ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
УЧРЕЖДЕНИЕ НАУКИ ВЫЧИСЛИТЕЛЬНЫЙ
ЦЕНТР ИМ. А.А. ДОРОДНИЦЫНА
РОССИЙСКОЙ АКАДЕМИИ НАУК

НЕКОТОРЫЕ АЛГОРИТМЫ ПЛАНИРОВАНИЯ ВЫЧИСЛЕНИЙ В МНОГОПРОЦЕССОРНЫХ СИСТЕМАХ

ВЫЧИСЛИТЕЛЬНЫЙ ЦЕНТР ИМ. А.А. ДОРОДНИЦЫНА
РОССИЙСКОЙ АКАДЕМИИ НАУК
МОСКВА 2012

УДК 519.86

Ответственный редактор
канд. физ.-матем. наук Л.Л. Вышинский

Сборник посвящен решению некоторых дискретных задач, возникающих при разработке многопроцессорных систем. В частности, в сборнике получены следующие результаты:

– разработан алгоритм построения многопроцессорного расписания без прерываний с минимальным числом процессоров для 3-несовместимой задачи составления расписаний;

– разработан приближенный алгоритм составления оптимального расписания для случая, когда часть работ допускает прерывания и переключения, а часть – не допускает;

– разработан эвристический алгоритм решения задачи оптимизации структуры базы данных;

– разработан метод ветвей и границ для построения оптимального по быстродействию расписания без прерываний.

Ключевые слова: многопроцессорная система, оптимальное расписание, 3-несовместимая задача, прерываемые и непрерываемые работы, метод ветвей и границ.

Рецензенты: *Я.И. Рабинович,*
Д.В. Денисов

Научное издание

СУЩЕСТВОВАНИЕ ЗАДАЧ СОСТАВЛЕНИЯ РАСПИСАНИЙ С ЗАДАННЫМ МИНИМАЛЬНЫМ ЧИСЛОМ ПРОЦЕССОРОВ

В.П. Козырев

Доказывается, что для любого числа процессоров и любого числа работ существуют задачи составления расписаний с данным числом работ и реализованных в точности на M процессорах.

1. Основные определения

Пусть имеется n (вычислительных) работ, для каждой из которых заданы директивные сроки выполнения $[a_i,b_i]$ и время выполнения $b_i-a_i \ge \tau_i > 0$. Каждая работа на одном процессоре выполняется без прерывания и на любом процессоре в любой момент времени выполняется не более одной работы. Требуется определить минимальное число процессоров $N(Z)$ для выполнения всех работ $Z=\{[a_i,b_i],\tau_i\}$ и порядок следования (т.е. расписание) их на процессорах.

Вводим два вспомогательных параметра для каждой работы: $\alpha_i = a_i + \tau_i$ – самое ранее время выполнения работы v_i $(1 \le i \le n)$, $\beta_i = b_i - \tau_i$ – самое позднее начало ее выполнения [1]. На множестве пар работ v_i, v_j определяем три типа бинарных отношений: 1) v_i, v_j – 2-совместимы, если эти работы могут быть выполнены на одном процессоре в любом порядке; 2) v_i, v_j – 2-несовместимы, если они не могут быть выполнены на одном процессоре ни в каком порядке; 3) v_i, v_j – 1-совместимы, если они могут быть выполнены на одном процессоре только в одном порядке – работа v_j после работы v_i. Тогда выполняются соотношения: 1) $\alpha_i \le \beta_j, \alpha_j \le \beta_i$, 2) $\alpha_i > \beta_j, \alpha_j > \beta_i$, 3) $\alpha_i \le \beta_j, \alpha_j > \beta_i$. Назовем такие пары ребром (i, j), неребром $\overline{(i, j)}$ и дугой $\overrightarrow{(i, j)}$ соответственно.

Будем рассматривать графы, содержащие только ребра и неребра. Чередующейся цепью (кратко П-цепью) назовем последовательность ребер и неребер, в которой за каждым ребром следует неребро, а за каждым неребром следует ребро. Граф, не содержащий замкнутых П-цепей, будем называть П-графом. Ранее было доказано, что любой П-граф G представляется набором $(n_1, n_2,\ldots, n_k)$, где $n_1 = |V_1|$, V_1 – множество вершин (т.е. работ) степени $(n–1)$ или 0, множество V_2 состоит из вершин степени 0 или $(n–n_1–1)$ в графе $G\backslash V_1$, множество V_3 – из вершин степени $(n–n_1–n_2–1)$ или 0 в графе $G\backslash V_1\backslash V_2$ и т.д. Обозначим через $G(1; n_1, n_2,\ldots, n_k)$ – П-граф, у которого вершины из V_1 имеют степень $(n–1)$, а через $G(0; n_1, n_2,\ldots, n_k)$ – П-граф, у которого вершины из V_1 имеют степень 0; эти графы взаимно дополнительные. Ниже будут рассмотрены простейшие нетривиальные задачи Z, у которых П-графами являются $G(1; n_1, n_2)$ и $G(0; n_1, n_2)$, причем $n = n_1+n_2$ – фиксированное число и в расписаниях используется минимальное число процессоров, равное фиксированному числу M.

2. Основные утверждения

Пусть K_n – задача Z, у которой любая пара является 2-совместимой, а $\overline{K_n}$ – задача Z, у которой каждая пара является 2-несовместимой. Показано [1], что $1 \le N(K_n) \le \lceil n/2 \rceil$ и $N(\overline{K_n}) = n$. Предельными значениями являются числа $1 \le N(Z) \le n$. В данной работе доказывается, что для любого целого числа M такого, что $1 \le M \le n$ существуют задачи Z, у которых $N(Z) = M$.

Задача $Z=\{[a_i,b_i],\tau_i\}$ называется 3-несовместимой, если любые три различные работы не могут быть выполнены на одном процессоре. Так, если для Z выполняются неравенства

$\{\tau_i + \tau_j\} \le \{b_p - a_q\} < \{\tau_r + \tau_l + \tau_t\}$ для любых $i \ne j$, $p \ne q$, $r \ne l \ne t$, $i, j, p, q, r, l, t \in \{1, 2, ..., n\}$, то Z является 3-несовместимой задачей составления расписаний. Из определения 3-несовместимости вытекает, что в любом расписании такой задачи Z на каждый используемый процессор назначается не более двух работ. Если все пары работ задачи Z являются 2-несовместимыми, то эта задача является и 3-несовместимой.

Утверждение 1. 3-несовместимая задача Z с П-графом $G(1; n_1, n_2)$, у которого $n_1 < n_2$, $n_1 + n_2 = n$, $n_2 = M$, обладает свойством $N(Z) = M$.

Доказательство. Так как граф с множеством верши V_2, $|V_2| = M$ состоит из 2-несовместимых пар, то M является нижней оценкой для $N(Z)$. Построим расписание с использованием M процессоров. Вводим n_1 пар (v_i, v_j), $i \in V_1$, $j \in V_2$; на каждую пару затрачивается один процессор и в любой такой процессор нельзя добавить ни одну работу в силу 3-несовместимости Z. На оставшиеся работы из $V_2 \backslash V_1$ затрачивается $M - n_1$ процессоров, а на два множества – $n_1 + M - n_1 = M$ процессоров.

Утверждение 2. 3-несовместимая задача Z с П-графом $G(0; n_1, n_2)$, у которого $n_1 = 2M - n$, $n_2' = n - M$ (для $n_2 = 2n_2'$) или $n_1 = 2M - n$, $n_2' = n - M - 1$ (для $n_2 = 2n_2' + 1$) обладает свойством $N(Z) = M$.

Доказательство. В силу 3-несовместимости

$$N(Z) = N(G) = n_1 + N(G(1; n_2)) = n_1 + \lceil n_2 / 2 \rceil.$$

Пусть $n_2 = 2n_2'$, тогда $N(Z) = n_1 + n_2' = (2M - n) + (n - M) = M$.

Пусть $n_2 = 2n_2' + 1$, тогда

$$N(Z) = n_1 + n_2' + 1 = (2M - n) + (n - M - 1) + 1 = M.$$

Свойство 3-несовместимости можно заменить на свойство наличия 1-процессорных расписаний [1] на множестве с $n_1 - n_2$ вершинами (работами), когда $n_1 > n_2$, и тогда $N(Z) = M = 1 + n_2$. Если $n_1 \le n_2$, то $N(Z) = n_2 = M$ для графа $G(1; n_1, n_2)$. Для

$G(0; n_1, n_2)$ имеем при $n_1 \ge n_2$ $N(Z) = M = n_1$, а при $n_1 < n_2$ $N(Z) = M = n_1 + 1$.

Литература

1. *Козырев В.П.* Составление многопроцессорных расписаний для 2-совместимых и 2-несовместимых работ // Проблемы теоретической кибернетики. Тезисы докл. XIII межд. конф. (Казань, 31 мая 2002 г.), часть I. М.: Мехмат МГУ. С. 89.

2. *Козырев В.П.* Характеризация графов, не имеющих чередующихся циклов // Тезисы докладов научной конференции «Математические модели сложных систем и междисциплинарные исследования» (23-24 октября 2002 г.). М.: ВЦ РАН. С. 33.

АЛГОРИТМ НАХОЖДЕНИЯ МИНИМАЛЬНОГО ЧИСЛА ПРОЦЕССОРОВ ДЛЯ 3-НЕСОВМЕСТИМОЙ ЗАДАЧИ СОСТАВЛЕНИЯ РАСПИСАНИЙ

В.П. Козырев, Е.А. Соколова

Описан алгоритм построения многопроцессорного расписания без прерываний с минимальным числом процессоров для 3-несовместимой задачи составления расписаний.

1. Основные определения

Пусть имеется n (вычислительных) работ, для каждой из которых заданы директивные сроки выполнения $[a_i,b_i]$ и время выполнения $b_i - a_i \geq \tau_i > 0$. Каждая работа на одном процессоре выполняется без прерывания и на любом процессоре в любой момент времени выполняется не более одной работы. Требуется определить минимальное число процессоров $N(Z)$ для выполнения всех работ $Z = \{[a_i, b_i], \tau_i\}$ и порядок следования (т.е. расписание) их на процессорах.

Тройка работ называется 3-несовместимой, если данные работы не могут быть выполнены на одном процессоре ни в каком порядке. Задача Z называется 3-несовместимой, если любые три различные работы являются 3-несовместимыми. Рассмотрим 3-несовместимую задачу $Z = \{[a_i, b_i], \tau_i\}$, в которой любая пара работ является либо 2-совместимой, либо 2-несовместимой. Для данной задачи требуется построить допустимое (относительно директивных сроков) расписание с использованием минимального числа процессоров. Из определения 3-несовместимости вытекает, что в любом расписании такой задачи Z на каждый используемый процессор назначается не более двух работ, т.е. $N(Z) \geq \lceil n/2 \rceil$. С другой стороны

все 2-несовместимые работы, должны выполняться на различных процессорах, т.е. число 2-несовместимых работ является оценкой снизу на $N(Z)$. Т.е. $N(Z) \geq \max\{\lceil n/2 \rceil, p\}$, где p – число 2-несовместимых работ в задаче.

Введем несколько вспомогательных определений из теории графов. Чередующейся цепью (кратко П-цепью) назовем последовательность ребер и неребер, в которой за каждым ребром следует неребро, а за каждым неребром следует ребро. Граф, не содержащий замкнутых П-цепей, будем называть П-графом. Ранее было доказано [2], что любой П-граф G с n вершинами представляется набором $(n_1,n_2,\ldots,n_k)$, где $n_1+n_2+\ldots+n_k = n$, $n_i = |V_i|$, а вершины множества V_{2i-1} либо смежны, либо не смежны со всеми вершинами графа $G\backslash\{V_1 \ldots V_{2i-2}\}$, вершины множества V_{2i} либо не смежны, либо смежны со всеми вершинами графа $G\backslash\{V_1 \ldots V_{2i-1}\}$ соответственно. Обозначим через $G(1;n_1,n_2,\ldots,n_k)$ – П-граф, у которого вершины из V_1 имеют степень $(n-1)$, а через $G(0;n_1,n_2,\ldots,n_k)$ – П-граф, у которого вершины из V_1 имеют степень 0; эти графы взаимно дополнительные. В работе [1] было показано, что задаче Z, в которой любая пара работ является либо 2-совместимой, либо 2-несовместимой, можно поставить в соответствие один из П-графов $G(1;n_1,n_2,\ldots,n_k)$ или $G(0;n_1,n_2,\ldots,n_k)$, вершинами которого является множество работ, и две вершины смежны тогда и только тогда, когда соответствующая им пара работ является 2-совместимой.

2. Алгоритм

Пусть рассматриваемой задаче Z соответствует П-граф $G(0;n_1,n_2,\ldots,n_k)$, т.е. множество V_1 – множество 2-несовместимых работ. По определению 2-несовместимых работ на их выполнение необходимо n_1 процессоров, причем на данных процессорах могут быть выполнены только указанные работы. Тогда множеству оставшихся работ можно сопоставить П-граф $G(1;n_2,\ldots,n_k)$ и $N(Z) = N(G(0;n_1,n_2,\ldots,n_k)) = n_1 + N(G(1;\ n_2,$

…, n_k)). Теперь разберем подробно случай, когда задаче Z соответствует П-граф $G(1;n_1,n_2,\ldots,n_k)$. Введем обозначение, $f(1;n_1,n_2,\ldots,n_k)$ – минимальное число процессоров необходимое для составления расписания для задачи, которой соответствует П-граф $G(1;n_1,n_2,\ldots,n_k)$, аналогично определяется $f(0;n_1,n_2,\ldots,n_k)$ для П-графа $G(0;n_1,n_2,\ldots,n_k)$. Составление расписания будем осуществлять с помощью редукции. Рассмотрим П-граф $G(1;n_1,n_2,\ldots,n_k)$, так как работы множества V_2 совместимы только с работами множества V_1, то для экономного размещения работ следует на процессоры, выполняющие работы из V_2, назначать работы из V_1. И в силу 3-несовместимости, на такой процессор может быть назначено не более одной работы из V_1. Таким образом, если $n_1 \le n_2$, то выделяем n_1 процессоров, на которые назначаем две работы: по одной из каждого множества V_1, V_2. Тогда множество работ V_1 полностью исчерпано, а в множестве V_2 остается $(n_2–n_1)$ 2-несовместимых работ, на выполнение которых необходимо выделить $(n_2–n_1)$ процессоров, и другие работы не могут быть назначены на эти процессоры. Выпишем $N(Z)$:

$$N(Z) = f(1;n_1,\ldots,n_k) = n_1 + f(0;n_2 - n_1,n_3,\ldots,n_k) =$$

$$= n_1 + n_2 - n_1 + f(1;n_3,\ldots,n_k) = n_2 + f(1;n_3,\ldots,n_k).$$

Для случая, когда $n_1 > n_2$, выделяем n_2 процессора, на которые назначаем две работы: по одной из каждого множества V_1, V_2. Теперь множество работ V_2 полностью исчерпано, а в множестве V_1 остается $(n_1–n_2)$ 2-совместимых работ, которые объединяем с работами V_3, так как множество V_2 полностью исчерпано. Выпишем $N(Z)$:

$$N(Z) = f(1;n_1,\ldots,n_k) = n_2 + f(1;n_1 - n_2,0,n_3,\ldots,n_k) =$$

$$= n_2 + f(1;n_1 - n_2 + n_3,n_4,\ldots,n_k).$$

Таким образом перешли к рассмотрению задачи составления расписания меньшей размерности, которой соответствуют П-графы $G(1;n_3,\ldots,n_k)$ и $G(1;n_1–n_2+n_3,\ldots,n_k)$, для случаев $n_1 \le n_2$ и $n_1 > n_2$ соответственно. Используя данную редукцию, можно построить дерево (табл. 1), описывающее все расписания для всех наборов $(1;n_1,n_2,\ldots,n_k)$.

Табл. 1. Первые два яруса дерева.

	$N(Z) = f(1;\, n_1,\ldots,\, n_k)$			
1 ярус	$\boldsymbol{n_1 \le n_2}$ $N(Z)= n_2+f(1;\, n_3,\ldots,\, n_k)$		$\boldsymbol{n_1 > n_2}$ $N(Z)=n_2+f(1;\, n_1–n_2+n_3,n_4,\ldots,\, n_k)$	
2 ярус	$\boldsymbol{n_3 \le n_4}$ $N(Z)=n_2+n_4+$ $+f(1;n_5,\ldots,n_k)$	$\boldsymbol{n_3 > n_4}$ $N(Z)=n_2+n_4+$ $+f(1;n_3 - n_4 +$ $+n_5,n_6,\ldots,n_k)$	$\boldsymbol{n_1–n_2+n_3 \le n_4}$ $N(Z)=n_2+n_4+$ $+f(1;n_5,\ldots,n_k)$	$\boldsymbol{n_1–n_2+n_3>n_4}$ $N(Z)=n_2+n_4+$ $+f(1;\, n_1–n_2+n_3–$ $–n_4+n_5,\, n_6,\ldots,\, n_k)$
…	…			

Закодируем все пути от корня к листьям в данном дереве векторами $\vec{w} = (w_1,\ldots,w_r)$, $w_i \in \{0,1\}$, $1 \le i \le r$, где $r = \lfloor k/2 \rfloor$, следующим образом: рассмотрим i-й ярус и соответствующий П-граф $G(1;m,n_{2i},\ldots,n_k)$, если $m \le n_{2i}$, то $w_i = 0$, иначе $w_i = 1$.

Если $w_r = 0$, то

$$N(Z) = \sum_{i=1}^{r} n_{2i}\text{, для } k = 2r;$$

$$N(Z) = \sum_{i=1}^{r} n_{2i} + f(1;n_{2r+1}) = \sum_{i=1}^{r} n_{2i} + \left\lceil \frac{n_{2r+1}}{2} \right\rceil\text{, для } k = 2r+1.$$

Т.е. для случая $k = 2r$ достигается оценка снизу на минимальное число процессоров. Если $w_r = 1$, то

$$N(Z) = \sum_{i=1}^{r} n_{2i} + f(1, m - n_{2r})\text{, для } k = 2r;$$

$$N(Z) = \sum_{i=1}^{r} n_{2i} + f(1, m - n_{2r} + n_{2r+1})\text{, для } k = 2r+1.$$

Определим m. Вектор $\vec{w}$ имеет следующий вид: либо $\vec{w}$ состоит из одних 1, либо существует такое $t > 1$, что, начиная с позиции t, в векторе стоят только 1, а в $(t–1)$ позиции – 0. В случае, когда вектор состоит из 1, считаем $t = 1$. Тогда $m = n_{2t-1} - n_{2t} + n_{2t+1} - \cdots + n_{2r-1} = \sum_{i=t}^{r} n_{2i-1} - \sum_{i=t}^{r-1} n_{2i}$. Таким образом, для $k = 2r$:

$$N(Z) = \sum_{i=1}^{r} n_{2i} + \left\lceil \frac{\sum_{i=t}^{r} n_{2i-1} - \sum_{i=t}^{r-1} n_{2i} - n_{2r}}{2} \right\rceil = \sum_{i=1}^{r} n_{2i} + \left\lceil \frac{\sum_{i=t}^{r} n_{2i-1} - \sum_{i=t}^{r} n_{2i}}{2} \right\rceil;$$

$$N(Z) = \sum_{i=1}^{r} n_{2i} + \left\lceil \frac{\sum_{i=t}^{r+1} n_{2i-1} - \sum_{i=t}^{r} n_{2i}}{2} \right\rceil, \text{ для } k = 2r+1.$$

Покажем, что сложность описанного алгоритма построения расписания является полиномиальной. Так набор (n_1, n_2, …, n_k) по графу G можно определить со сложностью $O(n^3)$. Например, граф задан матрицей смежности, степени его вершин определяются со сложностью $O(n^2)$; просматривая степени вершин, определяем множество V_1 – либо вершины степени $(n–1)$, либо вершины степени 0. И исключаем вершины этого множества из дальнейшего рассмотрения – на это необходимо $O(n)$ операций [3]. Аналогично определяются оставшиеся множества $V_2,\ldots,V_k$. Проверка, что любая тройка работ является 3-несовместимой, также является полиномиальной, $O(n^3)$: для любой тройки попарно 2-совместимых работ i, j, s должно выполняться неравенство $\tau_i + \tau_j + \tau_s > b_p - a_q$, для всех $p \neq q$, $p, q \in \{i, j, s\}$. Алгоритм построения расписания состоит из r шагов, на каждом шаге проводиться одно сравнение и не более двух операции сложения, т.е. $O(n)$ операций.

Литература

1. *Козырев В.П.* Составление многопроцессорных расписаний для 2-совместимых и 2-несовместимых работ // Проблемы теоретической кибернетики. Тезисы докладов XIII международной конференции (Казань, 31 мая 2002 г.), часть I. М.: Мехмат МГУ. С. 89.

2. *Козырев В.П.* Характеризация графов, не имеющих чередующихся циклов // Тезисы докладов научной конференции "Математические модели сложных систем и междисциплинарные исследования" (23-24 октября 2002 г.). М.: ВЦ РАН. С. 33.

3. *Кнут Д.Э.* Искусство программирования. М.: Вильямс, 2007. Т. 1.

АЛГОРИТМЫ СОСТАВЛЕНИЯ МНОГОПРОЦЕССОРНОГО РАСПИСАНИЯ ДЛЯ НЕОДНОРОДНОГО МНОЖЕСТВА РАБОТ С ДИРЕКТИВНЫМИ ИНТЕРВАЛАМИ И ПРОИЗОЛЬНЫМИ ПРОЦЕССОРАМИ

Д.Р. Гончар, М.Г. Фуругян

Рассматривается задача составления многопроцессорного расписания с директивными интервалами для случая, когда часть работ допускает прерывания и переключения с одного процессора на другой, а часть не допускает. В отличие от задачи, рассмотренной в [1]*, процессоры могут иметь произвольные производительности. Предлагается два приближенных алгоритма, минимизирующих максимальное запаздывание. Приводятся результаты численных экспериментов.*

1. Постановка задачи

Рассматривается вычислительная система, состоящая из m процессоров, производительности которых равны s_1, s_2, …, s_m. Работая в течение времени t, процессор j выполняет работу объемом ts_j. Имеется множество работ $N = N_1 \cup N_2$ ($N_1 \cap N_2 = \varnothing$), где N_1 – непрерываемые работы, N_2 – работы, допускающие прерывания и переключения с одного процессора на другой. Предполагается, что прерывания и переключения не требуют временных затрат. В известный момент времени каждая работа может выполняться не более чем одним процессором, а каждый процессор может выполнять не более одной работы. Работы N_1 могут выполняться только процессорами $j = 1,\ldots,m_1$, $m_1 < m$ (процессорами первой группы), а работы N_2 – как процессорами первой, так и процессорами второй группы ($j = m_1+1,\ldots,m$).

Заданы объемы w_i работ $i \in N$. Для работ $i \in N_1$ установлен единый директивный интервал [0; T], который не может быть нарушен. Для каждой работы $i \in N_2$ установлен директивный интервал $[b_i, f_i]$ ($f_i - b_i \geq t_i$). Выполнение работы $i \in N_2$ может быть начато не ранее момента b_i, а если она завершится в момент $\bar{f}_i$, то штраф за несвоевременное выполнение работ N_2 составит величину $F = \max\limits_{i \in N_2} \max(\bar{f}_i - f_i, 0)$. Требуется найти такое расписание выполнения работ N, при котором все работы N_1 выполняются в интервале [0; T] и при этом штраф F минимален.

Подробный обзор литературы для случая, когда все работы являются непрерываемыми, а также для случая, когда все работы допускают прерывания и переключения, содержится в [2]. Задачи со смешанным типом работ мало освещены в литературе. Так, например, в [3, 4] предполагается, что каждая работа строго закреплена за конкретным процессором, на множестве работ задан частичный порядок выполнения и, кроме того, только один из приборов допускает прерывания. В [5] рассмотрены случаи, когда директивные интервалы одинаковые, а также, когда директивные интервалы могут различаться, но с рядом дополнительных ограничений. В [1] исследована задача, аналогичная рассматриваемой в настоящей работе, но для идентичных процессоров, а в [2] – задача на быстродействие для смешанного множества работ и идентичных процессоров.

2. Планирование выполнения непрерываемых работ

Для построения расписания выполнения работ N_1 на m_1 процессорах первой группы используется приближенный оп-

тимизационный мультиоценочный алгоритм с калибровкой [6], который, как показали численные эксперименты, имеет высокую точность и является достаточно быстрым. Если длина полученного расписания не превышает T, строится расписание прерываемых работ (разд. 3). Далее будем предполагать, что построено расписание выполнения работ N_1 и его длина не превосходит T. Введем следующие обозначения:

$$M_1 = \{j : j = 1,\ldots,m_1\};\quad M_2 = \{j : j = m_1+1,\ldots,m\};$$
$$M = M_1 \cup M_2.$$

Для $j \in M_1$ определим величины: Q_j – длина интервала загруженности процессора j; $L_j = T - Q_j (L_j \geq 0)$; $N_1(j)$ – номера работ из N_1, назначенных на процессор j; $a_{ij} (i \in N_1(j))$ – момент начала выполнения работы i процессором j.

3. Планирование выполнения прерываемых работ

Для выполнения работ N_2 используются процессоры второй группы и частично процессоры первой группы. Каждой работе $i \in N_2$ приписывается заданная величина p_i, характеризующая ее срочность. Работы с меньшей величиной p_i являются более срочными. Рассмотрим два варианта вычисления величин p_i: в первом варианте $p_i = f_i$; во втором $p_i = f_i - b_i - t_i$.

Пусть $d_1 < d_2 < \ldots < d_s$ – все различные величины b_i, $i \in N_2$. Для интервала $[d_k; d_{k+1}]$ $(k = 1,\ldots,s-1)$ введем следующие обозначения:

$N_2(d_k)$ – список номеров работ $i \in N_2$, готовых к выполнению в интервале $[d_k; d_{k+1}]$ (т.е. работ $i \in N_2$, для которых $b_i \leq d_k$); список $N_2(d_k)$ будем хранить упорядоченным по невозрастанию величин p_i (т.е. первая работа в списке самая срочная);

τ_j – момент времени, начиная с которого на процессор j можно назначить очередную работу из $N_2(d_k)$;

$v_j (j \in M)$ – максимальное время, которое может быть выделено процессором j на выполнение работ в интервале $[d_k; d_{k+1}]$;

$M_{11}(d_k)$ – номера процессоров первой группы, на которые можно дополнительно назначить работы из $N_2(d_k)$ в интервале $[d_k; d_{k+1}]$, корректируя при этом построенное ранее расписание выполнения работ N_1;

$M_{12}(d_k)$ – номера процессоров первой группы, на которые можно дополнительно назначить работы из $N_2(d_k)$ в интервале $[d_k; d_{k+1}]$, не корректируя при этом построенное ранее расписание выполнения работ N_1.

При описании алгоритма составления расписания выполнения работ N_2 будут использованы следующие процедуры.

1. Процедура $\Pi_1(i_0, j_0)$ назначает работу $i_0 \in N_2(d_k)$ на процессор $j_0 \in M_2$ в интервале $[d_k; d_{k+1}]$.

Положить $t_{i_0} = w_{i_0} / s_{j_0}$; $\tau = \min(t_{i_0}; v_{j_0})$. Работу i_0 назначить на процессор j_0 в интервале $[\tau_{j_0}; \tau_{j_0} + \tau]$. Положить $\tau_{j_0} = \tau_{j_0} + \tau$; $w_{i_0} = w_{i_0} - \tau \cdot s_{j_0}$; $v_{i_0} = v_{i_0} - \tau$. Работу i_0 исключить из $N_2(d_k)$. Если $w_{i_0} \neq 0$ и $k < S$, то работу i_0 включить в $N_2(d_{k+1})$.

2. Процедура $M_{11}(d_k)$ строит множество $M_{11}(d_k)$ и вычисляет величины v_j и τ_j, $j \in M_{11}(d_k)$.

Множество $M_{11}(d_k)$ определяется по правилу:

$$M_{11}(d_k) = \{j : j \in M_1,\ L_j > 0,\ Q_j \geq d_{k+1};\ \exists\, i \in N_1(j),\ d_k \leq a_{ij} \leq d_{k+1}\}$$

Для каждого $j_0 \in M_{11}(d_k)$ положить

$\tau_{j_0} = \min\limits_{i \in N_1(j_0)} \{a_{ij_0} : d_k \le a_{ij_0}\}$; $v_{j_0} = \min(d_{k+1} - \tau_{j_0}; L_{j_0})$.

3. Процедура $\Pi_2(i_0, j_0)$ назначает работу $i_0 \in N_2(d_k)$ на процессор $j_0 \in M_{11}(d_k)$ в интервале $[d_k; d_{k+1}]$.

Положить $t_{i_0} = w_{i_0} / s_{j_0}$; $\tau = \min(t_{i_0}; v_{j_0})$.

Для всех работ $i \in N_1(j_0)$, у которых $a_{ij_0} \ge \tau_{j_0}$, положить $a_{ij_0} = a_{ij_0} + \tau$.

Назначить работу i_0 на процессор j_0 в интервале $[\tau_{j_0}; \tau_{j_0} + \tau]$.

Положить $w_{i_0} = w_{i_0} - \tau \cdot s_{j_0}$; $L_{j_0} = L_{j_0} - \tau$;

$\tau_{j_0} = \tau_{j_0} + \tau$; $Q_{j_0} = Q_{j_0} + \tau$; $v_{j_0} = v_{j_0} - \tau$.

Работу i_0 исключить из $N_2(d_k)$. Если $w_{i_0} \ne 0$ и $k < s$, то работу i_0 включить в $N_2(d_{k+1})$.

Если для j_0 условие принадлежности множеству $M_{11}(d_k)$ не выполняется, то исключить j_0 из $M_{11}(d_k)$.

4. Процедура $M_{12}(d_k)$ строит множество $M_{12}(d_k)$ и вычисляет величины v_j и τ_j, $j \in M_{12}(d_k)$.

Множество $M_{12}(d_k)$ определяется по правилу:

$M_{12}(d_k) = \{j : j \in M_1, Q_j < d_{k+1}\}$.

Для каждого $j_0 \in M_{12}(d_k)$ положить

$\tau_{j_0} = \max\{Q_{j_0}; d_k\}$;

$v_{j_0} = d_{k+1} - \tau_{j_0}$.

5. Процедура $\Pi_3(i_0, j_0)$ назначает работу $i_0 \in N_2(d_k)$ на процессор $j_0 \in M_{12}(d_k)$ в интервале $[d_k; d_{k+1}]$.

Положить $t_{i_0} = w_{i_0} / s_{j_0}$; $\tau = \min(t_{i_0}; v_{j_0})$.

Работу i_0 назначить на процессор j_0 в интервале $[\tau_{j_0}; \tau_{j_0} + \tau]$.

Положить $\tau_{j_0} = \tau_{j_0} + \tau$; $v_{j_0} = v_{j_0} - \tau$; $Q_{j_0} = Q_{j_0} + \tau_{j_0}$.

Перейдем к описанию алгоритма 1 распределения прерываемых работ и построения окончательного расписания выполнения работ *N*. Алгоритм является обобщением известного однопроцессорного алгоритма относительной срочности (RU-алгоритм Э.Г. Коффмана [6]), который при $N_1 = \varnothing$, $m = 1$ и $p_i = f_i$ находит допустимое расписание, если оно существует.

***Алгоритм* 1**

1. Положить $k = 1$.
2. Включить в $N_2(d_k)$ все работы $i \in N_2$, для которых $b_i = d_k$.
3. Если $k < s$, перейти на шаг 4,
   если $k = s$, перейти на шаг 10.
4. Положить $\tau_j = d_k$, $v_j = d_{k+1} - d_k$ для $j \in M_2$.
5. С помощью процедуры $M_{11}(d_k)$ построить множество $M_{11}(d_k)$ и вычислить величины τ_j и v_j, $j \in M_{11}(d_k)$.
   С помощью процедуры $M_{12}(d_k)$ построить множество $M_{12}(d_k)$ и вычислить величины τ_j и v_j, $j \in M_{12}(d_k)$.
   Для $j \in M_1 \setminus (M_{11}(d_k) \cup M_{12}(d_k))$ положить $v_j = 0$.
6. Если $N_2(d_k) = \varnothing$, перейти на шаг 9;
   если $N_2(d_k) \ne \varnothing$, перейти на шаг 7.
7. Пусть $\max\limits_{j \in M} v_j = v_{j_0}$.
   Если $v_{j_0} = 0$, перейти на шаг 9;

если $v_{j_0} \neq 0$, перейти на шаг 8.

8. Пусть i_0 – номер первой в списке $N_2(d_k)$ работы.
   Если $j_0 \in M_2$, выполнить процедуру $\Pi_1(i_0, j_0)$;
   если $j_0 \in M_{11}(d_k)$, выполнить процедуру $\Pi_2(i_0, j_0)$;
   если $j_0 \in M_{12}(d_k)$, выполнить процедуру $\Pi_3(i_0, j_0)$.
9. Положить $k = k + 1$. Перейти на шаг 2.
10. Положить $\tau_j = \max(Q_j; d_s)$ для $j \in M_1$ и $\tau_j = d_s$ для $j \in M_2$.
11. Если $N_2(d_s) = \varnothing$, остановиться, расписание построено; если $N_2(d_s) \neq \varnothing$, перейти на шаг 12.
12. Пусть i_0 – первая в списке $N_2(d_s)$ работа;
    $\min\limits_{j \in M} \tau_j = \tau_{j_0}$.; $t_{i_0} = w_{i_0} / s_{j_0}$. Назначить работу i_0 на процессор j_0 в интервале $[\tau_{j_0}; \tau_{j_0} + t_{i_0}]$; исключить работу i_0 из $N_2(d_s)$; положить $\tau_{j_0} = \tau_{j_0} + t_{i_0}$; перейти на шаг 11.

Дадим некоторые пояснения к алгоритму 1. На шаге 2 к работам, которые могли быть включены в $N_2(d_k)$ при выполнении процедур Π_1 и Π_2, добавляются работы с начальным директивным сроком b_i, равным d_k. На шаге 5 определяются процессоры $j \in M_1$, которые могут выполнять в интервале $[d_k; d_{k+1}]$ прерываемые работы. На шаге 7 определяется процессор, который может выделить в интервале $[d_k; d_{k+1}]$ наибольший объем процессорного времени. На шаге 8 выбирается наиболее срочная из числа готовых к выполнению работ из N_2 и назначается на процессор, выбранный на шаге 8. Шаги 10–12 описывают построение расписания после момента d_s, которое строится по «жадному» алгоритму.

Алгоритм 2 отличается от алгоритма 1 тем, что расписание выполнения работ N_1 остается неизменным, а процессор $j \in M_1$ может использоваться только после момента Q_j. На шаге 5 вызывается только процедура $M_{12}(d_k)$ (процедура $M_{11}(d_k)$ никогда не вызывается) и полагается $v_j = 0$ для $j \in M_1 \setminus M_{12}(d_k)$. На шаге 8 проверка $j_0 \in M_{11}(d_k)$ не выполняется (и поэтому процедура Π_2 вообще никогда не вызывается). Остальные шаги алгоритма 1 не изменяются.

4. Вычислительная сложность алгоритма

Определим сначала вычислительную сложность процедур, используемых в предложенных алгоритмах. Вычислительная сложность процедуры $\Pi_1(i_0, j_0)$ есть O(1), процедуры $\Pi_2(i_0, j_0)$ – O($|N_1|$), процедуры $\Pi_3(i_0, j_0)$ – O(1), процедуры $M_{11}(d_k)$ – O($m_1|N_1|$), процедуры $M_{12}(d_k)$ – O(m_1). Кроме того, s = O($|N_2|$). Шаги 2 – 10 выполняются s раз, поэтому сложность этой части алгоритма составляет O($m_1|N_1||N_2|$). Сложность части алгоритма, соответствующей шагам 11, 12, составляет O($m|N_2|$). Таким образом, вычислительная сложность алгоритмов 1 и 2 составляет O($|N_2| \cdot \max(m_1|N_1|, m)$).

5. Результаты численных экспериментов

Были проведены численные эксперименты по сравнительному анализу алгоритмов 1, 2 и алгоритма, описанного в [8], для решения задачи на быстродействие, в которой, как и в рассмотренной в настоящей работе, N_1 – непрерываемые работы, N_2 – работы, допускающие прерывания и переключения с одного процессора на другой, s_1, s_2, …, s_m – производительности процессоров, w_i – объемы работ. Алгоритм, описанный в [8], будем называть алгоритмом 3. Вычисления проводились

для частного случая, когда производительности процессоров одинаковые (т.е. заданы длительности $t_i = w_i$ выполнения работ $i \in N.$, а производительности процессоров равны $s_j = 1$, $j = 1, 2, \ldots, m$). Для проведения сравнительного анализа алгоритмы 1, 2 были приспособлены для решения задачи на быстродействие (которую решает алгоритм 3). Для этого рассматривался отрезок $[0; \widehat{T}\,]$, где $\widehat{T} = \max(t_{max}, 2 \cdot T^*)$, $t_{\max} = \max\limits_{i \in N} t_i$, $T^* = \left\lceil \left(\sum\limits_{i \in N} t_i \right) \Big/ m \right\rceil$. С помощью процедуры деления отрезка $[0; \widehat{T}\,]$ пополам определялось такое целое значение R, что $F \neq 0$ при значениях параметров $T = R$, $b_i = 0$, $f_i = R$ и $F = 0$ при $T = R + 1$, $b_i = 0$, $f_i = R + 1$ ($i \in N$).

Число работ n и число процессоров m полагались равными $n = 100$, $m = 20$, $n = 400$, $m = 60$ и $n = 1000$, $m = 100$ (см. табл.). Эксперименты проводились для различных значений числа n_1 непрерываемых работ и числа n_2 прерываемых работ. Для каждого набора значений n, m, n_1, n_2 проводилось по 50 экспериментов с произвольными значениями длительностей работ, полученных с помощью программного генератора случайных чисел, позволяющего получать псевдослучайные числа с равномерным распределением на отрезке [1, 2600]. В каждом эксперименте для алгоритмов 1, 2, 3 вычислялось среднее значение Δ оценки погрешности (по 50 расчетам) для каждого набора n, m, n_1, n_2. Относительная погрешность алгоритма вычислялась по формуле $\Delta = \left((R - T^*) / T^* \right) \times 100\ \%$.

Из результатов численных экспериментов можно сделать следующие выводы.

1. Алгоритм 3 во всех экспериментах показал наименьшую погрешность и поэтому является наиболее точным.

2. При увеличении доли числа непрерываемых работ в общем числе работ погрешность алгоритмов 2 и 3 уменьшается, а погрешность алгоритма 1 увеличивается.

Таблица. Результаты численных экспериментов

m	n_1	n_2	Δ (%)		
			Алгоритм 1	*Алгоритм* 2	*Алгоритм* 3
20	99	1	12.8758	12.8758	5.3422
	90	10	10.5927	9.9584	1.8969
	80	20	11.6868	9.7330	0.1402
	70	30	12.5248	8.3514	0.0350
	60	40	13.5868	7.4849	0.0485
	50	50	14.8923	6.5079	0.0110
	40	60	16.1697	5.3710	0.0100
	30	70	17.3155	4.9975	0.0017
	20	80	18.3677	4.9338	0.0000
	10	90	20.3087	3.8595	0.0000
	1	99	20.1817	2.2781	0.0001
60	399	1	10.7130	10.7130	2.1668
	350	50	10.5679	9.1204	0.2786
	300	100	13.6249	7.7860	0.0975
	250	150	13.0765	6.6231	0.0641
	200	200	14.1618	4.9706	0.0060
	150	250	14.8607	3.4407	0.0014
	100	300	15.3573	2.8331	0.0061
	50	350	15.8864	2.1975	0.0006
	1	399	16.6970	1.2249	0.0000
100	999	1	7.0942	7.0942	1.2892
	900	100	7.1364	6.1736	0.0746
	800	200	7.7099	5.4968	0.0662
	700	300	8.7774	5.0084	0.0172
	600	400	9.1372	3.9406	0.0218
	500	500	9.7690	3.3775	0.0215
	400	600	10.2068	2.6450	0.0002
	300	700	10.5510	0.5969	0.0041
	200	800	10.6646	2.1021	0.0000
	100	900	11.0718	1.0274	0.0005
	1	999	11.6089	0.4214	0.0006

Литература

1. *Гончар Д.Р., Фуругян М.Г.* Алгоритмы планирования вычислений в многопроцессорных системах с неоднородным множеством работ и директивными интервалами. М.: ВЦ РАН, 2011. С. 29–39.

2. *Фуругян М.Г., Гончар Д.Р.* Минимаксная задача планирования вычислений в многопроцессорной системе со смешанным набором работ. // Системы управления и информационные технологии. 2009, № 2 (36). С. 36–39. Москва-Воронеж: Научная книга, 2009.

3. Буланже Д.Ю., Сушков Б.Г. Алгоритмы управления вычислительными системами жесткого реального времени. // Изв. АН СССР, Техн. кибернетика, 1982, № 6. С. 160–169.

4. *Буланже Д.Ю.* Оптимальная коррекция директивных интервалов для задачи одного прибора. М.: ВЦ АН СССР, 1983.

5. *Скиндерев С.А., Фуругян М.Г.* Алгоритмы планирования вычислений в многопроцессорных системах с неоднородным множеством работ. М.: ВЦ РАН, 2006.

6. *Гончар Д.Р.* Мультиоценочный алгоритм решения минимаксной задачи составления расписания // Системы управления и информационные технологии, 2007. № 1.3 (27). Москва-Воронеж: Научная книга, 2007. С. 324–328.

7. *Коффман Э.Г.* Теория расписаний и вычислительные машины. М.: Наука, 1984.

8. *Гончар Д.Р., Фуругян М.Г.* Алгоритмы планирования вычислений в многопроцессорных системах с неоднородным множеством работ. М.: ВЦ РАН, 2010. С. 12–25.

ОПТИМИЗАЦИЯ СТРУКТУРЫ БАЗЫ ДАННЫХ РЕАЛЬНОГО ВРЕМЕНИ

С.Н. Мирошник, М.Г. Фуругян

Исследуется задача оптимизация структуры базы данных реального времени. Разработаны эвристический алгоритм решения данной задачи и алгоритм, основанный на сведении ее к задаче булевого программирования.

1. Введение

Одной из основных проблем проектирования баз данных реального времени является проблема минимизации избыточности информации. Эта проблема стала особенно актуальной с появлением наиболее популярных в настоящее время реляционных баз данных, предложенных К. Дейтом в 70-х годах прошлого века. В настоящее время появилось большое количество публикаций, посвященных этой тематике. Одно из основных направлений решения проблемы избыточности является нормализация баз данных. Идея нормализации состоит в декомпозиции исходных отношений базы данных на более простые с учетом наборов ограничений на отношения. Тем самым из таблиц базы данных удаляется избыточная информация. Этой проблеме посвящены работы [1–6].

Однако нормализации базы данных приводит к существенному увеличению времени выполнения запросов. Поэтому в последнее время разработан метод частичной денормализации базы данных, направленный на получение управляемой избыточности информации для повышения производительности системы. Настоящая статья посвящена вопросам минимизации избыточности информации базы данных.

2. Постановка задачи

Имеется n независимых друг от друга программных модулей i= 1,...,n. Для их выполнения требуются данные, содержащиеся в файле Φ, который состоит из полей $\Phi_1, \Phi_2, \ldots, \Phi_r$. При выполнении каждого модуля используется часть полей Φ_j файла Φ. Для заданного числа K строятся файлы $F_1, F_2, \ldots, F_k$ ($k \le K$), каждый из которых содержит некоторые поля Φ_j файла Φ, причем для каждого модуля i некоторый файл $F_{j(i)}$ должен содержать все поля Φ_j, необходимые для его работы. Суммарная длина файлов F_i не должна превышать заданной величины W. Внутрифайловая избыточность входной информации определяется как величина $I_1 = \sum_{i=1}^{n} (|F_{j(i)}| - |\overline{F}_{j(i)}|)$, где $|F_{j(i)}|$ – длина файла $F_{j(i)}$, $|\overline{F}_{j(i)}|$ – суммарная длина полей файла $F_{j(i)}$, используемых работой i. Межфайловая избыточность базы данных определяется как величина $I_2 = \sum_{j=1}^{r} (K_j - 1)|\Phi_j|$, где K_j – число файлов F_i, в которых содержится поле Φ_j, $|\Phi_j|$ – длина поля Φ_j. Задача заключается в создании для заданных K и W такого набора файлов $F_1, F_2, \ldots, F_k$ ($k \le K$), для которого

$$\sum_{j=1}^{k} |F_j| \le W \qquad (1)$$

и величина I_1 минимальна. Если решений более одного, то из их числа выбирается такое, при котором величина I_2 минимальна. В качестве критерия оптимальности можно взять линейную комбинацию величин I_1 и I_2, например, $I_1 + I_2$.

3. Сведение к задаче булевого программирования

Пусть v_{pj} (p = 1, 2, …, n; j = 1, 2, …, r) – массив, состоящий из нулей и единиц, причем v_{pj}= 1, если для выполнения модуля p требуется поле Φ_j, и v_{pj}= 0 в противном случае.

Введем переменные x_{ij} (i = 1, 2, …, k; j = 1, 2, …, r) и y_{ip} (i = 1, 2, …,k; p = 1, 2, …, n), принимающие значения 0 и 1, причем x_{ij} = 1, если файл F_i содержит поле Φ_j, и x_{ij} = 0 в противном случае; y_{ip} = 1, если для работы модуля p требуется файл F_i, и y_{ip} = 0, в противном случае.

Сформулированная задача может быть записана в виде следующей задачи булевого программирования:

$$\min_{x_{ij}, y_{ip}} \sum_{p=1}^{n} \sum_{i=1}^{k} \left\{ y_{ip} \left[\sum_{j=1}^{r} (x_{ij} - v_{pj}) \cdot |\Phi_j| \right] \right\}; \qquad (2)$$

$$\min_{x_{ij}} \sum_{j=1}^{r} \left[\left(\sum_{i=1}^{k} x_{ij} \cdot |\Phi_j| \right) - |\Phi_j| \right]; \qquad (3)$$

$$\sum_{i=1}^{k} \sum_{j=1}^{r} x_{ij} \cdot |\Phi_j| \le W; \qquad (4)$$

$$\sum_{i=1}^{k} y_{ip} = 1, \quad p = 1, 2, \ldots, n; \qquad (5)$$

$$y_{ip}(x_{ij} - v_{pj}) \ge 0; \quad i = 1, 2, \ldots, k;$$
$$p = 1, 2, \ldots, n; \quad j = 1, 2, \ldots, r; \qquad (6)$$

x_{ij}, y_{ip} принимают значения 0 и 1. Число переменных в данной задаче равно $k(r + m)$, а число ограничений – $(nkr + 1)$.

Условие (2) соответствует минимизации величины I_1, условие (3) – минимизации величины I_2, условие (4) – условию (1). Условие (5) приписывает каждому модулю $p \in N$ некоторый файл F_i. Благодаря условию (6) у каждого модуля $p \in N$ соответствующий ему файл F_i будет содержать все необходимые поля Φ_j. Сначала должна быть решена задача миними-

зации (2) при ограничениях (4) – (6). Затем на полученном множестве решений (если решение не однозначно) решается задача минимизации (3).

4. Эвристический алгоритм создания базы данных

4.1. Предварительные замечания

Целью настоящего раздела является разработка алгоритма построения базы данных с минимальной избыточностью информации для частного случая сформулированной в разд. 2 задачи: ограничение (1) отсутствует, число K не задано и минимизируется величина $I_1 + I_2$. Раздел состоит из двух частей. В первой части формулируются требования к эвристическому алгоритму создания базы данных. Во второй части дается описание алгоритма.

Введем следующие обозначения. Пусть $\{M_i\}, i = 1...n$ – множество модулей, которые необходимо выполнить, используя информацию, содержащуюся в файлах базы данных. Будем рассматривать общий случай, когда не все поля записи модуля M_i используются им. Заметим, что каждый модуль считывает целиком всю запись соответствующего файла, в том числе поля записи, которые не используются этим модулем. Пусть v_i – число неиспользуемых модулем M_i полей. Иногда поле Φ_j для краткости будем обозначать индексом j, (j = 1,…, r). Предполагается, что длины всех полей одинаковые и длина l_i записи модуля M_i определяется как $l_i = a_i - b_i$, где a_i и b_i – номера начального и конечного полей модуля соответственно.

С помощью некоторого алгоритма все модули будут разделены на группы, каждая из которых образует свой файл $F_i, i = 1,...,k$. Модули каждой группы используют информацию только своего файла. Пусть построены файлы $F_1,...,F_k$.

Длина записи файла F_i определяется как число полей Φ_j, используемых модулями группы, соответствующих этому файлу. Пусть s_i – число модулей, соответствующих файлу F_i. При формировании файлов будет введен числовой параметр Δ, смысл которого состоит в следующем. Для некоторого выделенного (в дальнейшем используется термин опорного) модуля M, для которого a и b – номера начального и конечного полей записи, длина его записи увеличивается на Δ и эта величина будет длиной файла. Алгоритм основывается на использовании различных значений Δ, при этом Δ может принимать значения от 0 до r. Файл создается из тех модулей M_i, для которых выполнены неравенства $a - \Delta_1 \le a_i$ и $b_i \le b + \Delta_2$, где $\Delta_1 + \Delta_2 = \Delta$. При создании последующих файлов эти модули исключаются из множества $M_1,...,M_n$.

Пусть построены файлы $F_1,...,F_k$. Выше определены понятия избыточности информации двух видов: внутрифайловой (I_1) и межфайловой (I_2). Внутрифайловая избыточность образуется вследствие того, что модули файла считывают всю запись файла целиком, в том числе не используемые ими поля записи файла. К внутрифайловой избыточности относятся также не используемые поля записей модулей этого файла. С другой стороны, записи разных файлов могут содержать одни и те же поля, т.е. поля, которые продублированы в разных файлах. Смысл распределения всех модулей $M_1,...,M_n$ по разным файлам $F_1,...,F_k$ заключается в том, чтобы минимизировать $I_1 + I_2$. Таким образом выбор Δ определяет создание файлов базы данных и, тем самым, определяет избыточность информации $I_1(\Delta)$ и $I_2(\Delta)$. Величина Δ для каждого файла определяется отдельно т.е на самом деле

$\Delta = (\Delta_1,...,\Delta_k)$ и потому $I_1(\Delta) = \sum_{i=1}^{k} I_1^i(\Delta_i)$,

$I_2(\Delta) = I_2(F_1(\Delta_1),...,F_k(\Delta_k))$, где, $I_1^i(\Delta_i)$ – внутрифайловая избыточность информации файла $F_i(\Delta_i)$.

Далее, можно определить максимальные и минимальные значения величин I_1 и I_2. Построим распределение P_1 повторяемости полей всех модулей $M_1,...,M_n$ следующим образом: $P_1 = (f_1, f_2, \ldots, f_r)$, где f_i – число модулей, использующих поле i. Область определения P_1 – множество $\{1,\ldots,r\}$ и число используемых полей всеми модулями равно $\sum_{i=1}^{n} l_i - \sum_{i=1}^{n} v_i$. Построенное распределение P_1 используется для вычисления I_1 и I_2, в том числе, их экстремальных значений.

Рассмотрим распределение $P_i(F_i)$ полей модулей одного файла F_i. Пусть M_t – произвольный модуль длиной записи l_t и Δ_t – параметр. Полагаем длину записи файла F_i равной $l_t + \Delta_t$. Число модулей M_{i_j}, использующих поля из записи длиной $l_t + \Delta_t$, равно s_i. Тогда

$$I_1^i(\Delta_t) = s_i(l_t + \Delta_t) - \sum_{j=1}^{s_i} l_{i_j} + \sum_{j=1}^{s_i} v_{i_j} .$$

Заметим, что величина l_t совпадает с одной из величин l_{i_j}. Здесь $\sum_{j=1}^{s_i} l_{i_j}$ – число всех полей, используемых модулями файла F_i, $\sum_{j=1}^{s_i} v_{i_j}$ – число неиспользуемых полей в записях модулей файла F_i. Пусть $\tilde{I}_2(F_i)$ – количество продублированных полей файла F_i, т.е. $\tilde{I}_2(F_i) = \sum_{j=1}^{s_i} l_{i_j} - \sum_{j=1}^{s_i} v_{i_j} - (l_t + \Delta_t)$. В дальнейшем формула, аналогичная этой, используется для вычисления величины $I_2(F_1,...,F_k)$.

Вернемся к распределению P_1. Рассмотрим два крайних случая. В первом случае $k = 1$ и $\Delta = r$. В этом случае все модули образуют один файл и межфайловая избыточность отсутствует, т.е $I_2(r) = 0$. Кроме того, длина записи этого файла максимальна и равна r. В этом случае $I_1(r) = n\,r - \sum_{i=1}^{n} l_i + \sum_{i=1}^{n} v_i$.

Во втором случае каждому модулю соответствует отдельный файл (т.е. $k = n$), и запись этого файла совпадает с записью модуля. В этом случае $\Delta=0$, а внутрифайловая избыточность информации равна $I_1(0) = \sum_{i=1}^{n} v_i$. Межфайловая избыточность информации в этом случае будет максимальной. Однако, возможны случаи, когда записи файлов не пересекаются и тогда $I_2(\Delta)=0$. Заметим, что запись файла может содержать не используемые поля (особенно это относится к файлам, которые образованы одним модулем). Тогда

$$I_2(F_1(0),...,F_n(0) = \sum_{i=1}^{n} l_i - \sum_{i=1}^{n} v_i - r .$$

Здесь в I_2 суммируется только число продублированных полей.

Рассмотрим поведение функций $I_1(\Delta)$ и $I_2(\Delta)$. Нетрудно заметить, что $I_1(\Delta)$ – неубывающая, а $I_2(\Delta)$ – невозрастающая функции. Увеличение Δ приводит к уменьшению числа файлов и наоборот. При $\Delta = 0$ число файлов максимально,

при $\Delta = r$ файл только один. Кроме того, $I_1(\Delta) \le I_1(r)$, $I_2(\Delta) \le I_2(0)$ и $I_1(\Delta) + I_2(\Delta) \le I_1(r) + I_2(0)$.

4.2. Алгоритм построения базы данных

1. Построим файл F_1.

1.1. Рассмотрим распределение P_1 полей Ф. Пусть c_1 – поле, которое большее число раз используется модулями $M_1,...,M_n$ (т.е. $\max\limits_{i=1,...,r} f_i = f_{c_1}$), $\tilde{n}$ – число повторений поля c_1. Область определения распределения P_1 – множество $\{1,\ldots, r\}$.

1.2. Определим опорный модуль, с которого начнем строить файл F_1. Найдем модуль M_t, такой, что $\max\limits_{i=1,...\tilde{n}} \min(b_i - c_1, c_1 - a_i) = \min(b_t - c_1, c_1 - a_i)$. Здесь запись модуля M_t содержит поле c_1. Поскольку не все модули могут содержать поле c_1, то $\tilde{n} \le n$. Длина записи этого модуля $l_t = b_t - a_t$.

1.3. Зададим Δ_1. Способ выбора Δ_1 определен ниже. Длину записи файла определим как $l_t = b_t + \Delta_1 - a_t$, интервал номеров полей этого файла

$G_t = [a_t - \tilde{\Delta}_1, b_t + \tilde{\tilde{\Delta}}_1]$, где $\tilde{\Delta}_1 + \tilde{\tilde{\Delta}}_1 = \Delta$.

1.4. Найдем все модули $\{M_{t_j}\}$, поля которых содержатся в G_t и записи которых содержат поле c_1. Совокупность записей модулей M_t и $\{M_{t_j}\}$ объединяются в файл $F_1(\Delta_1)$.

1.5. Построим распределение $P_1(F_1)$ полей файла $F_1(\Delta_1)$. Область определения для $P_1(F_1)$ есть G_t. Пусть n_t – число модулей, объединенных в файл $F_1(\Delta_1)$. Тогда $I_1^1(\Delta_1) = n_t l_t - \sum\limits_{j=1}^{n_t} l_{t_j} + \sum\limits_{j=1}^{n_t} v_{t_j}$.

1.6. Исключим из общего набора модулей $M_1,...,M_n$ модули M_t и $\{M_{t_j}\}$, найденные на шаге 1.4, и пусть $M_{t_1},...,M_{t_s}$ (где $t_s = n - n_t$) – оставшиеся модули.

2. Построим файл F_2.

Отметим некоторые особенности создания файлов $F_2(\Delta_2),...,F_k(\Delta_k)$. Рассмотрим процедуру построения файла F_2. Пусть P_2 – распределение полей модулей $M_{t_1},...,M_{t_s}$. Заметим, что P_2 образуется из исходного распределения P_1 следующим образом: $P_2 = P_1 - P_1(F_1)$.

Далее, аналогично шагам 1.1, 1.2 находим соответствующее поле c_2 и определяем опорный модуль. Согласно шагу 1.3 зададим Δ_2. Здесь уточним определение поля c_2. Таких полей в P_2 может быть несколько. В зависимости от выбора поля c_2 определяется опорный модуль. Опорным модулем для $F_2(\Delta_2)$ следует выбирать тот, пересечение записи которого с G_t минимально. Это позволит минимизировать межфайловую избыточность $I_2\big(F_1(\Delta_1), F_2(\Delta_2)\big)$. Согласно шагу 1.4 определяем модули, поля которых объединяются в файл $F_2(\Delta_2)$, а согласно шагу 1.5 находим $I_1^2(\Delta_2)$. Теперь $I_1(\Delta) = \sum\limits_{i=1}^{2} I_1^i(\Delta_i)$.

$I_2(\Delta) = \sum\limits_{i=1}^{2} L_i - r_1 - \sum\limits_{i=1}^{2} v_i$, где r_1 – суммарное количество полей в файлах F_1 и F_2, L_i – длина записи файла F_i. Затем выполня-

ется шаг 1.6. После этого аналогичным образом находим распределения $P_3,\ldots,P_k$ и соответствующие им файлы $F_3,\ldots,F_k$.

Следует отметить, что имеет место равенство: $P_1=\sum_{i=1}^{k}P_i\,(F_i)$, т.е. исходное распределение P_1 состоит из локальных распределений $P_i\,(F_i)$ повторяемости полей в файлах $F_i, i=1,...,k$. Таким образом построены файлы базы данных $F_1(\Delta_1),...,F_k(\Delta_k)$. Затем вычисляются внутри- и межфайловая избыточность $I_1(\Delta)=\sum_{i=1}^{k}I_1^i(\Delta_i)$, $I_2(\Delta)=\sum_{i=1}^{k}L_i-r-\sum_{i=1}^{k}v_i$ и величина $I(\Delta)=I_1(\Delta)+I_2(\Delta)$.

Сделаем несколько замечаний по выбору Δ. Предлагается подбирать параметр Δ_1 для файла F_1 отдельно. Пусть определено поле c_1 максимальной повторяемости в P_1 и опорный модуль M_t. Зададим $\Delta_1=0$ и найдем все модули, удовлетворяющие условиям шага 1.4 алгоритма. После этого построим распределение P_2. Если в P_2 поле максимальной повторяемости c_2 единственное и совпадает с c_1, т.е. $c_2=c_1$, то увеличиваем Δ_1 до тех пор, пока в P_2 поле максимальной повторяемости не изменится, т.е. пока не будет выполнено неравенство $c_2\neq c_1$. Поскольку $\Delta_1=\tilde{\Delta}_1+\tilde{\tilde{\Delta}}_1$, то следует поочередно увеличивать $\tilde{\Delta}_1$ и $\tilde{\tilde{\Delta}}_1$ до тех пор, пока не будет выполнено неравенство $c_2\neq c_1$. Далее с учетом c_2 определяется опорный модуль и подбирается параметр Δ_2 (из условия $c_3\neq c_2$).

Работа алгоритма заканчивается в тот момент, когда на шаге 1.6 не окажется ни одного модуля. Отметим особый случай, когда на последнем шаге алгоритма остаются модули с непересекающимися записями. Пусть число таких модулей

равно *m*. Тогда поля каждого такого модуля объединяются в отдельный файл и поля записей файла и модуля совпадают. Таким образом получим файлы $F_{k-m+1},\ldots,F_k$. Следует также заметить, что для файла F_{k-m+1} не следует искать Δ_{k-m}, т.к. модули $M_{k-m+1},\ldots,M_k$ не содержат поля c_{k-m}. Кроме того, не следует определять величину Δ_k для файла F_k, если поля этого файла составляются из полей всех оставшихся модулей. Описанный алгоритм позволяет находить приближенное решение задачи минимизации $I(\Delta^*)=\min_{\Delta}I(\Delta)$.

Сделаем одно замечание к предложенному алгоритму. Избыточная информация $\sum_{i=1}^{n}v_i$, содержащаяся в записях модулей M_i, $i=1,...,n$, участвует в минимизации величины $I_1(\Delta)$ косвенным образом. А именно, одним из слагаемых в выражении для $I_1^i(\Delta)$ является величина $\sum_{j=1}^{s_i}v_{i_j}$. Для каждого из файлов $F_1,...,F_k$ эти величины могут различаться. Однако, в выражении $I_1(\Delta)=\sum_{i=1}^{k}I_1^i(\Delta)$ это слагаемое равно $\sum_{i=1}^{k}\sum_{j=1}^{s_i}v_{i_j}$. Поскольку $\sum_{i=1}^{k}s_i=n$, то $\sum_{i=1}^{k}\sum_{j=1}^{s_i}v_{i_j}=\sum_{i=1}^{n}v_i$, т.е. эта величина не зависит от разбиения модулей по файлам и может быть учтена один раз при вычислении $I_1(\Delta)$. При минимизации I можно минимизировать величину $I_2(\Delta)$. Это возможно в случае, если в файлах имеются не используемые поля. Далее, в тех случаях, когда величина $\sum_{i=1}^{n}v_i$ сравнима или превышает величи-

ну $nr - \sum_{i=1}^{n} l_i$, предлагаемый алгоритм является некорректным. Потому область его применения ограничивается выполнением условия

$$nr - \sum_{i=1}^{n} l_i >> \sum_{i=1}^{n} v_i .$$

Для того, чтобы снять это ограничение, требуется определенная модификация алгоритма.

Определим вычислительную сложность отдельных шагов алгоритма: 1.1. – $O(nr)$; 1.2 – $O(n)$; 1.3 – $O(1)$; 1.4 – $O(n)$; 1.5– $O(nr)$; 1.6 – $O(n)$; 2 – $O(nr^3)$ (с учетом возможных изменений Δ_I). Поскольку число итераций алгоритма не превосходит числа модулей, то вычислительная сложность предложенного алгоритма составляет $O(n^2 r^3)$.

4.3. Модельный пример

Работа описанного выше алгоритма проиллюстрирована на примере, в котором $r = 15$, и $n = 8$. Поля, используемые модулями, определяются следующим образом:

M_1:	6, 7, 9, 10, 11, 13, 14;	$v_1 = 2$;
M_2:	7, 8, 9, 11, 13;	$v_2 = 2$;
M_3:	8, 9, 10, 12, 13;	$v_3 = 1$;
M_4:	4, 5, 6, 8, 9;	$v_4 = 1$;
M_5:	5, 8, 9, 10;	$v_5 = 2$;
M_6:	3, 5, 6, 7, 8;	$v_6 = 1$;
M_7:	9, 11, 13, 15;	$v_7 = 3$;
M_8:	2, 4, 5;	$v_8 = 1$.

Область определения распределения P_1 есть {1,…, 15}.

Распределение P_1 :1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15;
0, 1, 1, 2, 4, 3, 3, 5, 6, 3 , 3 , 1 , 4 , 1 , 1 .

$I_1(15) = 61 + 13 = 74$; $I_2(0) = 24$. Здесь $I_2(15) = 0$. Таким образом $I(15) = I_1(15) + I_2(15) = 74$. С другой стороны $I_1(0) = 13$, $I_2(0) = 24$ и $I(0) = 37$. Выберем: $c_1 = 9$, т.к. $n_1 = 6$; при этом опорным является модуль M_1. Пусть $\Delta_1 = 0$. Имеем $v_1 + v_2 + v_3 = 5$. $F_1(0)$ включает поля модулей M_1, M_2, M_3; $l_1 = 9$; $I_1^1(0) = 5+5 = 10$.

Область определения распределения P_2 есть {1,…,15}.
Распределение P_2 : 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15;
0, 1, 1, 2, 4, 2, 1, 3, 3, 1 , 1 , 1 , 0 , 0 , 1 .

Выберем $c_2 = 5$ ($c_2 \neq c_1$), т.к $n_2 = 4$. Опорным является модуль M_6. Пусть $\Delta_2 = 3$. $F_2(3)$ включает поля модулей M_4, M_5, M_6; $l_2 = 8$. $v_4 + v_5 + v_6 = 4$, $I_1^2(3) = 6+4 = 10$. Записи модулей M_7 и M_8 не пересекаются, поэтому $\Delta_3 = \Delta_4 = 0$. Опорным является модуль M_7. $F_3(0)$ включает поля модуля M_7; $l_3 = 7$, $v_7 = 3$, $I_1^3(0) = = 0 + 3 = 3$. Опорным является модуль M_8. $F_4(0)$ включает поля модуля M_8; $l_4 = 4$, $v_8 = 1$, $I_1^4(0) = 0 + 1 = 1$,

$I_1 = \sum_{i=1}^{4} I_1^i(\Delta_i) = 24$, $I_2(F_1(0), F_2(3), F_3(0), F_4(0)) = 10$.

Таким образом, $I = I_1 + I = 24 + 10 = 34$.

Литература

1. *Дейт К. Дж.* Введение в системы баз данных. 8-е издание. М.: Вильямс, 2006.

2. *Ullman L.* PHP6 and MySQL for Dynamic Web Sites. Peachpit Press, 2007.

3. *Коннолли Т., Бегг Б.* Базы данных. Проектирование, реализация и сопровождение. Теория и практика. М.: Вильямс, 2003.

4. *Грох М., Стокман Дж., Пауэлл Г.* Библия пользователя. М.: Вильямс, 2009.

5. *Кузнецов С.Д.* Основы современных баз данных. М.: Интернет-университет информационных технологий; БИНОМ. Лаборатория знаний, 2007.

6. *Эмблер С.В., Садаладж П.Дж.* Рефакторинг баз данных: эволюционное проектирование. М.: Вильямс, 2007.

МЕТОД ВЕТВЕЙ И ГРАНИЦ ДЛЯ РЕШЕНИЯ МИНИМАКСНОЙ ЗАДАЧИ СОСТАВЛЕНИЯ РАСПИСАНИЯ

М.Г. Фуругян

Рассматривается задача составления многопроцессорного расписания минимальной длины без прерываний. Для решения этой задачи разработан метод ветвей и границ, основу которого составляют эффективные алгоритмы вычисления нижних и верхних оценок минимальной длины расписания.

1. Постановка задачи

Рассматривается вычислительная система, состоящая из m процессоров, и множество работ $N = \{1, 2, \ldots, n\}$, подлежащее выполнению. Время выполнения работы i на процессоре j равно t_{ij} ($i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, m$). При выполнении работ не допускаются прерывания и переключения с одного процессора на другой. В фиксированный момент времени каждая работа может выполняться не более чем одним процессором, а каждый процессор может выполнять не более одной работы.

Под расписанием выполнения работ N будем понимать разбиение множества N на m непересекающихся подмножеств $N_1, N_2, \ldots, N_m$ ($N = \bigcup_{j=1}^{m} N_j$; $N_{j_1} \bigcap N_{j_2} = \varnothing$ при $j_1 \neq j_2$). Работы из множества N_j приписываются процессору j и выполняются на нем одна за другой в произвольном порядке. Величина $Q_j = \sum_{i \in N_j} t_{ij}$ – загруженность процессора j ($j = 1, 2, \ldots, m$), а $\max_{j=1,2,\ldots,m} Q_j$ – это длина расписания. Задача заклю-

чается в построении оптимального по быстродействию расписания, т.е. расписания минимальной длины.

Подобные задачи широко освещены в литературе. Отметим, например, такие методы, применяемые при их решении, как случайный и исчерпывающий поиск [1], методы математического программирования [2], метод ветвей и границ [3], муравьиные алгоритмы [4], поиск с запретами [5], вероятностные алгоритмы [6], генетические алгоритмы [7], метод имитации отжига [8], различные эвристические алгоритмы [9], алгоритмы агрегирования [10] и др.

2. Метод ветвей и границ

Для решения поставленной задачи предлагается метод ветвей и границ, основанный на результатах работы [3].

2.1. Ветвление

Множество всех расписаний (их число равно m^n) будем описывать в виде дерева расписаний, изображенного на рис. На нулевом уровне дерева находится корень, который соответствует множеству всех расписаний. На первом уровне находится m вершин, каждая из которых соответствует множеству всех расписаний, в которых первая работа назначена на определенный процессор. На втором уровне дерева находится m^2 вершин, каждая из которых соответствует множеству всех расписаний, в которых первые две работы назначены на один или два определенных процессора. На n-м уровне дерева расписаний находится m^n листьев, каждый из которых соответствует некоторому расписанию выполнения множества работ N.

Пусть x_k – некоторый узел уровня k дерева расписаний, $R(x_k)$ – множество всех расписаний, соответствующих этому узлу (т.е. множество расписаний, в которых работы 1, 2, …, k назначены на определенные процессоры), x_{k+1}^j – узел уровня

k+1 ($k < n$), связанный с узлом x_k ребром, соответствующим процессору j. Наша цель – вычисление нижней и верхней оценок минимальной длины расписания на множестве $R(x_k)$. Имея эти оценки, можно применить стандартную схему метода ветвей и границ [11] (например, одностороннего или фронтального ветвления).

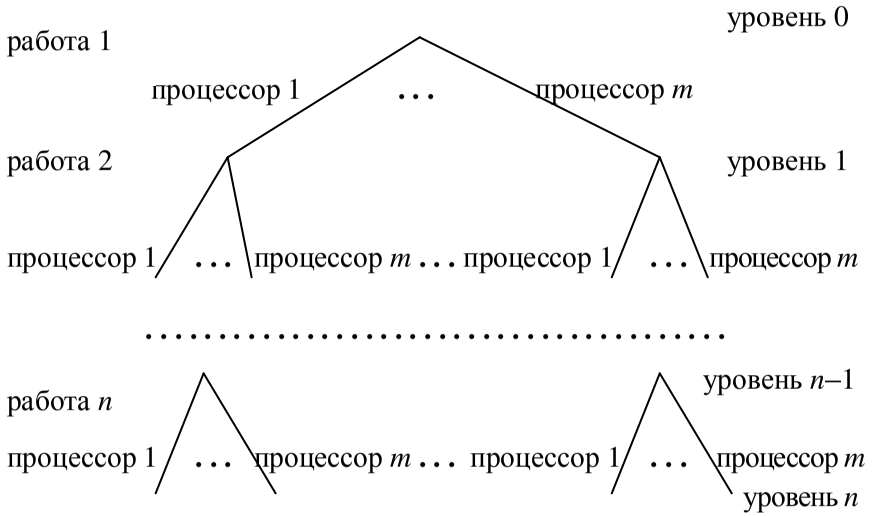

Рис. Дерево расписаний

2.2. Нижняя оценка

Пусть T_j (j = 1, …, m) – загруженность процессора j после назначения первых k работ (т.е. T_j – это суммарная длительность работ из числа 1, 2, …, k, назначенных на процессор j). Нижнюю оценку $L(x_k)$ минимальной длины расписания на множестве $R(x_k)$ будем вычислять следующим образом: $L(x_k) = \max(L_1(x_k), L_2(x_k), L_3(x_k))$, где $L_1(x_k)$, $L_2(x_k)$, $L_3(x_k)$ – это нижние оценки, вычисленные тремя различными способами.

Величина $L_1(x_k)$ вычисляется как следующий максимум: $L_1(x_k) = \max\limits_{j=1,2,\ldots,m} T_j$. Если величины $T_1, T_2, \ldots, T_m$ хранить в виде обычного массива, то сложность вычисления $L_1(x_k)$ составляет $\theta(m)$. Будем хранить эти величины в виде двоичной кучи (пирамиды) с максимальным элементом в корне [12]. Корню x_0 дерева расписаний соответствуют величины $T_1 = 0$, $T_2 = 0, \ldots, T_m = 0$ и $L_1(x_0) = 0$. Тогда сложность получения максимального элемента составит $\theta(1)$. Если работа $k+1$ будет назначена на процессор j (т.е. если от узла x_k в дереве расписаний перейти к узлу x_{k+1}^j), то будет получен новый массив $T_1, \ldots, T_{j-1}, T_j+t_{k+1,j}, T_{j+1} \ldots, T_m$. Для преобразования этого массива в кучу следует поднять элемент $T_j + t_{k+1,j}$ до нужного уровня в куче. Сложность такой процедуры составляет $O(\log_2(m))$ [13]. В полученной куче в корне по-прежнему будет находиться максимальный элемент. Таким образом, зная $L_1(x_k)$, можно вычислить $L_1(x_{k+1}^j)$ за время $O(\log_2(m))$.

Величина $L_2(x_k)$ вычисляется как следующий максимум: $L_2(x_k) = \max\limits_{i=k+1,\ldots,n} \min\limits_{j=1,\ldots,m} (T_j + t_{ij})$. Если при этом используется обычный двумерный массив A с элементами $a_{ij} = T_j + t_{ij}$, $i = k+1, \ldots, n$; $j = 1, 2, \ldots, m$, то сложность вычисления величины $L_2(x_k)$ составляет $\theta(mn)$. Будем хранить каждую строку массива A в виде двоичной кучи с минимальным элементом в корне. Для корня x_0 дерева расписаний $a_{ij} = t_{ij}$, $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, m$. Тогда сложность вычисления величины $L_2(x_k)$ составит $\theta(n)$. Если работа $k+1$ будет назначена на процессор j (т.е. если от узла x_k в дереве расписаний перейти к узлу x_{k+1}^j), то из массива A следует исключить $(k+1)$-ю строку, а в каждой оставшейся строке i увеличить элемент a_{ij} на величину $t_{k+1,j}$. Для того, чтобы каждая строка массива A по-прежнему образовывала кучу, следует поднять элемент $a_{ij} + t_{k+1,j}$ до нужного уровня [12]. Сложность такой процедуры составляет $O(n\log_2(m))$. Каждая строка массива A по-прежнему будет образовывать кучу с минимальным элементом в корне. Таким образом, зная $L_2(x_k)$, можно вычислить $L_2(x_{k+1}^j)$ за время $O(n\log_2(m))$.

Величина $L_3(x_k)$ вычисляется по формуле

$$L_3(x_k) = \frac{1}{m}\left(\sum_{j=1}^{m} T_j + \sum_{i=k+1}^{n} \min_{j=1,\ldots,m} t_{ij}\right).$$

Величину $\min\limits_{j=1,\ldots,m} t_{ij}$ вычислим сразу для всех $i = 1, 2, \ldots, n$ до начала вычисления нижних оценок. Тогда сложность вычисления величины $L_3(x_k)$ составляет $O(n + m)$. Перейдем в дереве расписаний от узла x_k к узлу $x_{k+1}^{j_0}$, $k < n$ (т.е. будем считать, что работа $k+1$ назначена на процессор j_0).

Тогда

$$L_3(x_{k+1}^{j_0}) = \frac{1}{m}\left(\left(\sum_{j=1}^{m} T_j + t_{k+1,j_0}\right) + \sum_{i=k+1}^{n} \min_{j=1,\ldots,m} t_{ij}\right).$$

Вычислим разность

$$L_3(x_{k+1}^{j_0}) - L_3(x_k) = \frac{1}{m}\left(t_{k+1,j_0} - \min_{j=1,\ldots,m} t_{k+1,j}\right).$$

Таким образом,

$$L_3(x_{k+1}^{j_0}) = L_3(x_k) + \frac{1}{m}\left(t_{k+1,j_0} - \min_{j=1,\ldots,m} t_{k+1,j}\right),$$

и с помощью данного рекуррентного соотношения, используя $L_3(x_k)$, величина $L_3(x_{k+1}^{j_0})$ вычисляется за время $O(1)$.

2.3. Верхняя оценка

В качестве верхней оценки $H(x_k)$ минимальной длины расписания на множестве $R(x_k)$ возьмем длину расписания, в котором работы $1, 2, \ldots, k$ назначены на процессоры в соот-

ветствии с вершиной x_k дерева расписаний, а работы k+1, …, n назначаются по следующему “жадному” алгоритму. Пусть уже назначены работы 1, 2, …, p ($k \le p < n$), T_j – загруженность процессора j (j = 1, 2, …, m) и $\min(T_1 + t_{p+1,1}, ..., T_m + t_{p+1,m}) = T_{j_0} + t_{p+1,j_0}$. Тогда работа p+1 назначается на процессор j_0. Указанная процедура повторяется для $p = k, k + 1, …, n - 1$. Сложность процедуры вычисления величины $H(x_k)$ составляет $O(mn)$. В случае, когда процессоры идентичные (т.е. $t_{ij_1} = t_{ij_2}$ при всех $1 \le j_1, j_2 \le m$) работа p+1 назначается на процессор j_0, определяемый соотношением $\min(T_1, T_2, ..., T_m) = T_{j_0}$.

В этом случае величины T_1, T_2, …, T_m можно хранить в виде двоичной кучи с минимальным элементом в корне. Корню x_0 дерева расписаний соответствуют величины $T_1 = 0$, $T_2 = 0, …, T_m = 0$. Тогда при переходе от узла x_k к узлу x_{k+1}^j применима процедура, аналогичная той, которая описана для вычисления величины $L_1(x_k)$, с той лишь разницей, что теперь в корне находится минимальный элемент и поэтому элемент $T_{j_0} + t_{p+1,j_0}$ следует опустить в дереве расписаний до нужного уровня. Сложность такой процедуры составляет $O(\log_2(m))$, а сложность вычисления величины $H(x_k)$ составляет $O(n\log_2(m))$.

Литература

1. *Гончаров Е.Н., Кочетов Ю.А.* Вероятностный поиск с запретами для дискретных задач безусловной оптимизации // Дискретный анализ и исследования операций. Сер. 2. 2002. Т. 9. № 2. С. 13–30.

2. *Кочетов Ю.А., Столяр А.А.* Использование чередующихся окрестностей для приближенного решения задачи календарного планирования с ограниченными ресурсами // Дискретный анализ и исследования операций. Сер. 2. 2003. Т. 10. № 2. С. 29–56.

3. *Алексеев О.Г.* Комплексное применение методов дискретной оптимизации. М.: Наука, 1987.

4. *Штовба С.Д.* Муравьиные алгоритмы // ExponentaPro. Математика в приложениях. 2003. № 4(4). С. 70–75.

5. *Glover F., Laguna M.* Chapter 3: Tabu search/ Ed. *R. Colin Reeves*, Modern Heuristics Techniques for Combinatorial Problems. Oxford, Blackwell Scientific Publications, 1993. P. 70–150.

6. *Raghavan R.* Probabilistic Construction of Deterministic Algorithms: Approximating Packing Integer Programs // J. Computer and System Sciences. 1988. V. 37. P. 130–143.

7. *Костенко В.А., Смелянский Р.Л., Трекин А.Г.* Синтез структур вычислительных систем реального времени с использованием генетических алгоритмов// Программирование. 2000. № 5. С. 63–72.

8. *Shen C., Pao Y., Yip P.* Scheduling multiple job problems with guided evolutionary simulated annealing approach // Proc. First IEEE Conf. on Evolutionary Computations. Orlando, 1994. P. 702–706.

9. *Brucker P.* Scheduling Algorithms. Heidelberg, Springer, 2001.

10. *Красовский Д.В.* Алгоритмы решения задачи составления оптимального расписания без прерываний. Диссертация на соискание ученой степени канд. физ.-матем. наук. М.: МФТИ, 2007.

11. *Коглер В., Штиглиц К.* Перечислительные и итеративные алгоритмы. В кн.: Теория расписаний и вычислительные машины. Под ред. Коффмана Э.Г. М.: Наука, 1984. С. 251–288.

12. *Кормен Т., Лейзерсон Ч., Ривест., Штайн К.* Алгоритмы: построение и анализ. М.: Вильямс, 2005.

Содержание