МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование кафедры)

01.04.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА
(код и наименование направления подготовки)

МАТЕМАТИЧЕСКОЕ МОДЕЛИРОВАНИЕ
(направленность (профиль))

# МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

на тему «Алгоритмизация труднорешаемой задачи размещения графа»

Студент В. А. Дудников ____________
(И.О. Фамилия) (личная подпись)

Научный руководитель О. В. Лелонд ____________
(И.О. Фамилия) (личная подпись)

Руководитель программы д.ф.-м.н., доцент, С. В. Талалов ____________
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»________________________20_____г.

**Допустить к защите**

Заведующий кафедрой к.т.н., доцент, А.В. Очеповский ____________
(ученая степень, звание, И.О. Фамилия) (личная подпись)
«_____»________________________20_____г.

Тольятти 2018

# ОГЛАВЛЕНИЕ

СОДЕРЖАНИЕ .......... 2
ВВЕДЕНИЕ .......... 4
Глава 1 АНАЛИЗ ЛИТЕРАТУРЫ .......... 9
1.1 Современная математическая модель .......... 11
1.3 Алгоритмы решения задачи размещения графа .......... 15
1.4 Выводы проведенного анализа литературы .......... 16
Глава 2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ .......... 17
2.1 Разработка математической модели .......... 17
2.1.1 Термины и определения .......... 17
2.1.2 Об NP-полноте задачи размещения графа .......... 21
2.1.3 Вспомогательные результаты .......... 23
2.2 Алгоритмы решения задачи размещения графа .......... 26
2.2.1 Алгоритм локального поиска .......... 26
2.2.2 Алгоритм имитации отжига .......... 27
2.2.3 Генетический алгоритм .......... 31
2.3 Вспомогательный метод оптимизации .......... 36
2.4 Выводы проведенного анализа предметной области .......... 39
Глава 3 РЕАЛИЗАЦИЯ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ РАЗМЕЩЕНИЯ ГРАФА .......... 41
3.1 Реализация алгоритмов решения задачи размещения графа .......... 41
3.1.1 Реализация алгоритма имитации отжига и алгоритма локального поиска .......... 41
3.1.2 Реализация генетического алгоритма .......... 42

3.1.3 Эвристический алгоритм Hebene ........................................ 44
3.1.4 Стохастический алгоритм Hebene ........................................ 46
3.2 Разработка программного обеспечения ........................................ 47
3.3 Демонстрация программного обеспечения ........................................ 52
3.4 Сбор данных для вычислительных экспериментов ........................................ 55
3.4.1 Генерация случайных графов ........................................ 56
3.4.2 Получение попарно неизоморфных графов ........................................ 59
3.4.3 Способы хранения большого количества графов ........................................ 61
3.5 Выводы ........................................ 69
Глава 4 ПРОВЕДЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ ЭКСПЕРИМЕНТОВ ...... 71
4.1 Сравнение скорости работы алгоритмов ........................................ 71
4.2 Сравнение поисковых способностей алгоритмов ........................................ 74
4.2.1 Сравнение поисковых способностей алгоритмов на графах малых порядков ........................................ 74
4.2.2 Сравнения поисковых способностей алгоритмов для графов больших порядков ........................................ 76
4.3. Некоторые вспомогательные вычислительные эксперименты ....... 84
4.3.1 Оценка решений графов на отклонение от оптимальных решений ........................................ 84
4.3.2 Пример размещения графа в сетке ........................................ 85
4.4 Выводы проведенных вычислительных экспериментов ........................................ 86
ЗАКЛЮЧЕНИЕ ........................................ 88
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ ........................................ 89

# ВВЕДЕНИЕ

**Актуальность работы**

Работа посвящена алгоритмизации задачи размещения графа. Приложением задачи размещения графа является задача размещения элементов СБИС.

Эта задача может помочь уменьшить время проектирования и достижения желаемых электрических параметров микросхемы.

Кроме того, алгоритмизация задачи размещения графа может помочь в решении других труднорешаемых задач, таких как задача коммивояжера, задача поиска гамильтонова цикла и т. д., так как они имеют много общего.

**Проблема**

На текущий момент для задачи размещения графа не существует эффективных алгоритмов ее решения. Это, прежде всего, связано с тем, что задача является труднорешаемой. Для решения задачи размещения графа применяют мультиэвристические алгоритмы такие, как генетические алгоритмы и другие. Применение этого класса алгоритмов вызывает трудности в эффективном исследовании этих алгоритмов, так как они в основном являются стохастическими и рассматривают задачу без ее конкретной структуры.

Кроме того, на данный момент неизвестно является ли задача размещения графа NP-полной, а лишь известно, что нет алгоритмов решающих ее эффективно.

Также не существует оценок, которые могли бы дать возможность оценивать решения для задачи размещения графа.

**Объект и предмет исследования**

Объектом исследования является задача размещения графа.

Предметом исследования является доказательство NP-полноты задачи размещения графа, разработка детерминированного эвристического алгоритма, нахождение оценки ее решений.

**Цель исследования**

Доказать NP-полноту задачи размещения.

Разработать эвристический алгоритм решения задачи размещения графа.

Выявить возможную оценку для решений задачи размещения графа.

**Гипотеза исследования**

Для доказательства NP-полноты нужна математическая модель на основании аппарата теории NP-полноты.

Существует эвристический алгоритм решения задачи размещения графа, который может быть сравним с существующими методами, но не являться стохастическим.

Существует оценка для решений задачи размещения графа на отклонение от оптимальный решений.

**Задачи исследования**

Рассмотрим этапы и задачи исследования, возникающие в процессе:

- анализ необходимости эвристического подхода;
    - построение математической модели с использованием аппарата теории NP-полноты;
    - доказательство NP-полноты задачи;
- построение эвристического алгоритма;
    - анализ существующих алгоритмов решения;
    - построение алгоритма решения;
- построение существующих алгоритмов для сравнения их результата с разработанным алгоритмом;
    - построение существующих алгоритмов решения для сравнительного анализа с разработанным алгоритмом;
- сбор данных для вычислительных экспериментов;
    - сбор всех попарно неизоморфных графов малых порядков для проведения сравнительного анализа алгоритма;

- разработка алгоритма генерации случайных графов с для анализа алгоритма на больших размерностях графов;
- разработка способа хранения большого количества графов;

**Теоретическая основа**

В связи с тем, что задача размещения графа является труднорешаемой, исследования в данной области направлены на нахождения эффективных эвристических и стохастических алгоритмов решения задачи.

Текущее состояние проблемы таково, что силы исследователей направлены на подбор параметров для существующих мультиэвристических алгоритмов.

Например, авторы [9], [1], [12] в своих работах используют различные мультиэвристические алгоритмы для решения задачи размещения графа.

Некоторые авторы [24], не заостряя свое внимание на конкретной задаче размещения графа, выделяют общие свойства графов, которые могут помочь в построении эффективных алгоритмов решения.

**Методологическая основа**

В качестве методологической основы использовались такие разделы, как теория формальных языков, теория NP-полноты, теория графов и теория алгоритмов.

**Основные этапы исследования и апробация результатов**

Этапы исследования:

- анализ необходимости эвристического подхода;
- построение эвристического алгоритма;
- построение существующих алгоритмов для сравнения их результата с разработанным алгоритмом;
- сбор данных для вычислительных экспериментов;

Для апробации результатов проводились сравнительные вычислительные эксперименты разработанного алгоритма с другими известными алгоритмами.

В работе [8] автором были опубликованы результаты построения математической модели и доказательство NP-полноты задачи.

В работах [13, 22] был представлен разработанный детерминированный эвристический алгоритм Hebene.

**Научная новизна, теоретическая и практическая значимость**

Во время исследования впервые была доказана NP-полнота задачи размещения графа.

Был построен эвристический алгоритм. Это, в первую очередь, означает, что требуются дальнейшие исследования этого алгоритма с целью выявления механизма его работы, которые могут помочь решению задач подобного рода.

Была получена оценка решений для задачи размещения графа. Такая оценка является полезной при исследовании алгоритмов для задачи, так как позволяет оценить приближенное отклонение найденного решения от оптимального.

Разработанный алгоритм может применяться для решения задачи размещения элементов СБИС.

**Основные положения**

Доказано, что задача размещения графа является NP-полно.

Разработан эвристический алгоритм решения задачи размещения графа.

Получена оценка оптимальных решений задачи.

Разработан специальный формат хранения большого количества графов graph7.

**Объем и структура**

Работа изложена на 91 страницах, содержит 41 рисунок и 6 таблиц.

Диссертационное исследование состоит из следующих глав.

Глава 1 АНАЛИЗ ЛИТЕРАТУРЫ

В этой главе делается анализ существующих публикаций по теме исследования, приводятся методы и алгоритмы, которые используют другие авторы.

Глава 2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

В этой главе приводится разработанная математическая модель для задачи размещения графа. Вводятся термины и определения. Доказывается теорема об NP-полноте задачи размещения графа. Проводится анализ существующих алгоритмов решения, их параметров и особенностей.

Глава 3 РЕАЛИЗАЦИЯ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ РАЗМЕЩЕНИЯ ГРАФА

В этой главе приводится реализации существующих алгоритмов, а также описываются разработанный алгоритм.

Приводятся методы сбора данных для вычислительных экспериментов.

Описывается разработанный формат хранения графов graph7.

Глава 4 ПРОВЕДЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ ЭКСПЕРИМЕНТОВ

В этой главе анализируется эффективность реализованных алгоритмов в сравнении с реализованным эвристическим алгоритмом.

ЗАКЛЮЧЕНИЕ

В заключении приводятся основные результаты работы, а также результаты проведенных вычислительных экспериментов.

# Глава 1 АНАЛИЗ ЛИТЕРАТУРЫ

В этой главе автор анализирует литературу, в которой делались попытки алгоритмизировать задачу размещения графа. Рассматривается математическая модель, которая применялась для формализации задачи. Приводятся алгоритмы, которые применялись к решению задачи.

В русской литературе термин труднорешаемая задача обозначает целый класс алгоритмически неразрешимых задач. Первые результаты о существовании таких задач были получены А. Тьюрингом и ставшие уже классическими результатами о неразрешимости ряда задач, для которых принципиально не существует алгоритмов их решения. В работе будет рассматриваться задача, алгоритмическая неразрешимость для которой не доказана. То есть задача, для которой неизвестны полиномиальные, детерминированные алгоритмы.

Разные алгоритмы имеют разную временную сложность, выяснением того, какие алгоритмы являются эффективней, занимается теория сложности вычислений. Один из подходов сравнения алгоритмов является оценка их функций сложности. Таким образом, можно разделить алгоритмы на классы полиномиальных и экспоненциальных алгоритмов.

Функция $f(n)$ есть $O(g(n))$, если существует константа $c$, такая, что $|f(n)| \leq c|g(n)|$ для всех значений $n \geq 0$. Полиномиальный алгоритм называется алгоритм, у которого временная сложность равна $O(p(n))$, где $p(n)$ – некоторая полиномиальная функция, а $n$ – входная длина. Алгоритмы, временная сложность которых не поддается такой оценке, называются «экспоненциальными».

Приведем пример сравнения полиномиальных и экспоненциальных алгоритмов на четырехъядерном процессоре AMD Phenom 9500 sAM2+ с тактовой частотой 2,2 ГГц пиковая производительность которого равна 35,2 млрд. операций в секунду. Из таблицы 1.1 видно, что время выполнения для

полиномиальных алгоритмов возрастает ни так быстро, как это делает экспоненциальный алгоритм.

Различие между полиномиальными и экспоненциальными алгоритмами проявляется еще больше, если проанализировать влияние увеличения быстродействия ЭВМ на время работы алгоритмов. Таблица 1.2 показывает, насколько увеличивается размер задач, решаемых за один час машинного времени, если благодаря совершенствованию технологий быстродействия ЭВМ возрастет в 100 или 1000 раз по сравнению с современными настольными системами. Заметим, что для $O(4^n)$ увеличение скорости в 1000 раз увеличивает размерность задачи, разрешимой за один час всего лишь на 4,98 (вообще говоря, $n$ не может быть дробным числом, здесь сделано допущение из-за того, что в час машинного времени, при заданной скорости ЭВМ, нельзя вместить размерность большую, чем $N_6 + 4{,}98$).

Таблица 1.1 – Сравнение нескольких полиномиальных и экспоненциальных функций временной сложности

| $g(n)$ | $n$ | | | | | |
|---|---|---|---|---|---|---|
| | 10 | 20 | 30 | 40 | 50 | 60 |
| $n$ | 2,84091e-10 с | 5,68182e-10 с | 8,52273e-10 с | 1,13636e-9 с | 1,42045e-9 с | 1,70455e-9 с |
| $n^2$ | 2,84091e-9 с | 1,13636e-8 с | 2,55682e-8 с | 4,54545e-8 с | 7,10227e-8 с | 1,02273e-7 с |
| $n^4$ | 2,84091e-7 с | 4,54545e-6 с | 2,30114e-5 с | 7,27273e-5 с | 1,77557e-4 с | 3,68182e-4 с |
| $n^8$ | 0,00284091 с | 0,727273 с | 18,6392 с | 186,182 с | 1109,73 с | 4771,64 с |
| $2^n$ | 2,90909e-8 с | 2,97891e-5 с | 0,030504 с | 31,2361 с | 31985,8 с | 3,27535e+7 с |
| $4^n$ | 2,97891e-5 с | 31,2361 с | 3,27535e+7 с | 3,43445e+13 с | 3,60128e+19 с | 3,77622e+25 с |

Таблица 1.2 – Влияние технического совершенствования ЭВМ на полиномиальные и экспоненциальные алгоритмы

| $g(n)$ | На современных ЭВМ | На ЭВМ, в 100 раз более быстрых | На ЭВМ, в 1000 раз более быстрых |
|---|---|---|---|
| $n$ | $N_1$ | $100N_1$ | $1000N_1$ |
| $n^2$ | $N_2$ | $10N_2$ | $31{,}6N_2$ |
| $n^4$ | $N_3$ | $3{,}16N_3$ | $5{,}62N_3$ |
| $n^8$ | $N_4$ | $1{,}78\mathrm{N}_4$ | $2{,}35\mathrm{N}_4$ |
| $2^n$ | $N_5$ | $N_5 + 6{,}64$ | $N_5 + 9{,}97$ |
| $4^n$ | $N_6$ | $N_6 + 3{,}32$ | $N_6 + 4{,}98$ |

Эти таблицы демонстрируют причины, по которым полиномиальные алгоритмы считаются лучше экспоненциальных.

Основополагающий характер различия между этими двумя классами впервые обсуждался в работах Кобхэма [20] и Эдмонса [25]. В частности, Эдмондс отождествлял полиномиальные алгоритмы с «хорошими» алгоритмами и высказал предположение, что некоторые задачи целочисленного программирования невозможно решить такими «хорошими» алгоритмами.

## 1.1 Современная математическая модель

На текущий момент существует несколько групп авторов, занимающиеся задачей размещения графа. Все современные алгоритмические подходы к решению задачи основаны на эвристических и стохастических методах. Это, прежде всего, связано с труднорешаемостью задачи, так как задача, по крайней мере, является NP-трудной (а как будет показано в главе 2 и NP-полной), что накладывает большие ограничения на нахождения решений с помощью современных детерминированных методов. Более точно, современные детерминированные алгоритмы неспособны решать задачи больших размерностей, так как все такие методы сводятся к вариациям алгоритмов полного перебора. А это говорит о том, что при решении задачи такими методами нужно перебирать экспоненциальное большое число вариантов, что

исключает перебор для графов больших размерностей. Для примера, рассмотрим общий вид задачи коммивояжера с 25 вершинами на основе ее постановки из [12, 21]. При скорости перебора 35 миллионов вариантов в секунду на проверку всех решений уйдет времени столько же, сколько прошло с момента Большого взрыва (приблизительно 13,7 миллиардов лет). А задачи, возникающие в реальных приложениях, в действительности могут иметь не 25, а много тысяч вершин. Поэтому очевидно, что переборный подход к решению подобного рода задач нечасто будет иметь успех.

Поскольку у современных авторов пользуются успехом эвристические и стохастические алгоритмы, то такой подход является наиболее перспективным.

В литературе разные авторы используют различную математическую модель задачи размещения графа. В частности, авторы используют более частную постановку, чем предлагает автор. Приведем постановку, которая используется в [1, 9, 12].

**Задача размещения графа в линейке.** Дан граф $G = (V, E)$. Найти $x^* = (x_1^*, x_2^*, \dots, x_n^*)$ такое, что

$$\forall x \in X \quad f(x^*) \leq f(x),$$

где

- $V$ – множество вершин графа $G$;
- $E$ – множество ребер графа $G$;
- $x^*$ – оптимальное решение;
- $x_i^*$ – номер позиции в линейке;
- $n = |V|$;
- $X$ – множество всех перестановок;
- $f$ – функция оценки (целевая функция).

Заметим, что задача формулируется только для частного случая задачи размещения графа в линейке. В качестве целевой функции авторы предлагают следующую.

$$L(G) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} d_{i,j} c_{i,j}. \quad (1.1)$$

Здесь

- $L(G)$ – общая суммарная длина ребер графа;
- $d_{i,j} = |i - j|$;
- $c_{i,j}$ – количество ребер, соединяющих вершины $x_i$ и $x_j$.

Такая целевая функция является естественной в одномерном случае задачи и, в дальнейшем, будет использоваться в искомой работе.

Другая модель, используемая в тех же работах [1, 9, 12], рассматривает другой случай задачи размещения – задачу размещения графа в плоскости.

**Задача размещения графа в плоскости.** Дан граф $G = (V, E)$ и координатная сетка $R$. Найти $x^* = (x_1^*, x_2^*, \ldots, x_n^*)$

$$\forall x \in X \quad f(x^*) \leq f(x),$$

где

- R – координатная сетка, размещение в которой нужно найти;
- $x^*$ – оптимальное решение;
- $x_i^*$ – координата в сетке $R$;
- $X$ – множество всех возможных размещений в сетке;
- $f$ – целевая функция.

В качестве целевой функции авторы предлагают несколько вариантов.

$$\Pi(G) = \frac{1}{2} \sum_{e \in E}^{n} \Pi(e). \quad (1.2)$$

Здесь

- $\Pi(G)$ – общее число пересечений ребер графа;
- $\Pi(e)$ – число пересечений ребра $e$.

$$L(G) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j} c_{i,j}. \quad (1.3)$$

Здесь $a_{i,j}$ предлагаются следующие:

$$a_{i,j} = \ s_i - s_j \ + \ t_i - t_j \ , \tag{1.4}$$

и

$$a_{i,j} = \ \overline{\ s_i - s_j \ ^2 + \ t_i - t_j \ ^2}. \tag{1.5}$$

где $s_i$, $t_i$ – координаты по двум направлениям в сетке.

Заметим, что формула (1.3) совпадает с формулой (1.1), за исключением первого сомножителя суммы, который определяется для разных размерностей по-разному. В качестве примера взглянем на рисунок 1.1. На нем изображена координатная сетка $R$. Жирным выделен путь, который считаются по формуле (1.4), а пунктиром, путь, который считается по формуле (1.5).

Формулы (1.4) и (1.5) являются метрикой городских кварталов и метрикой Евклида соответственно.

Также авторы предлагают для задачи размещения графа в плоскости целевую функцию, учитывающую одновременно формулы (1.2) и (1.3).

$$A \ G \ = \alpha L \ G \ + \beta \Pi \ G \ . \tag{1.6}$$

Здесь $\alpha$, $\beta$ – коэффициенты, учитывающие степень важности того или иного критерия ($\alpha + \beta = 1$).

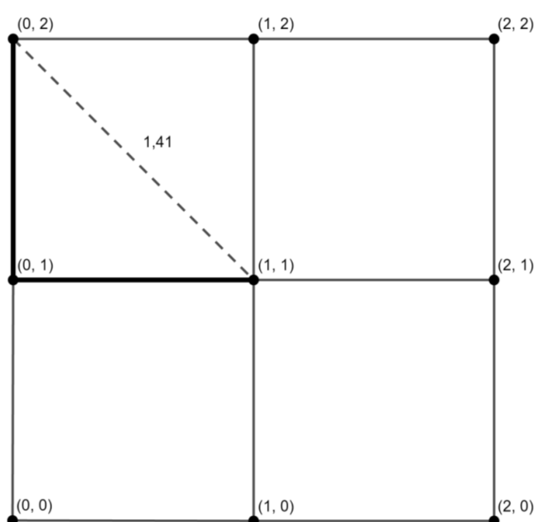


Рисунок 1.1 – Пример графа, расположенного в сетке

Такая целевая функция является более универсальным, но для ее применения требуется подбор коэффициентов. Обычно значения этих коэффициентов определяются на основе знаний лица принимающего решения или экспертных оценок.

Стоит отметить, что данные выше модели не являются точными копиями из работ авторов, а лишь подведены под стиль оформления искомой работы.

### 1.3 Алгоритмы решения задачи размещения графа

В настоящее время применяют несколько видов алгоритмов для решения задачи размещения:

- генетические алгоритмы;
- алгоритм имитации отжига;
- и другие мультиэвристические алгоритмы;

Кроме перечисленных алгоритмов также применяются эвристические алгоритмы, направленные только на решение задачи размещения.

В работах [1, 9, 10] внимание уделяется генетическим алгоритмам.

В [9] сказано, что алгоритм считается «вероятностно хорошим», если временная сложность является одной из следующих: $O(n)$, $O(n\log n)$, $O(n^2)$.

В работах [24, 4, 5] авторы используют понятия труднорешаемой задачи и предлагают методы алгоритмизации труднорешаемых задач. При этом, не фокусируя свое внимание на конкретных задачах.

Авторы этих работ рассматриваю множество алгоритмов:

- алгоритм локального поиска;
- алгоритм имитации отжига;
- алгоритм поиска с запретом;
- метод ветвей и границ;
- и др.

Все эти алгоритмы являются мультиэвристическими и могут быть применены к любой задаче комбинаторной оптимизации. Некоторые из этих алгоритмов обсуждаются в главе 2 и 3.

Сами авторы так описывают эвристический подход: «Среди алгоритмов мы в первую очередь рассматриваем эвристические – которые обычно не гарантируют получение оптимального решения, однако с приемлемо большой вероятностью дают решение, близкое к нему». И в действительно эвристические алгоритмы совсем не гарантируют нахождения оптимальных решений, но в связи с тем, что задачи, которые решают эвристические алгоритмы, обычно являются труднорешаемыми вычислительными задачами, приходится применять такие алгоритмы в практических целях.

Более подробный анализ алгоритмов делается в главе 2.

### 1.4 Выводы проведенного анализа литературы

В данной главе был проведен анализ литературы, в которой описывается задача размещения графа и делаются попытки (удачные) ее формализации алгоритмизации.

Были приведены математические модели, которые используются для формализации задачи размещения графа. Эти модели описывают две задачи: задачу размещения графа в линейке и задачу размещения графа в сетке. Также в основе этих моделей предлагаются целевые функции для оптимизации задачи.

Как было сказано, задача размещения графа является труднорешаемой и на текущий момент все применяемые алгоритмы являются эвристическими и стохастическими. Были приведены основные алгоритмы, которые современные исследователи применяют для получения приемлемых решений (по критериям работы и поисковым способностям).

# Глава 2 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

## 2.1 Разработка математической модели

### 2.1.1 Термины и определения

Когда перед исследователем стоит задача, для которой нужно найти точный алгоритм решения, нужно оценить задачу на возможность решения ее за полиномиальное время. Для всех NP-полных задач на текущий момент не существует полиномиального алгоритма, а, следовательно. Таким образом, доказательство NP-полноты задачи является важной частью в исследовании методов решения задачи.

В этом разделе повторяются термины и обозначения, используемые в [6], а также некоторые определения, взятые из работы автора этой работы [13].

Под графом, на протяжении всей работы, подразумевается неориентированный граф без петель и кратных ребер.

**Определение 2.1.** *Модель размещения графа.*

Моделью размещения графа назовем четверку:

$$\mathcal{M} = \langle G, M_\eta, H, f \rangle. \tag{2.1}$$

Здесь

- $G = \langle V, E \rangle$ – граф ($V$ – множество вершин, $E$ – множество ребер);
- $M_\eta$ – множество размещений многомерного пространства (то есть множество с элементами $x_\eta = \langle \boldsymbol{m_1}, m_2, \dots, m_n \rangle$, где $m_i \in \mathbb{N}^\eta$, $n = |V|$);
- $H: X \to M_\eta$ (здесь $X$ – множество всех размещений $x = \langle \boldsymbol{x_1}, x_2, \dots, x_n \rangle$, таких что $(\forall x_i, x_j \in V)(x_i \neq x_j)$);
- $f: M_\eta \to \mathbb{R}^+$.

В работе [8] автором было введено определение 2.1 для обобщения задачи размещения. На основе этого определения была составлена постановка задачи размещения как оптимизационной. Но в этой работе внимание уделялось алгоритмизации задачи без теоритических основ сложности задачи.

То есть делалось предположение, что задача является труднорешаемой без фактических доказательств ее сложности.

**Определение 2.2.** *Задача распознавания свойств.*

Задача распознавания свойств – это задача, которая формулируется в виде вопроса с двумя возможными ответам – «да» и «нет».

Такое определение задачи распознавания свойств является неформальным, но задачи распознавания свойств имеют формальный эквивалент, называемый «языком», и описывается этот эквивалент следующим образом.

Для любого конечного множества $\Sigma$ (называемого алфавитом) обозначим через $\Sigma^*$ множество всех конечных слов (то есть непрерывная последовательность символов алфавита), составленных из алфавита $\Sigma$. Подмножество $L \subseteq \Sigma^*$ называется языком.

Соответствие между задачами распознавания и языками устанавливается с помощью схемы кодирования. Схема кодирования $e$ задачи $\Pi$ описывает каждую индивидуальную задачу (то есть задачу, которая получается, если задать ей конкретные значения ее параметров) из $\Pi$ словом в некотором фиксированном алфавите $\Sigma$. Таким образом, задача $\Pi$ и схема кодирования $e$ разбивает множество $\Sigma^*$ на три класса: слова, которые не являются кодами индивидуальных задач, слова, которые являются кодами индивидуальных задач, но имеют отрицательный ответ на вопрос, и слова, являющиеся кодами индивидуальных задач с положительным ответом на вопрос. Третий класс слов является тем языком, который ставится в соответствие задаче $\Pi$ при кодировании $e$ и обозначается $L[\Pi, e]$.

Однако заметим, что схема кодирования $e$ задачи $\Pi$ не фиксируется и может быть любой «разумной» схемой.

В литературе по алгоритмизации (теории разработки алгоритмов) рассматриваются тысячи алгоритмических проблем, которые классифицируются согласно различных точек зрения. Здесь рассматриваются

так называемая труднорешаемая задача. Проблема называется трудной в том случае, если неизвестно детерминированных алгоритмов (компьютерных программ), эффективно ее решающих; при этом эффективность понимается как полиномиальное время решения, причем для полиномов с небольшой степенью. Такая интерпретация трудности связана с текущими знаниями в области алгоритмизации, а не с (пока) неизвестно реальной трудностью проблемы. Итак, здесь называется проблема трудно, если для ее практического решения с помощью известных в настоящее время детерминированных программ необходимы годы или даже тысячелетия.

Вообще говоря, каждый алгоритм (каждая компьютерная программа) может рассматриваться как выполнение специального отображения из $\Sigma_{\mathbf{1}}^*$ в $\Sigma_2^*$ для некоторых заданных алфавитов $\Sigma_{\mathbf{1}}$ и $\Sigma_2$. Вместо таких отображений можно рассматривать бинарные отношения на множестве $\Sigma_{\mathbf{1}}^* \times \Sigma_2^*$. Но обычно нет необходимости работать с подобными формализмами.

**Определение 2.3.** *Полиномиальная сводимость.*

Язык $L_{\mathbf{1}} \subseteq \Sigma_{\mathbf{1}}^*$ полиномиально сводится к языку $L_2 \subseteq \Sigma_2^*$, если существует функция $f: \Sigma_{\mathbf{1}}^* \to \Sigma_2^*$, удовлетворяющая двум условиям:

1. существует алгоритм, вычисляющий $f$ с временной сложностью, ограниченной полиномом;
2. для любого $x \in \Sigma_{\mathbf{1}}^*$, $x \in L_{\mathbf{1}}$ в том и только том случае, если $f(x) \in L_2$.

Если $L_{\mathbf{1}}$ сводится к $L_2$, то это обозначается как $L_{\mathbf{1}} \propto L_2$. Обозначение $\Pi_{\mathbf{1}} \propto \Pi_2$ означает, что $L\ \Pi_{\mathbf{1}}, e_{\mathbf{1}}\ \propto L[\Pi_2, e_2]$ для некоторых схем кодирования $e_{\mathbf{1}}$ и $e_2$.

В [13] показано, что для доказательства того, что задача $\Pi$ принадлежит классу $NPC$ (NP-полных) задач, нужно доказать следующее:

- $\Pi \in NP$;
- $\Pi' \propto \Pi, \Pi' \in NPC$.

Таким образом, для доказательства того, что задача размещения графа принадлежит классу $NPC$, нужно показать, что:

- задача принадлежит классу $NP$;
- известная задача из класса $NPC$ сводится к задаче размещения графа.

Также в работе используются следующие термины:

- сигнатура модели – это тройка $\mathcal{S} = \ M_{\eta}, H, f\ $;
- вариант модели – определяется некоторым подмножеством всех возможных входов. Вариантом модели выступает модель с заданной на ней сигнатурой;
- частный случай задачи – задаётся на варианте модели с заданием конкретного графа (в данном случае это определение совпадает с определением индивидуальной задачи распознавания свойств).

Сигнатура с заданными элементами описывают некоторое подмножество задачи размещения графа. Модель с сигнатурой $\mathcal{S} = \ M_{\mathbf{1}}, H, f\ $ называется задачей размещения графа в линейке, а $\mathcal{S} = \ M_2, H, f\ $ – задачей размещения графа в сетке (плоскости). Для краткости задачу размещения графа в линейке будем обозначать $\ G, f\ $ (предполагая, что $M_{\mathbf{1}} = X, H\ x\ = x$).

**Определение 2.4.** *Оптимизационная проблема.*

Оптимизационная проблема это семерка

$$U = \ \Sigma_{I}, \Sigma_{O}, L, L_{I}, M, f, goal\ ,$$

где

- $\Sigma_{I}$ – алфавит, называемый входным алфавитом проблемы $U$;
- $\Sigma_{O}$ – алфавит, называемый выходным алфавитом проблемы $U$;
- $L \subseteq \Sigma_{I}^{*}$;
- $L \subseteq L$;
- $M(x)$ – множество допустимых решений для (входа) $x$;
- $f$ – целевая функция
- $goal \in \{min, max\}$ – вариант оптимизации: минимум, максимум (здесь всюду, когда говорится об оптимизации, имеется в виду оптимизация на минимум).

Такое определение оптимизационной задаче используется в [24] и согласуется с терминами, введенными ранее.

### 2.1.2 Об NP-полноте задачи размещения графа

Как было сказано, для доказательства NP-полноты какой-либо задачи нужно показать, что к ней сводится уже известная NP-полная задача. В качестве такой задаче была выбрана задача поиска гамильтонова цикла, так как она наилучшим образом подходит для доказательства NP-полноты задачи размещения графа. Опишем задачу гамильтонова цикла и задачу размещения графа.

**Задача поиска гамильтонова цикла**. Дан граф $G = \langle V, E \rangle$. Найти размещение $x = \langle x_1, x_2, \ldots, x_n \rangle$ такое, что $x$ гамильтонов цикл, то есть простой цикл, содержащий все вершины графа.

**Задача размещения графа**. Дан вариант модели $\mathcal{M} = \langle G, M_\eta, H, f \rangle$ и граница $B \in \mathbb{R}$. Найти размещение $x = \langle x_1, x_2, \ldots, x_n \rangle$ такое, что

$$f(H(x)) \leq B. \qquad (2.2)$$

Далее приведена теорема, которая доказывает NP-полноту задачи размещения графа.

**Теорема 2.1**. Задача размещения графа NP-полна.

**Доказательство**. Возьмем частный случай задачи размещения графа и покажем, что задача поиска гамильтонова цикла сводится к нему. Положим, что $\mathcal{M} = \langle G, f \rangle$ – задача размещения графа в линейке, и $B = 2(n-1)$. Построим функцию $f_H(x)$ такую, что если $x$ гамильтонов цикл, то $f_H(x) = B$, то есть

$$f_H(x) = a(x_1, x_n)(n-1) + \sum_{i=1}^{n-1} a(x_i, x_{i+1}) = B, \qquad (2.3)$$

где $a(x_i, x_j) = \begin{cases} 1, \text{если } (x_i, x_j) \in E \\ 0, \text{если } (x_i, x_j) \notin E \end{cases}$ – функция, которая возвращает количество ребер между парой вершин (то есть 0 или 1). **Теорема 2.1 доказана**.

Если задачу поиска гамильтонова цикла описать в терминах задачи размещения графа, то получится следующее.

**Задача поиска гамильтонова цикла**. Дан вариант модели $\langle G, f_H \rangle$ и граница $B \in \mathbb{R}$. Найти размещение $x = (x_1, x_2, \ldots, x_n)$ такое, что

$$f_H(x) = B. \quad (2.4)$$

В качестве примера рассмотрим граф, изображенный на рисунке 2.1. На нем изображен граф, размещенный в линейке, жирным выделен путь, который соответствует гамильтонову циклу, а $f_H((1,2,3,4)) = B = 6$.

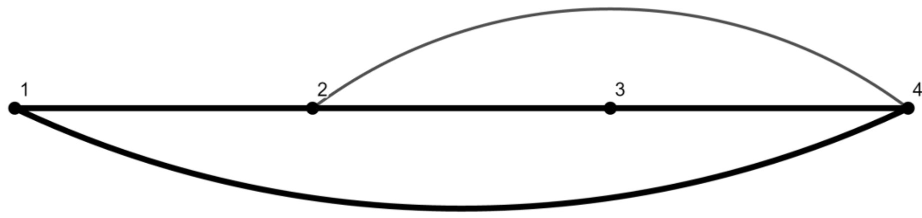


Рисунок 2.1 – Решение с гамильтоновым циклом

На рисунке 2.2 изображен тот же граф, размещенный в линейке, но, в отличие от предыдущего, цикла для этого решения нет, и значение $f_H((1,2,4,3)) = 3$.

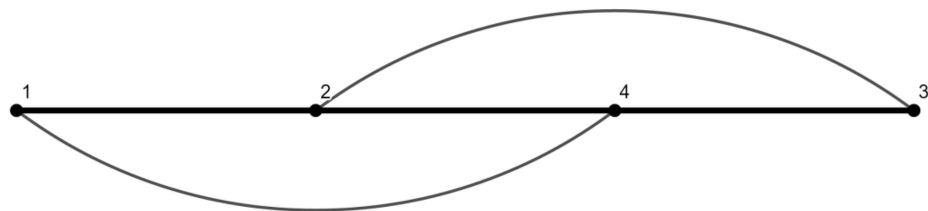


Рисунок 2.2 – Решение без гамильтонова цикла

Нужно сказать, что для доказательства того, что задача размещения графа является NP-полной, не использовались термины функций сводимости и схем кодирования, поскольку для доказательства показывается, что задача поиска гамильтонова цикла является частным случаем задачи размещения графа и функция сводимости не требуется, потому что задачи изначально эквивалентны.

### 2.1.3 Вспомогательные результаты

Опишем задачу размещения графа как оптимизационную. Отметим, что многие задачи распознавания свойств могут быть описаны как оптимизационные [6].

**Задача минимизации размещения графа**. Дан вариант модели $\mathcal{M} = (G, M_\eta, H, f)$. Найти размещение $x^* = (x_1^*, x_2^*, \dots, x_n^*)$ такое, что

$$\forall x \in X \quad f(H(x^*)) \leq f(H(x)), \tag{2.5}$$

где $X$ – множество всех перестановок $\{1,2,\dots,n\}$.

Приведем некоторые вспомогательные результаты, которые устанавливают связь между задачей размещения графа, как задачей распознавания свойств и оптимизационной задачей размещения графа.

Рассмотрим вариант модели задачи размещения графа в линейке с

$$f_L(x) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} a(x_i, x_j)\,|i-j|, \tag{2.6}$$

то есть рассматривается вариант модели $(G, f_L)$, тогда можно сформулировать следующую теорему.

**Теорема 2.2**. Если существует полиномиальный алгоритм A, решающий задачу размещения графа в линейке с вариантом модели $(G, f_L)$, то существует полиномиальный алгоритм A*для задачи минимизации размещения графа.

**Доказательство**. Точными границами для множества значений функции $f_L$ $\alpha = \sup f_L(x)$ и $\beta = \inf f_L(x)$ являются:

$$\alpha = n-1, \beta = \frac{n(n^2-1)}{6}. \tag{2.7}$$

Такие оценки получаются, если вычислить $f_L(x)$ для простой цепи и полного графа длины $n$ соответственно. Положим $m = \beta - \alpha$ – максимальное количество возможных значений $f_L(x)$. Алгоритм $A^*$ строится путем применения алгоритма линейного поиска для всех возможных значений

$f_L(x) \in [\alpha;\beta]$ и, тем самым, нахождения $x^*$. Если $O(g(n))$ – оценка алгоритма $A$, где $g(n)$ – полином, то $O(m \cdot g(n))$ – оценка алгоритма $A^*$. **Теорема 2.2 доказана**.

В дальнейшем, под задачей размещения графа будет всегда пониматься оптимизационная задача размещения графа, так как описание задачи размещения графа как задачи распознавания свойств нужно было лишь для доказательства ее NP-полноты.

Также для проведения вычислительных экспериментов нужна оценка, которая позволила бы более точно, чем (2.7) «угадывать» минимально и максимально возможное значение целевой функции. Естественным будет оценка, основанная на подсчете количества ребер.

Рассмотрим модель $(G, f)$, где $f = f_L$, так как в дальнейшем именно на ее основе проводятся вычислительные эксперименты. Предположим, что дан граф как на рисунке 2.3. На рисунке 2.4 изображен граф, расположенный в линейке с размещением $x = (1,2,3,4,5)$ и с ребрами, имеющими вес равный расстоянию между соответствующими вершинами. Таким образом, значение $f(x) = 16$ (то есть сумма взвешенных ребер для графа этого графа).

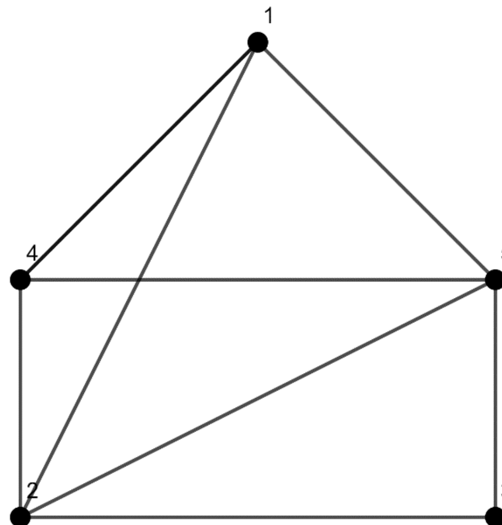


Рисунок 2.3 – Пример графа

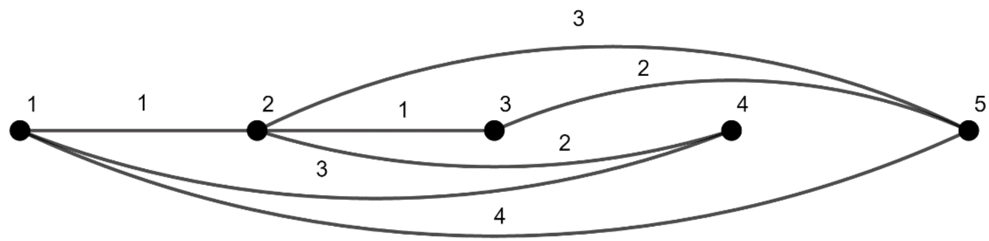


Рисунок 2.4 – Пример размещения графа в линейке

Представление размещения в виде взвешенного графа дает возможность построения оценки минимального и максимального возможного значение $f(x)$ для произвольного графа с $n$ вершинами и $k$ ребрами.

Рассмотрим полный граф с $n$ вершинами. Количество ребер у полного графа равно $k = \frac{n(n-1)}{2}$. Тогда значение $f(x)$ для любого $x$ равно:

$$\mu(n) = \sum_{i=1}^{n} \sum_{j=1}^{n-i+1} i.$$

Функция $\mu(n)$ вычисляет сумму $1_1 + 1_2 + \cdots + 1_n + 2_1 + \cdots + 2_{n-1} + \cdots + i_1 + i_2 + \cdots i_{n-i-1} + \cdots + n$, а количество членов суммы равно $k$. Функцию, вычисляющую сумму первых $p$ членов последовательности обозначим $\mu_{min}(n,p)$. Для произвольного графа с $n$ вершинами и $p$ ребрами выполняется неравенство:

$$f(x) \geq \mu_{min}(n,p).$$

Вычисление $p$ членов последовательности в обратном порядке приводит к функции $\mu_{max}(n,p)$, для которой справедливо неравенство:

$$f(x) \leq \mu_{max}(n,p).$$

Оценку $\mu_{min}(n,p)$ можно улучшить для графов, не содержащих гамильтонов путь, так как сумма, образованная вычислением $f(x)$ содержит в качестве членов значения расстояния между смежными вершинами в линейке, то граф, не содержащий гамильтонов путь, не может внести в общую сумму число больше, чем $n-2$. Таким образом, для графов, не содержащих гамильтонов путь верно неравенство:

$$f(x) \geq \mu_{min}(n,p) + 1.$$

На защиту выносится разработанная математическая модель, введенные термины, доказанные теоремы, а также предложенные вспомогательные оценки.

## 2.2 Алгоритмы решения задачи размещения графа

### 2.2.1 Алгоритм локального поиска

Локальный поиск – пример технологии разработки алгоритмов решения оптимизационных проблем. Рассмотрение алгоритма локального поиска важно по следующим причинам:

- он позволяет уменьшать сложность создаваемых алгоритмов и обеспечивать в оптимизационных проблемах приемлемые временные границы поиска локального оптимума;
- представленный здесь классический метод локального поиска является основой для другого алгоритма – алгоритма имитации отжига.

**Алгоритм 2.1** *Алгоритм локального поиска*

1: $S \leftarrow$ начальное потенциальное решение

2: $\boldsymbol{repeat}$

3: $R \leftarrow Mutator(S)$

4: $\boldsymbol{if}\ f(R) < f(S)$

$\boldsymbol{then}$

5: $S \leftarrow R$

6: $\boldsymbol{end\ if}$

7: $\boldsymbol{until}\ S$ – идеальное решение, или закончилось время поиска

8: $\boldsymbol{return}\ S$

Здесь $f$ – как и прежде, целевая функция, $Mutator(S)$ – функция, возвращающая измененную копию $S$ такую, что $|f(Mutator(S)) - f(S)| < \varepsilon$, т.е. эта функция возвращает копию решения, находящегося в окрестности с исходным решением.

### 2.2.2 Алгоритм имитации отжига

Алгоритм имитации отжига – метаэвристический метод поиска локального экстремума функции. Алгоритм основывается на физических процессах, которые происходят при кристаллизации вещества, в том числе при имитации отжига металлов (откуда алгоритм и берет название). Предполагается, что процесс отжига происходит при постепенном снижении температуры, при этом атомы, выстроенные в кристаллическую решетку, могут с определенной вероятностью совершить переход из одной ячейке в другую. Вероятность перехода атомов зависит от температуры, при снижении температуры вероятность уменьшается. Таким образом, в начальный момент времени вероятность перехода является максимально допустимая. Устойчивая кристаллическая решетка соответствует минимуму энергии атомов, при этом атом может либо перейти в состояние с меньшим уровнем энергии, либо остаться на месте.

Алгоритм имитации отжига характеризуется следующими параметрами:

- начальной температурой;
- законом изменения температуры;
- функции принятия нового состояния $\xi(\Delta f, t)$.

Здесь $\Delta f = f(x + \Delta x) - f(x)$ – приращение функции, $f(x)$ – целевая функция, $t$ – температура.

Главным отличием алгоритма имитации отжига от алгоритма локального поиска является способ изменения состояния на этапе принятия решения. Если градиентный спуск принимает любое решение лучше текущего, то алгоритм имитации отжига может принять решение хуже текущего с вероятностью

$$\xi(\Delta f, t) = e^{\frac{\Delta f}{t}} \qquad (2.8)$$

где $e$ – число Эйлера, при $t \geq 0$ и $\Delta f < 0$.

Из условия, что $\Delta f < 0$ следует два интересных свойства. Во-первых, если $f(x + \Delta x)$ значительно хуже $f(x)$, то модуль степени увеличивается, и вероятность становится близка к $0$, то есть шансы, что алгоритмы примет

решение намного хуже текущего стремятся к $0$. Если $f(x+\Delta x)$ мало отличается от $f(x)$, то вероятность стремится к $1$.

Во-вторых, имеется, настраиваемый параметр температуры $t$. Если значение $t$ мало, то степень увеличивается по модулю и вероятность также близка к $0$. Если же $t$ велико, то вероятность стремится к $1$.

Если значение $t$ будет слишком велико, то получаются случайные блуждания в пространстве поиска. Затем $t$ уменьшается в соответствии с законом уменьшении температуры, достигая, в конце концов, $0$, после чего алгоритм вырождается в алгоритм локального поиска.

Классическая схема алгоритма с помощью псевдокода описывается следующим образом.

**Алгоритм 2.2** *Алгоритм имитации отжига*

1: $t \leftarrow$ температура, большое начальное значение

2: $S$ $\leftarrow$ начальное потенциальное решение

3: $\boldsymbol{repeat}$

4: $R \leftarrow Mutator(S)$

5: $\boldsymbol{if}$ $f(R) < f(S)$ или случайное число из интервала $(0;1) < \xi(\Delta f, t)$ $\boldsymbol{then}$

6: $S \leftarrow R$

7: $\boldsymbol{end\ if}$

8: Уменьшить $t$

9: $\boldsymbol{until}$ $S$ – идеальное решение, или закончилось время поиска, или $t \leq 0$

10: $\boldsymbol{return}$ $S$

Скорость, с которой уменьшается $t$, называется расписанием алгоритма. Чем больше по времени «растягивается» расписание, тем больше алгоритм напоминает случайное блуждание и тем дольше идет исследование пространства поиска.

Алгоритм имитации отжига – широко используемый метод решения задач оптимизации. Основные причины успеха таковы [24, 18, 26]:

- он основан на простой и ясной идее и легко реализуем;
- он очень надежен и может применяться почти для всех задач оптимизации;
- благодаря замене детерминированного критерия принятия решений на стохастический критерий он ведет себя лучше алгоритма локального поиска, у которого тоже имеется сформулированные положительные качества простоты и надежности.

Очевидно, что успех применения алгоритма имитации отжига в конкретных оптимизационных задачах зависит еще и от выбора свободных параметров.

**Выбор начального значения температуры**

Практика показывает, что обычно наибольшее улучшение качества текущего решения происходит в середине этого графика, что объясняется следующими двумя обстоятельствами. Первая часть работы алгоритма имитации отжига выполняется при очень высокой температуре, которая разрешает почти все возможные ухудшения качества текущего решения. Поэтому начало работы алгоритма можно рассматривать как получения практически случайных последовательностей допустимых решения или как почти случайны поиск стартового допустимого решения. С другой стороны, если значение $t$ уже достаточно мало, то большие ухудшения вообще невозможны.

Итак, для графика охлаждения нужно определить такие параметры:

- начальную температуру $t$;
- функцию снижение температуры;

Начальное значение $t$ должно быть достаточно большим для того, чтобы позволить принять все возможные в будущем перемещения. По аналогии с физикой, оно должно соответствовать нагреванию твердого тело до тех пор,

пока тело достигнет жидкой фазы, и все его частицы станут располагаться случайным образом.

Одна из возможностей выбора $t$ состоит в том, чтобы взять максимальную (из имеющихся) разницу в стоимостях двух соседних решений. Но поскольку для вычисления этого значения часто необходимо большая вычислительная работа, то приходится с помощью какого-либо эффективного алгоритма находить его границу, приблизительное значение.

А другая практическая возможность заключается в том, чтобы начать с произвольного значения $t$.

**Выбор функции закона изменения температуры**

Обычным вариантов выбора является умножение $t$ после некоторого заранее зафиксированного количества $k$ итеративных шагов с фиксированным значением $t$ на некоторую константу $r$, обычно $0.8 \leq r \leq 0.99$. Итак, после каждых $k$ шагов выполняется переприсваивание

$$t(k) = r \cdot t(k-1) \quad (2.9)$$

То есть температура $t(k)$ после $k$ итераций равна $r^k \cdot t$.

Встречается и другой вариант функции:

$$t(k) = \frac{t_0}{\log(k+2)}. \quad (2.10)$$

Число $k$ (количество итераций для любой фиксированной температуры) при этом обычно выбирается как размер текущей окрестности.

Обе представленные здесь функции снижения температуры называются статическими. Кроме того, существует более сложные функции – динамические, в которых скорость охлаждения изменяется со временем. Их применение основано на нетривиальных выкладках стохастического анализа – и поэтому здесь не обсуждаются.

**Выбор условия завершения**

Один из возможных вариантов завершения работы алгоритм в том случае, когда за некоторое время не происходит улучшений стоимости текущих

решений: отметим, что этот вариант завершения работы не зависти от $t$. Другая возможность состоит в том, чтобы выбрать фиксированную константу $term$ и завершить работу в случае $T \leq term$. Третья, аналогично термодинамике, условие завершения может быть таким:

$$term \leq \frac{\varepsilon}{\ln \frac{M(x) - 1}{p}}.$$

Здесь $p$ – вероятность получения допустимого решения с аппроксимационным отношением $\varepsilon$, $M(x)$ – множество допустимых решений.

Рассмотрим некоторые практические результаты, полученные на основе опыта разработки вариантов алгоритма имитации отжига и их выполнения за полиномиальное время. Сначала перечислим несколько общих свойств, независимых от конкретных приложений:

- алгоритм имитации отжига выдает допустимые решения высокого качества, однако, за счет большой вычислительной работы;
- качество выходных данных существенно не зависит от выбора стартового допустимого решения, которое может быть найдено, например, локальным поиском;
- временная сложность алгоритма имитации отжига в среднем близка к их временной сложности в худшем случае.

Также отметим, что успех алгоритма имитации отжига может существенно зависеть и от конкретного класса рассматриваемых оптимизационных задач.

Стоит сказать, что в алгоритмах 2.1 и 2.2 уже выбран параметр $term = 0$.

### 2.2.3 Генетический алгоритм

Генетические алгоритмы – мультиэвристический алгоритм поиска, решающий задачи оптимизации путем подбора, комбинирования параметров с использованием механизмов, аналогичных природной эволюции. Генетические алгоритмы являются вариацией эволюционных вычислений [9], в которых также используются такие механизмы, как наследование, мутация, отбора и

скрещивания. Отличительной чертой генетических алгоритмов является акцент на использования оператора скрещивания (рекомбинации, кроссинговера).

Первые публикации на тему использования принципов биологической эволюции в вычислительной технике появились в 1975 г. В фундаментальной работе Д. Холланда [23] был предложен первый генетический алгоритм. Эта работа положила основу для многих исследований адаптивных систем, использующих принципы эволюции.

Генетические алгоритмы показали свою эффективность в решении многих задач оптимизации [11, 2]. В частности, они хорошо применимы к задачам с большим пространством поиска – к задачам комбинаторной оптимизации.

*Схема генетического алгоритма Холланда:*

1. сгенерировать исходную популяцию, состоящую из $N$ особей;
2. оценить приспособленность индивидов в популяции на основе целевой функции (фитнес-функции);
3. выполнить операцию селекции;
4. применить генетические операторы (мутация, скрещивание);
5. сформировать новую популяцию;
6. если критерий останова не достигнут, то перейти к шагу 2, иначе завершить работу.

Приведем, согласно [9] и др., общепризнанные достоинства и недостатки генетических алгоритмов.

Достоинства:

- широкая область применения;
- возможность проблемно-ориентированного кодирования решений, подбора начальной популяции, комбинирование генетических алгоритмов с любыми алгоритмами (в том числи и неэволюционными);
- пригодность для поиска в сложном пространстве решений большой размерности;

- отсутствие ограничений на вид целевой функции.

Недостатки:

- эвристический характер генетических алгоритмов не гарантирует, что полученное решение будет находиться в глобальном экстремуме;
- невысокая эффективность генетических алгоритмов на заключительных фазах моделирования; это объясняется тем, что генетические операторы не ориентированы на быстрое попадание в локальный оптимум.

Однако описание общей схемы алгоритма недостаточно, так как у генетического алгоритма есть очень важные части, которые могут меняться. К этим частям относятся:

- оператор скрещивания;
- оператор мутации;
- оператор селекции.

Существует большое количество вариаций этих операторов, которые сильно зависят от задачи. Поэтому целесообразно обозначить основные варианты, подходящие для задачи размещения.

**Оператор скрещивания**

Оператор скрещивания необходим для того, чтобы передавать признаки хромосом-родителей к хромосомам-потомкам.

Как и говорилось, существует большое количество вариантов операторов скрещивания, но здесь опишем только классические, так как в дальнейшем именно на базе них будет описан модифицированный оператор скрещивания.

*Одноточечное скрещивание.* Одноточечное скрещивание – классический оператор, предложенный в [7]. Выбирается пара хромосом-родителей из общей популяции решений. Далее выбирается точка разреза $p$ такая, что $p < n$, где $n$ – размер хромосомы родителей. После чего, части генов скрещиваются таким образом. Положим, что $m_{\mathbf{1}}, m_2$ – участки хромосом первого родителя до точки

разреза и после соответственно, а $d_1, d_2$ – участки хромосом второго родителя. Тогда потомок 1 будет иметь гены $m_1 d_2$, а потомок 2 $d_1 m_2$.

Проиллюстрируем вышесказанное. Допустим, имеются два родителя как на рисунке 2.5, и есть точка разреза $p = 3$. Тогда потомки будут иметь значение генов как на рисунке 2.6. Такой вид оператора имеет некоторые проблемы для задачи размещения графа, так как решениями задачи размещения графа являются перестановки, и элементы перестановок не могут повторяться. В главе 3 обсуждается этот вопрос и предлагается модифицированный оператор скрещивания.

| | | | | | |
|---|---|---|---|---|---|
| Родитель 1 | 1 | 2 | 3 | 4 | 5 |
| | 1 | 1 | 1 | 1 | 1 |
| Родитель 2 | 1 | 2 | 3 | 4 | 5 |
| | 0 | 0 | 0 | 0 | 0 |

Рисунок 2.5 – Пример применения одноточечного оператора скрещивания

| | | | | | |
|---|---|---|---|---|---|
| Потомок 1 | 1 | 2 | 3 | 4 | 5 |
| | 1 | 1 | 1 | 0 | 0 |
| Потомок 2 | 1 | 2 | 3 | 4 | 5 |
| | 0 | 0 | 0 | 1 | 1 |

Рисунок 2.6 – Пример применения одноточечного оператора скрещивания

*Многоточечное скрещивание*. Этот оператор похож на предыдущий, но с тем изменением, что количество точек разреза может быть несколько.

**Оператор мутации**

Оператор мутации в генетических алгоритмах необходим для того, чтобы когда решения начинают вырождаться, не достигнув глобального оптимума, «выбивать» их из локальных оптимумов.

Этот оператор всегда зависит от формы данных и изменяется от задаче к задаче. Однако на практике чаще всего применяется такой оператор.

Допустим, дана перестановка $x = 1,2,3,4,5,6$ и случайное число $r_i, r_j \leq n$, где $n = |x|$. Тогда мутацией считается обмен элементов $r_i$ и $r_j$ в перестановке $x$. Если, например, $r_i = 1$ и $r_j = 4$, то $x = 4,2,3,1,6$ . Кроме того, можно добавить параметр $p$, отвечающий за силу мутации, и такой, что мутация будет повторятся $p$ раз.

Для задач, в которых решениями являются перестановки, этот оператор является наиболее используемым.

**Оператор селекции**

Оператор селекции – это процедура, которая участвует в создании потомков для следующей популяции. По аналогии с естественным отбором, оператор селекции выбирает наиболее приспособленные (с наилучшим значением целевой функции) особи (хромосомы).

Если операторы скрещивания и мутации зависят от формы данных и задачи, то оператор селекции от нее не зависит и существует большое количество операторов селекции. Опишем лишь основные из них и их преимущества.

*Пропорциональная селекция*. Этот метод селекции является классическим и описывается следующим образом.

Пусть $f_i$ – значение целевой функции индивида, $i = 1,n$, где $n$ – размер популяции. Тогда $p_i = \frac{f_i}{{}_j f_j}$ – вероятность быть выбранным для следующего поколения, где ${}_j f_j$ – среднее значение целевой функции в популяции. В равновесных генетических алгоритмах размер популяции является

фиксированным, поэтому для следующего поколения с вероятностью $p_i$ выбирается $n$ особей.

Проблемы, связанные с этим типом селекции:

- преждевременная сходимость;
- стагнация.

*Турнирная селекция*. Данный тип селекции можно описать следующим образом. Из популяции, содержащей $n$ особей, выбирается случайным образом $t$ особей, и лучшая особь записывается в промежуточную популяцию (между выбранными особями проводится турнир). Эта операция повторяется ровно $n$ раз. Затем особи в полученной промежуточной популяции скрещиваются (также случайным образом). С ростом параметра $t$ ужесточается отбор между особями, т.к. если у особи низкий показатель приспособленности, у нее нет шансов «завести потомство». Преимуществом этой стратегии является то, что она не требует дополнительных вычислений и упорядочивания особей в популяции.

*Ранговая селекция*. В этом типе селекции все особи сортируются по значению $f_i$:

$$f_{i_1} < f_{i_2} < \cdots < f_{i_n}.$$

Затем каждому индивиду назначается вероятность $p_i$ из заранее заданного распределения такого, что $\quad_i p_i = 1$. Ранговая селекция состоит в применении оператора пропорциональной селекции с подстановкой рангов особей $p_i$ вместо значений целевой функции.

### 2.3 Вспомогательный метод оптимизации

Здесь опишем вспомогательный метод, который может помочь получать лучшие решения. Этот метод описан в [9] и приводится здесь без каких-либо изменений.

Рассмотрим граф в сетке, изображенный на рисунке 2.7.

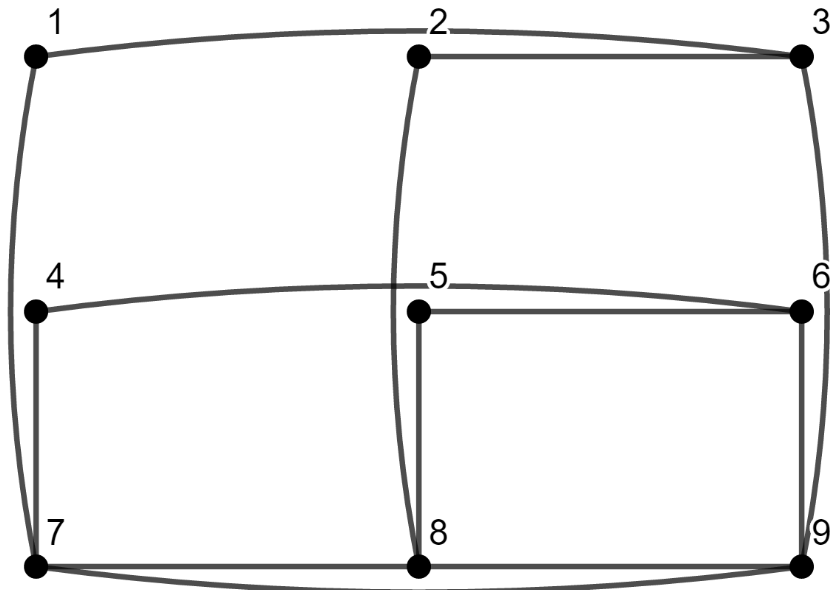


Рисунок 2.7 – Пример графа в сетке

Построим фрактальное множество, для этого каждый столбец графа представим в виде одной вершины как на рисунке 2.8. Повторим эту процедуру для строк как на рисунке 2.9. В результате получилось два графа, которые нужно оптимизировать относительно целевой функции. Например, можно использовать (1.1). Если перебрать все варианты, то оптимальными решениями будут решения как на рисунках 2.10 и 2.11. Если применить эти размещения к графу, изображенному на рисунке 2.7, то получится граф как на рисунке 2.12.

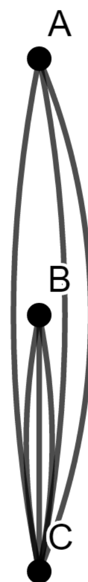


Рисунок 2.8 – Фрактальное множество относительно столбцов

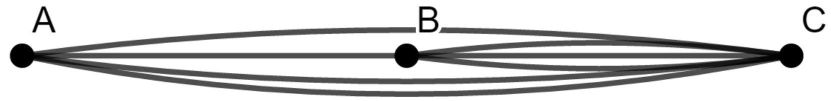


Рисунок 2.9 – Фрактальное множество относительно строк

Рисунок 2.10 – Оптимальное решение фрактального множества относительно столбцов

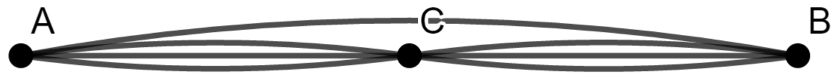


Рисунок 2.11 – Оптимальное решение фрактального множества относительно строк

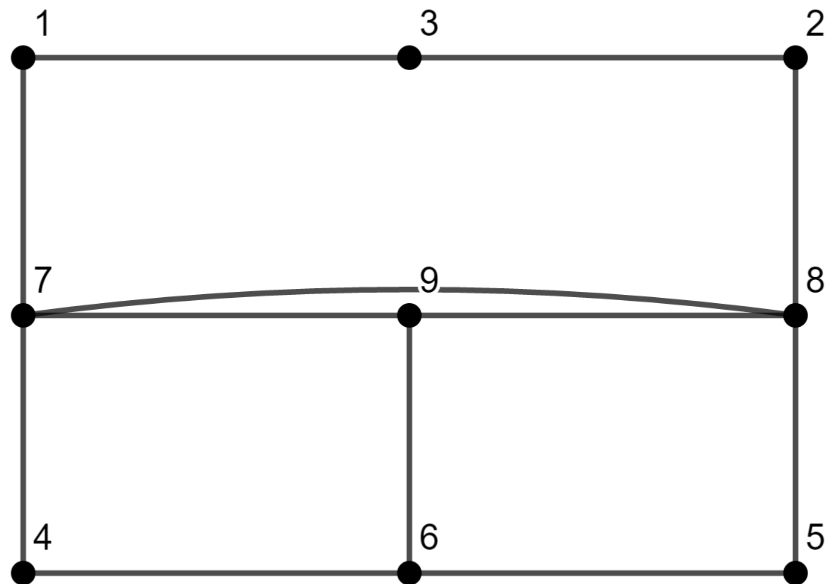


Рисунок 2.12 – Оптимальное размещение графа в сетке

Видно, что такой метод позволяет с некоторыми потерями информации разделить задачу на две маленькие задачи, которые могут быть решены любыми доступными методами решения задачи размещения в линейке (в том числе и перебором).

## 2.4 Выводы проведенного анализа предметной области

В главе этой были введены термины и определения необходимые для формализации задачи размещения графа. На основании введенных терминов и определений было приведено доказательство того, что задача является NP-полной, а значит найти эффективный алгоритм ее решения равносилен решению задачи $P = NP$ [27], что является трудной проблемой. Тем самым, обосновывается выбранный подход к алгоритмизации задачи.

Поставлена оптимизационная задача размещения графа. Была доказана теорема, что разрешение оптимизационной задачи размещения влечет за собой разрешение задачи размещения графа как задачи распознавания свойств.

Была введена оценка, которая позволяет приближенно оценивать возможное значение одной целевой функции для графов заданного порядка и количества ребер.

Подробно проанализированы известные методы решения задачи размещения. Приведены факторы, влияющие на успешность разных алгоритмов.

Приведен дополнительный метод оптимизации для задачи размещения графа в сетке, который позволяет сильно упростить эту задачу за счет некоторой потери информации.

# Глава 3 РЕАЛИЗАЦИЯ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ РАЗМЕЩЕНИЯ ГРАФА

## 3.1 Реализация алгоритмов решения задачи размещения графа

### 3.1.1 Реализация алгоритма имитации отжига и алгоритма локального поиска

Во время реализации алгоритма локального поиска не было внесено каких-либо изменений в сам алгоритм, и он реализовывался как есть.

Во время реализации алгоритма имитации отжига начальная температура выбиралась такой, чтобы алгоритм имел полиномиальное время выполнение. Оценка времени выполнения делалась на основе графика времени работы алгоритма с применением аппроксимационных методов.

В качестве функции закона изменения температуры была выбрана функция (2.9) со значением $r = 0.99$:

$$t(k) = r \cdot t(k-1),$$

а $t_0 = 10$.

Недостаток классической схемы алгоритма имитации отжига является то, что во время работы алгоритма возможна потеря хорошего решения. Это может случиться, если алгоритм примет с некоторой вероятностью худшее решение, и до своего окончание не сможет найти лучше. Поэтому можно описать улучшенную схему алгоритма, которая решает данную проблему.

**Алгоритм 3.1** *Модифицированный алгоритм имитации отжига*

1: $t \leftarrow$ температура, большое начальное значение

2: $S \leftarrow$ начальное потенциальное решение

3: $B \leftarrow S$

4: $\boldsymbol{repeat}$

5: $R \leftarrow Mutator(S)$

6: $\boldsymbol{if}\ f(R) > f(S)$ или случайное число из интервала $(0;1) < \xi(\Delta f, t)$ $\boldsymbol{then}$

7: $S \leftarrow R$

8: $\boldsymbol{end\ if}$

9: Уменьшить $t$

10: $\boldsymbol{if}\ f\ S\ > f\ B\ \boldsymbol{then}$

11: $B \leftarrow S$

12:$\boldsymbol{end\ if}$

13: $\boldsymbol{until}\,S$ – идеальное решение, или закончилось время поиска, или $t \leq 0$

14: $\boldsymbol{return}\,S$

В алгоритм 3.1 введена дополнительная переменная, которая хранит значение лучшего, на текущий момент, решения. Таким образом, решается проблема потери лучшего решения.

### 3.1.2 Реализация генетического алгоритма

В главе 2 была описана общая схема генетического алгоритм, но в классической схеме, как уже было сказано, оператор скрещивания не удовлетворяет условиям допустимого решения в задачи размещения графа. Покажем на примере, как работает классический одноточечный оператор скрещивания с решениями задачи размещения графа. Допустим, есть два хромосома-родителя, как на рисунке 3.1, с точкой разреза $p = 2$. Нужно применить к ним одноточечный оператор кроссинговера. На рисунке 3.2 показаны хромосомы-потомки, которые получились после применения этого оператора. Видно, получившиеся решения не удовлетворяют критериям кодирования перестановок. У первого потомка в 1 и 6 локусе есть значение, которое повторяется. Также и у второго потомка есть значение генов, которые повторяются в 2 и 5 локусе. Таким образом, появляется проблема применения такого оператора скрещивания. В связи с этим автором был разработан модифицированный одноточечный оператор скрещивания, который исключает подобный недостаток.

| | | | | | | |
|---|---|---|---|---|---|---|
| Родитель 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| | 6 | 5 | 2 | 1 | 3 | 4 |
| Родитель 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| | 5 | 3 | 1 | 4 | 2 | 6 |

Рисунок 3.1 – Классический оператор скрещивания, родители

| | | | | | | |
|---|---|---|---|---|---|---|
| Потомок 1 | 1 | 2 | 3 | 4 | 5 | 6 |
| | 6 | 5 | 1 | 4 | 2 | 6 |
| Потомок 2 | 1 | 2 | 3 | 4 | 5 | 6 |
| | 5 | 3 | 2 | 1 | 3 | 4 |

Рисунок 3.2 – Классический оператор скрещивания, потомки

Опишем новый оператор скрещивания. Допустим, имеется хромосомы-родители и точка разреза как на рисунке 3.1. Если перенести все гены до точки разреза из родителя 1 в потомка 1, а из родителя 2 в потомка 2, а потом перенести гены после точки разреза из родителя 1 в потомка 2, а из родителя 2 в потомка 1, но при встрече гена, который повторяется в потомке пропускать его, двигаясь, таким образом, до конца генов родителей. Если получается так, что гены в потомке не заполнены, то начать брать гены из сначала родителя, то есть делать циклический сдвиг влево.

Проиллюстрируем все вышесказанное. На рисунке 3.3 показаны потомки с примененным модифицированным оператором кроссинговера. Как видно,

этот оператор справился с поставленной на него задачей – ненарушение принципа кодирования перестановка.

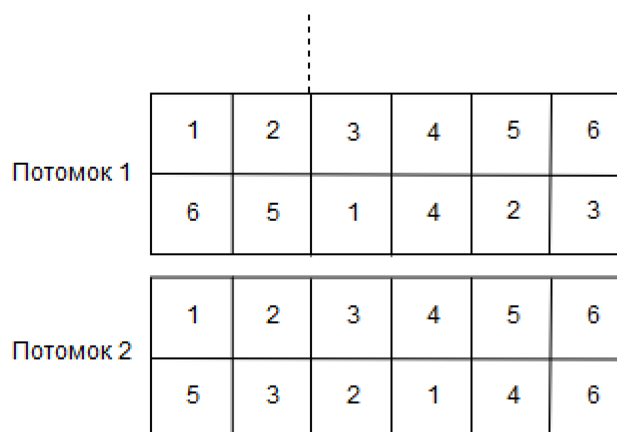


Рисунок 3.3 – Модифицированный оператор кроссинговера, потомки

Также опишем параметры генетического алгоритма, которые применялись при его реализации.

Приведем значения параметров, используемых автором в [7]:

- порог вероятности мутации 0,01-0,1;
- размер популяции 20-30;
- селекция – турнирная, параметр $t = 2$ (количество особей в турнире).

Заполнение исходной популяции – случайное.

### 3.1.3 Эвристический алгоритм Hebene

Кроме реализации уже существующих алгоритмов оптимизации для задачи размещения графа, опишем специальный эвристический алгоритм, который был разработан автором.

Опишем при помощи блок-схем алгоритм, названный акронимом «Hebene» (от англ. «Heuristics of best neighbor»). На рисунке 3.4 изображена блок-схема процедуры HebeneIter. Процедура HebeneIter представляет собой одну итерацию алгоритма Hebene.

Блок-схема алгоритма Hebene, использующая процедуру HebeneIter, приведена на рисунке 3.5 .

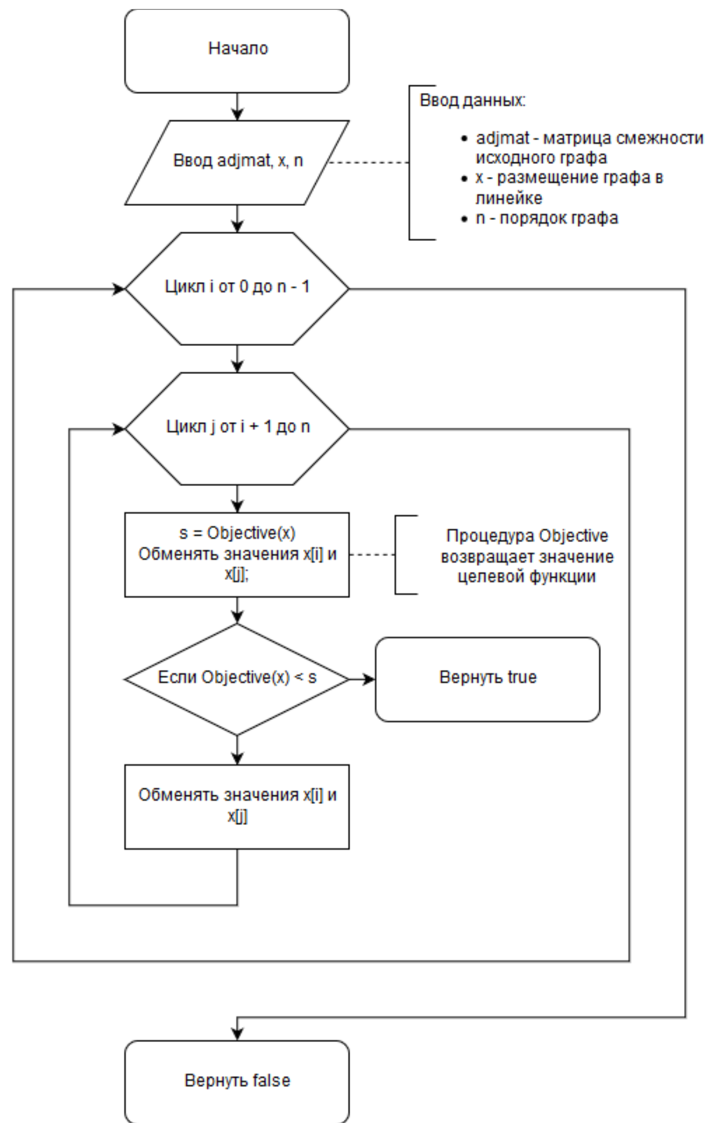


Рисунок 3.4 – Блок-схема итерации алгоритма Hebene

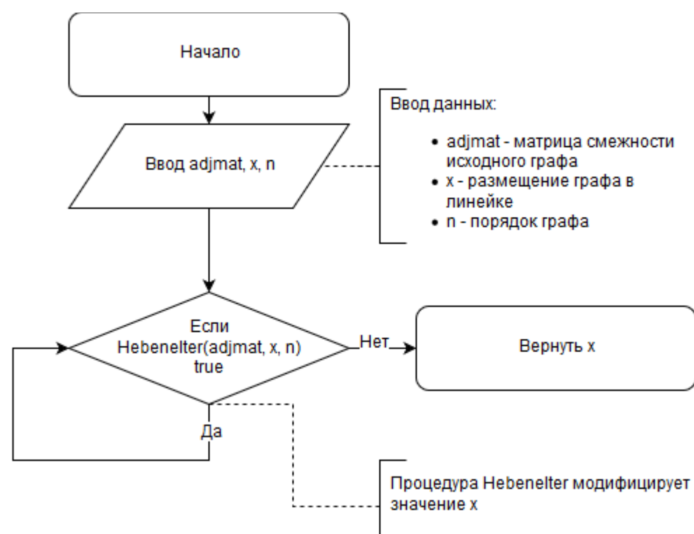


Рисунок 3.5 – Блок-схема алгоритма Hebene

Алгоритм, изображенный на блок-схемах, разделен на две процедуры. Первая процедура обменивает все пары вершин, и если значение целевой функции улучшилось, то она возвращает значение true, в противном случае false. Вторая процедура исполняет первую процедуру до тех пор, пока она не вернет false.

Такой алгоритм имеет некоторые преимущества относительно всех, ранее упомянутых алгоритмов: он является эвристическим и нестохастическим алгоритмом оптимизации для задачи размещения графа.

### 3.1.4 Стохастический алгоритм Hebene

Опишем модифицированный алгоритм Hebene – стохастический алгоритм Hebene (Hebene rand). На рисунке 3.6 показана блок-схема алгоритма.

Если описывать словами, то Hebene rand использует основную процедуру Hebene, которая повторяется $p$ раз. Каждый раз, подавая на вход случайное размещение.

Такой алгоритм, очевидно, при достаточно больших $p$ будет находить оптимальное решение, но в реальности задать достаточно большое $p$ нельзя, так как скорость алгоритма будет сравнима с полным перебором.

Во время реализации этого алгоритма значение $p$ устанавливалось равным 10. Для небольших размерностей графов этого, как увидим дальше, достаточно.

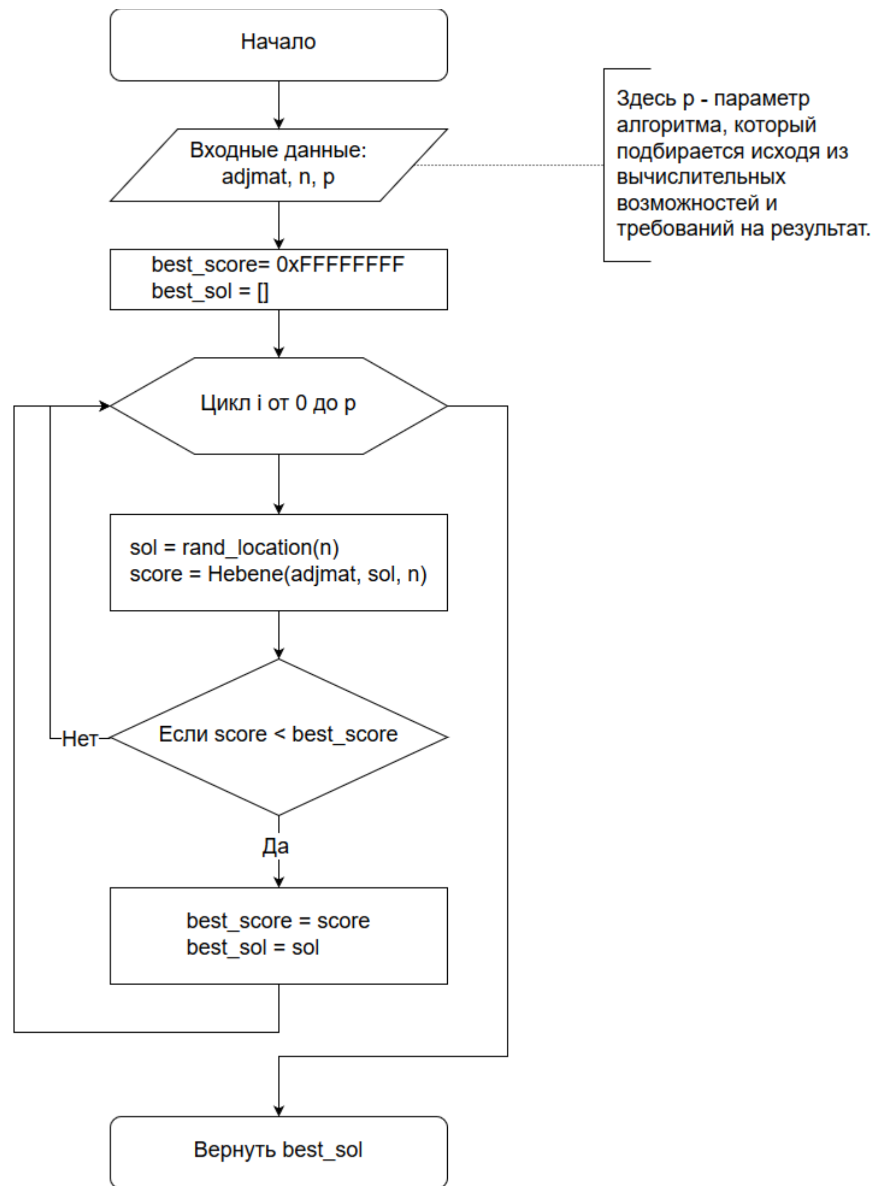


Рисунок 3.6 – Блок-схема стохастического алгоритма Hebene

### 3.2 Разработка программного обеспечения

Здесь опишем только основные моменты разработки программного обеспечения.

Разработка велась на языке C++11 и использованием системы сборки SCons.

Вся программа строится на основе шаблона проектирования Стратегия. Описать его можно следующим образом. Шаблон проектирования Стратегия – поведенческий шаблон проектирования, предназначенный для создания семейства алгоритмов, инкапсуляции каждого из них и создания возможности

их взаимодействия. Шаблон Стратегия позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют.

Этот шаблон позволяет удобным образом организовать иерархию классов алгоритмов. Причем с использованием этого шаблона можно добавлять новые алгоритмы, не изменяя при этом остальную часть программы.

Цель шаблона Стратегия:

- программа должна обеспечивать различные варианты алгоритма или поведения;
- нужно изменять поведение каждого экземпляра класса;
- нужно изменять поведение объектов во время выполнения;
- введение интерфейса позволяет инкапсулировать реализацию алгоритмов и давать возможность на создания своих алгоритмов клиенту.

На рисунке 3.7 показана общая UML-диаграмма шаблона Стратегия.

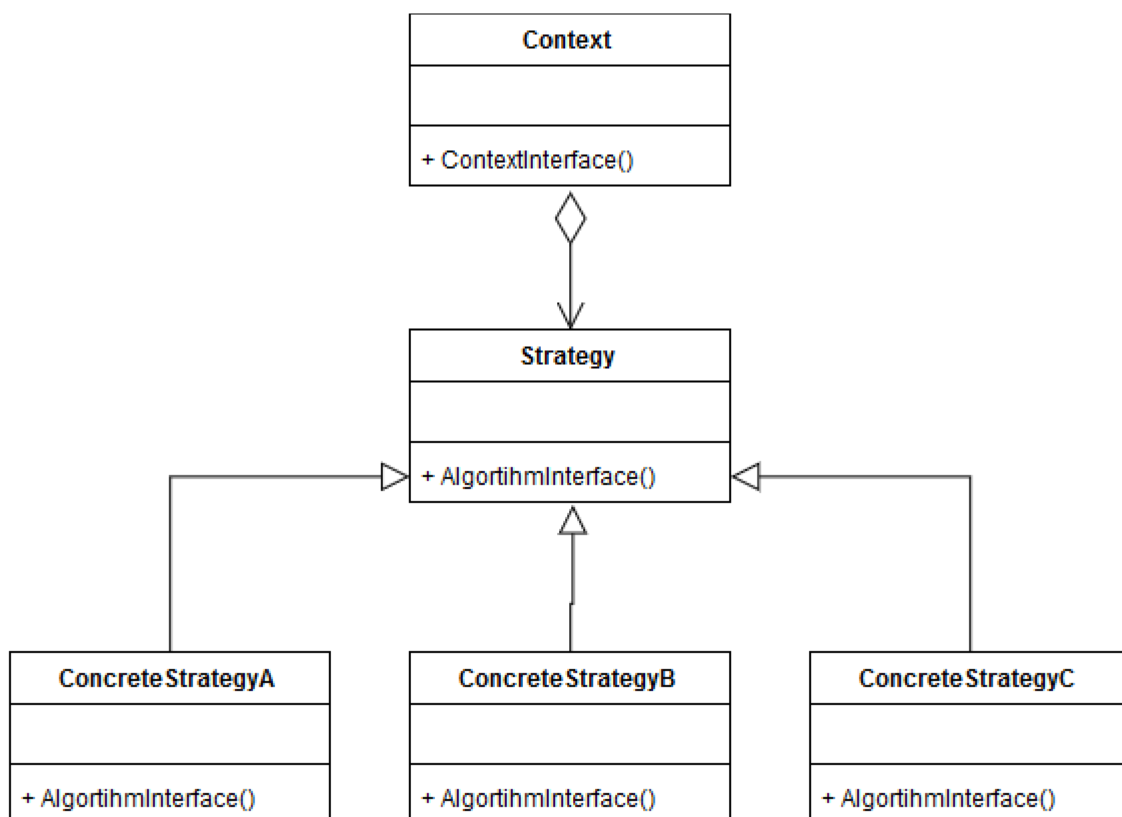


Рисунок 3.7 – UML диаграмма шаблона Стратегия

Для реализации этого шаблона на C++ нужно создать класс-интерфейс алгоритма, который будет выступать базовым классом для других алгоритмов.

В листинге 3.1 приведен интерфейс алгоритм, от которого должны наследоваться другие алгоритмы. Для создания алгоритма необходимо унаследовать этот класс и реализовать следующие методы:

- help() – метод, который нужен для того, чтобы вызывать справку алгоритма;
- run() – метод, который запускает алгоритм;
- __pars_params() – этот метод (приватный) нужен для парсинга параметров, вызывается во всех конструкторах.

Листинг 3.1 – Интерфейс алгоритма

```
class algorithm_t
{
public:
    algorithm_t();
    algorithm_t(scorer_t  scorer,  initializer_t  initializer  =
rand_initializer);

    virtual ~algorithm_t();

    void set_params(const params_t &params);
    void set_initializer(initializer_t initializer);
    void set_scorer(scorer_t scorer);
    std::string report(const std::string &format) const;
    const params_t& params() const;

    virtual std::string help() const = 0;
    virtual void run(const simple_mat_t &mat) = 0;

protected:
    virtual void __pars_params() = 0;

protected:
    scorer_t _scorer = nullptr;
    initializer_t _initializer = nullptr;
    params_t _params;
    params_t _report;
    std::string _format_default;
};
```

В качестве примера приведем полную реализацию алгоритма Hebene. В

листинге 3.2 приведено описания класса для алгоритма Hebene. Из листинга видим, класс algorithm_hebene_t наследуется от класса algorithm_t, далее помимо обязательных методов есть метод __iter, который выполняет одну итерацию алгоритма. Также есть параметра _retry_ti, который отвечает за количество повторений алгоритма.

Листинг 3.2 – Описания класса алгоритма Hebene

```
class algorithm_hebene_t : public algorithm_t
{
public:
    algorithm_hebene_t();
    algorithm_hebene_t(scorer_t scorer, initializer_t initializer
= rand_initializer);

    std::string help() const;
    void run(const simple_mat_t &mat);

private:
    virtual void __pars_params();
    bool __iter(const simple_mat_t &mat, solution_t &sol);

private:
    // params
    uint32_t _retry_ti;
};
```

В листинге 3.3 приведена реализация обязательного метода __pars_params. Этод метод обрабатывает индивидуальный параметры алгоритма, которые передаются через метод set_params.

Листинг 3.3 – Реализация обязательного метода __pars_params

```
void algorithm_hebene_t::__pars_params()
{
    if(_params["retry_ti"].size() != 0)
    {
        _retry_ti = std::stoi(_params["retry_ti"]);
        _retry_ti = (!_retry_ti) ? retry_ti_default : _retry_ti;
    }
}
```

В листинге 3.4 приведена реализация одной итерации алгоритма Hebene, описанная на рисунке 3.5. Переменна _scorer – это указатель на функции оценки решения, то есть на целевую функцию. Таким образом, подменяя этот указатель можно изменять целевую функцию.

Листинг 3.4 – Реализация итерации алгоритма Hebene

```
bool algorithm_hebene_t::__iter(const simple_mat_t &mat, solution_t &sol)
{
    for(int i = 0; i < mat.order() - 1; i++)
    {
        for(int j = i + 1; j < mat.order(); j++)
        {
            float score = _scorer(mat, sol);
            std::swap(sol[i], sol[j]);

            if(_scorer(mat, sol) < score)
                return true;
            else
                std::swap(sol[i], sol[j]);
        }
    }

    return false;
}
```

Наконец, в листинге 3.5 приведена функция работы алгоритма.

Листинг 3.5 – Реализация функции работы алгоритма

```
void algorithm_hebene_t::run(const simple_mat_t &mat)
{
    solution_t sol;
    solution_t best;
    time_interval_t ti;

    time_interval_start(&ti);

    float best_score = 1234567.;

    for(int i = 0; i < _retry_ti; i++)
    {
        _initializer(mat, sol);
        while(__iter(mat, sol));
```

```
        if(_scorer(mat, sol) < best_score)
        {
            best = sol;
            best_score = _scorer(mat, sol);
        }
    }

    // time interval
    _report["time"] = std::to_string(time_interval_notify(&ti));

    // best solution
    _report["solution"] = "";
    for(int i = 0; i < best.size() - 1; i++)
        _report["solution"] += std::to_string(best[i]) + " ";
    _report["solution"] += std::to_string(best[best.size() - 1]);

    _report["score"] = std::to_string(_scorer(mat, best));
}
```

В этом листинге можно заметить вызов функции _initializer. Эта также указатель на функцию, которая должна инициализировать решение задачи размещения графа. По умолчанию этот указатель указывает на функцию, которая случайным образом выбирает решение. А за счет параметра _retry_ti процедуру поиска решения можно повторить. Если параметр _retry_ti больше 1, то, алгоритм Hebene запускается это количество раз, которое содержит _retry_ti, и, таким образом, реализовывается стохастический алгоритм Hebene. Также в листинге можно увидеть обработку специальных переменных отчета. Переменная _report – это ассоциативный массив, ключом которого является имя переменной отчета. Это необходимо для формирования отчета о работе алгоритма.

### 3.3 Демонстрация программного обеспечения

Продемонстрируем программное обеспечение, которое было разработано в процессе исследований. Приложение разрабатывалось без применения графики. Короткая справка вызывается с применением ключа «-h» и показана на рисунке 3.8. Полная справка может быть вызвана, если применить ключ «--help» и показана на рисунке 3.9.

```
loki@Vladislav: ~/.projects/mt18
Файл  Правка  Вид  Поиск  Терминал  Справка
→  mt18 ./mt18
ArgumentParser error: too few required arguments passed to ./mt18
Использование: ./mt18 --algorithm ALGORITHM [--scorer SCORER] [--graph GRAPH [GRAPH..
.]] [--file FILE]
                                  [--params PARAMS [PARAMS...]] [--report-format REPO
RT-FORMAT [REPORT-FORMAT...]] [--outfile OUTFILE]
→  mt18
```

Рисунок 3.8 – Короткая справка программы

```
loki@Vladislav: ~/.projects/mt18
Файл  Правка  Вид  Поиск  Терминал  Справка
Описание алгоритмов
bruteforce
Описание:
        Полный перебор всего пространства решений (то есть все n!)
Переменные отчета:
        %time% - среднее время работы алгоритма
        %solution% - решение в виде номеров вершин
        %score% - значение ЦФ для %solution%
        %full_scan% - показать значение ЦФ всех решений с подсчетом одинаковых значен
ий ЦФ

ga
Описание:
        Генетический алгоритм
Переменные отчета:
        %time% - среднее время работы алгоритма
        %solution% - решение в виде номеров вершин
        %score% - значение ЦФ для %solution%

hebene
Описание:
        Эвристический алгоритм hebene
Параметры:
        retry_ti (по умолчанию 1) - количество повторов алгоритма, чтобы усреднить вр
емя замера
Переменные отчета:
        %time% - среднее время работы алгоритма
        %solution% - решение в виде номеров вершин
        %score% - значение ЦФ для %solution%
        %iters% - количество итераций алгоритма

icg
Описание:
        Проверяет является ли граф связным
Переменные отчета:
        %is% - is connected graph: true or false
        %graph7% - graph in graph7 format

ls
Описание:
        Локальный поиск
Параметры:
```

Рисунок 3.9 – Полная справка программы

Чтобы запустить программу нужно задать следующие обязательные параметры:

- «-a, --algorithm» – алгоритм, который будет обрабатывать данные;
- «-g, --graph» – граф в формате graph7 или «-f, --file» - файл такой, что каждая строка это граф в формате graph7.

Список алгоритмов, которые были реализованы:

- алгоритм имитации отжига;
- алгоритм локального поиска;
- генетический алгоритм;
- алгоритм полного перебора;
- эвристический алгоритм Hebene;
- стохастический алгоритм Hebene;
- алгоритм проверки графа на связность.

Кроме того, в программе есть дополнительные параметры, которые могут быть полезны:

- «-r, --report-format» – задает формат отчета работы алгоритма;
- «-p, --params» – задает параметры алгоритма;
- «-o, --outfile» – задает имя выходного файла, в который будут записываться результаты.

Параметры «-r» и «-p», вообще говоря, являются обязательными, но у них есть значение по умолчанию, то есть они задаются в неявном виде, если они не были переданы.

Пример запуска программы показан на рисунке 3.10. Программа запускалась с параметрами.

```
./mt18 --algorithm ls -f graphs/8c.lst --report-format %score%:%time%
```

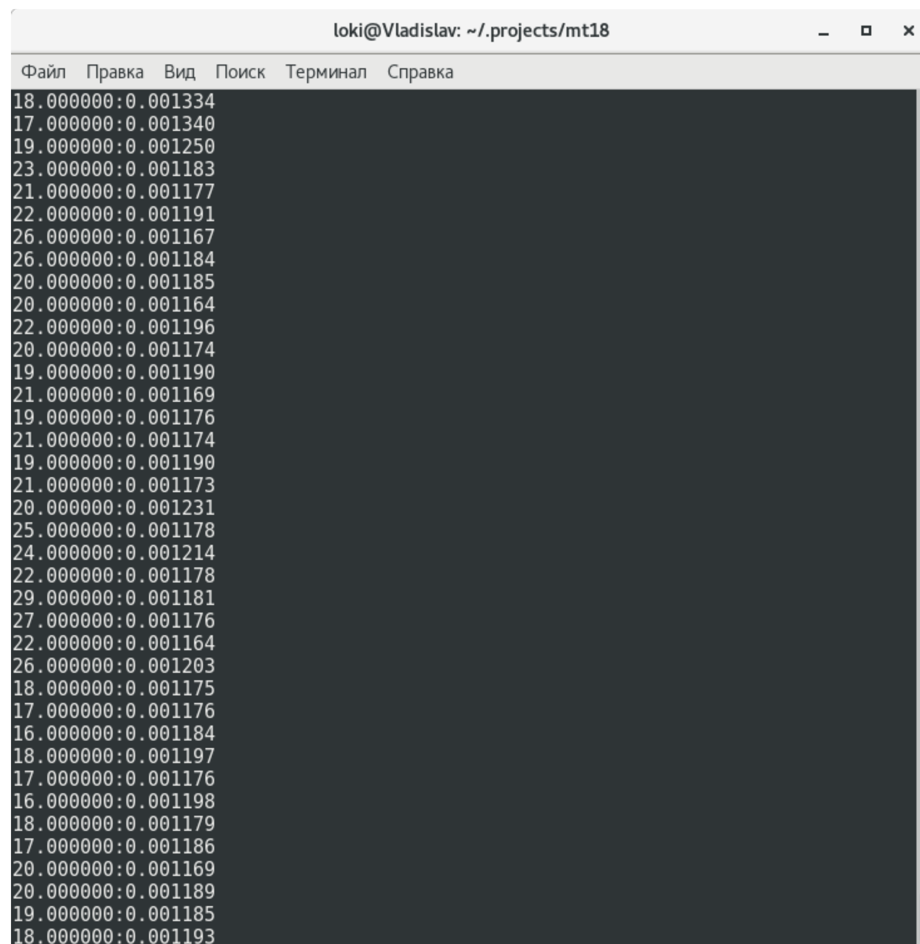


Рисунок 3.10 – Пример запуска программы

Значение параметра «--report-format» задается в виде комбинации обыкновенных символов и переменных отчета, которые задаются в виде %VAR%. Список общих переменных отчета:

- %time% – показывает время работы алгоритма;
- %score% – показывает значение целевой функции;
- %solution% – показывает решение в виде перестановки.

Существуют и другие переменные отчета, но они являются индивидуальными для каждого алгоритма и здесь описываться не будут.

### 3.4 Сбор данных для вычислительных экспериментов

После реализации алгоритмов перед автором встал естественный вопрос о проверки их эффективности. Вообще говоря, существует как минимум несколько способов проверки алгоритмов:

1. проверка эффективности алгоритмов относительно оценки (2.7);

2. проверка эффективности алгоритмов относительно оптимального решения;

3. проверка эффективности относительно друг друга.

Все способы реализуемы, но с некоторыми ограничениями. Для первого способа можно использовать графы больших размеров, так как существует возможность вычислить оценку быстро. Для второго способа графы больших размеров использовать затруднительно, так как невозможно точно сказать, что решение является оптимальным. Наконец, для третьего способа можно использовать любые графы.

### 3.4.1 Генерация случайных графов

Для проверки различных информационных моделей на графах часто требуется большое количество графов большой размерности. В таких случаях применяют алгоритмы генерации случайных графов.

Вообще говоря, существует большое количество способов генерации различных типов графов. В работе использовалось два типа генерации:

- генерация графа с равномерным распределением ребер;
- генерация графа с заранее заданным количеством ребер.

Алгоритм генерации графов с равномерным распределением ребер показан на рисунке 3.11. На рисунке параметр *order* – порядок графа, а *p* – вероятность появления ребра между парой вершин. Функция *frand()* генерирует число в промежутке $[0;1]$. Выходными данным алгоритма является матрица смежности графа *mat*. Функция *is_connected(mat)* нужно для проверки графа на связность, так как для проверки алгоритмов нужны связные графы.

Алгоритм генерации графа с заранее заданным количеством ребер показан на рисунке 3.12. Функция *rand(1, order)* возвращает случайное число от *1* до *order*.

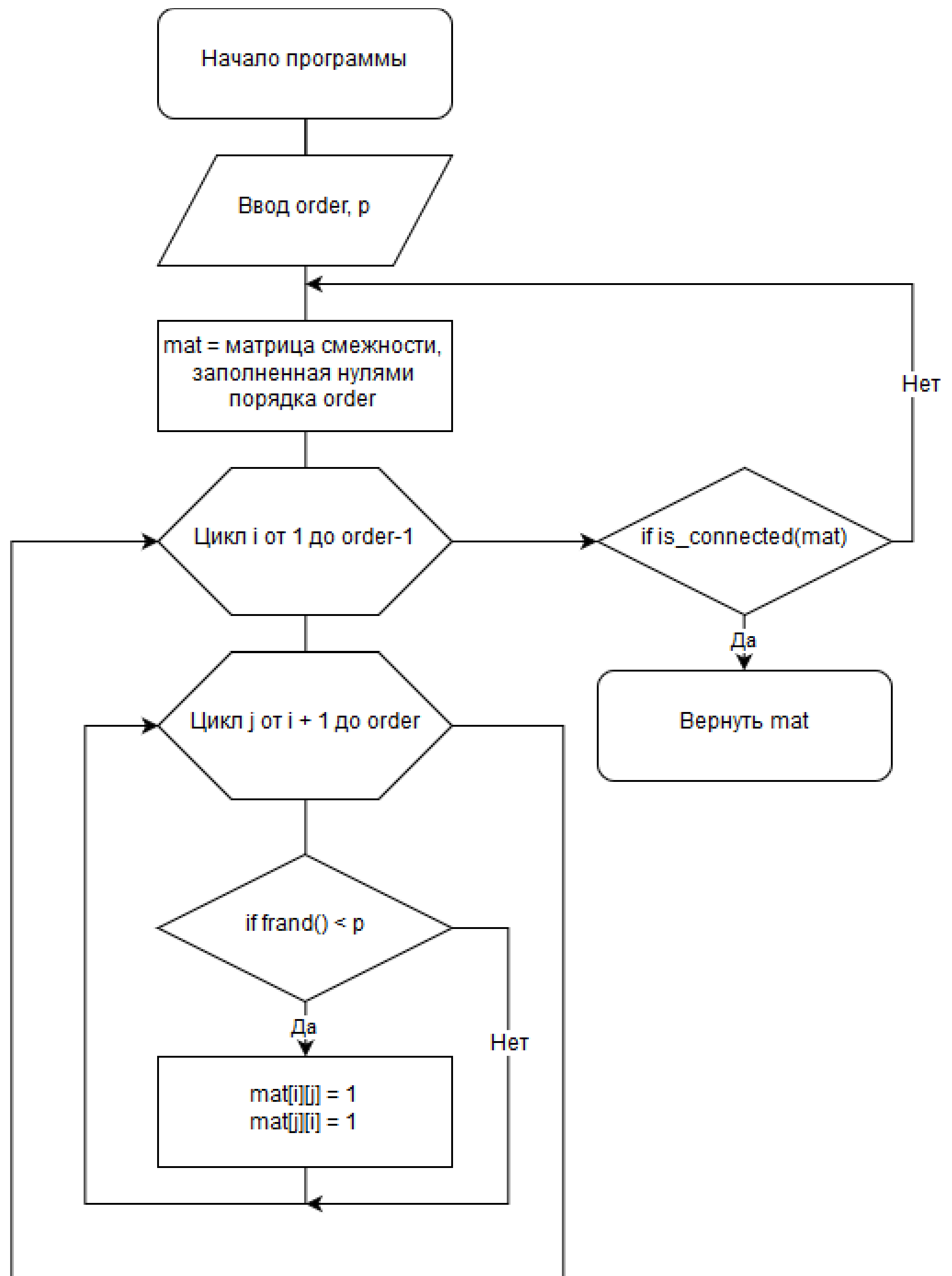


Рисунок 3.11 – Алгоритм генерации графа с равномерным распределением ребер

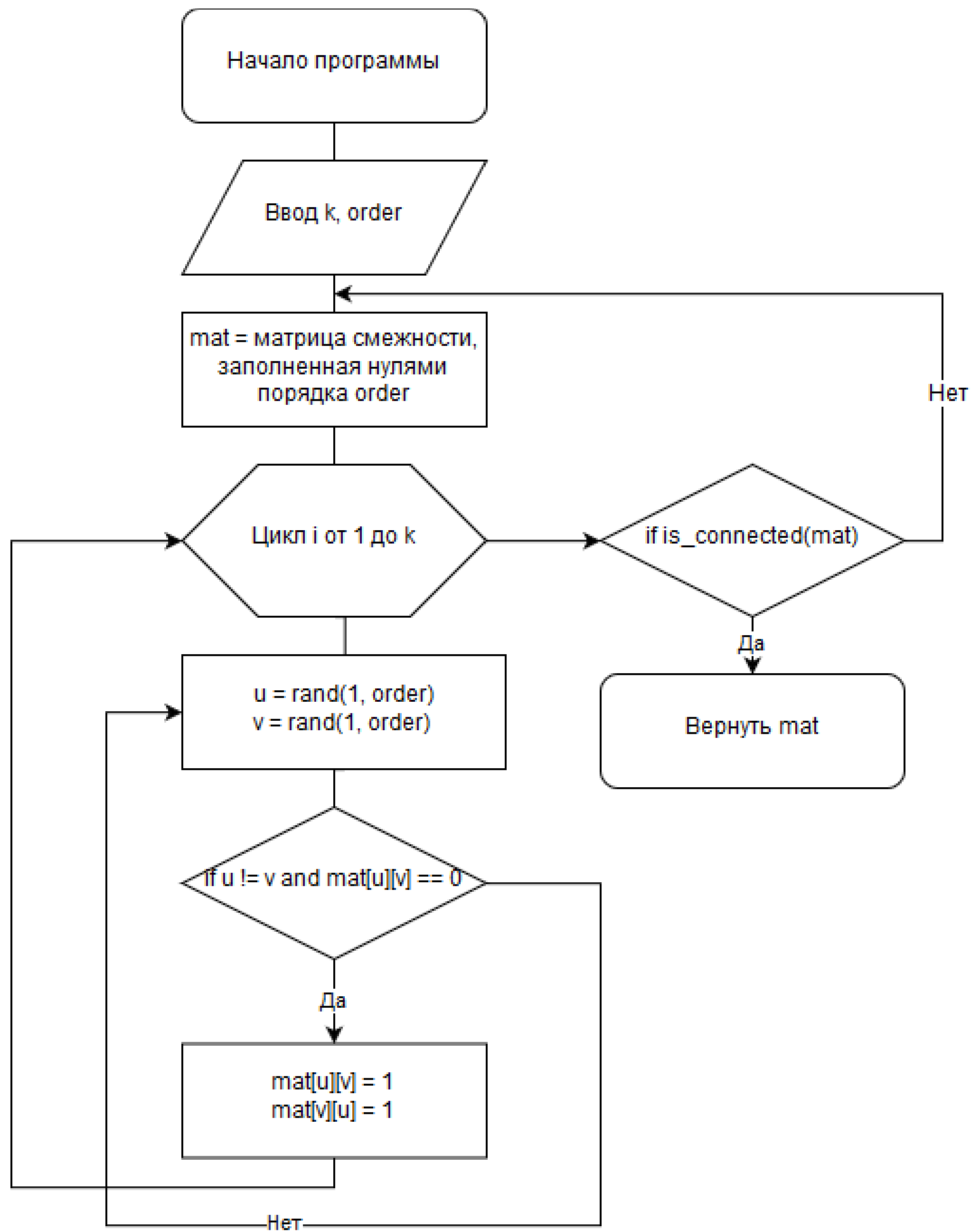


Рисунок 3.12 – Алгоритм генерации графа с фиксированным количеством ребер

Также стоит сказать, что функции $frand$ и $rand$ должны возвращать случайное число с нормальным распределением. Хотя, автор не исключает, что возможно получить интересные свойства, генерируемых графов, с другими распределениями, однако другие распределения во время реализации не использовались.

### 3.4.2 Получение попарно неизоморфных графов

Для проведения вычислительных экспериментов нужно, прежде всего, собрать базу попарно неизоморфных графов. Поскольку если анализировать все пространство графов, то часто будут повторяться результаты, потому что на всем пространстве большинство графов изоморфны друг другу.

Определим изоморфизм на графе формально. Изоморфизмом графов называется биекция между множеством вершин $f\colon V_G \to V_H$ такая, что если любая пара вершин $u$ и $v$ в графе $G$ смежны, то $f(u)$ и $f(v)$ в графе $H$ также смежны. Для примера рассмотрим графы на рисунке 3.13. На первый взгляд кажется, что это два разных графа, однако между ними установлена биекция, которая может перевести метки одного графа в метки другого:

$$f = \begin{matrix} a & b & c & d & g & h & i & j \\ 1 & 6 & 8 & 3 & 6 & 2 & 4 & 7 \end{matrix}$$

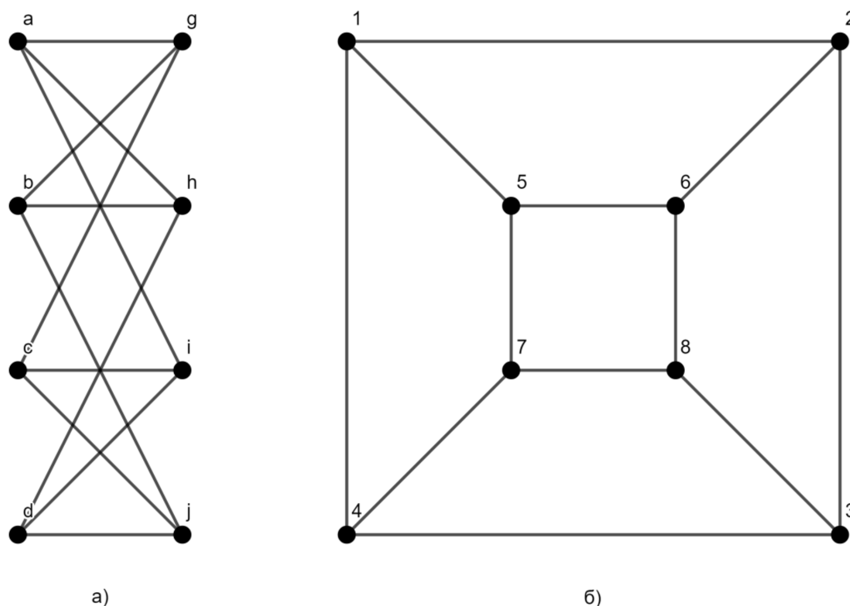


Рисунок 3.13 – Пример изоморфизма графов

В таблице 3.1 для наглядности приведено количество графов всех порядка $n$ и количество попарно неизоморфных графов порядка $n$. Из таблицы видно, что большое количество графов между собой изоморфны и дважды их рассматривать не имеет смысла.

Таблица 3.1 – Сравнение количества всех графов с количеством попарно неизоморфных графов

| $n$ | Количество всех графов | количество попарно неизоморфных графов |
|---|---|---|
| 2 | 2 | 2 |
| 3 | 8 | 4 |
| 4 | 64 | 11 |
| 5 | 1024 | 34 |
| 6 | 32768 | 156 |
| 7 | 2097152 | 1044 |
| 8 | 268435456 | 12346 |
| 9 | 68719476736 | 274668 |
| 10 | 35184372088832 | 12005168 |

Таким образом, для получения базы данных графов необходимо выделить только попарно неизоморфные графы.

К сожалению, задача проверки изоморфизма графов на данный момент является труднорешаемой, и для нее не существует эффективного алгоритма. Но исследования в этой области выделяют некоторые методы проверки изоморфизма графов, а именно нахождения инварианта двух графов и проверка их совпадения. Для начала определим инвариант графа.

Инвариант графа $G$ – это некоторое скалярное или векторное значение, характеризующее структуру графа и не зависящее от способа обозначения вершин. Обозначим его $I(G)$.

Инварианты разделяют на: полные и неполные. Полный инвариант $I_f(G)$ – это такой инвариант, что $I_f(G) = I_f(H)$ тогда и только тогда, когда $G$ изоморфен $H$. Неполный инвариант $I_n(G)$ может нарушать это ограничение, однако если $G$ изоморфен $H$, то $I_n(G) = I_n(H)$.

Примером полного инварианта может служить мини-код [16], обозначаемый $\mu_{min}(G)$. Однако вычисление значение полного инварианта занимает большое количество времени и на практике он сложно применим.

Неполных вариантов существует большое количество, что позволяет их комбинировать и определять, что графы неизоморфны для большинства графов.

Примеры неполных инвариантов (лишь небольшой список):

- диаметр графа;
- индекс Винера [17];
- индекс Рандича;
- обхват графа;
- вектор степеней [15];
- вектор степеней второго порядка [14];
- определить матрицы.

У каждого инварианта существуют свои достоинства и недостатки, которые здесь обсуждаться не будут.

Несмотря на то, что нахождения полного инвариант является труднорешаемой задачей, она может решаться для графов небольших размеров за приемлемое время. Поэтому во время создания базы данных неизоморфных графов был использован алгоритм, который здесь обсуждаться не будет, но на с его помощью была собрана база для всех неизоморфных графов до 9-го порядка.

### 3.4.3 Способы хранения большого количества графов

Во время получения большого числа графов возникает вопрос их хранения на компьютере. Существует несколько основных способов хранения:

1. хранения списка ребер;
2. хранения матрицы смежности;
3. хранения матрицы инцидентности;
4. хранения матрицы Киргофа.

Это основные способы хранения, но, кроме прочего, нужно выбрать способ кодирования этой информации. Возможно два варианта:

1. хранение в текстовом виде;
2. хранения в бинарном виде.

Однако при хранении большого количества графов способ 1 совершенно не подходит, так как требует большого количества памяти. Поэтому во время подготовки данных для вычислительных экспериментов был разработан и реализован специальный формат хранения графов – *graph7*.

Формат *graph7* – формат хранения матриц смежности графов в виде ASCII последовательности.

Перед разработкой формата были поставлены критерии, которым он должен удовлетворять:

1. поддержка разных типов графов (унификация формата для различных типов графов), таких как: неориентированные графы, ориентированные графы, графы с петлями и взвешенные графы;
2. каждый граф должен иметь представление в виде текстовой строки ASCII с использованием системы счисления с основанием 64;
3. «не плати за то, что не потребуется» – данный принцип выражается в том, что каждый тип графа должен занимать место столько, сколько ему нужно; то есть для хранения неориентированного графа необходимо место в два раза меньше, чем для ориентированного графа.

Также, во время разработки автор опирался и на другие критерии, которые будут описаны далее.

Цель формата graph7 является минимизация занятого пространства на носителях при хранении большого количества однотипных графов. Во время разработки формата автор опирался на опыт, существующего формата *graph6* (название graph7 является аллюзией на название graph6) [19]. Однако формат graph6 нарушает критерий (1) и поддерживает только неориентированные графы. Также формат graph6 нарушает критерий (3) из-за чего увеличивается

размер, занимаемый одним графов – это выражается в том, что каждая строка графа хранит дополнительный данные для определения порядка графа, и размер данных зависит от порядка графа. В реализации формата graph7 исключено хранения этих данных и является вычисляемым.

Сначала рассмотрим некоторые способы хранения графов на носителях. Основные способы, используемые исследователями, это:

- матрица инцидентности;
- список ребер графа;
- матрица Киргофа;
- матрица смежности.

Из этих общепринятых способов хранения графов автором был выбран способ хранения графа в виде матрицы смежности, потому что:

- имеет фиксированный размер для одного порядка графа;
- для хранения невзвешенных графов значение элемента матрицы является элементом булевого множества (что позволяет хранить элемент в виде одного бита).

Рассмотрим формат graph7 более подробно. На рисунке 3.14 изображена диаграмма расположения данных формата.

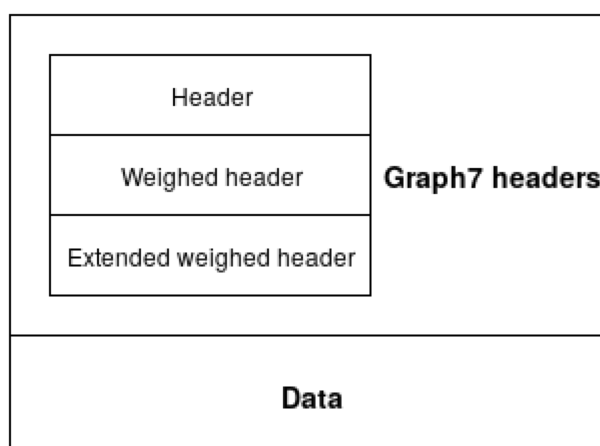


Рисунок 3.14 – Диаграмма расположения данных формата graph7

Опишем каждый элемент диаграммы:

- «Header» – обязательный заголовок;
- «Weighed header» – опциональный заголовок, необходим для взвешенных графов;
- «Extended weighed header» – расширенный опциональный заголовок, необходим для взвешенных графов;
- «Data» – данные (матрица смежности).

Рассмотрим элемент «Header» более подробно. На рисунке 3.15 изображена диаграмма этого элемента. На диаграмме показаны номера бит (порядок от младшего к старшему) и название групп бит (каждая группа бит имеет свое функциональное назначение).

| **Header** | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Weighed | Gtype | | Tail | | | Reserved | |

Рисунок 3.15 – Диаграмма заголовка формата graph7

Опишем функциональное назначение каждой группы бит:

- «Weighed» – бит, указывающий является ли граф взвешенным;
- «Gtype» – пара бит, указывающая на тип графа;
- «Tail» – тройка бит значение которой зависит от группы «Weighed»;
- «Reserved» – служебная пара бит.

Следующий элемент – «Weighed header» показан на рисунке 3.16.

| **Weighed header** | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| Extended | Width | | | | | Reserved | |

Рисунок 3.16 – Диаграмма опционального заголовка

Опишем функциональное назначение каждой группы бит:

- «Extended» – значение выставляется в 1, количество байт необходимое для хранения одной ячейки матрицы превышает 32;
- «Width» – зависит от «Extended» и будет описано далее.
- «Reserved» – служебные биты.

Если при кодировании графа выясняется, что для хранения одной ячейки матрицы нужно больше 32 байт, то бит «Extended» выставляется в 1, а в «Width» записывается значение количества байт, которое необходимо для хранения значения размера ячейки матрицы. Причем для хранения этого значения используются байты не целиком, а только младшие 6 бит, то есть эта информация, хранится в секстетах. Для чего это необходимо будет сказано далее.

Если при кодировании графа оказалось, что для хранения одной ячейки матрицы нужно меньше или равно 32 байт, то это значение записывается в «Width».

В формате graph7 невзвешенные графы можно упаковывать двумя способами:

1. упаковывание каждой ячейки как бита;
2. упаковывание каждой ячейки как какое-то количество байт;

Первый способ предпочтительней для неориентированных графов без дополнительной информации.

Ранее говорилось, что значение поля «Tail» зависит от «Width», эта зависимость выражается в том, что если нужно кодировать каждую ячейка как один бит, то значение «Width» принимает значение 0, а «Tail» будет иметь значение:

$$n \bmod 6.$$

где $n$ – количество ячеек в матрице смежности.

Это значение необходимо для того, чтобы при распаковывании знать какая группа бит лишняя и ее не нужно считать частью матрицы смежности.

Другой случай значение «Tail» рассматривается далее.

После всех заголовков идут данные. Опишем формулы для получения количества ячеек матрицы для разных типов графов:

1. $\frac{n(n-1)}{2}$ – неориентированный граф без петель;
2. $\frac{n(n+1)}{2}$ – неориентированный граф с петлями;
3. $n(n-1)$ – ориентированный граф без петель;
4. $n^2$ – ориентированный граф с петлями.

Следующая матрица смежности является матрицей графа. Выделенные элемента являются значимыми, и именно они сохраняются для 1 типа графа.

$$\begin{matrix} 0 & \mathbf{1} & \mathbf{0} \\ 1 & 0 & \mathbf{1} \\ 0 & 1 & 0 \end{matrix}$$

Для графов типа 2 сохраняется также главная диагональ.

$$\begin{matrix} \mathbf{0} & \mathbf{1} & \mathbf{0} \\ 1 & \mathbf{0} & \mathbf{1} \\ 0 & 1 & \mathbf{0} \end{matrix}$$

Для графов типа 3 сохраняются все элементы, кроме главной диагонали.

$$\begin{matrix} 0 & \mathbf{1} & \mathbf{0} \\ \mathbf{1} & 0 & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & 0 \end{matrix}$$

И наконец, для последнего типа 4 сохраняется вся матрица.

$$\begin{matrix} \mathbf{1} & \mathbf{1} & \mathbf{0} \\ \mathbf{1} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{1} \end{matrix}$$

Таким образом, если нужно закодировать неориентированный граф порядка, например 6, без петель, то количество значимых элементов равно $6 \cdot 5/2 = 15$, а значение «Tail» для 1-го типа упаковывания равно 3. Для того чтобы сохранить 15 байт нужно минимум 3 байта, так как данные для этого типа упаковывания хранятся в секстетах.

После того как заголовок и данные сформированы, происходит кодирование данных. Для кодирования используется численная система с основанием 64. Рассмотрим таблицу 3.2, эта таблица кодирования, которая переводит 6-битное число в ASCII символ. Эта таблица является классической для кодирования base64.

Для упаковывания первого типа достаточно каждому байту соотнести элемент таблицы, так как значение байт этого типа не превышает 63.

Таблица 3.2 – Таблица кодирования base64

| A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|
| I | J | K | L | M | N | O | P |
| Q | R | S | T | U | V | W | X |
| Y | Z | a | b | c | d | e | f |
| g | h | i | j | k | l | m | n |
| o | p | q | r | s | t | u | v |
| w | x | y | z | 0 | 1 | 2 | 3 |
| 4 | 5 | 6 | 7 | 8 | 9 | + | / |

Для второго типа упаковывания значения байт заголовка все равно не превышает 63. Для второго типа упаковывания каждый элемент матрицы состоит из некоторого количества байт (от 1). Поэтому все байты должны быть разбиты на секстеты, а незначащие биты должны быть заполнены 0 (биты, которые нужны, чтобы разбить $n$ байт на группы по 6 бит, где одна группа равна одному байту, а старшие два бита, равны 0). Значение «Tail» для данного типа упаковывания должно хранить значения для распознавания остатка бит, который остался, то есть может быть три типа остатков:

1. $0 \equiv n(\mathrm{mod}\ 3)$;
2. $1 \equiv n(\mathrm{mod}\ 3)$;
3. $2 \equiv n(\mathrm{mod}\ 3)$.

Это значение и сохраняется в «Tail».

Рассмотрим пример кодирования неориентированного графа без петель. На рисунке 3.17 изображен граф – упакуем его и закодируем.

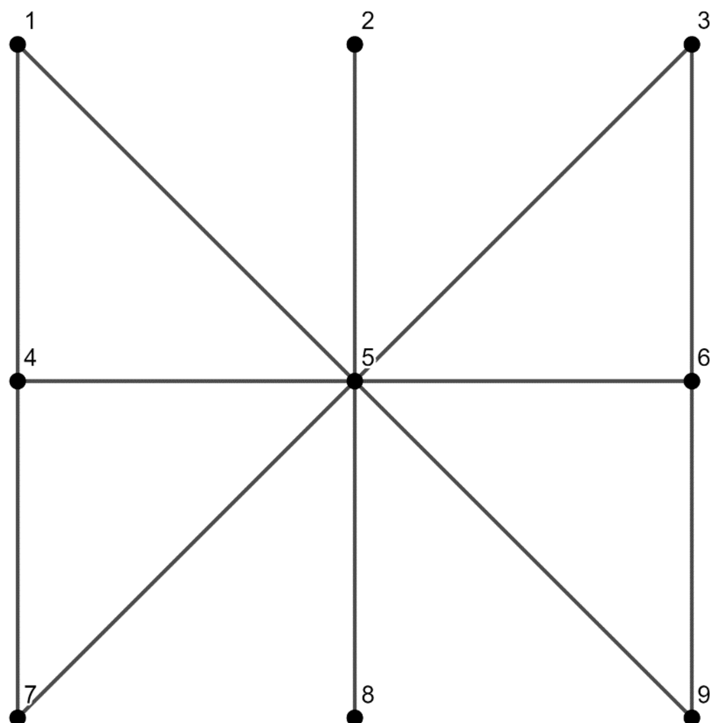


Рисунок 3.17 – Пример графа

Рассмотрим его матрицу смежности.

$$
\begin{matrix}
\mathbf{0} & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & \mathbf{0} & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
1 & 0 & \mathbf{0} & 0 & 1 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & \mathbf{0} & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 0 & \mathbf{0} & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 1 & 1 & \mathbf{0} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & \mathbf{0} & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & \mathbf{0} & 0 \\
0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & \mathbf{0}
\end{matrix}
$$

Поскольку этот граф не имеет петель, то достаточно взять элементы выше или ниже главной диагонали. Для примера возьмем элементы выше главной диагонали: 001100000011000010100111110001000000 – 36 элементов. Сначала упакуем их в соответствии с первым типом. Составим заголовок как в таблице 3.3.

Таблица 3.3 – Пример упаковки графа

| Weighed | Gtype | | Tail | | | Reserved | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | **0** | **0** |

Заголовок получился простым – значение каждого бита равно 0. Теперь разобьем данные на группы бит по 6 бит и добавим к ним два старших бита, равными 0: **00**001100 **00**000011 **00**000010 **00**100111 **00**110001 **00**000000. Теперь объединим заголовок и данные: **00**0000 **00**001100 **00**000011 **00**000010 **00**100111 **00**110001 **00**000000. После упаковки нужно закодировать в соответствии с таблицей 3.1. Рассмотрим каждую группу как число и поставим в соответствие элемент таблицы, тогда получим: «AMDCnxA». Таким образом, был закодирован граф в небольшую ASCII-строку, которая имеет небольшой размер и не хранит избыточную информацию.

Кроме прочего, для того, чтобы хранить большое количество однотипных графов можно заголовок выносить отдельно, и, таким образом, не повторять его каждый раз, экономя память.

### 3.5 Выводы

В главе были рассмотрена реализация различных алгоритмов, которые обсуждались в главе 2. Были описаны конкретные параметры и приведены замечания по реализации. Кроме того, был дополнен алгоритм имитации отжига, что позволило его улучшить. Разработан специальный модифицированный оператор скрещивания для генетического алгоритма.

Разработано два эвристических алгоритма для задачи размещения графов: Hebene и стохастический алгоритм Hebene. Оба алгоритма направлены исключительно на задачу размещения графа и не являются мультиэвристическими.

Также было спроектировано программное обеспечения, позволяющее удобным образом реализовывать все описанные алгоритмы и добавлять в программу.

Была продемонстрирована работа разработанного программного обеспечения.

Кроме того, были показаны методы сбора и хранения данных. Приведены алгоритмы для генерации графов. Приведены методы, которые применялись при сборе всех попарно неизоморфных графов. Описан формат хранения графов graph7, который позволяет существенно экономить место при хранении и передачи большого количества графов. Основой формата graph7 стал формат graph6 и метод кодирования двоичных данных base64. Однако, в отличии от graph6, graph7 имеет фиксированный размер служебных данных и не зависит от порядка графа и, таким образом, экономит дополнительное место. Кроме того, описанный формат позволяет выносить служебный данные для однотипных данных и, тем самым, экономя дополнительное место.

# Глава 4 ПРОВЕДЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ ЭКСПЕРИМЕНТОВ

В этой главе будут описаны вычислительные эксперименты, проведенные для всех реализованных алгоритмов.

Отметим, что все вычислительные эксперименты проводились для задачи размещения графа в линейке и целевой функции (2.6).

## 4.1 Сравнение скорости работы алгоритмов

Во время реализации всех алгоритмов, параметры алгоритмов (стохастических) побирались так, чтобы скорость работы не превышала некоторого полинома (второй степени).

На рисунке 4.1 показан график сравнение скоростей алгоритмов для графов одного порядка. Сравнивались следующие алгоритмы:

- алгоритм полного перебора;
- алгоритм Hebene;
- алгоритм локального поиска;
- алгоритм имитации отжига;
- стохастический алгоритм Hebene;
- генетический алгоритм.

На графике видно, что на небольших графах медленнее всех работает генетический алгоритм, однако это только для небольших размеров. Если же посмотреть на рисунок 4.2, то видно, что скорость работы генетического алгоритма близка к скорости других стохастических алгоритмов, а самым медленным является алгоритм Hebene. Это объясняется тем, что алгоритму Hebene требуется перебирать все пары вершин на каждой итерации алгоритма. Примерно на графах порядка 25 скорость Hebene и Hebene rand начинает заметно ухудшаться в сравнении с другими алгоритмами.

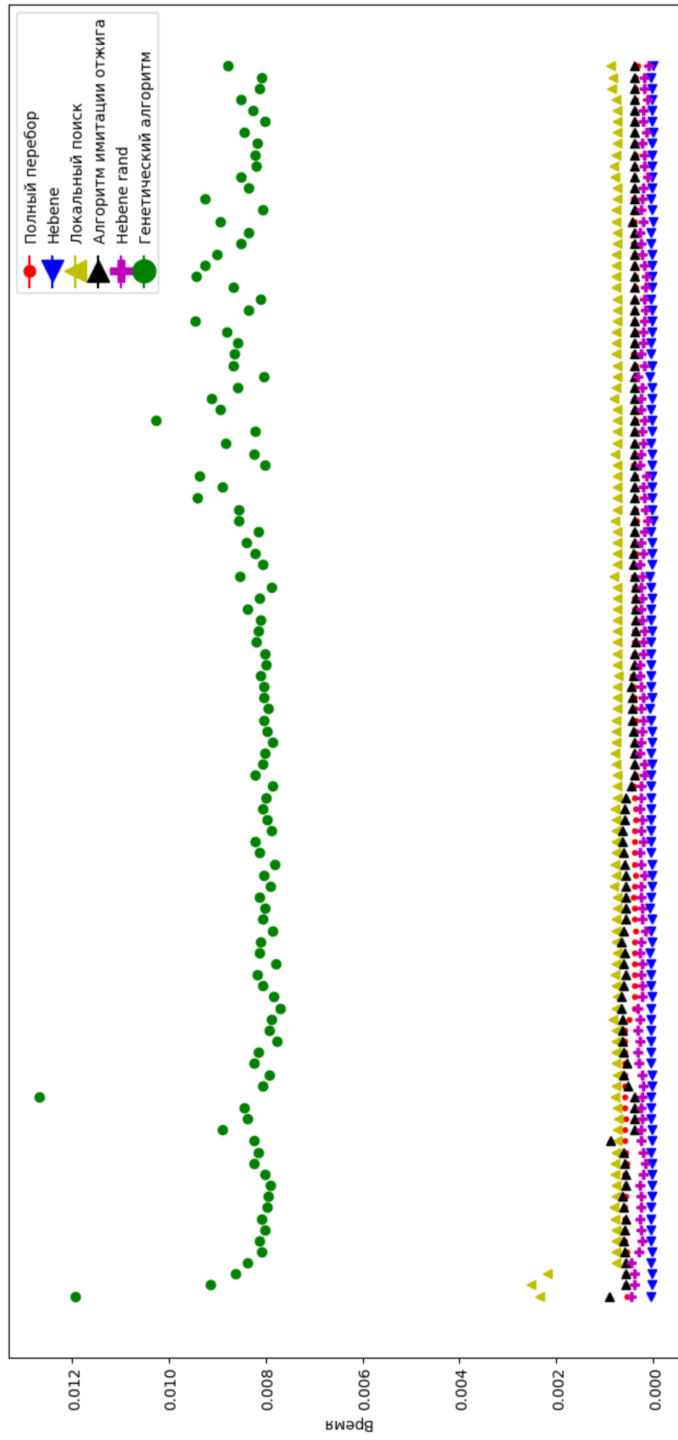


Рисунок 4.1 – Сравнение скоростей алгоритмов на малых размерностях графов

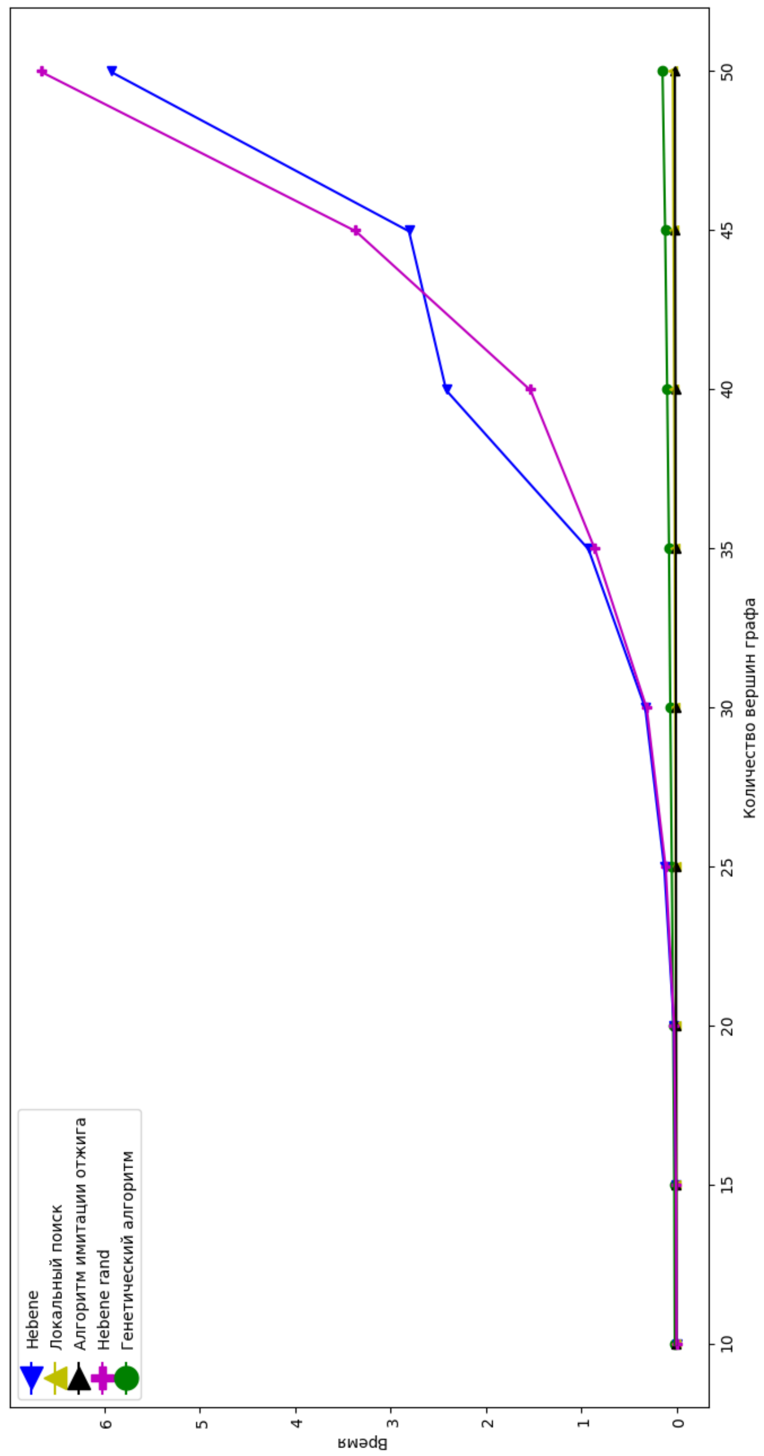


Рисунок 4.2 – График зависимости скорости работы от порядка графа

## 4.2 Сравнение поисковых способностей алгоритмов

Вообще говоря, сравнение поисковых способностей различных эвристических и стохастических алгоритмов всегда задача, которая скрывает за собой то, что сравниваются не только алгоритмы, но и подобранные параметры для этих алгоритмов. На основании любых сравнений этих алгоритмов нельзя сделать однозначный вывод о том, что один алгоритм лучше другого, но можно сказать о том, что один алгоритм лучше другого с конкретными параметрами. Для генетического алгоритма, например, кроме различных стандартных параметров (вероятность мутации, размер популяции и т.д.) на результат еще влияет выбор оператора мутации, кроссинговера, селекции. Для алгоритма имитации отжига большое влияние оказывает закон изменения температуры и начальное значение температуры.

Таким образом, говоря о результатах сравнение поисковых способностей алгоритмов, речь идет о конкретных реализациях (алгоритмических) этих алгоритмов.

### 4.2.1 Сравнение поисковых способностей алгоритмов на графах малых порядков

Взглянем на таблицу 4.1, в ней показаны результаты сравнения алгоритмов, для всех попрано неизоморфных графов малых порядков – 6, 7 и 8 порядка. Строки обозначают порядок графа, а столбцы алгоритм. Элементы таблицы – это процент попадания найденного алгоритмом решения в оптимум.

Таблица 4.1 – Сравнение алгоритмов для графов малых порядков

| | Алгоритм имитации отжига | Локальный поиск | Hebene | Hebene rand | Генетический алгоритм |
|---|---|---|---|---|---|
| 6 | 100% | 92% | 87% | 100% | 97% |
| 7 | 97% | 81% | 83% | 99% | 88% |
| 8 | 93% | 71% | 74% | 99% | 72% |

Из таблицы выше видно, что лучшие поисковые способности для графов малых порядков у стохастического алгоритма Hebene: почти 100% попадание в оптимум для всех рассматриваемых порядков.

Рассмотрим подробнее алгоритм Hebene (так как он является основой для стохастического алгоритма Hebene). Для алгоритма Hebene также были проведены вычисления для графов порядка 9 (и соответствующие вычисления для алгоритма полного перебора). Из всех попарно неизоморфных графов порядка 9 оптимальные решения были найдены для 18961 графов.

На рисунке 4.3 показан график сравнений между оптимальными решениями и решениями, найденными алгоритмом Hebene. На вертикальной оси располагается значение $\Delta = |f\ x^{*}\ \ - f(x^{h})|$, где $f = f_L$, $x^{*}$ – оптимальное решение, $x^{h}$ – решение, найденное алгоритмом Hebene. На горизонтальной оси располагается количество графов с данным значением целевой функции. Например, если есть $n$ графов со значением $\Delta = a$ и $m$ графов со значением $\Delta = b$, и $a < b$, то точки графика будут такими: $(0;n)$ и $(a;m)$. Таким образом, можно увидеть количество решений с соответствующими значениями $\Delta$.

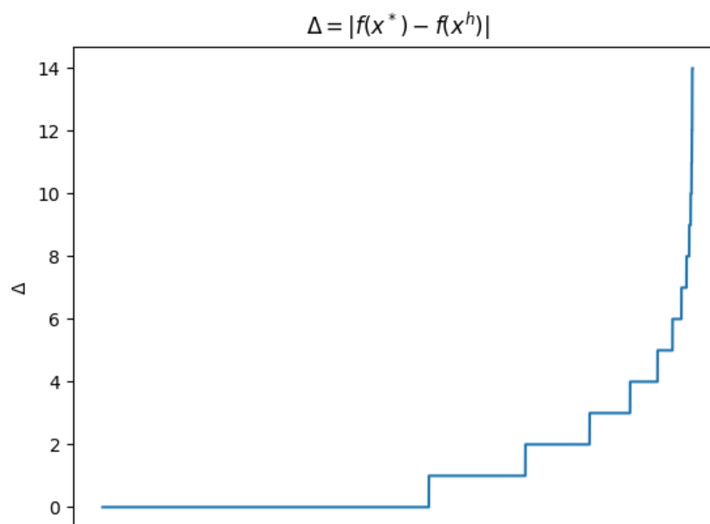


Рисунок 4.3 – График сравнений для графов порядка 9

Из графика на рисунке 4.3 видно, что с ростом значения Δ уменьшается количество графов с этим значением. Для сравнения, посмотрим на рисунок 4.4. На нем показан такой же тип графика для графов порядка 8. Видно, что графики на рисунках 4.3 и 4.4 похоже между собой распределением.

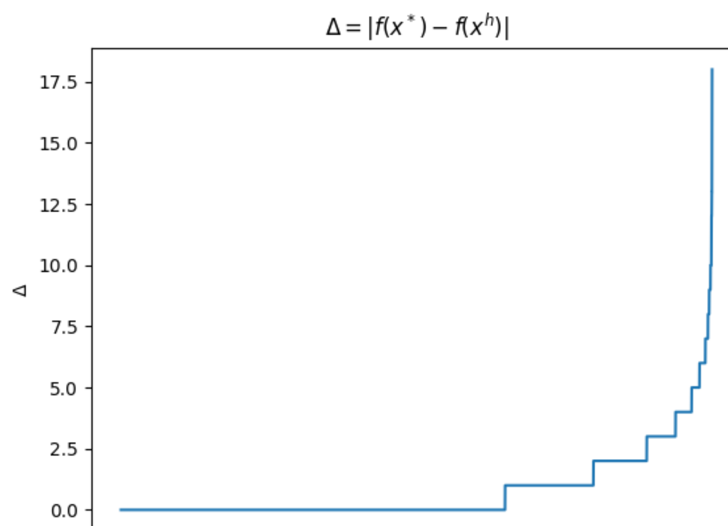


Рисунок 4.4 – График сравнения для графов порядка 8

### 4.2.2 Сравнения поисковых способностей алгоритмов для графов больших порядков

Рассмотрим рисунок 4.5. На нем показан график сравнения поисковых способностей алгоритмов для графов до 50-го порядка. На вертикальное оси располагается значение целевой функции для данного графа, на горизонтальной оси порядок данного графа. Напомним, что рассматривается оптимизационная задача на поиск минимума, поэтому меньшее значение целевой функции означает лучший результат.

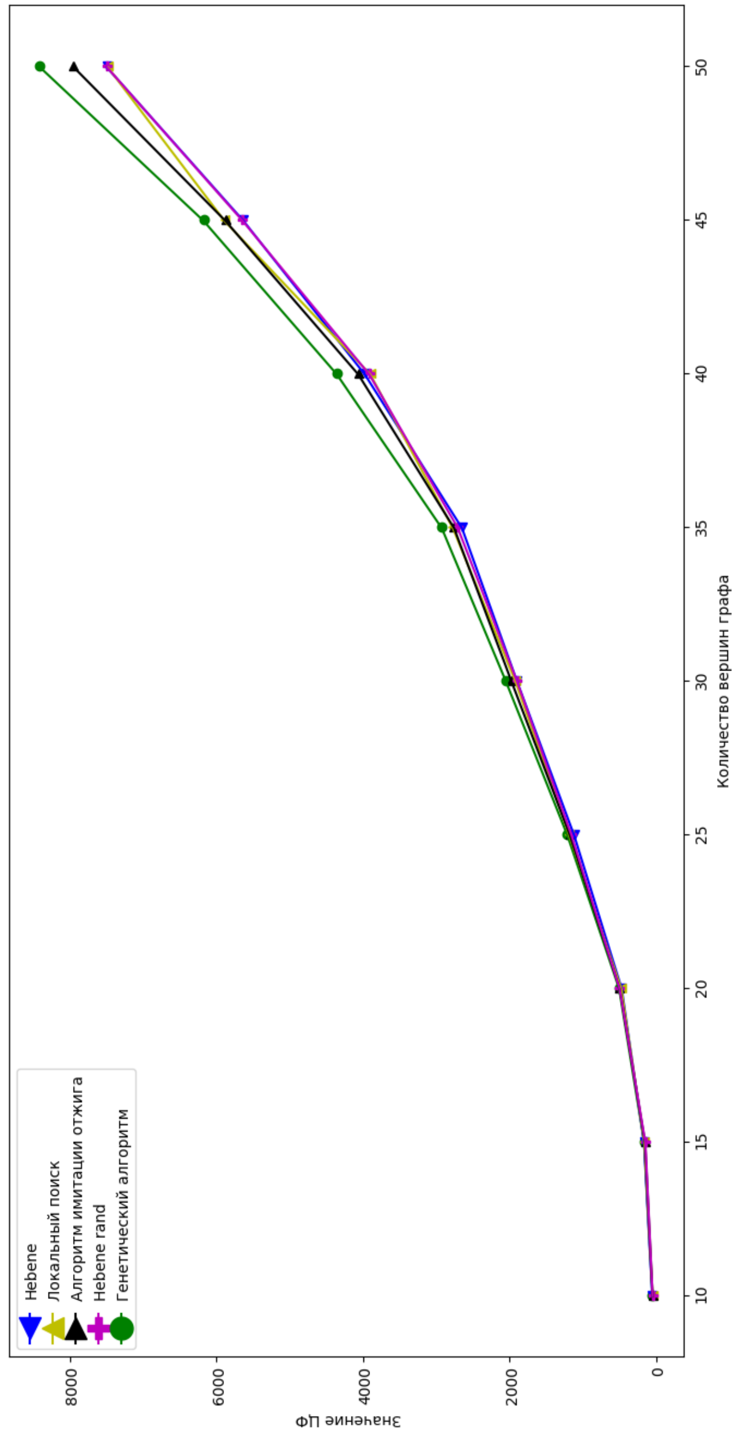


Рисунок 4.5 – Сравнение поисковых способностей алгоритмов для графов больших порядков

Из графика видно, что худшие результаты дает генетический алгоритм. А лучшие результаты дает Hebene и стохастический алгоритм Hebene.

Взглянем на рисунок 4.6. На нем показан график сравнения поисковых способностей алгоритма Hebene и стохастический алгоритм Hebene. Видно, что значение целевой функции, найденных ими решений, почти во всех точках совпадают или незначительно отличаются. Ранее было показано, что для малых порядков графов стохастический алгоритм Hebene давал лучше результат, чем Hebene, а сейчас, получается, результат совпадает. Это объясняется тем, что параметр $r$ для стохастического алгоритма Hebene не изменялся и всегда был равным 10. Для малых размеров значение этого параметра показало хорошие результаты, но на больших размерах этого недостаточно. И вправду, если увеличилось пространство поиска решений, то и вероятность найти оптимальное решение уменьшилась. Поэтому, все дальнейшие сравнения будут проводиться с алгоритмом Hebene, так как его результаты не сильно отличаются от результатов стохастического алгоритма Hebene.

На рисунке 4.7 показан график сравнения поисковых способностей алгоритма Hebene с генетическим алгоритмом. Из графика видно, что с увеличением количества вершин генетический алгоритм все сильнее отстает от Hebene. Для графа с порядком 50 разница между значением целевой функции превышает 900.

На рисунке 4.8 показан график сравнения поисковых способностей алгоритма Hebene с алгоритмом локального поиска. Результаты обоих алгоритмов схожи и показывают, что, несмотря на свою простоту, алгоритм локального поиска дает хорошие результаты. И, как было видно ранее, работает намного быстрее алгоритма Hebene. Однако алгоритм Hebene обладает тем преимуществом, что он не является стохастическим. Это дает возможность более выразительного описание его результатов. Автора полагает, что в дальнейшем сможет объяснить его поведения и, тем самым, изменить статус алгоритма на неэвристический.

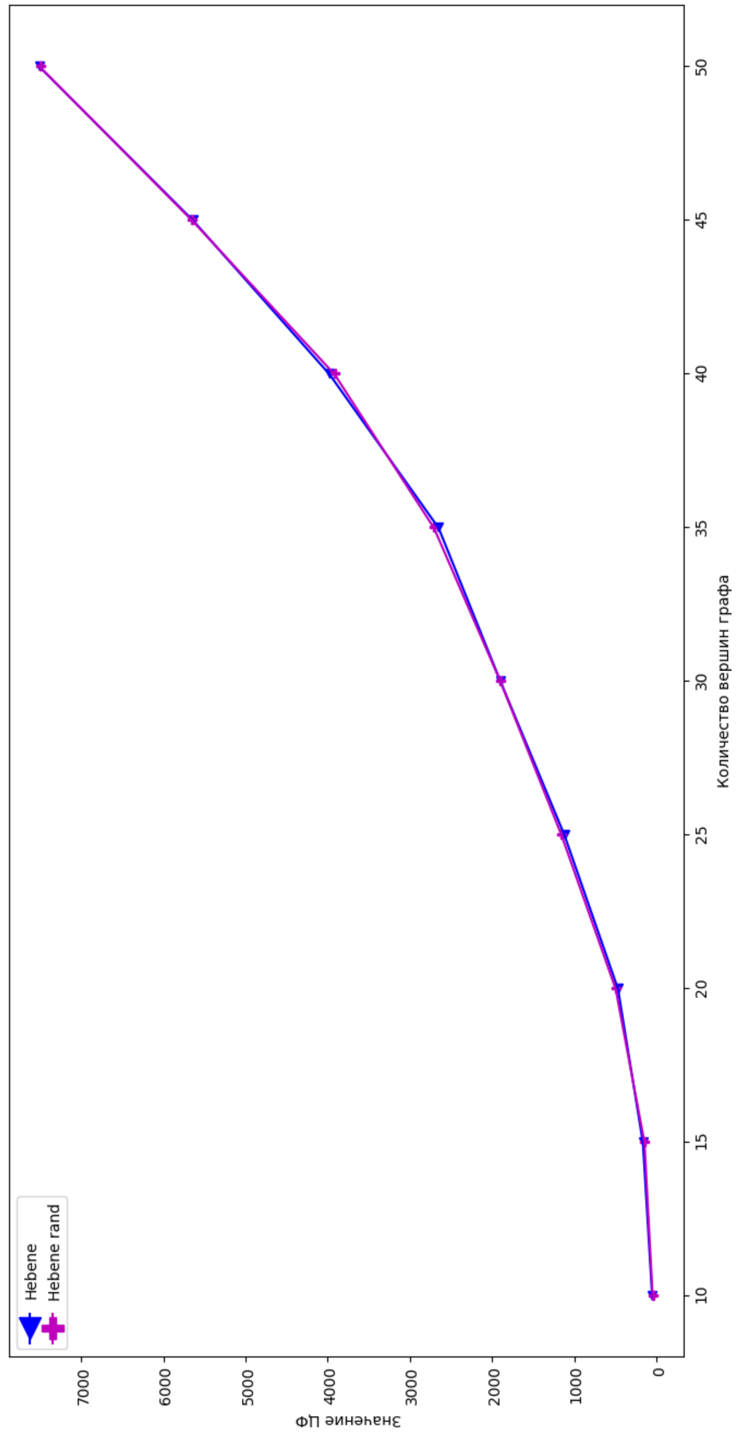


Рисунок 4.6 – График сравнения поисковых способностей алгоритмов Hebene и Hebene rand

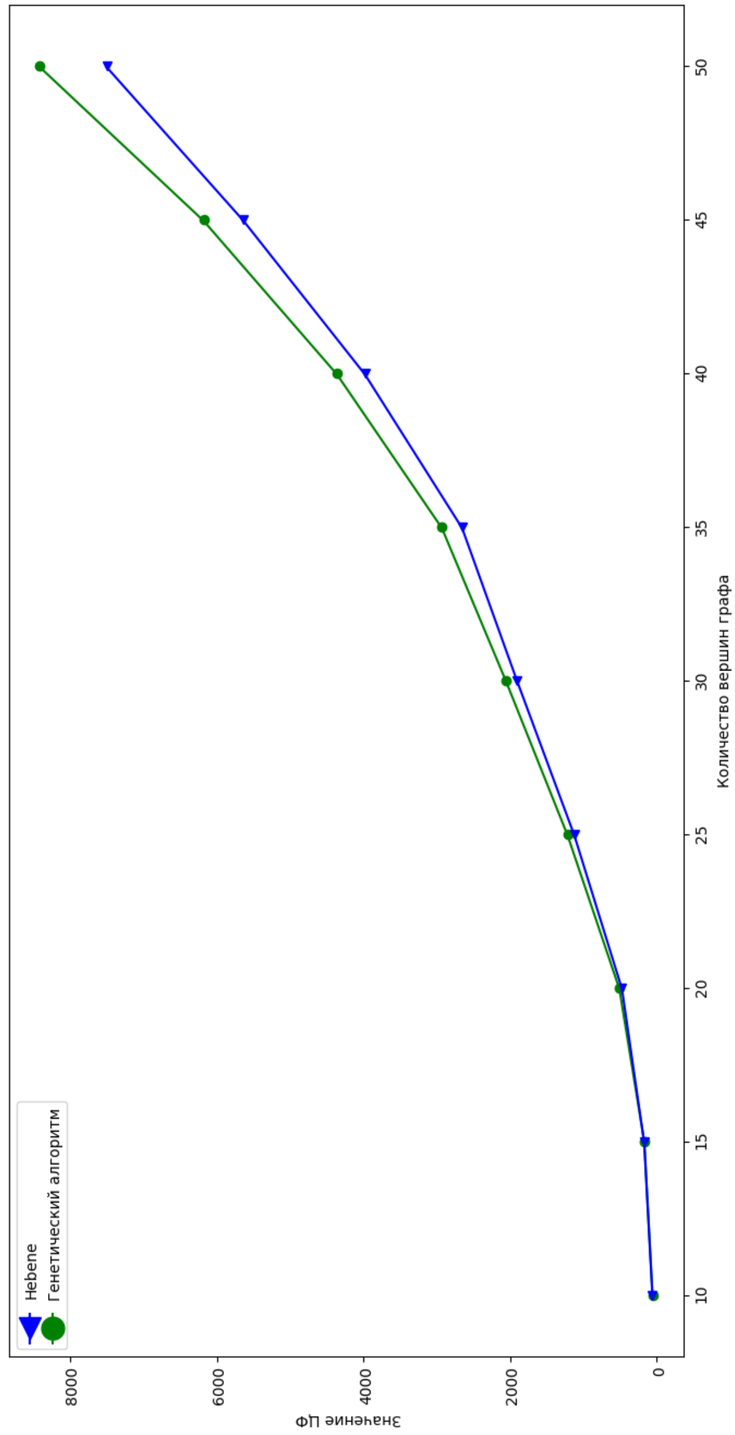


Рисунок 4.7 – График сравнения поисковых способностей алгоритма Hebene и генетического алгоритма

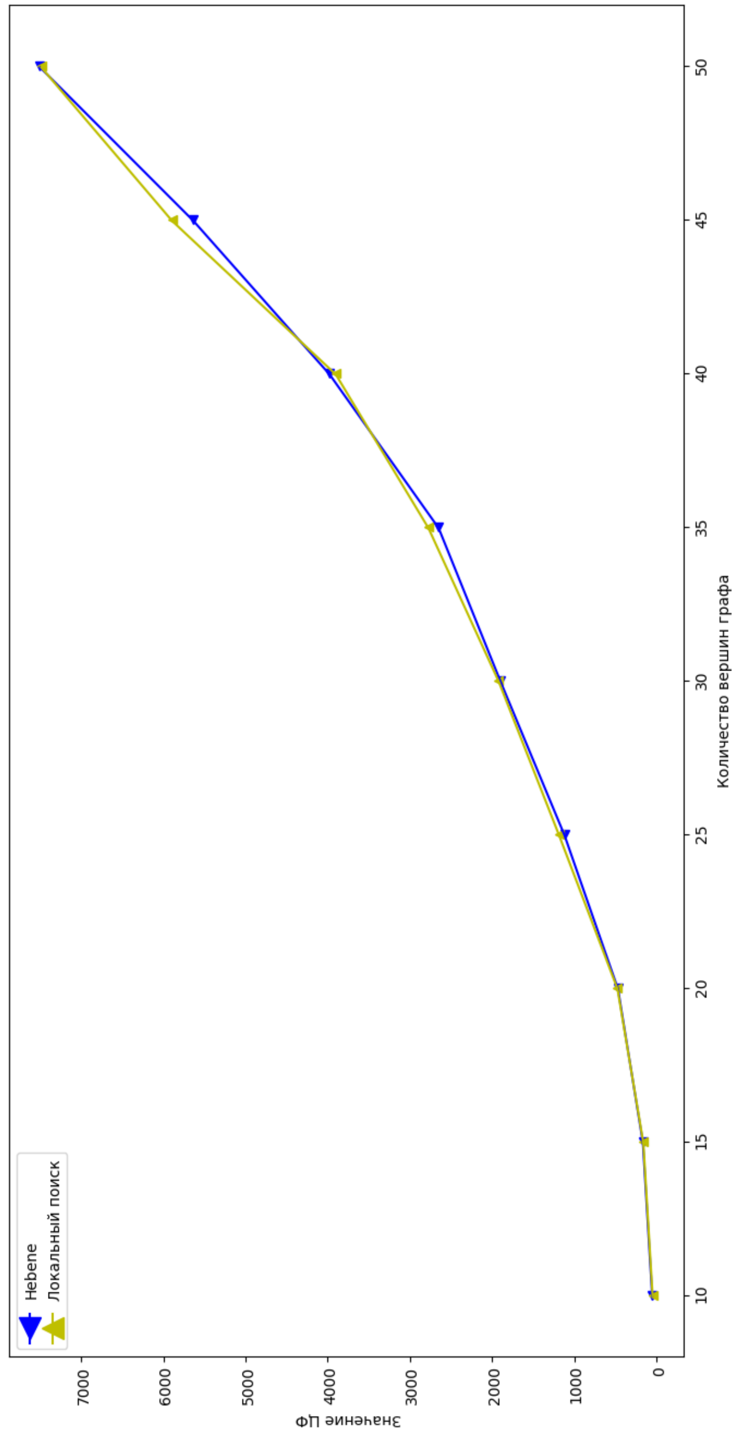


Рисунок 4.8 – График сравнения поисковых способностей алгоритма Hebene с алгоритмом локального поиска

На рисунке 4.9 показан график сравнения поисковых способностей алгоритма Hebene с алгоритмом имитации отжига. Видно, что алгоритм Hebene находит решения лучше с увеличением порядка графа. Однако стоит отметить, что для исследования графов больших порядков использовались те же параметры, что и при исследовании графов малых порядков. Если, например увеличить параметр температуры, то решения, по всей видимости, будут лучше этих, но тогда скорость алгоритма также уменьшится.

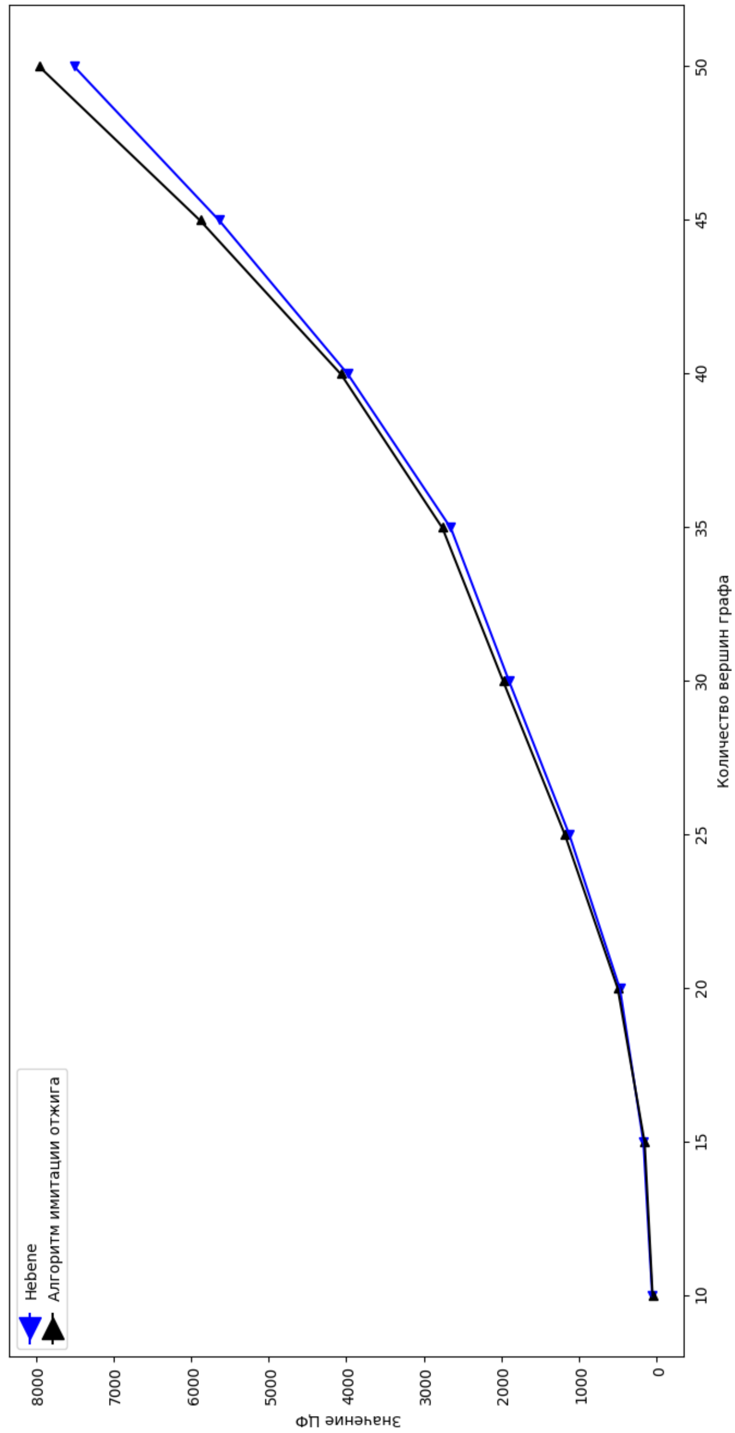


Рисунок 4.9 – График сравнения поисковых способностей алгоритма Hebene с алгоритмом имитации отжига

### 4.3. Некоторые вспомогательные вычислительные эксперименты

#### 4.3.1 Оценка решений графов на отклонение от оптимальных решений

В главе 2 была введена специальная оценка $\mu_{min}(n, p)$ на возможное минимальное значение целевой функции для графов, где $n$ – количество вершин графа, $p$ – количество ребер графа. Проведем некоторые результаты вычислительных экспериментов этой оценки, чтобы оценить реальную ее пользу.

Рассмотрим следующую функцию:

$$O^{\mu}\ G\ = \frac{{}_{g\in G}\,\mu_{min}(\ V\ ,\ E\ )}{{}_{g\in G}\, f(x^{*})}.$$

Здесь $G$ – множество всех попарно неизоморфных графов, а $g = (V, E)$. То есть функция $O^{\mu}$ суммирует значение оценки для всех графов и делит это значение на реальную сумму минимальных значений для всех графов. В результате получается среднее отклонение. В таблице 4.2 описано значение функции для попарно неизоморфных графов порядка 7, 8 и 9 (для порядка 9 результат не для всех графов, как говорилось ранее).

Таблица 4.2 – Результаты функции $O^{\mu}$

| $n$ | 7 | 8 | 9 |
|---|---|---|---|
| $O^{\mu}(G)$ | 0,849909 | 0,821893 | 0,731883 |

Видно, что с ростом порядка среднее отклонение увеличивается. Это связано с тем, что с ростом порядка графа увеличивается и пространство поиска, и, следовательно, количество разнообразных решений. Поэтому функция $\mu_{min}$ нуждается в доработках, а также требуется более надежный механизм ее проверка, а именно проверка на графах больших размерностях. Но, как уже говорилось, с этим связаны трудности, которые не позволяют узнать достоверность того, что решение является оптимальным для больших порядков графов.

### 4.3.2 Пример размещения графа в сетке

На рисунке 4.9 показан пример задачи размещения графа в сетки. Видно, что такое размещение имеет множество пересечений и трудно воспринимается человеком. На рисунке 4.10 приведен пример решения этого графа с помощью алгоритма Hebene.

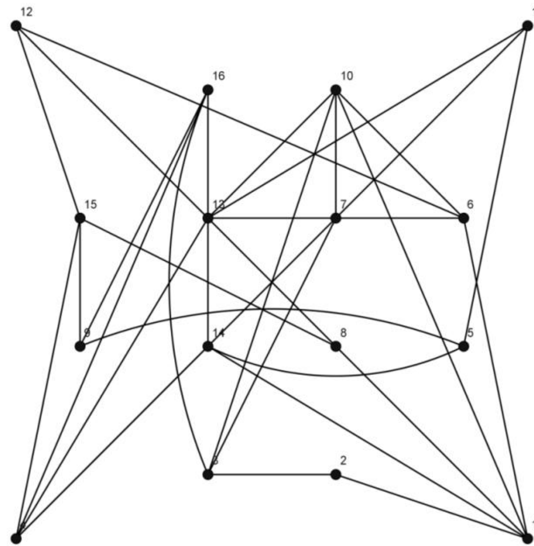


Рисунок 4.9 – Пример задачи размещения графа в сетке

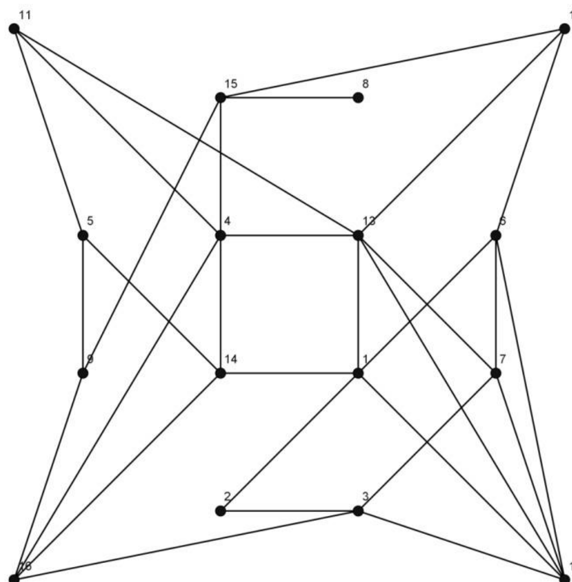


Рисунок 4.10 – Пример решения задачи размещения графа в сетке

Отметим, что для решения такого графа полным перебором пришлось перебирать 20922789888000 возможных решений (то есть $16!$).

### 4.4 Выводы проведенных вычислительных экспериментов

В этой главе были проведены вычислительные эксперименты с реализованными алгоритмами с целью их сравнения на скорость работы и поисковые способности для задачи размещения графа.

Также были проведены вычисления для выявления качества оценки $\mu_{min}$, предложенной в главе 2.

На основании проведенных сравнений скорости работы алгоритмов, можно сказать, следующее. Для малых порядков графов быстрее всех работают алгоритмы Hebene и Hebene rand. С увеличением порядка графов, функция зависимости скорости работы увеличивается быстрее для алгоритмов Hebene и Hebene rand, чем для других алгоритмов. Можно сказать, что скорость работы алгоритма локального поиска, алгоритма имитации отжига и генетического алгоритма близка и может регулироваться за счет изменения их параметров. Для алгоритма Hebene скорость работы изменяться за счет параметров не может и является некоторой полиномиальной функцией.

Анализ поисковых способностей всех алгоритмов для графов малых порядков выдвигает два лидера:

- стохастический алгоритм Hebene, результаты которого показывают не меньше 99% попаданий в оптимум;
- алгоритм имитации отжига показывает результаты не меньше 93% попадания в оптимум.

Остальные алгоритмы показали примерно равные результаты. Однако, рассмотренные графики распределения решений для малых порядков графов показывают, что алгоритм Hebene в большинстве случаев попадает в оптимум, а с увеличением разницы между оптимальным решением и решением, найденным алгоритмом Hebene уменьшается количество решений с таким значением разницы.

Стоит сказать, что поисковые способности алгоритма Hebene сильно зависят от начального значения, однако это свойство использовалось только в стохастическом алгоритме Hebene, и не было никак рассмотрено.

Сравнение поисковых способностей алгоритмов для графов больших порядков показал, что хуже всего работает генетический алгоритм. Остальные алгоритмы показали похожие результаты, но все же алгоритм имитации отжига показал себя хуже, чем другие алгоритмы (кроме генетического алгоритма). Это связано со слишком малым значением температуры, так как на малых значениях температуры алгоритм мало отличается от алгоритма локального поиска. Однако увеличение значения температуры уменьшило бы и скорость его работы.

Была проверена оценка $\mu_{min}$, предложенная в главе 2. Результаты проверки показали, что отклонение оценки от истинного значения увеличивается с увеличением порядка графов. Оценка не оправдала ожидания и плохо сравнима с истинным значением целевой функции, но она все равно может применяться для примерной оценки решений, так как других оценок нет.

На основании проведенных вычислительных экспериментов можно сказать, предлагаемый алгоритм Hebene показывает хорошие результаты на малых и больших порядках. На больших порядках он показал результаты сравнимые с другими алгоритмами (или лучше). На малых порядках модификация алгоритма Hebene стохастический алгоритм Hebene показал лучшие результаты с большим отрывом от других алгоритмов:

- на 6% больше попадания в оптимум для алгоритма имитации отжига;
- на 28% больше попадания в оптимум для алгоритма локального поиска;
- на 27% процентов больше попаданий в оптимум для генетического алгоритма.

Также в качестве примера была приведена задача размещения графа в сетки и ее решение при помощи алгоритма Hebene.

# ЗАКЛЮЧЕНИЕ

В результате проведенного исследования была построена математическая модель задачи размещения графа, и, на ее основе, доказана теорема об NP-полноте задачи.

Была доказана теорема о том, что решение оптимизационной задачи размещения графа влечет за собой решение задачи размещения графа как задачи распознавания свойств для одной целевой функции.

Были разработаны два алгоритма, решающие задачу размещения графа. Алгоритм Hebene и стохастический алгоритм Hebene. Во время проведения вычислительных экспериментов, разработанные алгоритмы оказались сравнимы (или лучше) других алгоритмов решения задачи. В частности, при проведении вычислительных экспериментов на малых размерностях графов стохастический алгоритм Hebene показал 99% попадания в оптимум. В это же время алгоритм Hebene показал 72% попадания в оптимум, что оказалось сравнимо с результатами алгоритма локального поиска и генетического алгоритма, но оказался хуже, чем алгоритм имитации отжига – 94%. С увеличением размерности графов ситуация стабилизировалась и алгоритм Hebene оказался сравним с другими алгоритмами или оказался лучше них.

Для проведения вычислительных экспериментов были собраны попарно неизоморфные графы порядка 6, 7, 8 и 9, и разработан алгоритм генерации случайных графов. Кроме того, был разработан специальный формат хранения графов graph7, который позволяет хранить большое количество графов с минимальной избыточностью данных. Формат graph7 оказался более успешным, чем формат graph6: есть возможность хранить разные типы графов, место, занимаемое одним графов, не зависит от порядка этого графа.

Была получена оценка, позволяющая оценивать решения на их отклонение от оптимального решения для одной целевой функции.

# СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ

*Научная и методическая литература*

1. Балюк, Л. В. Генетические алгоритмы решения задачи размещения элементов СБИС // Известия Южного федерального университета. Технические науки, 2006. - С. 65-71.

2. Александров А.Ю. Математическое моделирование и исследование устойчивости биологических сообществ: учеб. пособие / А.Ю. Александров, А.В. Платонов, В.Н. Старков, Н.А. Степенко. — СПб.: Издательство «Лань», 2016. — 272 с.

3. Жабко А.П. Дифференциальные уравнения и устойчивость: учебник / А.П. Жабко, Е.Д. Котина, О.Н. Чижова. — СПб.: Издательство «Лань», 2015. — 320 с.

4. Прасолов А.В. Математические методы экономической динамики: учеб. пособие / А.В. Прасолов. — СПб: Издательство «Лань», 2015. — 352 с.

5. Бондаренко, И. Б., Каляева, Е. А., Кокшаров, Д. Н. Адаптация параметров генетического алгоритма для оптимизации сложных функций // Известия высших учебных заведений. Приборостроение №9, 2011.

6. Борознов, В. О. Исследование решения задачи коммивояжера // Вестник астраханского государственного технического университета. Серия: Управление, вычислительная техника и информатика, №2, 2009.

7. Громкович, Ю., Мельников, Б. Ф. Алгоритмизация труднорешаемых задач. Часть I. Простые примеры и простые эвристики // Философские проблемы информационных технологий и киберпространства, 2013.

8. Громкович, Ю., Мельников, Б. Ф. Алгоритмизация труднорешаемых задач. Часть II. Более сложные эвристики // Философские проблемы информационных технологий и киберпространства, №1, 2014.

9. Гэри, М., Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон - М.: Мир, 1982.

10. Дудников, В. А. Генетический алгоритм решения оптимизационной задачи размещения вершин графа в линейке // Эвристические алгоритмы и распределённые вычисления, №2, 2015. - С. 60-68.
11. Дудников, В. А., Мельников, Б. Ф. Об NP-полноте задачи размещения графа // Прикладная математика и информатика: современные исследования в области естественных и технических наук, 2017.
12. Емельянов, В. В. Теория и практика эволюционного моделирования / В. В. Емельянов, В. В. Курейчик, В. М. Курейчик. - М.: Физматлит, 2003.
13. Лисяк, М. В. Гибридный алгоритм многокритериального размещения элементов СБИС // Известия Южного федерального университета. Технические науки, №7, 2012. – С. 77-84.
14. Кострова, В. Н., Шендрик, В. А. Генетический алгоритм многокритериальной оптимизации комбинированных графиков для производственной системы на основе гибкого цеха поточного производства // Вестник Воронежского государственного технического университета, 2009.
15. Курейчик, В. В., Запорожцев, Д. Ю. Современные проблемы при размещении элементов СБИС // Известия Южного федерального университета. Технические науки, 2011. - С. 65-71.
16. Мельников, Б. Ф., Дудников, В. А. О Задаче псевдооптимального размещения графа на плоскости и эвристиках ее решения // Информатизация и связь, №1, 2018. - С. 63-70.
17. Мельников, Б. Ф., Сайфулина, Е. Ф. Генерация графов с заданным вектором степеней второго порядка и задача проверки изоморфизма // Стохастическая оптимизация в информатике, №2, 2014.
18. Мельников, Б. Ф., Сайфулина, Е. Ф. Применение мультиэвристического подхода для случайной генерации графа с заданным вектором степеней // Известия высших учебных заведений. Поволжский регион, №3, 2013.

19. Назаров, М. Н. Альтернативные подходы к описанию классов изоморфных графов // Прикладная дискретная математика, №3, 2014.
20. Носов, Ю. Л. Индекс Винера максимальных внешнеплоских графов // Прикладная дискретная математика, №4, 2014.
21. Савин, А. М., Тимофеева, Н. Е. Применение алгоритма оптимизации методом имитации отжига на системах параллельных и распределённых вычислений // Известия Саратовского университета. Новая серия. Серия Математика. Механика. Информатика, №1, 2012.

*Электронные ресурсы*

22. McKay's, B. graph formats // Brendan McKay's Home Page URL [Электронный ресурс]: https://users.cecs.anu.edu.au/~bdm/data/formats.html

*Литература на иностранных языках*

23. Alan Cobham, The intrinsic computational difficulty of functions. North-Holland Publishing Company, 1964.
24. Boris F. Melnikov, Elena A. Melnikova, Svetlana V. Pivneva, Nadezhda P. Churikova, Vladislav A. Dudnikov, Michael Y. Prus, Multi-heuristic and game approaches in search problems of the graph theory. Предприятие "Новая техника" (Самара), 2018.
25. Boris F. Melnikov, Vladislav A. Dudnikov, The problem of pseudo-optimal placement of a graph on a plane. Предприятие "Новая техника" (Самара), 2018.
26. John H. Holland, Adaptation in natural and artificial systems. University of Michigan Press, 1975.
27. Juraj Hromkovič, Algorithmics for Hard Problems. Springer, 2004.
28. Robert S. Garfinkel, Minimizing Wallpaper Waste, Part 1: A Class of Traveling Salesman Problems // Operations Research, №25, 1977.
29. Sean Luke, Essentials of Metaheuristics. Lulu, 2013.
30. Stephen A. Cook, The Complexity of Theorem–Proving Procedures. University of Toronto, 1971.