Московский физико-технический институт (ГУ)
Физтех-школа прикладной математики и информатики

Задача об оптимальном расписании

Алгоритм 4/3-приближения и его анализ

Волович Василий
Б05-222

Сложность вычислений, осень 2024

Аннотация

Задача об оптимальном расписании является одной из самых известных задач комбинаторной оптимизации и была первой задачей, для которой был представлен конкурентный анализ, Грэмом в 1966 году. Существует множество вариаций данной задачи, и в данной работе мы вслед за Грэмом рассмотрим случай, когда производительности всех машин одинаковы. Автор доказывает NP-полноту задачи, а также приводит реализацию алгоритма 4/3-приближения и доказательство его корректности.

Содержание

1 Введение

1.1 Постановка задачи

Имеется множество работ J и множество машин M. Также задана функция $p : J \times M \to \mathbb{N}$. Значение p_{ij} означает время выполнения i-й работы на j-й машине. Требуется построить распределение работ по машинам так, чтобы все работы были выполнены и чтобы конечное время выполнения всех работ было минимально. Иначе говоря, требуется найти функцию $x : J \times M \to \{0, 1\}$ такую, что:

$$\max_{j \in M} \sum_i x_{ij} p_{ij} \to \min$$

$$\text{s.t.} \forall i \sum_{j \in M} x_{ij} = 1$$

В данной работе мы будем рассматривать частный случай этой задачи, когда производительности всех машин одинаковы, то есть $p_{ij} = p_i$.

1.2 Переформулировка задачи

Мы хотим переформулировать задачу так, чтобы ответ был бинарным. Это можно сделать следующим образом, заодно использовав что производительности всех машин одинаковые.

На вход подаются натуральные числа n и m — количество работ и машин соответственно и натуральное число C — максимальное время выполнения всех работ. Также дан массив p размера n, где p_i означает время выполнения i-й работы. Спрашивается, можно ли каждую работу выполнять ровно на одной машине таким образом, чтобы общее время выполнения всех работ получилось не больше C.

2 NP-полнота

Докажем, что сформулированная в предыдущем пункте задача является NP-полной. Для этого сведём к ней задачу Partition problem. Эта задача заключается в следующем: нам дан набор натуральных чисел $p_1, \ldots, p_n$, и нужно определить, можно ли его разбить на два поднабора с одинаковой суммой.

Итак, пусть нам даны натуральные $p_1, \ldots, p_n$. На вход нашей задаче о расписании подадим $m = 2$, $p_1, \ldots, p_n$ и $C = \frac{1}{2}(p_1 + \cdots + p_n)$. Тогда, если разбиение на два поднабора с одинаковой суммой существует, то суммы в поднаборах равны C, и задача о расписании тоже имеет решение. Обратно, если задача о расписании имеет решение, то так как первая машина работает $\leqslant C$ времени, и вторая тоже, а в сумме они работают $p_1 + \cdots + p_n = 2C$ единиц времени, то на самом деле каждая из них работает ровно C единиц времени, и тем самым мы получили разбиение на два поднабора с одинаковой суммой. Ясно, что сведение полиномиальное.

Так как задача Partition problem NP-полная (об этом можно прочитать, например, здесь [1]), то и наша задача об оптимальном расписании является NP-полной.

3 Алгоритм

3.1 Мотивировка и реализация

После переформулировки задачи сразу хочется решать её жадным алгоритмом: пусть вначале все машины свободны. Будем перебирать работы в порядке от сложных к простым. На каждом шаге мы рассматриваем новое время выполнения работы p_i — самое больше из пока не рассмотренных. Эту работу мы скармливаем машине с минимальной загрузкой на данный момент.

Реализация алгоритма на C++ очень проста:

```
int64_t greedy(int64_t n, int64_t m, std::vector<int64_t> p) {
    std::sort(p.begin(), p.end(), std::greater<>());
    std::multiset<int64_t> machines;
    for (int64_t i = 0; i < m; ++i) {
    machines.insert(0);
    }
    for (int64_t i = 0; i < n; ++i) {
    auto iter = machines.begin();
    int64_t min_value = *iter;
    machines.erase(iter);
    machines.insert(min_value + p[i]);
    }
    auto iter = machines.rbegin();
    int64_t max_value = *iter;
    return max_value;
}
```

Однако оказывается, что этот алгоритм не всегда выдаёт верный ответ на задачу.

Я нашёл в [2] пример, когда ответ, выдаваемый этим жадным алгоритмом, довольно сильно отличается от оптимального. Если m — чётное натуральное число, $n = 2m + 1$, $p = [2m - 1, 2m - 1, 2m - 2, 2m - 2, \ldots, m + 1, m + 1, m, m, m]$, то жадный алгоритм распределит работы по машинам таким образом:

- $2m - 1, m, m$
- $2m - 1, m$
- $2m - 2, m + 1$
- $2m - 2, m + 1$
- $\ldots$
- $3m/2, 3m/2 - 1$
- $3m/2, 3m/2 - 1$

и вернёт ответ $4m - 1$, но оптимальное разбиение это:

- m, m, m
- $2m - 1, m + 1$
- $2m - 1, m + 1$
- $2m - 2, m + 2$

- $2m-2, m+2$
- $\ldots$
- $3m/2, 3m/2$

и оптимальный ответ $3m$. Это следует из того, что сумма всех чисел равна $3m^2$ и, по принципу Дирихле, время работы хотя бы одной из m машин должно быть $\geqslant 3m$.

Таким образом, ответ жадного алгоритма в $\dfrac{4m-1}{3m}$ раз больше оптимального.

3.2 Доказательство корректности

Оказывается, что жадный алгоритм всегда даёт 4/3-приближение к ответу, то есть описанный выше пример — «худший случай». Я нашёл красивое доказательство этого факта здесь [3]. Ниже привожу свою переформулировку этого рассуждения.

Теорема. Пусть при оптимальном разбиении работ по машинам время выполнения всех работ равно ANS. Докажем, что ответ, полученный жадным алгоритмом, будет $\leqslant \frac{4}{3}ANS$.

Доказательство. Разделим p_i на три класса:
1) *простые* $p_i \leqslant \frac{1}{3}ANS$,
2) *средние* $\frac{1}{3}ANS < p_i \leqslant \frac{2}{3}ANS$,
3) *сложные* $\frac{2}{3}ANS < p_i \leqslant ANS$.

Если $n \leqslant m$, то жадный алгоритм распределит задачи по $\leqslant 1$ на машину и получит оптимальный ответ ANS. Пусть $n > m$. Обозначим h — количество сложных задач. Тогда $h \leqslant m$ и за первые h шагов жадный алгоритм назначит первым h машинам по одной из этих h сложных задач. Заметим, что это же сделает и оптимальный алгоритм, потому что назначить две сложные задачи одной машине нельзя.

Заметим, что оптимальный алгоритм не может также назначить сложную и среднюю задачу одной машине, значит все средние задачи как-то распределены между оставшимися $m-h$ машинами. Но он не может назначить три средних задачи одной машине, значит средних задач $\leqslant 2(m-h)$. Из этого следует, что жадный алгоритм тоже не назначит сложную и среднюю задачи одной машине, т.к. чтобы это произошло, каждой из оставшихся $m-h$ машин уже должно быть назначено по $\geqslant 2$ задачи, то есть все средние задачи уже будут израсходованы.

Значит, все средние задачи как-то распределены между оставшимися $m-h$ машинами. При этом каждая из них будет выполнять $\leqslant 2$ средних задач, потому что иначе к моменту распределения третьей средней задачи все из оставшихся $m-h$ машин должны были бы иметь по $\geqslant 2$ средние задачи, то есть все средние задачи уже будут израсходованы. Значит, если машине дали $\geqslant 3$ задачи, то последней ей дали лёгкую.

Таким образом, мы доказали две леммы.

Лемма 1. Жадный алгоритм ни одной машине не назначил две сложные или сложную и среднюю задачи.

Лемма 2. Если жадный алгоритм назначил машине $\geqslant 3$ задачи, то последней он назначил ей лёгкую задачу.

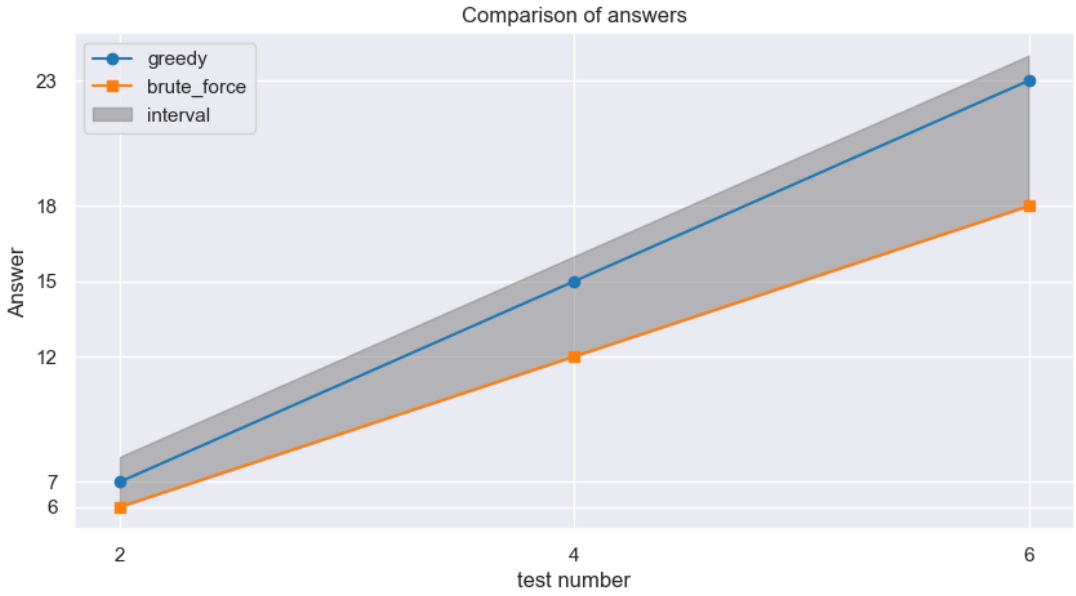
Теперь используем эти две леммы для доказательства требуемого утверждения.

Пусть p_i — время выполнения задачи, которая заканчивает выполняться самой последней. Тогда перед тем, как эта задача была назначена на свою машину, нагрузка этой машины была $\leqslant ANS$ (т.к. иначе к этому моменту нагрузка на всех машинах была бы $> ANS$, что противоречит оптимальному примеру с ответом ANS). Если это единственная задача на данной машине, то время её выполнения $\leqslant ANS$ — победа.

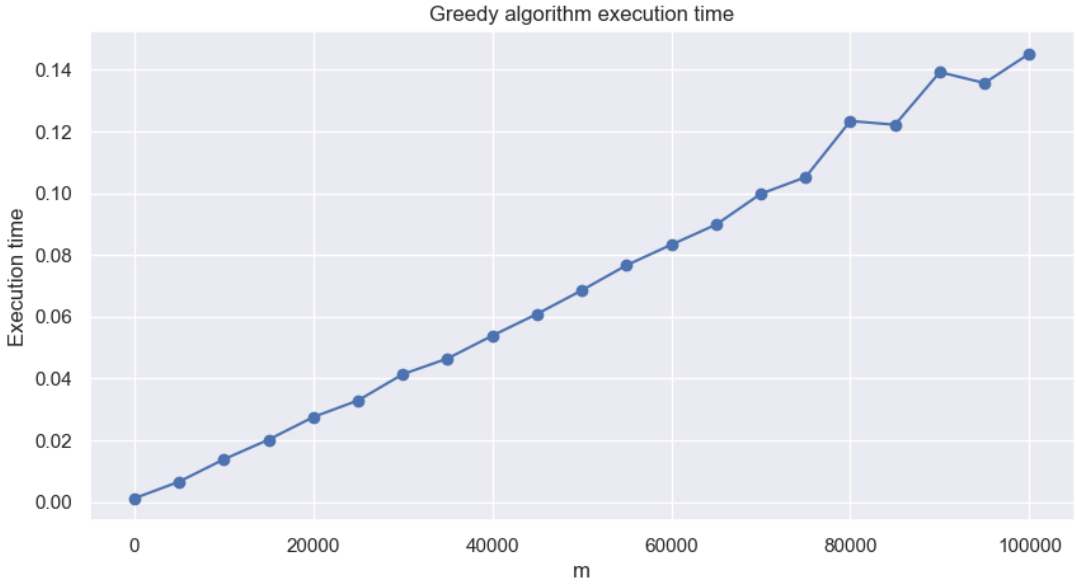
Если это вторая задача на данной машине, то, так как эти задачи не могут быть обе сложными или сложной и средней, то время их выполнения $\leqslant \frac{2}{3}ANS + \frac{2}{3}ANS = \frac{4}{3}ANS$ — победа. Если же это третья (или более) задача на данной машине, то она простая, а значит время выполнения всех задач $\leqslant ANS + \frac{2}{3}ANS = \frac{4}{3}ANS$ — победа. Во всех случаях мы получили 4/3-приближение, ч.т.д.

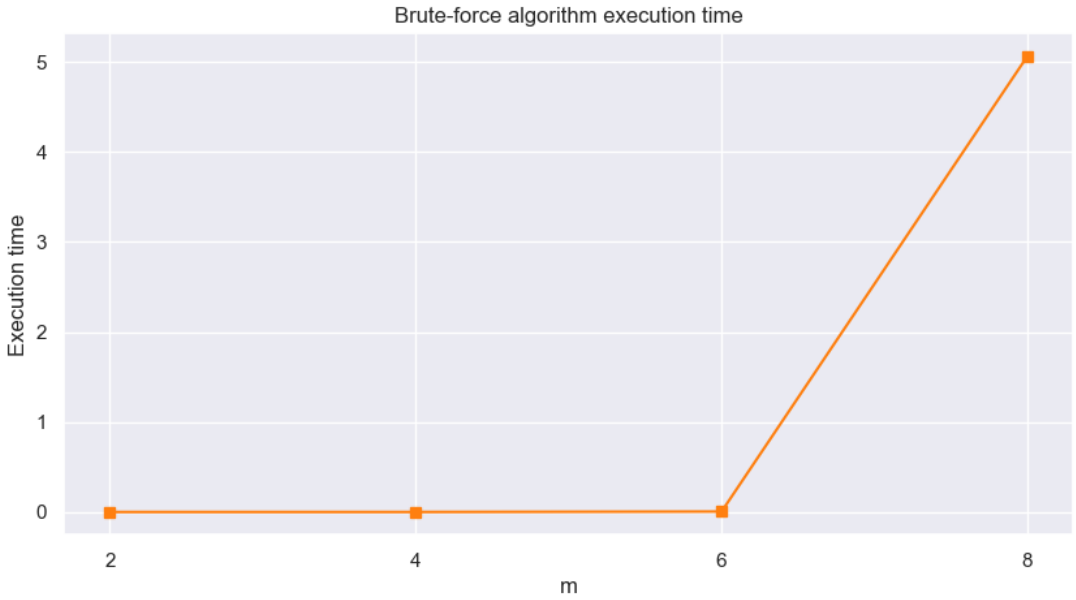
3.3 Анализ работы

Я написал переборный алгоритм, который гарантированно находит верный ответ, и теперь мы сравним его с жадным алгоритмом. Вначале я проверил, что на тестах вида, описанного при мотивировке алгоритма, они выдавали такие ответы, как нужно.



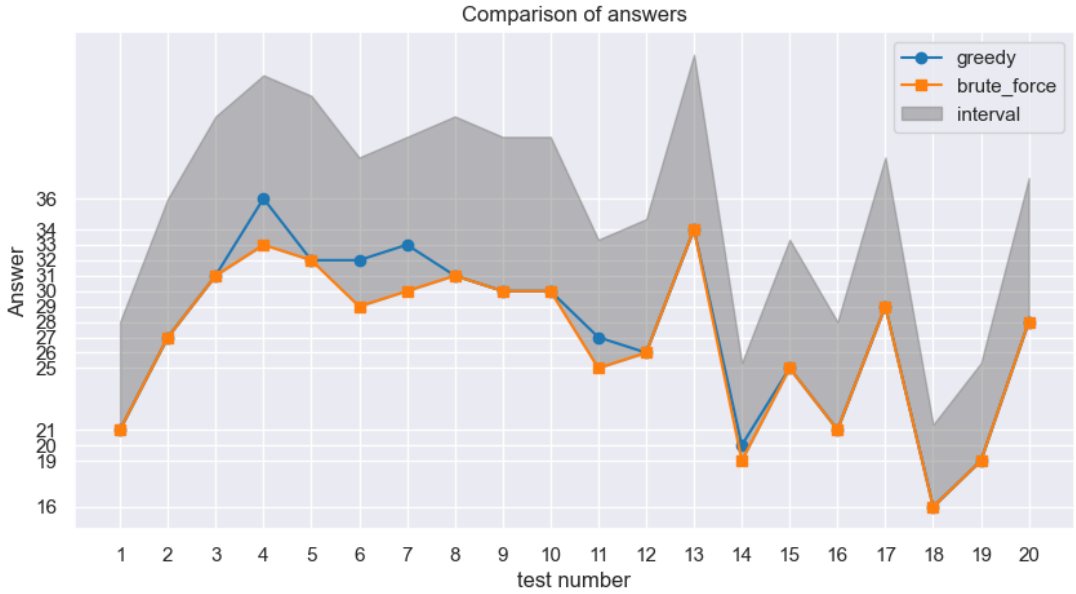
Ответы действительно соответствуют тому что мы теоретически вывели выше. Жадный алгоритм выдаёт $4m-1$, а переборный $3m$. Теперь сравним скорость работы алгоритмов.



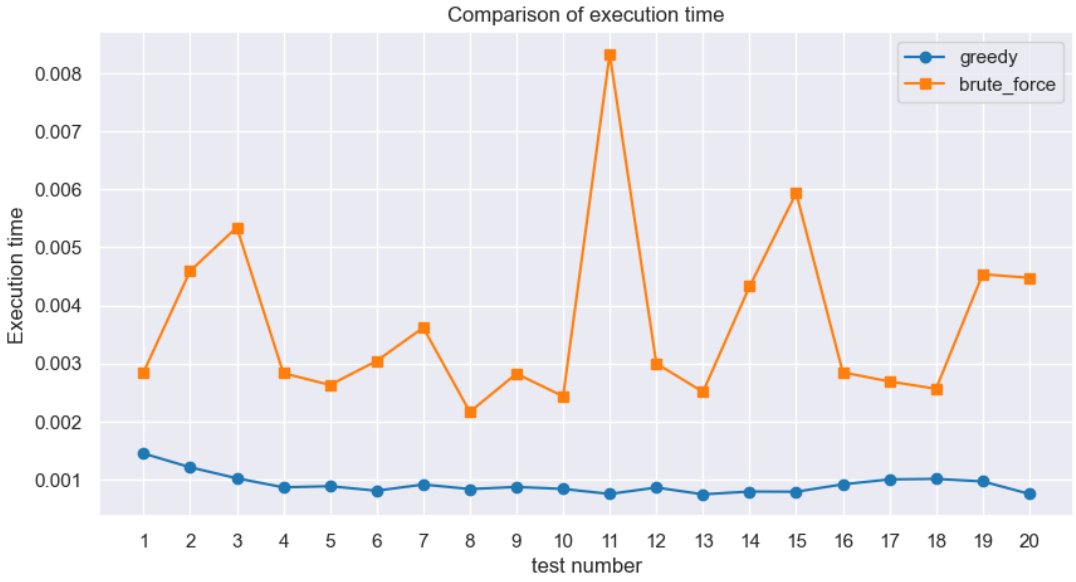


Мы видим, что жадный алгоритм работает очень быстро, практически линейно в зависимости от m, а переборный уже для $m = 8$ работает 5 секунд.

Теперь сравним ответы алгоритмов на случайных тестах. Возьмём $m = 5$, $n = 12$ и будем генерировать массив p из n случайных чисел от 1 до 20.



Мы видим, что для маленьких m и n жадный алгоритм довольно часто выдаёт правильный ответ, его ответ не сильно отличается от ответа переборного алгоритма. Теперь сравним время работы.



Мы видим, что даже на маленьких тестах переборный алгоритм работает медленнее, при больших m и n дело обстоит ещё намного хуже, как мы уже видели выше.

4 Вывод

Мы доказали NP-полноту задачи об оптимальном расписании, то есть придумать быстрый алгоритм её решения вряд ли получится. Однако оказывается, что простой жадный алгоритм даёт 4/3-приближение к ответу, и это доказывается довольно красивым способом. В добавок, жадный алгоритм работает очень быстро и выдаёт ответ, который на практике не сильно отличается от верного.

5 Ссылки

1. Wikipedia, *Partition problem*. Ссылка
2. Wikipedia, *Longest-processing-time-first scheduling*. Ссылка
3. Xin Xiao, *A Direct Proof of the 4/3 Bound of LPT Scheduling Rule*. Ссылка
4. Волович Василий, *Задача об оптимальном расписании. Алгоритм 4/3-приближения и его анализ*. Ссылка