Санкт–Петербургский государственный университет

КУДРЯВЦЕВ Константин Александрович

Выпускная квалификационная работа

Программная реализация алгоритма решения задачи о раскраске

Уровень образования: бакалавриат
Направление 02.03.02 «Фундаментальные информатика и информационные технологии»
Основная образовательная программа СВ.5003.2016 «Программирование и информационные технологии»

Научный руководитель:
профессор, кафедра информационных систем
д.ф. - м.н. Олемской Игорь Владимирович

Рецензент:
заведующий кафедрой, кафедра высшей математики
Федеральное государственное бюджетное учреждение высшего образования «Санкт-Петербургский государственный университет телекоммуникаций им. проф. М.А.БончБруевича»
Голоскоков Дмитрий Петрович

Санкт-Петербург
2020 г.

Содержание

Введение

Впервые проблема раскраски графа была официально поднята на встрече Лондонского математического сообщества в 1878 году, где она рассматривалась в контексте задачи о раскраске карты. В 1970-х годах раскраска графа стала рассматриваться как алгоритмическая проблема.

Данная задача постоянно исследуется по двум основным причинам:

1. Она находит применение в реальных задачах, например в составлении расписаний[1], распределение регистров в микропроцессорах[2], распараллеливание численных методов[3].

2. На данный момент не найдено точного и эффективного алгоритма её решения, т.к. доказано что задача о раскраске является NP-полной.

Точные алгоритмы в основном используют полный перебор возможных вариантов раскраски. Таким образом, основным отличием таких алгоритмов является порядок перебора и используемые отсечения.

Например, в данной работе рассматривается алгоритм основанный на переборе максимальных независимых множеств, описанный в книге Новикова Ф.А.[4] и две его модификации.

Главным объектом исследования является разработанный Олемким И.В. алгоритм, и его сравнение с другими существующими алгоритмами.

Постановка задачи

Целью данной работы является реализация и исследование алгоритма Олемского И.В. Для достижения цели были выделены следующие задачи:

1. Изучение существуюших методов раскраски графа

2. Реализация алгоритма Олемского И.В.

3. Реализация существующих методов

4. Проведение сравнительного анализа реализованных алгоритмов

Глава 1. Используемые термины

- Раскраска графа — разбиение графа на множества таким образом, чтобы вершины входящие в одно множество были несмежными.
- Хроматическое число графа — наименьшее возможное число множеств в раскраске графа.
- Независимое множество графа — множество попарно несмежных вершин графа.
- Максимальное независимое множество графа — независимое множество вершин графа, которое перестанет являться независимым при добавлении любой другой вершины этого графа в множество.
- Число независимости графа — число элементов в наибольшем максимальном независимом множестве.
- Плотность графа — отношение числа рёбер графа к максимальному числу рёбер графа с таким числом рёбер, т.е. если m - число рёбер в графе G, а n — число вершин, то

$$D(G) = \frac{m}{n(n-1)}$$

Глава 2. Условные обозначения

- $\lceil x \rceil$ — означает наименьшее целое число не меньше, чем .
- $for\ i\ in [a; b)$ — в псевдокоде означает, что счётчик i принимает целые значения в интервале $[a; b)$.
- $X = []$ — в псевдокоде означает, что в переменную X помещается пустой список.
- $X.append(a)$ — в псевдокоде означает, что в список добавляется элемент .

- $A-B$,где и B множества — разность множеств, т.е. $A-B = A\backslash(A\cap B)$
- Конфигурация графа $G(n, m, s)$ — случайно сгенерированный граф G с числом вершин n, числом рёбер m, параметром генератора случайных чисел s.

Глава 3. Вспомогательные алгоритмы и структуры данных

3.1 Битовые множества

Все описанные алгоритмы были реализованы на языке программирования Python, в нём множества реализованы с помощью хэш-таблиц. Это позволяет хранить в множестве объекты любых хэшируемых типов и выполнять операции добавления и извлечения элемента со средней сложностью $O(1)$. Однако операции пересечения и объединения двух множеств S и T выполняются за $O(min(|S|, |T|))$ и $O(|S| + |T|)$ соответственно[7], а эти операции часто применяются в исследуемом алгоритме Олемского И.В. Значит для более эффективной реализации этого алгоритма необходима структура данных, позволяющая выполнять объединение и пересечение множеств за меньшее число операций.

Если заранее известны все элементы, которые могут содержаться в множестве, то можно пронумеровать их:

$$U = \{u_0, u_1, ...u_{n-1}\}$$

В таком случае любое подмножество $S \subset U$ можно представить в виде числа s, в двоичном представлении которого поднятый бит с номером i означает наличие элемента u_i в множестве S: $u_i \in S$. Например, $s = 37 = 2^0 + 2^2 + 2^5$ соответствует $S = \{u_0, u_2, u_5\}$.

Тогда объединение двух множеств S и T ($S, T \subset U$) можно представить в виде битовой операции «или» над соответствующими им числами. Аналогично пересечение множеств можно представить в виде битовой опе-

рации «и», а дополнение реализовать как «xor» с полными множеством U.

$$S \cup T \sim s|t$$
$$S \cap T \sim s\&t$$
$$\overline{T} \sim t \oplus u$$
$$S - T \sim s\&(t \oplus u)$$

На 64-битных процессорах такой подход позволяет выполнять операции над множествами за $O(1)$ при $|U| <= 64$. В качестве реализации данной идеи была использована библиотека bitarray для Python.[8]

3.2 Алгоритм Брона-Кербоша

В описанных далее алгоритмах применяется перебор по максимальным независимым множествам. Задача поиска таких множеств также является NP-полной, в данной работе для её решения применяется модификация алгоритма Брона-Кербоша[5]. Ниже приведён псевдокод этого алгоритма с подробными комментариями.

Листинг 1: Алгоритм Брона-Кербоша

```
1  Function check(G, candidates, used):
2  Input:
3  G - граф в виде списков смежности
4  candidates - нераспределенные   вершины
5  used - рассмотренные ранее вершины
6  Returns:
7  False, если дальнейшее расширение независимого множества
     невозможно, True в противном случае
8  Begin
9      for u in used:
10         if not (G[u] & candidates): #Если вершина из used не
     соединена ни с одной из оставшихся вершин
11             return False
12     return True
13 End
14
15 Procedure extend(indSet, candidates, used, result)
16 Input:
17 indSet - текущее независимое множество
18 candidates - вершины, которыми можно расширить indSet
```

```
used - вершины, которые уже использовались для расширения indSet
result - список максимальных независимых множеств, который
   расширяется по мере выполнения процедуры, является
   результатов работы этой процедуры
Begin
    while candidates and check(candidates, used):
        c = candidates.pop() # выбираем произвольный текущий элемент
        indSet = indSet ∪ c # добавляем текущий элемент в indSet
        newCandidates = candidates - {c} - G[c] #indSet не может
   быть расширено вершинами, смежными с добавленным элементом
        newWrong = wrong - {c} - G[c] #так как indSet не может быть
   расширен этими вершинами, то помечать их рассмотренными тоже не нужно
        if newCandidates == ∅ and newWrong == ∅:  #Если в
   множество нельзя добавить элементов
            result.append(indSet) # То построено максимальное
   множество
            extend(indSet, newCandidates, newWrong, result) #
   Иначе рекурсивно строим продолжение

        candidates = candidates - {c} # Исключаем рассмотренную
   вершину
        indSet = indSet - {c}
        wrong = wrong ∪ {c} # Помечаем вершину как рассмотренную
End

Function Bron(G, nodes)
Input:
G - граф в виде списков смежности
nodes - множество всех вершин графа
Returns:
result - список всех максимальных независимых множеств
Begin
    result = [] # Пустой список независимых множеств
    extend(∅, nodes, ∅, result) #Первый вызов рекурсивной процедуры
    return result
End
```

Глава 4. Алгоритм Олемского

Алгоритм разработан Олемским И.В. как модификация алгоритма выделения структурных особенностей системы обыкновенных дифференциальных уравнений, описанного в книге[9]. Задача раскраски графа рассматривается с точки зрения выделения минимального количества нульдиагональных блоков в матрице смежности этого графа.

4.1 Описание множеств и переменных

Формальное определение вводимых множеств и переменных будет приведено в описании алгоритма, ниже же помимо формального описания присутствует семантическое — назначение этих переменных и множеств в алгоритме.

- $I = \{1, 2, ..n\}$ — множество чисел, соответствующих вершинам графа.
- A — матрица смежности графа (размера $[n * n]$).
- $h_q(A) = \{r | a_{q,r} = 0, r \in I\}, q \in I$ — горизонтальные структурные множества.
- $v_r(A) = \{q | a_{q,r} = 0, q \in I\}, r \in I$ — вертикальные структурные множества.
- $d_{q,r}(A) = \{x | x \in h_q, x \in v_r\}, q \in I, r \in I$ — вспомогательные множества.
- $P = (q, r) | \{q, r\} \in I,\ q < r,\ A_{q,r} = 0$ — множество упорядоченных пар несмежных вершин графа.
- $D_{(q,r)} = d_{q,r} \cap d_{r,q}$, где $(q, r) \in P$ — множества, характеризующие структурные особенности матрицы A. В контексте задачи о раскраске множество $D_{(q,r)}$ содержит вершины, которые не смежны ни с вершиной q, ни с вершиной r.
- j — номер нульдиагонального блока, который строится (ветка перебора).

- s — шаг построения нульдиагонального блока (уровень перебора).
- $J^j = \{i_1^j, i_2^j, ..i_{n(j)}^j\} \subset I$ — множество элементов, входящих в нульдиагональный блок номер j. После окончания построения всех нульдиагональных блоков (пусть их построено k), они образуют раскраску $J = (J^1, J^2..J^k)$, для которой должны выполняться свойства:
  - $J^a \neq \varnothing$, где $a \in \{1, 2, ..k\}$ — отсутствуют пустые нульдиагональные блоки
  - $J^a \cap J^b = \varnothing$, где $a \neq b,\ a, b \in \{1, 2, ..k\}$ — блоки не пересекаются
  - $\cup_{j=1}^{k} J^j = I$ — блоки включают в себя все вершины графа
  - $|J^a| \geq |J^b|$, где $a > b,\ a, b \in \{1, 2, ..k\}$ — блоки упорядочены по числу элементов
- J_{best} — текущая лучшая (наименьшая по числу нульдиагональных блоков) раскраска.
- $v_{best} = |J_{best}|$
- $\omega^{j,1} = I - (\cup_{i=1}^{j-1} J^i)$ — опорное множество, строится на первом шаге($s = 1$) построения очередного нульдиагонального блока. Содержит вершины, которые не использовались при построении предыдущих $j-1$ нульдиагональных блоков. $\omega^{j,s}$ при $s > 1$ содержит вершины, которыми можно продолжить построение нульдиагонального блока J^j.
- $D_{(q,r)}^{j,s} = D_{(q,r)} \cap \omega^{j,s}$ — характеристические особенности для матрицы графа, в котором остались только вершины из множества $\omega^{j,s}$.
- $G^{j,s} = \{(q, r) | \{q, r\} \in D_{(q,r)}^{j,s}\}$ — множество пар вершин, которые можно использовать для построения текущего нульдиагонального блока.
- $Q^{j,s}$ — множество элементов $G^{j,s}$, которые уже использовались при построении нульдиагонального блока j на шаге s.
- $\alpha^{j,s} \in (G^{j,s} - Q^{j,s})$ — узловой элемент, т.е. пара вершин, для которой выполняется $|D_{\alpha^{j,s}}^{j,s}| = \max\limits_{\alpha} |D_{\alpha}^{j,s}|$, где $\alpha \in (G^{j,s} - Q^{j,s})$. Именно по таким парам строятся нульдиагональные блоки.

- F^j — множество, в котором запоминаются одиночные вершины, которые уже использовались для построения нульдиагонального блока нечетной длины номер j .

- $\psi^{j,s} = J^j - (\cup_{i=1}^{s-1}\alpha^{j,i})$ — множество, в котором содержатся те узловые элементы, которые были использованы при построении J^j, начиная с шага s включительно.

- $Z^{j,s} = \{x | x \in (G^{j,s} - Q^{j,s}) \cap \psi^{j,s} \times \psi^{j,s},\ D_x^{j,s} \subset \psi^{j,s}\}$ — множество узловых элементов, использование которых для построения J^j на шаге s позволит в лучшем случае построить $J_*^j \subset J^j$.

- B — множество, в котором хранятся все нульдиагональные блоки, полученные на первом уровне (J^1).

- $\rho^{j,s} = \begin{cases} |D^{j,s}_{alpha[j,s]}|, \text{ если } G^{j,s} - Q^{j,s} \neq \varnothing \\ 1, \text{ иначе} \end{cases}$ — величина, используемая при проверках, отражает оптимистичную оценку максимального числа вершин, которые могут быть добавлены к текущему нульдиагональному блоку.

4.2 Описание проверок

В алгоритме Олемского И.В. с целью уменьшения числа рассматриваемых вариантов перед каждой попыткой расширить нульдиагональный блок J^j выполняются три проверки, позволяющие без большого объёма требуемых вычислений определить неперспективность дальнейшего рассмотрения такого расширения.

- $j - 1 + \left\lceil \frac{\omega^{j,1}}{\rho^{j,1}} \right\rceil \geq v_{best}$ — проверка типа A, выполняется на первом шаге построения очередного нульдиагонального блока $(s = 1)$. Выполнение этого неравенства означает, что к построенным $j - 1$ нульдиагональным блокам добавится как минимум $\left\lceil \frac{\omega^{j,1}}{\rho^{j,1}} \right\rceil$ блоков, что превысит число блоков в текущей лучшей раскраске J_{best}.

- $2(s-1) + ro \leq \frac{|I|}{v_{best}}$ — проверка типа B, выполняется на каждом шаге построения первого нульдиагонального блока $(j = 1)$. Выполнение

этого неравенства означает, что максимальное число элементов в нульдиагональном блоке J^1(наибольшем из всех блоков по определению) $2(s-1)+ro$ не больше среднего числа элементов в блоках лучшей раскраски $\frac{|I|}{v_{best}}$, что не позволит построить раскраску меньше, чем с v_{best} блоками.

- $2(s-1)+\rho^{j,s} \neq |\omega^{j,1}|$ — проверка типа C, выполняется при $j = v_{best}-1$. Выполнение этого неравенства означает, что число элементов в текущем нульдиагональном блоке не достигнет числа элементов в опорном множестве $\omega^{j,1}$, то есть возникнет необходимость строить блок номер $j = v_{best}$.

4.3 Описание алгоритма

0. По матрице смежности вычисляются последовательно

   i $h_q(A) = \{r|a_{q,r} = 0, r \in I\}, q \in J$ — горизонтальные структурные множества

   ii $v_r(A) = \{q|a_{q,r} = 0, q \in I\}, r \in J$ — вертикальные структурные множества

   iii $d_{q,r}(A) = \{x|x \in h_q, x \in v_r\}, q \in J, r \in J$ — вспомогательные множества

   iv $P = (q,r)|\{q,r\} \in I,\ q < r,\ A_{q,r} = 0$ — множество упорядоченных пар несмежных вершин графа

   v $D_{(q,r)} = d_{q,r} \cap d_{r,q}$, где $(q,r) \in P$ — множества, характеризующие структурные особенности матрицы A.

   Задаются начальные значения

   - $G^{1,1} = \varnothing$
   - $Q^{1,1} = \varnothing$
   - $F^{1,1} = \varnothing$
   - $B = []$

- $s = 1$
- $j = 1$
- $J_{best} = \{\{i\}, i \in I\}$
- $v_{best} = |J_{best}|$

1. В первом пункте алгоритма выполняется построение опорного множества

$$\omega^{j,s} = I - (\cup_{i=1}^{j-1} J^i)$$

Понятно, что на первом шаге построения первого блока $\omega^{1,1} = I$

2. Если построенное опорное множество оказалось пустым ($\omega^{j,s} = \varnothing$), то текущий блок невозможно расширить.

Если при этом построение находится на первом шаге $(s = 1)$, это означает, что раскраска построена и нужно сравнить её с лучшей раскраской, т.е. перейти на пункт 8.

Иначе $(s > 1)$ необходимо добавить текущий блок в строящуюся раскраску, т.е. перейти на пункт 5.

3. Опорное множество не пусто ($\omega^{j,s} \neq \varnothing$), значит можно сформировать множество пар возможных продолжений

$$G^{j,s} = \{(q, r) | (q, r) \in P, \{q, r\} \in D^{j,s}_{(q,r)}\}$$

Множество $Q^{j,s}$ содержит множество пар из $G^{j,s}$, которые уже были использованы при построении блока j на шаге s.

4. При рассмотрении множества возможных продолжений $G^{j,s} - Q^{j,s}$ возможны два варианта:

- $(G^{j,s} - Q^{j,s}) \neq \varnothing$ — множество возможных продолжений не пусто. В этом случае из этого множества выбирается узловой элемент $\alpha^{j,s}$

$$|D^{j,s}_{\alpha^{j,s}}| = \max_{\alpha} |D^{j,s}_{\alpha}| \quad (1)$$

После выбора узлового элемента выполняются проверки A,B,C.

– В случае выполнения неравенства A или C производится возврат к рассмотрению блока $j = j - 1$ на последнем шаге его построения $s = \left\lceil \frac{|J^j|}{2} \right\rceil$ в пункте 4.

– В случае выполнения неравенства B производится возврат к предыдущему шагу $s = s - 1$ построения текущего блока в пункте 4.

Если проверки пройдены, выполняется построение нового опорного множества

$$\omega^{j,s+1} = (\omega^{j,s} \cap D^{j,s}_{\alpha^{j,s}})$$

Узловой элемент запоминается в множестве $Q^{j,s}$ как рассмотренный $Q^{j,s} = Q^{j,s} \cup \alpha^{j,s}$.

Множества использованных узловых и конечных элементов для следующего шага обнуляются $Q^{j,s+1} = \varnothing,\ F^{j,s+1} = \varnothing$. После чего шаг увеличивается $(s = s + 1)$ и производится переход к пункту 2.

- $(G^{j,s} - Q^{j,s}) == \varnothing$ — множество возможных продолжений пусто, в этом случае из множества $\omega^{j,s} - F^{j,s}$ выбирается концевой элемент β и после прохождения проверок A,B,C записывается как использованный $F^{j,s} = F^{j,s} \cup \beta$. После чего выполняется переход к пункту 5.

  Если множество $\omega^{j,s} - F^{j,s}$ оказалось пустым, то производится возврат к рассмотрению блока $j = j - 1$ на последнем шаге его построения $s = \left\lceil \frac{|J^j|}{2} \right\rceil$ в пункте 4.

5. Построение нульдиагонального блока J^j было завершено одним из двух способов:

   - Добавлением пары $\alpha^{j,s-1}$ в этом случае число элементов в блоке равно $2(s-1)$ и блок $J^j = \{\alpha_1^{j,1}, \alpha_2^{j,1}, .., \alpha_1^{j,s-1}, \alpha_2^{j,s-1}\}$.
   - Добавлением конечного элемента β в этом случае число элементов в блоке равно $2s-1$ и блок $J^j = \{\alpha_1^{j,1}, \alpha_2^{j,1}, .., \alpha_1^{j,s-1}, \alpha_2^{j,s-1}, \beta\}$.

Определить, что построение блока было завершено добавлением пары $\alpha^{j,s-1}$ можно по пустому текущему опорному множеству $\omega^{j,s} = \varnothing$

6. В шестом пункте алгоритма выполняется прореживание, которое можно разбить на два этапа

   - Исключение из рассмотрения тех пар в множествах $G^{j,ss},\ ss = \overline{1, s-1}$, рассмотрение которых не приведёт к построению блока отличного от текущего J^j. Для этого выполняются следующие действия

     – Построение множества, в котором содержатся те узловые элементы, которые были использованы при построении J^j, начиная с шага ss включительно$\psi^{j,ss} = J^j - (\cup_{i=1}^{ss-1}\alpha^{j,i})$ -

     – Построение множества узловых элементов, использование которых для построения J^j на шаге ss позволит в лучшем случае построить $J^j_* \subset J^j\ Z^{j,ss} = \{x | x \in (G^{j,ss} - Q^{j,ss}) \cap \psi^{j,ss} \times \psi^{j,ss},\ D^{j,ss}_x \subset \psi^{j,ss}\}$

     – Производится само прореживание $Q^{j,ss} = Q^{j,ss} \cap Z^{j,ss}$

   - В множестве B хранятся рассмотренные ранее блоки J^1. Соответственно, если сейчас рассматривается блок $j = 1$, то необходимо проверить его наличие в множестве B.

     – Если $J^1 \in B$, это означает, что ранее уже были рассмотрены все раскраски, у которых первый блок был J^1, то есть необходимо перестроить полученный блок, для этого выполняем переход на пункт 3, спустившись на последний шаг построения $s = \left\lceil \frac{|J^1|}{2} \right\rceil$.

     – Если $J^1 \notin B$, то необходимо запомнить, что такой блок был рассмотрен $B = B \cup J^1$.

7. Построение текущего блока J^j закончено, он добавлен в раскраску J. Значит необходимо переходить к построению следующего блока, для этого увеличивается номер блока $j = j + 1$, устанавливается номер

шага $s = 1$. После чего очищаются множества рассмотренных узловых и концевых элементов для нового блока $Q^{j,s} = \varnothing$, $F^{j,s} = \varnothing$ и совершается переход к построению нового опорного множества — на пункт 1.

8. Новая раскраска построена ($\omega^{j,s} = \varnothing$, $\cup_{i=1}^{j-1} J^i = I$), необходимо сравнить число блоков в новой раскраске $v = j - 1$ с числом блоков в текущей лучшей раскраской v_{best}.

   Если $v < v_{best}$, значит найдена новая лучшая раскраска, необходимо её запомнить $J_{best} = J$, $v_{best} = v$.

   Выполнение равенства

   $$\left\lceil \frac{|\omega^{1,1}|}{\rho^{1,1}} \right\rceil = v_{best}$$

   означает, что найденная лучшая раскраска J_{best} не может быть улучшена и является ответом на задачу.

   Если равенство не было выполнено, то необходимо продолжать перебор вариантов, для этого переходим к перестройке предпоследнего блока $j = j - 2$ на последний шаг его построения $s = \left\lceil \frac{|J^j|}{2} \right\rceil$ в пункте 4.

4.4 Псевдокод алгоритма

Листинг 2: Алгоритм Олемского И.В.

```
Input:
A - матрица смежности для графа
I - множество вершин графа
Returns:
J_best - лучшая раскраска
Begin
    for q in I: # построение горизонтальных структурных множеств
        h_q = ∅
        for r in I:
            if A_{q,r} == 1:
                h_q = h_q ∪ {r}
    for q in I: # построение вертикальных структурных множеств
        v_q = ∅
        for r in I:
```

```
15              if A_{r,q} == 1:
16                  v_q = v_q ∪ {r}
17
18      for q in I: # построение вспомогательных множеств d
19          for r in I:
20              d_{q,r} = h_q ∩ v_r
21
22      P = ∅ # множество пар несмежных вершин
23      for q in I: # построение D
24          for r in (I - {1, 2, .. q}):
25              if ({r, q} ⊂ d_{q,r})and({r, q} ⊂ d_{r,q}):
26                  D_{(q,r)} = d_{q,r} ∩ d_{r,q}
27                  P = P ∪ (q, r)
28
29      for q in I: # построение начальной лучшей раскраски
30          J_best.append({q})
31
32      # начальные значения множеств G,Q,F, списка B, номера блока и номера
    шага
33      G^{1,1} = ∅
34      Q^{1,1} = ∅
35      F^{1,1} = ∅
36      B = []
37      s = 1
38      j = 1
39
40      Label p1: # построение текущего опорного множества
41          ω^{j,s} = I
42          for i in [1;j):
43              ω^{j,s} = ω^{j,s} − J^i
44
45      Label p2: # проверка блока на завершенность
46          if ω^{j,s} == ∅: # если текущий блок нельзя дополнить
47              if s == 1: # Если это замечено в начале построения нового
    блока
48                  Goto p8 # переход к сравнению J с лучшим результатом
49              else:
50                  Goto p5 # переход к записи блока J^j в раскраску J
51
52      Label p3: # формирование множества возможных продолжений
53          for key in P: # итерация по всем парам несмежных вершин
54              if {key_1, key_2} ⊂ ω^{j,s} # если обе вершины пары присутствуют в
    опорном множестве
55                  G^{j,s} = G^{j,s} ∪ key
56
57      Label p4: # добавления вершин в текущий блок
```

```
58          tmp = G^{j,s} − Q^{j,s}  # множество пар, которыми можно дополнить J^j
59          if tmp ≠ ∅:  # если не все пары были проверены
60              α^{j,s} = max(tmp, key = lambda x : |D[x] ∩ ω^{j,s}|)  # поиск пары x, которой в D соответствует наибольшее множество D_x^{j,s}
61              ρ = |D[α^{j,s}] ∩ ω^{j,s}|
62
63              if s == 1:
64                  if j − 1 + ⌈ω^{j,1}/ρ^{j,1}⌉ ≥ v_best:  # проверка типа A
65                      j = j - 1
66                      s = ⌈|J^j|/2⌉  # переход на на последний шаг построения предыдущего блока
67                      Goto p4
68
69              if j == 1:
70                  if 2(s − 1) + ro ≤ |I|/v_best:  # проверка типа B
71                      s = s - 1  # переход на предыдущий шаг построения блока J^1
72                      Goto p4
73
74              if  j == v_best − 1:
75                  if 2(s − 1) + ρ^{j,s} ≠ |ω^{j,1}|:  # Проверка типа C
76                      j = j - 1
77                      s = ⌈|J^j|/2⌉  # переход на на последний шаг построения предыдущего блока
78                      Goto p4
79
80              # проверки пройдены, расширение блока имеет смысл
81              Q^{j,s} = Q^{j,s} ∪ α^{j,s}  # добавление α^{j,s} в множество рассмотренных пар
82              s = s + 1
83              # формирование нового опорного множества
84              ω^{j,s} = (ω^{j,s−1} ∩ D[α^{j,s}]) − {x|x ∈ α^{j,s}}
85              Q^{j,s} = ∅
86              F^{j,s} = ∅
87              Goto p2 # переход к проверке блока на завершённость
88          else:  # G^{j,s} − Q^{j,s} == ∅
89              # выбор произвольного конечного элемента, который не рассматривался ранее
90              β = x|x ∈ (ω^{j,s} − F^{j,s})
91              if β ≠ ∅:
92                  # т.к. G^{j,s} − Q^{j,s} = ∅, то
93                  ρ = 1
94                  # выполнение проверок A,B,C
95                  if s == 1:
96                      if j − 1 + ⌈ω^{j,1}/ρ^{j,1}⌉ ≥ v_best:
```

```
                        j = j - 1
                        s = ⌈|J^j|/2⌉
                        Goto p4
                if j == 1:
                    if 2(s − 1) + ro ≤ |I|/v_best:
                        s = s - 1
                        Goto p4
                if  j == v_best − 1:
                    if 2(s − 1) + ρ^{j,s} ≠ |ω^{j,1}|:
                        j = j - 1
                        s = ⌈|J^j|/2⌉
                        Goto p4
                # в случае прохождения всех проверок выполняется переход
    на p5
            else: # β == ∅
                s = s-1 # переход на предыдущий шаг построения блока J^j
                Goto p4

     Label p5 # запись блока в раскраску
         J[j] = ∅
         # восстановление J^j по узловым элементам
         for i in [1;s):
             J[j] = J[j] ∪α^{j,i}
         if ω^{j,s} ≠ ∅:
             # в блок J^j на шаге s была добавлена вершина β
             J[j] = J[j] ∪β

     Label p6 # прореживание
         # получение числа узловых элеметов, учавтвовавших в построении J^j
         k = ⌊|J^j|⌋
         for ss in [1;k):
             # формирование ψ^{j,ss}
             ψ^{j,ss} = J^j
             for i in [1;ss):
                 ψ^{j,s} = ψ^{j,s} − α^{j,i}

             # формирование множества узловых элементов, которые не
    приведут к блоку, отличному от сформированного
             Z = ∅
             for a in (G^{j,ss} − Q^{j,ss}):
                 if (a∈ ψ^{j,ss}) and D_a^{j,ss} ⊂ ψ^{j,ss}
                     Z = Z ∪a
             Q^{j,ss} = Q^{j,ss} ∪ Z
```

```
        # проверка множества B
        if j == 1:
            for el in B:
                if el == ψ^{1,1}: # если такой первый блок уже рассматривался
                    # переход к перестройке блока J^1
                    s = ⌈|J^j|/2⌉
                    Goto p3
        B.append(ψ^{1,1})

    Label p7 # переход к построению нового блока
        j = j + 1 # увеличение номера блока
        s = 1 # строится новый блок, поэтому шаг первый
        # необходимо обнулить рассмотренные узловые и концевые элементы
        G^{j,1} = ∅
        F^{j,1} = ∅
        Goto p1 # переход к построению нового опорного множества

    Label p8
        if ω^{j,1} == ∅ and ∪_{i=1}^{j-1} J^i == I: # Если опорное множество не содержит вершин и полученная раскраска покрывает все вершины
            if j − 1 < v_best: # если число блоков в раскраске меньше, чем в лучшей раскраске
                J_best = J

            # вычисление ρ
            if G^{1,1} − Q^{1,1} == ∅:
                ρ = 1
            else
                ρ = D^{1,1}_{α1,1} ∩ ω^{1,1}

            if ⌈|ω1,1|/ρ⌉ == v_best: # Если построена лучшая возможная раскраска, возвращается результат
                return J0

            #переход к перестройке предпоследнего блока
            j = j - 2
            s = ⌈|J^j|/2⌉
            Goto p4
End
```

Глава 5. Алгоритмы перебора независимых множеств

В книге Новикова Ф.А.[4] описан алгоритм поиска раскраски графа:

1. Выбрать в графе G максимальное независимое множество S
2. Покрасить вершины множества S в очередной цвет
3. Перейти к пункту 1 с новым графом $G - S$

В виде псевдокода данный алгоритм может быть записан так:

Листинг 3: Алгоритм построения раскраски Новикова Ф.А.

```
1  Function Colorize(G, nodes, C):
2  Input:
3  G - граф, заданный списками смежности
4  nodes - нераскрашенные вершины графа
5  C - текущая раскраска
6  Returns:
7  C - правильная раскраска графа G, список множеств
8  Begin
9      if nodes == ∅: # если все вершины были покрашены
10         return C
11     S = SelectMax(G, nodes) # выбор множества независимых вершин в G
12     C.append(S) # добавление множества в раскраску
13     Colorize(G, nodes - S, C) # рекурсивный вызов для графа без вершин
       из S
14 End
```

Далее в книге доказывается теорема о том, что существует такая последовательность выбора множеств S в пункте 1 алгоритма (строка 11 псевдокода), что полученная раскраска будет минимальной. Для нахождения такой последовательности можно применить различные подходы.

5.1 Перебор всех возможных вариантов

Самый простой способ построить наименьшую раскраску — построить все раскраски и выбрать из них наименьшую. Для реализации этой идеи нужно немного изменить алгоритм раскраски Новикова Ф.А.

1. Если построена раскраска и она оказалась лучше предыдущей лучшей, то запоминаем её

2. Если раскраска еще не построена, то выбираем очередное максимальное независимое множество S в графе G

3. Красим вершины множества S в цвет, равный глубине рекурсии

4. Переходим к пункту 1 с новым графом $G - S$

Псевдокод алгоритма:

Листинг 4: алгоритм перебора независимых множеств

```
Function Colorize(G, nodes, C, C_best):
Input:
G - граф, заданный списками смежности
nodes - нераскрашенные вершины графа
C - текущая раскраска
C_best - лучшая найденная раскраска
Returns:
C_best - лучшая раскраска графа G, список множеств
Begin
    if nodes == ∅: # если все вершины были покрашены
        if |C| < |C_best|: # если раскраска улучшена
            C_best = C
        return C_best
    indSets = Bron(G, nodes) # список всех максимальных независимых
    множеств, подробно функция Bron(G, nodes) описана в пункте 3.2
    for S in indSet: # перебор по всем максимальным независимым
    множествам
        C.append(S) # добавление множества в раскраску
        C_best = Colorize(G, nodes - S, C, C_best) # рекурсивный вызов
    для графа без вершин из S
    return C_best
End
```

5.2 Оптимизированный перебор

Проблема подхода, приведенного в предыдущем пункте, в том, что каждый раз при рекурсивном вызове функции Colorize она вызывает функцию Bron, которая решает NP-полную задачу построения максимальных независимых множеств. Понятно, что время работы такого алгоритма слишком велико. Для решения этой проблемы необходимо доказать следующее:

Утверждение: Если A - подграф графа B, полученный удалением множества вершин X, то если S_A - максимальное независимое множество графа A, то в графе B существует максимальное независимое множество S_B, такое, что $S_A = S_B - X$.

Доказательство: S_A является независимым множеством в графе B, значит в графе B существует максимальное независимое множество S_B, такое, что $S_A \subset S_B$, причём $S_A \cap X = \varnothing$, значит $S_A \subset (S_B - X)$.

Покажем, что $(S_B - S_A) \subset X$. Пусть это не так, т.е. существует вершина $v \notin X$, $v \in (S_B - S_A)$. Тогда эта вершина присутствует в графе A, т.к. $v \notin X$. $v \in S_B$, значит она не смежна ни с одной вершиной из S_B, а значит не смежна ни с одной вершиной из $S_A \subset S_B$, но $v \notin S_A$, значит S_A - не является максимальным независимым множеством, противоречие. Значит $(S_B - S_A) \subset X$, т.е. $(S_B - X) \subset S_A$.

$S_A \subset (S_B - X)$ и $(S_B - X) \subset S_A$, значит $S_A = S_B - X$, что и требовалось доказать.

Доказанное утверждение позволяет построить максимальные независимые множества только для начального графа и в дальнейшем использовать их для выбора множества S в подграфах. Теперь можно алгоритм перебора максимальных независимых множеств можно записать так:

0. Построить максимальные независимые множества $indSets$ для начального графа G

1. Если построена раскраска и она оказалась лучше предыдущей лучшей, то запоминаем её

2. Если раскраска еще не построена, то выбираем очередное максимальное независимое множество S из $indSets$

3. Если множество $S \cap G$ было рассмотрено переходим к пункту 2

4. Красим вершины множества $S \cap G$ в цвет, равный глубине рекурсии

5. Переходим к пункту 1 с новым графом $G - S$

Псевдокод алгоритма:

Листинг 5: оптимизированный алгоритм перебора независимых множеств

```
Function Colorize(G, nodes, indSets, C, C_best):
Input:
G - граф, заданный списками смежности
nodes - нераскрашенные вершины графа
indSets - список всех максимальных независимых множеств для
   начального графа G
C - текущая раскраска
C_best - лучшая найденная раскраска
Returns:
C_best - лучшая раскраска графа G, список множеств
Begin
    if nodes == ∅: # если все вершины были покрашены
        if |C| < |C_best|: # если раскраска улучшена
            C_best = C
        return C_best
    used = {∅}
    for S in indSets: # перебор по всем максимальным независимым
   множествам
        if (S ∩ nodes) in used: # если такое множество было проверно
            continue # переход на следующую итерацию цикла
        used = used ∪ (S ∩ nodes) # сохранение множества как
   рассмотренного
        C.append(S) # добавление множества в раскраску
        C_best = Colorize(G, nodes - S, indSets, C, C_best) #
   рекурсивный вызов для графа без вершин из S
    return C_best
End
```

5.3 Перебор в матричном виде

На сегодняшний день существует множество библиотек для различных языков программирования, позволяющих эффективно выполнять матричные операции, например пакет numpy[10] для языка Python. Поэтому имеет смысл представить алгоритм перебора независимых множеств в матричном виде.

Пусть для графа G с вершинами $V = \{v_1, v_2, .., v_n\}$ найдены все максимальные независимые множества $I = \{I_1, I_2, .., I_k\}$. Тогда для такого графа можно построить матрицу принадлежности $M = \{m_{q,r}\}$ размерности $[n \times k]$,

в которой

$$m_{q,r} = \begin{cases} 1, \text{ если } v_q \in I_r \\ 0, \text{ иначе} \end{cases}$$

Тогда чтобы узнать какие вершины содержатся в объединении множеств $\cup_{l=1}^{r} I_{j_l}$ необходимо матрицу M умножить на вектор-столбец "выбора"$x = \{x_r\}$ размерности $[k \times 1]$, в котором

$$x_r = \begin{cases} 1, \text{ если } r \in (\cup_{l=1}^{r} j_l) \\ 0, \text{ иначе} \end{cases}$$

В Полученной вектор-строке "включения"$y = \{y_q\}$ размерности $[1 \times n]$

$$y_q = \begin{cases} 1, \text{ если } v_q \in (\cup_{l=1}^{r} I_{j_l}) \\ 0, \text{ иначе} \end{cases}$$

Используя такой способ представления независимых множеств, задачу о раскраске можно переформулировать следующим образом: необходимо найти такой вектор-столбец "выбора"x с минимальным числом единиц, при умножении на который матрицы M получим вектор-строку y, состоящую только из единиц. Т.е.

$$Mx = (1, 1, .., 1)$$

Пусть такой вектор x с r единицами найден, причем $x_{l_j} = 1,\ j = \overline{1, r}$, тогда разбиение множества вершин графа на цвета будет следующим $C_j = I_{l_j} - (\cup_{z=1}^{j-1} I_{l_z}), j = \overline{1, r}$.

Запишем итоговый алгоритм:

0. Построить максимальные независимые множества для начального графа G, построить по ним матрицу M, получить нижнюю оценку хроматического числа k

1. Сгенерировать все возможные векторы x с k единицами и $n-k$ нулями

2. Выбрать очередной вектор с k единицами и $n - k$ нулями

3. Если все векторы были использованы, увеличиваем $k = k + 1$ и переходим на пункт 1

4. Если $Mx == (1, 1, .., 1)$, то x - искомый вектор

5. Иначе переходим к пункту 2

Для нижней оценки используется оценка с помощью числа независимости графа $\lceil \frac{n}{\alpha} \rceil$.

Псевдокод алгоритма:

Листинг 6: матричный алгоритм перебора независимых множеств

```
Function Colorize(G, nodes):
Input:
G - граф, заданный списками смежности
nodes - нераскрашенные вершины графа
Returns:
x - векторстобел- выбора
Begin
    indSets = Bron(G, nodes) # получение максимальных независимых множесв
    M = 𝕆
    for q in [1; |nodes|]:
        for r in [1; |indSets|]:
            if nodes[q] in indSets[r]:
                M[q,r] = 1 # заполнение матрицы M
    α = max{|S| : S ∈ indSets} # число независимости
    k = ⌈n/α⌉ # нижня оценка

    While True:
        variants = GeneratePermutations(k, |nodes| - k) # сгенерировать векторы с k единицами и |nodes| - k нулями
        for x in variants:
            if sum(M*x) == |nodes|: # если раскраска покрывает все вершины
                return x
        k = k + 1 # предполагаем, что граф k + 1 раскрашиваемый
End
```

Глава 6. Измерение быстродействия алгоритмов

Время работы описанных алгоритмов сложно оценить аналитически, так как оно зависит не только от числа вершин и числа рёбер, но и от формы графа, образующегося из этих вершин и рёбер. Поэтому быстродействие алгоритмов измерялось эмпирическим способом.

Программы, реализующие алгоритмы, запускались в операционной системе Windows 10 на компьютере с процессором Ryzen 5 2600. Частота процессора была зафиксирована на отметке 3.6 GHz, чтобы исключить влияние на результаты технологии Precision Boost[11], которая может кратковременно повышать производительность процессора. Время работы алгоритмов может отличаться при проведении тестирования на системах с другой операционной системой или процессором, однако время работы алгоритмов относительно друг друга будет схожим.

Для измерения времени работы алгоритма на конкретной конфигурации графа(2) использовался модуль timeit[12] для Python, который позволяет избежать большинства неточностей, возникающих при эмпирическом измерении быстродействия алгоритмов.

Время работы алгоритма на графе с n вершинами и плотностью ρ считалось как среднее время работы алгоритма на конфигурациях $G(n, m, s)$, для всех m, для которых верно $\left\lceil \frac{m}{n(n-1)/2} \right\rceil = \rho$ и для $s = \overline{1, 5}$.

Для визуализации результатов использовалась библиотека matplotlib для Python[13].

6.1 Алгоритм Олемского И.В.

На приведенных графиках (рис.1) приведена зависимость времени работы реализации алгоритма Олемского И.В. от числа вершин в графе и плотности этого графа.

На первом графике наблюдается явное ухудшение производительности при плотности графа 0.8 и 0.3, в то время как на втором графике это ухудшение проявляется при плотности 0.8 и 0.5, а на третьем при плотности 0.8 и 0.4.

Замедление работы алгоритма объясняется особенностями отсечения вариантов в проверках A,B,C(пункт 4.2) и выбора очередного узлового элемента в пункте 4 алгоритма(равенство 1). Оба этих действия работают неэффективно, когда мощности большинства множеств $|D_{(q,r)}|$ совпадают с максимальной мощностью $\max\limits_{\alpha \in P} |D_\alpha|$. Это возможно при высокой плотности графа (например 0.8), когда в нём образуется большая клика и для большинства пар несмежных вершин находится только одна вершина, несмежная ни с одной из пары. Также большинство множеств $D_{(q,r)}$ имеют одинаковую мощность, когда образуется k-связный граф низкой плотности (например 0.4), в этом случае для большинства пар несмежных вершин находится $(n-k-2)$ вершин, несмежных ни с одной из пары (n - число вершин в графе).

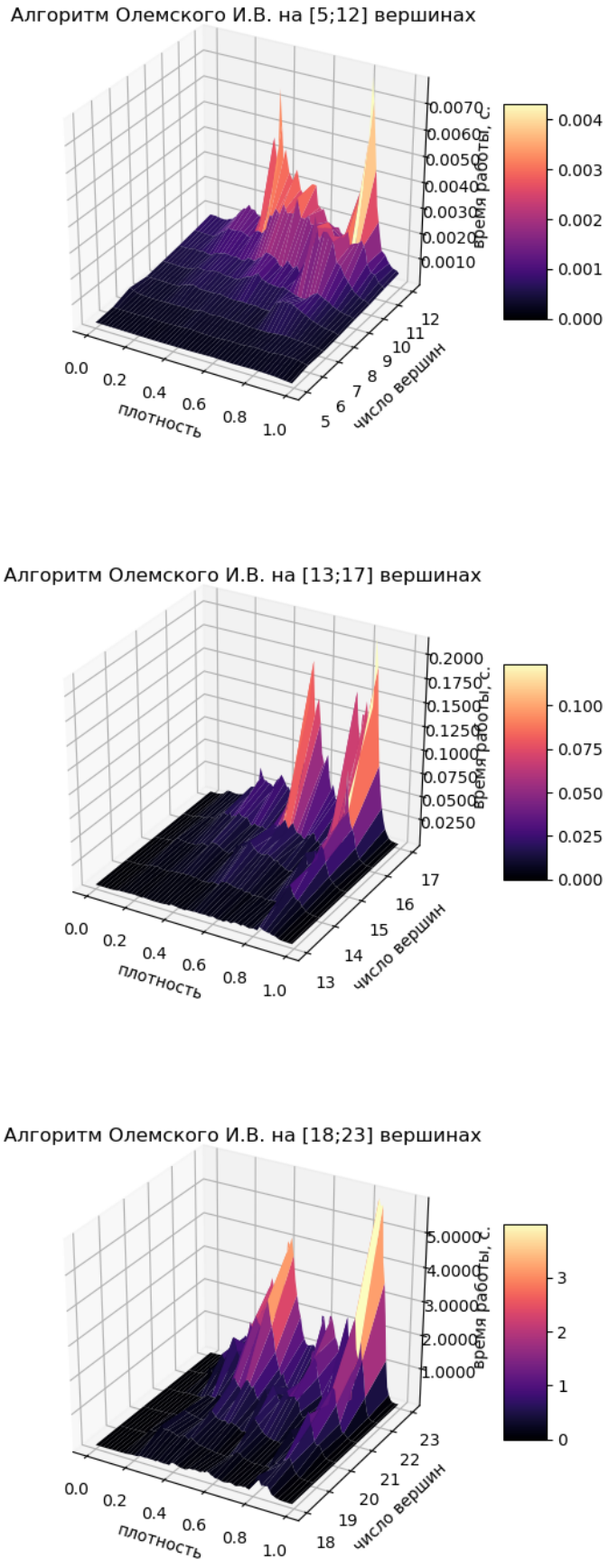


Рис. 1: Время работы алгоритма Олемского И.В. на [5;23] вершинах

6.2 Алгоритм перебора независимых множеств

На графиках(рис. 2) приведена зависимость времени работы реализации алгоритма перебора независимых множеств от числа вершин в графе и плотности этого графа.

На обоих графиках видно сильное увеличение времени работы алгоритма при увеличении плотности графа. На графиках(рис. 3) рассмотрено время работы алгоритма при плотности меньше 0.8, на них постепенное замедление алгоритма на графах высокой плотности представлено более явно.

Увеличение времени работы объясняется тем, что на графах высокой плотности максимальные независимые множества состоят из малого числа элементов и почти не пересекаются. Поэтому на каждом шаге рекурсии алгоритм находит большое количество множеств для перебора и общая глубина рекурсии (число цветов в раскраске) повышается.

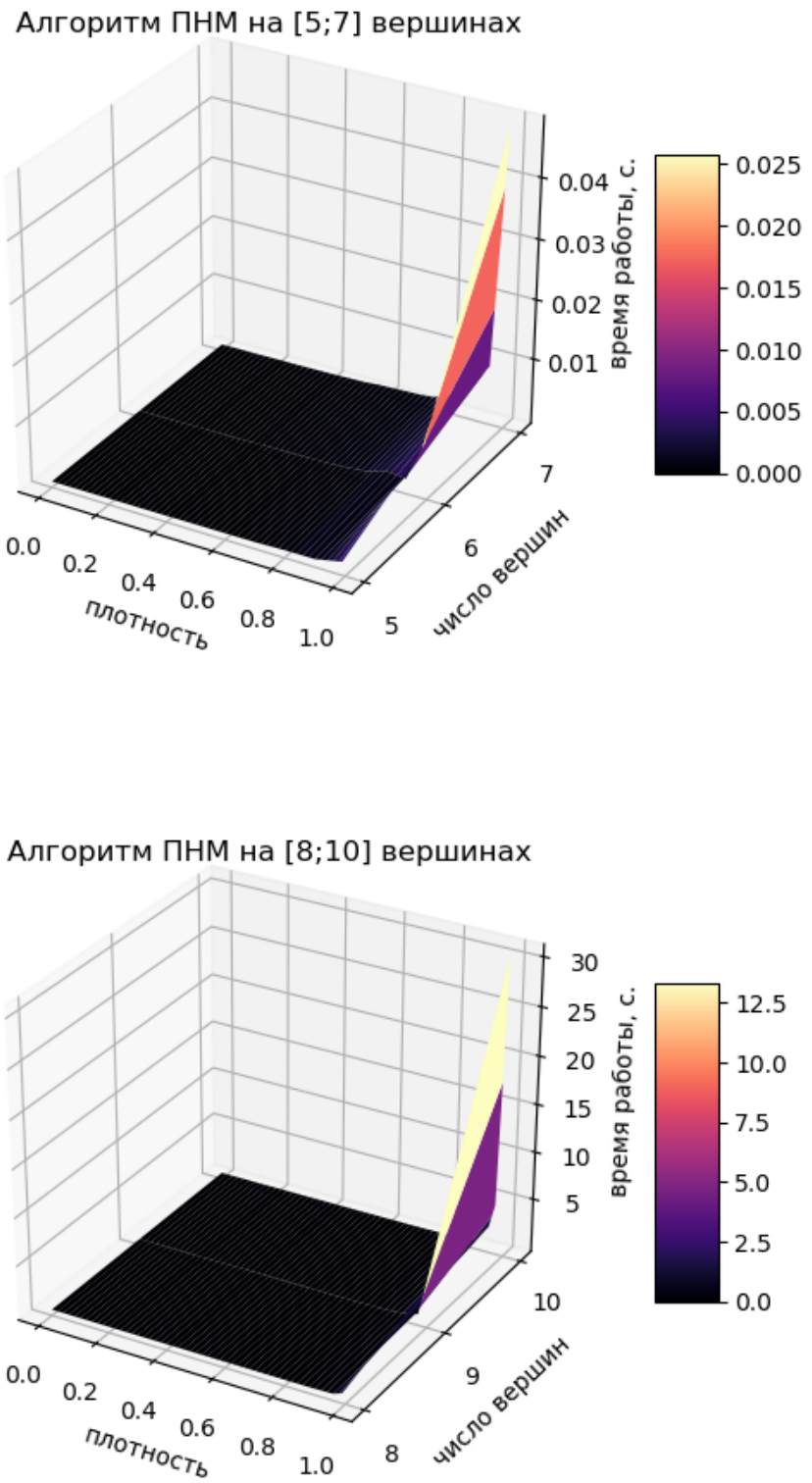


Рис. 2: Время работы алгоритма ПНМ на [5;10] вершинах

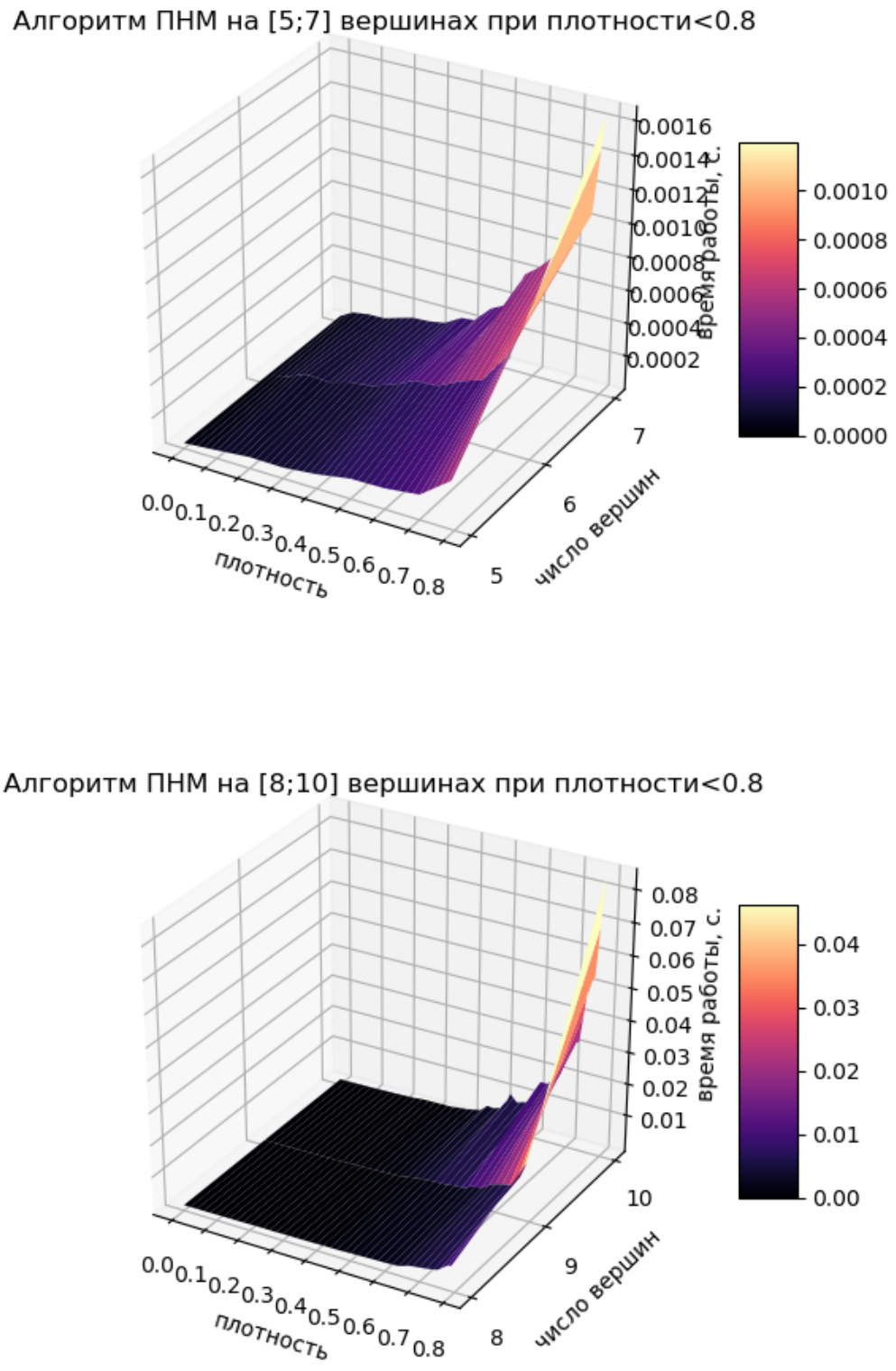


Рис. 3: Время работы алгоритма ПНМ на [5;10] вершинах при плотности <0.8

6.3 Оптимизированный перебор

На графиках (рис.4) приведена зависимость времени работы реализации алгоритма оптимизированного перебора независимых множеств от числа вершин в графе и плотности этого графа.

Видно, что при плотности графа от 0.2 до 0.8 алгоритм работает значительно дольше, чем при остальных значениях плотности.

Быстрая работа алгоритма на разреженных графах объясняется тем, что в них два различных максимальных независимых множества, как правило, имеют большое количество общих вершин. Из-за этого проверка, проводимая на пункте 3 отсекает бóльшую часть множеств.

Повышение эффективности алгоритма на плотных графах с большим числом вершин можно объяснить тем, что число максимальных независимых множеств k в них не сильно превышает хроматическое числоχ. Поэтому остаётся немного вариантов выбора $c(c \geq \chi)$ множеств из k.

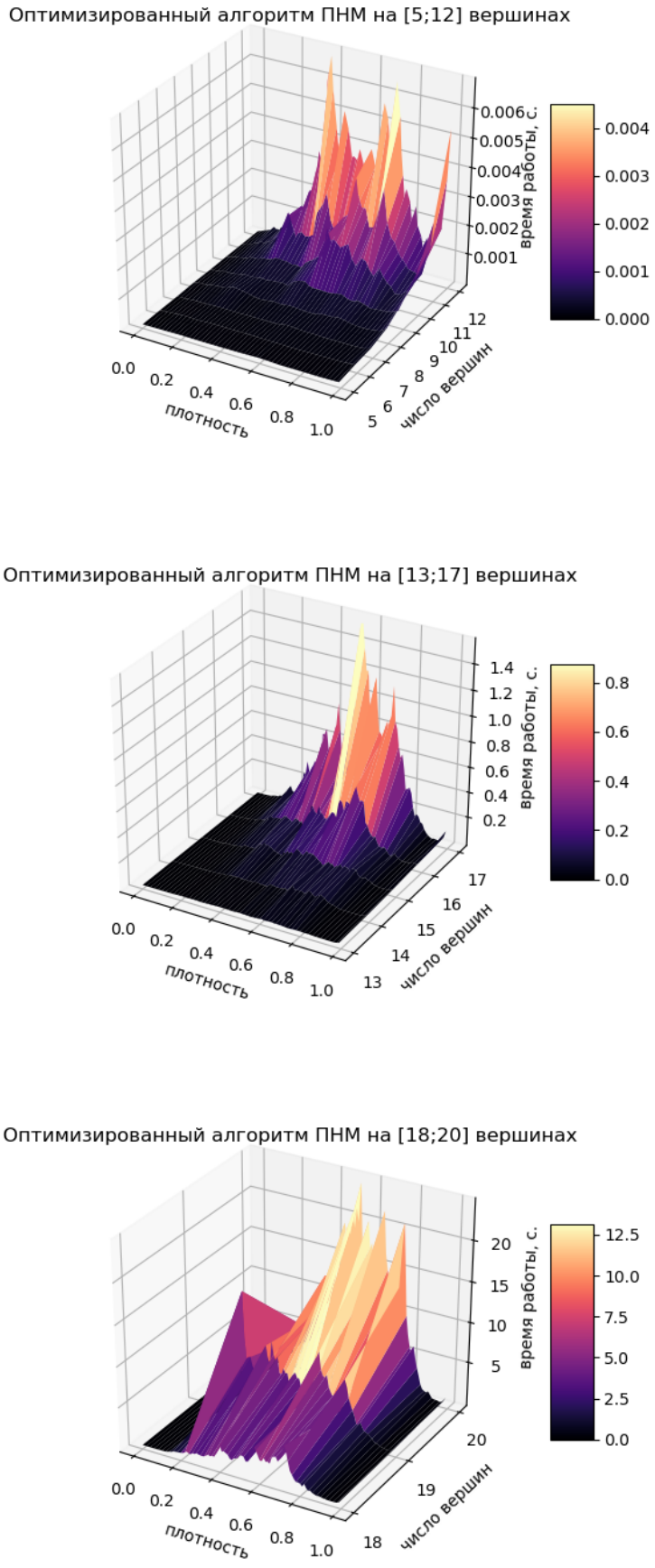
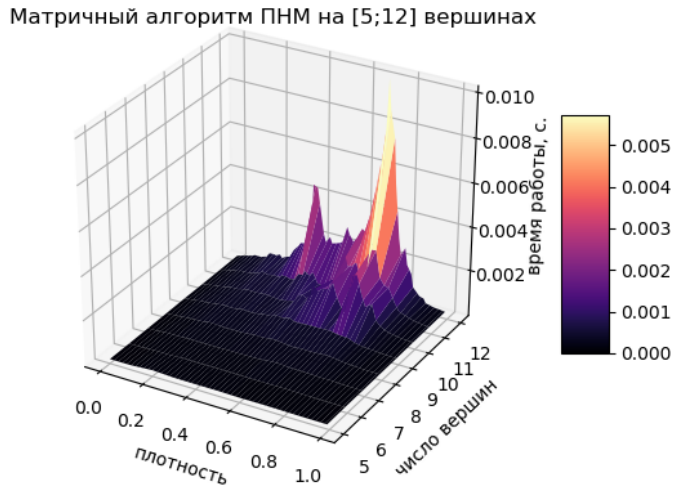


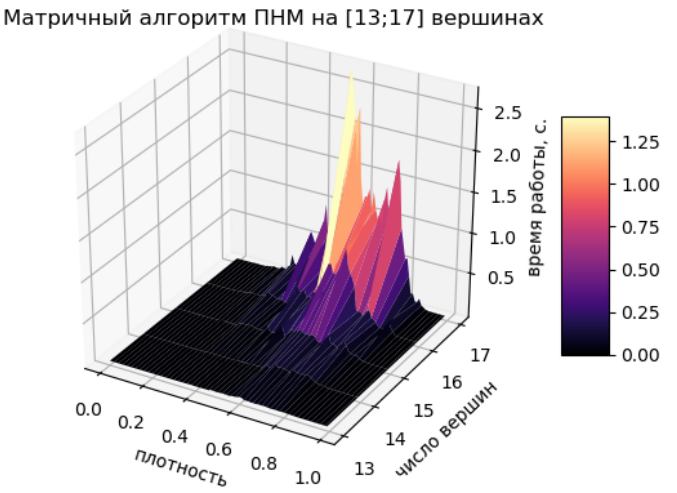
Рис. 4: Время работы оптимизированного алгоритма ПНМ на [5;20] вершинах

6.4 Перебор в матричном виде

На графиках(5) приведена зависимость времени работы реализации матричного алгоритма перебора независимых множеств от числа вершин в графе и плотности этого графа.

Видна схожесть результатов с предыдущим пунктом.





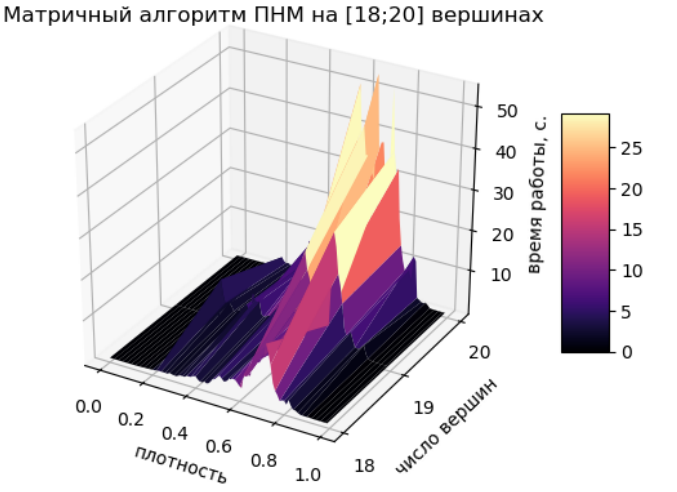


Рис. 5: Время работы матричного алгоритма ПНМ на [5;20] вершинах

Глава 7. Сравнение быстродействия алгоритмов

В графиках (рис. 6,7,8) используются следующие обозначения:

- Olemskoy - алгоритм Олемского И.В.
- PNMopt - оптимизированный алгоритм перебора независимых множеств
- PNMmatr - матричный алгоритм перебора независимых множеств

На графах небольшого размера (рис. 6) видно, что время работы алгоритма Олемского И.В. превышает время работы PNMopt и PNMmatr.

При раскраске графов с 11 (верх рис. 7) вершинами алгоритмы показывают схожее быстродействие. Однако стоит отметить, что PNMmatr показывает лучшую производительность за исключением графов с плотностью от 0.7 до 0.8.

Уже на графах с 14 вершинами (низ рис. 7) заметно, что алгоритм Олемского И.В. выполняется быстрее, это объясняется более низкой скоростью прироста требуемого на раскраску времени у алгоритма. Замеры производительности на 17 и 20 вершинах (рис. 8) лишь подтверждают это наблюдение.

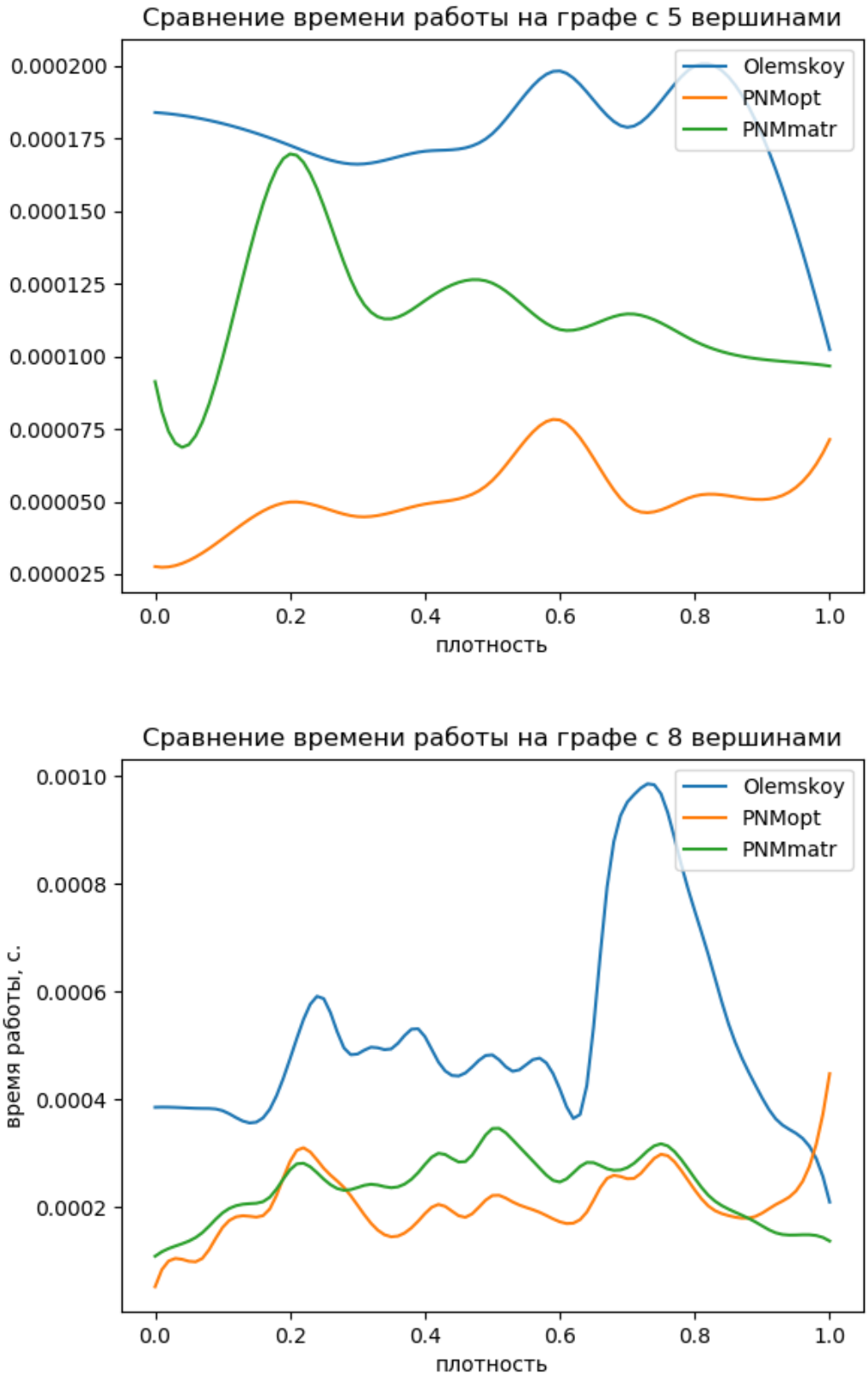


Рис. 6: Сравнение алгоритмов на графах с 8 и 11 вершинами

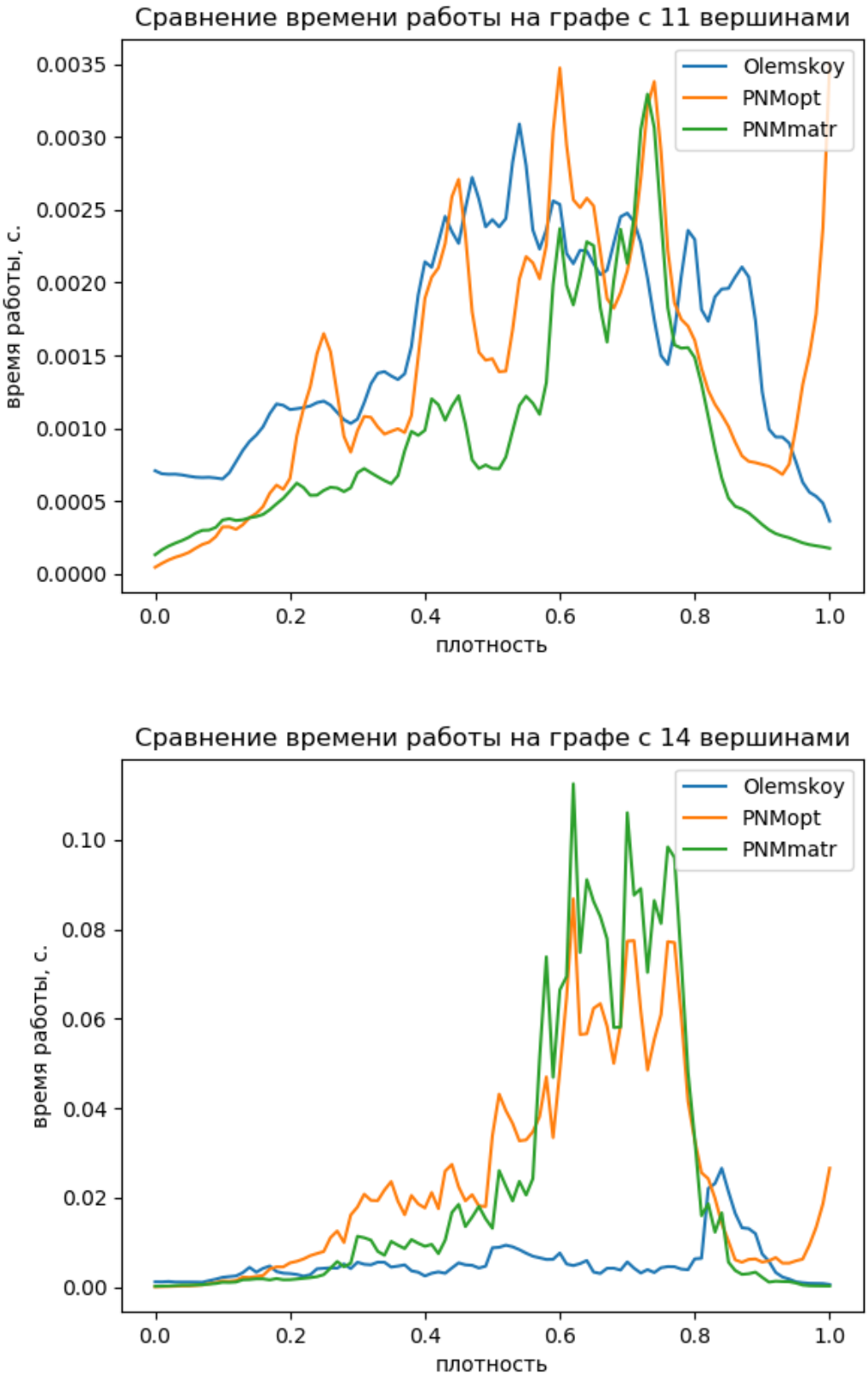


Рис. 7: Сравнение алгоритмов на графах с 14 и 17 вершинами

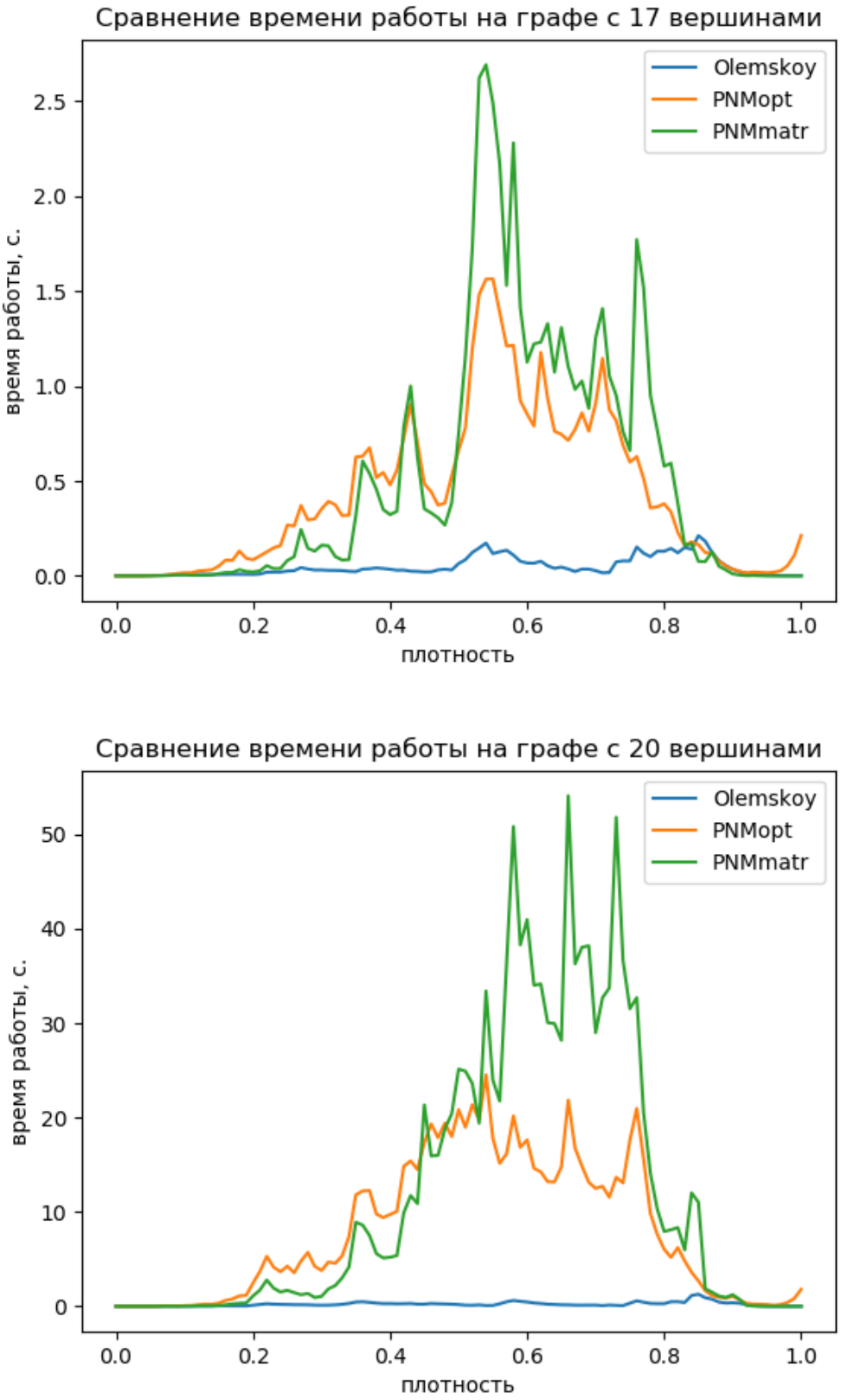


Рис. 8: Сравнение алгоритмов на графах с 20 вершинами

Глава 8. Заключение

В результате проделанной работы были реализованы[14] алгоритмы раскраски графа Олемского И.В. и Новикова Ф.А. Было доказано утверждение, позволившее разработать и реализовать две модификации алгоритма Новикова Ф.А.

Для всех реализованных алгоритмов было измерено время работы на графах с различным число вершин и плотностью. Результаты были представлены в виде трёхмерного графика зависимости времени от числа вершин и плотности. По полученным результатам было решено провести прямое сравнение алгоритма Олемского И.В. и двух модификаций алгоритма Новикова Ф.А.

Сравнение показало, что алгоритм Олемского И.В. работает дольше на графах малой размерности. Однако имеет более низкую скорость прироста требуемого на раскраску времени. Поэтому при раскраске небольших графов с числом вершин менее 12 применение модифицированных алгоритмов Новикова Ф.А. предпочтительнее, но с ростом числа вершин алгоритм Олемского И.В. становится более рациональным выбором.

Список литературы

[1] Marx, Daniel (2004). "Graph Coloring Problems and Their Applications in Scheduling". in Proc. John von Neumann PhD Students Conference: 1–2.

[2] Боханко, А. С; А. Ю Дроздов, С. В Новиков, С. Л Шлыков «Распределение регистров методом раскраски графа несовместимости для VLIW-архитектур»Russian Academy of Science : журнал. — 2005. — Т. 8.

[3] Jones, Mark T.; Paul E. Plassmann. «Scalable Iterative Solution of Sparse Linear Systems»Parallel Computing : journal. — 1994. — Vol. 20, no. 5. — P. 753—773.

[4] Новиков Ф. А. «Дискретная математика для программистов»стр.358-359 Питер, 2009 г.

[5] Bron C., Kerbosh J. (1973) «Algorithm 457 — Finding all cliques of an undirected graph»Comm. of ACM, 16, p. 575—577
https://dl.acm.org/doi/10.1145/362342.362367

[6] Карп Р. (1972) «Reducibility Among Combinatorial Problems»
https://people.eecs.berkeley.edu/ luca/cs172/karp.pdf

[7] Время выполнения операций в Python
https://wiki.python.org/moin/TimeComplexity

[8] Модуль bitarray для Python
https://pypi.org/project/bitarray/

[9] Олемской И.В. «Методы интегрирования систем структурно разделенных дифференциальных уравнений»стр.180, Издательство Санкт-Петербургского университета, 2009 г.

[10] Пакет numpy для Python
https://numpy.org/

[11] Технология Precision Boost
https://www.amd.com/ru/support/kb/faq/cpu-pb2

[12] Модуль timeit
https://docs.python.org/3/library/timeit.html

[13] Бибилиотека matplotlib для Python
https://matplotlib.org/3.2.1/index.html

[14] Реализация всех использованных алгоритмов
https://github.com/N0fail/GraphsColorize