Теория расписаний

Е. В. Щепин

Школа Яндекса по анализу данных
2007

Оглавление

1. Алгоритмы теории расписаний

Задачи одного процессора. Дана совокупность $\mathbb{J}$ из n-заданий, для которых известны времена их исполнения процессором, так что через $|J|$ обозначается время выполнения задания J.

О каждом задании также известны времена их готовности $\underline{D}(J)$ и времена требуемого срока их исполнения $\overline{D}(J)$.

Для каждого задания также задан весовой коэффициент $w(J)$, участвующий в функции-критерии, которую нужно оптимизировать.

Расписание работы процессора определяется заданием порядка выполнения заданий $J_1, J_2, \ldots, J_n$ и времен t_i начала выполнения J_i, удовлетворяющих неравенствам $t_i \geq D_i = D(J_i)$ и $t_i \geq t_{i-1} + T_{i-1} = T(J_{i-1})$.

Время окончания исполнения i-го задания обозначаем t^i. Таким образом $t^i = t_i + T_i$.

Для числа x через x^+ обозначается его положительная часть определяемая формулой $x^+ = \frac{1}{2}(x + |x|)$. Тогда задержка исполнения i-го задания равная $(t^i - \overline{D}_i)^+$ фигурирует во многих функциях-критериях.

В качестве примеров функций-критериев рассмотрим следующие четыре

1) минимизировать суммарный штраф за задержки

$$f_1 = \sum_{k=1}^{m-1} w(J_k)(t^k - \overline{D}_k)^+$$

2) минимизировать максимальный штраф за задержки

$$f_2 = \max_k w(J_k)(t^k - \overline{D}_k)^+$$

3) минимизировать сумму штрафов за пролеживание

$$f_3 = \sum_{k=1}^{n-1} w(J_k)(t^k - \underline{D}_k)$$

4) минимизировать сумму связанных средств

$$f_4 = \sum_{k=1}^{n-1} w(J_k)t^k$$

Решение по критерию f_4

Мы рассмотрим решение для функции f_4 только в случае $\underline{D}(J) = 0$ при любом $J \in \mathbb{J}$. Рассмотрим оптимальную последовательность и переставим в ней два соседних задания J_i и J_{i+1}. При этом время исполнения последнего из них может только возрасти (иначе рассматриваемое решение не оптимально). Но разность функции-критерия между исполнением первых $i+1$ заданий у модифицированного и оптимального порядков не превосходит, очевидно,

$$(w(J_{i+1})|J_{i+1}| + w(J_j)(|J_i| + |J_{i+1}|)) - (w(J_i)|J_i| + w(J_{j+1})(|J_i| + |J_{i+1}|)) \geq 0$$

Откуда после сокращений получаем

$$w(J_i)|J_{i+1}| \geq w(J_{i+1})|J_i|$$

Следовательно для оптимального расписания для любого i получаем неравенство отношений

$$\frac{|J_i|}{w(J_i)} \leq \frac{|J_{i+1}|}{w(J_{i+1})} \tag{1.1}$$

Заметим, что при равенстве этого отношения перестановка i-ой и $(i+1)$-ой работ не меняет значения критерия. Поэтому любое расписание (плотное) в котором работы расставлены в порядке неубывания отношений $\frac{|J_i|}{w(J_i)}$ будет f_4-оптимальным.

Решение по критерию f_2

Мы рассмотрим решение для функции f_2 также только в случае $\underline{D}(J) = 0$ при любом $J \in \mathbb{J}$. Рассмотрим оптимальную последовательность и переставим в ней последнее задание J_n и предпоследнее J_{n-1}. При этом, очевидно, значение функции критерия не возрастет, если выполнено неравенство

$$w(J_{n-1})(T - \overline{D}^{n-1})^+ \leq w(J_n)(T - \overline{D}^n)^+$$

Аналогочные соображения применимы к частичным расписанием, поэтому серией перестановок, не увеличивая значения критерия f_2, можно выстроить задания таким образом, чтобы последним исполнялось то, для которого минимально значение

$$w(J)(T - \overline{D}(J))$$

Таким образом определилось последнее задание оптимального расписания. Аналогично определяются предпоследнее и так далее до самого начала задания.

Паросочетание. Задача паросочетания может быть сформулирована следующим образом: Даны n множеств $S_1, \dots S_n$ нужно в каждом из них выбрать по элементу $x_i \in S_i$ так что $x_i \neq x_j$ при $i \neq j$.

Назовем семейство множеств *избыточным*, если количество элементов в объединении этого семейства меньше чем число множеств, входящих в это семейство. Ясно, что задача паросочетания для избыточного семейства неразрешима. Действительно, ведь все выбранные элементы принадлежат объединению, все они различны, поэтому количество выбранных элементов равно количеству множеств. Задача паросочетания неразрешима также и для неизбыточного семейства в том случае, когда это семейство содержит избыточное подсемейство. Как мы увидим далее при отсутствии избыточных подсемейств задача паросочетания разрешима.

Простейший алгоритм построения паросочетания выглядит так: В качестве x_1 берем первый элемент S_1 (множество всех элементов предполагаем линейно упорядоченным). Если уже выбраны элементы $x_i \in S_i$ для $i < k$, то в качестве x_k выбираем первый свободный (то есть не выбранный ранее) элемент S_k.

Если этот алгоритм отработает до конца, то нужное паросочетание будет построено. Но может случиться, что в S_k не окажется ни одного свободного элемента. В этом случае следует провести ревизию и изменить выбор.

Обозначим через C текущую функцию выбора. А именно, для множества S_i через $C(S_i)$ обозначается выбранный для него элемент и для элемента x_i через $C^{-1}(x_i)$ обозначается множество, которым он избран.

Ревизию можно проводить следующим образом: берем первый элемент $x_{k1} \in S_k$ и рассматриваем множество $S_{k1} = C^{-1}x_{k1}$, которым он избран. Если в этом множестве имеются свободные элементы, то избираем для него первый свободный, а освободившийся первый элемент S_k выбираем в качестве $C(S_k)$. Если же в S_{k1} свободных элементов нет, то рассматриваем второй элемент $x_{k2} \in S_k$ и ищем свободные элементы в избравшем его множестве S_{k2}. Если таковые найдутся, то для S_{k2} избираем свободный, а освободившийся второй элемент назначаем в качестве $C(S_k)$. Если свободного элемента не найдется в S_{k2}, то переходим к третьему элементу S_k и повторяем для него ту же процедуру. Так мы делаем до тех пор пока не переберем все элементы S_k.

Если же эта процедура — первичная ревизия — не даст результата, то есть ни одно из множеств $C^{-1}S_k$ не содержит свободных элементов, то нужно делать вторичную ревизию: просмотреть все элементы объединения $\cup C^{-1}S_k$, с целью обнаружить свободный элемент в $C^{-1}(\cup C^{-1}S_k)$. Если мы такой элемент $x \in S \in C^{-1}(\cup C^{-1}S_k)$ сумеем обнаружить, то мы переопределим функцию выбора полагая $C(S) := x$. Тогда ранее избранный для S элемент y освобождается, а так как $y \in C^{-1}z$, для некоторого $z \in S_k$, то мы можем выбрать y для $C^{-1}z$ вместо z и освободившийся z избрать для S_k.

Если и вторичная ревизия не даст результата, то нужно провести третичную, проведя поиск свободных элементов в $C^{-1}(\cup C^{-1}(\cup C^{-1}S_k))$ и так далее.

Если число элементов объединения $\cup_{i=1}^{k} S_i$ равно m, то мы утверждаем, что если m-ичная ревизия не даст результата, то паросочетание невозможно. Определим по индукции множество S_k^i так что $S_k^0 = S_k$ и $S_k^{i+1} = \cup C^{-1}S_k^i$. И определим по индукции семейство множеств $\tilde{S}_k^i$ так что $\tilde{S}_k^0$ состоит из одного элемента S_k и $\tilde{S}_k^{i+1} = \tilde{S}_k^i \cup C^{-1}S_k^i$. Заметим, что $\cup\tilde{S}_k^j = S_k^j$. Через n_i обозначим количество элементов разности $S_k^i \setminus S_k^{i-1}$ и обозначим через n_0 количество элементов S_k. В таком случае $n_0 + n_1 + \ldots n_m$ равняется числу элементов S_k^m.

Количество множеств в разности семейств $\tilde{S}_k^{i+1} \setminus \tilde{S}_k^i$ равняется числу элементов в $S_k^i \setminus S_k^{i-1}$, если последняя не содержит свободных элементов. Таким образом в случае неудачи всех попыток ревизии количество элементов семейства $\tilde{S}_k^i$ выражается суммой $1 + n_0 + n_1 + \ldots n_{i-1}$.

А так как количество точек в S_k^j ограничено сверху, с ростом j оно стабилизируется. Следовательно, для некоторого j будет $n_j = 0$. Если до этого момента ревизия не нашла свободных элементов, то мы получим, что семейство $\tilde{S}_k^j$ содержит множеств $1 + \cdots + n_{j-1}$ больше чем S_k^j имеет элементов. То есть получаем избыточность семейства $\tilde{S}_k^j$ и невозможность построения паросочетания.

Предложенный выше алгоритм решает задачу паросочетания, но не слишком быстр. Чтобы сделать быстрый алгоритм введем дополнительный целочисленный массив порядков свободы. Для каждого свободного

элемента его *порядок свободы* полагается равным 1. И если уже определены элементы порядка свободы k, то элемент x порядка свободы $k+1$, определяется, как не имеющий порядка свободы $\leq k$, но для которых $C^{-1}x$ содержит элемент порядка свободы k. Элементы не имеющие никакого положительного порядка свободы имеют нулевой порядок свободы. Почитать порядки свободы всех элементов можно за время $O(m^2)$, где m — число точек в $\cup S_i$. Для этого нужно сделать двойной цикл: внешний - по порядкам свободы, внутренний по точкам множества.

Располагая массивом порядков свободы, очередной выбор элемента из S_k можно сделать следующим образом: найти элемент из S_k с наименьшим положительным порядком свободы. Это займет время $O(m)$. Если таких элементов нет, то паросочетание невозможно, как было доказано выше и в этом случае работу алгоритма следует прекратить. Для выбранного элемента x порядка свободы i построить цепочку $x_1 \dots x_i = x$ такую что x_1 — свободен и $x_j \in C^{-1}x_{j-1}$ для любого $j \leq i$. Это займет время $O(m)$. Переопределить функцию выбора, полагая $CS_j = x_{j-1}$, (вместо $CS_j = x_j$) (время $O(m)$) и, наконец, пересчитать порядки свободы элементов. Этот пересчет займет $O(m^2)$ времени. Таким образом весь алгоритм в целом отработает за $O(m^3)$ времени. Можно сделать лучше за $O(m^2)$. Придумайте как.

Задачи.

(1) Пусть M_i семейство подмножеств множества M. Построить алгоритм, который выбирает в каждом M_i по одному элементу таким образом, чтобы все элементы M оказались выбраны.
(2) Пусть M_i семейство подмножеств множества M. Построить алгоритм, который выбирает в каждом M_i ровно 2 различных элемента, так что выбранные для разных i множества не пересекаются.
(3) Пусть M_i семейство подмножеств множества M. Построить алгоритм, который выбирает в каждом M_i по одному элементу таким образом, чтобы все элементы M оказались выбраны 2 раза.
(4) Пусть M_i семейство подмножеств множества M. Построить алгоритм, который выбирает в каждом M_i по одному элементу таким образом, что элементы M оказались выбраны не более двух раз.

2. NP-полные задачи

Классы P и NP Алгоритм, время работы которого оценивается сверху некоторым многочленом от длины условий задачи, называется полиномиальным. Например, рассмотренная на предыдущей лекции задача паросочетания решается алгоритмами время работы которых оценивается многочленом третьей степени от длины входа.

Класс задач, допускающих решение в виде полиномиального алгоритма, обозначается буквой P. Класс P не очень широк, намного шире его следующий класс NP, который образован задачами распознавания (с ответом да или нет) положительный ответ, на которые может быть проверен с помощью полиномиального алгоритма.

Для булевой переменной x (то есть переменной принимающей значения 0 и 1) и показателя d принимающего значения $0, 1, -1$ определим

$$x^d = \begin{cases} x, & \text{если } d = 1; \\ 0, & \text{если } d = 0; \\ 1 - x, & \text{если } d = -1. \end{cases} \tag{2.1}$$

Например, к классу NP относится следующая задача ВЫПОЛНИМОСТЬ:

дана булева формула вида

$$\bigwedge_{j=1}^{m} \bigvee_{i=1}^{n} x_i^{d(i,j)} \tag{2.2}$$

(здесь x_i обозначает булеву переменную, $d(i, j)$ принимает значения $0, 1, -1$) нужно определить принимает ли эта формула значение 1 для какого-нибудь значения набора переменных x_i ?

Теорема Кука

Кук доказал универсальность задачи ВЫПОЛНИМОСТЬ для класса NP. А именно, указал как построить за полиномиальное время для любой задачи класса NP булеву формулу вида (2.2), решения которой однозначно соответствуют ответам этой задачи. Это свойство задачи ВЫПОЛНИМОСТЬ получило название NP-полноты. В последствии была доказана NP-полнота многих известных задач. Доказательство NP-полноты задачи по существу означает несуществование никакого неперебороного алгоритма ее решения.

3-выполнимость

Для любого дизъюнкта $\bigvee_{i=1}^{n} x_i^{d(i)}$ введем $n-3$ новых булевых переменных $y_1, \ldots, y_{n-3}$ и определим последовательность из $n-2$ дизъюнктов

$$\begin{cases} x_1^{d(1)} + x_2^{d(2)} + y_1 \\ y_1^{-1} + x_3^{d(3)} + y_2 \\ y_2^{-1} + x_4^{d(4)} + y_3 \\ \cdots\cdots \\ y_{n-3}^{-1} + x_{n-1}^{d(n-1)} + x_n^{d(n)} \end{cases}$$

(здесь операция дизъюнкции записана в форме сложения). Очевидно, что конъюнкция вышеописанных дизъюнктов выполнима относительно

новых переменных для тех и только тех наборов x_i, для которых истинным является значение первоначального дизъюнкта. Таким образом замена дизънкта в булевой формуле (2.2) совокупностью дизъюнктов от трех переменных дает эквивалентную исхолдной задачу. Поэтому задача 3-выполнимость, в которой все дизъюнкты содержат ровно три слагаемых также является NP-полной.

Трехмерное сочетание

Даны три множества U, V, и W одинаковой мощности и подмножество T множества троек $U \times V \times W$. Спрашивается существует ли в T такое подмножество M мощности одинаковой с U, V, и W , что различные тройки из M не имеют общих элементов ни в одном из множителей.

Для доказательства NP-полноты задачи трехмерного сочетания мы построим полиномиальную редукцию к ней задачи ВЫПОЛНИМОСТЬ. Пусть F булева формула содержащая переменные $x_1, \ldots, x_n$ и дизъюнкты $C_1 \ldots, C_m$. По этим данным мы построим вход задачи трехмерное сочетание положительное решение которой существует только если выполнима формула F и построенное сочетание порождает в полиномиальное время набор значений булевых переменных, для которого F истинна.

Множество U содержит $2nm$ элементов обозначаемых $\{\pm x_i^j\}$, $i = 1, \ldots, n$, $j = 1, \ldots, m$

Множество V для любого $j = 1, \ldots, m$ содержит $2n$ элементов трех типов $\{a_i^j : i = 1, \ldots, n; v_j; c_i^j : i = 1, \ldots, n-1\}$

Аналогичную структуру имеет множество $W = \cup_{j=1}^{m}\{b_i^j : i = 1, \ldots, n; w_j; d_i^j : i = 1, \ldots, n-1\}$

И множество T является объединением троек следующих трех типов

1) Тройки $\{a_i^j, b_i^j, +x_i^j\}$ и $\{a_{i+1}^j, b_i^j, -x_i^j\}$ где $a_i^{m+1} = a_i^1$ и $j = 1, \ldots, m$, $i = 1, \ldots, n$. Вершины типа a и b не входят больше ни в какие тройки.

2) Второй тип троек образуют множества $v_j, w_j, \pm x_i^j$, где $d(i,j) \neq 0$. Вершины типа v, w не входят ни в какие другие тройки.

5) Третий тип троек образуют тройки вида c_i^j, d_i^j, x, где x произвольный элемент U.

Предположим нам удалось найти полной тройное сочетание M. Тогда для каждого i у нас в тройки первого типа взяты или все $+x_i^j$ или все $+x_i^j$. В первом случае мы присваиваем x_i значение ложь, во втором — истина. Тогда исходная булева формула будет выполнена при этих значениях переменных. Действительно, все ложные переменные разобраны тройками первого типа поэтому тройки второго типа указывают в каждом дизъюнкте истинную переменную. Следовательно, коъюнкция дизъюнктов также истинна.

Обратно, если булева формула выполнима, то полное тройное сочетание M можно построить выбирая те тройки первого типа $a_i^{j+x_i}, b_i^j, (-1)^{x_i} x_i^j$ и в тройки второго типа забираем для каждого j такое x_i^j, для которого истинным является x_i.

3-покрытие

Следующая NP-полная задача называется ТОЧНОЕ 3-ПОКРЫТИЕ:

Дано покрытие множества Φ *его трехэлементными подмножествами* $\{F_1, \ldots, F_n\}$. *Выделить из него подпокрытие, состоящее из попарно непересекающихся подмножеств.*

NP-полнота задачи ТОЧНОЕ 3-ПОКРЫТИЕ доказывается сведением к ней задачи ТРЕХМЕРНОЕ СОЧЕТАНИЕ. А именно, пусть U,V,W,T представляет собой вход задачи ТРЕХМЕРНОЕ СОЧЕТАНИЕ. Положим $\Phi = U \sqcup V \sqcup W$ и определим элементы покрытия F_i как образы троек из T при естественной проекции прямого произведения на прямую сумму. Тогда, очевидно, прообразом точного подпокрытия будет полное трехмерное сочетание и наоборот.

Рюкзак

Даны целые числа c_j, $j = 1, \ldots, n$ и K. Спрашивается существуют ли такие целые числа x_j, такие что

$$\sum_{i=1}^{n} c_j x_j = K \tag{2.3}$$

Сформулированная выше задача называется ЦЕЛЫЙ РЮКЗАК и является NP-полной. Если переменные x_i в этой задаче булевы (то есть принимают значения 0 или 1) то задача называется БУЛЕВ РЮКЗАК.

Мы докажем здесь NP-полноту БУЛЕВА РЮКЗАКА, сведя к нему задачу 3-ПОКРЫТИЕ. Пусть $\Phi, F_1, \ldots, F_n$ представляет собой вход задачи 3-ПОКРЫТИЕ. Пусть занумеруем элементы множества Φ натуральными числами. $\Phi = \{\phi_1 \ldots \phi_{3m}\}$. Каждому множеству $F_i = \{\phi_{i_1}, \phi_{i_2}, \phi_{i_1}\}$ поставим в соответствие следующее число

$$c_i = (m+1)^{i_1} + (m+1)^{i_2} + (m+1)^{i_3}$$

и положим $K = \sum_{k=1}^{m-1} (m+1)^k$. Предположим x_i дают булево решение уравнения рюкзака (2.3), тогда соответствующие F_i дают точное покрытие в силу следующего простого замечания

Лемма 2.1. *Если натуральные* y_k *и* z_k *не превосходят* m, *то равенство*

$$\sum_{k=1}^{m-1} y_k (m+1)^k = \sum_{k=1}^{m-1} z_k (m+1)^k$$

влечет равенства $z_k = y_k$ *при любом* k.

Доказательство. Предположжим противное. Пусть m наибольшее число для которого $y_n \neq z_n$. Пусть для определенности $y_n > z_n$. Тогда

$$(y_n - z_n) = \sum_{k=1}^{n-1-1} (z_k - y_k)(m+1)^{k-n}$$

Правая часть по абсолютной величине не превосходит $m \sum_{k=1}^{n-1-1} (m+1)^{k-n} <$ 1, что противоречит целочисленности $y_n - z_n$. $\square$

Разбиение

Построению оптимальных расписаний полиномиальными алгоритмами нередко препятствует NP-полнота следующей задачи РАЗБИЕНИЕ

Для данных целых чисел $c_1, \ldots, c_n$ выяснить существует ли подмножество $S \subset \{1, 2, \ldots, n\}$, такое что $\sum_{j \in S} c_j = \sum_{j \notin S} c_j$

Доказательство NP-полноты РАЗБИЕНИЯ основано на редукции к нему БУЛЕВА РЮКЗАКА. Пусть $c_1, \ldots, c_n$, K представляет собой вход задачи БУЛЕВ РЮКЗАК. Рассмотрим задачу РАЗБИЕНИЕ со входом $c_1, \ldots, c_n$, $c_{n+1} = 2M$, $c_{n+2} = 3M - 2K$, где $M = \sum_{k=1}^{n-1} c_k > K$. Утверждается, что в множестве $\{1, 2, \ldots, n\}$ существует подмножество $S(K)$ такое что $\sum_{j \in S(K)} c_j = K$ в том и только том случае, когда в множестве $\{1, 2, \ldots, n, n+1, n+2\}$ существует подмножество S решающе задачу РАЗБИЕНИЕ.

Действительно, если к $S(K)$ добавить $n + 2$, то получим искомое S. Наоборот по данному S в качестве $S(K)$ берем $S \cap [1, n]$, если $n + 2 \in S$ и берем $S(K) = [1, n] \setminus S$ в противном случае.

Задачи.

(1) Доказать NP-полноту задачи построения расписания выполнения независимых заданий процессором за указанное время, если скорость работы процессора удваивается спустя некоторое (заранее известное) время.
(2) Доказать NP-полноту задачи построения расписания выполнения независимых заданий на двух идентичных процессорах за указанное время, если один из них начинает работу спустя некоторое (заранее известное) время после другого.
(3) Доказать NP-полноту задачи построения расписания выполнения независимых заданий на двух идентичных процессорах за указанное время, если один из них заканчивает работу спустя некоторое (заранее известное).
(4) Доказать NP-полноту задачи построения расписания выполнения независимых заданий парой процессорах за указанное время, если один из них в два раза быстрее другого.
(5) Доказать NP-полноту задачи распределения независимых заданий между тремя идентичными процессорами так чтобы загрузка каждого не превышапа указанного времеми, если задания можно произвольно делить между первыми двумя процессорами.
(6) Доказать NP-полноту задачи построения расписания выполнения независимых заданий на двух идентичных процессорах за указанное время, если один из них начинает работу сразу после того как другой завершит выполнение первого задания.

(7) Доказать NP-полноту задачи распределения независимых заданий между тремя идентичными процессорами так чтобы загрузка каждого не превышапа указанного времеми, если только одно (любое) задание разрешается произвольно разделить между двумя (любыми) процессорами.