Задача о поиске наименьшего доминирующего множества

Денис Алексеевич Лейбман

Январь 2024

Аннотация

В этой статье будут представлены простой алгоритм поиска наименьшего доминирующего множества, работающий за $\mathcal{O}(1.8021^n)$, его усовершенствованная версия, работающая за $\mathcal{O}(1.5263^n)$, а также результаты тестирования обоих алгоритмов.

1 Введение

1.1 Постановка задачи

Определение 1.1. Пусть дан граф $G = (V, E)$. *Множеством соседей* вершины $v \in V$ назовем $N(v) := \{u \in V \mid (v, u) \in E\}$. Также введем обозначение $N[v] := \{v\} \cup N(v)$.

Определение 1.2. Пусть дан граф $G = (V, E)$. Подмножество $U \subset V$ называется *доминирующим множеством* графа G, если

$$V \setminus U \subset \bigcup_{u \in U} N(u).$$

Задача ставится следующим образом: дан граф $G = (V, E)$, требуется найти наименьшее по размеру доминирующее множество в графе G.

1.2 NP-трудность

Теорема 1.1. *Соответствующая задача распознавания*

$$\mathsf{DS} = \{(G, k) \mid \textit{в графе } G \textit{ есть доминирующее множество размера } k\}$$

является **NP***-полной.*

Доказательство. Принадлежность к **NP** очевидна, так как сертификатом будет просто искомое доминирующее множество. Проверка осуществляется за полином.

Сведем к нашей задаче задачу о вершинном покрытии

$$\mathsf{VC} = \{(G, k) \mid \text{в графе } G \text{ есть вершинное покрытие размера } k\}.$$

Пусть дана пара (G, k). Построим новый граф G' следующим образом: изначально положим $G' = G$, затем для каждого ребра $(u, v) \in E(G)$ введем новую вершину w_{uv} и ребра $(u, w_{uv}), (w_{uv}, v)$. Также положим $k' = k + |I(G)|$, где $I(G)$ — это множество изолированных вершин в графе G.

Покажем, что $(G, k) \in \mathsf{VC} \Leftrightarrow (G', k') \in \mathsf{DS}$.

($\Rightarrow$) Пусть $(G, k) \in \mathsf{VC}$. Во-первых, заметим, что все изолированные вершины вынужденно попадают в доминирующее множество в новом графе G'. Теперь пусть U — вершинное покрытие размера k в графе G. Тогда $U' = U \sqcup I(G)$ — доминирующее множество в графе G': пусть $v \in V(G') \setminus U'$. Тогда, либо вершина v была добавлена искусственно в G' и из нее выходит 2 ребра и оба инцидентны вершинам из U, так как U — вершинное покрытие G, либо вершина v уже была в графе G и снова есть ребро, инцидентное вершине из U, так как $v \notin I(G)$. Итак, $|U'| = k'$ и U' — доминирующее множество в графе G'.

($\Leftarrow$) Пусть $(G', k') \in \mathsf{DS}$. Пусть U' — доминирующее множества размера k' в графе G'. Как мы знаем, $I(G) \subset U'$. Положим $W = U' \setminus I(G)$ и покажем, как получить из W вершинное покрытие G. Рассмотрим ребро $(u, v) \in E(G)$ и пусть оно не покрыто вершинами из W. Но тогда вершина $w_{uv} \in U'$, так как иначе бы она не была смежна вершинам из U' в силу того, что единственные ребра, выходящие из нее — это $(u, w_{uv}), (w_{uv}, v)$. В этом случае можем просто заменить w_{uv} в W на u (или на v). Таким образом, получили вершинное покрытие U графа G размера k.

Итак, мы показали, что $\mathsf{VC} \le_p \mathsf{DS}$. Но мы знаем, что VC — **NP**-полная задача, значит, и DS — **NP**-полная задача. ■

Следствие 1.1.1. *Задача о наименьшем доминирующем множестве является* **NP**-*трудной.*

1.3 Связь с задачей о наименьшем покрытии множества

1.3.1 Дополнительные обозначения

Пусть дано семейство $\mathcal{S}$ подмножеств конечного объемлющего множества $\mathcal{U}$. Будем говорить, что $\mathcal{S}' \subset \mathcal{S}$ — это *покрытие* $\mathcal{U}$, если

$$\mathcal{U} = \bigcup_{S \in \mathcal{S}'} S =: \mathcal{U}(\mathcal{S}').$$

Для простоты будем считать, что $\mathcal{S}$ покрывает $\mathcal{U}$. *Задача о наименьшем покрытии множества* заключается в нахождении $msc(\mathcal{S})$ – мощности наименьшего покрытия. Без ограничения общности можем считать, что $\mathcal{S}$ не содержит пустого множества, так как оно точно не входит в наименьшее покрытие. *Размерностью* k семейства $\mathcal{S}$ будем называть сумму мощностей $\mathcal{S}$ и $\mathcal{U}$:

$$k = |\mathcal{S}| + |\mathcal{U}|.$$

Пусть $R \subset \mathcal{S}$. Введем $del(\mathcal{S}, R)$ – семейство, полученное из $\mathcal{S}$ вычитанием R из каждого $S \in \mathcal{S}$, оставляя только непустые множества:

$$del(\mathcal{S}, R) := \{S' \neq \varnothing \mid S' = S \setminus R, S \in \mathcal{S}\}.$$

1.3.2 Сведение

Пусть дан граф $G = (V, E)$. Тогда задача о наименьшем доминирующем множестве легко переформулируется в задачу о наименьшем покрытии множества, если положить $\mathcal{S} = \{N[v] \mid v \in V\}$. Тогда можно заметить, что размерность задачи равна $k = 2n$, где $n = |V|$.

2 Базовый алгоритм

2.1 Алгоритм с полиномиальной памятью

Здесь будет представлен базовый рекурсивный алгоритм поиска наименьшего покрытия множества с полиномиальной памятью, описанный в статье [2], работающий за $\mathcal{O}(1.3803^k)$, где k – это размерность задачи.

Лемма 2.1. *Пусть поставлена задача поиска $msc(\mathcal{S})$. Тогда верны следующие утверждения:*

(1) Если есть множества $S, R \in \mathcal{S}$, такие что $S \subset R$, то существует наименьшее покрытие, не содержащее S, то есть:

$$msc(\mathcal{S}) = msc(\mathcal{S} \setminus S).$$

(2) Если есть такой элемент $s \in \mathcal{U}(\mathcal{S})$, что s принадлежит единственному $S \in \mathcal{S}$, то любое наименьшее покрытие должно содержать S, то есть:

$$msc(\mathcal{S}) = 1 + msc(del(\mathcal{S}, S)).$$

(3) Для всех оставшихся $S \in \mathcal{S}$ верно следующее равенство:

$$msc(\mathcal{S}) = \min\{msc(\mathcal{S} \setminus \{S\}), 1 + msc(del(\mathcal{S}, S))\}.$$

Замечание 1. Сразу отметим, что любое множество мощности 1 обладает одним из свойств (1) или (2) из Леммы 2.1.

Algorithm 1 Алгоритм MSC1

function MSC1($\mathcal{S}$)
  if $|\mathcal{S}| = 0$ **then**
    return 0
  if $\exists S, R \in \mathcal{S} : S \subset R$ **then**
    return MSC1($\mathcal{S} \setminus S$)
  if $\exists s \in \mathcal{U}(\mathcal{S}) \exists$ единственное $S \in \mathcal{S} : s \in S$ **then**
    return 1 + MSC1($del(\mathcal{S}, S)$)
  $S \leftarrow \arg\max_{\tilde{S} \in \mathcal{S}} |\tilde{S}|$
  return $\min\{\text{MSC1}(\mathcal{S} \setminus \{S\}), 1 + \text{MSC1}(del(\mathcal{S}, S))\}$

Базовый алгоритм $MSC1$, использующий Лемму 2.1, представлен выше. Сначала он исключает тривиальный случай, когда $|\mathcal{S}| = 0$, затем проверяет, выполнены ли свойства (1) или (2) из Леммы 2.1. В конце он использует свойство (3) к множеству с наибольшей мощностью.

Теорема 2.2. *Алгоритм $MSC1$ решает задачу о наименьшем покрытии множества за время $\mathcal{O}(1.3803^k)$, где k – это размерность задачи.*

Доказательство. Корректность алгоритма следует из Леммы 2.1.

Для доказательства времени работы введем $N_h(k)$ – количество подзадач размерности h, решенных в ходе решения задачи размерности k. Считаем, что $N_h(k) = 0$, если $h > k$ (так как подзадачи могут быть только меньшей размерности, чем исходная задача), а также считаем, что $N_k(k) = 1$ (считаем исходную задачу свой подзадачей).

Теперь рассмотрим случай $h < k$ (из чего сразу следует, что $|\mathcal{S}| > 0$). Если выполнено одно из условий из строчек 4 или 6, то алгоритм создает единственную подзадачу размерности не более $k - 1$, то есть верно:

$$N_h(k) \leq N_h(k-1).$$

Иначе алгоритм выбирает самое мощное множество $S \in \mathcal{S}$, причем $|S| > 1$. В таком случае он создает 2 подзадачи $\mathcal{S}_{out} = \mathcal{S} \setminus S$ и $\mathcal{S}_{in} = del(\mathcal{S}, S)$. Размерность $\mathcal{S}_{out}$ равна $k - 1$ (из $\mathcal{S}$ удалено одно множество), а также если $|S| \geq 3$, то размерность $\mathcal{S}_{in}$ не более $k - 4$ (из $\mathcal{U}$ удалено хотя бы 3 элемента и удалено одно множество из $\mathcal{S}$). Тогда верно:

$$N_h(k) \leq N_h(k-1) + N_h(k-4).$$

В случае, когда $|S| = 2, \forall S \in \mathcal{S}$, размерность $\mathcal{S}_{in}$ равна $k - 3$, к тому же найдется S' мощности 1 в полученной подзадаче (потому что не было выполнено свойство 2 из Леммы 2.1). Значит, и в этом случае:

$$N_h(k) \leq N_h(k-1) + N_h(k-3-1) = N_h(k-1) + N_h(k-4).$$

Тогда, если ограничить $N_h(k)$ линейной рекуррентой, то получится оценка $N_h(k) \leq c^{k-h}$, где c – это единственный положительный корень характеристического уравнения $\lambda^4 - \lambda^3 - 1 = 0$, $c = 1.3802... < 1.3803$. Тогда получаем, что:

$$N(k) = \sum_{h=0}^{k} N_h(k) \leq \sum_{h=0}^{k} c^{k-h} = \mathcal{O}(c^k).$$

Стоимость решения задачи размерности $h \leq k$ (не включая стоимость подзадач) ограничена полиномом $poly(k)$. Тогда получаем, что время работы алгоритма равно $\mathcal{O}(c^k poly(k)) = \mathcal{O}(1.3803^k)$. ■

Следствие 2.2.1. *Существует алгоритм с полиномиальной памятью, решающий задачу о наименьшем доминирующем множестве за $\mathcal{O}(1.9053^n)$, где n – количество вершин в графе.*

2.2 Алгоритм с экспоненциальной памятью

Здесь будет получена новая версия алгоритма поиска наименьшего покрытия множества, использующая экспоненциальную память и работающая за время $\mathcal{O}(1.3424^k)$.

Модифицируем алгоритм $MSC1$ следующим образом: для каждой подзадачи будем сохранять ответ в некоторой структуре данных, а при каждом рекурсивном запуске будем проверять, не была ли уже решена подзадача. Таким образом, каждая подзадача будет решена не более 1 раза. Структуру данных можно реализовать таким образом, что время ответа на запрос будет логарифмическим от количества хранимых данных (например, можно рассмотреть некоторое сбалансированное дерево поиска). Таким образом, получим новый алгоритм $MSC2$.

Теорема 2.3. *Алгоритм $MSC2$ решает задачу о наименьшем покрытии множества за время $\mathcal{O}(1.3424^k)$, где k – это размерность задачи.*

Доказательство. Корректность алгоритма следует из Леммы 2.1.

Для доказательства времени работы воспользуемся терминологией, введенной ранее при доказательстве Теоремы 2.2. Там была получена оценка $N_h(k) \leq c^{k-h}$, где $h \in \{0, 1, \ldots, k\}$ и $c = 1.3802... < 1.3803$ – это единственный положительный корень характеристического уравнения $\lambda^4 - \lambda^3 - 1 = 0$. Теперь заметим, что каждая подзадача $\mathcal{S}'$ была получена удалением некоторых множеств из $\mathcal{S}$ и некоторых элементов из $\mathcal{U}$. То есть:

$$\mathcal{S}' = del(\mathcal{S} \setminus \mathcal{S}^*, R^*)$$

для некоторых $\mathcal{S}^* \subset \mathcal{S}$ и $R^* \subset \mathcal{U}$. Тогда в силу того, что каждая подзадача решается не более 1 раза, получаем оценку $N_h(k) \leq C_k^h$. Рассмотрим число h' – наибольшее число из $\{0, 1, \ldots, [k/2]\}$, такое что $C_k^{h'} < c^{k-h'}$. Тогда можем построить следующую оценку:

$$N(k) = \sum_{h=0}^{k} N_h(k) \leq \sum_{h=0}^{h'} C_k^h + \sum_{h=h'+1}^{k} c^{k-h} = \mathcal{O}(c^{k-h'}).$$

Для нахождения такого h' достаточно найти $a \in (0, 1/2]$, такое что

$$c^{1-a} = \frac{1}{a^a(1-a)^{1-a}},$$

так как

$$C_k^{[ak]} = \left(\frac{1}{a^a(1-a)^{1-a}} + o(1) \right)^k.$$

Потребовав логарифмическое время запроса от количества подзадач, получаем полиномиальное время от k. Отсюда снова стоимость решения задачи размерности $h \leq k$ (не включая стоимость подзадач) ограничена полиномом $poly(k)$. Получаем $\mathcal{O}(c^{(1-a)k} poly(k)) = \mathcal{O}(1.3424^k)$ ■

Следствие 2.3.1. *Существует алгоритм с экспоненциальной памятью, решающий задачу о наименьшем доминирующем множестве за $\mathcal{O}(1.8021^n)$, где n – количество вершин в графе.*

3 Улучшенный алгоритм

В данном разделе будет представлен улучшенный алгоритм поиска наименьшего покрытия множества, описанный в статье [1], использующий дополнительное отсечение.

Algorithm 2 Алгоритм MSC3

1: **function** MSC3($\mathcal{S}$)
2:     **if** $|\mathcal{S}| = 0$ **then**
3:         **return** 0
4:     **if** $\exists S, R \in \mathcal{S} : S \subset R$ **then**
5:         **return** MSC3($\mathcal{S} \setminus S$)
6:     **if** $\exists s \in \mathcal{U}(\mathcal{S})\exists$ единственное $S \in \mathcal{S} : s \in S$ **then**
7:         **return** 1 + MSC3($del(\mathcal{S}, S)$)
8:     $S \leftarrow \arg\max_{\tilde{S} \in \mathcal{S}} |\tilde{S}|$
9:     **if** |S| = 2 **then**
10:         **return** PolyMSC(S)
11:     **return** $\min\{\text{MSC3}(\mathcal{S} \setminus \{S\}), 1 + \text{MSC3}(del(\mathcal{S}, S))\}$

На 9 строке алгоритма $MSC3$ используется дополнительное отсечение, рассчитанное на случай, когда мощность всех множеств $S \in \mathcal{S}$ равна 2. В этом случае запускается некоторый алгоритм $PolyMSC$ поиска наименьшего покрытия множества, работающий за полиномиальное время и на полиномиальной памяти. Например, этот алгоритм может взять наибольшее паросочетание и жадно добавить в него недостающие ребра.

С помощью более подробного анализа, использующего нестандартные меры, можно доказать, что алгоритм $MSC3$ работает за время $\mathcal{O}(2^{0.305k})$, где k – это размерность задачи. Тогда получаем алгоритм решения задачи о наименьшем доминирующем множестве, работающий за время $\mathcal{O}(2^{0.61n}) = \mathcal{O}(1.5263^n)$, где n – количество вершин в графе.

4 Результаты тестирования алгоритмов

В этом разделе будут представлены результаты тестирования алгоритма $MSC1$ и алгоритма $MSC3$, где в качестве $PolyMSC$ был реализован алгоритм Эдмондса сжатия цветков.

Оба алгоритмы были протестированы на всех графах с не более, чем 8 вершинами путем сравнения с выводом тривиального алгоритма, перебирающего всевозможные варианты доминирующего множества.

Представленные далее результаты получены на выборке из 100 тестов из каждого распределения $G(n, p)$.

4.1 Тестирование базовой версии алгоритма

Тип	Среднее	Медиана	0.25-квантиль	0.75-квантиль
$G(20, 1/10)$	2.58	2.0	2.00	3.00
$G(20, 1/5)$	7.17	4.0	2.00	8.00
$G(20, 3/10)$	20.95	20.0	11.00	31.00
$G(20, 1/2)$	21.09	20.5	16.00	25.00
$G(20, 7/10)$	10.45	10.5	8.00	13.00
$G(40, 1/10)$	146.57	46.0	21.00	147.25
$G(40, 1/5)$	3883.49	3821.5	2224.00	4926.25
$G(40, 1/2)$	1040.01	1046.5	871.50	1187.00
$G(40, 7/10)$	134.08	128.5	111.75	155.25
$G(50, 1/10)$	2933.55	1054.0	183.75	3233.00
$G(50, 1/5)$	64304.58	59604.5	43050.00	80416.50
$G(50, 3/10)$	42080.48	40946.5	34079.50	48281.50
$G(50, 1/2)$	4549.93	4407.0	3928.75	5179.50
$G(50, 7/10)$	382.06	377.0	324.75	438.50
$G(60, 1/10)$	71418.37	40161.5	11896.25	93271.75
$G(60, 1/5)$	842583.11	777196.5	632192.75	976067.50
$G(60, 3/10)$	350394.95	343992.0	287411.25	383040.00
$G(60, 1/2)$	18046.69	17616.0	15607.00	20408.25
$G(60, 7/10)$	1024.64	974.5	889.75	1139.25
$G(65, 1/10)$	458797.60	344066.0	160127.00	576648.75
$G(65, 1/5)$	2869026.31	2603538.0	2158357.75	3416341.75
$G(65, 3/10)$	864690.66	850250.0	733584.50	952602.50
$G(65, 1/2)$	30750.04	30167.5	26778.25	33781.50
$G(65, 7/10)$	1494.42	1449.0	1327.00	1658.75

4.1.1 Распределение времени работы

Приведем также гистограммы плотности времени работы для $n = 65$ и разных p.

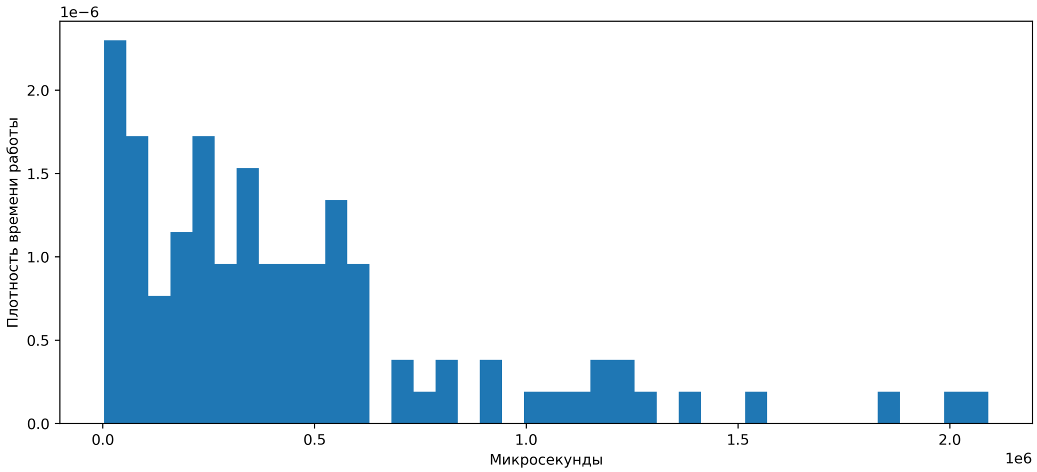

Рис. 1: Плотность для $G(65, 1/10)$

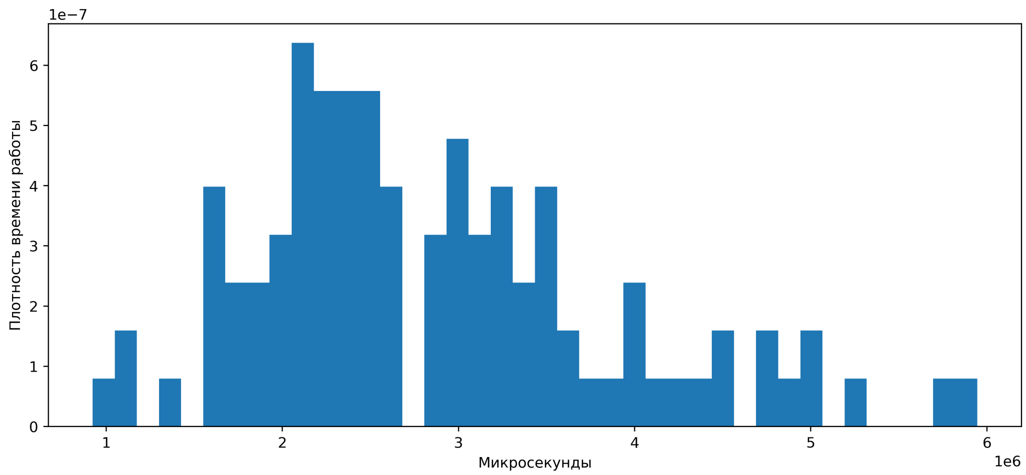

Рис. 2: Плотность для $G(65, 1/5)$

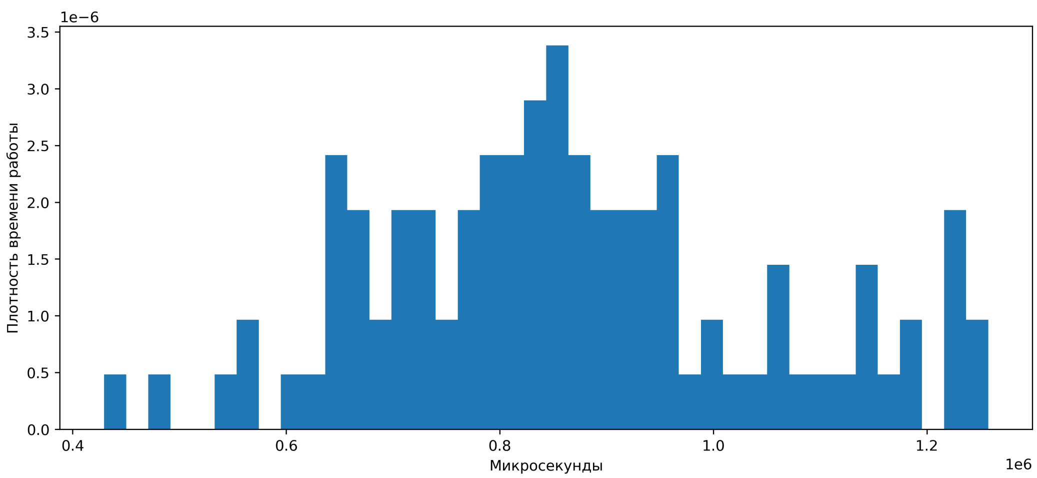

Рис. 3: Плотность для $G(65, 3/10)$

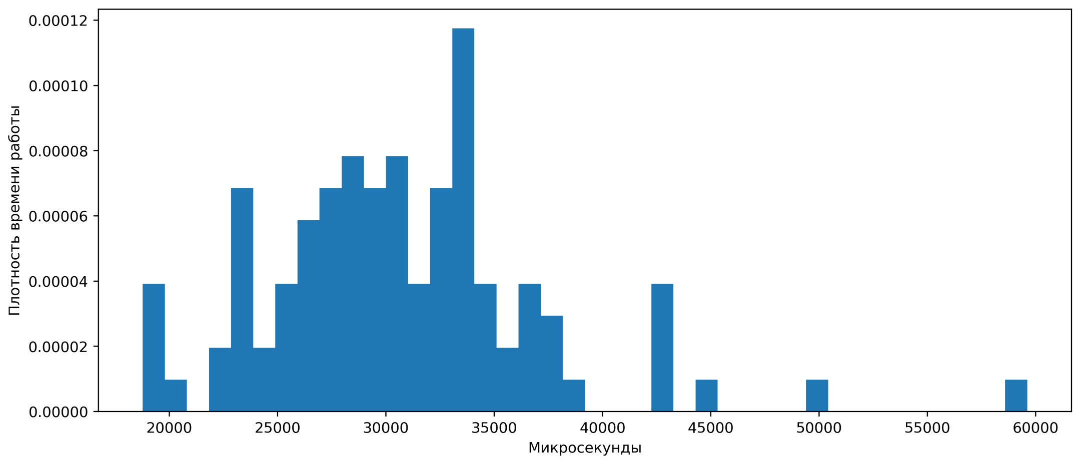

Рис. 4: Плотность для $G(65, 1/2)$

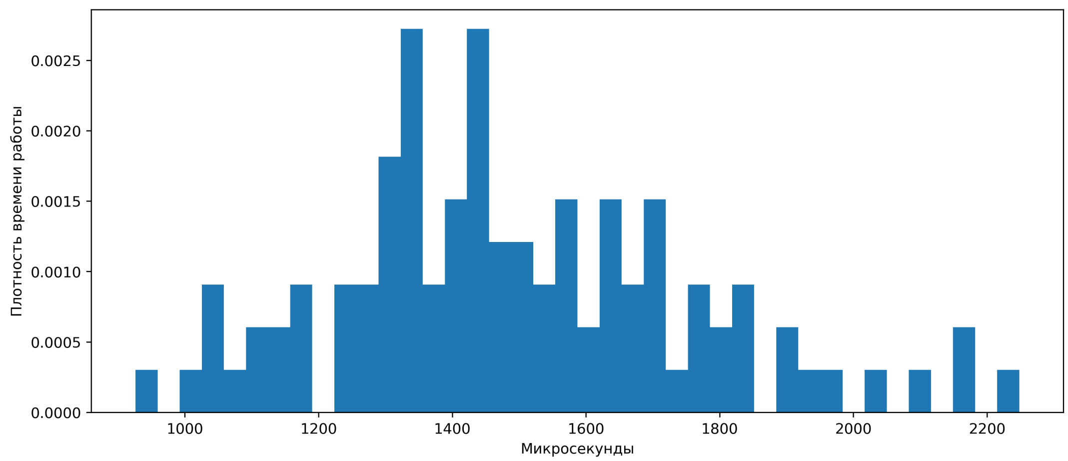

Рис. 5: Плотность для $G(65, 7/10)$

4.2 Тестирование улучшенной версии алгоритма

Тип	Среднее	Медиана	0.25-квантиль	0.75-квантиль
$G(20, 1/10)$	3.44	2.0	2.00	3.00
$G(20, 1/5)$	14.10	7.0	3.00	19.00
$G(20, 3/10)$	51.06	38.0	19.75	68.00
$G(20, 1/2)$	41.90	37.0	25.00	53.75
$G(20, 7/10)$	10.07	10.0	8.00	12.00
$G(40, 1/10)$	154.60	64.0	27.75	171.25
$G(40, 1/5)$	8146.74	7069.5	3775.50	11108.00
$G(40, 3/10)$	11196.26	10919.0	8504.25	13242.50
$G(40, 1/2)$	2152.94	2024.0	1729.25	2470.50
$G(40, 7/10)$	186.21	173.5	150.00	209.75
$G(50, 1/10)$	5151.67	2050.5	589.75	7332.25
$G(50, 1/5)$	155839.38	149146.5	103251.25	195761.00
$G(50, 3/10)$	107816.36	109894.0	86286.75	123666.75
$G(50, 1/2)$	8320.48	7953.5	6951.50	9273.75
$G(50, 7/10)$	436.16	432.0	361.50	487.50
$G(60, 1/10)$	172691.16	58640.0	25789.50	133658.50
$G(60, 1/5)$	2107943.48	1975976.0	1568499.50	2691828.50
$G(60, 3/10)$	835474.51	834159.5	680734.50	951240.00
$G(60, 1/2)$	33657.59	33152.5	27504.25	39111.75
$G(60, 7/10)$	1128.78	1063.0	907.00	1241.25
$G(65, 1/10)$	640063.98	344424.5	126929.50	893258.75

4.3 Выводы

По полученным данным можно сделать вывод, что когда p слишком близко к 0 или слишком близко к 1 алгоритм начинает работать в среднем очень быстро, а наибольшее время работы всегда достигается на $p = \frac{1}{5}$ среди выбранных значений p. Объяснить это можно тем, что когда p мало или наоборот велико, все множества в задаче наименьшего покрытия малы по размеру или наоборот велики, а в таких случаях глубина рекурсии сильно уменьшается, так как либо, часто выполняются случаи (1) и (2) из Леммы 2.1, либо применение функции del сильно уменьшает размерность задачи, так как вычитается множество большой мощности.

Размер графов был взят не превосходящим 65, так как при больших n время работы тестов превышало грани разумного.

Относительно сравнения времени работы базовой и улучшенной версии алгоритма можно сказать, что улучшенная версия работает стабильно медленнее базовой. Скорее всего, это связано с тем, что действительно весомое улучшение асимптотики может быть заметно только при очень больших n, на которых протестировать алгоритм не удастся за разумное время. Также константа, которую привносит алгоритм Эдмондса, может быть довольно большой, что не дает увидеть улучшения времени на небольших тестах.

Список литературы

[1] Fedor V. Fomin, Fabrizio Grandoni, and Dieter Kratsch. A measure & conquer approach for the analysis of exact algorithms. *J. ACM*, 56(5), aug 2009.

[2] Fabrizio Grandoni. A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms*, 4(2):209–214, 2006.