Задача об оптимальном расписании

Гриб Александр

Октябрь 2020

В проекте рассмотрена задача об оптимальном расписании, когда производительность всех машин одинакова. Так же данная задача называется задачей многопроцессорного планирования. Доказана **NP**-полнота, построен эвристический алгоритм и показано что он дает $\frac{4}{3}$-приближение

Введение

Пусть имеется множество работ J и множество машин M. Также задана функция $p\colon\ J \times M \to \mathbb{R}_+$. Значение p_{ij} означает время выполнение i-ой работы на j-ой машине. Требуется построить распределение работ по машинам так, чтобы все работы были выполнены и чтобы конечное время выполнения всех работ было минимально. То есть требуется найти функцию $x : J \times M \to \{0, 1\}$ такую, что:

$$\sum_{j \in M} x_{ij} = 1, \text{для всех } i$$

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

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

Данная задача также называется задачей многопроцессорного планирования и используется для распределения задач на кластерах с большим количеством почти одинаковых процессоров

Как будет показано далее, задача является **NP**-полной и существует полиномиальный алгоритм дающий $\left(\frac{4}{3} - \frac{1}{3m}\right)$-приближение, где m – количество машин

NP-полнота

Обозначим язык

$$\{(T, m, k) \mid \text{на } m \text{ машинах можно выполнить все задачи из } T \text{ за время} \leq k\}$$

как MULTIPROCESSOR-SCHEDULING

Утверждение 1. *MULTIPROCESSOR-SCHEDULING $\in$ **NP**, так как сертификатом можно выбрать набор из чисел 1-m, где i-ое число обозначает, что i-ая задача выполняется на машине с этим номером. Теперь вернем 0, если есть машина на которой суммарное время $> k$ и 1 иначе. Проверка суммарного времени выполнения всех задач на каждой машине выполняется за линию.*

Утверждение 2. *$A \leq_p B$, A - **NP**-полная, $B \in$ **NP** $\Rightarrow B$ - **NP**-полный*

Доказательство. Доказывается по определению **NP**-полной задачи и транзитивности сводимости по Карпу. Если любая задача C из **NP** такая, что $C \leq_p A$ (из определения **NP**-полной задачи) и $A \leq_p B$ (из условия), то $C \leq_p B$ и $B \in$ **NP** (из условия), то по определению B - **NP**-полный □

Определение 1. *SUBSETSUM* $=$
$\{(n_1, n_2, ..., n_k, N) \mid$ из набора чисел n_1, ..., n_k можно выбрать подмножество с суммой $N\}$

Теорема 1. *SUBSETSUM - **NP**-полный*

Доказательство приведено в Huilgol, Chhabra, and Luhadia [2013]

Осталось доказать, что SUBSETSUM $\leq_p$ MULTIPROCESSOR-SCHEDULING , тогда из MULTIPROCESSOR-SCHEDULING $\in$ **NP**, SUBSETSUM — **NP**-полный и утверждению 2 следует что MULTIPROCESSOR-SCHEDULING — **NP**-полный

Теорема 2. *SUBSETSUM $\leq_p$ MULTIPROCESSOR-SCHEDULING*

Доказательство. Приведем полиномиально вычислимую функцию f из SUBSETSUM в MULTIPROCESSOR-SCHEDULING .

1. Пусть
$$\sum_{1 \leq i \leq k} n_i = 2N$$

Тогда f преобразует $(n_1, n_2, ..., n_k, N)$ в $((n_1, n_2, ..., n_k), 2, N)$ – обозначим Q. И если Q имеет решение в MULTIPROCESSOR-SCHEDULING , то у нас есть 2 набора в каждом из которых сумма - N, а если решения MULTIPROCESSOR-SCHEDULING нет, то никак нельзя поделить $(n_1, n_2, ..., n_k)$ на 2 множества по N (иначе бы MULTIPROCESSOR-SCHEDULING поделило бы и положила на 2 машины и Q принадлежало бы MULTIPROCESSOR-SCHEDULING).

2. Пусть
$$\sum_{1 \leq i \leq k} n_i < 2N$$

Тогда f преобразует $(n_1, n_2, ..., n_k, N)$ в $\left(\left(n_1, n_2, ..., n_k, 2N - \sum_{1 \leq i \leq k} n_i\right), 2, N\right)$ – обозначим Q. И если Q имеет решение в MULTIPROCESSOR-SCHEDULING , то у нас есть 2 набора в каждом из которых сумма - N. В одном из наборов лежит n_{k+1}, а второй состоит только из исходных элементов - его и берем как ответ. Обратное аналогично 1 пункту

3. Пусть
$$\sum_{1 \leq i \leq k} n_i > 2N$$

Тогда f преобразует $(n_1, n_2, ..., n_k, N)$ в $\left(\left(n_1, n_2, ..., n_k, \sum_{1 \leq i \leq k} n_i - 2N\right),\ 2,\ \sum_{1 \leq i \leq k} n_i - N\right)$ – обозначим Q. И если Q имеет решение в MULTIPROCESSOR-SCHEDULING , то у нас есть 2 набора в каждом из которых сумма - $\sum_{1 \leq i \leq k} n_i - N$. В одном из наборов лежит n_{k+1}. Убираем из этого набора n_{k+1} и там остаются элементы в сумме дающие $\sum_{1 \leq i \leq k} n_i - N - \left(\sum_{1 \leq i \leq k} n_i - 2N\right) = N$. Вот мы и нашли нужное подмножество, которое дает в сумме N. Обратное аналогично 1 пункту

□

NP-полнота доказана.

Эвристический алгоритм

Алгоритм

1) Просортируем задачи по убыванию времени выполнения

2) Будем идти по отсортированному массиву и для задачи выбирать машину, где сейчас меньше всего суммарное время выполнения

Время работы данного алгоритма при использовании кучи для поддержания машины с минимальным временем выполнения - $O(n(\log n + \log m))$. $\log n$ из-за сортировки и $\log m$ из-за того что для каждой задачи нужно выбрать машину из кучи с m элементами.

Теорема 3. *Данный алгоритм дает* $\left(\frac{4}{3} - \frac{1}{3m}\right)$-приближение, где m – количество машин *оптимального расписания.*

Доказательство. Многие идеи доказательства будут взяты из Graham [1969]. Так же в Graham [1969] можно найти пример показывающий что оценка точная и меньше взять нельзя.

Предположим противное, что существуют входные данные $[(T_1, T_2, T_3, ..., T_n), m]$ (сразу же перенумеруем T, так чтобы: $T_1 \geq T_2 \geq ... \geq T_n$), противоречащие условию, то есть если ω_O - оптимальное решение, а ω_L - решение нашим алгоритмом, то $\frac{\omega_L}{\omega_O} > \frac{4}{3} - \frac{1}{3m}$ и из всех таких наборов, возьмем тот у которого n - минимально.

Тогда заметим, что n-ая задача - единственная, которая заканчивается в ω_L в распределении задач нашим алгоритмом. Предположим противное, тогда существует $r < n$. Такая, что r-ая задача заканчивается в ω_L, но тогда оставив только $(T_1, ..., T_r)$ не измениться ω_L, а оптимальное решение не увеличится и значит $\frac{\omega_L}{\omega_O}$ будет больше $\frac{4}{3} - \frac{1}{3m}$ для $(T_1, ..., T_r)$ и это противоречит выбору входных данных с минимальным n.

Значит только n-ая задача заканчивается в ω_L. Теперь пусть τ - время начала выполнения последней задачи в решении алгоритма

$$\omega_L = \tau + T_n$$

$$\sum_{i=1}^{n-1} T_i \geq \tau m$$

Также

$$\omega_O \geq \frac{1}{m} \sum_{i=1}^{n} T_i$$

из того что оптимальное время больше чем время на каждом процессоре, а в сумме это время равно сумме времен всех задач.

$$\frac{\omega_L}{\omega_O} = \frac{\tau + T_n}{\omega_O} \leq \frac{T_n}{\omega_O} + \frac{\sum_{i=1}^{n-1} T_i}{m\omega_O} = \frac{(m-1)T_n}{m\omega_O} + \frac{\sum_{i=1}^{n} T_i}{m\omega_O} \leq 1 + \frac{(m-1)T_n}{m\omega_O}$$

С другой стороны, из предположения: $\frac{\omega_L}{\omega_O} > \frac{4}{3} - \frac{1}{3m}$

Значит

$$1 + \frac{(m-1)T_n}{m\omega_O} > \frac{4}{3} - \frac{1}{3m} = 1 + \frac{1}{3}(1 - \frac{1}{m})$$

$$\frac{(m-1)T_n}{m\omega_O} > \frac{m-1}{3m}$$

$$T_n > \frac{\omega_O}{3}$$

А T_n - задача с минимальным временем выполнения, значит $\forall i : T_i > \frac{\omega_O}{3}$

Значит на каждой машине выполняется не больше 2 задач

Теперь осталось заметить, что если у нас $n2m$, то наш алгоритм будет работать не хуже оптимального, так как первые m задач разложит на каждую машину, а потом $m+i$ задачу будет класть к $m-i+1$ с точностью до задач с одинаковым временем выполнения. Далее от перестановки верхнего слоя ответ не будет улучшаться (если мы меняем верхние задачи, то максимальная сумма увеличивается. Просортируем задачи по убыванию времени выполнения. Так как у нас первый и последняя задача были на одной машине, а 2-ая и 3-яя на другой из алгоритма, а теперь 1-ая и 2-ая будут на одной машине, то ответ не уменьшится). А можно заметить что обязательно есть оптимальное решение, где в нижнем слою будут первые m элементов. Иначе можно прийти к этому не ухудшив решение.

$\square$

При проверки алгоритма на практике он оказался достаточно быстрым и при 10^6 задачах и 10^6 машинах выполнялся в среднем меньше чем за полторы секунды.

Так же при сравнении с алгоритмом перебора, ответ эвристического алгоритма превышал оптимальный ответ не более чем в 1.1 раз.

Библиография

R. L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(2): 416–429, 1969. doi: 10.1137/0117039. URL `https://doi.org/10.1137/0117039`.

Rahul R. Huilgol, Simrat Singh Chhabra, and Shubham Luhadia. Np-completeness of subset-sum problem. 2013. URL `https://www.iitg.ac.in/deepkesh/CS301/assignment-2/subsetsum.pdf`.