Санкт-Петербургский государственный университет
Факультет прикладной математики — процессов управления
Кафедра механики управляемого движения

Гармашов Константин Сергеевич

Магистерская диссертация

Программная реализация алгоритмов вычисления хроматического числа и их сравнение

Прикладная математика и информатика 010402
Математическое моделирование в задачах естествознания

Научный руководитель,
доктор физ.-мат. наук Олемской И.В.

Санкт-Петербург
2018

Содержание

Введение

История задачи «о раскраске графа» насчитывает более 200 лет. Первый раз она была упомянута Францисом Гутри, который ,пытаясь раскрасить карту округов Англии, сформулировал «проблему четырёх красок»,отметив, что четырёх цветов достаточно, чтобы раскрасить карту так, чтобы любые два смежных региона имели разные цвета. В последствии данная задача получила более широкое распространение только с 1970-х годов. Это было связано с развитием вычислительной техники. Так, например, с 1981 года раскраска графа стала применяться для распределения регистров в компиляторах.

Разнообразные задачи, возникающие при планировании производства, составлении графиков осмотра, хранении и транспортировке товаров и т.д., могут быть представлены часто как задачи теории графов, тесно связанные с «задачей о раскраске».

Сегодня для решения данных задач прибегают к вычислительной технике, на которой обычно работает тот или иной алгоритм. В связи с этим, при выборе алгоритма необходимо знать скорость его работы, сложность реализации, объём требуемой памяти.

Зачастую, из-за сложности нахождения хроматического числа используются эвристические алгоритмы, которые находят приближённое решение, т.к. скорость точных алгоритмов оставляет желать лучшего. Таких алгоритмов достаточно много, но они не являются темой данной работы.

Большинство из точных алгоритмов перебирают полностью все решения тем или иным способом, например, алгоритм, который был упомянут в книгах [1] и [2], основан на рассмотрении максимально независимых множество. Для упрощения в этой работе будем называть его алгоритмом Лорьера-Новикова. Науке на данный момент известно довольно много алгоритмов построения максимально независимых множеств. В качестве алгоритма построения максимально независимых множеств в книге [2] предложен один из вариантов, который, по сути, является алгоритмом Брона-Кербоша.

Ещё один довольно известный алгоритм нахождения максимально независимых множеств - алгоритм Магу. Алгоритм нахождения хроматического числа, основанный на нём, полностью описан в методических материалах [4].

Олемской И.В. разработал новый алгоритм, который был основан на алгоритме выделения структурных особенностей систем обыкновенных дифференциальных уравнений [3]. Именно с ним и будет проведено основное сравнение. Следует отметить, что для него ещё нет аналитической оценки его работы.

Постановка задачи

Определение 1 *Пусть G(V,E) — неориентированный граф без кратных рёбер и петель, где V – множество вершин, E – множество рёбер.*

Определение 2 ***Правильной раскраской*** *графа G(V,E) будем называть такое отображение ϕ из множества вершин V в множество красок $\{c_1, ..., c_t\}$, что для любых двух смежных вершин u и v выполняется $\phi(u) \neq \phi(v)$. Также её называют* ***t-раскраской****.*

Определение 3 *Минимальное число t, для которого существует t-раскраска графа G(V,E), называется* ***хроматическим числом*** *и обозначается $\nu(G)$.*

Определение 4 *Матрица смежности(инцидентности) графа G с конечным числом вершин n (пронумерованных числами от 1 до n) — это квадратная матрица A размера n, в которой:*

$$a_{ij} = \begin{cases} 0, & \text{если } \nexists E_{ij} \in E, \\ 1, & \text{если } \exists E_{ij} \in E. \end{cases}$$

Задачей о раскраске графа является нахождение $\nu(G)$-раскраски, которая относится к классу NP-полных задач, т.е. все известные решения относятся к типу «перебери все варианты».

Определение 5 *Плотность графа определяется по формуле*

$$D = \frac{2|E|}{|V|(|V|-1)} \tag{1}$$

Максимальное число рёбер можно найти по формуле: $\frac{1}{2}|V|(|V|-1)$.

Задачи данного исследования:

1. Реализовать алгоритмы нахождения хроматического числа наиболее эффективным способом. Например, в некоторых случаях понадобятся особые структуры, такие как **битовые шкалы**.
2. Реализовать функцию построения случайного графа размерности и плотности
3. Построить таблицы скорости вычисления для графов с разной размерностью и плотностью
4. Построить графики зависимостей скорости работы алгоритмов от плотности и размерности
5. Описать результаты

Битовые шкалы

Пусть задано конечное множество(*универсум*) U, размерность которого равна n. Элементы этого множества можно пронумеровать:

$$U = \{u_1, ..., u_n\} \tag{2}$$

Подмножество A множества U можно представить **битовой шкалой**(**кодом**) C : **array**$[1..n]$ of 0..1, в котором:

$$C[i] = \begin{cases} 0, & \text{если } u_i \in A \\ 1, & \text{если } u_i \notin A \end{cases} \tag{3}$$

Операции над множествами A и B, являющимися подмножествами U, можно описать простыми операциями. Код пересечения множеств A и B – это логическое произведение кодов множеств. Код объединения – логическая сумма кодов. Код дополнения множества A – инверсия кода множества A. В большинстве компьютеров для этих операций есть соответствующие машинные команды.

Таким образом, такое представление для множеств позволяет вычислять эти операции быстрее в сравнении с такими же операциями для списков, массивов и подобных структур, реализованных и содержащихся в стандартных библиотеках многих языков программирования.

Алгоритм Олемского

Алгоритм разработан на алгоритме выделения структурных особенностей СОДУ [3]. В рассматриваемом алгоритме используется матрица смежности размерности n, решается задача о поиске элементарного преобразования матрицы, обеспечивающего приведение к нульдиагональному виду с минимальным значением нульдиагональных блоков ν. Алгоритм является модификацией алгоритма выделения структурных особенностей систем обыкновенных дифференциальных уравнений, описанный в книге [3].

Введём некоторые определения для формализации алгоритма. Здесь и в дальнейшем $I = \{1, ..., n\}$.

Опредление 6 *q-ое горизонтальное структурное множество матрицы* A

$$h_q(A) = \{r | a_{qr} = 0, r \in I\}, q \in I. \tag{4}$$

Опредление 7 *r-ое горизонтальное структурное множество матрицы* A

$$v_r(A) = \{r | a_{qr} = 0, q \in I\}, r \in I. \tag{5}$$

Определения данных множества нужны для формирования множеств

$$D_{(q,r)}(A) = h_q(A) \cap v_r(a),\ q, r \in I,\ q < r, \tag{6}$$

характеризующих структурные особенности исходной матрицы A.

Будем предполагать, что нам каким-либо образом удалось выделить ν подмножеств (нульдиагональные блоки)

$$J^j = \{i_1^j, i_2^j, ..., i_{k(j)}^j\} \subset I,\ j = 1, ..., \nu;$$
$$\cup_{j=1}^{\nu} J^j = I, J^p \cap J^q = \varnothing, p \neq q,\ p, q \in I_\nu$$
$$I_\nu = \{1, ..., \nu\}$$

Их элементы задают номера строк и столбцов исходной матрицы A, на пересечении которых находятся нулевые матрицы размерности k(j). Это позволяет построить подстановку строк и столбцов

$$\pi = \begin{pmatrix} 1 & 2 & ... & k(1) & k(1)+1 & ... & \sum_{k=1}^{2} k(r) & ... \\ i_1^1 & i_2^1 & ... & i_{k(1)}^1 & i_1^2 & ... & i_{k(2)}^2 & ... \\ & & & ... & \sum_{r=1}^{\nu-1} k(r)+1 & ... & \sum_{r=1}^{\nu} k(r) & \\ & & & ... & i_1^\nu & ... & i_{k(\nu)}^\nu & \end{pmatrix},$$

так и перестановочную матрицу P=$\{b_{\xi\nu}\}$:

$$b_{\xi\nu} = \begin{cases} 0, & \text{если } \nu = \pi(\xi), \\ 1, & \text{если } \nu \neq \pi(\xi), \end{cases}$$

такую, что матрица PAP' и будет матрицей, с числом нульдиагональных блоков ν. Причём равенство $\pi(\xi) = \nu$ в перестановке $\pi = (\pi(1), ..., \pi(n))$ означает, что ν-ый столбец(строка) исходной матрицы будет ξ-м столбцом(строкой) в преобразованной.

Очевидное свойство множеств $D_{(q,r)}$ для элементов подматрицы $O_{k(j)}$(j-го нульдиагонального блока, $j = 1, ..., \nu$) в этом случае

$$\begin{aligned} \overline{D}^{j,s}_{(i_s, i_{k(j)+1-s})} \equiv D_{(i_s, i_{k(j)+1-s})} \cap \{i_s, ..., i_{k(j)+1-s}\} = \\ = \{i_s, ..., i_{k(j)+1-s}\}, s = 1, ..., \left[\frac{k(j)}{2}\right], \end{aligned} \tag{7}$$

является конструктивным, где [a] – целая часть от деления. Верхние индексы j и s будут обозначать: j–номер нульдиагонального блока; s–номер пары, удовлетворяющей свойству (7), именуемый в дальнейшем, как s-й уровень j-ой ветви дерева перебора. Именно опираясь на приведённое свойство, в рассматриваемом ниже алгоритме, происходит отбор элементов $i^j_r \in I, r = 1, ..., k(j)$, которые должны войти в подмножество $J^j = \{i^j_1, ..., i_k(j)^j\}$.

Следует отметить, что на каждом s-ом уровне j-ой ветви дерева перебора алгоритма при построении j-го нульдиагонального блока формируются, и в дальнейшем активно используются, несколько множеств.

1. Первое из них – опорное множество $\omega^{j,s}$. Оно строится для каждого первого (s=1) уровня j-ой ветви дерева перебора по правилу

$$\omega^{j,1} := I \ (\cup_{p=1}^{j-1} J^p). \tag{8}$$

   Естественно, что для первой ветви множество $\omega^{1,1} = I$. Здесь же вводится изначально пустое множество $F^{j,1}$, выполняющее функцию памяти о уже рассмотренных элементах множества $\omega^{j,1}$ при переборе на этом уровне для j-й ветви.

2. Если опорное множество $\omega^{j,s} = \varnothing$, то при $s = 1$ переходим на выполнение пп.8, а при s>1 на пп.5, иначе выполняем следующий пункт алгоритм - пп.3.

3. При непустом опорном множестве($\omega^{j,s} \neq \varnothing$), формируются ещё два множества этого уровня:

   • множество $G^{j,s}$, состоящие из элементов (q,r), удовлетворяющих

свойству (7)

$$G^{j,s} = \{(q,r) | \{q,r\} \subset D^{j,s}_{(q,r)} \equiv D_{(q,r)} \cap \omega^{j,s}, q, r \in \omega^{j,s}, q < r\} \quad (9)$$

- множество $Q^{j,s}$ изначально пустое выполняет функцию памяти о использовании на s-ом уровне j-ой ветви элементов множества $G^{j,s}$.

4. Здесь рассматривается множество $G^{j,s} \backslash Q^{j,s}$ – *множество возможных продолжений*. Причём следует выделить два случая.

   - Если множество возможных продолжений не пусто$(G^{j,s} \backslash Q^{j,s} \neq \varnothing)$, то выбирается *узловой элемент* $\alpha^{j,s} = (\alpha_1^{j,s}, \alpha_2^{j,s}) \in G^{j,s} \backslash Q^{j,s}$, для которого

   $$\left| D^{j,s}_{\alpha^{j,s}} \right| = \max_{\alpha} \left| D^{j,s}_{\alpha} \right|, \alpha^{j,s}, \alpha \in G^{j,s} \backslash Q^{j,s}. \quad (10)$$

   И на базе выбранного узлового элемента $\alpha^{j,s}$ и множества $D^{j,s}_{\alpha^{j,s}}$ строятся: опорное множество

   $$\omega^{j,s+1} = \left(\omega^{j,s} \cap D^{j,s}_{(\alpha_1^{j,s}, \alpha_2^{j,s})} \right) \backslash \{\alpha_1^{j,s}, \alpha_2^{j,s}\} \quad (11)$$

   и $F^{j,s+1} := \varnothing$, следующего уровня. Множество $F^{j,s+1}$ (также, как и введённые выше множества $F^{j,1}$ и $Q^{j,1}$) будет нести информацию о элементах опорного множества $\omega^{j,s+1}$, использованных уже на этом уровне.

   Узловой элемент запоминается как рассмотренный на s-м уровне на этой ветви дерева перебора $Q^{j,s} := Q^{j,s} \cup \alpha^{j,s}$. Причём завершается рассмотрение этого случая увеличением номера уровня на единицу (s:=s+1) и переходим на пп.2.

   - Если же множество возможных продолжений пусто$(G^{j,s} \backslash Q^{j,s})$, то из элементов опорного множества $\omega^{j,s} \backslash F^{j,s}$ выбирается концевой элемент $\beta \in \omega^{j,s} \backslash F^{j,s}$. После этого он запоминается как использованный в этом качестве, т.е. становится элементом множества $F^{j,s} := F^{j,s} \cap \beta$. Дальнейшее продвижение по этой ветви регламентируется следующим пп.5 алгоритма.

     При пустом же множестве $\omega^{j,s} \backslash F^{j,s} = \varnothing$, реализуется возврат на предыдущий (s:=s-1) уровень j-ой ветви дерева перебора с переходом на пп.4.

5. Проход по j-ой ветви дерева перебора, при построении j-го нульдиагонального блока J^j, заканчивается в случае выполнения одного из следующих условий:

   - либо, когда опорное множество s-го уровня пусто$(\omega^{j,s} = \varnothing)$ (в этом

случае число элементов, образующих j-й нульдиагональный блок — чётно и содержит 2(s-1) элементов $J^j := \{\alpha_1^{j,1}, \alpha_2^{j,1}, ..., \alpha_1^{j,s-1}, \alpha_2^{j,s-1}\}$);

- либо когда все элементы множества $G^{j,s}$ были уже рассмотрены в качестве *узловых* $(G^{j,s} \backslash Q^{j,s})$ при непустом опорном множестве $(\omega^{j,s} \neq \varnothing)$, причём не все элементы опорного множества были рассмотрены на этот момент в качестве концевых $(\beta \in \omega^{j,s} \backslash F^{j,s} \neq \varnothing)$.

  Во втором случае число элементов, образующих j-й нульдиагональный блок - нечётно. Оно состоит из 2s-1 элементов: $J^j := \{\alpha_1^{j,1}, \alpha_2^{j,1}, ..., \alpha_1^{j,s-1}, \alpha_2^{j,s-1}, \beta\}$. В r^j запоминаем количество уровней, пройденных при построении j-го нульдиагонального блока. Причём для чётного числа элементов множества J^j количество уровней $r^j := s - 1$, для нечётного $r^j := s$.

6. Использование свойств сформированного множества J^j, позволяет существенно сократить количество ветвей перебора. Для этого, после построения каждого множества F^j, необходимо проводить *прореживание* - исключать из рассмотрения элементы, выбор которых в качестве узловых на s-м уровне j-й ветви не может изменить ни элементного состава множества J^j, ни числа нульдиагональных блоков.

   Формализация этой операции требует введения трёх вспомогательных множеств.

   - Множество элементов, удовлетворяющих свойству (7) и использованных, начиная с s-го уровня в качестве узловых при построении j-го нульдиагонального множества

   $$\Psi^{j,s} := J^j \backslash \left(\cup_{\mu=1}^{s-1} \{\alpha_1^{j,\mu}, \alpha_2^{j,\mu}\} \right). \tag{12}$$

   Множество неперспективных элементов

   $$Z^{j,s} := \{\alpha \in (G^{j,s} \backslash Q^{j,s}) \cap (\Psi^{j,s} \times \Psi^{j,s}) | D_\alpha^{j,s} \equiv \Psi^{j,s}\},$$

   содержащее элементы, использование которых в качестве узловых, позволит построить уже сформированное множество J^j. Здесь

   $$s = \begin{cases} r^j - 1, r^j - 2, \ldots, 1, & \text{если k(j) - нечётно;} \\ r^j, r^j - 1, \ldots, 1, & \text{если k(j) - чётно.} \end{cases}$$

   Заключительная операция прореживания - пересылка неперспективных элементов во множество уже использованных этого уровня при построении j-го нульдиагонального блока: $Q^{j,s} := Q^{j,s} \cup Z^{j,s}$.

   - Множество B формируется из рассмотренных в процессе перебора множеств J^1. Это означает, что $B := B \cup \{\Psi^{1,1}\}$, если $\{\Psi^{1,1}\} \in B$. В

начале работы алгоритма $B := \varnothing$. Причём, если $J^1 \in B$, то прямой ход прекращаем и начинаем реализовывать возврат с последнего уровня первого блока ($s := r^1$) с переходом на пп.3 алгоритма.

7. Построение очередного нульдиагонального блока начинаем с увеличения их общего числа (текущего номера) на единицу ($j := j + 1$) и установки в начальное положение счётчика уровня (s:=1) и множеств уже использованных на этом уровне элементов в качестве *узловых*($Q^{j,1} := \varnothing$) и *концевых*($F^{j,1} := \varnothing$). После чего переходим на пп.1 алгоритма.

8. В случае, если $\omega^{j,1} = \varnothing$, (т.е построена полная ветвь дерева перебора - раскраска найдена и $I \equiv (\cup_{p=1}^{j-1} J^p)$, причём число её нульдиагональных блоков равно $\nu = j - 1$), то проводится сравнение ν с *рекордным* ν_0 – минимальным числом нульдиагональных блоков из всех уже построенных раскрасок на полных ветвях дерева перебора алгоритма.

   Если число нульдиагональных блоков ν, вновь построенной раскраски , меньше чем ν_0, то сама вновь построенная раскраска и число её нульдиагональных блоков $\nu_0 := \nu$ становятся рекордными.

   Причём при выполнении равенства

$$\left[\frac{|\omega^{1,1}|}{\rho^{1,1}}\right] = \begin{cases} \nu_0 - 1, & \text{если } \left[\frac{|\omega^{1,1}|}{\rho^{1,1}}\right] \neq \frac{|\omega^{1,1}|}{\rho^{1,1}}; \\ \nu_0, & \text{если } \left[\frac{|\omega^{1,1}|}{\rho^{1,1}}\right] = \frac{|\omega^{1,1}|}{\rho^{1,1}}; \end{cases}$$

   алгоритм заканчивает работу, так как дальнейший перебор не позволит построить разбиение с меньшим числом нульдиагональных блоков. Полученное разбиение и является решением поставленной задачи, а число нульдиагональных блоков ν_0 – хроматическим числом.

Стартовое *рекордное* значение ν_0 равно размерности матрицы n. Причём после сравнения и проверки на рекордность начинаем процедуру возврата вернувшись на последний уровень ($s := r^j$)ν-й ветви дерева перебора с переходом на пп.4 алгоритма.

Отметим, что при движении по дереву перебора как вверх - прямой ход, так и вниз - обратный ход, осуществляются блочные проверки на перспективность дальнейшего продвижения по текущей ветви дерева перебора.

А) Для блочной проверки этого типа характерно, что на каждом первом уровне при построении очередного j-го нульдиагонального блока, в соответствии с оценкой (13), будет приниматься решение о перспективности дальнейшего продвижения по текущей ветви перебора. Очевидно,

что при выполнении неравенства

$$j-1+\frac{|\omega^{1,1}|}{\rho^{1,1}} \geq \nu_0 \tag{13}$$

дальнейшее продвижение по этой ветви нецелесообразно, так как приведёт к построение раскраски с числом нульдиагональных блоков ν из интервала

$$j-1+\frac{|\omega^{1,1}|}{\rho^{1,1}} \leq \nu \leq j-1+|\omega^{j,1}|. \tag{14}$$

Это значит, что количество нульдиагональных блоков будет не меньше, чем рекордное. В этом случае необходимо вернуться на последний уровень предыдущего блока(j:=j-1,$s := r^j$) с переходом на пп.4 алгоритма.

Здесь и в дальнейшем

$$\rho^{j,s} = \begin{cases} |D^{j,s}_{\alpha^{j,s}}|, & \text{если } G^{j,s}\backslash Q^{j,s} \neq \varnothing; \\ 1, & \text{если } G^{j,s}\backslash Q^{j,s} = \varnothing \text{ и } \omega^{j,s}\backslash F^{j,s} \neq \varnothing. \end{cases}$$

B) Проверка этого типа проводится только при построении первого нульдиагонального блока. Возврат по дереву перебора с понижением уровня (s:= s1, при s >1) производится на пп.4 алгоритма, если при рассмотрении очередного элемента $\alpha^{1,s} \in G^{1,s}\backslash Q^{1,s}$ на s-ом уровне на роль узлового при построении первого нульдиагонального блока, выполняется неравенство

$$2(s-1)+\rho^{1,s} < \left[\frac{n}{\nu_0}\right] \tag{15}$$

C) Третий тип блочной проверки используется только при выполнении равенства $j = \nu_0 - 1$. Если при рассмотрении очередного элемента $\alpha^{j,s} \in G^{j,s}\backslash Q^{j,s}$ на s-ом уровне на роль узлового при построении j-го нульдиагонального блока, не выполняется равенство

$$2(s-1)+\rho^{j,s} = |\omega^{j,1}|, \tag{16}$$

то осуществляется возврат по дереву перебора($j := j-1, s := r^j$) с переходом на пп.4 алгоритма.

Отметим, что проверки всех трёх типов реализуются в пп.4 алгоритма при выборе очередного элемента на роль *узлового* или *концевого*.

Окончание работы алгоритма, если ранее его работа не была прервана по приведённым выше ограничениям, происходит при выполнении неравенства $\frac{|\omega^{1,1}|}{\rho^{1,1}} \geq \nu_0$.

Algorithm 1 Псевдокод алгоритма Олемского

$G(V, E)$ ▷ Граф
$A = A(G)$ ▷ Матрица смежности
$I_n := \{1, ..., n\}$ ▷ пронумерованные вершины
$bestJ := \varnothing$ ▷ Лучшая раскраска
$\nu_0 := n$ ▷ лучшее число цветов
$B := \varnothing$ ▷ Множество рассмотренных $J[1]$
D ▷ Множества возможных продолжений каждого уровня
function ColorGraph
    $J[1] := \varnothing$
    $\omega[1, 1] := \varnothing$
    Build($\omega[1, 1], J[1]$)
    return $bestJ$
end function
▷ Процедура прохода по дереву
procedure Build($\omega[k, s], J[k]$) ▷ ω-текущее опорное множество, J[k]-текущее множество вершин для строящегося блока, k-номер свободного цвета, s-уровень
    if $\omega[k, s] = \varnothing$ **then**
        $\omega[k, s] := I_n \setminus \cup_{p=1}^{k} J[p]$
    end if
    if $\omega[k, s] = \varnothing$ **then** ▷ Если опорное множество пусто, то построение блока закончено
        if $k = 1$ **then** ▷ Если мы построили первый блок
            if $\{J[1]\} \in B$ **then** ▷ Проверяем на наличие такого блока в уже использованных
                return ▷ Строим следующий блок первый блок
            else
                $B := B \cup \{J[1]\}$
            end if
        end if
        Thinning(k) ▷ Процедура прореживания множества возможных продолжений D
        if $\cup_{p=1}^{k} J[p] = I_n$ **then**
            if $bestJ = \varnothing \ \vee \ \nu_0 > k$ **then**
                $\nu_0 := k$
                $bestJ := J$
            end if
        else
            $J[k + 1] := \varnothing$

Algorithm 2 Псевдокод алгоритма Олемского

```
            ω[k+1,1] = I_n \ ∪_{p=1}^{k} J[p]
            BUILD(ω[k+1,1], J[k+1])
        end if
    else
        D[k,s] := CreateVariants(ω[k,s])          ▷ Создание множества возможных продолжений по формуле (9)
        if D[k,s] = ∅ then
            BUILDENDBYCENTER(ω[k,s], J[k])
        else
            BUILDNOTNULLVARIANTS(ω[k,s], J[k])
        end if
    end if
end procedure

procedure BUILDNOTNULLVARIANTS(ω[k,s], J[k]))
    for ∀α ∈ D[k,s] do
        ρ := |D[k,s][α]|
                                                  ▷ блочные проверки
        if (s = 1 & k − 1 + |ω[k,s]|/ρ ≥ ν_0)∨
           (k = 1 & 2(s − 1) + ρ < [n/ν_0])∨
           (k = ν_0 − 1 & 2(s − 1) + ρ ≠ |ω[k,1]|)∨
           k = 1 & s = 1 & n/ρ ≥ ν_0
then
            return
        end if
        J[k] := J[k] ∪ {α_1, α_2}
        ω[k,s+1] = ω[k,s] \ {α_1, α_2}
        BUILD(ω[k,s+1], J[k])
        J[k] := J[k] \ {α_1, α_2}
    end for
end procedure

procedure BUILDENDBYCENTER(ω[k,s], J[k])
    if (s = 1 & k − 1 + |ω[k,s]| ≥ ν_0) then      ▷ блочные проверки
        return
    end if
```

Algorithm 3 Псевдокод алгоритма Олемского

```
    for ∀β ∈ ω[k, s] do
        ω[k, s + 1] := ∅
        J[k] := J[k] ∪ {β}
        Build(ω[k, s + 1], J[k])
        J[k] := J[k] \ {β}
    end for
end procedure
procedure Thinning(k)
    l:= length(J[k])
    s:=[l/2]
    if l-нечётно then
        s:=s-1
    end if
    for i := s, ..., 1 do
        Φ[k, i] := J[k] \ (∪_{ν=1}^{i−1} {α1[k, ν], α2[k, ν]})
        for ∀α ∈ Z do
            if D[k, i][α] = Φ[k, i] then
                Remove(D[k, i], α)    ▷ Удаление из D[k,i] элемента D[k,i][α]
            end if
        end for
    end for
end procedure
```

Алгоритм Лорьера-Новикова

Существует точный алгоритм, описанный Новиковым в книге [2]. Для его формализации потребуется постановка задачи нахождения максимально независимых множеств.

Пусть задан граф *G(V,E)*. Найти такое множество $X, X \in V$, что

$$\omega(X) = \max_{Y \in \epsilon} \omega(Y) \tag{17}$$

где $\omega(Y) = |Y|, \epsilon = \{Y \in V | \forall u, v \in Y((u, v) \notin E)\}$.

Основная идея алгоритма состоит построении рекурсивной процедуры *P*:

1. Выбрать в графе *G* некоторое максимальное независимое множество вершин *S*.
2. Покрасить вершины множества *S* в очередной цвет.
3. Применить процедуру *P* к графу *G-S*

На псевдокоде процедура *P* выглядит следующим образом.

Algorithm 4 Псевдокод функции *P*

```
C                                  ▷ Номера цветов, прописанные вершинам
procedure P(G(V,E),i)              ▷ i-номер свободного цвета
    if V = ∅ then
        return                     ▷ раскраска закончена
    end if
    S:=Selectmax(G)                ▷ S-максимальное независимое множество
    C[S] := i                      ▷ раскрашиваем вершины множества S в цвет i
    P(G-S,i+1)                     ▷ Рекурсивный вызов
end procedure
```

Надо заметить, что функция *Selectmax* может быть реализована множествами вариантов, мы же будем использовать алгоритм Брона-Кербоша:

Algorithm 5 Построение максимально независимых множеств

procedure SELECTMAX(граф G(V,E), заданный списками смежности $\Gamma[v]$)
$k := 0$ ▷ количество элементов в текущем независимом множестве
$S[k] := \varnothing$ ▷ независимое множество из k вершин
$Q^-[k] := \varnothing$ ▷ множество вершин уже использованных для расширения S[k]
$Q^+[k] = V$ ▷ Множество вершин, которые можно использовать для расширения S[k]
M1: ▷ шаг вперёд
select $v \in Q^+[k]$ ▷ расширяющая вершина
$S[k+1] := S[k] \cup \{v\}$ ▷ Расширенное множество
$Q^-[k+1] := Q^-[k] \setminus \Gamma[v]$ ▷ вершина v использована для расширения
$Q^+[k+1] := Q^+[k] \setminus (\Gamma[v] \cup \{v\})$ ▷ все вершины, смежные с v, не могут быть использованы для расширения
$k := k + 1$
M2: ▷ проверка
for $u \in Q^-[k]$ **do**
if $\Gamma[u] \cap Q^+[k] = \varnothing$ **then**
goto$M3$ ▷ можно возвращаться
end if
end for
if $Q^+[k] = \varnothing$ **then**
if $Q^-[k] = \varnothing$ **then**
yield $S[k]$ ▷ множество S[k] максимально
end if
goto$M3$ ▷ можно возвращаться
else
goto$M1$ ▷ можно идти вперёд
end if
M3: ▷ шаг назад
$v := last(S[k])$ ▷ Последний добавленный элемент
$k := k - 1$
$S[k] := S[k+1] - \{v\}$
$Q^-[k] := Q^-[k] \cup \{v\}$ ▷ вершина v уже добавлялась
$Q^+[k] := Q^+[k] \setminus \{v\}$
if $k = 0$ & $Q^+[k] = \varnothing$ **then**
stop ▷ перебор завершён
else
goto$M2$ ▷ переход на проверку
end if
end procedure

Алгоритм Вейсмана

Алгоритм состоит из двух частей

1. Построение семейства максимально независимых множеств(метод Магу)
2. Выбор минимального числа максимально независимых множеств, покрывающих все вершины(метод Петрика).

Полная структура алгоритма выглядит следующим образом:

1. В матрице инцидентности R для каждой вершины подсчитывается число ненулевых элементов r_i
2. Находится вершина x_i с максимальным r_i
3. Для выбранной вершины x_i записывается выражение $C_i = (x_i \vee x_a x_b ... x_q)$, где $\Gamma x_i = \{x_a, x_b, ..., x_q\}$ - соседи x_i.
4. Из матрицы R удаляются строка и столбец, соответствующие вершине x_i
5. Если R $\neq \varnothing$, то переход на п.2, иначе к п.6
6. Составляется конъюнкция $P_1 = \wedge C_i$. Раскрываются скобки и в полученной дизъюнкции на основе законов булевой алгебры выполняется минимизация. Т.к. в выражении нет отрицаний, используются только два закона: тавтологии $(aa = a)$ и поглощения $(a \vee ab = a)$
7. Результат минимизации записывается в виде $P_1 = \vee K_j$
8. Для каждого K_j строятся дополнения до всех вершины. Полученное семейство множеств $\Phi - \{\phi_1, \phi_2, ..., \phi_l\}$
9. Для каждой вершины $x_i \in X$ определяются подмножества ϕ_j, в которые входит вершина $x_i \in \phi_j$. Составляется дизъюнкция $t_i = \vee \phi_j$
10. Составляется конъюнкция $P_2 = \wedge t_i$. Раскрываются скобки и выполняется минимизация полученной булевой функции
11. Получена дизъюнкция конъюктивных термов $P_2 = \vee(\wedge \phi_j)$. Выбирается конъюнктивный терм $\wedge \phi_j$ с минимальным числом сомножителей. Стоит заметить, что их может быть несколько.
12. Количество сомножителей в полученном терме - хроматическое число графа, а каждое ϕ_j - множество вершин, которые можно окрасить в один цвет.

Отметим, что п.п.1-8 составляют метод Магу, а п.п. 9-11 - метод Петрика.

Также метод Магу может быть использован в качестве функции Selectmax в предыдущем алгоритме.

Algorithm 6 Псевдокод алгоритма

```
function MAGU(R, X)      ▷ Алгоритм Магу. R- матрица инцидентности
    {C,D} = SELECTVERTICES(R,X)
    P1 = GetExpression(C, D)      ▷ Построение и минимизация булевой функции. C_i
    Φ = Complement(P_1)      ▷ Построить дополнения для каждого элемента из P_1
    return Φ
end function
function SELECTVERTICES(R, X)      ▷ Выбрать вершины с максимальными r_i. R-матрица инцидентности, X-вершины
    C = ∅      ▷ Выбранные вершины
    D = ∅      ▷ Соседи выбранных вершин
    while R ≠ ∅ do
        Γx= GetNeighbours(R,X)      ▷ Построить соседей для текущих вершин
        x_max = GetMax(Γx, X)
        C.Add(x_max)
        D.Add(Γx[x_max])
        X.Remove(x_max)
        R.Remove(x_max)      ▷ Удалить выбранную строку и столбец
    end while
    return C,D
end function
function PETRIK(Φ,X)      ▷ Метод Петрика
    T = BUILDT(Φ,X)
    P_2 = GetExpression(T)      ▷ Построение и минимизация булевой функции. C_i
    z = GetMinTerm(P_2)
    return z
end function
function BUILDT(Φ, X)
    T = ∅
    for all x_i ∈ X do
        t_i = ∅
        for all φ_j ∈ Φ do
            if x_i ∈ φ_j then
                t_i.ADD(φ_j)
            end if
        end for
        T.ADD(t_i)
    end for
    return T
end function
function WEISMAN(R,X)      ▷ Алгоритм Вейсмана
    Φ = MAGU(R,X)
    Z = PETRIK(Φ, X)
    return Z
end function
```

Программная реализация

В данном разделе будут рассмотрены нюансы при реализации алгоритмов.

Все программы были написаны на языке C# в среде MS Visual Studio.

Аппаратные характеристики:

1. Процессор: Intel(R) Core(TM) i7-3517U CPU @ 1.90GHz, 2401 МГц, ядер: 2, логических процессоров: 4
2. Объём оперативной памяти: 4 ГБ
3. ОС: MS Windows 10 Pro x64

Надо отметить, что входной граф G(V,E) рассматривался как его матрица смежности в каждом алгоритме.

Алгоритм Олемского

При проходе по дереву построения множеств на каждом уровне перебора, в связи с реализацией рекурсии, при переходе на следующий уровень необходимо копировать текущую конфигурацию.

В конце построения каждой конфигурации раскраски необходимо просеять множества возможных продолжений D(процедура Thinning). Использование битовых шкал уменьшает время обработки данной процедуры.

Алгоритм Лорьера-Новикова

Данный алгоритм разбит на две части: первая часть - реализация функции P, а вторая - построение максимально независимых множеств по алгоритму Брона-Кербоша.

Алгоритм был реализован двумя подходами: с помощью структуры списка(т.к. требуется большое кол-во операций добавления и удаления) и битовой шкалы для множеств Q^-, Q^+, S.

Вместо перехода по меткам, предложенным в псевдокоде Новиковым, были созданы отдельные функций.

Для научного интереса был реализован ещё один вариант, где в качестве функции Selectmax был выбран метод Магу.

Алгоритм Вейсмана

Особенность данного алгоритма в том, что нужно произвести минимизацию булевых функций P_1 в пп.6 и P_2 в пп.10. Для минимизации надо построить предварительно все слагаемые в выражении. Чтобы построить все слагаемые, необходимо пройтись по дереву построения.

Для начала рассмотрим реализацию метода Магу.

При построении P_1 каждый узел состоит либо из x_i, либо из Γx_i. Для упрощения можно использовать битовую шкалу, универсумом которой является множество вершин. Тогда операция построения в каждом узле будет просто пересечением текущего слагаемого с множеством в узле. В процессе добавления слагаемого в множество P_1 проверяется закон поглощения, т.к. закон тавтологии при использовании битовых шкал выполняется при построении множеств.

После всех проделанных операций имеются построенные слагаемые. Используя операцию дополнения для них, мы получим все максимально независимые множества.

Рассмотрим реализацию метода Петрика.

Далее составляются T_i, которые являются битовыми шкалам, универсумом которого являются все ϕ_j. по п.10 и уже из них производится построение слагаемых P_2 и их минимизация по той же схеме, как и для P_1. Из полученных слагаемых выбирается то, которое имеет наименьшее число сомножителей, которое и будет является решением задачи.

Полученную конфигурацию остаётся перевести из битовых шкал в номера вершин и вывести результат программы.

Программа создания отчётов

Чтобы построить зависимость каждого алгоритма от размерности и плотности, необходимо получить статистические данные. Для этого была написана консольная программа, которая создаёт отчёты для выбранных алгоритмов.

Программа для каждого выбранного алгоритма параллельно в своём потоке строит отчёт. Границы и шаг для размерности и плотности графов заданы в программе, а также количество повторов для каждой плотности и размерности. После каждой итерации происходит запись результатов работы алгоритма в текущей точке в виде строки, содержащей размерность, плотность и время работы алгоритма.

Для генерации матриц смежности по заданной плотности используется функция, которая считает кол-во рёбер, а затем случайным образом генерирует матрицу смежности для выбранной размерности.

Программа построения графиков

Программа написана с использованием технологии ASP.NET WebAPI. На сервере Контроллер AlgDataController содержит метод GetSplittedByDimension, который на вход принимает название алгоритма, для которого нужно считать отчёт.

Главная страница сайта представляет графики зависимости скорости работы от плотности для каждой размерности и от размерности для определённой плотности.

Полученные результаты

1. BinMIS - алгоритм Лорьера-Новикова с использованием битовых шкал.
2. WeismanMIS - алгоритм Лорьера-Новикова с использованием вместо Selectmax метода Магу.
3. Olemskoy - алгоритм Олемского.
4. MIS - алгоритм Лорьера-Новикова без использования битовых шкал(используются списки).
5. Weisman - алгоритм Вейсмана.

Для анализа средней скорости работы были построены интерполяционные графики с помощью библиотеки ChartJS. Важно отметить, что результаты работы относительны и на другой машине с другими конфигурациями они могут отличаться.

На размерности 5(рис.1) Weisman показывает большую разницу во времени работы в отличие от всех остальных. Далее на размерности 6 на плотностях до 0.5 алгоритм показывает результаты не значительно хуже, чем другие алгоритмы, но на плотностях больше 0.5 его скорость работы приближается к скорости работы алгоритма Олемского, что показывает его быстродействие на больших плотностях. На размерности 6 скорость работы на промежутке до 0.5 разница становится более существенной, но на размерности более 0.5 алгоритм начинает показывать результаты лучше, в сравнении с алгоритмами Новикова, но не превосходит по скорости алгоритма Олемского. На размерности 8(рис.4) алгоритм начинает считать значительно дольше на плотностях до 0.45, в остальных случаях время работы не значительно отличается. На следующих размерностях(рис.4 и рис.6) можно заметить большой скачок во времени работы на плотности 0.2 и дальнейшее сглаживание, но во втором случае разность времени работы относительно других алгоритмов на этой плотности уже достигает почти 17 секунд. Последняя просчитанная размерность(рис.8) для алгоритма Вейсмана показывает значительное замедление в скорости работы на низких плотностях. Как и на предыдущих размерностях наблюдаются скачки до 320с на плотностях от 0.15-0.2 и до 930с на плотности 0.3, а на 0.45 скачок не превышает 100с.

В связи с тем, что среднее время работы алгоритма Вейсмана(Weisman) уже на размерности до 10 на плотностях ниже 0.5 варьируется до 900с, то данные для размерностей больше 10 для данного алгоритма не собирались. Такой большой скачок связан с тем, что минимизация булевой функции требует построения всех слагаемых и проверки правила поглощения. Такая операция является трудоёмкой. Но. с другой стороны, данный алгоритм

имеет такое преимущество, как нахождения всех оптимальных раскрасок заданного графа.

Далее рассмотрим различные реализации алгоритма Лорьера-Новикова. Можно заметить, что с увеличением размерности функции зависимости времени работы от плотности для алгоритмов WeismanMIS и MIS стремятся к экспоненте, а графики времени работы алгоритма BinMIS можно описать как предэкспоненциальными, т.к. имеют относительно низкую скорость работы на небольших плотностях, но с ростом размерности разность во времени работы с остальными алгоритмами на этих плотностях становится незначительной. Так, например, на размерности 11 переломная точка находится примерно в плотности 0.3, а уже 13 в точек 0.15. Таким образом, алгоритм BinMIS показывает результат хуже, на более низких плотностях. Также, можно отметить, что в некоторых точках на плотностях до 0.5 и размерностях от 8 до 16 алгоритмы WeismanMIS и MIS обходят алгоритм Олемского в скорости работы. Например, на размерности 9 и плотности 0.3 скорости работы алгоритмов WeismanMIS, MIS и Olemskoy составляют 0.0005c,0.0007c и 0.002c соответственно, а на размерности 15 и плотности 0.25 уже 0.024c, 0.02c и 1.29c соответственно. Анализ плотностей больше 0.5 показывает, что определить явного претендента в скорости работы этой группы алгоритмов невозможно,т.к в разных случаях каждый из них показывает лучший результат, но важно отметить, что на этих они все вместе проигрывают в скорости работы алгоритму Олемского.

В целом, на рассмотренных размерностях можно отметить, что алгоритм Олемского является фаворитом на высоких плотностях. Но на плотностях от 0.15 до 0.4 алгоритм считает дольше, но не значительно. Так на размерности 15 время работы на плотности 0.25 составляет 1.29 секунды, а ,например, MIS на этой же плотности даёт результат за 0.02 секунды, т.е. разница составляет 1.27 секунды. А на размерности 16 и на той же плотности разница уже 5.16 в пользу MIS. Это вызвано тем, что данный алгоритм производит построение конфигураций раскраски и их оценку, в то время как другие алгоритм проводят полный перебор всех имеющихся. В результате, построение на низкой плотности проводится медленнее, в отличии от перебора.

Таким образом, можно сказать, что на рассмотренных размерностях на плотностях в среднем меньше 0.5, применять лучше алгоритм Лорьера-Новикова, а для плотностей в среднем больше 0.5 - алгоритм Олемского, а алгоритм Вейсмана потерял актуальность уже на размерности 10, т.к. имел значительно низкую скорость работы на плотностях до 0.5 и дальнейшее общее сравнение с другими алгоритмами не были произведены.

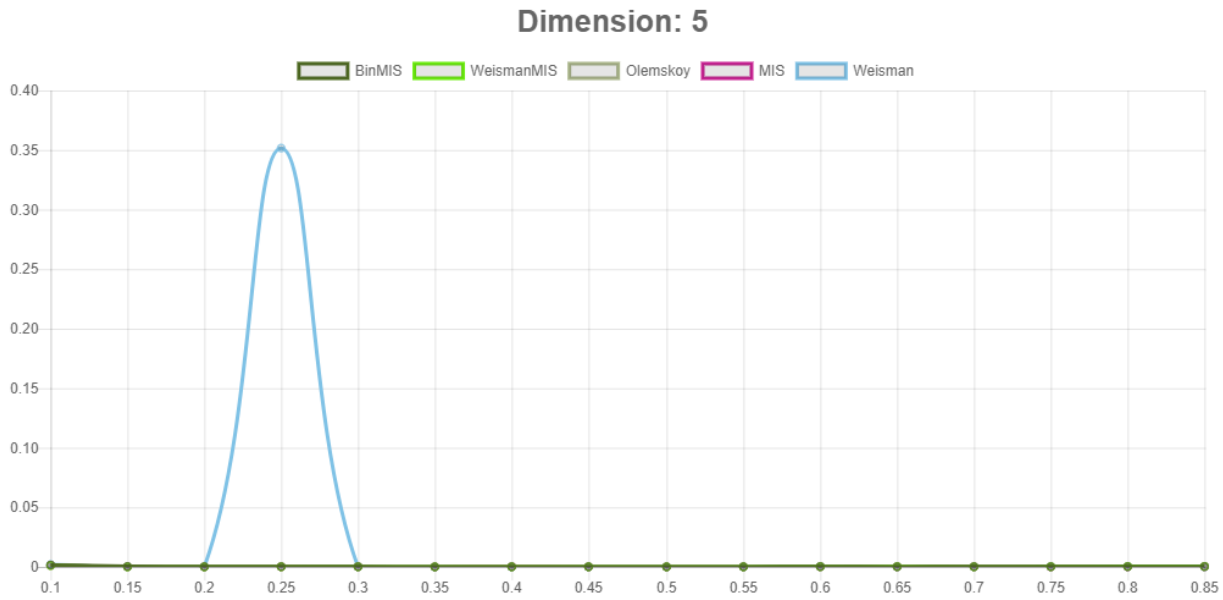

Рис. 1: График зависимости скорости работы от плотности для размерности 5

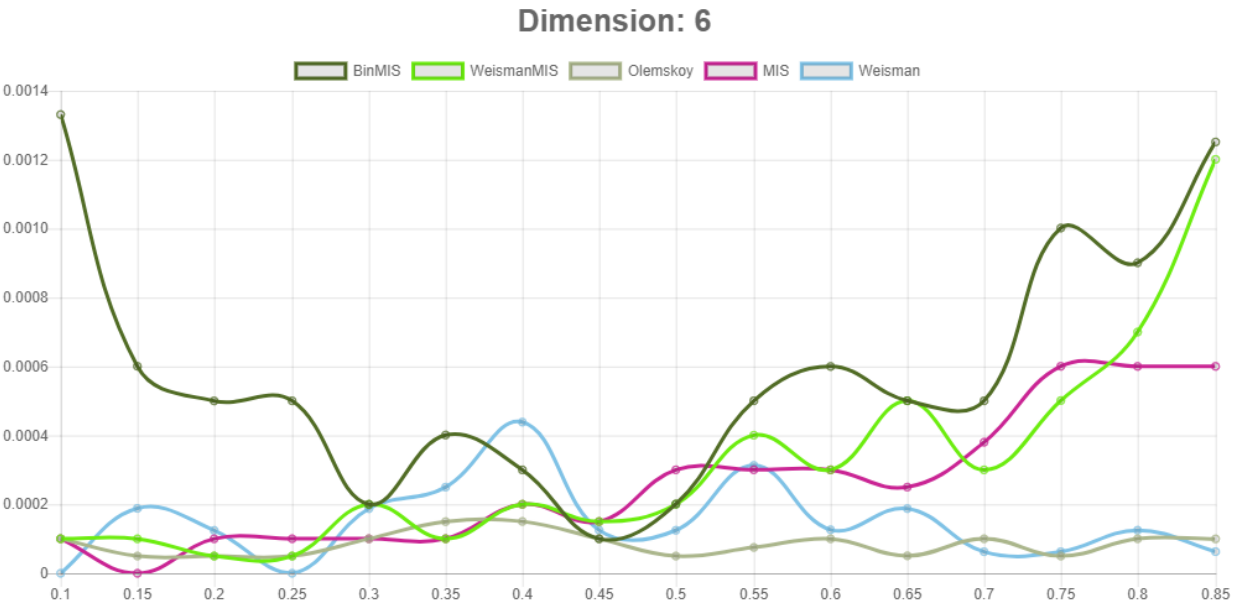

Рис. 2: График зависимости скорости работы от плотности для размерности 6

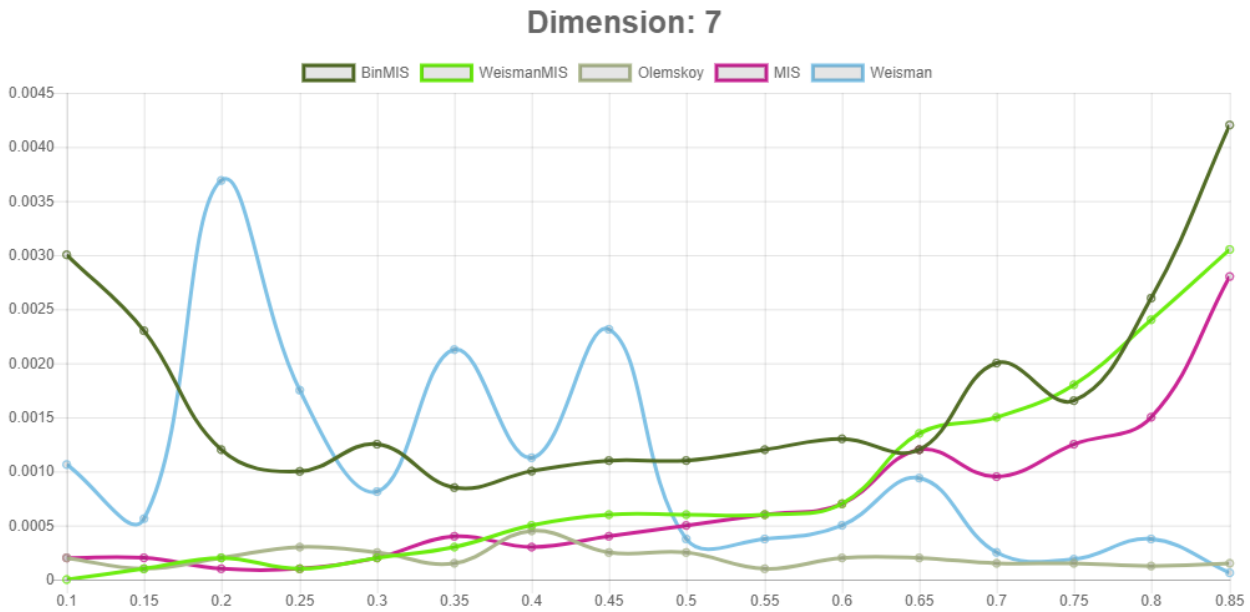

Рис. 3: График зависимости скорости работы от плотности для размерности 7

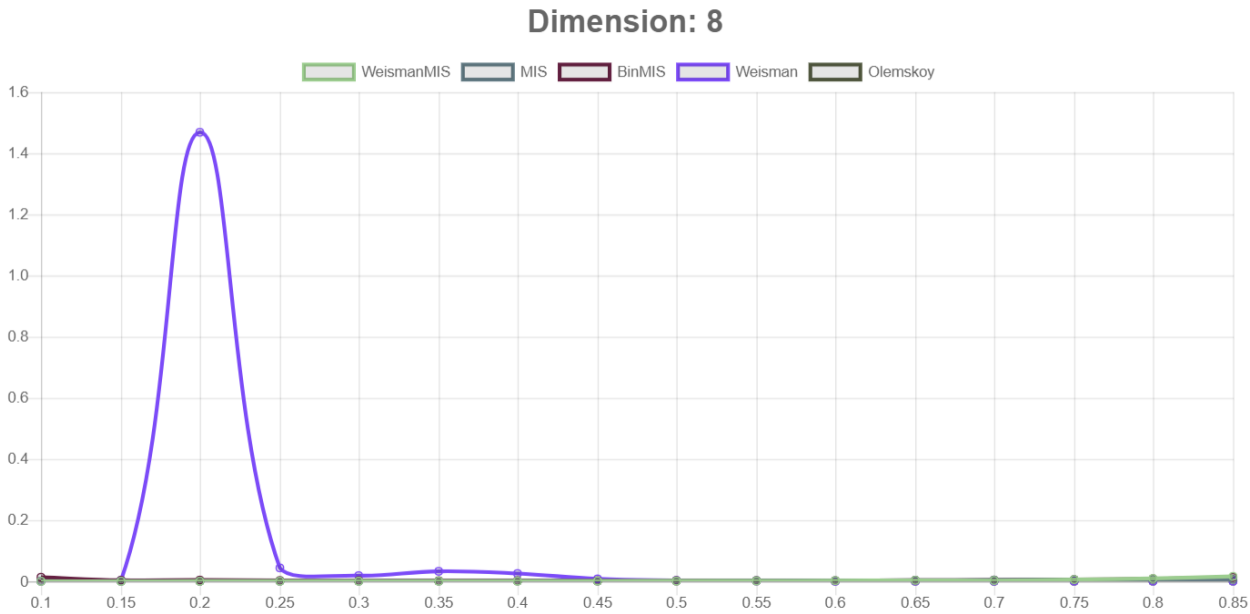

Рис. 4: График зависимости скорости работы от плотности для размерности 8

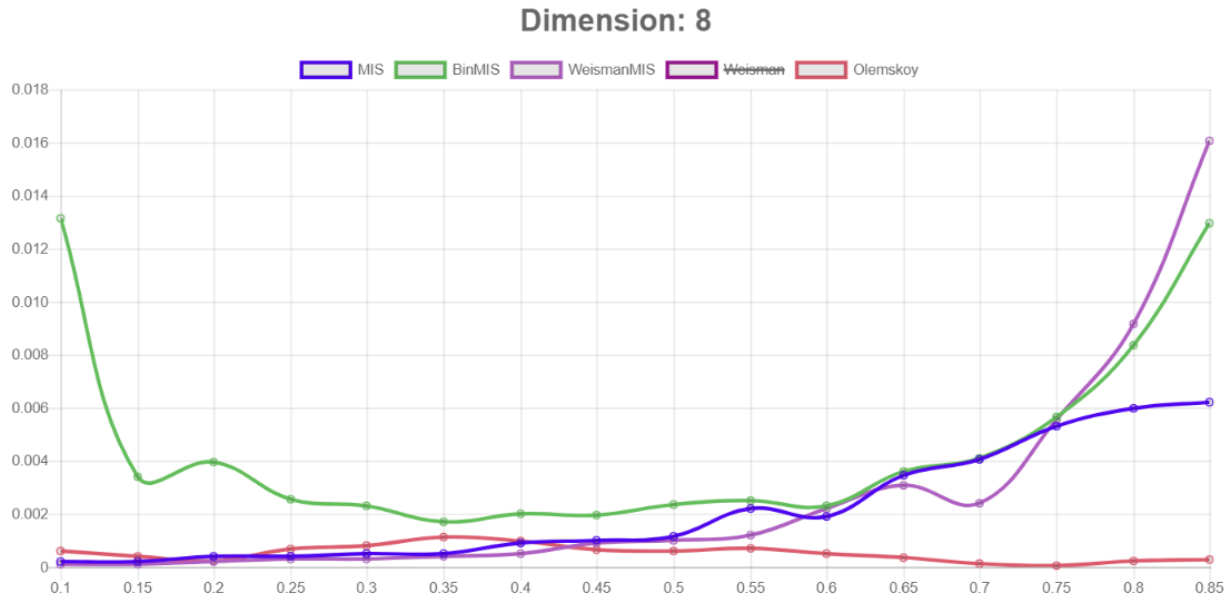

Рис. 5: График зависимости скорости работы от плотности для размерности 8 без алгоритма Вейсмана

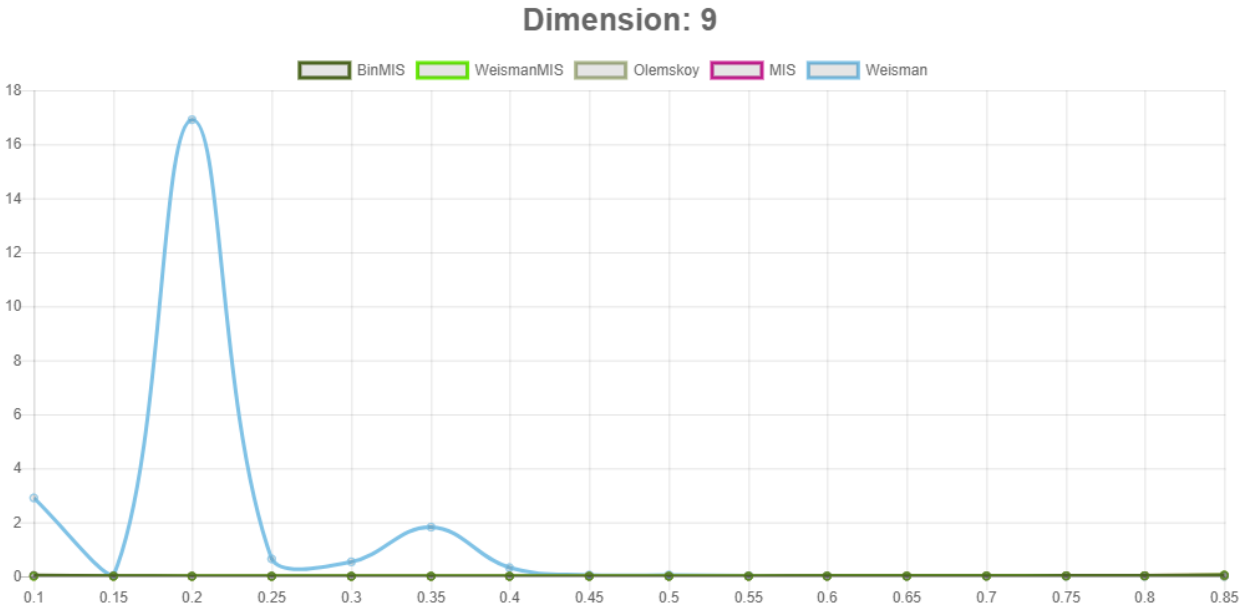

Рис. 6: График зависимости скорости работы от плотности для размерности 9

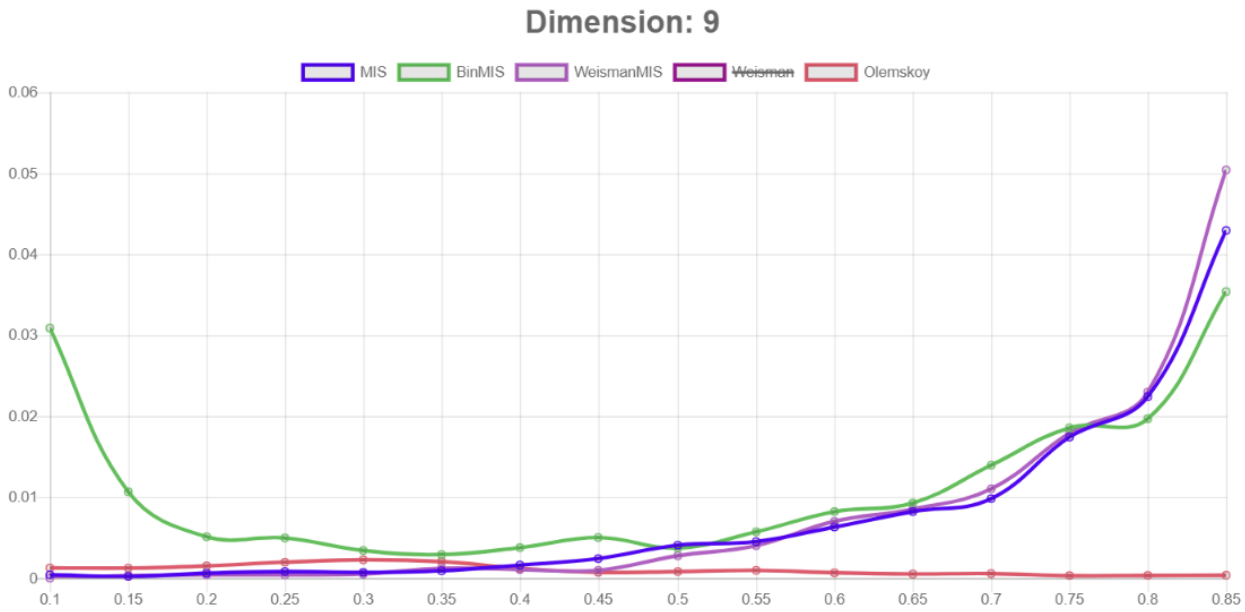

Рис. 7: График зависимости скорости работы от плотности для размерности 9 без алгоритма Вейсмана

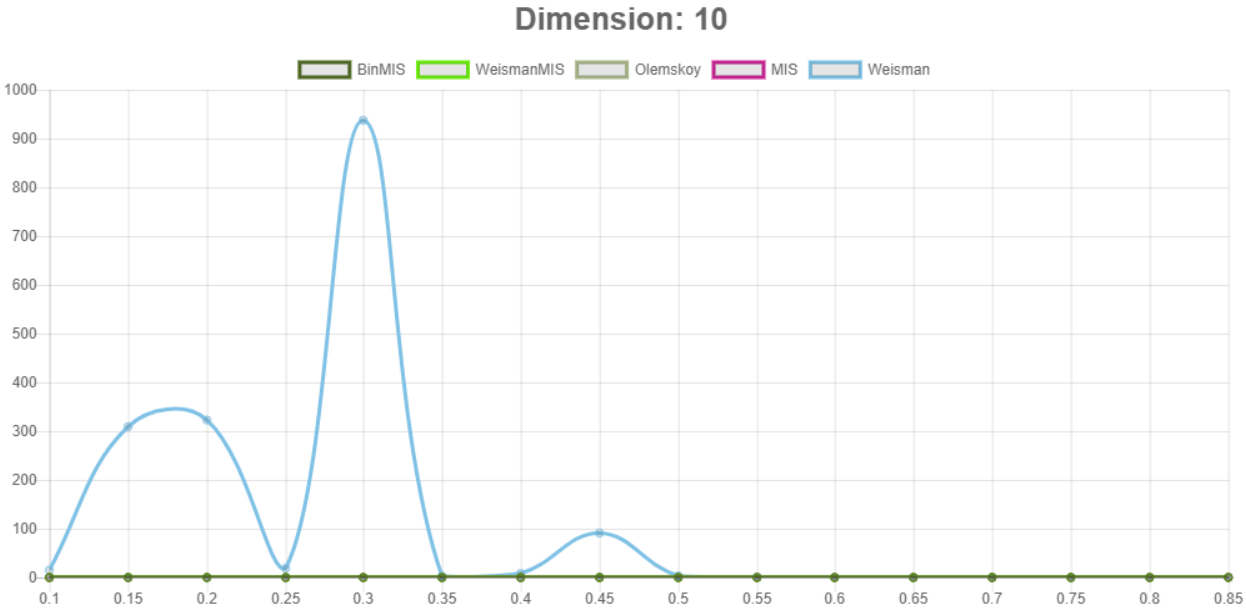

Рис. 8: График зависимости скорости работы от плотности для размерности 10

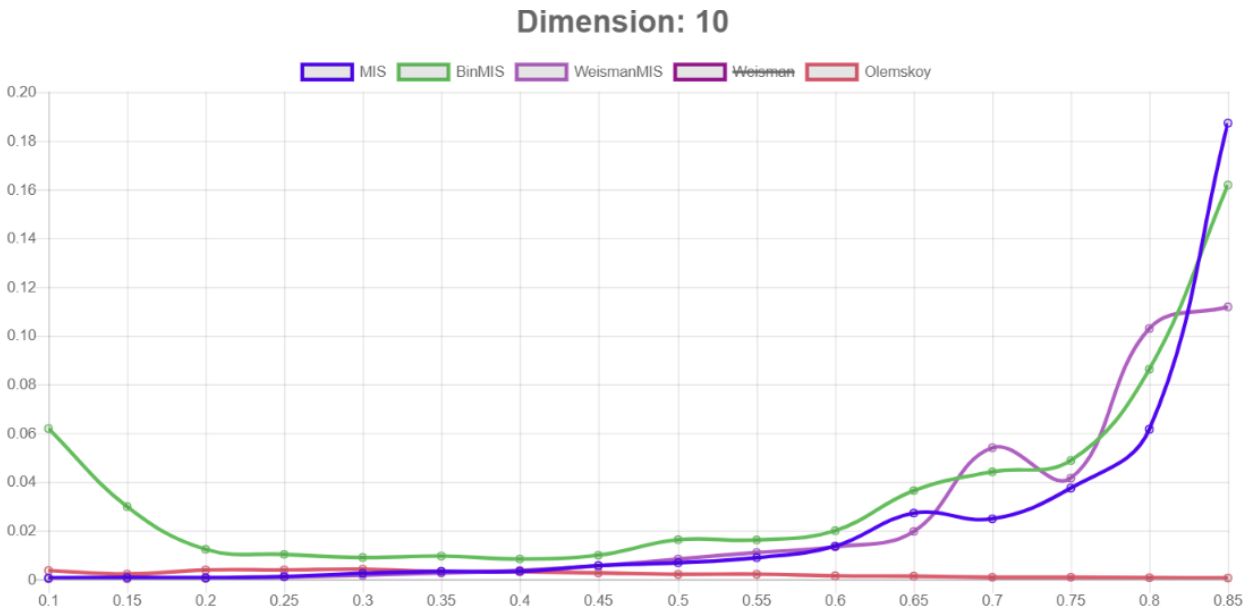

Рис. 9: График зависимости скорости работы от плотности для размерности 10 без алгоритма Вейсмана

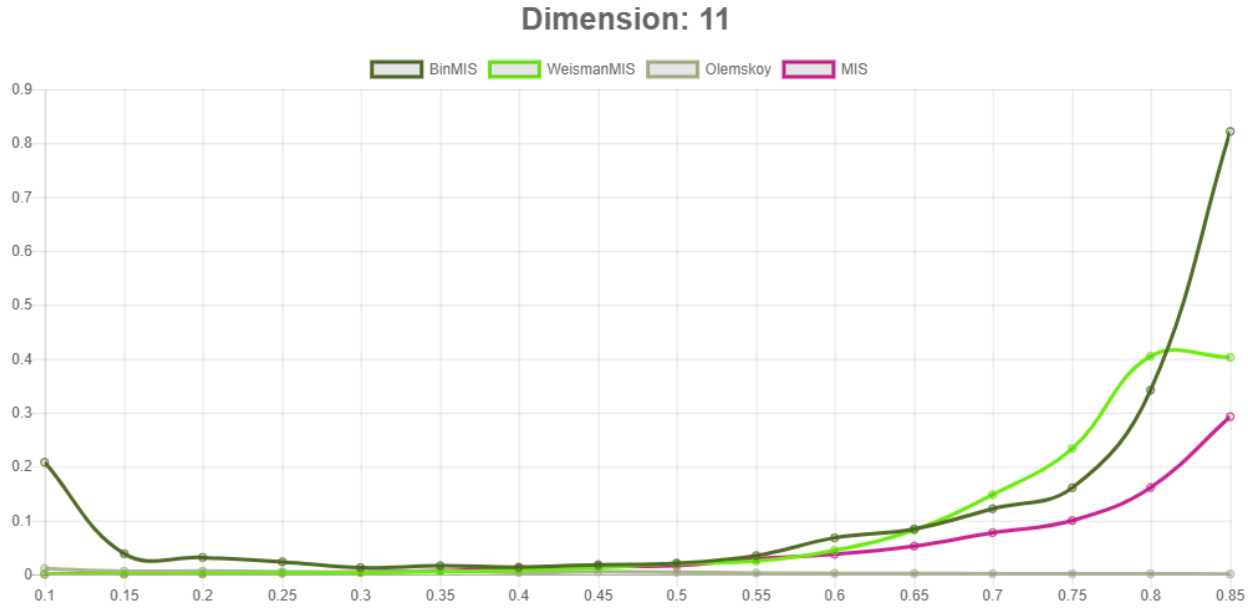

Рис. 10: График зависимости скорости работы от плотности для размерности 11

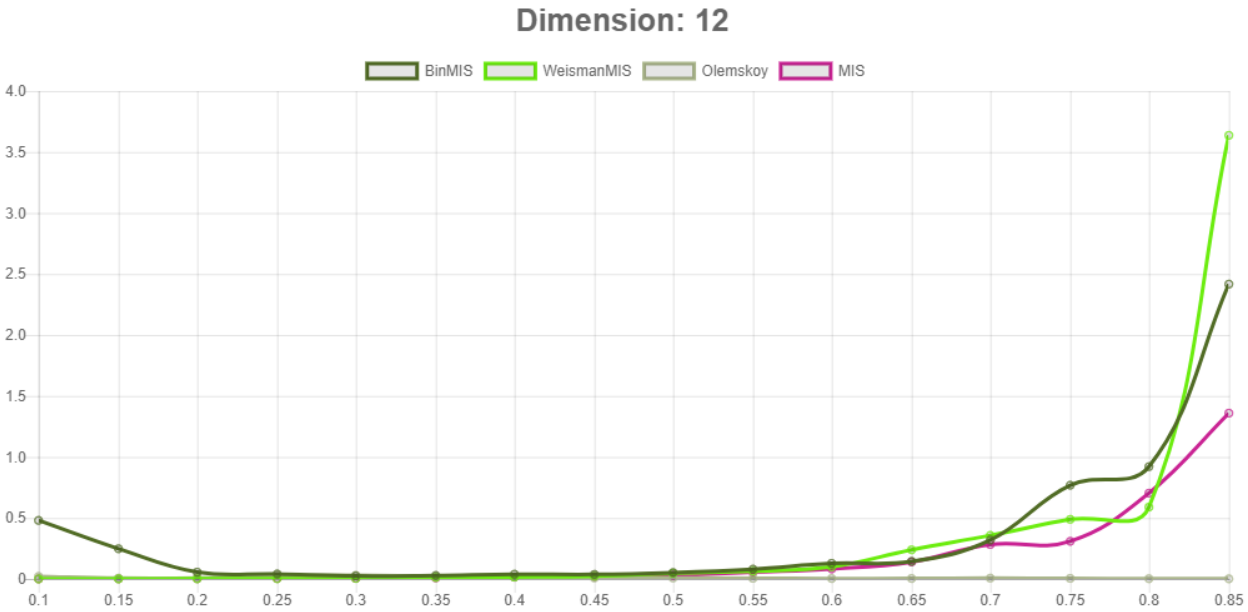

Рис. 11: График зависимости скорости работы от плотности для размерности 12

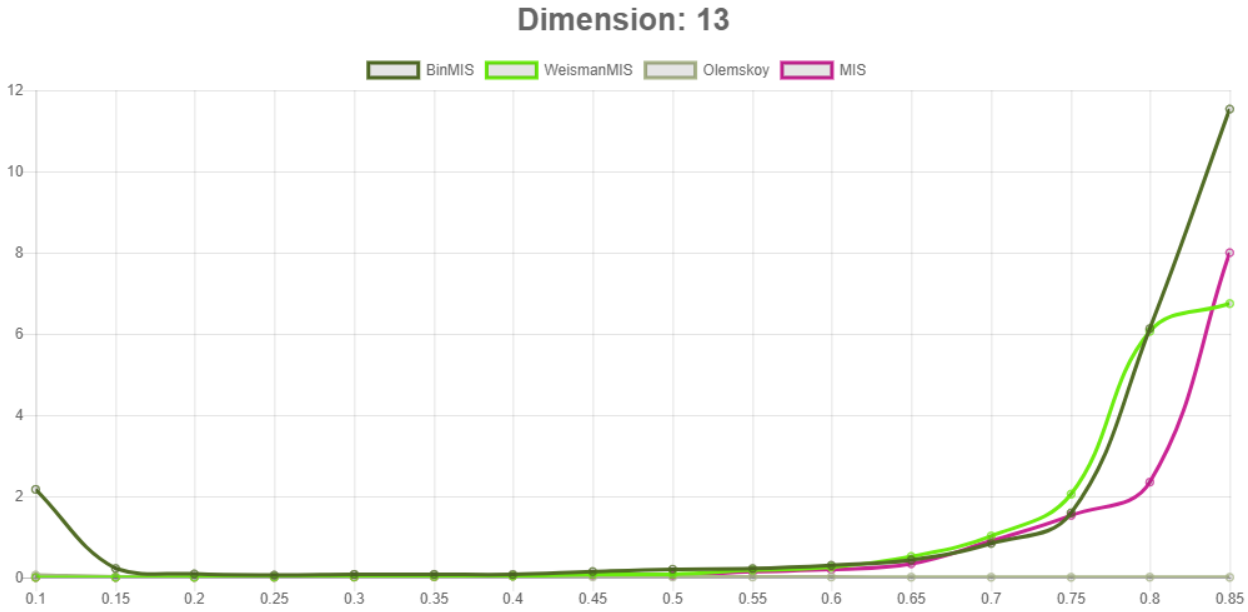

Рис. 12: График зависимости скорости работы от плотности для размерности 13

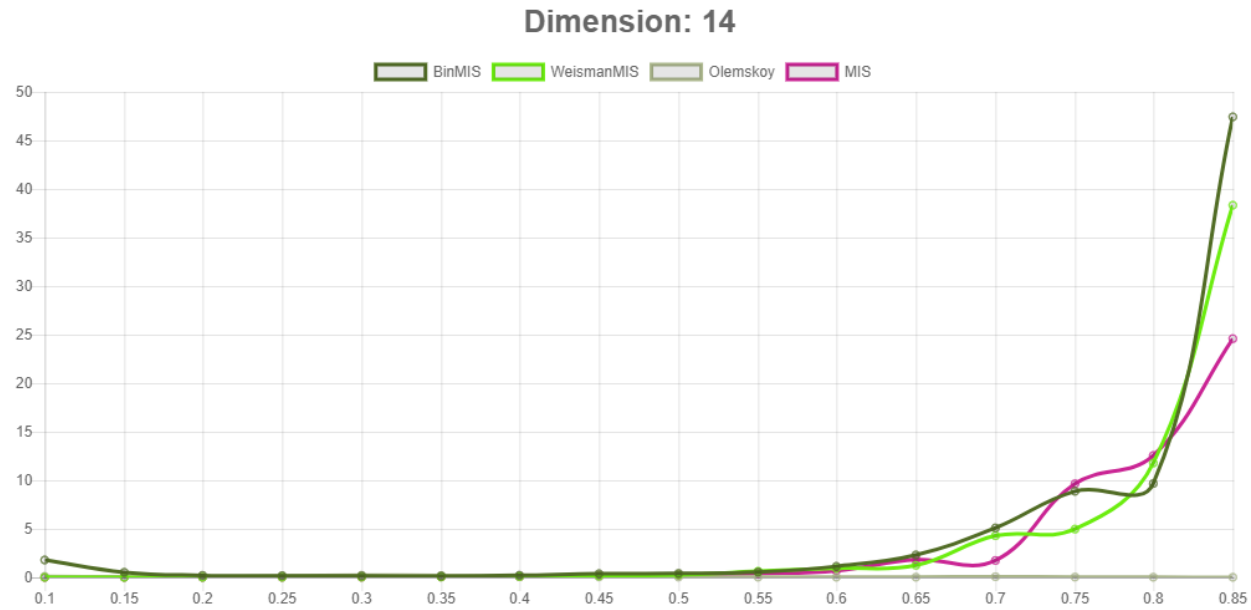

Рис. 13: График зависимости скорости работы от плотности для размерности 14

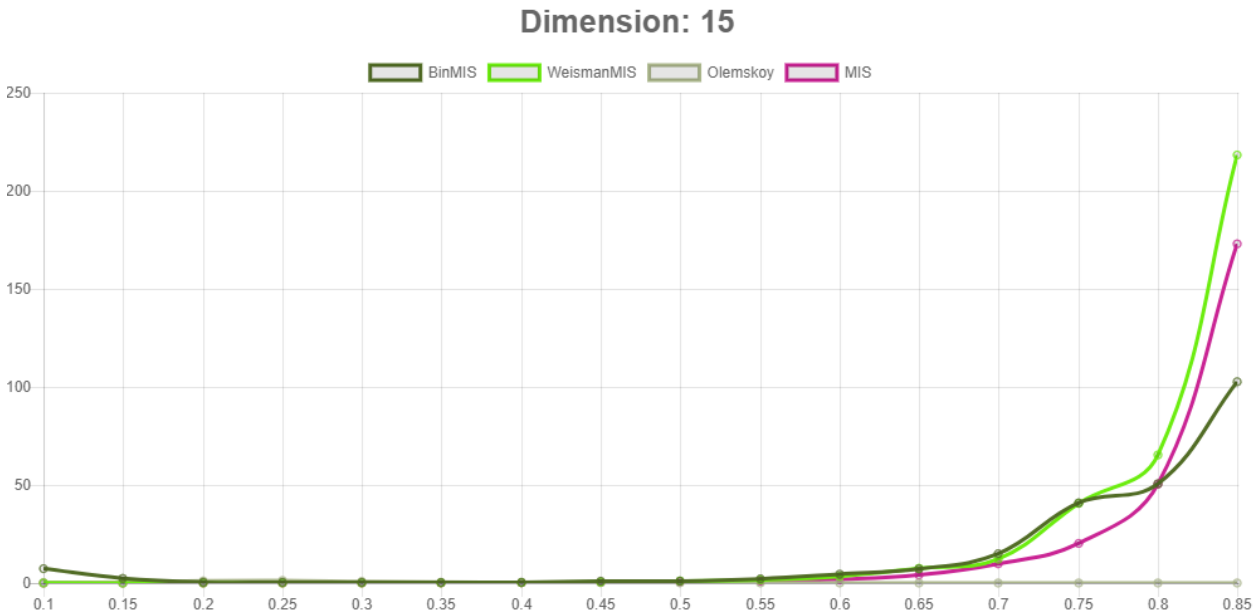

Рис. 14: График зависимости скорости работы от плотности для размерности 15

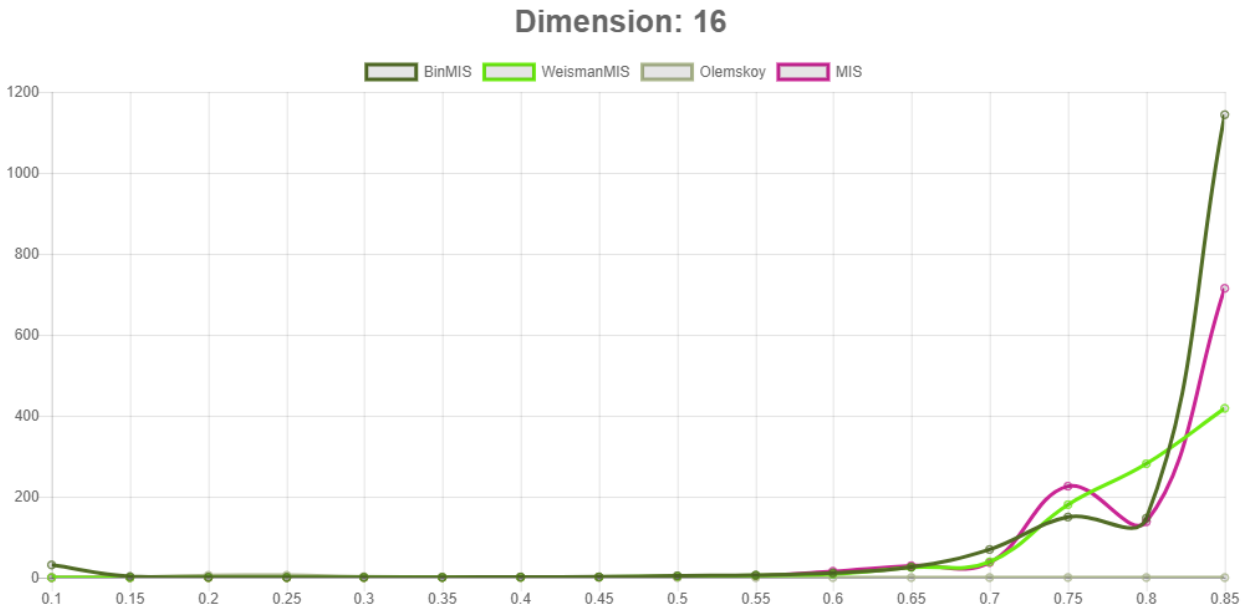

Рис. 15: График зависимости скорости работы от плотности для размерности 16

Заключение

В ходе проделанной работы были реализованы алгоритмы нахождения хроматического числа. Для сравнения алгоритмов были построены графики зависимости скорости работы алгоритма от плотности для разных размерностей. Алгоритмы сравнивались в диапазонах: для размерности - от 5 до 16, для плотности - от 0.1 до 0.85. В связи с тем, что средняя скорость работы алгоритмов на некоторых плотностях превышает сотни секунд, а также отсутствием более мощной ЭВМ, были получены результаты только до размерности 16.

Исследование показало, что на рассмотренных размерностях на плотностях в среднем меньше 0.5, стоит применять алгоритм Лорьера-Новикова, а для плотностей в среднем больше 0.5 - алгоритм Олемского, а алгоритм Вейсмана потерял актуальность уже на размерности 10, т.к. имел значительно низкую скорость работы на плотностях до 0.5 и дальнейшее общее его сравнение с другими алгоритмами не было произведено.

Список литературы

[1] Лорьер Ж.Л. Системы искусственного интеллекта — М.: Мир, 1991. 568 с.

[2] Новиков Ф. А. Дискретная математика для программистов — Изд-во: Питер , 2009. 383 с.

[3] Олемской И. В. Методы интегрирования систем структурно разделенных дифференциальных уравнений. — СПб.: Изд-во С.-Петерб. ун-та, 2009. 180 с.

[4] Зыков А.Г, Поляков В.И. Алгоритмы конструкторского проектирования ЭВМ — Санкт-Петербург.: Университет ИТМО, 2014. 136 с.

[5] Альфред В.Ахо, Джон. Э. Хопкрофт, Джеффри Д. Ульман. Структуры данных и алгоритмы — Издательский дом “Вильямс”, 2007. 400 с.

[6] Алгоритм раскраски графа с перекраской двуцветных компонент `http://bibliofond.ru/view.aspx?id=564470`

[7] Раскраска графа `http://neerc.ifmo.ru/wiki/index.php?title=Раскраска_графа`