Unweighted Set Cover

Равилов Игорь Б05-325

Май 2025

Содержание

Аннотация

В данной работе исследуется NP-полная задача покрытия множества. Задача формулируется как поиск минимального подмножества $\{S_i\}$, покрывающего элементы M. Актуальность проблемы обусловлена применением методов покрытия во множестве областей, таких как логистика, оптимизация сетей, распределение портфеля (финансы), планирование ресурсов, машинное обучение и хеширование баз данных. Основное внимание уделено следующим аспектам:

- Доказательству NP-полноты задачи
- Описанию и анализу жадного алгоритма с $O(\ln|M|)$ приближением
- Применению методов линейного программирования для получения k-приближения при ограничении частоты вхождения каждого элемента не более чем в k подмножеств и анализу полученного алгоритма

1 Формальная постановка задачи и предварительные сведения

Определения задач SETCOVER и VERTEXCOVER

Оптимизационная версия SETCOVER. Даны конечное множество M и набор его подмножеств $\mathcal{S} = S_1, S_2, \ldots, S_m$, где $S_i \subseteq M$ для каждого i. Требуется найти такое $\mathcal{C} \subseteq \mathcal{S}$ минимальной мощности, что $\bigcup_{S \in \mathcal{C}} S = M$. Оптимальный размер обозначается $\mathrm{OPT}_{\mathsf{SETCOVER}}(M, \mathcal{S})$ (далее просто OPT).

Поисковая версия SETCOVER. Задан параметр $k \in \mathbb{N}$. Нужно определить, существует ли $\mathcal{C} \subseteq \mathcal{S}$ такое, что $|\mathcal{C}| \leq k$ и $\bigcup_{S \in \mathcal{C}} S = M$. Язык всех пар $\langle (M, \mathcal{S}), k \rangle$, для которых ответ существует называется SETCOVER.

Поисковая версия VERTEXCOVER. Дан неориентированный граф $G = (V, E)$ и число $k \in \mathbb{N}$. Нужно определить, существует ли множество вершин $C \subseteq V$ такое, что $|C| \leq k$ и каждое ребро из E инцидентно хотя бы одной вершине из C. Язык всех пар $\langle G, k \rangle$, для которых ответ существует, называется VERTEXCOVER.

Предварительные сведения

В дальнейшем будем ссылаться на следующие результаты из работы Д.В. Мусатова[4]:

Теорема 1.1 (Теорема 3.11, Кука-Левина)**.** *Язык* SAT **NP**-*полон.*

Теорема 1.2 (Теорема 3.14)**.** *Язык* INDSET **NP**-*полон.*

Утверждение 1.3 (Утв. 3.13)**.** *Языки* CLIQUE*,* INDSET *и* VERTEXCOVER *полиномиально сводятся друг к другу.*

И на теорему Кенига из теории графов:

Теорема 1.4 (Теорема Кенига)**.** *В произвольном двудольном графе мощность максимального паросочетания равна мощности минимального вершинного покрытия.*

2 Доказательство NP-полноты задачи SETCOVER

Лемма 2.1. *Задача* SETCOVER *принадлежит классу* **NP**.

Доказательство. Сертификатом является набор индексов множеств из $\mathcal{C} \subseteq \mathcal{S}$. Нужно проверить, что каждый элемент $x \in M$ содержится хотя бы в одном множестве $S_i \in \mathcal{C}$. Это проверяется за $O(|M| + \sum_{S \in \mathcal{C}} |S|)$ времени, то есть полиномиально от размера входа. □

Теорема 2.2 (NP-полнота SETCOVER)**.** *Задача* SETCOVER **NP**-*полна.*

Доказательство. Докажем, что VERTEXCOVER $\leq_P$ SETCOVER. Построим конструкцию приведенную впервые Р. Карпом [1, Chapter 8, page 94] для задач (в оригинале) *NODE COVER* и *SET COVERING*.

Конструкция. Пусть множество M совпадает с множеством ребер: $M := E$. Для каждой вершины $v \in V$ задаем подмножество

$$S_v := \{e \in E \mid e \text{ инцидентно вершине } v\}.$$

Определим его подмножества как $\mathcal{S} := \{S_v \mid v \in V\}$. Параметр k остается тем же.

Корректность. Покажем эквивалентность решений.

($\Rightarrow$) Если $C \subseteq V$ — вершинное покрытие размера $\leq k$, то семейство $\mathcal{C} = \{S_v \mid v \in C\}$ покрывает M: каждое ребро инцидентно вершине из C т.е. принадлежит хотя бы одному множеству из $\mathcal{C}$, причем $|\mathcal{C}| = |C| \leq k$.

($\Leftarrow$) Обратно, пусть $\mathcal{C} \subseteq \mathcal{S}$ покрывает M и $|\mathcal{C}| \leq k$. Пусть $C := \{v \in V \mid S_v \in \mathcal{C}\}$. Получим $|C| = |\mathcal{C}|$. Т.к. каждое ребро $e = \{u, w\}$ лежит в некотором $S_v \in \mathcal{C}$, а e принадлежит только S_u и S_w, то $v \in \{u, w\}$ и ребро покрыто вершиной из C.

Сложность. Построение M и $\mathcal{S}$ требует перебрать ребра и для каждой вершины выписать инцидентные ей ребра. Это выполняется за $O(|E| + |V|)$, т.е. полиномиально от размера входа.

Таким образом, $\mathsf{VERTEXCOVER} \leq_P \mathsf{SETCOVER}$. Применяя теорему 1.2 и утверждению 1.3 получаем NP-полноту $\mathsf{SETCOVER}$. $\square$

3 Жадный алгоритм с приближением $O(\ln |M|)$

В этом разделе описывается жадный алгоритм решения задачи $\mathsf{SETCOVER}$ в невзвешенном варианте (т.е. вес каждого множества равен 1). На каждом шаге выбирается подмножество, покрывающее максимальное число еще не покрытых элементов.

Псевдокод

Algorithm 1: Greedy Unweighted Set Cover

$I \leftarrow \varnothing$ `// выбранные индексы множеств`

$\widehat{U} \leftarrow \varnothing$ `// уже покрытые элементы`

while $\widehat{U} \neq M$ **do**

  берем индекс ℓ такой, что $\ell = \arg\max_j |S_j \setminus \widehat{U}|$

  $I \leftarrow I \cup \{\ell\}$

  $\widehat{U} \leftarrow \widehat{U} \cup S_\ell$

return I

Реализация на Python

```
def choose_best_subset(remaining, uncovered):
    best_index = None
    best_gain = -1
    for i, subset in enumerate(remaining):
```

```
        gain = len(subset & uncovered)
        if gain > best_gain:
            best_gain = gain
            best_index = i
    return best_index


def greedy_set_cover(universe, subsets):
    U = set(universe)
    remaining = [set(s) for s in subsets]
    selected = []
    covered = set()
    while U != covered:
        best = choose_best_subset(remaining, U)

        selected.append(best)
        covered |= remaining[best]

        for s in remaining:
            s -= remaining[best]

    return selected
```

Замечание. В взвешенном варианте алгоритм (см. [3, Algorithm 1.2]) минимизирует отношение $w_j/|\widehat{S}_j|$, где w_j — вес множества, а $\widehat{S}_j$ — новые элементы, которые оно покрывает. В невзвешенном случае все веса равны 1, поэтому достаточно каждый раз выбирать множество с максимальной прибавкой.

Корректность. Очевидно по построению.

Анализ приближения

При доказательстве мы ссылаемся на [2, Part I, 2.1], но в невзвешенном варианте задачи и делая дополнительные пояснения.

Пусть $n = |M|$, а $U_t \subseteq M$ — множество еще *непокрытых* элементов перед t-ой итерацией алгоритма ($U_1 = M$, затем переход $U_{t+1} = U_t \setminus S_{\ell_t}$). Обозначим через $e_1, \dots, e_n$ элементы в том порядке, в каком алгоритм их покрывает (e_k — первый элемент U_t в очередной итерации).

Стоимость элемента. Когда выбирается множество S_{ℓ_t}, его «цена» 1 распределяется поровну между новыми элементами $S_{\ell_t} \cap U_t$:

$$\mathrm{price}(e) = \frac{1}{|S_{\ell_t} \cap U_t|} \quad \text{для каждого } e \in S_{\ell_t} \cap U_t.$$

Тогда

$$|\mathcal{C}_{\mathrm{greedy}}| = \sum_{S \in \mathcal{C}_{\mathrm{greedy}}} 1 = \sum_{k=1}^{n} \mathrm{price}(e_k). \tag{*}$$

Лемма 3.1. *Для каждого* $k \in \{1, \dots, n\}$ *выполняется*

$$\text{price}(e_k) \le \frac{\text{OPT}}{n-k+1}.$$

Доказательство. Рассмотрим момент, когда элемент e_k еще не покрыт (значит, $e_k \in U_t$). Оптимальное решение использует не более OPT множеств, чтобы покрыть все элементы U_t,где $|U_t| = n-k+1$. Следовательно, среди множеств оптимума найдется такое, которое в этот момент закрывает как минимум $\frac{n-k+1}{\text{OPT}}$ еще непокрытых элементов. Наш жадный выбор максимизирует размер $S_j \cap U_t$, поэтому

$$|S_{\ell_t} \cap U_t| \ge \frac{n-k+1}{\text{OPT}},$$

а значит

$$\text{price}(e_k) = \frac{1}{|S_{\ell_t} \cap U_t|} \le \frac{\text{OPT}}{n-k+1}.$$

□

Теорема 3.2. *Жадный алгоритм дает приближение* $H_n = 1 + \frac{1}{2} + \dots + \frac{1}{n} \le \ln(n) + 1$.

Доказательство. Применяя лемму 3.1 по всем k и пользуясь (*), получаем

$$|\mathcal{C}_{\text{greedy}}| = \sum_{k=1}^{n} \text{price}(e_k) \le \text{OPT} \cdot \sum_{k=1}^{n} \frac{1}{n-k+1} = \text{OPT} \cdot H_n.$$

□

4 LP-алгоритм с k-приближением

В этом разделе рассматривается случай, когда каждый элемент множества M присутствует не более чем в k множествах.

Описанный в [2, гл. 14] метод округления (в оригинале: «LP-rounding» или «rounding») сводится к двум шагам:
1) Решить линейную программу в виде дроби
2) Превратить полученный дробный вектор в целочисленный, стараясь при этом не сильно увеличить стоимость.
Покажем, что эти методом можно получить k-приближение.

Целочисленная и линейная формулировки

При описании ЦЛП и ЛП для SETCOVER мы ссылаемся на [3, Ch 7.1, page 162].
Обозначим через $\hat{x}_j \in \{0, 1\}$ флаг выбора множества S_j (0 – не взяли, 1 – взяли). Тогда исходная ЦЛП имеет вид:

$$\begin{aligned} \min \quad & \sum_{j=1}^{m} \hat{x}_j, \\ \text{s.t.} \quad & \sum_{j: e \in S_j} \hat{x}_j \ge 1 \quad (\forall e \in M), \\ & \hat{x}_j \in \{0, 1\} \quad (\forall j). \end{aligned} \tag{ЦПЛ}$$

Переходим к ЛП: $\hat{x}_j \in \{0,1\} \Rightarrow 0 \le x_j \le 1$ (в оригинале такой переход называется «LP-relaxation»), нам это нужно т.к. если бы мы работали с ЦЛП, то задача была бы **NP**-трудной, об этом говорится в [4, ч. 3.4.6, стр. 53]:

$$\begin{aligned} \min \quad & \sum_{j=1}^{m} x_j, \\ \text{s.t.} \quad & \sum_{j: e \in S_j} x_j \ge 1 \quad (\forall e \in M), \\ & 0 \le x_j \le 1 \quad (\forall j). \end{aligned} \tag{ЛП}$$

Обозначим оптимальное дробное решение ЛП через x^*, тогда $\mathrm{OPT}_{LP} = \sum_j x_j^*$. Также ясно, что $\mathrm{OPT}_{LP} \le \mathrm{OPT}_{ILP}$, т.к. при переходе к дробным x_j мы расширяем множество ответов.

Алгоритм округления

1. Находим оптимальное решение x^*.
2. Формируем покрытие $\mathcal{C} = \{S_j \mid x_j^* \ge 1/k\}$, или, что эквивалентно, целочисленное решение
$$\hat{x}_j = \begin{cases} 1, & x_j^* \ge \frac{1}{k}, \\ 0, & \text{иначе} \end{cases}$$

Реализация на Python

Минимальная реализация с использованием OR-Tools (pywraplp):

```
def lp_set_cover(universe, subsets, k):
    m = len(subsets)
    solver = pywraplp.Solver.CreateSolver('GLOP')
    x = [solver.NumVar(0, 1, f'x_{j}') for j in range(m)]
    for e in universe:
        solver.Add(sum(x[j] for j in range(m) if e in subsets[j]) >= 1)
    solver.Minimize(solver.Sum(x))
    solver.Solve()
    selected = [j for j in range(m) if x[j].solution_value() >= 1 / k]
    return selected
```

Интуиция. Так как каждый элемент встречается не более чем в k множествах, дробное ограничение $\sum_{j:e\in S_j} x_j^* \ge 1$ означает, что хотя бы одно из входящих $x_j^* \ge 1/k$. Взяв все множества с $x_j^* \ge 1/k$, мы гарантируем покрытие M, а число выбранных множеств не превосходит $k \cdot \mathrm{OPT}_{LP} \le k \cdot \mathrm{OPT}$. Далее формально.

Корректность алгоритма

Лемма 4.1. *Пусть $a_1 + \dots + a_k \ge 1$ и $a_i \ge 0$ для всех i. Тогда $\exists i$ такое, что $a_i \ge 1/k$.*

Доказательство. Предположим противное: $a_i < 1/k$ для всех i. Тогда $a_1 + \dots + a_k < k \cdot (1/k) = 1$, противоречие. □

Лемма 4.2. *Каждый элемент $e \in M$ покрыт.*

Доказательство. Возьмем любой элемент $e \in S_j$. В ЛП решении:

$$\sum_{j:e\in S_j} x_j^* \geq 1$$

В этой сумме участвует не более k слагаемых (по условию). Значит, по лемме 4.1 существует j такой, что $e \in S_j$ и $x_j^* \geq 1/k$, значит $\hat{x}_j = 1$, и элемент e покрыт. □

Анализ приближения

Теорема 4.3. *Округление по порогу $1/k$ дает k-приближение для невзвешенной задачи* SETCOVER.

Доказательство. Лемма 4.2 гарантирует, что $\mathcal{C}$ — целочисленное покрытие.

Рассмотрим любое множество $S_j \in \mathcal{C}$. По построению $x_j^* \geq 1/k$, откуда $1 \leq k\,x_j^*$. Суммируя по всем $S_j \in \mathcal{C}$, получаем

$$|\mathcal{C}| = \sum_{S_j\in\mathcal{C}} 1 \leq k \sum_{S_j\in\mathcal{C}} x_j^* \leq k \sum_{j=1}^{m} x_j^* = k \cdot \mathrm{OPT}_{LP} \leq k \cdot \mathrm{OPT},$$

□

Доказательство по сути совпадает с рассуждением в [2, Theorem 14.2] (но чуть формальнее и в невзвешенном случае): каждый элемент встречается в пределе k множеств, поэтому порог $1/k$ обеспечивает и корректность, и k-приближение.

5 Описание тестов и анализ

Датасеты

Для эмпирического сравнения жадного алгоритма и ЛП сформировано 4 датасета:

1. **Случайные покрытия, разреженные & плотные.** Для заданных $n = |M|$ и $m = 2n$ каждое множество S_j формируется включением элемента $e \in M$, после чего генерируются две выборки с плотностями $p_1 = 0.05$ и $p_2 = 0.3$, позволяя оценить влияние плотности на размер покрытия (для ЛП k случайные, высчитываются после генерации).

2. **Случайные покрытия, разреженные с большим k.** Берется модель (п. 1), но $m = n$ и $p = 0.05$ после чего все множества дополнительно содержат один общий элемент, тем самым $k = 2m$.

3. **Двудольные графы.** Случайные графы $G_{l,r}(p)$ переводятся в задачу SETCOVER через сведение VERTEXCOVER $\rightarrow$ SETCOVER (см. 2.2).

   - deg ≈ 4 – фиксированная средняя степень ($p = 4/N$).
   - разреженные & плотные – две выборки со степенями ≈ 2 и ≈ 10.

   Для двудольных графов минимальное вершинное покрытие равно размеру максимального паросочетания (Теорема Кенига 1.2), поэтому можно вычислить OPT за линейное время. Также, в силу того, что это *граф*: $k = 2$.

Метрики

Для датасетов фиксируются

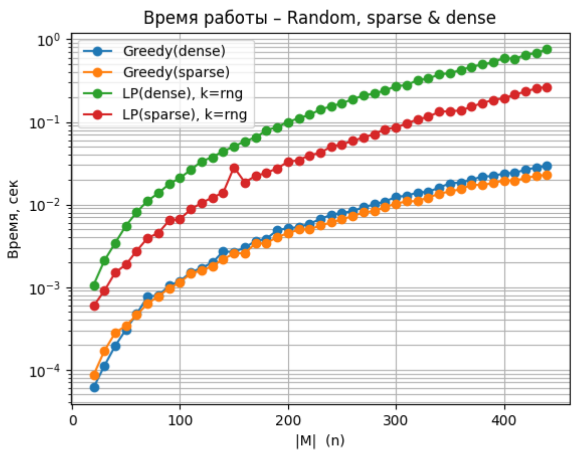
- $|\mathcal{C}_{\text{greedy}}|$ и $|\mathcal{C}_{LP}|$ – размеры полученных покрытий.
- Время работы алгоритмов.
- $|\mathcal{C}|/\text{OPT}$, при наличии OPT, т.е. только для двудольных графов.

Алгоритмы

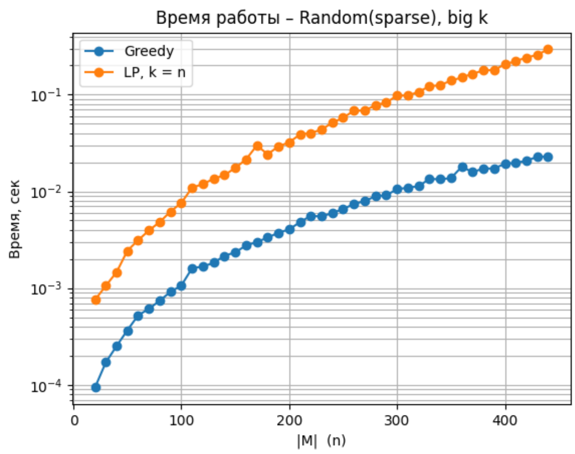
1. **Greedy** — жадный алгоритм из §3.
2. **LP** — ЛП, описанная в §4.

Анализ

По времени выполнения.

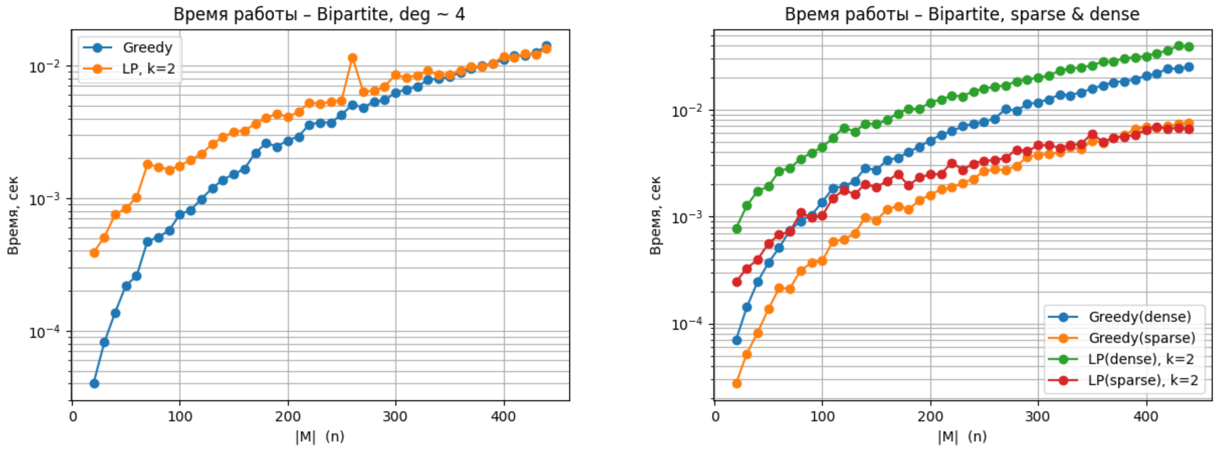


(а)



(б)

Рис. 1: Время работы на случайных покрытиях



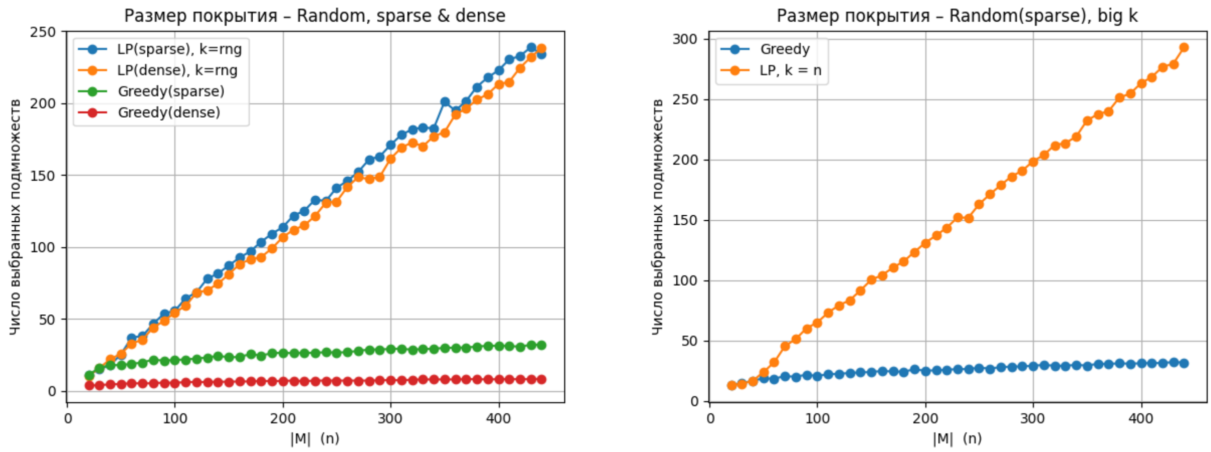
(а) (б)

Рис. 2: Время работы на двудольных графах

- **Случайные покрытия.** На рис. 1 видно, что жадный алгоритм существенно быстрее ЛП. Эта разница сохраняется даже при $k = m$. Кроме того, график (рис. 1,(в)) показывает, что жадный алгоритм тратит почти одинаковое время как на разреженных, так и на плотных случайных покрытиях, тогда как время ЛП возрастает при увеличении плотности, но на больших множествах имеет примерно одинаковое отношение.

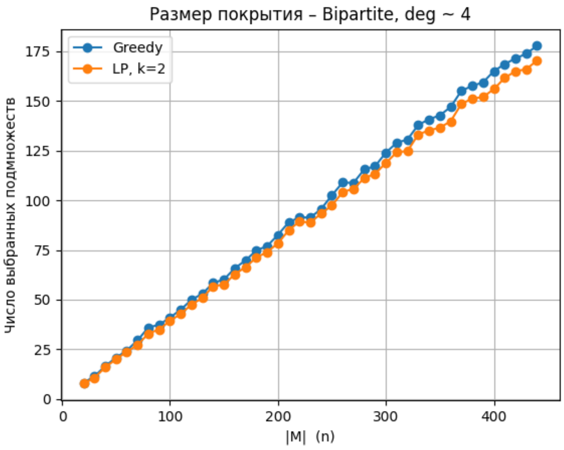
- **Двудольные графы.** Для графов со степенью ≈ 4 (рис. 2, (а)) жадный алгоритм быстрее лишь при $|M| < 250$. На больших графах оба работают почти одинаково. На рис. 2, (б) показано, что на плотных графах оба алгоритма работают медленнее, чем на разряженных, что закономерно, но отношение Greedy $\approx$ LP сохраняется.

По покрытию.

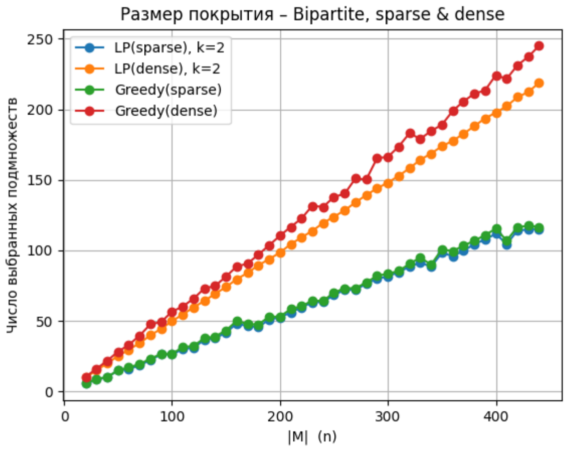


(а) (б)

Рис. 3: Размер покрытия на случайных покрытиях



(а)



(б)

Рис. 4: Размер покрытия на двудольных графах

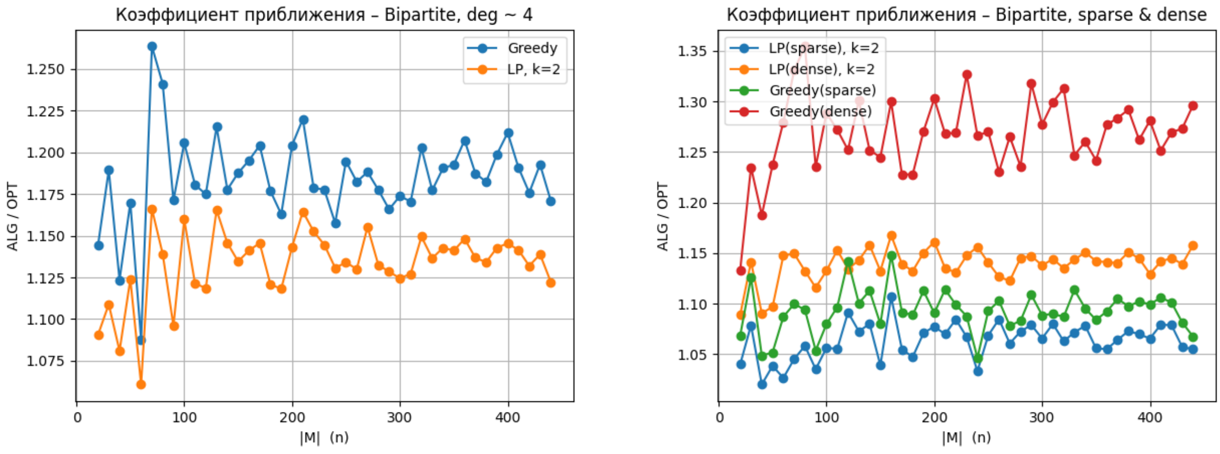
Обратите внимание на то, как мы генерируем данные: $2n$ множеств, каждый элемент $e \in M$ попадает в S_j с вероятностью p, т.е. для каждого $e \in M$ выполняется $freq(e) \sim Binom(m, p)$, а $k = \max_{e \in M} freq(e)$.

- **Случайные покрытия.**
- На рис. 3(а) жадный метод растет медленно, его кривая почти логарифмическая, поскольку каждый выбранный набор покрывает $\Theta(pn)$ новых элементов. С другой стороны, ЛП дает почти линейный рост, так как порог $1/k$ слишком мал, (при $n = 400$ $1/k \approx 0.012$ для разреженных и $1/k \approx 0.003$ для плотных), в то время как оптимальное дробное решение $x_j^* \approx 1/\mathbb{E}[freq(e)]$, поэтому в решение попадает большая часть множеств.
- На рис. 3(б) параметр $k = m$ делает порог $1/k$ еще меньше и, закономерно, ЛП выбирает еще больше множеств.

Двудольные графы.

- При степени ≈ 4 (рис. 4(а)) оба алгоритма показывают очень близкий результат, ЛП даже немного лучше в силу того, что параметр k достаточно мал и фиксирован.
- На рис. 4(б) видно, что размеры покрытий на разреженных графах существенно ниже, чем на плотных, что естественно и очевидно.

По качеству.



(а) (б)

Рис. 5: Отношение ALG/OPT ($|\mathcal{C}|$/OPT) на двудольных графах

Двудольные графы. На двудольных графах (рис. 5) ЛП стабильно ближе к OPT, что ожидаемо, и мы это уже наблюдали в анализе по покрытию, при $k = 2$. В разреженном случае оба алгоритма почти совпадают с OPT. Нетривиальным наблюдением является то, что при увеличением плотности разрыв между качеством алгоритмов увеличивается: жадного алгоритм заметно ухудшается, а ЛП незначительно.

6 Заключение

Сравнение алгоритмов

- **Время работы.** Жадный алгоритм растет линейно по времени и очень быстрый на случайных покрытиях. ЛП существенно медленнее, но при фиксированном $k = 2$ на двудольных графах его время быстро приближается к жадному алгоритму.
- **Размер покрытия.** Для случайных покрытий, где количество множеств, содержащих элемент, распределено биноминально, жадный алгоритм дает почти логарифмическую зависимость от $|M|$. ЛП алгоритм, напротив, выбирает почти все множества из-за малого $1/k$. Для иных распределений порог $1/k$, и, соответственно, результаты ЛП могут отличаться. Если же k фиксирован и мал (двудольные графы), покрытия обоих методов почти не отличаются.
- **Качество.** На двудольных графах и жадный, и ЛП решения укладываются в $[1.00, 1.4]$, что существенно лучше теоретических приближений (H_n для жадного и k для ЛП). Разрыв в пользу ЛП растет вместе с плотностью графа.

Таким образом, в практических задачах с умеренными k и разреженными множествами ЛП обеспечивает наилучшее качество при приемлемом времени, а жадный остается качественным вариантом в общем случае, особенно когда k велик или решение ЛП слишком дорогое.

Список литературы

[1] R. M. Karp, *Reducibility Among Combinatorial Problems* in R. E. Miller and J. W. Thatcher (eds.), 1972, pp. 85–103.

[2] V. V. Vazirani, *Approximation Algorithms*, Springer, 2013.

[3] D. P. Williamson, D. B. Shmoys, *The Design of Approximation Algorithms*, Cambridge University Press, 2011.

[4] Д. В. Мусатов, *Сложность вычислений: классика и современность*, МФТИ, 2024.