

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

Кафедра математической кибернетики и компьютерных наук

**ИССЛЕДОВАНИЕ ПАРАМЕТРОВ ГЕНЕТИЧЕСКОГО АЛГОРИТМА  
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  
к. ф.-м. н. \_\_\_\_\_ С. В. Миронов

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

## **СОДЕРЖАНИЕ**

<b>ВВЕДЕНИЕ .....</b>	<b>3</b>
1 Описание задачи .....	4
2 Генетические алгоритмы .....	5
2.1 Представление решения и начальная популяция .....	6
2.2 Оператор скрещивания .....	6
2.3 Оператор мутации .....	7
2.4 Оператор естественного отбора .....	7
3 Алгоритмы для поиска центральных вершин .....	8
3.1 Тривиальный алгоритм .....	8
3.2 Алгоритмы с улучшенной асимптотикой .....	8
3.3 Алгоритмы, использующие матричное умножение .....	9
4 Описание разработанного генетического алгоритма .....	10
4.1 Старт алгоритма .....	11
4.2 Естественный отбор .....	11
4.3 Этап скрещивания .....	11
4.4 Этап мутации .....	12
5 Реализация алгоритма .....	16
6 Модели случайных графов .....	17
6.1 Модель случайного графа Эрдеша-Ренъи .....	17
6.2 Модель случайного графа Барабаши-Альберт .....	17
6.3 Геометрический случайный граф .....	18
7 Результаты вычислительных экспериментов .....	19
8 Исследование параметров генетического алгоритма .....	22
<b>ЗАКЛЮЧЕНИЕ .....</b>	<b>26</b>
<b>СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ .....</b>	<b>27</b>

## **ВВЕДЕНИЕ**

Введение

## 1 Описание задачи

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 предлагается и исследуется задача поиска центральных вершин в графах, поэтому далее приводится формальное описание задачи. Для начала можно дать определение эксцентриситета вершины в графе. Эксцентриситетом  $e$  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.

## **2 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, мутации 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.

Вместе с тем элементом, который отвечает за селекцию и определение 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 низким уровнем приспособленности.

При рассмотрении природы, которая окружает организм, как некоторой сложно организованной системы легко заметить, что в процессе эволюции популяции 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. Главный толчок для развития генетических алгоритмов был дан в работе Дж. Г. Холланда [1].

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

## 2.1 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  $N$ .

## 2.2 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 с этим процесс скрещивания происходит не всегда, а с вероятностью заданной в виде параметра  $p_c$ .

## **2.3 Оператор мутации**

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

## **2.4 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 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 результате 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.

### 3 Алгоритмы для поиска центральных вершин

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.

При этом следует отметить, что для поиска центральных вершин существует ряд точных алгоритмов, гарантирующих верное решение во всех случаях.

#### 3.1 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 эксцентризитетом 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  $O(n + m)$  [2]. При этом, если запустить его из каждой вершины, то время работы всего алгоритма будет равным  $O(n^2 + nm)$ .

#### 3.2 Алгоритмы с улучшенной ассимптотикой

Кроме этого данная задача была хорошо изучена различными исследователями [ссылки], которые предлагали несколько подходов, для решения подобных задач. Например, в работе [3] 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.

### **3.3 Алгоритмы, использующие матричное умножение**

В работе [4] 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 асимптотика этого подхода равна времени выполнения матричного умножения. Тривиальный алгоритм имеет асимптотику  $O(n^3)$ , но при этом существует алгоритм быстрого матричного умножения [5], способный решить эту задачу за время  $O(n^{2.81})$ .

#### 4 Описание разработанного 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).

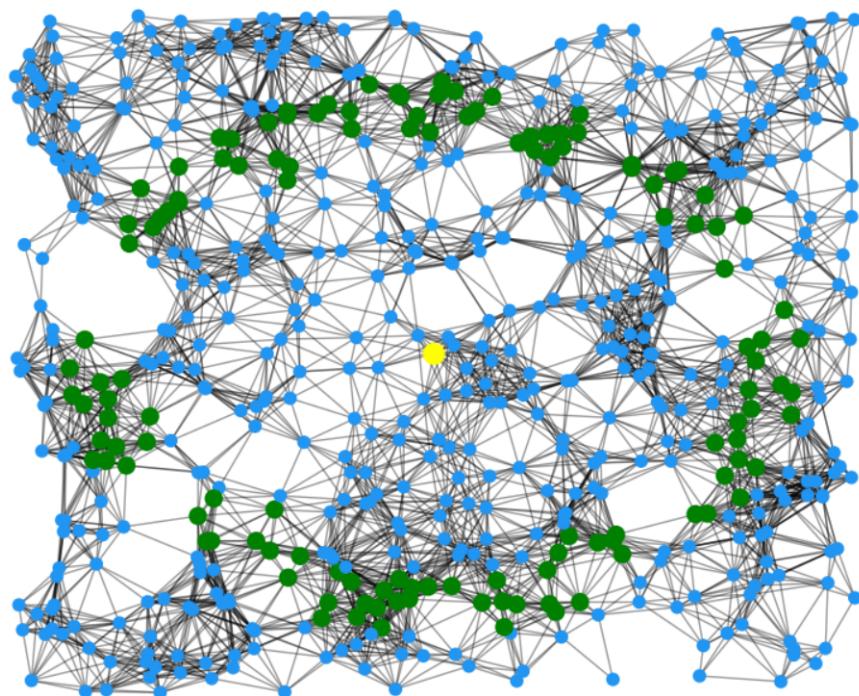


Рисунок 1 – Начальное состояние алгоритма (желтый цвет – центральная вершина, зеленый цвет – начальная популяция)

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 эксцентричитетами. 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.

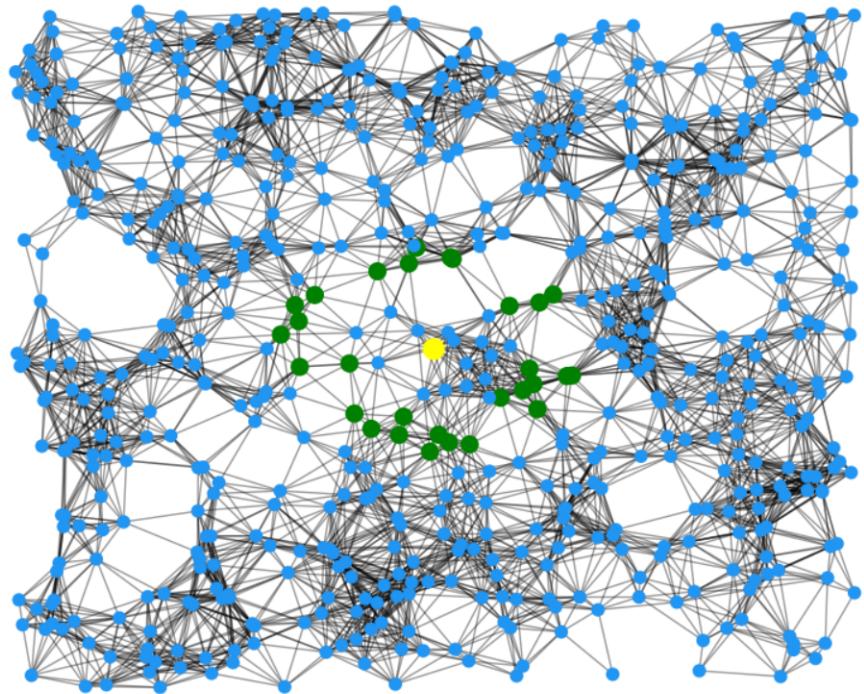


Рисунок 2 – Состояние алгоритма после нескольких итераций (желтый цвет – центральная вершина, зеленый цвет – начальная популяция)

#### 4.1 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.

#### 4.2 Естественный отбор

Как уже говорилось ранее естественный отбор занимается выбором оптимальных вершин для продолжения работы алгоритма. Для каждой вершины находится ее эксцентриситет при помощи обхода в ширину (см. рисунок 3), после чего методом колеса рулетки, при этом приоритет отдается вершинам с меньшим эксцентриситетом.

#### 4.3 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

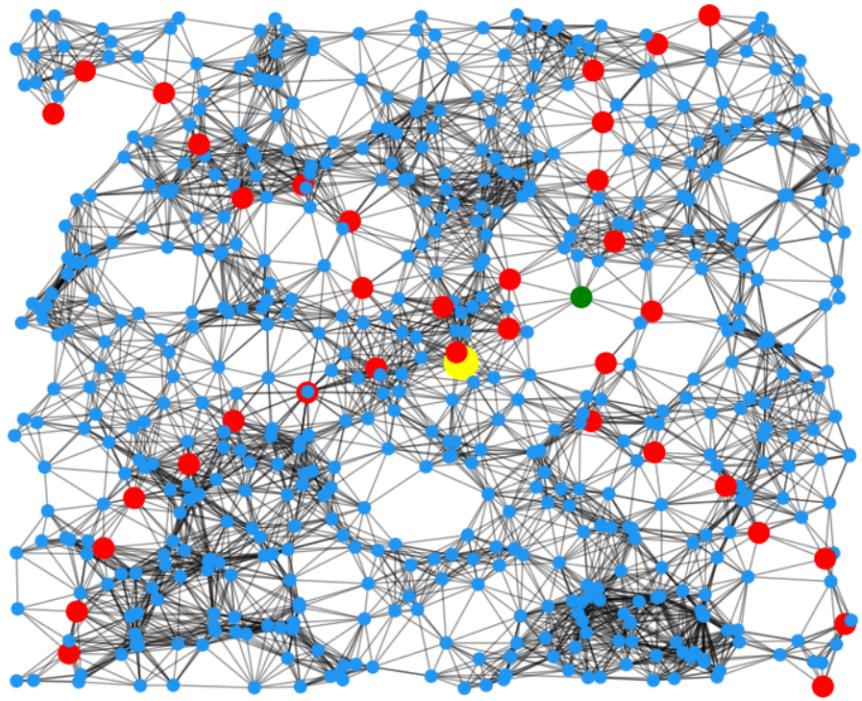


Рисунок 3 – Процесс поиска самой удаленной вершины (желтый цвет – центральная вершина, зеленый цвет – начальная популяция, красный – вершины на путях в самые удаленные точки графа)

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  $p_c$ .

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

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 та 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.

**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**

$eccentricity[i] \leftarrow population[i].eccentricity$

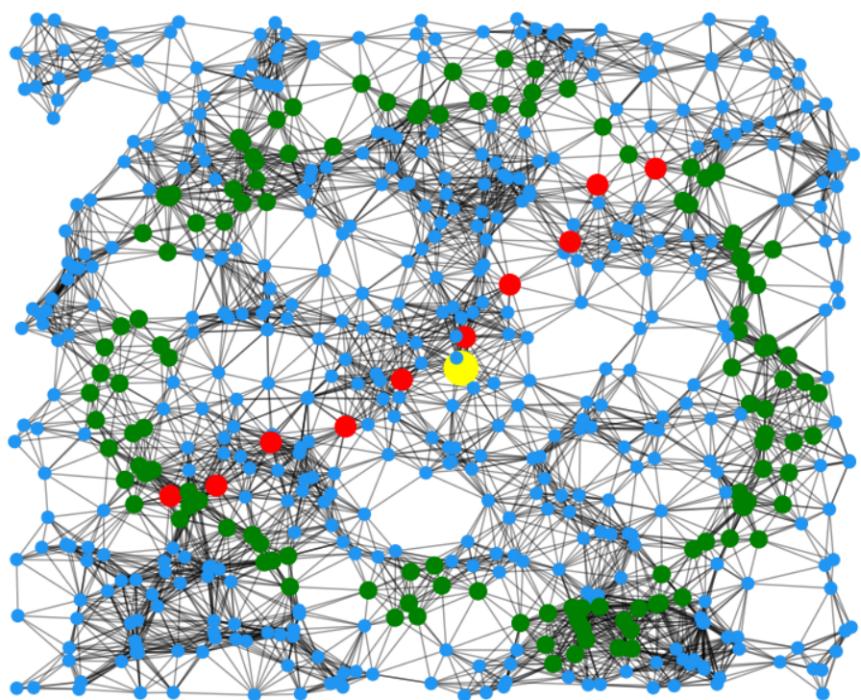


Рисунок 4 – Процесс скрещивания (желтый цвет – центральная вершина, зеленый цвет – начальная популяция, красный – кратчайший путь между вершинами популяции)

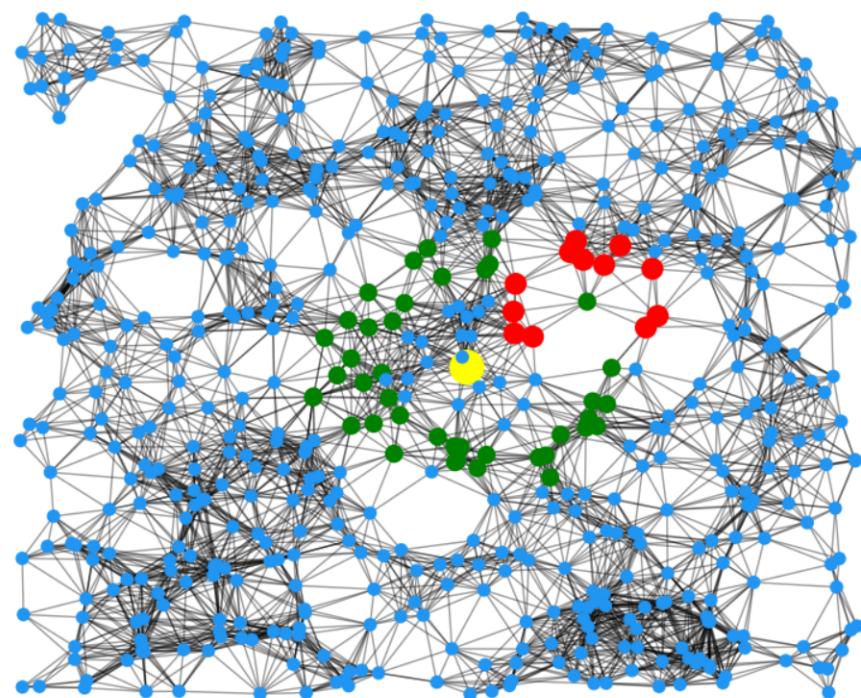
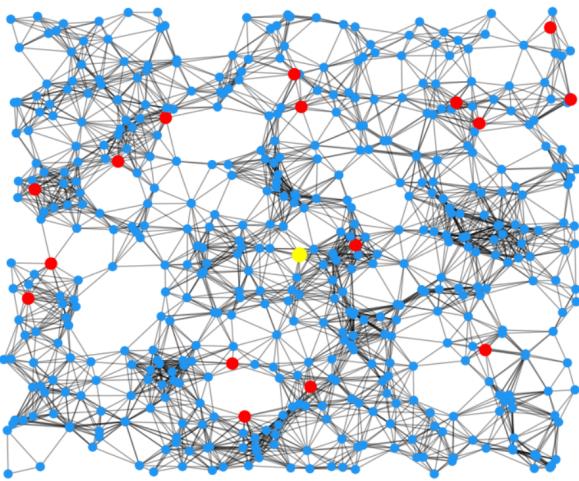
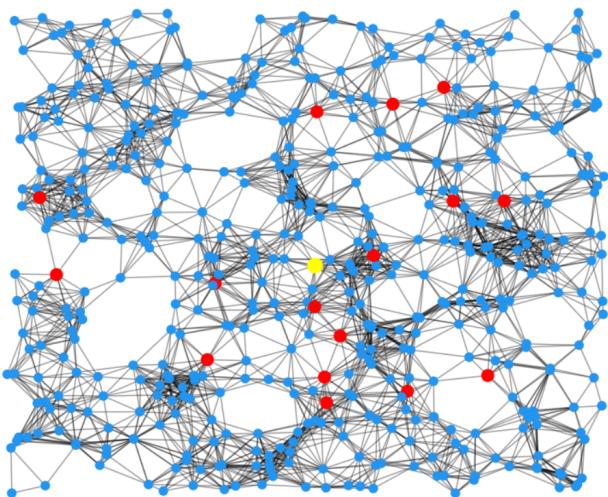


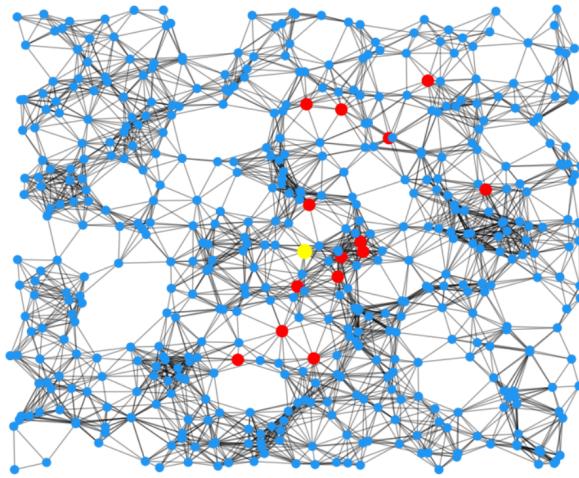
Рисунок 5 – Процесс мутации (желтый цвет – центральная вершина, зеленый цвет – начальная популяция, красный – соседние вершины одной из вершин в популяции)



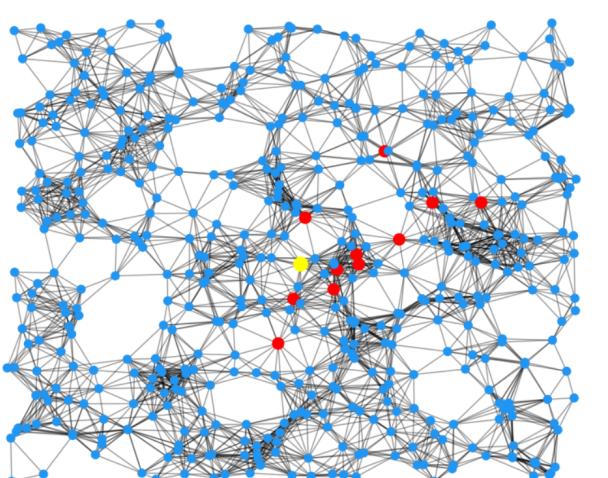
Начальная популяция



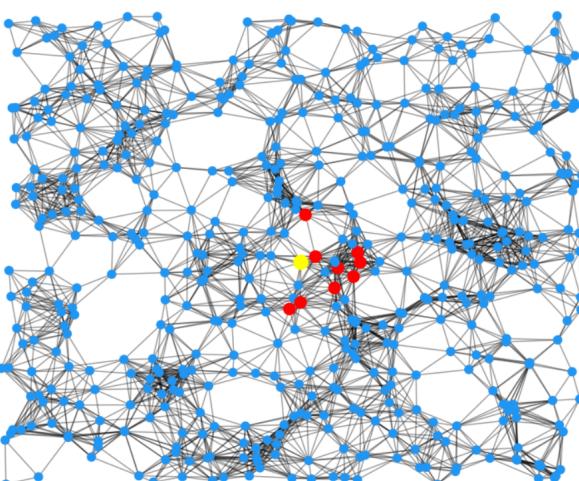
Популяция после двух шагов



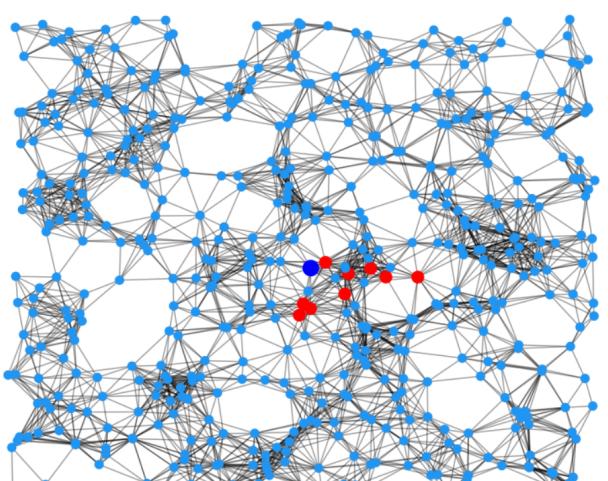
Популяция после четырех шагов



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)

## **5 Реализация алгоритма**

В качестве языка программирования для имплементации алгоритма был выбран язык программирования C++. Кроме этого в качестве среды разработки, в которой производилась реализация алгоритма, была выбрана Visual Studio 2019, 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 графов на основе различных моделей случайных графов.

## 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, такие как компьютерные или социальные сети.

### 6.1 Модель случайного графа Эрдеша-Ренъи

Данная модель представленная в работе [6] 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 график в рассматриваемой задаче близок по свойствам с графиком Эрдеша-Ренъи, то можно без труда получить много информации об изучаемом графике.

### 6.2 Модель случайного графа Барабаши-Альберт

Данная модель случайного графа была предложена в работе [7]. 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-Альберта. 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 случайнym 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  $m$  — число ребер, которые проводится из каждой новой добавляемой вершины в граф. Алгоритм создания достаточно прост, в качестве начального графа берется связный граф с числом вершин большим или равным  $m$ , после чего добавляются новые вершины до необходимого количества, при этом каждая новая вершина соединяется с  $m$  вершинами случайным образом с вероятностным распределением, описанным формулой выше.

### 6.3 Геометрический случайный граф

Данная модель случайного графа [8] 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-Ренъи и Барабаши-Альберта.

## 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-Ренъи в качестве параметра  $p$  было выбрано значение 5%, для модели Барабаши-Альберта  $m = 2$ , 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 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.

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

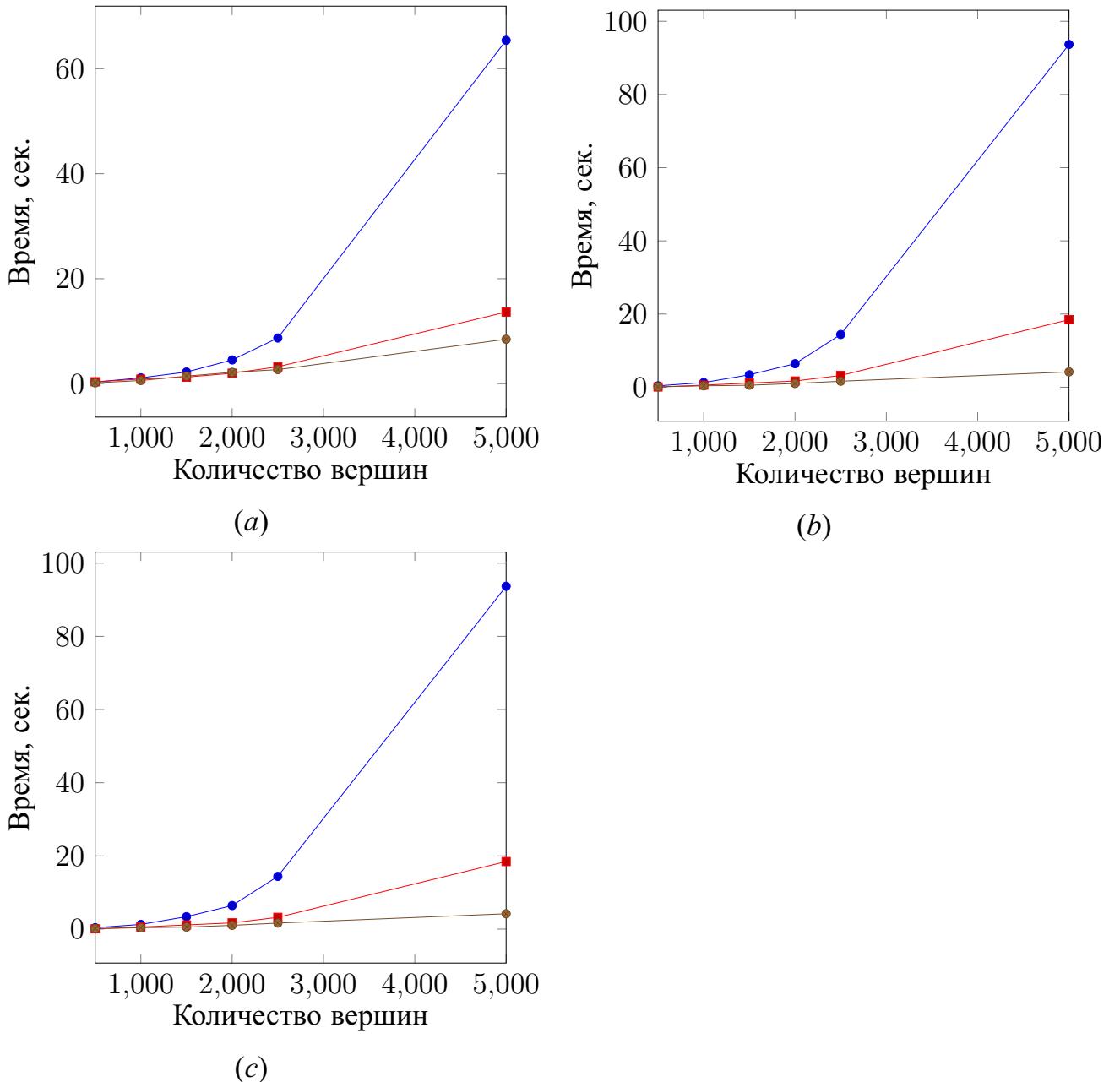


Рисунок 7 – Графики зависимости времени работы от размеров графа (синий цвет – тривиальный  $O(nm + n^2)$  алгоритм, красный – алгоритм с улучшенной асимптотикой  $O(m\sqrt{n})$ , коричневый – генетический алгоритм): (a) – модель Эрдеша-Ренъи  $p = 1\%$ , (b) – модель Барабаши-Альберта  $m = 2$ , (c)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

Таблица 1 – Время работы и процент ошибки алгоритмов на случайных геометрических графах

№	Размер графа		Время, сек.		Ошибка, %	
	N	M	Созданный алг.	N4N алг.	Созданный алг.	N4N алг.
1	500	3572	0.11	0.39	38.0	40.0
2	1000	14202	0.31	0.77	21.0	50.0
3	1500	31861	0.71	1.44	13.0	62.0
4	2000	57438	1.20	1.92	8.0	48.0
5	2500	90268	1.76	2.68	4.0	30.0
6	5000	358553	4.48	8.61	0.0	0.0
7	10000	1439255	13.54	26.0	0.0	0.0

Таблица 2 – Время работы и процент ошибки алгоритмов на модели случайного графа Барабаши-Альберта

	Размеры графа		Время, сек.		Ошибка, %	
	N	M	Созданный алг.	N4N алг.	Созданный алг.	N4N алг.
1	500	996	0.07	0.29	16.0	0.0
2	1000	1996	0.18	0.68	12.0	0.0
3	1500	2996	0.37	1.24	4.0	0.0
4	2000	3996	0.55	1.67	1.0	0.0
5	2500	4996	0.69	2.18	0.0	0.0
6	5000	9996	1.84	8.28	0.0	0.0
7	10000	19996	3.90	15.8	0.0	0.0

Таблица 3 – Время работы и процент ошибки алгоритмов на модели случайного графа Эрдеша-Ренни

	Размеры графа		Время, сек.		Ошибка, %	
	N	M	Созданный алг.	N4N алг.	Созданный алг.	N4N алг.
1	500	1288	0.16	0.44	29.0	43.0
2	1000	4905	0.60	1.30	0.0	0.0
3	1500	11153	1.44	1.90	0.0	0.0
4	2000	20201	2.17	2.79	0.0	0.0
5	2500	31187	2.68	2.94	0.0	0.0
6	5000	124658	8.47	11.0	0.0	0.0
7	10000	500471	25.4	39.3	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.

## 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.

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

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:

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

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

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

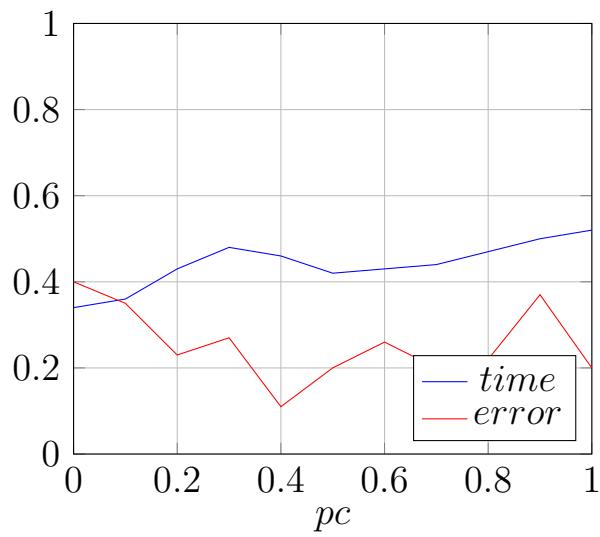


Рисунок 8 – Нормализованные значения времени выполнения алгоритма и процента ошибок с параметрами  $pm = 0.4$ ,  $|V| = 2500$ ,  $|E| = 90268$ ,  $N = 20$

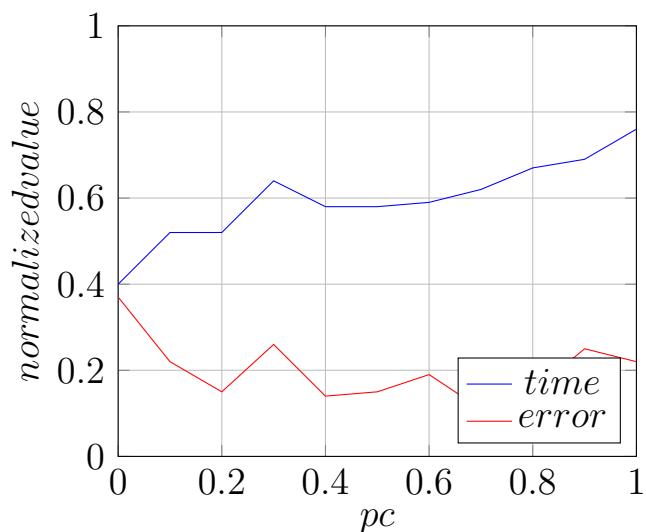


Рисунок 9 – Нормализованные значения времени выполнения алгоритма и процента ошибок с параметрами  $pm = 0.6$ ,  $|V| = 2500$ ,  $|E| = 90268$ ,  $N = 20$

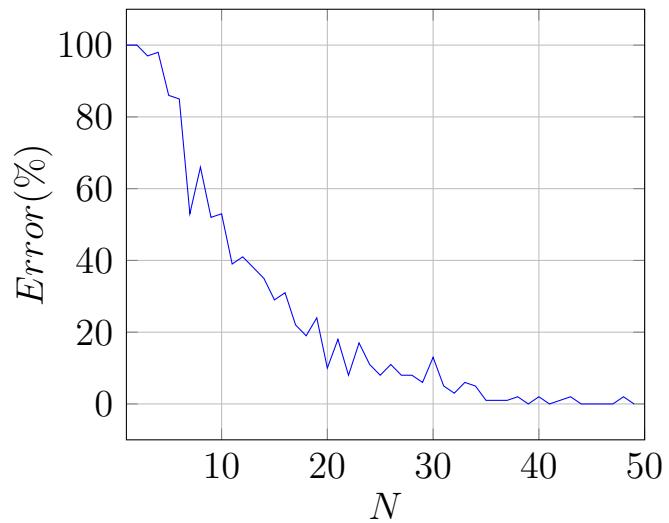


Рисунок 10 – Зависимость процента ошибок (%) от размера популяции  $N$ , с параметрами  
 $p_m = 0.4, p_c = 0.6$

## **ЗАКЛЮЧЕНИЕ**

Заключение

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

- 1 *Holland, J.* Adaption in Natural and Artificial Systems Adaption in Natural and Artificial Systems / J. Holland. — University of Michigan Press, 1975.
- 2 *Кормен, Т.* Алгоритмы: построение и анализ / Т. Кормен, . Лейзерсон, Р. Ривест, . Штайн. — Вильямс, 2019.
- 3 *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.
- 4 *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.
- 5 *Strassen, V.* Gaussian elimination is not optimal / V. Strassen // *Numerische Mathematik*. — Aug 1969. — Vol. 13, no. 4. — Pp. 354–356.
- 6 *Erdös, P.* On random graphs I / P. Erdös, A. Rényi // *Publicationes Mathematicae Debrecen*. — 1959. — Vol. 6. — P. 290.
- 7 *Albert, R.* Statistical mechanics of complex networks / R. Albert, A.-L. Barabasi // *Reviews of Modern Physics*. — 2002. — Vol. 74, no. 1. — Pp. 47–97.
- 8 *Gilbert, E.* Random plane networks / E. Gilbert // *Journal of the Society for Industrial and Applied Mathematics*. — 1961. — Vol. 9, no. 4. — Pp. 533–543.