



Московский государственный университет имени М.В.Ломоносова  
Факультет вычислительной математики и кибернетики  
Кафедра информационной безопасности

## ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

**Разработка итерационных алгоритмов поиска автоморфизмов  
и изоморфизмов комбинаторных объектов.**

*Автор:*  
группа 419

Ефремов Степан Сергеевич

*Научный руководитель:*

доцент, к.ф.- м.н.

Егоров Владимир Николаевич

Москва, 2018

# Содержание

<b>Введение</b>	<b>4</b>
<b>1 Постановка задачи</b>	<b>8</b>
<b>2 Основные определения и обозначения</b>	<b>9</b>
2.1 Определения . . . . .	9
2.2 Обозначения . . . . .	10
<b>3 Структура алгоритма</b>	<b>11</b>
3.1 Проверка по критерию . . . . .	11
3.2 Построение множеств частичных отображений . . . . .	12
<b>4 Исследование алгоритма</b>	<b>14</b>
4.1 Определение класса решаемых задач . . . . .	14
4.2 Анализ . . . . .	14
4.3 Вероятностная сложность . . . . .	17
4.4 Теоретическая возможность распараллеливания . . . . .	17
<b>5 Модернизация</b>	<b>19</b>
5.1 Оптимизация по времени работы . . . . .	19
5.2 Обработка особых случаев . . . . .	21
5.3 Ресурс параллелизма . . . . .	21
<b>6 Практическое применение</b>	<b>22</b>
<b>7 Разработка программных модулей</b>	<b>23</b>
<b>8 Результаты опробывания программ</b>	<b>24</b>
8.1 Основные результаты . . . . .	24
8.2 Масштабируемость реализации алгоритма . . . . .	25
<b>Заключение</b>	<b>28</b>
<b>Список литературы</b>	<b>29</b>

Приложение А	31
Приложение Б	33

## Введение

В дипломной работе рассматривается итерационный алгоритм поиска автоморфизмов и изоморфизмов графов, предложенный в статье В.Н.Егорова [1].

Проблема изоморфизма графов в настоящее время имеет статус одной из сложных и важных с теоретической точки зрения задач комбинаторной математики. Известно, что решение задачи нахождения изоморфного подграфа с точки зрения алгоритмической сложности позволило бы ответить на вопрос о совпадении или различии классов  $P$  и  $NP$ . В последние годы сформировались два направления изучения и решения данной проблемы:

1. Теоретическое, в котором проблема изоморфизма рассматривается с позиций современной теории сложности алгоритмов и вычислений
2. Практическое, предполагающее разработку алгоритмов, решающих задачу изоморфизма графов за «практически приемлемое» время.

Первые серьёзные результаты в теоретическом направлении появились в 50-х и 60-х годах прошлого века. Для нескольких семейств графов была установлена полиномиальная сложность нахождения изоморфизма. В 70-х годах, особенно после выхода в свет замечательной монографии Нормана Биггса «Алгебраическая теория графов», к решению этой задачи подключились крупные специалисты в области линейной алгебры, теории групп и ряда других направлений общей алгебры. Результаты не замедлили сказаться - открыто много новых интересных алгебраических свойств графов. В 1979 году Егорову В.Н. удалось получить положительное решение гипотезы Адама об изоморфизме графов с циркулянтными матрицами смежности вершин порядка свободного от квадратов [2]. В 80-х годах усилиями

большой группы математиков был получен наилучший на настоящие момент результат о том, что для графов с ограниченной валентностью вершин задача распознавания изоморфизма имеет полиномиальную сложность [3].

Алгоритмы разрабатываются не только профессиональными математиками и программистами, но и людьми, не имеющими фундаментального образования. Известно, что эта тематика в разных модификациях имеет широкий спектр приложений в таких областях:

1. Криптография.
2. Теория кодирования.
3. Информационные технологии.
4. Химия.
5. Биология.

Следует отметить, что и здесь сложились два принципиально различных подхода. Первый подход использует понятие инвариантов графа, т.е. таких функций от вершин графа или от определённых структур графа, включая и сам граф, которые не меняют своего значения при перестановках вершин графа. Примеров таких инвариантов можно привести достаточно много, причём с их помощью часто удавалось решать самые разнообразные частные задачи. Однако, полной системы инвариантов, которая позволяла бы определить наличие или отсутствие изоморфизма до настоящего времени получить не удалось. Кроме того, слабостью этого подхода является то, что эти инварианты бесполезно применять в случае, если рассматриваемые графы имеют заведомо транзитивные группы автоморфизмов.

Второй подход не связан с вычислением различных характеристик графов. Предлагаемые алгоритмы являются итерационными и определяют наличие или отсутствие изоморфизма методом направленного перебора, основной идеей которого является построение частичных отображений с последующим промежуточным критерием. Понятно, что это одна из версий метода ветвей и границ. Пожалуй, первое упоминание о таком алгоритме содержится в [4] (стр. 397-402). В настоящее время имеется несколько алгоритмов такого рода, которые авторами рекламируются как «эффективные».

Ниже приведены итерационные алгоритмы и их сложности:

Авторы	Ограничение	Сложность
Егоров В.Н., Егоров А.В. [1]	—	$O(n^2(\frac{\varepsilon}{2})^{\ln(n)^2} \ln(n))$
Babai [5]	Сильно регулярные графы	$O(\text{Exp}(2fi \log^2 n))$
Автор 3	1.12400	пум пум
Автор 4	1.12400	пум пум
Автор 5	1.12400	пум пум
Автор 6	1.12400	пум пум

Таблица 1: Алгоритмы поиска автоморфизмов

Работа разбита на следующие пункты:

1. Постановка задачи: сформулированы задачи дипломной работы.
2. Основные определения и обозначения: приводятся основные понятия, используемые в ходе работы.
3. Структура алгоритма: описывается алгоритм из статьи.
4. Исследование алгоритма: в данном пункте описаны полученные оценки сложности алгоритма, а также способы модернизации, которые уда-

лось найти в ходе исследования. Особое внимание уделяется исследованию распараллеливания вычислений программной реализации.

5. Модернизация алгоритма.
6. Практическое применение: рассказывается о различных применениях алгоритма для графов и некоторых других комбинаторных объектов. В том числе, рассматриваются способы использования алгоритма для задачи Коши.
7. Разработка программных модулей: рассматриваются ключевые моменты, на которые стоило уделить особое внимание при реализации программных продуктов. В результате получены 2 программы: упрощенный вариант с графическим интерфейсом для использования на графах с небольшим количеством вершин; вариант реализации со всеми модернизациями, упомянутыми во 2-ом пункте, для более удобного сбора информации и исследования эффективности.
8. Результаты опробывания программ: в этом пункте приводится анализ результатов, полученных при тестировании разработанных программ на различных данных, также подробно проанализированы результаты запусков программы на суперкомпьютере Ломоносов.

# 1 Постановка задачи

Целью данной работы является:

## 1. Исследование свойств алгоритма:

- Определение класса решаемых задач
- Вероятностная сложность
- Теоретическая возможность распараллеливания

## 2. Задачи, связанные с реализацией для ПК:

- Реализация в виде программы с графическим интерфейсом
- Эксперименты поиска автоморфизмов на известных графах
- Поддержка функционала нахождения изоморфного вложения графов

## 3. Задачи, связанные с реализацией для суперкомпьютера:

- Модернизация алгоритма на основе исследований
- Реализация в виде программы для запуска на суперкомпьютере
- Эксперименты поиска автоморфизмов графов на суперкомпьютере

## 4. Исследование практического применения алгоритма для задачи Коши



## 2 Основные определения и обозначения

### 2.1 Определения

В работе часто упоминается понятие изоморфизма графов, которое описано ниже. В общем случае, **изоморфизм** можно описать так: обратимое отображение (биекция) между двумя множествами, наделёнными структурой, называется изоморфизмом, если оно сохраняет эту структуру.

- **Аutomорфизм алгебраической системы** – изоморфизм, отображающий алгебраическую систему на себя.
- **Аutomорфизм графа** – отображение множества вершин на себя, сохраняющее смежность.
- **Изоморфизм графов**  $G = \langle V_G, E_G \rangle$  и  $H = \langle V_H, E_H \rangle$  – биекция между множествами вершин графов  $f: V_G \rightarrow V_H$  такая, что любые две вершины  $u$  и  $v$  графа  $G$  смежны тогда и только тогда, когда вершины  $f(u)$  и  $f(v)$  смежны в графе  $H$ .
- **Инвариант графа** – некоторое числовое значение или упорядоченный набор значений (хэш-функция), характеризующее структуру графа  $G = \langle A, V \rangle$  и не зависящее от способа обозначения вершин или графического изображения графа.
- **Регулярный (однородный) граф** – граф, степени всех вершин которого равны, то есть каждая вершина имеет одинаковое количество соседей.
- **Сильно регулярный граф**  $srg(v, k, \lambda, \mu)$  – регулярный граф  $G = \langle V_G, E_G \rangle$  с  $v$  вершинами и степенью  $k$ , для которого существуют целые

$\lambda$  и  $\mu$  такие, что:

1. Любые две смежные вершины имеют  $\lambda$  общих соседей.
2. Любые две несмежные вершины имеют  $\mu$  общих соседей.

Приложение такое то.

## 2.2 Обозначения

- $A = A^{n \times n}$  - матрица смежности графа  $G(V, E)$  размера  $n \times n$ .
- $a_{ij}$  - элементы матрицы  $A$ ,  $i, j = 1, \dots, n = |V|$ .
- Подстановка изоморфизма - биекция  $f$  вида:

$$\begin{pmatrix} a_1 & a_2 & \dots & a_n \\ f(a_1) & f(a_2) & \dots & f(a_n) \end{pmatrix}$$

- Частичная подстановка изоморфизма (частичное отображение) - подстановка вида:

$$\begin{pmatrix} a_1 & a_2 & \dots & a_k \\ f(a_1) & f(a_2) & \dots & f(a_k) \end{pmatrix}$$

, где  $k \leq n$ .

- $M_k$  - множество, состоящее из элементов  $g_k^s$ ,  $s = 1, \dots, n_k = |M_k|$ .
- $|M_k|$  - количество элементов  $g_k^s$  в множестве.
- $M'_k$  - множество элементов, которые содержатся в  $M_k$  и удовлетворяют критерию  $h$ .
- $h$  - критерий (описанный в разделе 2.2).
- $\{M'_k\}$  - последовательность множеств  $M'_k$ ,  $k = 1, \dots, n$ .

### 3 Структура алгоритма

Поставленная задача не является простой, особенно для графов, имеющих большое количество вершин. Для ее решения граф удобно представить в виде матрицы смежности. Пусть, для удобства, элементы этой матрицы состоят из единиц и нулей. Если на пересечении строки  $i$  и столбца  $j$  стоит 1(0), и  $i \neq j$  - это означает, что существует (не существует) ребро  $(i,j)$ , если  $i = j$ , вершина под номером  $i$  имеет (не имеет) петлю. При отображении одной вершины в другую, в матрице смежности меняются местами строки и столбцы соответствующих вершин. Автоморфизмом является следующее отображение:  $\hat{g}A\hat{g}^{-1} = A$ , где  $\hat{g}$  - матричный вид подстановки  $g$ . Польза такого представления графа заключается в том, что для поиска автоморфизма  $g$ , являющегося отображением множества вершин графа  $G$  на себя, достаточно проверить равенство элементов матрицы смежности до и после применения подстановки  $g$ .

Идея алгоритма заключается в построении последовательности множеств частичных отображений по матрице смежности графа. Последнее множество из этой последовательности будет содержать в себе все автоморфизмы графа.

Для описания построения такой последовательности будут использоваться следующие определения и обозначения.

#### 3.1 Проверка по критерию

Критерием  $h$  является проверка подматриц на равенство. Предположим, требуется определить, удовлетворяет ли элемент  $g_k^s$  критерию  $h$ . Для этого необходимо, чтобы элементы  $a_{ij}$ , для всех  $i, j \leq k$ , были равны соответ-

ствующим элементам  $a_{r_i r_j}$ . Подматрица  $g_k^s$  выглядит так:

$$\begin{matrix} & r_1 & r_2 & \dots & r_k \\ \begin{matrix} r_1 \\ r_2 \\ \vdots \\ r_k \end{matrix} & \begin{pmatrix} a_{r_1 r_1} & a_{r_1 r_2} & \dots & a_{r_1 r_k} \\ a_{r_2 r_1} & a'_{r_2 r_2} & \dots & a_{r_2 r_k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r_k r_1} & a_{r_k r_2} & \dots & a_{r_k r_k} \end{pmatrix} \end{matrix}$$

Если хотя бы одно из равенств не выполнено, это означает, что по данной частичной подстановке хотя бы одна вершина отобразилась в другую так, что структура графа изменилась. А так как частичная перестановка  $g_k^s$  фиксирует  $r_1, \dots, r_k$ , то структура останется измененной, что означает отображение не будет являться автоморфизмом.

### 3.2 Построение множеств частичных отображений

Описание построения  $M'_n$ .

На первом этапе рассматриваются все  $g_1^s$ , образующие множество  $M_1$ . Каждый элемент  $g_1^s$  проверяется по критерию  $h$ . Элементы, удовлетворяющие  $h$ , образуют  $M'_1$ . В каждый элемент из  $M'_1$  добавляется еще одно отображение  $(2 \rightarrow r_2)$ , т.о. происходит переход от  $g_1^s$  к  $g_2^s$ . Из каждого  $g_1^s$  получается разных  $(n - 1)$  элементов  $g_2^s$ . Всевозможные элементы  $g_2^s$  образуют  $M_2$ . Далее строится  $M'_2$ , добавляя в него только те элементы из  $M_2$ , которые удовлетворяют критерию. Аналогично получаются множества  $M_3, M'_3, M_4, \dots, M_n, M'_n$ .

Таким образом, получается последовательность множеств частичных отображений:

$$\{M'_k\}: M'_1 \subseteq M'_2 \subseteq \dots \subseteq M'_n.$$

Множество  $M'_n$  является множеством всех возможных автоморфизмов (или пустое, если их нет).

## 4 Исследование алгоритма

### 4.1 Определение класса решаемых задач

Предложенный алгоритм является универсальным. Используя различный формат вводимых данных, можно решать следующие задачи:

1. Нахождение группы автоморфизмов произвольной квадратной матрицы.
2. Нахождение левой и правой групп автоморфизмов произвольной матрицы (т.е. блок-схем, дискретных функций и т.д.).
3. Нахождение всех изоморфизмов матрицы  $A$  на матрицу  $B$ .
4. Нахождение всех изоморфных вложений матрицы  $A$  в матрицу  $B$ .
5. Поиск клик.
6. Решение задачи Коши в подстановках.
7. Нахождение группы автоморфизмов матрицы Адамара.
8. Нахождение всех изоморфизмов кода  $A$  на код  $B$ .

В рамках дипломной работы рассматриваются первые шесть задач.

### 4.2 Анализ

Алгоритм применим для любых графов, но для удобства рассматриваются случайные графы, матрицы смежности которых состоят из нулей и единиц. В этом случае получена оценка сложности алгоритма в среднем.

Так как граф случайный, элементы матрицы смежности графа равны 1 или 0 с вероятностью  $\frac{1}{2}$ . Необходимо вычислить наиболее вероятные размеры для каждого множества  $M'_k$ .

Если рассмотреть построение множеств, не учитывая промежуточного критерия, то на  $k$ -ом шаге множество увеличивается в  $(n - k)$  раз (так как для каждого элемента  $g_{(k-1)}^s = (r_1, \dots, r_{k-1})$  добавляется  $r_k$  из оставшихся  $(n - k)$  вариантов):  $|M_{k+1}| = |M_k|(n - k) = n(n - 1) \dots (n - k)$ .

С учетом критерия получается:

1. Из построения множества  $M'_1$  следует  $|M'_1| \approx \frac{|M_1|}{2}$  (элемент  $a_{11} = a_{r_1 r_1}$  с вероятностью  $\frac{1}{2}$ ).

2. На  $(k+1)$ -ом шаге требуется совпадение  $((k+1) + (k+1) - 1)$  элементов.

Так как граф случайный (матрица с равновероятным распределением 0 и 1), то на этом шаге  $|M'_{k+1}| = \frac{1}{2^{2m+1}} |M_k|$  элементов.

Учитывая построение  $\{M'_k\}$  и пункты 1, 2 получим

$|M'_{k+1}| \approx \frac{n!}{(n-k-1)! 2^{(k+1)^2}} \approx \frac{1}{e^{k+1}} \frac{n^{n+1/2}}{(n-k-1)^{(n-k-1/2)} 2^{(k+1)^2}}$ . Последнее равенство получено используя формулу Стирлинга:  $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$ , при большом  $n$ .

Далее необходимо рассмотреть последовательность  $\{|M'_{k+1}|\}$ .

$$|M'_{k+1}| = \frac{n(n-1)\dots(n-k)}{2^{(k+1)^2}} \leq \frac{n^{k+1}}{2^{(k+1)^2}}$$

Равенство  $n^{k+1} = 2^{(k+1)^2}$  означает, что в множестве осталось небольшое количество элементов. Данное равенство далее будет называться стабилизацией последовательности  $\{M'_k\}$ , а  $k$  - значением стабилизации.

После решения данного равенства, получается, что стабилизация наступает при  $k \approx \log_2(n)$ . Это означает, что с вероятностью близкой к единице (для случайного графа) уже на  $k = \lceil \log_2(n) \rceil$  шаге мы обнаружим отсутствие или наличие автоморфизма (в статье это выдвигается как тезис).

Вычисление номера множества  $k$  в последовательности  $\{|M'_k|\}$ , соответствующий самому большому множеству:

$$\text{Пусть } |M'_{k+1}| : |M'_{k+1}| = f(k+1),$$

$$f'(k+1) = \frac{\ln(n)n^{k+1}2^{(k+1)^2} - (2k+2)\ln(2)2^{(k+1)^2}n^{k+1}}{(2^{(m+1)^2})^2}.$$

Получается уравнение:

$$\ln(n) - (2k+2)\ln(2) = 0 \Rightarrow k+1 \approx \frac{5}{7}\ln(n) - 1.$$

Так как  $k$  и  $n$  целые, то

$$k+1 = \lceil (\frac{5}{7}\ln(n) - 1) \rceil.$$

Необходимо отметить, что в тот момент, когда  $k \approx \log_2(n)$ , можно судить о том, существуют автоморфизмы в графе или нет. Если размер множества  $M_k = 1$  (только тождественная подстановка), можно утверждать с вероятностью близкой к 1, что автоморфизмов, кроме тривиальной подстановки, не существует. Если  $M_k > 1$ , то, наоборот, с вероятностью близкой к 1 автоморфизм существует. Данный факт позволяет сэкономить память и уменьшить количество операций при поиске автоморфизмов в тех графах, в которых они существуют.

Опираясь на вышесказанное, получены оценки:

- значение стабилизации

$$k \approx \log_2(n)$$

- номер наибольшего по количеству элементов множества

$$k \approx \frac{5}{7}\ln(n)$$

- ограничение на размер множества

$$|M'_k| = \frac{n(n-1)\dots(n-k-1)}{2^{(k)^2}} \leq \frac{n^k}{2^{(k)^2}}$$

- приближительное количество операций в секунду:



$$\frac{n^{(5/7) \ln(n)}}{2^{((5/7) \ln(n))^2}} \times n(2(\frac{5}{7} \ln(n)) + 1) \times \log_2(n)$$

- затраты оперативной памяти:

$$2 \times \frac{n^{(5/7) \ln(n)}}{2^{((5/7) \ln(n))^2}} \times \log_2(n)$$

### 4.3 Вероятностная сложность

Сложность всего алгоритма представляет собой сумму сложностей вычисления  $M'_1 \dots M'_n$ .

Количество элементов в каждом множестве (с вероятностью близкой к единице):

$$|M'_k| \approx \frac{n!}{(n-k-2)! 2^{(k)^2}} \approx \frac{1}{e^k} \frac{n^{n+1/2}}{(n-k-2)^{(n-k-3/2)} 2^{(k)^2}}$$

Можно оценить:  $|M'_k| = \frac{n(n-1) \dots (n-k-1)}{2^{(k)^2}} \leq \frac{n^k}{2^{(k)^2}}$

Количество операций сравнения для подсчета всех множеств:

$$\sum_{k=1}^n \frac{1}{e^k} \frac{n^{n+1/2}}{(n-k-2)^{(n-k-3/2)} 2^{(k)^2}} \times (n-k)(2k+1)$$

Оценка (были учтены точка стабилизации и точка максимума последовательности множеств):

$$n \times \frac{n^{(5/7) \ln(n)}}{2^{((5/7) \ln(n))^2}} \times n(2(\frac{5}{7} \ln(n)) + 1) \times \log_2(n) = O(n^2(\frac{e}{2})^{\ln(n)^2} \ln(n))$$

Сложность экспоненциальная.

### 4.4 Теоретическая возможность распараллеливания

Каждое множество  $M_i (i = 2 \dots n)$  вычисляется из  $M'_{i-1}$  путем добавления в каждый элемент (частичная перестановка, представленная вектором целых чисел) из  $M'_{i-1}$  всех возможных оставшихся чисел, которых нет в соответствующем элементе.

Для этого удобно использовать 2 не фиксированных по количеству строк двумерных массива размера  $t$  на  $n$  ( $t$  может увеличиваться), хранящих

частичные отображения, такого вида:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1i} & ? & \dots & ? \\ a_{21} & a_{22} & \dots & a_{2i} & ? & \dots & ? \\ \dots & & & & & & \\ a_{k1} & a_{k2} & \dots & a_{ki} & ? & \dots & ? \\ ? & ? & \dots & ? & ? & \dots & ? \\ \dots & & & & & & \\ ? & ? & \dots & ? & ? & \dots & ? \end{pmatrix}$$

? - эти значения не известны и заполняются постепенно.

Один массив для  $M'_{i-1}$ , другой для вычисления из предыдущего  $M_i$ . Из каждой строки множества  $M'_{i-1}$  (первый массив), в соответствие с алгоритмом, получается  $n$  новых строк длины  $i$ , которые записываются во второй массив (расширяется, если нужно).

После этого, второй массив копируется в первый и запускается цикл по всем векторам. Вектор исключается из массива, если  $a_{ji}$  равен какому-либо числу в данном векторе или вектор с  $a_{ji}$  является частичной перестановкой, не удовлетворяющей критерию.

## 5 Модернизация

### 5.1 Оптимизация по времени работы

На основе результатов тестирования программы и анализа выяснилось, что эффективность алгоритма тесно связана с тем, как нумеруются вершины графа. Другими словами, время работы программы зависит от того, какой матрицей смежности (из многих) представляется граф.

Модернизация заключается в том, что на каждом этапе можно требовать, чтобы мощность множества частичных отображений была минимальна. Однако представить матрицу смежности нужным образом не представляется возможным при больших размерах, и время работы, затрачиваемой на это, превышает время работы алгоритма. Например, при  $n = 100$  потребуется всего лишь 100 операций, чтобы выяснить, с какой вершины эффективнее всего начать отсчет на первой итерации. Но для того, чтобы получить выгоду на второй итерации, уже требуется более 10 000 операций [3]. Поэтому оптимальным является уменьшить только начальное множество частичных отображений.

Для получения первого множества, мощность которого будет наименьшей, в изначальной матрице меняется порядок строк/столбцов (переименование вершин), а именно, необходимо поменять строки и соответствующие им столбцы таким образом, чтобы на месте первого элемента главной диагонали стояло то значение, которое встречается меньше всех других на диагонали. То есть, если на главной диагонали 70% нулей и 30% единиц, необходимо поместить на первую позицию единицу. Эффективность данной модификации исходит из того, что первая итерация алгоритма составляет множество из тех номеров строк (столбцов), в которых значение

на главной диагонали совпадает с первым элементом главной диагонали. Значит, выбрав на эту позицию наименьший по количеству встречаний на диагонали элемент, получается наименьшее по мощности множество.

Оценки получены для случайных матриц (вероятность нуля и единицы в каждой позиции одинакова и равна  $\frac{1}{2}$ ). Предположим, что в матрице  $n \times n$  на главной диагонали находится  $k$  нулей, где  $k \leq \frac{1}{2}n$  (если количество нулей больше половины, то за  $k$  обозначается количество единиц). Тогда, если на первое место главной диагонали выбирать элемент случайным образом, получим, что математическое ожидание размера полученного множества будет  $\frac{k}{n} * k + \frac{n-k}{n} * (n - k)$ . В модифицированном алгоритме всегда будет получаться  $k$ . Таким образом, улучшение составляет  $\frac{\frac{k}{n} * k + \frac{(n-k)}{n} * (n-k)}{k} = 2\frac{k}{n} + \frac{n}{k} - 2$  раз. В случае диагонали, состоящей из одних нулей (единиц), то есть  $k = 0$ , улучшение равняется 1 (мощность множеств одинакова). Для получения полной оценки необходимо посчитать матожидание улучшения для произвольного числа нулей и единиц на главной диагонали. Вероятность  $k$  нулей составляет  $\frac{C_n^k}{2^n}$ . Тогда матожидание улучшения  $\frac{1}{2^{n-1}} + 2 \sum_{i=1}^{\lceil \frac{n}{2} \rceil} \frac{C_n^k}{2^n} * (2\frac{k}{n} + \frac{n}{k} - 2)$ . Далее в таблице приведены значения для различных  $n$ :

Количество вершин графа	Коэф. уменьшения мощности
4	2.12500
8	2.11198
14	1.69565
20	1.51723
50	1.27697
100	1.18312
200	1.12400

## 5.2 Обработка особых случаев

## 5.3 Ресурс параллелизма

Для поиска автоморфизмов матрицы порядка  $n$  итерационным алгоритмом в параллельном варианте требуется выполнить:

\*  $n$  ярусов сравнений (количество сравнений  $k = 1 \dots n : \frac{1}{e^k} \frac{n^{n+1/2}}{(n-k-2)^{(n-k-3/2)} 2^{(k)^2}} \times (n-k)(2k+1)$ )

При классификации по высоте ЯПФ, таким образом, алгоритм имеет сложность  $O(n)$ .

При классификации по ширине ЯПФ его сложность  $O(n(\frac{e}{2})^{\ln(n)^2} \ln(n))$ .

## 6 Практическое применение

## 7 Разработка программных модулей

## 8 Результаты опробывания программ

### 8.1 Основные результаты

Получена приблизительная сложность работы усовершенствованного алгоритма и в случае случайного графа, модернизация алгоритма дает неплохой результат на небольших размерах. Но эффективность стремительно падает, при увеличении количества вершин графа.

Время работы программы в зависимости от размера графа на среднем по мощности компьютере (затраты оперативной памяти  $\approx 2 \times 10^9$  байт):

1. До 200 вершин -  $< 1$  секунды.
2. В пределах 300 вершин - около 2 секунд.
3. 400 — 500 вершин - до 20 секунд.

Приблизительная оценка времени работы программы на мощной вычислительной технике: оперативная память  $10^{14}$  байт, количество операций в секунду  $10^{15}$  [5].

1. 1000 вершин -  $< 1$  секунды ( $10^{12}$  операций).
2. 3000 вершин -  $< 5$  секунд ( $5 \times 10^{15}$  операций).
3. 10000 вершин - около 2,5 часов ( $10^{19}$  операций).
4. 100000 вершин - около 320000 лет ( $10^{28}$  операций).

В последнем случае требуется  $10^{21}$  байт оперативной памяти.



## 8.2 Масштабируемость реализации алгоритма

Исследование проводилось на суперкомпьютере "Ломоносов"[6]

Набор и границы значений изменяемых параметров запуска реализации алгоритма:

\* число процессоров  $[4 : 128]$  с шагом  $2^n$  (точки отображены с шагом 4, усреднив результаты); \* размер матрицы  $[300 : 1000]$ .

В результате проведённых экспериментов был получен следующий диапазон алгоритма:

\* минимальная эффективность реализации 17,44\* максимальная эффективность реализации 88,73

На следующих рисунках приведены графики производительности и эффективности выбранной реализации поиска автоморфизмов графов в зависимости от изменяемых параметров запуска.

Оценки масштабируемости выбранной реализации: \* По числу процессов: -0,00837. \* По размеру задачи: 0,04612. \* По двум направлениям: 0,00253.

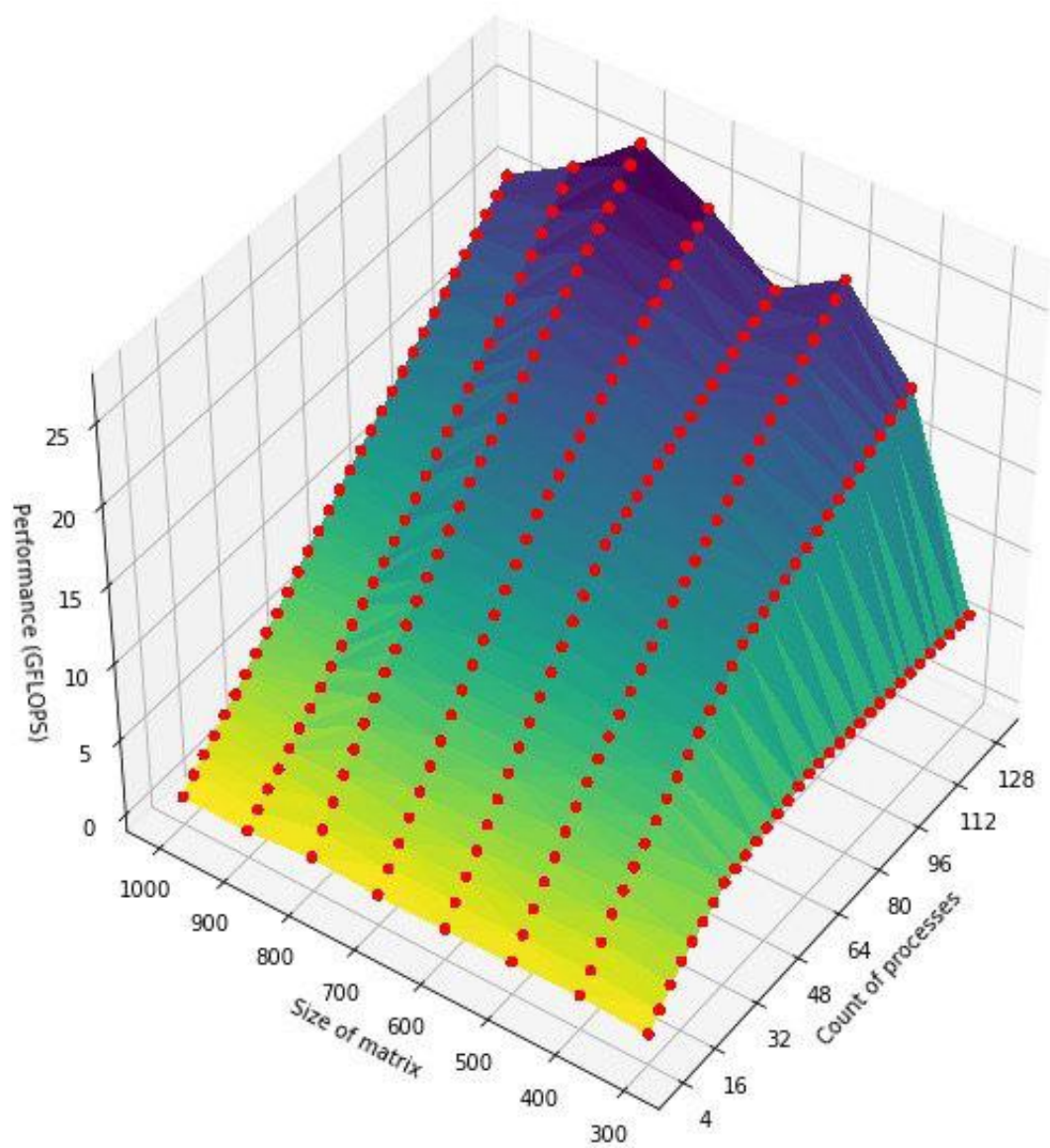


Рис. 1: Параллельная реализация алгоритма поиска автоморфизмов. Изменение производительности в зависимости от числа процессоров и размера матрицы.

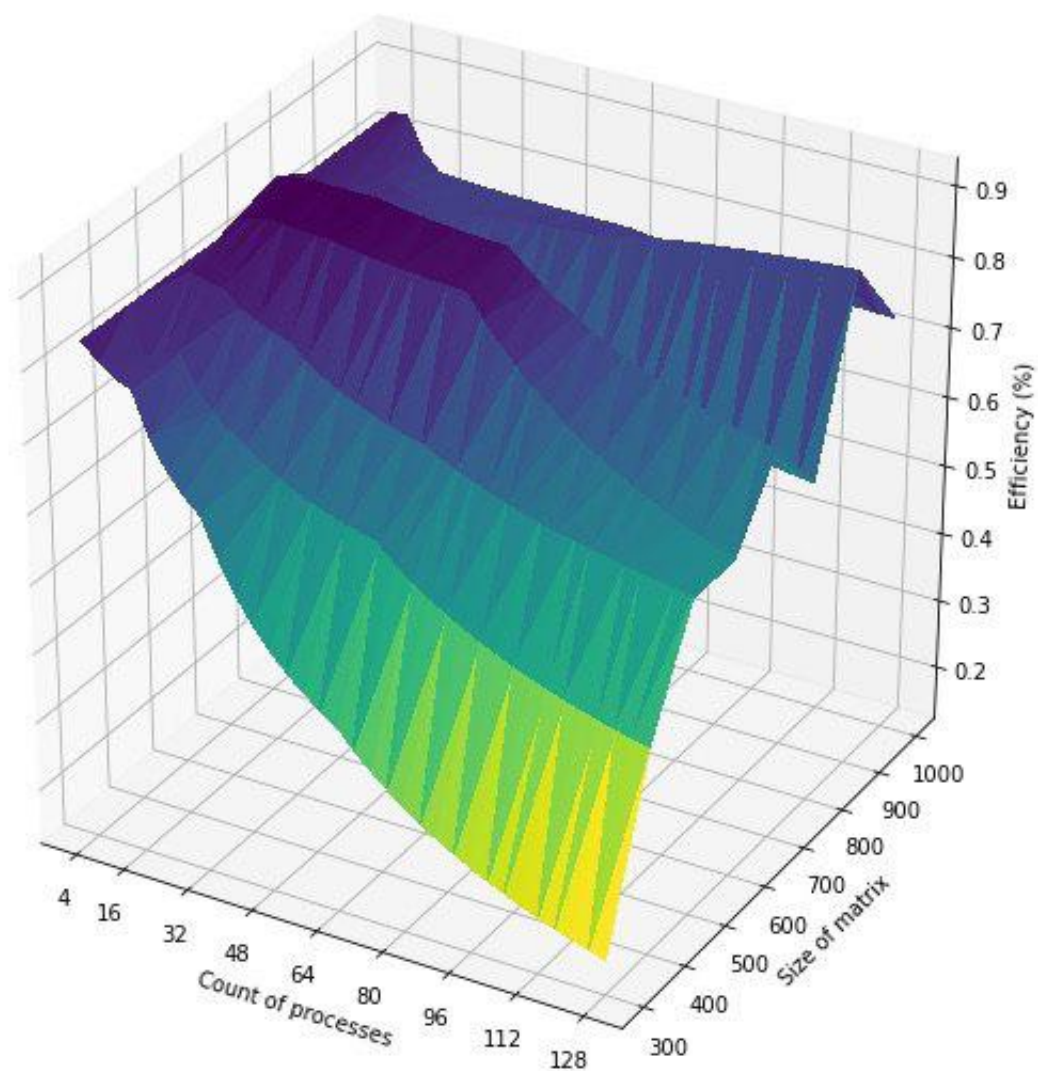


Рис. 2: Параллельная реализация алгоритма поиска автоморфизмов. Изменение эффективности в зависимости от числа процессоров и размера матрицы.

## Заключение

В результате данной работы разработан модернизированный итерационный алгоритм поиска автоморфизмов и изоморфизмов графов, основанный на алгоритме, предложенном в статье [1]. Получены оценки сложности алгоритма при больших размерах графа (количество вершин  $n > 1000$ ).

Реализована программа, решающая поставленную задачу на персональном компьютере, при  $n \leq 500$ , а также вычислены значения затрачиваемых ресурсов, при работе программы на суперкомпьютере (при  $n \leq 10^4$ ).

## Список литературы

- [1] *Егоров В.Н., Егоров А.В.* Группы автоморфизмов и изоморфизм комбинаторных объектов и алгебраических структур // *Ломоносовские чтения, Научная конференция, Москва, факультет ВМК МГУ имени М.В.Ломоносова.* — 2016.
- [2] *Егоров В.Н., Марков А.И.* О гипотезе Адама для графов с циркулянтными матрицами смежности вершин ДАН СССР // *Доклады Академии наук.* — 1979. — Т. 249, № 3. — С. 529–532.
- [3] *Luks Eugene M.* Isomorphism of graphs of bounded valence can be tested in polynomial time // *Journal of Computer and System Sciences.* — 1982. — Pp. 25:42–64.
- [4] *Рейнгольд Э., Нивергельт Ю., Део Ю.* Комбинаторные алгоритмы. Теория и практика. — Мир, Москва, 1980.
- [5] *Babai Laszld.* On the complexity of canonical labeling of strongly regular graphs // *SIAM, J. Comput.* 9(1). — 1980.
- [6] Практика суперкомпьютера «Ломоносов» / Вл. Воеводин, С. Жуматий, С. Соболев и др. // *Открытые системы.* — 2012.
- [7] *Goldberg M.K.* A nonfactorial algorithm for testing isomorphism of two graphs // *Discrete Appl. Math.*, 6. — 1983. — P. 229–236.
- [8] *Егоров В.Н.* О группах автоморфизмов матриц // *Прикладная дискретная математика.* — 2010.
- [9] Алгоритмы: построение и анализ. / Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн.

- [10] *Воеводин В.В.* Математические основы параллельных вычислений. — Изд. Моск. ун-та, 1991.
- [11] *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. — БХВ - Петербург, 2002.

## Приложение А

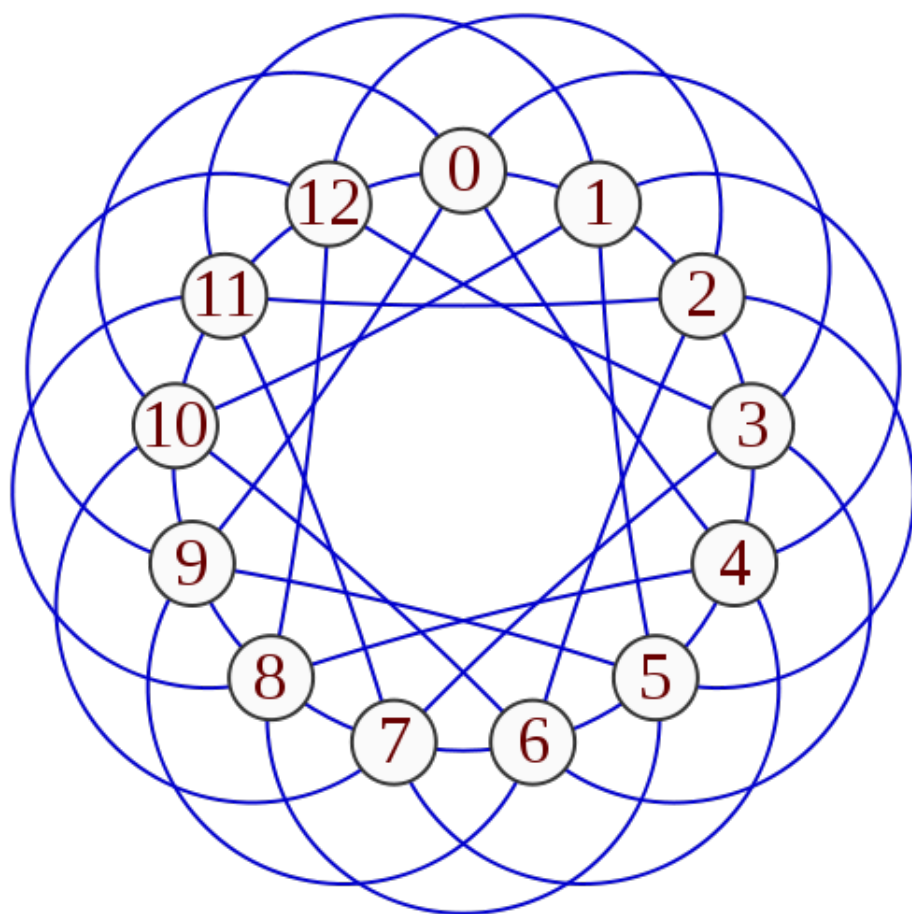


Рис. 3: Граф Пейли 13-го порядка, сильно регулярный граф с параметрами  $\text{srg}(13, 6, 2, 3)$ .

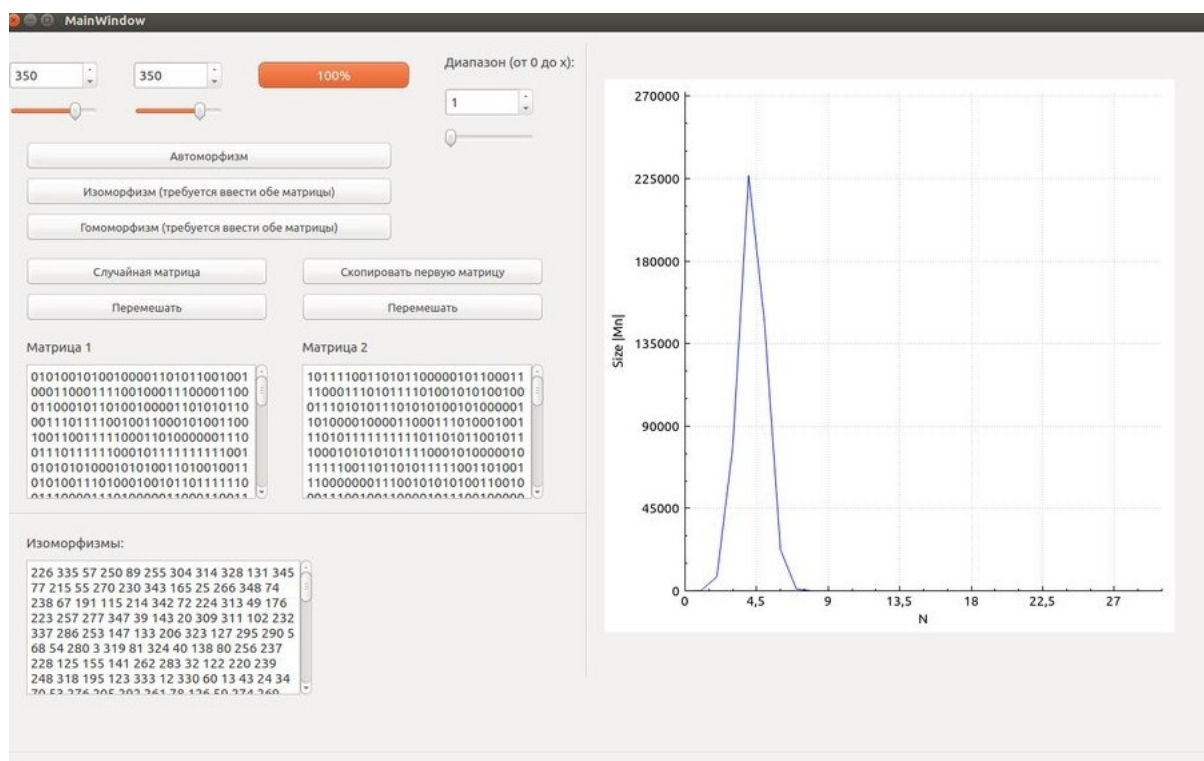


Рис. 4: Примеры использования программы



## Приложение Б

Ссылка на дипломную работу с программой на github:  
<https://github.com/fullincome/university>

Схема реализации представлена на языке C++

```
#define GRAPH_SIZE 300
#define ROW_ARR_SIZE 10000
#define COL_ARR_SIZE GRAPH_SIZE

const int N = GRAPH_SIZE;

vector<array<unsigned short, COL_ARR_SIZE>>
Mi(ROW_ARR_SIZE, COL_ARR_SIZE);
vector<array<unsigned short, COL_ARR_SIZE>>
M'i(ROW_ARR_SIZE, COL_ARR_SIZE);
unsigned char **matrix;

struct _M' i make_M' i(Mi) {
    struct _M' i M' i;
    for (auto x: Mi) {
        for (int i = 0; i < n; ++i) {
            if (check_h(x) && !eq(x, i)) {
                M' i.push_back(x.add(i));
            }
        }
    }
}
```

```

        return M' i;
    }

struct _Mi make_Mi(M' i) {
    struct _Mi Mi;
    for (auto x: M' i) {
        for (int i = 0; i < n; ++i) {
            Mi.push_back(x.add(i));
        }
    }
    return Mi;
}

int main() {

    // fill matrix
    init_automorphisms(matrix);

    //parallelization of program
    switch(process) {
    case 0:
        begin = 0;
        end = stop_0;
    case 1:
        begin = start_1;
        end = stop_1;

```

```

...
case N:
    begin = start_n;
    end = M_0.size ();

//main cycle
for (i = begin; i < end; ++i) {
    Mi = make_Mi(M' i - 1);
    M' i = make_M' i (Mi);
}

//results save (automorphisms output)
final_automorphism(M' i);
}

```