

Проект по сложности вычислений

Задача о покрытии множества

Хазраткулов Зиёбек

November 2022

Аннотация

Задача о покрытии множества является одним из классических вопросов комбинаторики, информатики, исследования операций и теории сложности. Эта задача является одним из 21 **NP**-полной задачи Карпа, которая в 1972 году доказали **NP**-полноту. В данной статье показана принадлежность задачи к классу **NP**-полные и описаны различные методы решения.

1 Формулировка задачи

Сформулируем задачу о покрытии множествами **SETCOVER**. Пусть дано конечное множество \mathcal{U} , называемое вселенной и конечная система его подмножеств $\mathcal{F} = \{S_1, \dots, S_n\}$. Без ограничения общности можно считать, что $\mathcal{U} = \{1, \dots, m\}$. В задаче поиска требуется найти минимальное по мощности покрытие $\mathcal{G} \subset \mathcal{F}$, такое что

$$\mathcal{U} = \bigcup_{S \in \mathcal{G}} S$$

В задаче распознавания требуется проверить, если ли такое покрытие мощности $\leq k$.

2 **NP**-полнота задачи **SETCOVER**

Сначала напомним формулировку задачи **VERTEXCOVER**. Пусть дан неориентированный граф $G = (V, E)$ и число k . Требуется проверить, есть ли в графе вершинное покрытие размера $\leq k$. Вершинным покрытием называется такое $V' \subset V$, что для любого ребра $\{u, v\} \in E$ выполнено, что $u \in V'$ или $v \in V'$. Известно, что данная задача является **NP**-полной¹.

Теорема 1. *Задача **SETCOVER** является **NP**-полной.*

¹Это показано, например в [1]

Доказательство. Докажем сначала **NP**-трудность задачи SETCOVER, сведя к ней **NP**-полную задачу VERTEXCOVER.

Пусть дан граф $G = (V, E)$ и число k . Пусть $|V| = n, |E| = m$. Положим $\mathcal{U} = E, \mathcal{F} = \{A_v : v \in V\}$, где $A_v = \{e \in E : v \in e\}$ – множество рёбер, покрываемых вершиной v . Ясно, что такая сводимость является полиномиальной, чтобы построить множество A_v , достаточно пройти по всем рёбрам графа, которых m . Значит, чтобы построить \mathcal{F} нужно $O(nm)$ времени, а на построение \mathcal{U} нужно $O(m)$ времени. Суммарно это $O(|G|)$. Докажем, что эта сводимость корректна.

Пусть в графе G нашлось вершинное покрытие $V', |V'| \leq k$. Тогда $\mathcal{G} = \{A_v : v \in V'\}$ будет покрытием множества \mathcal{U} (по определению вершинного покрытия V') мощности $\leq k$ (так как $|\mathcal{G}| = |V'| \leq k$).

Наоборот, пусть нашлось покрытие $\mathcal{G} = \{A_v : v \in V'\}$ множества \mathcal{U} размера $\leq k$. Тогда $|V'| = |\mathcal{G}| \leq k$ и вершины из V' покрывают все рёбра графа G по определению покрытия \mathcal{G} .

Итак, $\text{VERTEXCOVER} \leq_p \text{SETCOVER}$. Осталось показать, что SETCOVER \in **NP**. Действительно, для задачи SETCOVER легко построить верификатор $V(x, s)$. Сертификатом будет служить само покрытие. Действительно, покрытие является подмножеством входа программы, то есть сертификат имеет полиномиальную длину от размера входа. Задачей верификатора будет проверить, действительно ли s является покрытием вселенной \mathcal{U} . Для этого нужно для каждого элемента вселенной проверить, лежит ли он в каком-либо множестве в покрытии. Это можно сделать за полиномиальное время. \square

3 Жадный алгоритм и $\log n$ приближение

Опишем жадный алгоритм GREEDY решения задачи о покрытии. В каждый момент времени будем выбирать множество $A \in \mathcal{F}$, которое покрывает как можно больше ещё не покрытых элементов в множестве \mathcal{U} , и добавлять множество A в покрытие. Поскольку этот алгоритм пытается максимизировать покрытие только на данном шаге, он называется "жадным". Так как "не смотрит в будущее".

Приведём реализацию этого алгоритма на языке Python:

```

1 from typing import List, Set
2
3 def set_cover_greedy(universe: Set[int], sets: List[Set[int]]
4   ) -> List[Set[int]]:
5     if sets:
6         covered = set()
7         solution = []
8         while covered != universe:
9             best_set = None
10            best_coverage = 0
11            for i, s in enumerate(sets):
12                if i not in solution:
```

```

12         coverage = len(s - covered)/len(s)
13         if coverage > best_coverage:
14             best_set = i
15             best_coverage = coverage
16         solution.append(sets[best_set])
17         covered |= sets[best_set]
18     return solution
19 else:
20     return []

```

Оказывается, жадный алгоритм даёт $\log n$ -приближение, в смысле, который сформулирован в следующей теореме.

Теорема 2. Пусть OPT – оптимальный ответ для задачи *SETCOVER* на входе $(\mathcal{U}, \mathcal{F})$. Обозначим $n = |\mathcal{U}|$. Тогда алгоритм *GREEDY* запущенный на входе $(\mathcal{U}, \mathcal{F})$ выдаст ответ, по размеру не превосходящий $\log n * |OPT|$.

Доказательство. Пусть $\mathcal{A} = \{A_1, \dots, A_k\} \subset \mathcal{F}$ – ответ, выданный алгоритмом *GREEDY*, причём множества A_i пронумерованы в том порядке, в котором их выбирал алгоритм. Так как оптимальное решение использует $|OPT|$ множеств, то по принципу Дирихле, есть множество в \mathcal{F} , которое покрывает хотя бы $\frac{n}{|OPT|}$ точек вселенной \mathcal{U} . Поскольку жадный алгоритм выбирает каждый раз множество, которое покрывает как можно больше точек, то A_1 покрывает хотя бы $\frac{n}{|OPT|}$. Значит ещё не покрытых точек осталось не больше $n\left(1 - \frac{1}{|OPT|}\right)$. Далее, либо $A_1 \notin OPT$, либо $A_1 \in OPT$. В первом случае в OPT , а следовательно и в \mathcal{F} всё ещё найдётся по принципу Дирихле множество, покрывающее $\frac{1}{|OPT|}$ -долю оставшихся точек. Во втором случае по тому же самому принципу Дирихле найдётся множество в $OPT \setminus \{A_1\}$ покрывающее $\frac{1}{|OPT|-1}$ -долю оставшихся точек, а значит и $\frac{1}{|OPT|}$ -долю. В любом случае, в силу природы жадного алгоритма, A_2 покрывает хотя бы $\frac{1}{|OPT|}\left(n - \frac{n}{|OPT|}\right)$ точек. Значит осталось $n\left(1 - \frac{1}{|OPT|}\right)^2$ точек. Тогда после $|OPT| \log n$ шагов по индукции получаем, что осталось

$$n\left(1 - \frac{1}{|OPT|}\right)^{|OPT| \log n} < n\left(\frac{1}{e}\right)^{\log n} = 1$$

точек, а значит, к этому времени алгоритм завершился и $k < |OPT| \log n$. \square

4 Частные случаи и k -приближение

Рассмотрим следующий частный случай задачи *SETCOVER*. Пусть каждый элемент вселенной \mathcal{U} присутствует не более чем в k подмножествах семейства \mathcal{F} . Оказывается, в этом случае можно построить полиномиальный алгоритм, дающий k -приближение, сведя задачу к задаче о линейном программировании (ЛП)

Действительно, сначала сведём задачу к задаче о целочисленном линейном программировании (ЦЛП). Пусть x_i – индикатор того, что S_i множество входит в покрытие. Тогда задача SETCOVER формулируется как задача ЦЛП в следующем виде

$$\begin{cases} \min x_1 + \dots + x_n \\ \forall i : x_i \in \{0, 1\} \\ \forall u \in \mathcal{U} : \sum_{j: u \in S_j} x_j \geq 1 \end{cases} \quad (1)$$

Заметим, что в каждой сумме

$$\Sigma_u = \sum_{j: u \in S_j} x_j$$

не более чем k слагаемых. Значит, если $\Sigma_u \geq 1$, то найдётся $x_j \geq 1/k$, даже если отсутствуют ограничения на целочисленность переменных x_j . Это даёт нам право сформулировать следующий алгоритм LPSETCOVER.

Решим задачу ЛП 1 без ограничения на целочисленность за полиномиальное время², то есть ту, где все $x_i \geq 0$. В каждой сумме есть хотя бы один $x_j \geq 1/k$, округлим все такие x_j до 1, а все остальные положим равными 0. Понятно, что полученный набор (x_1, \dots, x_n) задаёт покрытие множества \mathcal{U} в силу вышенаписанного утверждения.

Осталось понять, что в процессе округления мы каждый x_j из решения ЛП умножили на некоторое число, не превосходящее k , чтобы получить 0 или 1. Значит, верно следующее

$$|LPSETCOVER| \leq k * |LPOPT| \leq k * |ILPOPT| = k * |OPT|$$

, где $LPSETCOVER$ – ответ, выданный алгоритмом, $LPOPT$, $ILPOPT$, OPT – оптимальные ответы для задач линейного программирования, целочисленного линейного программирования и задачи SETCOVER соответственно.

Список литературы

- [1] D. Musatov *Lecture notes on computational complexity*
<http://ru.discrete-mathematics.org/fall2015/3/complexity/lecture-3-4-np-complete.pdf>
- [2] Robert J. Vanderbei, Marc S. Meekon, Barry A. Freedman *A Modification of Karmarkar's Linear Programming Algorithm*
<https://link.springer.com/article/10.1007/BF01840454>
- [3] Avrim Blum *Lecture notes on computer science*
<https://www.cs.cmu.edu/~avrim/451f11/lectures/lects1-10.pdf>

²Алгоритм приведён в [2]