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

Ефремов Степан Сергеевич

Разработка итерационных алгоритмов поиска автоморфизмов и изоморфизмов комбинаторных объектов.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
доцент, к.ф.- м.н.
Егоров Владимир Николаевич

Москва, 2018

Содержание

Аннотация

Дипломная 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 описан класс решаемых задач алгоритмом, полученны оценки сложности. А также определены способы модернизации, которые удалось найти в ходе исследования. Особое внимание уделяется исследованию расспараллеливания вычислений программной реализации.
5. Модернизация алгоритма: описаны все найденные способы модернизации.
6. Практическое применение: рассказывается о различных применениях разработанного 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 уделялось особое внимание при реализации программных модулей. В результате получены 2 программы: упрощенный вариант с графическим интерфейсом для использования на графах с небольшим количеством вершин; вариант реализации для суперкомь-

пютеров для более удобного сбора информации и исследования эффективности.

8. Результаты опробования программ: в этом пункте приводится анализ результатов, полученных при тестировании разработанных программ на различных данных, также подробно проанализированы результаты запусков программы на суперкомпьютере Ломоносов.

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 настоящий 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:

Алгоритм	**Ограничения**	**Сложность**
Полный перебор	без ограничений	$n!$
Егоров В.Н., Егоров А.В. [1]	без ограничений	$O(n^2(\frac{e}{2})^{\ln(n)^2}\ln(n))$
László Babai, Eugene M. [5]	без ограничений	$O(e^{\sqrt{n\times\log(n)}})$
Daniel A. Spielman [6]	сильно регулярные	$n^{\emptyset(n^{1/3}\log^2 n)}$
Vaibhav Amit Patel	специальный вид	$O(n^4)$

Таблица 1: Алгоритмы поиска автоморфизмов

1 Постановка задачи

Целью данной работы является:

1. Исследование свойств алгоритма:
    - Определение класса решаемых задач
    - Вероятностная сложность
    - Теоретическая возможность распараллеливания
    - Модернизация алгоритма на основе исследований
2. Задачи, связанные с реализацией для ПК:
    - Реализация в виде программы с графическим интерфейсом
    - Эксперименты поиска автоморфизмов на известных графах
    - Поддержа функционала нахождения изоморфного вложения графов
3. Задачи, связанные с реализацией для суперкомпьютера:
    - Исследование ресурса параллелизма
    - 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.

- **Автоморфизм алгебраической системы** – изоморфизм, отображающий алгебраическую систему на себя.
- **Автоморфизм 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 $f\colon V_G \to V_H$ 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.
- **Матрица смежности графа G** – квадратная матрица A размера n, в которой значение элемента a_{ij} равно числу рёбер из i-й вершины графа в j-ю вершину.
- **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 (однороодный) 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 λ и μ такие, что:

  1. Любые две смежные вершины имеют λ общих соседей.
  2. Любые две несмежные вершины имеют μ общих соседей.

2.2 Обозначения

- $\boldsymbol{A = A^{n \times n}}$ – матрица смежности графа $G(V, E)$ размера $n \times n$ с элементами a_{ij}, $i, j = 1, \ldots, n = |V|$.
- $\boldsymbol{f(u) = v}$ – отображение вершины u в вершину v.
- **Подстановка изоморфизма g_n** – биекция f вида:

$$\begin{pmatrix} u_1 & u_2 & \ldots & u_n \\ f(u_1) & f(u_2) & \ldots & f(u_n) \end{pmatrix}$$

- **Частичная подстановка изоморфизма g_k (частичное отображение)** – подстановка вида:

$$\begin{pmatrix} u_1 & u_2 & \ldots & u_k \\ f(u_1) & f(u_2) & \ldots & f(u_k) \end{pmatrix}$$

  , где $k \leq n$.

- $\widehat{\boldsymbol{g}}$ – матричный вид подстановки g.
- $\boldsymbol{\widehat{g} A \widehat{g}^{-1} = A}$ – автоморфизм графа A.
- $\boldsymbol{\widehat{g} A \widehat{g}^{-1} = B}$ – изоморфность графов A и B.

- $\boldsymbol{M_k}$ – множество, состоящее из элементов $g_k^s,\ \ s = 1, \ldots, m_k = |M_k|$.
- $\boldsymbol{|M_k|}$ – количество элементов множества M_k.
- $\boldsymbol{M'_k}$ – множество элементов, которые содержатся в M_k и удовлетворяют определенному критерию.
- $\boldsymbol{\{M'_k\}}$ – последовательность множеств M'_k, $k = 1, \ldots, n$.

3 Структура алгоритма

3.1 Входные данные

На вход алгоритму подается матрица смежности $A^{n\times n}$ графа $G(V, E)$, $n = |V|$.

3.2 Процедура построения множеств частичных отображений

Для матрицы A размера $n \times n$ строится последовательность множеств частичных отображений M'_n следующим образом. $M_0 = M'_0 = \{g_0\}$, g_0 - пустая подстановка. Для всех $i = 1 \dots n$:

1. Фиксируем подматрицу A_i размера $i \times i$.
2. Строим множество M_i:
   (a) Рассматриваем все возможные частичные подстановки g_i, полученные из подстановок g^s_{i-1} путем добавления в них отображения $f(u_i) = v_l$.
   (b) Генерируем множество M_i из подстановок g_i, для которых в отображении $f(u_i) = v_l$:
   $$l \in \{l \mid v_l \neq f(u_j), j = 1 \dots i-1\}$$
3. Строим множество M'_i (для этого убираем из множества M_i элементы g_i, которые не удовлетворяют критерию):
   (a) Применяем частичные подстановки к матрице A по отдельности: $\widehat{g}_i A \widehat{g}_i^{-1} = A^{i\times i}_{g_i}$.
   (b) Генерируем множество M'_i из частичных подстановок, для которых выполнено: $A^{i\times i}_i = A^{i\times i}_{g_i}$.

4. Если $i = n$, процедура завершена.

Таким образом, получается последовательнось множеств частичных отображений:

$$\{M'_k\} : M'_1 \supseteq M'_2 \supseteq \ldots \supseteq M'_n$$

.

Множество M'_n является множеством всех возможных частичных подстановок g_n длины n, то есть всех подстановок изоморфизма графа G на себя. Следовательно M'_n содержит все автоморфизмы.

3.3 Выходные данные

Множество подстановок изоморфизма g_n^s графа G на себя (автоморфизмы). Для всех $s = 1, \ldots, |M'_n|$, g_n^s имеет вид:

$$\begin{pmatrix} u_1 & u_2 & \ldots & u_n \\ f_s(u_1) & f_s(u_2) & \ldots & f_s(u_n) \end{pmatrix}$$

3.4 Примечания к процедуре

В случае задачи определения изоморфизма (или гомоморфизма) графов на вход падается матрица смежности $A^{n\times n}$ и матрица смежности $B^{m\times m}$. Не ограничивая общности считаем, что $n \geq m$. Тогда отличия в процедуре следующие:

- Случай изоморфизма: вместо подматрицы A_i используется подматрица B_i.
- Случай гомоморфизма: вместо подматрицы A_i используется подматрица B_i. Процедура завершается при $i = m$.

Критерием h является проверка подматриц на равенство. Предположим, требуется определить, удовлетворяет ли элемент g_k^s критерию. Для этого необходимо, чтобы элементы a_{ij}, для всех $i, j \leq k$, были равны соответствующим элементам $a_{r_i r_j}$. Подматрица $\widehat{g_k^s} A \widehat{g_k^s}^{-1} = A_{g_k^s}$ выглядит так:

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

Если хотя бы одно из равенств не выполнено, это означает, что после применения частичной подстановки к A подматрица A_i не совпадает с $A_{g_k^s}$. Но добавление отображения в частичную подстановку не изменяют подматрицу $A_{g_k^s}$, следовательно для g_n^s образованной от g_k^s $A_n \neq A_{g_n^s}$, ни для какого s. Это значит, что частичную подстановку g_k^s можно не рассматривать.

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 автоморфизомв произваольной 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 изомофризмов кода A на код B.

В рамках дипломной работы рассматриваются первые шесть задач.

4.2 Анализ

Алгоритм применим для любых графов, но для удобство рассматриваются случайные графы, матрицы смежности которых состоят из нулей и едениц. 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 $g^s_{(k-1)} = (r_1, \ldots, r_{k-1})$ добавляется r_k из оставшихся $(n-k)$ вариантов): $|M_{k+1}| = |M_k|(n-k) = n(n-1)\ldots(n-k)$.

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: $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$, при большом n.

Далее необходимо рассмотреть последовательность $\{|M'_{k+1}|\}$.
$|M'_{k+1}| = \frac{n(n-1)\ldots(n-k)}{2^{(k+1)^2}} \leq \frac{n^{k+1}}{2^{(k+1)^2}}$
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:

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

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

- приблизительное 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.

Гипотиза 1. *Сложность алгоритма не привышает*

$$O(n^2(\frac{e}{2})^{\ln(n)^2}\ln(n))$$

с вероятностью $p = 1 - \frac{1}{n}$.

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, затрачеваемой 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 (мнощьность множеств одинакова). Для получения полной оценки необходимо посчитать матожидание улучшения для произвольного числа нулей и единиц на главной диагонали. Вероятноть 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

Из описания алгоритма следует: если в начале построения последовательности множеств частичных отображений, множества имееют размер $\approx n \times (n-1) \times \ldots \times i$ для нескольких M'_i, то возможны 2 случая:

1. Граф действительно имеет много автоморфизмов, в этом случае работа алгоритма будет занимать большее время, чем обычно
2. Матрица смежности алгоритма оказалась не совсем удобной для использования, в этом случае, можно просто выбрать случайную другую матрицу смежности (например, случай двудольного графа)

5.3 Ресурс параллелизма

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

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

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

Разработанный алгоритм является универсальным и может применяться для решения большого количества задач (как минимум всех указанных в пункте 4.1).

Написанная программа с интерфейсом имеет масштабируемые функционал. На данный момент поддерживает функционал решения следующих задач:

- Поиск автоморфизмов
- Поиск изоморфизмов
- Поиск гомоморфизмов
- Решение задачи Коша в подстановках

7 Разработка программных модулей

Разработано 2 программы:

1. Приложение с графическим интерфейсом, написанное на Qt
   - не поддерживает распараллеливание
   - справляется с графами размера до 500 вершин
   - удобный анализ графов
   - помимо нахождения автоморфизмов, решает задачи изоморфизма и гомоморфизма графов
2. Программа на языки $C++$ с поддержкой $OpenMPI$
   - поддерживает распараллеливание средствами $OpenMPI$
   - помимо нахождения автоморфизмов, сохраняет много информации для анализа графов с большим количеством вершин

8 Результаты опробования 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 (5×10^{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"[7]

Набор и границы значений изменяемых параметров запуска реализации алгоритма:

1. число процессоров [4 : 128] с шагом 2^n (точки отображены с шагом 4, усреднив результаты);
2. 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.

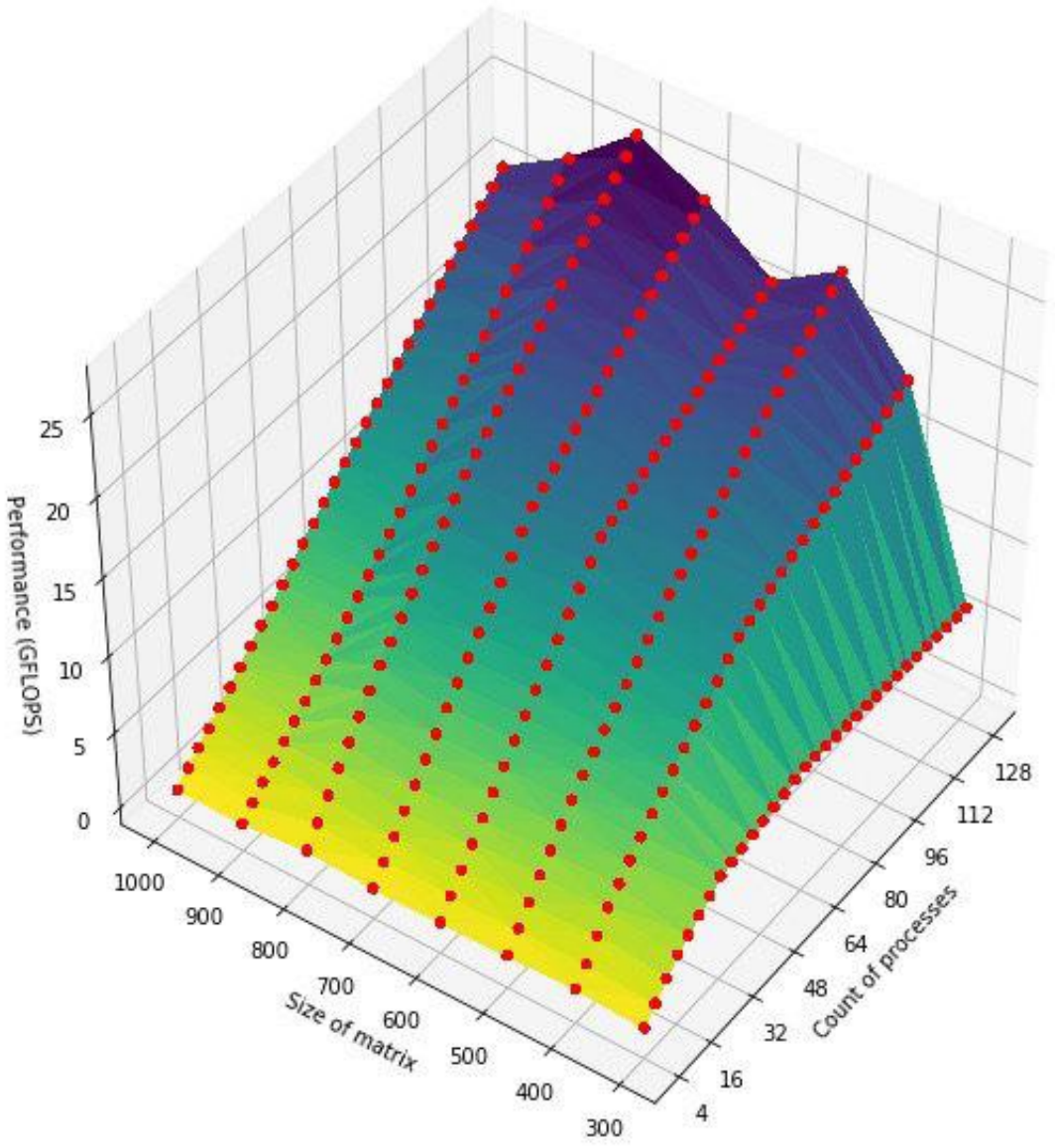

Рис. 1: Параллельная реализация алгоритма поиска автоморфизмов. Изменение производительности в зависимости от числа процессоров и размера матрицы.

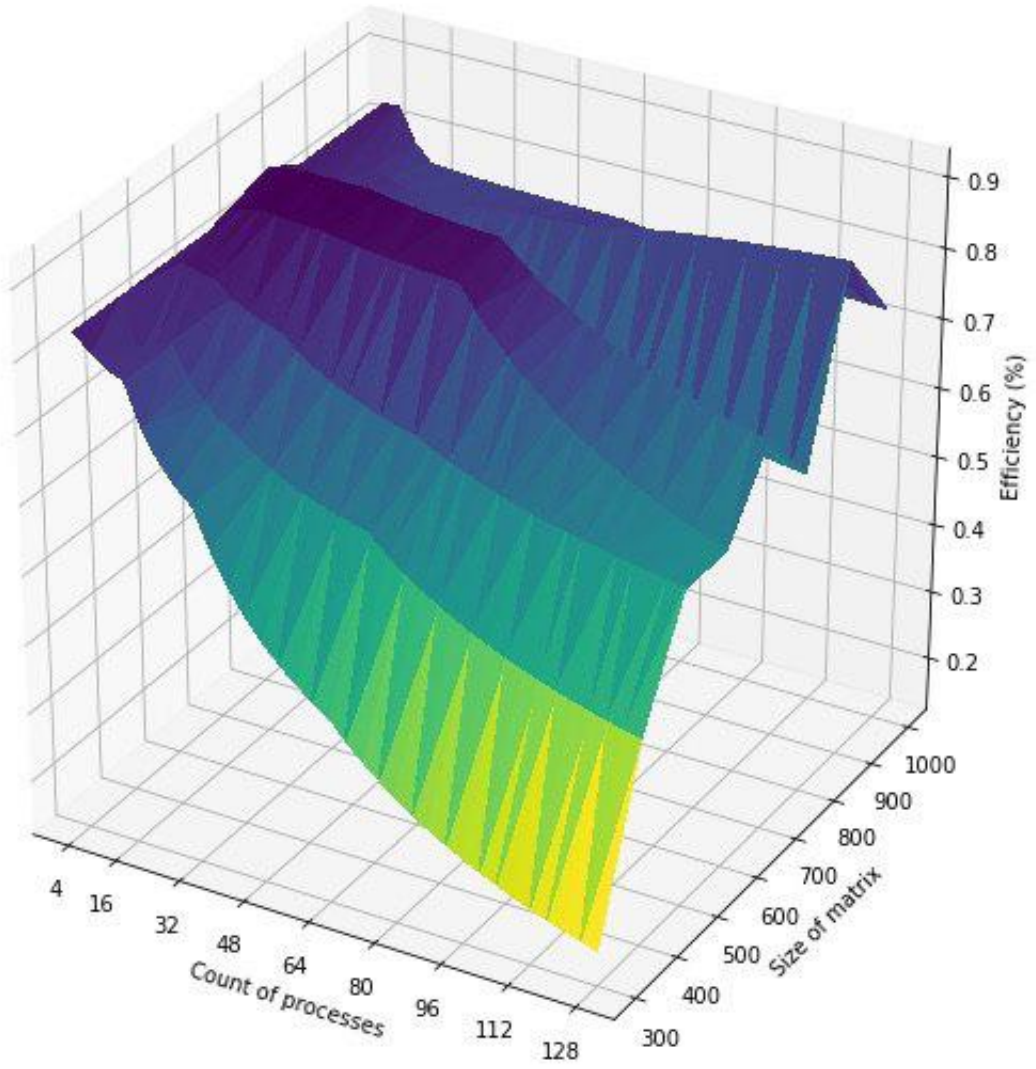

Рис. 2: Параллельная реализация алгоритма поиска автоморфизмов. Изменение эффективности в зависимости от числа процессоров и размера матрицы.

Заключение

- Выполнена модернизация алгоритма
- Сформулирована гипотеза вероятностной сложности алгоритма.

  Разработан алгоритм с вероятностной сложностью:

  $$\approx O(n^2(\frac{e}{2})^{\ln(n)^2}\ln(n))$$

  Алгоритм является универсальным и может применяться для решения большого количества задач (как минимум всех указанных в пункте 4.1 дипломной работы).
- Реализовано 2 програмы: с графическим интерфейсом для удобного использования, с консольным интерфейсом для запуска на суперкомпьютере

  Написанная программа с интерфейсом имеет масштабируемый функционал. На данный момент поддерживает функционал решения следующих задач (при $n \leq 500$):

  - Поиск автоморфизмов
  - Поиск изоморфизмов
  - Поиск гомоморфизмов
  - Решение задачи Коши в подстановках
- Проведены опыты на суперкомпьютере «Ломоносов»

  Получены оценки сложности алгоритма при больших размерах графа (количество вершин $n > 1000$).

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.* Комбинаторыне алгоритмы. Теория и практика. — Мир, Москва, 1980.

[5] *Babai László, Luks. Eugene M.* Canonical labeling of graphs. // *In Proceedings of STOC.* — 1983. — P. 171–183.

[6] *Spielman Daniel A.* Faster isomorphism testing of strongly regular graphs. // *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing.* — 1996. — Pp. 576–584.

[7] Практика суперкомпьютера «Ломоносов» / Вл. Воеводин, С. Жуматий, С. Соболев и др. // *Открытые системы.* — 2012.

[8] *Babai László.* On the complexity of canonical labeling of strongly regular graphs // *SIAM, J. Comput. 9(1).* — 1980.

[9] *Goldberg M.K.* A nonfactorial algorithm for testing isomorphism of two graphs // *Discrete Appl. Math., 6.* — 1983. — P. 229–236.

[10] *Егоров В.Н.* О группах автоморфизмов матриц // *Прикладная дискретная математика.* — 2010.

[11] Алгоритмы: построение и анализ. / Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн.

[12] *Воеводин В.В.* Математические основы параллельных вычислений. — Изд. Моск. ун-та, 1991.

[13] *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. — БХВ - Петербург, 2002.

Приложение А

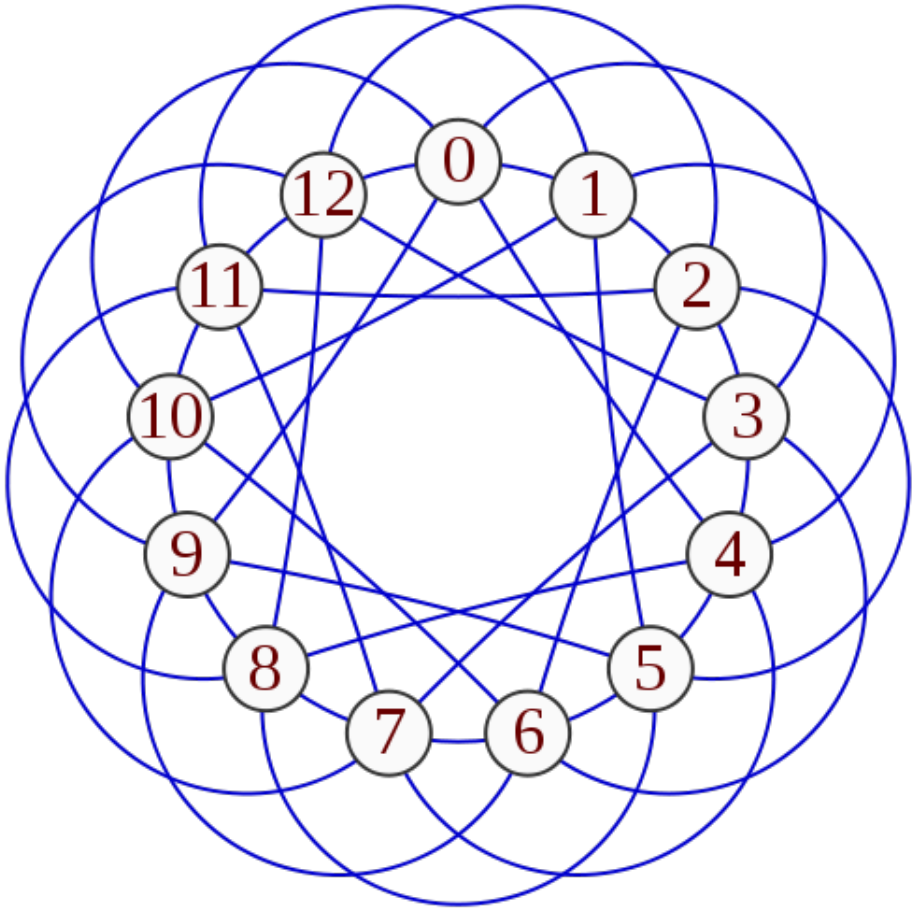

Рис. 3: Граф Пейли 13-го порядка, сильно регулярный граф с параметрами srg(13,6,2,3).

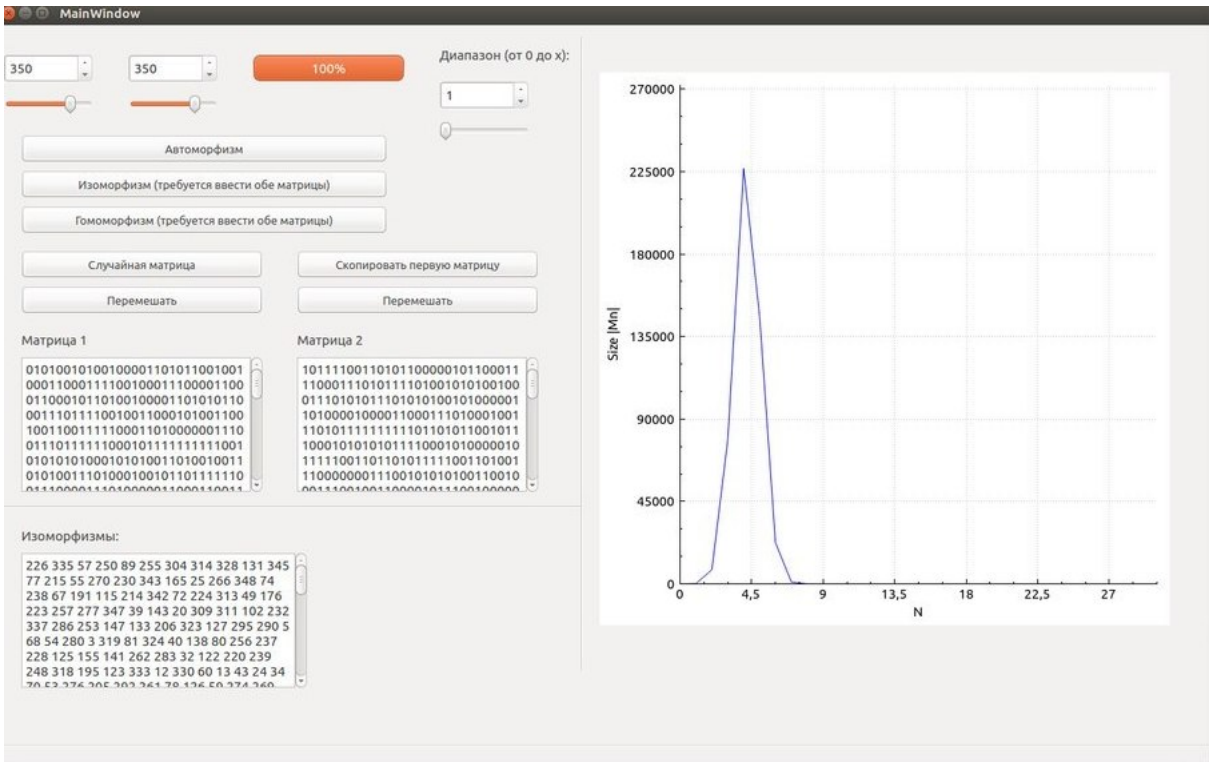

Рис. 4: Примеры использования программы

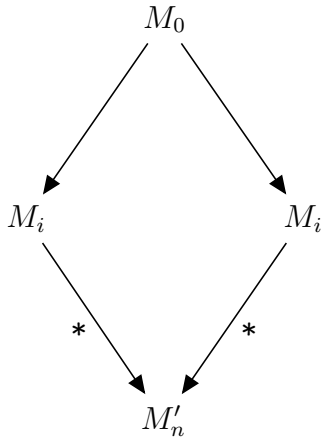

Рис. 5: Схема алгоритма

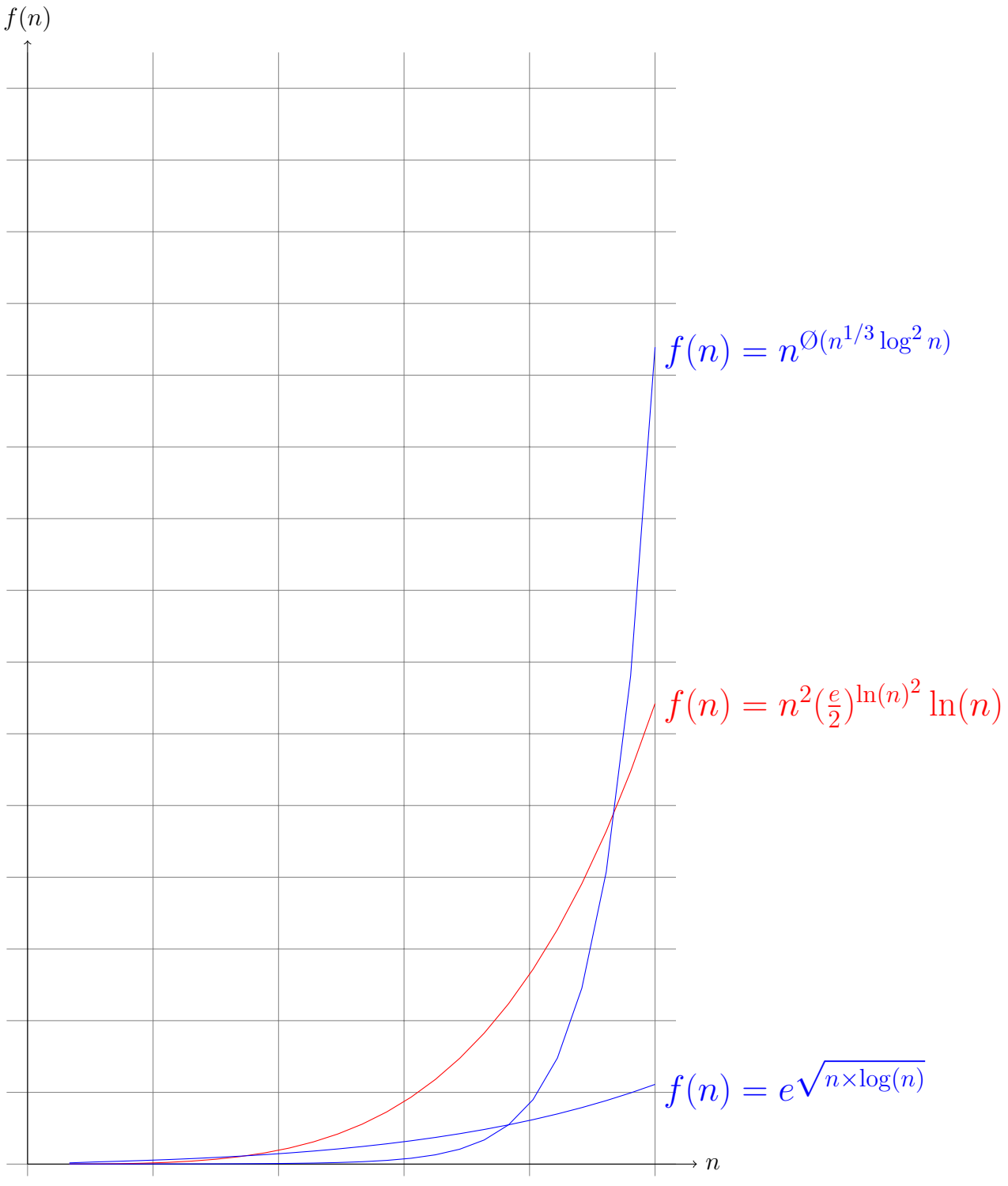

Рис. 6: График сложности алгоритма

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);
}
```