



ПРОЕКТ ПО КУРСУ
«СЛОЖНОСТИ ВЫЧИСЛЕНИЙ»

Задача об оптимальном расписании

Студент:
Хорунжий И. Б.

Преподаватель:
Букреев Ф. С.

17 декабря 2022 г.

Содержание

1	Аннотация	2
2	Введение	3
2.1	Историческая справка	3
2.2	Постановка задачи	3
3	NP-полнота	5
3.1	Принадлежность к NP	5
3.2	Язык SUBSETSUM	5
3.3	Сводимость к SUBSETSUM	5
4	Алгоритм	7
4.1	Описание алгоритма	7
4.2	Доказательство $\frac{4}{3}$ приближения алгоритма	7
4.3	Описание наборов тестов	7
4.4	Анализ запусков	7
4.5	Выводы по алгоритму	7
5	Дополнительный материал	8

1 Аннотация

В данной работе исследована задача об оптимальном расписании. Суть ее заключается в построении распределения работ по машинам так, чтобы все работы были выполнены и чтобы конечное время выполнения всех работ было минимально с учетом того, что дано множество работ, машин, и производительность всех машин одинакова. В теоретической части доказана **NP**-полнота языка $\{ (J, m, k) \mid \text{на } m \text{ машинах можно выполнить все работы из } J \text{ за время не более } k \}$ при помощи сводимости к **NP**-полному языку **SUBSETSUM**. Был написан алгоритм, который дает $\frac{4}{3}$ - приближение для случая машин одинаковой производительности, проверен результат работы на различных тестах, сделаны выводы по полученным данным.

2 Введение

2.1 Историческая справка

Теория расписаний — раздел дискретной математики, занимающийся проблемами упорядочения. В общем случае проблемы формулируются так: Задано некоторое множество работ (требований) $J = \{J_1, J_2, \dots, J_n\}$, с определённым набором характеристик: длительность обработки требования (простейший случай), стоимость обработки требования, момент поступления требования, директивный срок окончания обслуживания требования. Задано некоторое множество машин (приборов) $M = \{M_1, M_2, \dots, M_m\}$, на которых требования должны обслуживаться в соответствии с некоторым порядком.

Ставится задача дискретной оптимизации: построить расписание, минимизирующее время выполнения работ, стоимость работ и т. п. Расписание — указание, на каких машинах и в какое время должны обслуживаться требования (выполняться работы).

Задачи теории расписаний можно разделить на две группы:

1. Задачи с прерываниями. В любой момент обслуживание требования на машине может быть прервано (с возможностью завершения позже на той же или другой машине) ради обслуживания другого требования.
2. Задачи без прерываний — каждое требование на машине обслуживается от начала до конца без прерываний.

Существуют различные варианты задач теории расписаний, часть из них является **NP**-полными, часть принадлежит к классу полиномиальных задач, для части задач так и не удалось доказать принадлежности к какому-либо классу сложности. Существует гипотеза, что задача, допускающая прерывания, не сложнее задачи без прерываний. Для большинства задач она соблюдается, кроме одной, где для варианта без прерывания доказана его принадлежность к классу полиномиальных задач, в то время как для аналогичной задачи с прерываниями не существует доказательств принадлежности к какому-либо классу сложности.

2.2 Постановка задачи

В нашем случае будет рассматривать простейший случай задачи построения оптимального расписания, когда у нас дано только время исполнения каждой работы на определенной машине, и каждая задача на машине работает без прерываний. Имеется множество работ J и множество машин M . Также задана функция $p: J \times M \rightarrow \mathbb{R}_+$. 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 $x : J \times M \rightarrow \{0, 1\}$ такую, что:

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

$$\forall i \sum_{j \in M} x_{ij} = 1,$$

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

Далее работа состоит из двух частей:

1. Доказательство, что задача об оптимальном расписании является NP-полной (подразумевается задача поиска расписания длительности не более k).
2. Построение алгоритма, дающего $\frac{4}{3}$ -приближение для случая машин одинаковой производительности, а также его имплементация.

Обозначим язык $\{ (J, m, k) \mid \text{на } m \text{ машинах можно выполнить все работы из } J \text{ за время не более } k \}$ как MULTIPROCESSOR-PLANNING

3 NP-полнота

3.1 Принадлежность к NP

Одним из определением класса **NP** является определение на языке сертификатов.

Definition: Будем говорить, что y является сертификатом принадлежности x языку Q , если существует полиномиальное отношение (верификатор) R , такое что $R(x, y) = 1$ тогда и только тогда, когда x принадлежит Q .

Statement: MULTIPROCESSOR-PLANNING \in NP

Proof: В качестве сертификата рассмотрим вектор размера n чисел от 1 до m , где n - количество работ, m - количество машин. Этот вектор обозначает соответствие всех работ машинам. Верификатор же будет при данном сертификате считать время, затраченное каждой машиной. Если для каждой машины при заданном распределении работ затраченное время не более k , тогда верификатор возвращает 1, иначе - 0. Эта проверка выполняется не более, чем за полиномиальное время.

3.2 Язык SUBSETSUM

Для дальнейшего доказательства **NP** - полноты языка MULTIPROCESSOR-PLANNING нам потребуется язык SUBSETSUM, который определяется следующим образом: $\{(n_1, n_2, \dots, n_k, N) \mid \text{из набора чисел } n_1, \dots, n_k \text{ можно выбрать набор } n_i \text{ с суммой } N\}$

Definition: **NP** - полная задача - в теории алгоритмов задача с ответом «да» или «нет» из класса NP, к которой можно свести любую другую задачу из этого класса за полиномиальное время.

Statement: Задача SUBSETSUM является **NP** - полной задачей.

Proof: Ссылка

3.3 Сводимость к SUBSETSUM

Statement: SUBSETSUM \leq_p MULTIPROCESSOR-PLANNING

Proof: Нам нужно построить функцию сводимости f , чтобы доказать **NP** - полноту MULTIPROCESSOR-PLANNING. Рассмотрим SUBSETSUM как частный случай MULTIPROCESSOR-PLANNING для двух машин. Имеется три варианта:

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

Тогда f задает соответствие из $(n_1, n_2, \dots, n_k, N)$ в $((n_1, n_2, \dots, n_k), 2, N)$. То есть если у нас найдется набор n_i с суммой N , отдадим этот набор задач первой машине, оставшиеся задачи - второй. Тогда каждая отработает за время N . Если же такой набор не нашелся, тогда найдется набор с суммой $\geq N + 1$. Это означает, что время работы на какой-то машине будет $\geq N + 1$, то есть такие задачи не принадлежат языку MULTIPROCESSOR-PLANNING. Значит, если есть решение в SUBSETSUM, то и есть решение в MULTIPROCESSOR-PLANNING, и наоборот.

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

Тогда f действует из $(n_1, n_2, \dots, n_k, N)$ в $((n_1, n_2, \dots, n_k, 2N - \sum_i n_i), 2, N)$. То есть создаем еще одну задачу с оставшимся временем. Повторяем рассуждения из первого пункта: если есть набор n_i с суммой N , его отдаем первой машине. Второй машине приходит набор с оставшимися n_i вместе с дополнительной задачей $2N - \sum_i n_i$. Понятно, что набор таких задач принадлежит языку, так как обе машины отработают за время $\geq N$. Если не нашлось набора с суммой N , делаем выводы, как в пункте 1.

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

Тогда f из $(n_1, n_2, \dots, n_k, N)$ в $((n_1, n_2, \dots, n_k, \sum_i n_i - 2N), 2, \sum_i n_i - N)$. Если есть решение в MULTIPROCESSOR-PLANNING, то у нас есть 2 набора в каждом из которых сумма - $\sum_i n_i - N$. В одном из наборов содержится n_{k+1} . Без него в наборе остаются элементы в сумме дающие $\sum_i n_i - N - (\sum_i n_i - 2N) = N$. Таким образом, мы нашли набор, который дает в сумме N . В случае отсутствия решения в MULTIPROCESSOR-PLANNING выводы аналогичные предыдущим.

NP - полнота MULTIPROCESSOR-PLANNING доказана.

4 Алгоритм

4.1 Описание алгоритма

Построим алгоритм следующим образом:

1. Получим массив задач, отсортированный по убыванию времени их выполнения.
2. Изначально текущее время для всех машин равно нулю.
3. При последовательной итерации по этому массиву задачу отдаем машине, имеющей наименьшую нагрузку, то есть текущее время для этой машины наименьшее.
4. Добавляем этой машине время задачи, которую мы ей отдали.

4.2 Доказательство $\frac{4}{3}$ приближения алгоритма

Доказательство имеется здесь 421 - 426 стр.

4.3 Описание наборов тестов

4.4 Анализ запусков

4.5 Выводы по алгоритму

5 Дополнительный материал

1. Wikipedia - Job-shop scheduling
2. NP-полнота задачи о сумме подмножества
3. Graham - Bounds on multiprocessing timing anomalies