УДК 004.023

СРАВНИТЕЛЬНЫЙ АНАЛИЗ ЖАДНОГО АЛГОРИТМА И ЛЕНИВОГО АЛГОРИТМА ДЛЯ РЕШЕНИЯ ЗАДАЧИ О ВЕРШИННОМ ПОКРЫТИИ ГИПЕРГРАФА

Шиян Валерий Игоревич
Ассистент
Курченко Екатерина Алексеевна
Мирошниченко Артем Николаевич
Студенты
ФГБОУ ВО «Кубанский государственный университет»

Аннотация: В статье рассматривается одна из фундаментальных задач комбинаторной оптимизации – задача о вершинном покрытии гиперграфа. Предлагаются жадный алгоритм и ленивый алгоритм, адаптированные для решения задачи о вершинном покрытии гиперграфа, приводятся примеры работы, а также анализ запусков предложенного жадного алгоритма и ленивого алгоритма.

Ключевые слова: задача комбинаторной оптимизации, задача о вершинном покрытии гиперграфа, жадный алгоритм, ленивый алгоритм

COMPARATIVE ANALYSIS OF A GREEDY ALGORITHM AND A LAZY ALGORITHM FOR SOLVING THE PROBLEM OF VERTEX COVERING OF A HYPERGRAPH

Shiyan Valery Igorevich
Kurchenko Ekaterina Alekseevna
Miroshnichenko Artyom Nikolaevich

Abstract: The article considers one of the fundamental problems of combinatorial optimization – the problem of vertex covering of a hypergraph. A greedy algorithm and a lazy algorithm adapted to solve the problem of vertex covering of a hypergraph are proposed, examples of work are given, as well as an analysis of the launches of the proposed greedy algorithm and lazy algorithm.

Key words: combinatorial optimization problem, hypergraph vertex cover problem, greedy algorithm, lazy algorithm

Задача

Разработать алгоритм решения задачи о вершинном покрытии гиперграфа [1, 2].

Гиперграф – пара (V, E), где V – непустое множество вершин гиперграфа, а E – семейство непустых (необязательно различных) подмножеств множества V, называемых рёбрами гиперграфа.

На рис. 1 изображён пример гиперграфа.

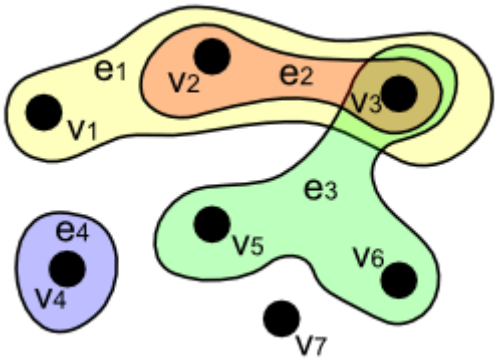

Рис. 1. Пример гиперграфа

Вершинное покрытие в гиперграфе – множество вершин, такое, что каждое ребро гиперграфа содержит по меньшей мере одну вершину из этого множества.

Размер вершинного покрытия – число входящих в него вершин.

Задача о вершинном покрытии состоит в поиске вершинного покрытия наименьшего размера для заданного гиперграфа.

На входе: гиперграф $G = (V, E)$.

Результат: множество $C \subseteq V$ – наименьшее вершинное покрытие гиперграфа G.

Решение

Для представления гиперграфа $G = (V, E)$ в памяти компьютера используются списки инцидентности, в которых указываются связи между инцидентными элементами гиперграфа (ребро и вершина). В программе списки инцидентности хранятся в словаре, ключи которого – это вершины, значения – список рёбер, инцидентные той или иной вершине. В программе списки рёбер

хранятся в списках. Реализация функции, которая позволяет сгенерировать произвольный гиперграф, изображена на рис. 2.

```
def generate(self, number_of_vertexes, number_of_edges):
    '''
    Генерация гиперграфа.

    Ключевые аргументы:
    number_of_vertexes -- количество вершин
    number_of_edges -- количество рёбер
    '''
    self.number_of_vertexes = number_of_vertexes
    self.number_of_edges = number_of_edges
    self.lists_of_incidence = dict()
    if self.number_of_vertexes >= 3:
        fixed_edge = randint(0, number_of_edges - 1)
    fixed_vertexes = sample(
        list(range(self.number_of_vertexes)),
        randint(3, self.number_of_vertexes)
    )
    for vertex in range(self.number_of_vertexes):
        edges = sample(
            list(range(self.number_of_edges)),
            randint(0, min(5, self.number_of_edges))
        )
        if self.number_of_vertexes >= 3 and vertex in fixed_vertexes and \
           not(fixed_edge in edges):
            edges.append(fixed_edge)
        self.lists_of_incidence[vertex] = edges
```

Рис. 2. Функция, которая позволяет сгенерировать произвольный гиперграф

Жадный алгоритм

Для решения поставленной задачи реализован жадный алгоритм. Жадный алгоритм заключается в следующем.

На каждом шаге, пока не закончатся все рёбра в графе:

1. Выбираем вершину, покрывающую наибольшее число рёбер.
2. Добавляем её в решение.
3. Удаляем из графа все рёбра, её инцидентные.

Реализация жадного алгоритма изображена на рис. 3.

```
def greedy_algorithm(self):
    '''
    Жадный алгоритм.
    '''
    res = set()
    lists_of_incidence = deepcopy(self.lists_of_incidence)
    vertices = list(range(self.number_of_vertexes))
    degrees = dict([(vertex, self.degree(vertex)) for vertex in vertices])
    while len(vertices) > 0:
        max_vertex = max_degree = -1
        for vertex in vertices:
            if degrees[vertex] > max_degree:
                max_vertex, max_degree = vertex, degrees[vertex]
        res.add(max_vertex)
        i = 0
        a = set(lists_of_incidence[max_vertex])
        while i < len(vertices):
            vertex = vertices[i]
            b = set(lists_of_incidence[vertex])
            lists_of_incidence[vertex] = list(set(b) - set(a))
            if len(lists_of_incidence[vertex]) == 0:
                lists_of_incidence.pop(vertex)
                vertices.remove(vertex)
                degrees.pop(vertex)
            else:
                degrees[vertex] = len(lists_of_incidence[vertex])
                i += 1
    self.cover = res
    return res
```

Рис. 3. Реализация жадного алгоритма

Пример работы жадного алгоритма представлен на рис. 4.

Рис. 4. Пример работы жадного алгоритма

Ленивый алгоритм

Также для решения поставленной задачи реализован ленивый алгоритм. Ленивый алгоритм заключается в следующем.

На каждом шаге, пока не закончатся все рёбра в графе:

1. Выбираем первую попавшуюся вершину.
2. Добавляем её в решение.
3. Удаляем из графа все рёбра, её инцидентные.

Реализация ленивого алгоритма изображена на рис. 5.

```
def lazy_algorithm(self):
    '''
    Ленивый алгоритм.
    '''
    res = set()
    lists_of_incidence = deepcopy(self.lists_of_incidence)
    vertices = list(range(self.number_of_vertexes))
    while len(vertices) > 0:
        first_vertex = vertices[0]
        res.add(first_vertex)
        i = 0
        a = set(lists_of_incidence[first_vertex])
        while i < len(vertices):
            vertex = vertices[i]
            b = set(lists_of_incidence[vertex])
            lists_of_incidence[vertex] = list(set(b) - set(a))
            if len(lists_of_incidence[vertex]) == 0:
                lists_of_incidence.pop(vertex)
                vertices.remove(vertex)
            else:
                i += 1
    self.cover = res
    return res
```

Рис. 5. Реализация ленивого алгоритма

Пример работы ленивого алгоритма представлен на рис. 6.

Рис. 6. Пример работы ленивого алгоритма

Заключение

При запуске обоих алгоритмов количество вершин и рёбер равнялось 10, 20, 30, …, 100.

На рис. 7 изображена зависимость количества вершин в вершинном покрытии от количества вершин в гиперграфе.

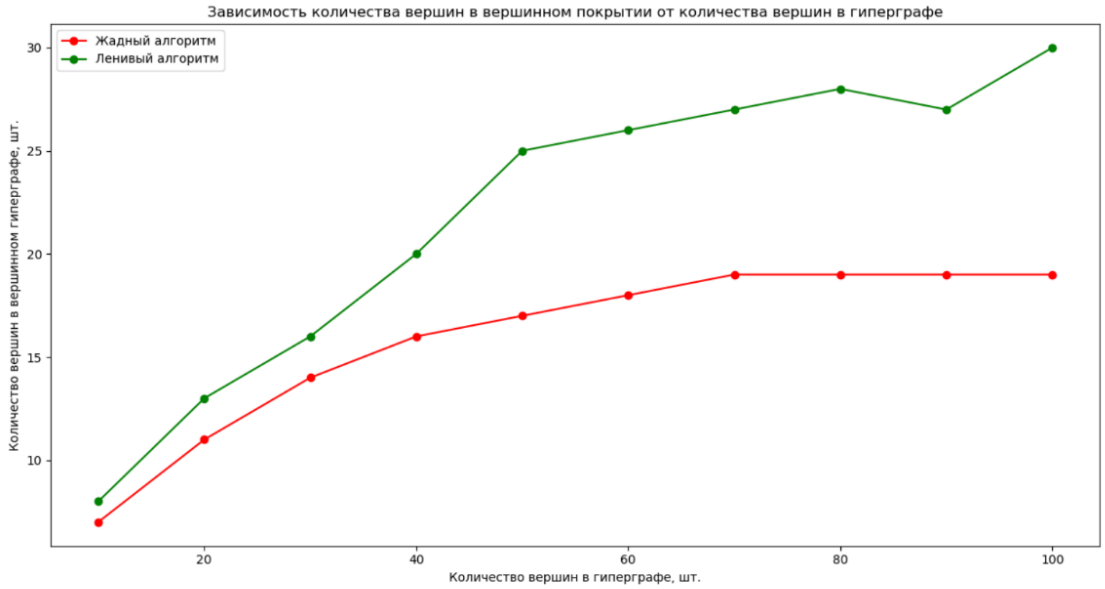

Рис. 7. Зависимость количества вершин в вершинном покрытии от количества вершин в гиперграфе

Из рисунка видно, что чем больше количество вершин в гиперграфе, тем жадный алгоритм находит более оптимальные решения в отличие от ленивого алгоритма.

На рис. 8 изображена зависимость времени выполнения от количества вершин в гиперграфе.

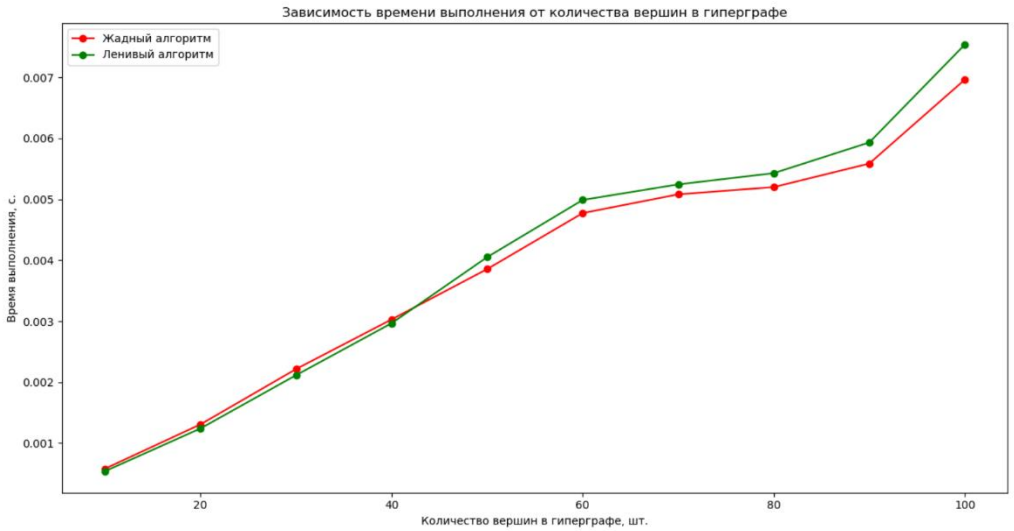

Рис. 8. Зависимость времени выполнения от количества вершин в гиперграфе

Из рисунка видно, что оба алгоритма выполняются примерно за одно и то же время. Это связано с тем, что запуски производились на относительно небольших графах.

На рис. 9 изображена зависимость количества вершин в вершинном покрытии от количества рёбер в гиперграфе.

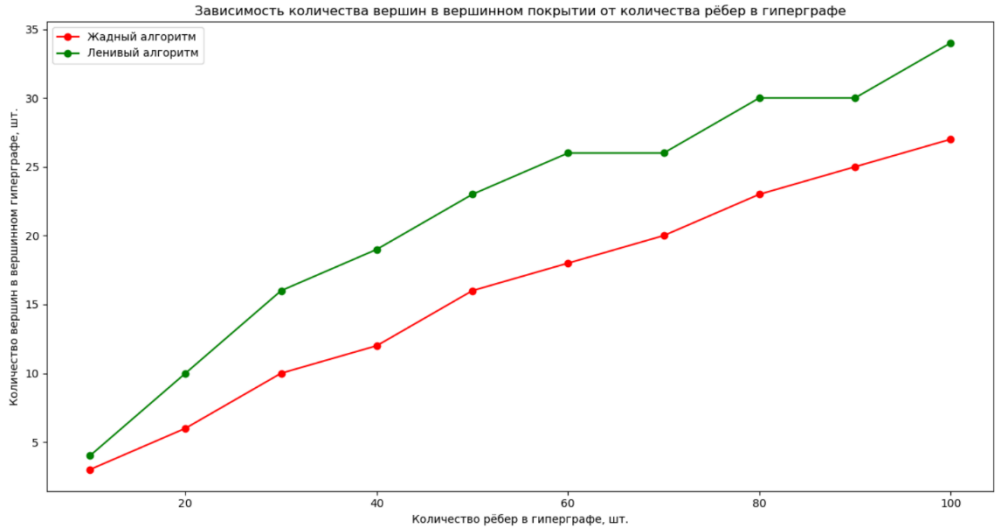

Рис. 9. Зависимость количества вершин в вершинном покрытии от количества рёбер в гиперграфе

Из рисунка видно, что чем больше количество рёбер в гиперграфе, тем жадный алгоритм находит более оптимальные решения в отличие от ленивого алгоритма.

На рис. 10 изображена зависимость времени выполнения от количества рёбер в гиперграфе.

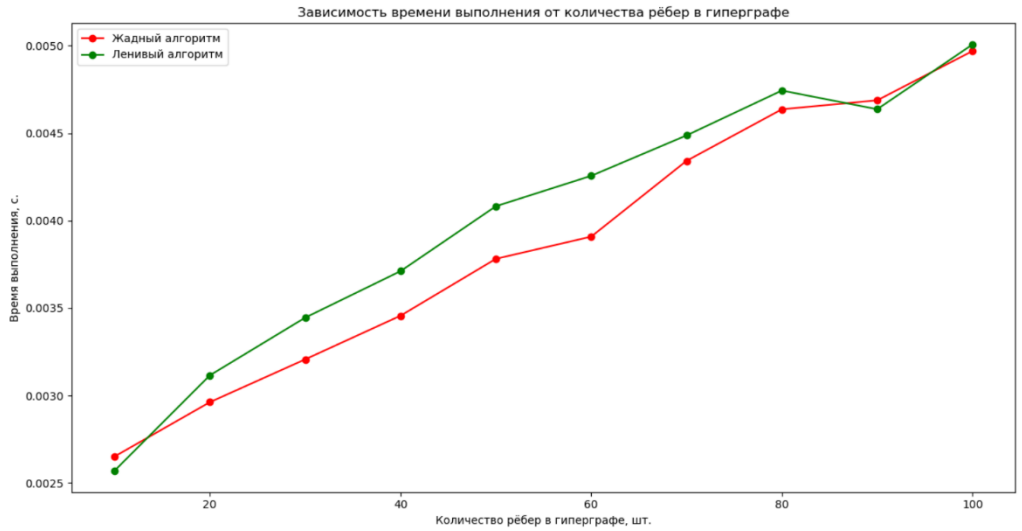

Рис. 10. Зависимость времени выполнения от количества рёбер в гиперграфе

Из рисунка видно, что оба алгоритма выполняются примерно за одно и то же время. Связано это с тем, что запуски производились на относительно небольших графах. На основании имеющихся результатов можно сделать вывод, что жадный алгоритм находит более оптимальные решения в отличие от ленивого алгоритма. Оба алгоритма выполняются примерно за одно и то же время, но на относительно небольших графах, на которых производились запуски.

Список литературы

1. Зыков А. А. Гиперграфы // Успехи математических наук. – 1974. – № 6. – С. 89-154.

2. В. А. Емеличев, О. И. Мельников, В. И. Сарванов, Р. И. Тышкевич. Глава XI: Гиперграфы // Лекции по теории графов. – 1990. – С. 298-315. – 384 с.