа на связность и проверка на размеры графа. При неудачном прохождении проверки выбрасывается исключение, которое отлавливается на уровне представления.

Кроме этого для работы с генетическим алгоритмом создан интерфейс `IAlgorithm`:

```
1 public interface IAlgorithm
2 {
3     FindingVertexResponse FindCentralVertex(Graph graph);
4     ResearchAlgorithmResponse ResearchAlgorithm(ResearchRequest param);
5 }
```

Интерфейс определяет набор методов, в которых будет реализована логика для работы с генетическим алгоритмом. Вместе с тем класс Algorithm реализует данный интерфейс и в нем содержится вся логика работы с алгоритмом — получение результатов поиска центральных вершин, замеры времени работы и процента неверно найденных решений.

Результаты измерений возвращаются из методов при помощи классов FindingVertexResponse и ResearchAlgorithmResponse:

```
1 public class FindingVertexResponse
2 {
3     public int[] Center { get; set; }
4     public int R { get; set; }
5     public double Time { get; set; }
6 }

1 public class ResearchAlgorithmResponse
2 {
3     public double AvgTime { get; set; }
4     public double Error { get; set; }
5 }
```

Полный код классов уровня бизнес-логики можно увидеть в приложении [Д](#).

3.5 Уровень представления

Так как проект представляет собой веб-приложение, то уровень, отвечающий за пользовательский интерфейс реализован при помощи технологии ASP .NET MVC 5. В связи с чем весь код этого уровня разделен на:

- контроллеры, которые отвечают на HTTP запросы клиента с помощью представлений,
- представления, которые написаны с использованием технологии Razor, позволяющей внедрять серверный C# код,
- модели данных, внутри которых происходит передача данных от клиента серверу и обратно.

3.5.1 Контроллеры

Для взаимодействия с клиентской частью приложения и обработки пользовательских данных было создано несколько контроллеров: `HomeController`, `GraphController`, `LoginController`, `ResearchController`.

Класс `HomeController` содержит один метод `Index`, который отвечает на GET-запрос и возвращает домашнюю страницу.

Класс `GraphController` включает в себя методы, определяющие URL-адреса при взаимодействии с которыми клиентской части приложения предоставляется возможность запускать генетический алгоритм для поиска центральных вершин или добавлять новый граф в базу данных. В целом данный контроллер ответственен за обработку следующих запросов:

- `/Graph` — обрабатывает GET-запрос и возвращает загрузочную страницу для поиска центральных вершин,
- `/Graph/FindCentralVertex` — обрабатывает POST-запрос, в котором передается файл с графиком, после чего запускается генетический алгоритм. В качестве результата возвращается страница с сообщением об ошибке, которая могла произойти при обработке запроса из-за неверного формата данных в файле с графиком, или слишком большого размера загружаемого графа. При успешном запуске возвращает страницу с результатами работы — радиус графа и возможные центральные вершины,
- `/Graph/Add` — обрабатывает GET-запрос, который возвращает форму для загрузки графа в базу данных,
- `/Graph/Add` — обрабатывает POST-запрос, в котором пользователь передает на сервер для сохранения файл с графиком и название графа. При успешном добавлении перенаправляет пользователя на домашнюю страницу, при возможной ошибке — на страницу с описанием ошибки.

Класс `ResearchController` содержит методы, через которые пользователю предоставляется возможность запускать генетический алгоритм с различными параметрами (p_c, p_m, N), а также график на котором будет тестироваться алгоритм. Контроллер содержит следующие методы:

- `/Research/Index` — обрабатывает GET-запрос и возвращает форму с выбором параметров для генетического алгоритма, при этом в результат подгружаются описания графов, сохраненных в базе данных,
- `/Research/ResearchAlgorithm` — обрабатывает POST-запрос, в который

передаются выбранные параметры алгоритма и выбранный граф.

В классе `LoginController` определены методы, за счет которых происходит регистрация и аутентификация пользователей. Пользователь, который вошел в систему получает возможность для добавления новых графов и добавление новых пользователей, залогиненному пользователю предоставляются права администратора. В контроллере `LoginController` реализованы следующие методы:

- `/Login/Index` — обрабатывает GET-запрос и возвращает форму заполнения для входа в систему,
- `/Research/SignIn` — обрабатывает POST-запрос, в который передается логин и пароль. В этом методе происходит валидация введенных данных, проверка принадлежности пароля введенному пользователю и при успешном прохождении этих этапов, пользователю отправляется набор cookie-данных, вследствие чего происходит аутентификация пользователя. При неверном логине или пароле пользователю возвращается сообщение об ошибке, при успешном прохождении аутентификации — метод возвращает переадресацию на домашнюю страницу,
- `/Login/SignUp` — обрабатывает GET-запрос и возвращает форму для регистрации нового пользователя,
- `/Login/SignUp` — обрабатывает POST-запрос и добавляет нового пользователя при успешном прохождении валидации и отсутствии пользователя с таким же логином.

Полный код контроллеров можно увидеть в приложении [E](#).

3.5.2 Модели данных и валидация

Основными классами, с помощью объектов которых происходит передача данным в контроллеры, `AddGraphRequest`, `CreateUserRequest`, `LoginUserRequest`, `ResearchRequest`, при этом результаты вычислений возвращаются из контроллеров в виде представлений, которые представляют собой HTML разметку с внедрением данных, переданных через классы `AlgorithmResultResponse`, `FindingVertexResponse`, `ResearchAlgorithmResponse`.

При этом очевидно, что при введенные пользовательские данные должны удовлетворять некоторым условиям. Для того, чтобы передаваемые данные можно было проверить из любого участка кода для некоторых свойств были

использованы атрибуты валидации [19] Required, Compare, StringLength:

```
1 public class CreateUserRequest
2 {
3     [Required(ErrorMessage = "Введите логин")]
4     [StringLength(50, MinimumLength = 3,
5         ErrorMessage = "Длина логина должна быть от 3 до 50 символов")]
6     public string Login { get; set; }
7
8     [Required(ErrorMessage = "Введите пароль")]
9     [StringLength(50, MinimumLength = 3,
10        ErrorMessage = "Длина логина должна быть от 3 до 50 символов")]
11    public string Password { get; set; }
12
13    [Required(ErrorMessage = "Повторите пароль")]
14    [Compare("Password", ErrorMessage = "Пароли не совпадают")]
15    public string ConfirmPassword { get; set; }
16 }
```



```
1 public class LoginUserRequest
2 {
3     [Required(ErrorMessage = "Введите логин")]
4     public string Login { get; set; }
5
6     [Required(ErrorMessage = "Введите пароль")]
7     public string Password { get; set; }
8 }
```

При таком использовании атрибутов валидации проверить модель на соответствие выдвинутым требованиям можно при помощи следующего кода:

```
1 if (ModelState.IsValid) {
2     ...
3 }
```

внутри любого из контроллеров, где ModelState — свойство класса Controller, которое инкапсулирует состояние модели, переданной в качестве параметра запроса. В случае неудачного прохождения валидации в свойство ModelState при помощи метода AddModelError добавляется сообщение об ошибке, которое затем будет вставлено в HTML разметку. Атрибут Required установлен для логина и пароля, вводимого пользователем, что гарантирует тот факт,

что в базу данных не будет помещена запись с пустыми полями. В добавок к этому у свойства `ConfirmPassword` установлен атрибут `Compare`, который требует, чтобы свойство, отвечающее за хранение пароля, совпадало со свойством, отвечающим за хранение повтора пароля. Также используется атрибут `StringLength`, в котором устанавливаются минимальная и максимальная длина логина и пароля.

3.5.3 Аутентификация

Как уже отмечалось ранее доступ к возможности добавлять графы в базу данных и добавлять туда же новых пользователей имеют доступ только пользователи, которые вошли в систему. В связи с этим в качестве технологии аутентификации в созданном приложении используется аутентификация с помощью форм [20]. Для ее включения в файл `Web.config` были добавлены следующие строки:

```
1 <authentication mode="Forms">
2     <forms loginUrl="~/login" timeout="60" />
3 </authentication>
```

в которых указывается по какому адресу будет отправлен пользователь в случае, если он не имеет прав доступа к запрошенным ресурсам и время действия cookie-файлов. При успешном прохождении проверки на принадлежность пользователю введенного им пароля, при помощи следующей строки кода клиентская часть получает cookie-файлы, которые затем будут присоединяться ко всем остальным запросам:

```
1 FormsAuthentication.SetAuthCookie(user.Login, true);
```

Для того, чтобы к определенным методам был доступ только авторизированным пользователям к каждому методу применяется атрибут `Authorize`, который гарантирует проверку на доступность для пользователя этих методов. При этом для пользователя вошедшего в систему несколько изменяется HTML разметка, что достигается при помощи использования свойства `User.Identity.IsAuthenticated`.

3.6 Хеширование паролей

Очевидно, что хранение паролей пользователей в открытом виде представляет собой подход нарушающий основные требования к безопасности

приложения. В связи с чем каждый пароль при регистрации пользователя хешируется и полученный хеш сохраняется в базе данных. Хеширование паролей происходит в классе `Encryption` (см. приложение [Д](#)), где определены публичные методы `CreatePassword` и `CheckPassword`. Создание хеша пароля происходит при помощи объекта класса `Rfc2898DeriveBytes` [21]:

```
1 var pbkdf2 = new Rfc2898DeriveBytes(password, salt, iterations);
2 byte[] hash = pbkdf2.GetBytes(hashSize);
```

В конструктор класса передается строка с паролем, «соль» — псевдослучайная последовательность байт, которая используется для повышения криптоустойчивости хеша и параметр, отвечающий за искусственную временную задержку, которая позволяет избежать попытки грубого перебора. В базу данных сохраняется полученный хеш и сгенерированная «соль». При проверке подлинности пароля из базы данных извлекается хеш с «солью», после чего введенный пароль хешируется с сохраненной «солью» и результат сравнивается с тем, что было сохранено в базе данных.

3.7 Внедрение зависимостей

Ранее описывались независимые уровни, на которые разделено приложение, при этом гибкость и заменяемость каждого из уровней гарантируется существованием интерфейсов. Каждый из уровней содержит ссылки на объекты, которые реализуют тот или иной интерфейс, при этом эти объекты передаются в качестве параметров в конструкторы. Для того, чтобы гарантировать тот факт, что во все конструкторы будут переданы одни и те же реализации интерфейсов и избежать дублирования кода в созданном приложении используется IoC-контейнер `Ninject`, который связывает интерфейсы с объектами, которые их реализуют и предоставляет их при необходимости. Связывание интерфейсов и реализаций происходит в методе класса `NinjectRegistrations`:

```
1 public class NinjectRegistrations : NinjectModule
2 {
3     public override void Load()
4     {
5         Bind<IUserDao>().To<UserDao>();
6         Bind<IGraphDao>().To<GraphDao>();
7         Bind<IAlgortithm>().To<Algorithm>();
8         Bind<IGraphBL>().To<GraphBL>();
```

```
9     Bind<IUserBL>().To<UserBL>();  
10    }  
11 }
```

При этом в глобальном файле запуска приложения происходит регистрация этого класса в качестве основного способа разрешения зависимостей.

С полноценным кодом всей работы можно ознакомиться на CD диске [Ж](#).

ЗАКЛЮЧЕНИЕ

В рамках работы поставленная цель была достигнута. Для поиска центральных вершин был предложен генетический алгоритм, а кроме этого созданы программные приложения позволившие его исследовать. Исходя из полученных результатов, можно сказать, что во многом время и качество работы алгоритма зависит от параметров p_m , p_c и N , и при правильно подобранных значениях алгоритм не уступает по своим характеристикам разработанным ранее алгоритмам. Кроме этого создано веб-приложение для работы с генетическим алгоритмом.

Результаты исследований были представлены в статье [22], на Студенческих научных конференциях факультета КНиИТ СГУ в 2019 и 2020 году, а также на VIII Международной молодежной научно-практической конференции «Математическое и компьютерное моделирование в экономике, страховании и управлении рисками» на базе СГУ.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 Network Analysis / Ed. by Ulrik Brandes, Thomas Erlebach. — Springer Berlin Heidelberg, 2005.
- 2 *Watts, Duncan J.* Collective dynamics of ‘small-world’ networks / Duncan J. Watts, Steven H. Strogatz // *Nature*. — jun 1998. — Vol. 393, no. 6684. — Pp. 440–442.
- 3 *Кормен, Т.* Алгоритмы: построение и анализ / Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. — Москва: Вильямс, 2019.
- 4 *Chan, Timothy M.* All-pairs shortest paths for unweighted undirected graphs in $o(mn)$ time / Timothy M. Chan // *ACM Trans. Algorithms*. — oct 2012. — Vol. 8, no. 4. — Pp. 34:1–34:17.
- 5 *Berman, Piotr.* Faster approximation of distances in graphs // Algorithms and Data Structures / Ed. by Frank Dehne, Jörg-Rüdiger Sack, Norbert Zeh. — Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.
- 6 *Roditty, Liam.* Fast approximation algorithms for the diameter and radius of sparse graphs // Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing. — STOC ’13. — New York, NY, USA: ACM, 2013. — Pp. 515–524.
- 7 *Aingworth, D.* Fast estimation of diameter and shortest paths (without matrix multiplication) / D. Aingworth, C. Chekuri, P. Indyk, R. Motwani // *SIAM J. Comput.* — 1999. — Vol. 28, no. 1. — Pp. 1167–1181.
- 8 *Seidel, R.* On the all-pairs-shortest-path problem in unweighted undirected graphs / R. Seidel // *J. Comput. Syst. Sci.* — 1995. — Vol. 51, no. 3. — Pp. 400–403.
- 9 *Strassen, Volker.* Gaussian elimination is not optimal / Volker Strassen // *Numerische Mathematik*. — Aug 1969. — Vol. 13, no. 4. — Pp. 354–356.
- 10 *Holland, J.* Adaption in Natural and Artificial Systems Adaption in Natural and Artificial Systems / J. Holland. — University of Michigan Press, 1975.
- 11 *Панченко, Т. В.* Генетические алгоритмы / Т. В. Панченко. — Астрахань: Издательский дом «Астраханский университет», 2007.

- 12 *Alkhalifah, Y.* A genetic algorithm applied to graph problems involving subsets of vertices // Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No.04TH8753). — Vol. 1. — June 2004. — Pp. 303–308.
- 13 *Erdös, P.* On random graphs I / P. Erdös, A. Rényi // *Publicationes Mathematicae Debrecen*. — 1959. — Vol. 6. — P. 290.
- 14 *Albert, Reka.* Statistical mechanics of complex networks / Reka Albert, Albert-Laszlo Barabasi // *Reviews of Modern Physics*. — 2002. — Vol. 74, no. 1. — Pp. 47–97.
- 15 *Gilbert, Edgar.* Random plane networks / Edgar Gilbert // *Journal of the Society for Industrial and Applied Mathematics*. — 1961. — Vol. 9, no. 4. — Pp. 533–543.
- 16 *Фримен, А.* ASP.NET MVC 5 Framework с примерами на C# для профессионалов / А. Фримен. — Вильямс, 2015.
- 17 Начало работы с ASP.NET MVC 5 [Электронный ресурс]. — URL: <https://docs.microsoft.com/ru-ru/aspnet/mvc/overview/getting-started/introduction/getting-started> (Дата обращения 24.05.2020). Загл. с экрана. Яз. рус.
- 18 Обзор Entity Framework 6 [Электронный ресурс]. — URL: <https://docs.microsoft.com/ru-ru/ef/ef6/> (Дата обращения 24.05.2020). Загл. с экрана. Яз. рус.
- 19 System.ComponentModel.DataAnnotations Пространство имен [Электронный ресурс]. — URL: <https://docs.microsoft.com/ru-ru/dotnet/api/system.componentmodel.dataannotations?view=netcore-3.1> (Дата обращения 24.05.2020). Загл. с экрана. Яз. рус.
- 20 Класс FormsAuthentication [Электронный ресурс]. — URL: <https://docs.microsoft.com/ru-ru/dotnet/api/system.web.security.formsauthentication?view=netframework-4.8> (Дата обращения 24.05.2020). Загл. с экрана. Яз. рус.
- 21 Rfc2898DeriveBytes Класс [Электронный ресурс]. — URL: <https://docs.microsoft.com/ru-ru/dotnet/api/system.security.cryptography.rfc2898derivebytes?view=netcore-3.1> (Дата обращения 24.05.2020). Загл. с экрана. Яз. рус.

- 22 Vlasov, Andrew. Ball-shrinking genetic search algorithm for finding central vertices in graphs // Fourth Workshop on Computer Modelling in Decision Making (CMDM 2019).— Atlantis Press, 2019.— Pp. 94–97. <https://doi.org/10.2991/ahcs.k.191206.017>.

ПРИЛОЖЕНИЕ А

Код генетического алгоритма

Далее приводится код класса GeneticAlgorithm:

```
1 #pragma once
2
3 #include <vector>
4 #include <algorithm>
5 #include <set>
6 #include <ctime>
7
8 #include "GraphWork.h"
9 #include "Random.h"
10 #include "Strassen.h"
11 #include "VectorOutput.h"
12
13 using std :: begin;
14 using std :: end;
15 using std :: max_element;
16 using std :: min_element;
17 using std :: pair ;
18 using std :: vector ;
19 using std :: set ;
20
21 // Класс для работы с генетическим алгоритмом
22 class GeneticAlgorithm {
23 private :
24     // Ссылка на граф
25     Graph* G;
26     // Размер популяции
27     int populationSize ;
28     // Параметры, отвечающие за мутацию и скрещивание
29     double mutationP, crossP;
30     // Вектор с набором вершин в популяции
31     vector<int> population ;
32     // Модуль декомпьютер для работы с генерацией псевдо случайных величин
33     Random rd;
34     // Метод отвечающий за старт алгоритма,
35     // принимает ссылку на переменную для записи времени работы алгоритма.
36     void startAlgorithm (double& time) {
37         // число итераций
38         int stepN = 20;
```

```

39 // начало измерения времени работы алгоритма
40 double start = clock();
41 // итерационное выполнение алгоритма
42 for (int i = 0; i < stepN; i++) {
43     this->evolutionStep();
44 }
45 // окончание измерения времени
46 double finish = clock();
47 // вычисление времени работы всего алгоритма
48 time = (finish - start) / CLOCKS_PER_SEC;
49 }
50
51 public:
52 // Конструктор класса, принимает ссылку на граф, размер популяции,
53 // вероятность скрещивания и вероятность мутации
54 GeneticAlgorithm(Graph* g, int popSize, double pC, double pM) {
55     // инициализация внутренних полей класса
56     this->G = g;
57     this->populationSize = popSize;
58     this->crossP = pC;
59     this->mutationP = pM;
60     // Генерация случайной начальной популяции
61     for (int i = 0; i < popSize; i++) {
62         population.push_back(rand() % g->Size());
63     }
64 }
65
66 // Этап селекции
67 void makeSelection() {
68     // Поиск эксцентриситетов всех вершин в популяции
69     vector<int> e;
70     for (int i = 0; i < population.size(); i++) {
71         e.push_back(G->getEccentricity(population[i]));
72     }
73     // вычисление вероятности для попадания в следующее поколение
74     // вычисление ее таким образом, чтобы вероятность была выше
75     // у вершин с меньшим эксцентриситетом
76     vector<double> probability;
77     double maxV = *max_element(e.begin(), e.end()), leftSum = 0;
78     for (int i = 0; i < e.size(); i++) {
79         leftSum += maxV / double(e[i]);

```

```

80 }
81 double x = 1.0 / leftSum;
82 for (int i = 0; i < e. size (); i++) {
83     probability .push_back(maxV / double(e[i]) * x);
84 }
85 // этап селекции на основе вычисленных вероятностей
86 vector<int> nextPopulation ;
87 for (int i = 0; i < this ->populationSize; i++) {
88     nextPopulation .push_back(rd.choice(population , probability ));
89 }
90 this ->population = nextPopulation ;
91 }
92 // этап скрецивания
93 void crossing () {
94     // вектор для результата скрецивания
95     vector<int> crossed ;
96     for (int i = 0; i < populationSize ; i++) {
97         // скрецивание происходит с заданной вероятностью
98         if (rd.getRandomLowerOne() < crossP) {
99             // выбор индексов вершин из популяции
100            int ind1 = rd. randint (0, populationSize - 1);
101            int ind2 = rd. randint (0, populationSize - 1);
102            // получение самих вершин
103            int u = population [ind1 ], v = population [ind2 ];
104            // поиск кратчайших путей между вершинами
105            vector<int> path = G->getPath(u, v);
106            // выбор вершины из середины пути
107            crossed .push_back(path[path. size () / 2]);
108        }
109    }
110    // добавление потомков в популяцию
111    this ->population. insert ( population .end(), crossed .begin(), crossed .end())
112    ;
113 }
114 // процесс мутации
115 void mutation () {
116     for (unsigned int i = 0; i < this ->population.size () ; i++) {
117         // с учетом вероятности мутации
118         if (rd.getRandomLowerOne() < mutationP) {
119             // поиск соседей
120             vector<int> n = G->getNeighbour(population[i]);

```

```

120                                // выбор одного из соседей
121                                population[ i ] = rd.choice(n);
122                            }
123                        }
124                    }
125                    // вывод популяции
126                    void printPopulation () {
127                        for (int i = 0; i < this->population.size(); i++) {
128                            cout << population[ i ] << " ";
129                        }
130                        cout << endl;
131                    }
132                    // метод для эволюционной итерации
133                    void evolutionStep () {
134                        // скрещивание
135                        this->crossing();
136                        // селекция
137                        this->makeSelection();
138                        // мутация
139                        this->mutation();
140                    }
141                    // поиск вершины с минимальным эксцентризитетом после работы алгоритма
142                    int getBestResult (double& time) {
143                        // запуск генетического алгоритма
144                        this->startAlgorithm(time);
145                        // вектор для хранения эксцентризитетов
146                        vector<int> e;
147                        // поиск эксцентризитетов для всех вершин в популяции
148                        for (int i = 0; i < population.size(); i++) {
149                            int x = G->getEccentricity( population[ i ] );
150                            e.push_back(x);
151                        }
152                        // возвращение индексов вершин эксцентризитетов
153                        vector<int> ind = this->getCentralVertex(e);
154                        // возвращение первой из найденной оптимальной вершины
155                        return this->G->getEccentricity(population[ind[0]]);
156                    }
157                    // получение популяции
158                    vector<int> getPopulation () {
159                        return this->population;
160                    }

```

```
161 // поиск индексов вершин с минимальным эксцентризитетом
162 vector<int> getCentralVertex ( vector<int> e) {
163     // минимальный эксцентризитет
164     int minV = *min_element(e.begin(), e.end());
165     // выделение индексов минимальных эксцентризитетов
166     vector<int> res ;
167     for (int i = 0; i < e.size () ; i++) {
168         if (e[i] == minV) {
169             res.push_back(i);
170         }
171     }
172     return res;
173 }
174 };
```

ПРИЛОЖЕНИЕ Б

Код классов для работы с графами

Далее приводится код класса GraphWork:

```
1 #pragma once
2
3 #include <vector>
4 #include <queue>
5 #include <iostream>
6 #include <algorithm>
7 #include <stdio.h>
8
9 using std :: vector ;
10 using std :: queue;
11 using std :: pair ;
12 using std :: exception ;
13 using std :: string ;
14 using std :: to_string ;
15 using std :: cin ;
16 using std :: cout ;
17 using std :: endl ;
18
19 using std :: max;
20
21 class Graph {
22 private :
23     int N, M;
24     // матрица смежности
25     vector<vector<int>> adjacency;
26     // матрица для расстояний между вершинами
27     vector<vector<int>> distance ;
28     // найденные расстояния между вершинами
29     vector<vector<vector<int>>> path;
30     // флаги на найденные эксцентрикитеты
31     vector<bool> calculatedEccentricity ;
32     // вектор с эксцентрикитетами
33     vector<int> eccentricity ;
34
35     void initializeMatrix () {
36         // начальная инициализация всех матриц
37         // матрица смежности
38         adjacency . resize ( this ->N, vector<int>());
```

```

39 // матрица n x n для расстояний
40 distance . resize ( this ->N, vector<int>( this ->N, INT_MAX));
41 // матрица n x n для путей
42 path . resize ( this ->N, vector<vector<int>>(this ->N));
43 // флаги для отображения был ли найден эксцентриситет этой вершины
44 calculatedEccentricity . resize ( this ->N, false);
45 // вектор для хранения найденных эксцентриситетов
46 eccentricity . resize ( this ->N, INT_MAX);
47 // инициализация матриц
48 for ( int i = 0; i < this ->N; i++) {
49     this ->distance[i][ i] = 0;
50     this ->path[i][ i] = vector<int>(1, i);
51 }
52 }
53
54 public :
55 Graph(int n, int m, vector<pair<int, int>> adj) {
56     this ->N = n, this->M = m;
57     initializeMatrix ();
58     // создание списка смежности для хранения для графа
59     for ( int i = 0; i < m; i++) {
60         int x = adj[ i]. first , y = adj[ i]. second;
61         this ->adjacency[x].push_back(y);
62         this ->adjacency[y].push_back(x);
63     }
64 }
65
66 // запуск обхода в ширину из переданной вершины
67 void bfsFromVertex(int x) {
68     // реализация классического алгоритма поиска в ширину
69     // очередь обхода
70     queue<int> q;
71     // добавление стартовой вершины
72     q.push(x);
73     // расстояния от стартовой вершины до всех остальных
74     vector<int> d( this ->N, INT_MAX);
75     // начальная инициализация расстояния
76     d[x] = 0;
77     // обход до тех пор, пока очередь не пуста
78     while ( !q.empty()) {
79         // извлечение из очереди

```

```

80         int u = q. front () ;
81         q.pop();
82         // получение пути от старта до рассматриваемой вершины
83         vector<int> currentPath = this ->path[x][u];
84         // просмотр всех соседних вершин
85         for (unsigned int i = 0; i < this ->adjacency[u].size () ; i++) {
86             // извлечение соседней вершины
87             int v = this ->adjacency[u][i ];
88             // если вершина не была посещена, то
89             // она добавляется в очередь
90             if (d[v] > INT_MAX / 2) {
91                 // добавление вершины в текущий путь
92                 currentPath .push_back(v);
93                 // сохранение найденного пути
94                 this ->path[x][v] = currentPath ;
95                 this ->path[v][x] = currentPath ;
96                 currentPath .pop_back();
97                 // увеличение найденного пути
98                 d[v] = d[u] + 1;
99                 // добавление вершины в очередь
100                q.push(v);
101            }
102        }
103    }
104    // сохранение найденных расстояний
105    for (unsigned int i = 0; i < d. size () ; i++) {
106        this ->distance[x][ i ] = d[i ];
107        this ->distance[i ][ x ] = d[i ];
108    }
109 }
110 // получение пути между вершинами,
111 // если путь еще неизвестен, то
112 // запускается обход в ширину
113 vector<int> getPath( int x, int y) {
114     if ( this ->path[x][y].empty()) {
115         this ->bfsFromVertex(x);
116     }
117     return this ->path[x][y];
118 }
119 // получение эксцентриситета вершины
120 int getEccentricity ( int x) {

```

```

121         if (x >= this->N) {
122             throw exception("Vertex doesn't exist in graph");
123         }
124         // проверка, что эксцентриситет был найден ранее
125         if (!this->calculatedEccentricity [x]) {
126             // запуск обхода
127             this->bfsFromVertex(x);
128             // пометка вершины, что ее эксцентриситет найден
129             this->calculatedEccentricity [x] = true ;
130             int result = -INT_MAX;
131             // поиск самой удаленной вершины
132             for (int i = 0; i < this->N; i++) {
133                 result = max(this->distance[x][i], result );
134             }
135             this->eccentricity [x] = result ;
136         }
137         return this->eccentricity [x];
138     }
139     // получение соседей заданной вершины
140     vector<int> getNeighbour(int x) {
141         return this->adjacency[x];
142     }
143     // получение размеров графа
144     int Size() {
145         return this->N;
146     }
147 };

```

Методы для чтения графов:

```

1 // метод для чтения графа из файла
2 vector<pair<int, int>> readFromFileWhereEdges(string file , int& N, int& M) {
3     // создание файлового потока
4     ifstream in( file );
5     // ребра графа
6     vector<pair<int, int>> e;
7     string inputLine;
8     int n = 0, m = 0;
9     // построчное чтение файла
10    while ( getline (in, inputLine)) {
11        if (inputLine[0] == '%') {
12            continue;
13        }

```

```

14         // увеличение счетчика ребер
15         m++;
16         stringstream s(inputLine);
17         // чтение вершин
18         int x, y;
19         s >> x >> y;
20         // увеличение размера графа
21         n = max(n, max(x, y));
22         // перевод в ноль индексацию
23         x--, y--;
24         e.push_back(make_pair(x, y));
25     }
26     N = n;
27     M = m;
28     return e;
29 }
30 // метод для аналогичного чтения
31 // файла с графиком, в котором на первой строке указаны размеры графа
32 vector<pair<int, int>> readFromNM(string file, int& N, int& M) {
33     ifstream in(file);
34     vector<pair<int, int>> e;
35     int n, m;
36     in >> n >> m;
37     for (int i = 0; i < m; i++) {
38         int x, y;
39         in >> x >> y;
40         e.push_back(make_pair(x, y));
41     }
42     N = n;
43     M = m;
44     return e;
45 }
```

Класс Repository для хранения имен файлов и радиусов графов:

```

1 class Repository {
2     private :
3         struct GraphParams
4         {
5             string fileName;
6             int realR;
7             GraphParams(string fileN, int r) {
8                 this->fileName = fileN;
```

```

9             this ->realR = r;
10            }
11        };
12        // реальные предподсчитанные радиусы графов
13        vector<int> RADIUS_BA = { 4, 4, 5, 4, 5, 5, 5 };
14        vector<int> RADIUS_GEOM = { 9, 9, 8, 8, 8, 8, 8 };
15        vector<int> RADIUS_ER = { 5, 4, 4, 4, 3, 3, 3 };
16        // Номер места графа
17        int TEST = 4;
18
19        // Названия файлов с графиками
20        vector<string> BA_FILE = {
21            "BA_Graph\\BarabasiAlbertGraph1_M2.txt",
22            "BA_Graph\\BarabasiAlbertGraph2_M2.txt",
23            "BA_Graph\\BarabasiAlbertGraph3_M2.txt",
24            "BA_Graph\\BarabasiAlbertGraph4_M2.txt",
25            "BA_Graph\\BarabasiAlbertGraph5_M2.txt",
26            "BA_Graph\\BarabasiAlbertGraph6_M2.txt",
27            "BA_Graph\\BarabasiAlbertGraph7_M2.txt" };
28
29        vector<string> GEOM_FILE = {
30            "GEOM_Graph\\GeometricGraph1_R01.txt",
31            "GEOM_Graph\\GeometricGraph2_R01.txt",
32            "GEOM_Graph\\GeometricGraph3_R01.txt",
33            "GEOM_Graph\\GeometricGraph4_R01.txt",
34            "GEOM_Graph\\GeometricGraph5_R01.txt",
35            "GEOM_Graph\\GeometricGraph6_R01.txt",
36            "GEOM_Graph\\GeometricGraph7_R01.txt" };
37
38        vector<string> ER_FILE = {
39            "ERDOSRENYI_Graph\\ErdosRenyi1_P001.txt",
40            "ERDOSRENYI_Graph\\ErdosRenyi2_P001.txt",
41            "ERDOSRENYI_Graph\\ErdosRenyi3_P001.txt",
42            "ERDOSRENYI_Graph\\ErdosRenyi4_P001.txt",
43            "ERDOSRENYI_Graph\\ErdosRenyi5_P001.txt",
44            "ERDOSRENYI_Graph\\ErdosRenyi6_P001.txt",
45            "ERDOSRENYI_Graph\\ErdosRenyi7_P001.txt" };
46        // чтение графа в зависимости от переданного параметра,
47        // отвечающего за тип этого графа
48        GraphParams getGraphData(string graphType) {
49            string fileName;

```

```

50     int real_r = 0;
51     // проверка параметра и выбор файла
52     // в зависимости от параметра
53     if (graphType == "BA_GRAPH") {
54         fileName = BA_FILE[TEST];
55         real_r = RADIUS_BA[TEST];
56         cout << "BA_GRAPH" << endl;
57     }
58     else if (graphType == "GEOM_GRAPH") {
59         fileName = GEOM_FILE[TEST];
60         real_r = RADIUS_GEOM[TEST];
61         cout << "GEOM_GRAPH" << endl;
62     }
63     else if (graphType == "ERDOSH_GRAPH") {
64         fileName = ER_FILE[TEST];
65         real_r = RADIUS_ER[TEST];
66         cout << "ERDOSH_GRAPH" << endl;
67     }
68     return GraphParams(fileName, real_r);
69 }
70
71
72 public:
73     // метод для получения графа из файла
74     GraphDescription getGraph( string graphType) {
75         // получение параметров графа
76         GraphParams params = this->getGraphData(graphType);
77         int n, m;
78         // чтение графа из файла
79         edges e = readFromFileNM(params.fileName, n, m);
80         cout << "N = " << n << " M = " << m << endl;
81         return GraphDescription(n, m, e, params.realR);
82     }
83 };

```

Классы GraphDescription и GaTestResult:

```

1 #pragma once
2 #include <vector>
3
4 using std :: vector ;
5 using std :: pair ;
6

```

```

7  using edges = vector<pair<int, int>>;
8  // Сущности, отвечающие
9  // за передачу информации о графах между методами
10 enum GraphType
11 {
12     None = 0,
13     ER = 1,
14     BA = 2,
15     GEOM = 3,
16 };
17 // описание графа в виде набора вершин,
18 // размеров и радиуса графа
19 class GraphDescription {
20 public:
21     int n, m;
22     edges e;
23     int realR;
24     GraphDescription() {}

25
26     GraphDescription(int n, int m, edges e, int realR) {
27         this->n = n;
28         this->m = m;
29         this->e = e;
30         this->realR = realR;
31     }
32 };
33
34 // описание результата вычислительного теста
35 // параметры генетического алгоритма и значение функции
36 class GaTestResult {
37 public:
38     GaTestResult() {}

39
40     GaTestResult(double popSize, double pm, double pc, double functionValue) {
41         this->pm = pm;
42         this->pc = pc;
43         this->popSize = popSize;
44         this->functionValue = functionValue ;
45     }
46
47     double pc, pm;

```

```
48     int popSize;  
49     double functionValue;  
50 };
```

ПРИЛОЖЕНИЕ В

Код классов для экспериментов с алгоритмом

Далее приводится код класса Experiment:

```
1 // Класс отвечающий за проведение эксперимента
2 class Experiment {
3     private :
4         // число тестовых итераций
5         const int ITER = 100;
6         // Параметры графа
7         GraphDescription graph;
8         // функция для вычисления времени работы
9         // алгоритма с заданными параметрами
10        pair<double, double> calculateTimeErrorValue( int popSize, double pm, double pc ) {
11            // число ошибок и среднее время работы
12            int error = 0;
13            double sum_time = 0.0;
14            // итеративное выполнение тестов
15            for (int i = 0; i < ITER; i++) {
16                // инициализация графа
17                Graph* g1 = new Graph(graph.n, graph.m, graph.e);
18                // инициализация генетического алгоритма
19                GeneticAlgorithm genAlg(g1, popSize, pc, pm);
20                // замер времени работы алгоритма
21                double time = 0.0;
22                // получение найденного радиуса
23                int R = genAlg.getBestResult(time);
24                sum_time += time;
25                // проверка на верно найденный радиус
26                if (R != graph.realR) {
27                    error++;
28                }
29                delete g1;
30            }
31            // вычисление среднего времени работы и процента ошибок
32            double avg_time = sum_time / double(ITER);
33            double avg_error = error / double(ITER) * 100.0;
34            return make_pair(avg_time, avg_error);
35        }
36        // запуск алгоритма с одним из изменяющихся параметров
37        // popSize – размер популяции, probParam – значение одного из параметров
```

```

38 // pmProb – флаг, отвечающий за то какой из параметров передан в качестве
39 // фиксированного
40 // true – параметр pm, false – параметр pc
41 pair<vector<GaTestResult>, vector<GaTestResult>> testWithChangebleProb(int
42 popSize, double probParam, bool pmProb) {
43     // шаг, с которым будет перебираться параметр
44     double step = 0.1;
45     // число итераций для перебора параметров
46     int itarationCount = ceil(1.0 / step);
47     // вывод информации о параметрах
48     cout << "popSize = " << popSize;
49     if (pmProb) {
50         cout << " pm = " << probParam << endl;
51     }
52     else {
53         cout << " pc = " << probParam << endl;
54     }
55     // вывод шапки для данных
56     if (pmProb) {
57         cout << "pc \t AVG_TIME \t ERROR" << endl;
58     }
59     else {
60         cout << "pm \t AVG_TIME \t ERROR" << endl;
61     }
62     // переменные для хранения результатов измерений
63     vector<GaTestResult> time;
64     vector<GaTestResult> error ;
65     // перебор одного из параметров
66     for (int i = 0; i <= itarationCount ; i++) {
67         double pm, pc;
68         // в зависимости от флага pmProb перебирается или параметр
69         // pm или pc
70         if (pmProb) {
71             pm = probParam;
72             pc = i * step ;
73         }
74         else {
75             pc = probParam;
76             pm = i * step ;
77         }
78         // вычисление среднего времени работы и процента ошибок

```

```

76         pair<double, double> metering = this ->calculateTimeErrorValue(
77             popSize, pm, pc);
78         double avg_time = metering. first ;
79         double error_percent = metering. second;
80         // вывод информации о полученных результатах
81         cout << (pmProb ? pc : pm) << "\t" << avg_time << "\t" <<
82             error_percent << endl;
83         time.push_back(GaTestResult(popSize, pm, pc, avg_time));
84         error.push_back(GaTestResult(popSize, pm, pc, error_percent));
85     }
86 }
87
88 public :
89     Experiment(GraphDescription graph) {
90         srand(time(NULL) % INT_MAX);
91         this ->graph = graph;
92     }
93     // Тест с изменяющимся параметром pc
94     pair<vector<GaTestResult>, vector<GaTestResult>> oneDimentionFixedGATest() {
95         // параметры алгоритма для теста
96         int populationN = 20;
97         double pm = 0.4;
98         return this ->testWithChangebleProb(populationN, pm, true);
99     }
100    // запуск простого теста с переданными параметрами
101    pair<double, double> simpleTimeErrorTest(double pm, double pc, int popSize) {
102        return this ->calculateTimeErrorValue(popSize, pm, pc);
103    }
104    // запуск алгоритма N4N
105    void simpleNANTest() {
106        // действия аналогичны – измеряется среднее
107        // время работы и процент ошибки
108        int error = 0;
109        double avg_time = 0.0;
110        for (int i = 0; i < ITER; i++) {
111            Graph* g1 = new Graph(graph.n, graph.m, graph.e);
112            cout << "Start OTHER genetic algorithm" << endl;
113            SimpleGeneticAlgorithm sgen(g1, 50, 10, 0.7, 0.1);
114            double time = 0.0;

```

```

115         int R = sgen. getBestResult (time);
116         avg_time += time;
117         if (R != graph.realR)
118             error++;
119     }
120     cout << "AVG Time = " << avg_time / double(ITER) << endl;
121     cout << "Error = " << double(error) / double(ITER) << endl;
122 }
123 // тестирование алгоритма с перебором параметров pm и pc
124 pair<vector<GaTestResult>, vector<GaTestResult>> pmpcGaTest() {
125     // размер популяции
126     int popSize = 20;
127     // шаг для параметров
128     double step = 0.1;
129     // начальные значения
130     double pm = 0.0;
131     // число итераций
132     int sectionNumber = ceil (1.0 / step);
133     // результаты измерений времени
134     vector<GaTestResult> time;
135     // результаты измерений ошибок
136     vector<GaTestResult> error ;
137     // перебор параметров
138     for (int i = 0; i <= sectionNumber; i++) {
139         pm = i * step ;
140         // получение результатов тестов по второму параметру
141         pair<vector<GaTestResult>, vector<GaTestResult>> metrings = this
142             ->testWithChangebleProb(popSize, pm, true);
143         // добавление результатов тестов
144         time. insert (time.end(), metrings. first .begin(), metrings. first .
145             end());
146         error . insert ( error .end(), metrings.second.begin(), metrings.
147             second.end());
148     }
149     return make_pair(time, error );
150 }
151 // тестирование алгоритма с перебором
152 // размера популяции
153 void nGATest() {
154     // начальные значения вероятностей
155     double pm = 0.2;

```

```

153     double pc = 0.3;
154     // максимальное значение для размера популяции
155     int maxN = 50;
156     // вывод информации о параметрах
157     cout << "pm = " << pm << " pc = " << pc << endl;
158     vector<double> time;
159     vector<double> error;
160     // перебор размеров популяции
161     for (int popSize = 1; popSize < maxN; popSize++) {
162         // вычисление размеров и временных затрат
163         pair<double, double> metering = this->calculateTimeErrorValue(
164             popSize, pm, pc);
165         time.push_back(metering.first);
166         error.push_back(metering.second);
167         // вывод информации на экран
168         cout << "(" << popSize << ", " << metering.second << ")" << endl;
169     }
170     // вывод информации о времени работы
171     for (int i = 0; i < time.size(); i++) {
172         cout << "(" << i + 1 << ", " << time[i] << ")" << endl;
173     }
174 };

```

ПРИЛОЖЕНИЕ Г

Уровень доступа данных

Далее приводится программный код основных файлов, описывающих уровень доступа к данным.

Программный код класса GraphContext:

```
1  using GeneticAlgorithm. Entities ;
2  using System;
3  using System.Collections .Generic;
4  using System.Data.Entity ;
5  using System.Data.SqlClient ;
6  using System.Linq;
7  using System.Text;
8  using System.Threading.Tasks;
9
10 namespace GeneticAlgorithmWEB.Dao
11 {
12     // Класс отвечает за предоставление
13     // доступа к коллекции объектов графов.
14     public class GraphContext : DbContext
15     {
16         static GraphContext() {
17             // Установка класса, за счет которого происходит
18             // начальная инициализация базы данных графиками.
19             Database. SetInitializer (new GraphContextInitializer ());
20         }
21         public GraphContext() : base("DB") { }
22         // Набор графов из базы данных.
23         public DbSet<Graph> Graphs { get; set; }
24     }
25 }
```

Программный код класса GraphDao:

```
1  using GeneticAlgorithm. Entities ;
2  using GeneticAlgorithmWEB.DAL.Interfaces;
3  using System;
4  using System.Collections .Generic;
5  using System.Data.Entity ;
6  using System.Linq;
7
8  namespace GeneticAlgorithmWEB.Dao
9  {
```

```

10 // Класс реализующий уровень доступа к данным.
11 // В классе определены основные методы для
12 // получения графов из базы данных.
13 public class GraphDao : IGraphDao
14 {
15     // Сохраняет новый график в базе данных.
16     public Graph Add(Graph graph)
17     {
18         using (GraphContext context = new GraphContext())
19         {
20             // добавление нового графа с помощью контекста
21             Graph res = context.Graphs.Add(graph);
22             // сохранение изменений
23             context.SaveChanges();
24             // возврат графа с заполненным Id
25             return res;
26         }
27     }
28
29     // Получение описательной информации обо всех графах в базе данных.
30     public IEnumerable<GraphInfo> GetAllGraphInfo()
31     {
32         // Список для информации о графах
33         List<GraphInfo> res = new List<GraphInfo>();
34         using (GraphContext context = new GraphContext())
35         {
36             // Получение основной информации о графах
37             foreach (var graph in context.Graphs)
38             {
39                 // добавление нового графа
40                 res.Add(new GraphInfo()
41                 {
42                     Id = graph.Id,
43                     N = graph.N,
44                     M = graph.M,
45                     Name = graph.Name,
46                 });
47             }
48             return res;
49         }
50     }

```

```

51
52     // Получение графа по Id.
53     // Выбрасывает исключение,
54     // если график с переданным Id не удалось найти в БД.
55     public Graph GetById(int id)
56     {
57         using (GraphContext context = new GraphContext())
58         {
59             // Поиск графа при помощи LINQ запроса к контексту с графиками.
60             // поиск по совпадающему Id, после чего явно включается набор вершин в
61             // график
62             Graph res = context.Graphs
63                 .Where(g => g.Id == id)
64                 .Include(g => g.Edges)
65                 .FirstOrDefault();
66             // если не найден график, то выбрасывается исключение
67             if (res == null)
68             {
69                 throw new ArgumentException($"Invalid graph id = {id}");
70             }
71             return res;
72         }
73     }
74 }
```

Программный код класса UserContext:

```

1  using GeneticAlgorithm.Entities.Users;
2  using System;
3  using System.Collections.Generic;
4  using System.Data.Entity;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace GeneticAlgorithmWEB.Dao
10 {
11     public class UserContext : DbContext
12     {
13         // Класс отвечает за предоставление
14         // доступа к коллекции объектов пользователей.
15         static UserContext() {
```

```

16     // Установка класса, за счет которого происходит
17     // начальная инициализация базы данных пользователями.
18     Database. SetInitializer (new UserContextInitializer ());
19 }
20 public UserContext() : base("DB") {}
21     // Набор графов из базы данных.
22     public DbSet<User> Users { get; set; }
23 }
24 }
```

Программный код класса UserContextInitializer:

```

1 using GeneticAlgorithm. Entities .Requests;
2 using GeneticAlgorithm. Entities .Users;
3 using GeneticAlgorithmWEB.BLL;
4 using GeneticAlgorithmWEB.Encrypt;
5 using System;
6 using System.Collections .Generic;
7 using System.Data.Entity ;
8 using System.Linq;
9 using System.Net. Security ;
10 using System.Text;
11 using System.Threading.Tasks;
12
13 namespace GeneticAlgorithmWEB.Dao
14 {
15     class UserContextInitializer : CreateDatabaseIfNotExists<UserContext>
16     {
17         // Инициализация базы данных одним пользователем.
18         protected override void Seed(UserContext context)
19         {
20             // Сохранение в базе данных пользователя
21             // с паролем admin и логином admin
22             CreateUserRequest userRequest = new CreateUserRequest() {
23                 Login = "admin",
24                 Password = "admin"
25             };
26             // Хеширование пароля пользователя
27             Encryption encryption = new Encryption();
28             User user = new User() {
29                 Login = userRequest.Login,
30                 Password = encryption.CreatePassword(userRequest.Password),
31             };
32 }
```

```

32         // Сохранение графа
33         context .Users.Add(user);
34         context .SaveChanges();
35     }
36 }
37 }
```

Программный код класса UserDao:

```

1  using GeneticAlgorithm. Entities .Users;
2  using GeneticAlgorithmWEB.DAL.Interfaces;
3  using System;
4  using System.Collections .Generic;
5  using System.Linq;
6  using System.Text;
7  using System.Threading.Tasks;
8
9  namespace GeneticAlgorithmWEB.Dao
10 {
11     // Класс реализующий уровень доступа к данным.
12     // В классе определены основные методы для
13     // получения пользователей из базы данных.
14     public class UserDao : IUserId
15     {
16         // Сохраняет нового пользователя в базе данных.
17         public User Add(User user)
18         {
19             // Добавление нового пользователя
20             using (UserContext context = new UserContext())
21             {
22                 User res = context .Users.Add(user);
23                 context .SaveChanges();
24                 // возвращение пользователя с заполненным Id
25                 return res ;
26             }
27         }
28         // Получение пользователя по Id
29         public User GetById(int id)
30         {
31             using (UserContext context = new UserContext())
32             {
33                 // поиск пользователя с заданным Id
34                 return context .Users.Where(u => u.Id == id) .FirstOrDefault () ;
```

```
35         }
36     }
37     // Получение пользователя по логину
38     public User GetByLogin(string name)
39     {
40         using (UserContext context = new UserContext())
41         {
42             // поиск пользователя с заданным логином
43             return context.Users.Where(u => u.Login == name).FirstOrDefault();
44         }
45     }
46 }
47 }
```

ПРИЛОЖЕНИЕ Д

Уровень бизнес-логики

Далее приводится программный код, за счет которого происходит реализация уровня бизнес-логики.

Программный код класса Algorithm:

```
1  using GA;
2  using GeneticAlgorithm;
3  using GeneticAlgorithm.Entities ;
4  using GeneticAlgorithmWEB.BLL.Interfaces;
5  using System;
6  using System.Collections .Generic;
7  using System.Diagnostics ;
8  using System.Linq;
9  using System.Text;
10 using System.Threading.Tasks;
11 using GeneticAlgorithm.Entities .Requests;
12
13 namespace GeneticAlgorithmWEB.BLL
14 {
15     public class Algorithm : IAlgorithm
16     {
17         private readonly IGraphBL _graphBL;
18         // Фиксированные параметры для работы генетического алгоритма.
19         private double _fixedPM = 0.4;
20         private double _fixedPC = 0.4;
21         private int _fixedPopSize = 30;
22         // Число запусков при тестировании алгоритма с заданными параметрами.
23         private int _testCount = 10;
24
25         public Algorithm(IGraphBL graphBL)
26         {
27             _graphBL = graphBL;
28         }
29         // Поиск центральных вершин при помощи генетического алгоритма.
30         // Алгоритм запускается с фиксированными параметрами мутации, скрецивания
31         // и размер популяции.
32         public FindingVertexResponse FindCentralVertex (Graph graph)
33         {
34             // поиск центральных вершин при помощи генетического алгоритма с
35             // заданными параметрами
```

```

34     GeneticAlgorithmCore ga = new GeneticAlgorithmCore(graph, _fixedPopSize,
35             _fixedPM, _fixedPC);
36     return ga. StartAlgorithm () ;
37
38     // Исследование алгоритма с переданными параметрами.
39     public ResearchAlgorithmResponse ResearchAlgorithm(ResearchRequest param) {
40         Graph graph = _graphBL.GetById(param.GraphId);
41         double avgTime = 0.0;
42         int error = 0;
43         // Инициализация алгоритма с переданными параметрами.
44         GeneticAlgorithmCore ga = new GeneticAlgorithmCore(graph, param.
45             PopulationSize , param.Pm, param.Pc);
46         // Многократный запуск алгоритма для оценки времени работы и процента
47         // ошибок.
48         for (int i = 0; i < _testCount; i++)
49         {
50             // Запуск алгоритма и получение результатов
51             FindingVertexResponse algResult = ga. StartAlgorithm () ;
52             // проверка на верность найденного решения
53             if ( algResult .R != graph.R) {
54                 error++;
55             }
56             // увеличение суммарного времени работы
57             avgTime += algResult. Time;
58         }
59         // возвращение результата в виде среднего
60         // значения времени работы и процента неверных ответов
61         return new ResearchAlgorithmResponse() {
62             AvgTime = avgTime / _testCount,
63             Error = error / (double)_testCount * 100.0,
64         };
65     }
66 }

```

Программный код класса Encryption:

```

1  using System;
2  using System.Collections .Generic;
3  using System.Diagnostics .Eventing.Reader;
4  using System.Linq;
5  using System.Security .Cryptography;
6  using System.Text;

```

```

7  using System.Threading.Tasks;
8
9  namespace GeneticAlgorithmWEB.Encrypt
10 {
11     // Класс отвечающий за хеширование паролей и их проверку.
12     public class Encryption
13     {
14         // Размер соли.
15         private readonly int saltSize = 12;
16         // Размер хеша.
17         private readonly int hashSize = 20;
18         // Число итераций для задержки.
19         private readonly int iterations = 10000;
20
21         // Создание хеша пароля.
22         // Результат представляет собой последовательно
23         // записанные соль и хеш пароля в виде массива байтов.
24         public byte[] CreatePassword(string password)
25         {
26             // Создание "соли" для пароля
27             byte[] salt = new byte[ saltSize ];
28             new RNGCryptoServiceProvider().GetBytes(salt);
29             // Хеширование пароля с солью
30             return HashPassword(password, salt);
31         }
32
33         private byte[] HashPassword(string password, byte[] salt )
34         {
35             // Генерация хеша.
36             var pbkdf2 = new Rfc2898DeriveBytes(password, salt , iterations );
37             // Выбор из хеша заданного числа байтов.
38             byte[] hash = pbkdf2.GetBytes(hashSize);
39             // Выделение памяти под результат.
40             byte[] result = new byte[ saltSize + hashSize];
41             // Копирование соли и хеша в единый массив данных.
42             Array.Copy(salt , 0, result , 0, saltSize );
43             Array.Copy(hash, 0, result , saltSize , hashSize);
44             return result ;
45         }
46
47         // Проверка верности пароля.

```

```

48 // Передаются байты из базы данных и пароль для проверки.
49 public bool CheckPassword(byte[] realPassword, string check) {
50     // Выделение соли из массива байтов.
51     byte[] salt = GetSaltFromHash(realPassword);
52     // Хеширование пароля с сохраненной солью
53     byte[] hashedPassword = HashPassword(check, salt);
54     if (hashedPassword.Length != realPassword.Length) {
55         throw new ArgumentException("Длина реального хеша не совпадет с длиной
56                                     хеша из базы данных");
57     }
58     // Побайтовая проверка на соответствие пароля и хеши.
59     for (int i = 0; i < hashedPassword.Length; i++) {
60         if (hashedPassword[i] != realPassword[i]) {
61             return false;
62         }
63     }
64     return true;
65 }
66 // Выделение "соли" из массива байтов.
67 private byte[] GetSaltFromHash(byte[] hash)
68 {
69     // создание массива байт
70     byte[] salt = new byte[ saltSize ];
71     // копирование в этот массив элементов из хеши
72     Array.Copy(hash, 0, salt, 0, saltSize );
73     return salt;
74 }
75 }
```

Программный код класса GraphBL:

```

1 using GA;
2 using GeneticAlgorithm;
3 using GeneticAlgorithm.Entities ;
4 using GeneticAlgorithmWEB.BLL.Interfaces;
5 using GeneticAlgorithmWEB.DAL.Interfaces;
6 using System;
7 using System.Collections.Generic;
8
9 namespace GeneticAlgorithmWEB.BLL
10 {
11     // Уровень бизнес-логики.
```

```

12 // Реализует работу с графами.
13 public class GraphBL : IGraphBL
14 {
15     // Ссылка на объект, реализующий доступ к графикам
16     private readonly IGraphDao _graphDao;
17     // максимальный размер графа
18     private readonly int _maxN = 2500;
19
20     public GraphBL(IGraphDao graphDao)
21     {
22         _graphDao = graphDao;
23     }
24
25     // Добавление графа в базу данных.
26     // Выбрасывается исключение, если график превышает максимальные размеры или
27     // не является связным.
28     public Graph Add(Graph graph)
29     {
30         // Проверка графа на максимальный размер.
31         if (graph.N > _maxN) {
32             throw new FormatException($"Количество вершин в графике должно быть
33             меньше, чем {_maxN}");
34         }
35         GraphContext context = new GraphContext(graph);
36         // Проверка на связность.
37         if (!context.CheckConnectivity())
38         {
39             throw new FormatException("Граф должен быть связанным");
40         }
41         // Работа точного алгоритма
42         ExactAlgorithmCore exactAlgorithm = new ExactAlgorithmCore();
43         // нахождение реального радиуса графа при помощи точного алгоритма
44         int R = exactAlgorithm.FindRadius(context);
45         graph.R = R;
46         // добавление нового графа
47         return _graphDao.Add(graph);
48     }
49
50     // Получение описания всей информации о графах.
51     public IEnumerable<GraphInfo> GetAllGraphInfo()
52     {

```

```

52         return _graphDao.GetAllGraphInfo();
53     }
54
55     // Получение графа по Id
56     public Graph GetById(int id) {
57         return _graphDao.GetById(id);
58     }
59 }
60 }
```

Программный код класса GraphBL:

```

1  using GA;
2  using GeneticAlgorithm;
3  using GeneticAlgorithm.Entities ;
4  using GeneticAlgorithmWEB.BLL.Interfaces;
5  using GeneticAlgorithmWEB.DAL.Interfaces;
6  using System;
7  using System.Collections.Generic;
8
9  namespace GeneticAlgorithmWEB.BLL
10 {
11     // Уровень бизнес-логики.
12     // Реализует работу с графиками.
13     public class GraphBL : IGraphBL
14     {
15         // Ссылка на объект, реализующий доступ к графикам
16         private readonly IGraphDao _graphDao;
17         // максимальный размер графа
18         private readonly int _maxN = 2500;
19
20         public GraphBL(IGraphDao graphDao)
21         {
22             _graphDao = graphDao;
23         }
24
25         // Добавление графа в базу данных.
26         // Выбрасывается исключение, если график превышает максимальные размеры или
27         // не является связным.
28         public Graph Add(Graph graph)
29         {
30             // Проверка графа на максимальный размер.
31             if (graph.N > _maxN) {
```

```

32         throw new FormatException($"Количество вершин в графе должно быть
33             меньше, чем {_maxN}");  

34     }  

35     GraphContext context = new GraphContext(graph);  

36     // Проверка на связность.  

37     if (!context.CheckConnectivity())  

38     {  

39         throw new FormatException("Граф должен быть связанным");  

40     }  

41     // Работа точного алгоритма  

42     ExactAlgorithmCore exactAlgorithm = new ExactAlgorithmCore();  

43     // нахождение реального радиуса графа при помощи точного алгоритма  

44     int R = exactAlgorithm.FindRadius(context);  

45     graph.R = R;  

46     // добавление нового графа  

47     return _graphDao.Add(graph);  

48 }  

49 // Получение описания всей информации о графах.  

50 public IEnumerable<GraphInfo> GetAllGraphInfo()  

51 {  

52     return _graphDao.GetAllGraphInfo();  

53 }  

54 // Получение графа по Id  

55 public Graph GetById(int id) {  

56     return _graphDao.GetById(id);  

57 }  

58 }  

59 }  

60 }

```

ПРИЛОЖЕНИЕ Е

Уровень пользовательского интерфейса

Далее приводится описание основных классов-контроллеров.

Программный код контроллера GraphController:

```
1  using GeneticAlgorithm. Entities ;
2  using GeneticAlgorithm. Entities .Requests;
3  using GeneticAlgorithmWEB.BLL;
4  using GeneticAlgorithmWEB.BLL.Interfaces;
5  using System;
6  using System.Collections .Generic;
7  using System.IO;
8  using System.Web;
9  using System.Web.Mvc;

10
11 namespace SimplePages.Controllers
12 {
13     public class GraphController : Controller
14     {
15         // ссылка на модуль с алгоритмом
16         private readonly IAlgorithm _algorithmWork;
17         // ссылка на модуль с логикой для работы с графиком
18         private readonly IGraphBL _graphBL;

19
20         public GraphController(IAlgorithm algorithmWork, IGraphBL graphBL) {
21             _algorithmWork = algorithmWork;
22             _graphBL = graphBL;
23         }

24
25         // Метод для получения страницы поиска вершины.
26         [HttpGet]
27         public ActionResult Index()
28         {
29             return View("FindCenter");
30         }

31         // Метод для нахождения центральных вершин в графике.
32         [HttpPost]
33         public ActionResult FindCentralVertex (HttpPostedFileBase upload)
34         {
35             // проверка выбрал ли пользователь график
36             if (upload == null)
```

```

37    {
38        return View("~/Views/Shared/Error.cshtml", model: "Файл не был выбран");
39    }
40    try
41    {
42        // чтение строк файла
43        Graph graph = ReadGraph(upload);
44        // запуск алгоритма
45        FindingVertexResponse algorithmResult = _algorithmWork.FindCentralVertex(
46            graph);
47        // возврат представления с полученными результатами
48        return View("CalculationResult", algorithmResult);
49    }
50    catch (FormatException e)
51    {
52        // при выбрасывании исключения возвращается
53        // представление с сообщением об ошибке
54        return View("~/Views/Shared/Error.cshtml", model: e.Message);
55    }
56    // Метод для возвращения страницы с добавлением графа.
57    // Работает только для авторизованных пользователей.
58    [HttpGet]
59    [Authorize]
60    public ActionResult Add()
61    {
62        return View("Add");
63    }
64    // Метод для добавления графа.
65    // Работает только для авторизованных пользователей.
66    [HttpPost]
67    [Authorize]
68    public ActionResult Add(AddGraphRequest request)
69    {
70        // проверка, что пользователь выбрал файл
71        if (request.Upload == null)
72        {
73            return View("~/Views/Shared/Error.cshtml", model: "Выберите файл с
74            графиком");
75        }
76        try
77        {
78            // чтение графа из файла
79            Graph graph = ReadGraph(request.Upload);

```

```

76    // проверка на существование имени файла
77    if (request.Name == null) {
78        graph.Name = "Граф пользователя";
79    }
80    else {
81        graph.Name = request.Name;
82    }
83    // добавление файла в базу данных
84    _graphBL.Add(graph);
85    // перенаправление на главную страницу
86    return Redirect("/");
87}
88 catch (FormatException e) {
89     // при выбрасывании исключения возвращается представление с
90     // сообщением об ошибке
91     return View("~/Views/Shared/Error.cshtml", model: e.Message);
92 }
93 // Чтение файла из потока данных.
94 private string[] ReadFile(Stream stream)
95 {
96     // читатель потока
97     StreamReader reader = new StreamReader(stream);
98     // результат чтения
99     List<string> lines = new List<string>();
100    // текущая строка файла
101    string line;
102    // построчное чтение файла до тех пор,
103    // пока не будет достигнут конец файла
104    while ((line = reader.ReadLine()) != null)
105    {
106        lines.Add(line);
107    }
108    return lines.ToArray();
109}
110 // Чтение графа из строк файла.
111 private Graph ReadGraph(HttpPostedFileBase upload)
112 {
113     // выделение строк из входного потока
114     string[] fileLines = ReadFile(upload.InputStream);
115     // парсинг строк

```

```

116         return GraphParser.ParseTxtFormat( fileLines );
117     }
118 }
119 }
```

Программный код контроллера HomeController:

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Web;
5 using System.Web.Mvc;
6
7 namespace SimplePages.Controllers
8 {
9     public class HomeController : Controller
10    {
11        // Метод для получения домашней страницы.
12        public ActionResult Index()
13        {
14            return View("Index");
15        }
16    }
17 }
```

Программный код контроллера LoginController:

```

1 using GeneticAlgorithm.Entities.Requests;
2 using GeneticAlgorithmWEB.BLL.Interfaces;
3 using System;
4 using System.Collections.Generic;
5 using System.Linq;
6 using System.Runtime.InteropServices;
7 using System.Web;
8 using System.Web.Mvc;
9 using System.Web.Security;
10
11 namespace SimplePages.Controllers
12 {
13     public class LoginController : Controller
14     {
15         // ссылка на модуль с бизнес-логикой для доступа к пользователям
16         private readonly IUserBL _userBL;
17         // имена представлений
```

```

18     private readonly string signUpViewName = "SignUp";
19     private readonly string signInViewName = "SignIn";
20     private readonly string successSignUpViewName = "SignUpSuccess";
21
22     public LoginController(IUserBL userBL)
23     {
24         _userBL = userBL;
25     }
26     // Страница для входа в систему.
27     [HttpGet]
28     public ActionResult Index()
29     {
30         return View(signInViewName, new LoginUserRequest());
31     }
32     // Метод для входа в систему.
33     [HttpPost]
34     public ActionResult SignIn(LoginUserRequest user) {
35         // проверка валидности входных данных
36         if (!ModelState.IsValid) {
37             return View(signInViewName, user);
38         }
39         // проверка корректности пароля
40         if (_userBL.CheckPassword(user))
41         {
42             // выделение cookie для аутентификации
43             FormsAuthentication.SetAuthCookie(user.Login, true);
44             // перенаправление на главную страницу
45             return Redirect("/");
46         }
47         else {
48             // если пароль не верный, то пользователь увидит сообщение об ошибке
49             // данные не удовлетворяют требованиям
50             ModelState.AddModelError("LOGIN_PASSWORD", "Неверный логин или
51                                         пароль");
52             // возвращение представления с сообщением об ошибке
53             return View(signInViewName, user);
54         }
55         // Страница для регистрации нового пользователя.
56         [HttpGet]
57         [Authorize]

```

```

58     public ActionResult SignUp() {
59         // представление для добавления нового пользователя
60         return View(signUpViewName);
61     }
62     // Метод для добавления нового пользователя.
63     [HttpPost]
64     [Authorize]
65     public ActionResult SignUp(CreateUserRequest creatingUser) {
66         // проверка на валидность переданных данных
67         if (!ModelState.IsValid) {
68             // возврат представления при неверных данных
69             return View(signUpViewName, creatingUser);
70         }
71         // проверка на существование пользователя с заданным логином
72         if (!_userBL.UserExists(creatingUser)) {
73             // если пользователь не существует,
74             // то новый пользователь добавляется
75             _userBL.Add(creatingUser);
76             // возврат представления с сообщением об успешной регистрации
77             return View(successSignUpViewName, creatingUser);
78         }
79         else {
80             // добавление сообщения об ошибке
81             ModelState.AddModelError("LOGIN_PASSWORD", "Пользователь с таким
82             логином уже существует");
83         }
84         // возврат представления с сообщением об ошибке
85         return View(signUpViewName, creatingUser);
86     }
87     // Метод для выхода из системы.
88     [HttpGet]
89     public ActionResult SignOut() {
90         // выход из аккаунта
91         FormsAuthentication.SignOut();
92         // перенаправление на главную страницу
93         return Redirect("/");
94     }
95 }
```

Программный код контроллера ResearchController:

```
1 using GeneticAlgorithm.Entities ;
```

```

2  using GeneticAlgorithm. Entities .Requests;
3  using GeneticAlgorithm. Entities .Response;
4  using GeneticAlgorithmWEB.BLL.Interfaces;
5  using System. Collections .Generic;
6  using System.Web.Mvc;

7
8  namespace SimplePages. Controllers
9  {
10     public class ResearchController : Controller
11     {
12         // ссылки на модули для работы с
13         // графиками и генетическим алгоритмом
14         private readonly IGraphBL _graphBL;
15         private readonly IAlgorithm _algorithmWork;

16
17         public ResearchController (IGraphBL graphBL, IAlgorithm algorithmWork)
18         {
19             _graphBL = graphBL;
20             _algorithmWork = algorithmWork;
21         }
22         // Метод получения страницы для выбора параметров для запуска генетического
23         // алгоритма.
24         [HttpGet]
25         public ActionResult Index()
26         {
27             // получение информации о сохраненных графах
28             IEnumerable<GraphInfo> graphs = _graphBL.GetAllGraphInfo();
29             // возврат представления для выбора параметров с информацией о графах
30             return View("ResearchParametrs", model: graphs);
31         }
32         // Метод запуска алгоритма с выбранными параметрами.
33         [HttpPost]
34         public ActionResult ResearchAlgorithm(ResearchRequest request) {
35             // получение ответа от алгоритма
36             ResearchAlgorithmResponse response = _algorithmWork.ResearchAlgorithm(request
37                 );
38             // возврат представления с полученными результатами работы алгоритма
39             return View("ResearchResult", model: new AlgorithmResultResponse() {
40                 ResearchRequest = request ,
41                 AlgorithmResponse = response,
42             });
}

```

41 }

42 }

43 }

ПРИЛОЖЕНИЕ Ж

CD-диск с отчетом о выполненной работе

На приложенном диске можно ознакомиться со следующими файлами:

Папка Diploma — L^AT_EX-вариант отчета о практике;

Папка GAWeb — Visual-Studio проект, написанный на C++, с кодом для тестирования алгоритма;

Папка GAWeb — Visual-Studio проект, написанный на C#, с полным кодом веб-приложения;