ЭКОНОМИКО-
МАТЕМАТИЧЕСКАЯ
БИБЛИОТЕКА

ТЕОРИЯ РАСПИСАНИЙ И ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

Под редакцией Э. Г. КОФФМАНА

Перевод с английского В. М. АМОЧКИНА

Под редакцией Б. А. ГОЛОВКИНА

МОСКВА «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ ЛИТЕРАТУРЫ
1984

22.18
Т 33
УДК 519.6

COMPUTER AND JOB-SHOP SCHEDULING THEORY

Ed. by E. G. COFFMAN, Jr.

Coauthors:
J. L. BRUNO, E. G. COFFMAN, Jr.,
R. L. GRAHAM, W. H. KOHLER,
R. SETHI, K. STEIGLITZ,
J. D. ULLMAN

John Wiley & Sons
1976

Т $\frac{1702070000-047}{053(02)-84}$ 49-83

ОГЛАВЛЕНИЕ

ОТ РЕДАКТОРА ПЕРЕВОДА

Предлагаемая книга посвящена задачам теории расписаний — важной области прикладной математики, имеющей многочисленные и разнообразные применения. Она содержит достаточно полное и одновременно компактное изложение результатов решения основных задач, таких, как минимизация конечного и среднего времени завершения работ одним и несколькими исполнителями с учетом различных условий. Ранее советский читатель мог ознакомиться с теорией расписаний по книге В. С. Танаева и В. В. Шкурбы «Введение в теорию расписаний» и по переводу на русский язык книги Р. В. Конвея, В. Л. Максвелла и Л. В. Миллера «Теория расписаний», вышедшими в свет в издательстве «Наука» в 1975 г.

Материал данной книги изложен в терминах расписаний для одиопроцессорных и многопроцессорных вычислительных систем. Для расписаний использованы, в основном, детерминированные математические модели (с заранее известными параметрами), в которых предусмотрены наборы как взаимосвязанных, так и независимых заданий или работ и соответствующие совокупности ресурсов для их исполнения. Эти модели, несмотря на их простоту, настолько содержательны, что охватывают очень широкий круг практически важных задач упорядочения, выходящих за рамки начальных формулировок. Результаты решений даны в виде эффективных оптимальных алгоритмов, эвристических алгоритмов с оценкой их эффективности и соответствующими границами, а также эффективных алгоритмов перебора и приближенных алгоритмов для построения разнообразных расписаний. Специфические вопросы применения ЭВМ не рассматриваются.

Книга содержит шесть глав. В первой (вводной) главе приведены основные понятия детерминированной теории расписаний и дан обзор последующих глав. Главы со второй по шестую фактически независимы между собой, и каждую из них можно читать после первой без обращения к другим главам. Порядок следования глав, однако, выбран таким, чтобы облегчить восприятие материала в целом при чтении книги подряд (от первой главы до последней).

Во второй и третьей главах рассмотрены основные алгоритмы составления расписаний, обеспечивающих минимизацию времени завершения комплекса работ и минимизацию среднего времени завершения работ соответственно для случаев различного числа тех или иных процессоров, различных видов заданий, их исходного упорядочения и условий исполнения. Четвертая глава посвящена вычислительной сложности решения задач упорядочения при построении расписаний. В пятой главе рассмотрены верхние оценки длин расписаний, а в шестой — точные и приближенные алгоритмы ветвей и границ для составления расписаний и результаты вычислительных экспериментов, а также осуществлено сравнение с алгоритмами динамического программирования.

В книге умело раскрыто смысловое содержание постановок задач и методов их решения и одновременно даны строгие математические доказательства. Широко использована наглядная табличная форма представления сводок результатов, приведено большое число поясняющих примеров и графических иллюстраций. Список литературы включает около полутора сотен работ зарубежных авторов. При переводе добавлен краткий список работ отечественных и зарубежных авторов по теме книги, опубликованных после написания данной книги, в котором акцент сделан на многопроцессорные расписания.

Вследствие того, что главы книги написаны разными авторами, имеются некоторые различия в стиле изложения, способах представления результатов и обозначениях. Эти обстоятельства, а также все еще недостаточно устоявшаяся терминология в теории расписаний вызвали определенные трудности при переводе.

Предлагаемая книга выдержала проверку временем и является одной из наиболее цитируемых в своей области. Несмотря на то, что после ее выхода в свет получен ряд новых результатов (см., например, [165, 179, 188, 214, 219)], материал книги представляет большой интерес для читателей в нашей стране, поскольку многие приведенные в ней результаты имеют основополагающий характер в теории расписаний и, кроме того, большинство изложенных результатов не публиковалось на русском языке.

Декабрь 1982 г.

Б. А. Головкин

В последние годы отмечается все возрастающий интерес к детерминированной теории расписаний, сопровождающийся новыми результатами. Данная книга является попыткой представить современное состояние этой области с точки зрения направлений исследований и характера решаемых задач. Поэтому большая часть материала посвящена относительно недавним результатам, которые впервые обобщены и собраны воедино. Хотя книга и состоит из шести отдельных согласованных глав, ее следует рассматривать не как сборник статей, а, скорее, как текст, специально посвященный определенной проблеме и имеющий много авторов. При таком количестве авторов возникла необходимость в редакторе, который скоординировал бы усилия семи человек.

Книга представляет собой теоретическое исследование проблем упорядочения, возникающих при организации выполнения работ в вычислительных системах. При этом рассматриваемые модели являются простыми по структуре и содержательными по отношению к большому количеству прикладных проблем. Если говорить кратко, изучаемая общая модель предполагает наличие множества заданий или работ и множества ресурсов, которые используются при обслуживании заданий. Почти во всех случаях модели являются детерминированными в том смысле, что информация, описывающая задания, известна заранее. Эта информация включает времена выполнения заданий, ограничения предшествования при обслуживании заданий, стоимости пребывания в системе и требуемые ресурсы. Рассматриваются основные задачи упорядочения по критериям минимизации длины расписания и минимизации времени пребывания в системе (взвешенного по стоимостям пребывания заданий в системе), а также упорядочения, связанного с выполнением директивных или крайних сроков. Рассматривается также ряд сопутствующих задач. Представленные результаты включают эффективные оптимальные алгоритмы, эвристические алгоритмы и оценки их характеристик, эффективные перечислительные и приближенные методы, а также математическое обоснование оценок сложности широкого класса задач упорядочения.

Вычислительные машины рассматриваются, по крайней мере, в трех аспектах. Во-первых, для нас они представляют практически универсальную область приложений. Прикладную сторону рассматриваемых задач можно проиллюстрировать на примере конструирования операционных систем общего назначения, хотя ряд проблем может иметь первостепенную важность в других применениях. Во-вторых, вычислительные машины должны рассматриваться как инструмент, позволяющий реализовать перечислительные и приближенные методы. Наконец, информатика породила теорию вычислительной сложности, которую мы здесь рассматриваем применительно к задачам упорядочения.

Как и другие разделы прикладной математики, теория расписаний позволяет решать задачи из многих научных дисциплин. В нашем случае к ним относятся организация производственных процессов, наука об управлении, экономика, исследование операций, информатика и электротехника. Книга непосредственно предназначена для тех подразделений исследовательских и учебных институтов, в которых занимаются задачами в области детерминированной теории расписаний. Сущность рассматриваемого материала требует от читателей определенной математической подготовки. Таким образом, предполагается, что книга доступна студентам, по крайней мере, после первого года обучения.

Я благодарен Исследовательскому институту по информации и автоматике (Роканкур, Франция) за главную поддержку в создании этой книги. Начальный этап и бо́льшая часть работы над главами 2 и 3 проходили в университете штата Пенсильвания при поддержке со стороны Национального научного фонда по субсидии NSF-28290. Работа над главой 6 была поддержана Национальным научным фондом по субсидии NSF-GK-37400 и NSF-GK-42048 и Исследовательским бюро армии США по контракту DANCO4-69-C-0012. Выражаю благодарность миссис Тедди Поттер и миссис Ханне Кресс за перепечатку части рукописи.

Наконец, выражаю особую признательность д-ру М. Гэри и д-ру Д. Джонсону, взявшим на себя труд по прочтению и исправлению рукописи. Они особенно помогли мне как редактору в постоянном стремлении включать в книгу как можно больше новых результатов, чтобы не дать ей устареть до выхода в свет.

Э. Г. Коффман

Пенсильванский университет Май 1975 г.

ГЛАВА 1

ВВЕДЕНИЕ В ДЕТЕРМИНИРОВАННУЮ ТЕОРИЮ РАСПИСАНИЙ

В этой главе мы рассмотрим с общих позиций круг проблем, составляющих содержание книги. Для этого введем общую систему обозначений, применяемую при описании моделей, и приведем ряд результатов, которые служат для обоснования основных положений или расширяют и обобщают область задач, охватываемых книгой. Поэтому мы рекомендуем читателю вначале ознакомиться с этой главой, а затем уже приступать к изучению глав 2—6. Хотя каждая глава может быть прочитана независимо, читатель заметит, что в выбранном порядке изложения заложен определенный смысл.

Взаимная независимость глав 2—6 является в некоторой степени результатом того, что книга написана многими авторами; в данной ситуации независимость глав представляет, конечно, удобство для авторов. Однако предмет исследования разбит на части таким образом, что эта независимость не кажется неестественной, ввиду чего некоторое несовпадение стилей и обозначений, которое может быть замечено при переходе от одной главы к другой, не помешает пониманию, особенно если предварительно ознакомиться с настоящей главой.

В § 1.1 обсуждаются цели и предпосылки книги. Общая модель, изучаемая в дальнейшем, представлена в § 1.2. Параграф 1.3 посвящен предшествующим результатам, заимствованным в основном из [28]. В § 1.4 дан обзор результатов глав 2—6, с использованием, где это возможно, табличного представления. Ряд сопутствующих результатов, которые служат дополнением к главам 2—6, приведен в § 1.5. Наконец, § 1.6 содержит некоторые замечания по поводу обозначений, использованных в последующих главах.

1.1. Цели и предпосылки

В наиболее общей формулировке задачи составления расписаний, изучаемые в этой книге, состоят в следующем. С помощью некоторого множества ресурсов или обслужи-

вающих устройств должна быть выполнена некоторая фиксированная система заданий. Цель заключается в том, чтобы при заданных свойствах заданий и ресурсов и наложенных на них ограничениях найти эффективный алгоритм упорядочения заданий, оптимизирующий или стремящийся оптимизировать желаемую меру эффективности. В качестве основных мер эффективности изучаются длина расписания и среднее время пребывания заданий в системе. Модели этих задач являются детерминированными в том смысле, что вся информация, на основе которой принимаются решения об упорядочении, известна заранее. В частности, задания и все данные о них предполагаются известными в начальный момент, за который обычно принимается время $t=0$.

Поскольку невозможно эффективно организовать рабочий день, составить расписание экзаменов в университете или даже приготовить пищу, не решая на каждом шагу задачи упорядочения, нет нужды доказывать обоснованность предпосылок к изучению этих проблем. С другой стороны, наше внимание должно быть сосредоточено на формулировках задач, которые были бы ориентированы на применение в тех случаях, когда неэффективное упорядочение обходится слишком дорого. Допущения, в рамках которых задача может быть формализована, должны, например, отражать общие черты производственных задач. В данном случае, в силу научных интересов авторов, природа рассматриваемых моделей, в основном, связана с вычислительными системами.

Читатели, хорошо знакомые с архитектурой вычислительных систем общего назначения и с проблемами экономной организации вычислительных работ, убедятся, что при этом мы практически не теряем общности. Редко можно встретить фундаментальную проблему упорядочения, которая не была бы интересной и важной применительно к существующим или разрабатываемым вычислительным системам. С другой стороны, специальные вопросы, связанные с применениями, рассматриваться не будут. Как мы увидим позднее, книга отражает тот факт, что важные теоретические результаты (за исключением тех, которые связаны с вопросами сложности) могут быть получены на базе весьма простых моделей, которые применимы к широкому спектру практических задач составления расписаний.

В книге использованы многие сходные термины без их формального уточнения. Например (если не оговорено особо), такие слова, как «работы», «задания», «программы»

(в вычислительной машине) и «заявки», могут, с учетом наших целей, считаться эквивалентными. Понятие «ресурсы» может быть отнесено к машинам, накопителям и, наиболее часто, к процессорам. По отношению к соответствующим ресурсам работы могут быть совершены, просчитаны, выполнены, записаны в память или обслужены. Вместо слова «алгоритм» мы можем использовать такие слова, как «правило», «процедура», или соответствующая «функция», или «отображение». Такие слова и выражения, как «составление расписаний», «упорядочение», «распределение» и «назначение», используются как синонимы или аналоги, в зависимости от контекста. Как обычно в современной литературе, нетривиальные алгоритмы описываются с помощью неформальной алголоподобной нотации; здесь не используются никакие специальные средства, которые могли бы оказаться читателю непонятными.

Вообще говоря, цель данной книги состоит в описании детерминированных задач упорядочения и в столь исчерпывающем их анализе, чтобы создать уверенность в том, что:

1) полностью охвачено состояние исследований в этой области, выполненных в последние годы;

2) хорошо представлены основополагающие принципы, которые могут быть абстрагированы от проведенного здесь и ранее анализа и формальных подходов к этим задачам.

В наши дни любая попытка написать книгу, которая имела бы явный успех в обоих аспектах, обречена на неудачу, так как такая книга будет постоянно находиться в стадии написания. Основная причина этого заключена в обычной трудности находиться au courant *) в области науки, которая набрала такие темпы, как теория расписаний. Но проблема усугубляется еще тем, что теория расписаний проникла в несколько дисциплин, каждая из которых представляет собой обширную и развивающуюся область науки. В нашей книге учтены многие результаты в области исследования операций, информатики, науки об управлении, организации производственных процессов, электротехники и прикладной математики, полученные как в промышленности, так и в научных организациях, где трудится большое (и все возрастающее) число ученых, стремящихся продвинуть вперед теорию расписаний. Невозможно охватить огромное разнообразие специальных си-

*) Au courant — в курсе (быть в курсе) (фр. яз.). (*Прим. перев.*)

стем, которые образуют принципиальные предпосылки для исследований в рамках этих дисциплин. Таким образом, наша более скромная цель заключается в том, чтобы сосредоточить усилия на простых и характерных моделях, имеющих простую структуру, которые с точки зрения комбинаторной сложности и методов анализа напоминают или уточняют различные структуры, встречающиеся в перечисленных разделах науки и техники.

После более формального обсуждения результатов последующих глав мы вернемся к вопросу о целях, когда будем обсуждать задачи и подходы, не подвергшиеся детальному исследованию. В заключение данного параграфа сделаем некоторые замечания по поводу литературы, для того чтобы указать нашу отправную точку. Основное внимание в последующих главах уделено результатам, которые появились в течение последних 7—8 лет; фактически, большинство из них стало известно в последние 4—5 лет. Что касается книг, то первые главы монографии [28] (опубликованной в 1967 г.) составили естественную основу настоящей книги. Мы особенно рекомендуем эту монографию читателям, интересующимся предпосылками из области промышленности, которые здесь не рассматриваются. Мы снова подчеркиваем, что наше исследование носит почти полностью математический характер и в очень незначительной степени касается прикладных аспектов. Применимость (и, конечно, неприменимость) нашей книги будет совершенно очевидна практически во всех случаях, главным образом благодаря простоте моделей. Но [28] дает хорошее представление об общих практических задачах и содержит многие черты таких задач, которые расширяют рассматриваемые здесь модели, но для которых сравнимые результаты отсутствуют (см. также [9]).

В данной книге представлены, фактически, все теоретические результаты, содержащиеся в [28], которые попадают в область детерминированной теории расписаний. Они образуют основу для новых результатов. В действительности исходный материал лишь кратко излагается в отдельных параграфах первой и второй глав. Дополнительные результаты, относящиеся к главам 2—5 и связанные с проблемами упорядочения в вычислительных системах, которые также можно рассматривать как исходные, имеются в более поздней книге по теории операционных систем [22]. Именно ограниченные размеры третьей главы книги [22] дали толчок к написанию настоящей книги. Таким образом, книга содержит, в основном, новые результаты, до сих пор

не публиковавшиеся в виде отдельного издания. Из недавних обзорных работ, касающихся многих вопросов, которым посвящена данная книга, следует отметить [5, 14, 29, 47].

1.2. Основная модель

Модель процесса упорядочения, в терминах которой формулируются все последующие задачи, представлена в виде совокупности моделей, описывающих ресурсы, систему заданий, ограничения предшествования и меры оценки расписаний. (В конце главы мы кратко рассмотрим вопросы, связанные с системой обозначений в этой и последующих главах.)

1.2.1. Ресурсы. В большинстве изучаемых моделей ресурсы состоят просто из набора процессоров $P=\{P_1,\ldots,P_m\}$. В зависимости от особенностей задачи они являются либо идентичными, либо одинаковыми только по функциональным возможностям, но разными по быстродействию, либо разными как по возможностям, так и по быстродействию.

В наиболее общей модели еще имеется набор дополнительных типов ресурсов $\mathcal{R}=\{R_1,\ldots,R_s\}$, некоторое (возможно, пустое) подмножество которых используется на протяжении всего выполнения задания на некотором процессоре. Общее количество ресурса типа R_j задается положительным числом m_j. Применительно к вычислительным машинам, например, такими ресурсами могут быть устройства оперативной и внешней памяти, устройства ввода-вывода или библиотеки подпрограмм *). Хотя возможно включить в $\mathcal{R}$ и процессоры, однако более удобно рассматривать их отдельно, потому что:

1) они образуют ресурс такого типа, который с необходимостью является общим для всех заданий (хотя для двух различных заданий нет необходимости требовать один и тот же процессор);

2) дискретность ресурсов этого типа такова, что задание в каждый момент времени может выполняться, самое большее, на одном процессоре.

*) В данной главе и в гл. 4 (если не оговорено особо) ресурс типа R_i рассматривается как набор, состоящий из m_i единиц одинакового ресурса. В несколько более общей модели гл. 5 m_i обозначает *количество* ресурса R_i, обычно приведенное путем нормализации к 1, произвольная доля которого может потребоваться для выполнения данного задания.

Эти особенности не относятся к ресурсам типа R_i ($1 \leqslant i \leqslant s$).

1.2.2. Система заданий. Общая система заданий для заданного набора ресурсов может быть определена как система $\mathscr{T}$, $\prec$, $[\tau_{ij}]$, $\{\mathscr{R}_j\}$, $\{w_j\}$ следующим образом.

1. $\mathscr{T} = \{T_1, \ldots, T_n\}$ есть набор заданий, подлежащих выполнению.

2. $\prec$ обозначает заданное на $\mathscr{T}$ отношение частичного порядка (нерефлексивное), которое определяет ограничения на последовательность выполнения заданий. Запись $T_i \prec T_j$ означает, что T_i должно быть завершено раньше, чем начнется выполнение T_j.

3. $[\tau_{ij}]$ представляет собой матрицу размера $m \times n$, элемент которой $\tau_{ij} > 0$ есть время выполнения задания T_j ($1 \leqslant j \leqslant n$) на процессоре P_i ($1 \leqslant i \leqslant m$). Мы полагаем, что $\tau_{ij} = \infty$, если T_j не может быть выполнено на P_i, и что для каждого j существует по крайней мере одно i, для которого $\tau_{ij} < \infty$. В случае, когда все процессоры идентичны, τ_j обозначает время выполнения T_j на любом процессоре.

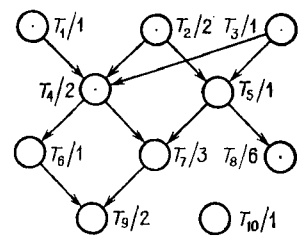
4. $\mathscr{R}_j = [R_1(T_j), \ldots, R_s(T_j)]$ ($1 \leqslant j \leqslant n$) есть набор, i-я компонента которого представляет собой количество ресурса типа R_i, необходимое для выполнения задания T_j. Далее везде предполагается, что $R_i(T_j) \leqslant m_i$ для всех i и j.

5. Веса w_i ($1 \leqslant i \leqslant n$) интерпретируются как стоимости пребывания заданий в системе (или, точнее, удельные стоимости), которые в общем случае могут быть произвольными функциями параметров, связанных с T_i. Мы, однако, считаем w_i постоянными величинами. Таким образом, «стоимость» завершения задания T_i в момент t равна просто $w_i t$.

Данная формулировка является намного более общей, чем те, которые мы собираемся использовать в последующих главах, зато модель каждой из исследуемых нами задач может быть представлена как частный случай описанной общей модели. Стоит особо отметить ограничение, наложенное на отношение порядка между заданиями. Мы не можем, например, представить в рассмотренном виде программу, содержащую циклы. Отметим, что отношение частичного порядка $\prec$ обычно представляется в виде ориентированного ациклического графа *(орграфа)*, у которого отсутствуют транзитивные (избыточные) дуги. Если не оговорено особо, мы полагаем, что $\prec$ задается в виде списка дуг в таком графе. Однако, вообще говоря, способ, с помощью которого описывается частичное упорядочение, вли-

яет на сложность решения задачи. (Мы вернемся к этому вопросу позднее.)

На рис. 1.1, например, чтобы отметить вершины графа, использованы обозначения T_i/τ_i. Задания иногда обозначаются просто их номерами (т. е. «задание 1» может быть использовано вместо T_1, особенно когда вершины графа более удобно помечать целыми числами).



Задание/время выполнения, идентичные процессоры

Рис. 1.1. Представление системы заданий $(\mathscr{T}, \prec, \{\tau_i\})$ в виде ациклического орграфа. Обозначения и свойства: 1) Ацикличность. 2) Отсутствие транзитивных ребер (таким могло бы быть ребро (T_1, T_6)). 3) Начальные вершины T_1, T_2, T_3, T_{10}, конечные вершины T_8, T_9, T_{10}. 4) T_7, например, является преемником T_1, T_2, T_3, T_4, T_5, но непосредственным преемником — только T_4, T_5; T_5 является предшественником T_7, T_8, T_9, но непосредственным предшественником — только T_7, T_8. 5) Уровни: T_1—8, T_2—9, T_3—8, T_4—7, T_5—7, T_6—3, T_7—5, T_8—6, T_9—2, T_{10}—1. 6) Критические пути: T_2, T_5, T_8 и T_2, T_4, T_7, T_9.

Как и следовало ожидать, мы предпочитаем графический способ определения систем заданий описанию их с помощью соответствующей пятерки объектов. В связи с этим мы выбрали ряд более или менее общеупотребительных терминов, относящихся к орграфам. В частности, *путем* длины k от T к T' в заданном графе G называется последовательность вершин *) (заданий) $T_{i_1}, \ldots, T_{i_k}$ такая, что $T = T_{i_1}$, $T' = T_{i_k}$ $(k \geqslant 1)$ и $(T_{i_j}, T_{i_{j+1}})$ есть дуга в G для всех $1 \leqslant j \leqslant k-1$. Кроме того, если такой путь существует, T будет называться *предшественником* T', а T'—*преемником* T. Если $k = 2$, то будут использоваться термины *непосредственный предшественник* и *непосредственный преемник*. *Начальными* являются вершины, не имеющие предшественников, а *конечными*—не имеющие преемников. Граф образует *лес*, если либо каждая вершина имеет не более одного предшественника, либо каждая вершина имеет не более одного преемника. Если лес в первом случае имеет ровно одну вершину без предшественников, а во втором—ровно одну вершину без преемников, то он называется также *деревом*. В обоих случаях термины *корень* и *листья* имеют обычное значение. *Уровнем* вершины T называется сумма времен выполнения, приписан-

*) В качестве синонима употребляется также термин *узел*.

ных вершинам, находящимся на таком пути от T к конечной вершине, на котором эта сумма имеет максимальное значение. Такой путь называется *критическим* путем, если вершина T имеет наибольший уровень в графе.

1.2.3. Ограничения при составлении расписаний. Под словом «ограничения» здесь подразумевается то, что мы ограничиваем круг алгоритмов составления расписаний двумя (хотя и широкими) классами. Рассматриваются два основных ограничения.

1. *Составление расписаний без прерываний.* При этом ограничении выполнение задания, раз начавшись, не может быть прервано, т. е. выполнение задания всегда доходит до конца. В общем случае — при составлении расписаний *с прерываниями* — разрешается прерывать задания и снимать их с процессора; при этом полагается, что общее время, требуемое для выполнения задания, остается неизменным и при прерываниях отсутствуют потери времени обслуживания (т. е. выполнение прерванного задания возобновляется с того места, в котором произошло прерывание) *).

2. *Составление расписаний с помощью списка.* При таком способе составления расписаний предполагается, что вначале готовится упорядоченный список заданий из $\mathscr{T}$. Этот список часто называется *списком приоритетов.* Последовательность, в соответствии с которой задания назначаются на процессоры, составляется путем многократного просмотра списка. В частности, если появляется освободившийся процессор, то список начинает просматриваться сначала и просматривается до тех пор, пока не найдется первое невыполненное задание T, которое готово к выполнению. Задание считается готовым к выполнению на данном процессоре, если выполнение всех предшественников T завершено и имеющегося количества ресурсов достаточно для того, чтобы обеспечить $R_i(T)$ для всех i, $1 \leqslant i \leqslant s$. Это задание назначается для выполнения на свободный процессор. Мы предполагаем, что каждый просмотр осуществляется мгновенно. Кроме того, если одновременно два или более процессоров оказываются свободными, мы предполагаем, что сначала назначается задание на P_1, затем на P_2, затем на P_3 и так далее. Для удобства записи предполагаем, что список упорядочивается (и просматривается) слева направо и записывается в виде $L=(T_{i_1}, \ldots, T_{i_n})$.

*) Рассматриваемый вид прерываний не следует путать с прерываниями, возникающими в машине по ходу вычислительного процесса и связанными с обработкой, например, внешних сигналов. *(Прим. перев.)*

При этом прерывания не рассматриваются. Таким образом, расписания, составленные с помощью списка, образуют подмножество расписаний без прерываний.

Перед тем как перейти к рассмотрению показателей эффективности расписаний, проиллюстрируем способ графического представления расписаний, полагая $s=0$. Для этого используем временну́ю диаграмму, показанную на рис. 1.2, которая соответствует системе заданий, изображенной на рис. 1.1. Число процессоров очевидным образом определяет число горизонтальных линий, обозначающих оси времени. Заштрихованные области обозначают периоды, когда процессоры свободны. Для того чтобы при необходимости можно было ссылаться на свободные периоды, они обозначены символом $\varnothing$. Для обозначения таких диаграмм используется при необходимости символ D.

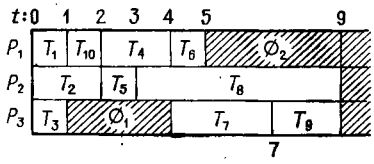


Рис. 1.2.

Символы s_i и f_i обозначают соответственно моменты начала и окончания выполнения задания T_i. Когда необходимо подчеркнуть зависимость от конкретного расписания S, используются обозначения $s_i(S)$ и $f_i(S)$.

Для задач, в которых предполагается наличие дополнительных ресурсов, мы будем пользоваться временной диаграммой только при $s=1$. В этом случае вертикальная ось обозначает количество требуемого дополнительного ресурса, число вертикальных сегментов соответствует количеству использованных процессоров. Такой пример дан чуть ниже на рис. 1.4, *б*.

Временная диаграмма на рис. 1.2 дает неформальное и интуитивное понятие о расписании. Несколько более формально расписание может быть определено как некоторое отображение, которое, в общем случае, сопоставляет каждому заданию последовательность из одного или более непересекающихся временных интервалов, лежащих в $[0, \infty)$, так что:

1. На каждый интервал назначается один процессор.
2. Сумма длин интервалов в точности равна времени выполнения задания, с учетом, если это необходимо, разной скорости выполнения на разных процессорах.
3. Никакие два интервала, относящиеся к разным заданиям, назначенным на один и тот же процессор, не перекрываются.

4. Учтены ограничения на порядок выполнения заданий и использование дополнительных видов ресурсов.

5. На отрезке $[0, \max\{f_i\}]$ не существует интервалов, на которых ни один из процессоров не был бы выделен какому-либо заданию (т. е. в расписании не разрешается иметь все процессоры свободными, если остаются незавершенные задания).

В расписаниях без прерываний для каждого задания отводится ровно один интервал, а при составлении расписаний с помощью списка еще накладывается требование, чтобы ни один процессор не мог быть свободным, если имеются готовые к выполнению задания, которые могут быть на него назначены. Мы не будем пытаться дальше углубляться в формализацию понятия расписаний, поскольку строгое математическое определение не является необходимым, к тому же оно чрезвычайно сложно для общей модели.

1.2.4. Показатели эффективности расписаний. Будем рассматривать два основных показателя эффективности, а именно: *длину расписания*, или *максимальное время завершения*,

$$\omega(S) = \max_{1 \leqslant i \leqslant n} \{f_i(S)\} \tag{1}$$

и *среднее взвешенное время завершения*, или *прохождения*,

$$\bar{\omega}(S) = \frac{1}{n} \sum_{i=1}^{n} w_i f_i(S). \tag{2}$$

Основная проблема, таким образом, заключается в нахождении эффективных алгоритмов, позволяющих находить среди всех расписаний (относящихся, возможно, к выделенным ранее классам) такие, для которых эти величины достигают минимума.

Здесь уместно показать, что алгоритмы составления расписаний с прерываниями, без прерываний и использующие список приоритетов дают различные результаты с точки зрения минимальной длины расписания и среднего взвешенного времени прохождения, причем «мощность» этих алгоритмов убывает в том порядке, в котором они перечислены. На рис. 1.3, *а*, *б*, *в*, *г* приведены, соответственно, граф системы заданий, расписание минимальной длины, в котором разрешены прерывания, расписание минимальной длины без прерываний и расписание минимальной длины, составленное с помощью списка приоритетов. Предоставляем читателю убедиться в оптимальности расписаний. Заме-

тим, что при $w_i=1$ $(1\leqslant i\leqslant n)$ среднее взвешенное время минимально во всех рассмотренных случаях и может быть использовано как критерий для составления расписаний с минимальной длиной.

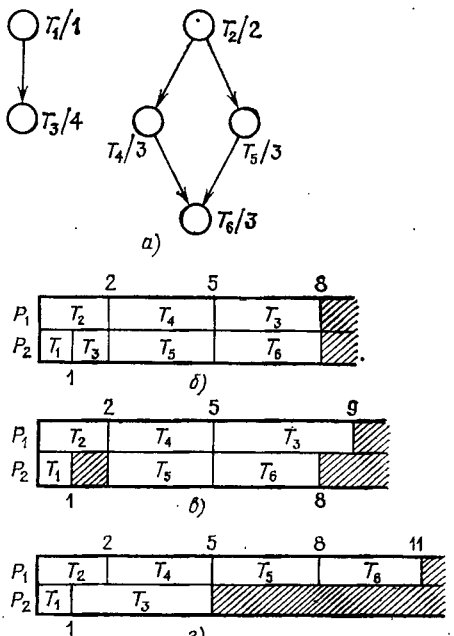


Рис. 1.3. Иерархия методов. *а*) $m=2$, идентичные процессоры, $s=0$, $\omega_i=1$ $(1\leqslant i\leqslant n\leqslant 6)$. *б*) Оптимальное расписание с прерываниями; $\omega=8$, $\bar{\omega}=29/6$. *в*) Оптимальное расписание без прерываний; $\omega=9$, $\bar{\omega}=30/6$. *г*) Оптимальное списочное расписание; $\omega=11$, $\bar{\omega}=32/6$.

Расписания с прерываниями могут быть более короткими по сравнению с расписаниями без прерываний и в том случае, когда отношение $\prec$ пусто (например, для $m=2$ и трех заданий с единичными временами выполнения). Заметим, однако, что для идентичных процессоров списочное расписание минимальной длины совпадает при пустом отношении $\prec$ с расписанием минимальной длины в классе всех расписаний без прерываний. Для критерия минимума среднего взвешенного времени прохождения при пустом отношении $\prec$ и идентичных процессорах списочное расписание также является оптимальным в этом классе. Таким образом, в указанных условиях различие между этими двумя классами отсутствуют.

Многие результаты, особенно относящиеся к сложности задач и приближенным и перечислительным методам, могут быть распространены и на другие важные критерии эффективности упорядочения. Например, предположим, что процессоры идентичны и величина $W_i(S)=f_i(S)-\tau_i$ определена как *время ожидания* обслуживания T_i в S. Как легко заметить, расписание, минимизирующее $\bar{\omega}$, одновременно минимизирует среднее взвешенное время ожидания.

Расширим общую модель, сопоставив каждому заданию T_i положительное число $d_i\,(1\leqslant i\leqslant n)$ которое назовем *директивным сроком* для этого задания. Директивный срок представляет собой момент времени, к которому желательно завершить выполнение задания. Определим *временнбе смещение* задания T_i в расписании S как разность $f_i(S)-d_i$ и *запаздывание* — как $\max\{0,\ f_i(S)-d_i\}$. Такие критерии эффективности, как минимальные и средние значения временного смещения и запаздывания, также представляют интерес. Как и в случае с временем ожидания, легко видеть, что расписание, минимизирующее среднее взвешенное время выполнения, минимизирует и среднее взвешенное временное смещение; для среднего взвешенного запаздывания это не так.

Существуют задачи, в которых директивные сроки *не должны нарушаться* (т. е. $f_i(S)\leqslant d_i,\ 1\leqslant i\leqslant n$); тогда они называются *крайними сроками*. Одной из таких задач, на которую мы хотели бы обратить особое внимание, является задача о нахождении минимального числа процессоров, необходимых для выполнения системы заданий к общему крайнему сроку $d_i=d,\ 1\leqslant i\leqslant n$, при условиях, что $<$ пусто, $s=0$, а процессоры идентичны. Данная задача связана с задачей об упаковке в контейнеры и рассматривается в гл. 5. Любопытно, что, как мы установили, она эквивалентна задаче нахождения расписания минимальной длины для случая идентичных процессоров при условиях: $<$ пусто, $\tau_i=1\ (1\leqslant i\leqslant n)$, $m\geqslant n$ и $s=1$. На рис. 1.4, *а*, *б* показана эта эквивалентность. Отметим, что если из списка условий исключить ограничение $m\geqslant n$, то получаем задачу, эквивалентную упорядочению с общим крайним сроком с целью минимизации числа процессоров при условии назначения на один процессор в целом не более m заданий. Хотя для нужд гл. 2—6 нет необходимости вводить в общую модель крайние сроки, дополненная система с произвольным набором $\{d_i\}$ кратко рассмотрена в следующем параграфе для случая, когда имеется один процессор, а также в п. 1.4.1.

В качестве последнего критерия эффективности, представляющего большой интерес, рассмотрим $\overline{N}(S)$ — среднее количество заданий, выполненных на интервале [0,

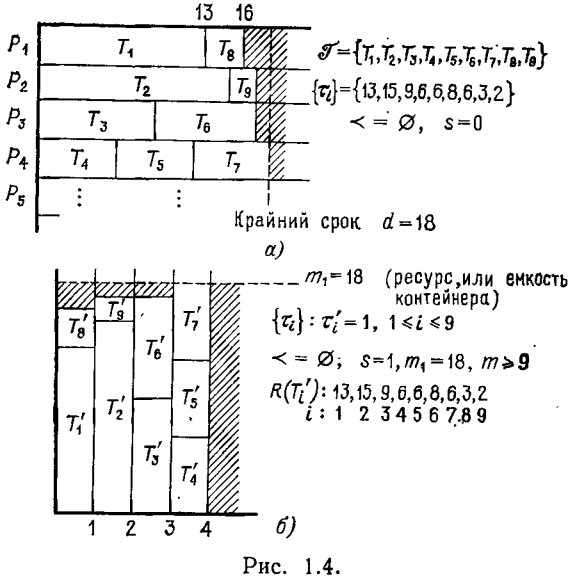


Рис. 1.4.

$\omega(S)$]. В общем случае эта величина характеризует ожидаемые потребности системы в процессорах или накопителях, возникающие при выполнении вычислений.

Мы можем записать

$$\overline{N}(S)=\frac{1}{\omega(S)}\int_0^{\omega(S)} N(t)\,dt,$$

где $N(t)$ есть количество заданий, не выполненных к моменту t. Очевидно, что $N(t)$ представляет собой убывающую ступенчатую функцию, которая изменяет значение в моменты окончания выполнения заданий (для нескольких заданий эти моменты могут совпадать). Действительно, интеграл можно представить в виде суммы

$$\overline{N}(S)=\frac{1}{\omega(S)}\{f_1(S)\,n+[f_2(S)-f_1(S)](n-1)+\ldots$$
$$\ldots+[f_n(S)-f_{n-1}(S)]\times 1\},$$

где нумерация заданий предполагается такой, что $f_i(S)\leqslant f_j(S)$ для $1\leqslant i<j\leqslant n$. При единичных весах в (2)

получим

$$\overline{N}(S) = n\frac{\bar{\omega}(S)}{\omega(S)}. \qquad (3)$$

Заметим, что данное выражение получено независимо от дисциплины обслуживания, числа процессоров и других параметров, отличных от времен окончания выполнения заданий. Из этого выражения сразу следует, что при $m=1$ (единственный процессор) $\overline{N}(S)$ минимально тогда, когда минимально $\bar{\omega}(S)$, так как при $m=1$ $\omega(S)$ не зависит от S. Позже мы рассмотрим случай $m\geqslant 2$.

1.3. Предшествующие результаты для случая одного процессора

В этом параграфе мы приведем классические результаты для частного случая, когда имеется один процессор ($m=1$), $<$ пусто и $s=0$. При $m=1$ задача минимизации $\max\{f_i\}$ не имеет смысла, поэтому будем рассматривать другие меры эффективности, введенные в предыдущем параграфе. Важнейшие результаты перечислены в следующих теоремах.

Теорема 1.1 [136]. *Минимальные значения среднего взвешенного времени прохождения, временного смещения и времени ожидания достигаются в том случае, когда назначение заданий на выполнение производится в порядке неубывания отношений* τ_i/w_i. *Поскольку* $\max\{f_i\}$ *фиксировано, то из* (3) *следует, что при таком расписании* $\overline{N}$ *также минимально.*

Теперь предположим, что имеются директивные сроки и задача заключается в нахождении расписаний, оптимальных в смысле временного смещения и запаздывания. Правило упорядочения в этом случае состоит в следующем.

Теорема 1.2 [81]. *Наибольшие величины временного смещения и запаздывания достигают минимума при обслуживании заданий в порядке неубывания директивных сроков.*

Наконец, для систем с директивными сроками определим величину $d_i-\tau_i-t$, которую назовем *резервным временем* T_i в момент t. Результат упорядочения заданий в соответствии с кажущимся привлекательным правилом по неубыванию резервного времени дается следующей несколько неожиданной теоремой.

Теорема 1.3 [28]. *Упорядочение по неубыванию резервного времени максимизирует минимальное временное смещение и минимальное запаздывание.*

Все три теоремы доказываются одинаковым способом, суть которого заключается в следующем. Пусть имеется последовательность S, обладающая требуемыми оптимальными характеристиками, но в которой нарушен указанный порядок следования. (В случае заданий с равными приоритетами упорядочиваем их произвольно и получаем линейный порядок.) Тогда должны существовать два соседних задания, которые нарушают порядок следования. Покажем, что перестановка этих заданий местами улучшит расписание, что противоречит предположению об оптимальности S.

Проиллюстрируем эту процедуру на примере доказательства первой части теоремы 1.1 для случая среднего взвешенного времени.

Пусть в рассматриваемой гипотетической последовательности S величины $f(i)$, $\tau(i)$ и $w(i)$ обозначают соответственно момент завершения, время выполнения и стоимость пребывания задания, занимающего i-е место в последовательности, и предположим, что

$$\frac{\tau(k)}{w(k)} > \frac{\tau(k+1)}{w(k+1)}, \quad \text{или} \quad w(k)\tau(k+1) < w(k+1)\tau(k). \quad (4)$$

Теперь рассмотрим, что произойдет с величиной среднего взвешенного времени $\bar{\omega}$ при перестановке местами k-го и $(k+1)$-го заданий. Перестановка приведет к увеличению времени окончания задания k на $\tau(k+1)$ единиц и внесет в $\bar{\omega}$ дополнительное слагаемое $(1/n)w(k)\tau(k+1)$. В то же время задание $k+1$ завершится на $\tau(k)$ единиц раньше, что приведет к уменьшению $\bar{\omega}$ на величину $(1/n)w(k+1)\tau(k)$. В результате получаем

$$\bar{\omega}' = \bar{\omega} + [w(k)\tau(k+1) - w(k+1)\tau(k)]/n.$$

Учитывая (4), получим $\bar{\omega}' < \bar{\omega}$, что противоречит минимальности $\bar{\omega}$.

Большое внимание привлекли задачи с директивными сроками для случая одного процессора, в которых в качестве показателя эффективности выступает величина запаздывания. С подходами, рассмотренными в гл. 6, связаны работы [98, 68], а также [121], где имеется обзор последних работ. В § 1.4 мы рассмотрим вопросы, связанные с вычислительной сложностью этих задач.

Хорошо известен результат, относящийся к задаче о нахождении такой последовательности, при которой минимизируется число заданий, завершенных с опозданием [114]. Оптимальный алгоритм заключается в следующем. Вна-

чале составляется последовательность в соответствии с правилом теоремы 1.2. Далее отыскивается первое задание T, выполнение которого завершено с опозданием, и из начальной последовательности, оканчивающейся заданием T, исключается такое задание T', которое имеет максимальное время выполнения. Этот процесс повторяется с новой последовательностью до тех пор, пока не останется запаздывающих заданий. Любая последовательность, начинающаяся с подпоследовательности, найденной описанным способом, содержит минимальное число заданий, завершенных с опозданием. Обобщения этой задачи рассмотрены в [135] *).

В [99] исследуется задача о минимизации максимального взвешенного времени завершения. Найден простой алгоритм, минимизирующий $\max\{w_i f_i\}$ для системы с одним процессором и произвольным $<$.

Представляет интерес оценка эффективности алгоритмов упорядочения путем их сравнения с процедурой случайного выбора. Важно также исследовать алгоритм, являющийся оптимальным в предположении о детерминированных временах выполнения заданий, в условиях, когда имеется лишь частичная информация (например, если задано распределение вероятностей). Исследование проблем, возникающих при такой постановке задачи, проведенное в последующих главах, показывает, что применение данного подхода во всех практически важных случаях связано со значительными трудностями. Тем не менее мы здесь приведем ряд результатов для системы с одним процессором с целью иллюстрации возможностей вероятностных моделей.

Рассмотрим алгоритм случайного выбора и алгоритм упорядочения по наименьшему времени выполнения (SPT—shortest processing time) в предположении, что $m=1$, $w_i=1$ $(1\leqslant i\leqslant n)$, а времена выполнения $\{\tau_i\}$ — независимые случайные величины с функцией распределения $F(\tau)$. Пусть $E(\tau)$ обозначает первый момент функции $F(\tau)$. Прежде всего, если $\tau(i)$ есть случайная величина, обозначающая время выполнения i-го задания, то

$$\sum_{i=1}^{n} f(i) = \sum_{i=1}^{n} \sum_{j=1}^{i} \tau(j) = \sum_{j=1}^{n} (n-j+1)\,\tau(j).$$

Отсюда для упорядочения методом случайного выбора имеем

$$E(\bar{\omega}_R) = \frac{n+1}{2} E(\tau). \qquad (5)$$

*) Дальнейшее обсуждение и ссылки см. в [9].

Предположим теперь, что мы выбрали какие-то значения времен выполнения и упорядочили последовательность в соответствии с правилом SPT. Для того чтобы рассчитать математическое ожидание $E(\bar{\omega}_S)$ среднего времени прохождения заданий для последовательности, вначале получим величину $E(\bar{\omega}_L)$, соответствующую обратному алгоритму, когда упорядочение производится по максимальному времени выполнения (LPT—largest processing time), а затем простым способом перейдем к $E(\bar{\omega}_S)$. Пусть выбранные значения времен выполнения пронумерованы в порядке возрастания их величин ($\tau_1 \leqslant \tau_2 \leqslant \ldots \leqslant \tau_n$). Тогда имеем

$$E(\bar{\omega}_L) = E\left[\frac{1}{n}\sum_{i=1}^{n}(n-i+1)\,\tau(n-i+1)\right] = E\left[\frac{1}{n}\sum_{i=1}^{n} i\tau(i)\right] = \frac{1}{n}\sum_{i=1}^{n} iE[\tau(i)].$$

Вычисляя распределение $F_i(\tau)$ для времени выполнения задания, занимающего i-е место, получим

$$dF_i(\tau) = \frac{n!}{(i-1)!\,(n-i)!}F^{i-1}(\tau)[1-F(\tau)]^{n-i}\,dF(\tau),$$

откуда получаем, что

$$E[\tau(i)] = \int_0^\infty \tau\,dF_i(\tau).$$

В частности, находим

$$E(\bar{\omega}_L) = E(\tau) + (n-1)\int_0^\infty \tau F(\tau)\,dF(\tau).$$

Теперь мы имеем соотношение

$$E(\bar{\omega}_S) = \frac{1}{n}\sum_{i=1}^{n}(n-i+1)\,E[\tau(i)] = (n+1)\,E(\tau) - E(\bar{\omega}_L),$$

и, следовательно,

$$E(\bar{\omega}_S) = nE(\tau) - (n-1)\int_0^\infty \tau F(\tau)\,dF(\tau). \qquad (6)$$

Это позволяет сравнивать результаты, полученные для последовательности, составленной с помощью алгоритма SPT, с результатами случайного выбора. Заметим, что

$E(\bar{\omega}_S)$ достигает максимума, когда дисперсия $F(\tau)$ равна нулю, и принимает при этом значение $E(\bar{\omega}_R)$.

Теперь рассмотрим задачу, отличающуюся от предыдущей тем, что информация о случайных величинах известна лишь частично. Пусть имеется семейство функций распределения $\{G_i(\tau)\}$, описывающих независимые случайные величины $\tau_1, \tau_2, \ldots, \tau_n$. Если известны только математические ожидания $E(\tau_i)$, то для последовательности, составленной по неубыванию значений $E(\tau_i)$, получим

$$E_1(\bar{\omega}) = \frac{1}{n}\sum_{i=1}^{n}(n-i+1)\,E(\tau_i). \qquad (7)$$

Оценим теперь погрешность, возникающую из-за того, что использовались только математические ожидания рассматриваемых величин.

Выберем некоторые значения времен выполнения в соответствии с функциями распределения $G_i(\tau)$ и расположим их в порядке неубывания. В этом случае для математического ожидания среднего времени прохождения может быть доказано следующее [28]:

$$E_2(\bar{\omega}) = \sum_{j=1}^{n}\left[E(\tau_j) - \frac{1}{n}\int_0^{\infty}\tau\left(\sum_{i\neq j}^{n}G_i(\tau)\right)dG_i(\tau)\right]. \qquad (8)$$

Идея доказательства состоит в следующем. Рассмотрим слагаемое $E_2(\bar{\omega}_j)$ внешней суммы, представляющее собой вклад задания T_j в $E_2(\bar{\omega})$. Для заданного значения времени выполнения τ задания T_j обозначим через $p_{in}(\tau)$ вероятность того, что задание T_j занимает i-е место в последовательности, содержащей n заданий. Непосредственное вычисление дает рекуррентные соотношения

$$p_{i,\,n+1}(\tau) = p_{in}(\tau)\,[1-G_{n+1}(\tau)] + p_{i-1,\,n}(\tau)\,G_{n+1}(\tau),$$
$$i = 2,\ 3,\ \ldots,\ n,$$
$$p_{1,\,n+1}(\tau) = p_{1n}(\tau)\,[1-G_{n+1}(\tau)],$$
$$p_{n+1,\,n+1}(\tau) = p_{nn}(\tau)\,G_{n+1}(\tau),$$

на основании которых по индукции получим формулу для $E_2[\bar{\omega}(j)]$. Базис индукции ($n=1$) легко получить из (8). Далее, пусть $E_2[\bar{\omega}_n(j)]$ обозначает результат для n заданий; тогда

$$E_2[\bar{\omega}_{n+1}(j)] = \frac{1}{n+1}\int_0^{\infty}\tau\sum_{i=1}^{n+1}[(n+1)-i+1]\,p_{i,\,n+1}(\tau)\,dG_j(\tau),$$

или

$$E_2[\bar{\omega}_{n+1}(j)] = \frac{n}{n+1} E_2[\bar{\omega}_n(j)] + \frac{E(\tau_j)}{n+1} - \frac{1}{n+1} \int_0^\infty \tau G_{n+1}(\tau)\, dG_j(\tau).$$

Подставив сюда $E_2[\bar{\omega}_n(j)]$ из (8), завершим индукцию:

$$E_2[\bar{\omega}_{n+1}(j)] = E(\tau_j) - \frac{1}{n+1} \int_0^\infty \tau \left(\sum_{l \neq j}^{n+1} (G_l(\tau)) \right) dG_j(\tau).$$

Сравнение (7) и (8) дает нам возможность оценить погрешность в определении характеристик расписаний в условиях, когда известны только математические ожидания времен выполнения.

1.4. Точные и приближенные результаты для задач минимизации длины расписаний и среднего времени прохождения

Результаты, приведенные в гл. 2 и 3, удобно рассматривать совместно с результатами гл. 4, так как обсуждение проблем вычислительной сложности позволяет оценить как значение результатов гл. 2 и 3, так и возможности их обобщения. Необходимо, однако, сделать ряд предварительных замечаний по поводу мер вычислительной сложности. Несмотря на краткость этих замечаний, их достаточно для понимания материала данного параграфа. Подробно эти вопросы рассмотрены в гл. 4.

В общем случае термин «сложность» алгоритма решения какой-либо задачи связывают только с временем получения решения, выраженного в виде функции от длины входа, т. е. количества битов, необходимых для задания конкретного набора параметров задачи. В нашем случае будем определять сложность в виде функции от основных параметров задачи, в первую очередь от количества заданий n. В ряде ситуаций это является существенным упрощением, которое, однако, вполне допустимо для наших целей. (Мы попросту полагаем, что время считывания и преобразования чисел всегда постоянно *), что оправдано с точки зрения практики.)

Детальное описание функций, определяющих сложность, зависит от особенностей алгоритма и структуры дан-

*) И не зависит от их длины. (*Прим. перев.*)

ных и не входит в сферу нашего рассмотрения. Поэтому мы будем пользоваться символом $O(\cdot)$, обозначающим зависимость по порядку величины, ограничиваясь тем самым описанием свойств, определяющих поведение функции на бесконечности. Так, если мы говорим, что алгоритм имеет сложность $O(n^2)$, то это просто означает, что существует постоянная c такая, что функция cn^2 ограничивает сверху время выполнения алгоритма как функции от n. Например, если необходимо упорядочить произвольную последовательность из n времен, то, как известно, существуют алгоритмы, использующие только попарные сравнения и имеющие сложность $O(n \log_2 n)$. Однако точное выражение для сложности в различных алгоритмах содержит величины порядка $O(n)$, $O(\log_2 n)$ или $O(1)$ и может иметь коэффициент перед $n \log_2 n$.

Алгоритм упорядочения, сложность которого ограничена полиномом от n, называется полиномиальным алгоритмом или алгоритмом, работающим полиномиальное время. Соответствующая задача называется полиномиально разрешимой. В дальнейшем мы будем называть алгоритм *эффективным*, если он полиномиален. Для такого определения имеются веские основания с практической точки зрения, особенно при больших n. В этой связи необходимо отметить, что задачи упорядочения, которые мы рассматриваем, относятся или могут быть сведены к конечным задачам в том смысле, что множество их решений конечно. Поэтому введенное понятие эффективного алгоритма соответствует непереборным алгоритмам; «неэффективные» алгоритмы требуют перебора (перечисления) множества решений и имеют сложность, определяемую, по крайней мере, экспоненциальной функцией от n. Однако в гл. 6, где рассматриваются приближенные и перечислительные алгоритмы *), понятие эффективности используется в обычном смысле, а не в указанном выше.

Существует группа задач, называемых NP-полными (иногда их называют полиномиально полными или просто полными), в которую входят многие классические трудноразрешимые задачи. К их числу относятся задача коммивояжера, задача нахождения хроматического числа графа, задача о ранце и другие. Известно, что с точки зрения сложности все NP-полные задачи эквивалентны друг другу в том смысле, что если найдется полиномиальный алгоритм

*) Под перечислительными алгоритмами понимаются алгоритмы направленного перебора. (*Прим. перев.*)

для одной из этих задач, то такой же алгоритм можно построить и для всех других задач данного класса в соответствии с процедурой перехода, которая, правда, различна для различных задач. Таким образом, либо алгоритм, имеющий полиномиальную сложность, существует для всех NP-полных задач, либо не существует ни для одной из них, причем неизвестно, какой из ответов правильный. Однако, несмотря на отсутствие ответа на этот «основной» вопрос, имеются большие основания считать, что NP-полные задачи чрезвычайно трудны. Как мы увидим в дальнейшем, почти все задачи составления расписаний в общем случае являются NP-полными *).

Предыдущие интуитивные рассуждения являются, конечно, совершенно неформальными. Для формализации такого анализа нужно разработать математические средства представления алгоритмов и конкретных задач, определить вычислительную сложность в терминах введенной модели, а также описать процесс сведения одной задачи к другой. Такая формализация позволяет для каждой конкретной задачи выразить ее сложность в виде функции от характерных для данной задачи параметров, установить зависимость между временно́й и емкостной сложностью, определить те аспекты задачи, которые дают наибольший вклад в ее сложность, и так далее. Математическая сторона этой проблемы рассмотрена в гл. 4. Далее мы, снова неформально, иллюстрируем применение этой техники, рассматривая новую комбинаторную задачу и доказывая, что она полиномиально разрешима, только если полиномиально разрешима некоторая известная NP-полная задача.

1.4.1. Задачи минимизации длины расписаний. В гл. 2 рассматриваются задачи минимизации длины расписаний для $m \geqslant 2$. В табл. 1.1 представлены результаты, полученные в данной области к настоящему времени. Чтобы облегчить изучение таблицы, необходимо сделать ряд замечаний.

1. Столбцы соответствуют различным параметрам задач; каждая строка определяет конкретную задачу. В некоторых позициях указаны значения параметров, при которых справедливы описанные результаты. Рассмотрим, например, задачу (строку) 2. В этой задаче m не является параметром, а имеет фиксированное значение $m=2$. Одинаковая для всех заданий величина τ_i также не является

*) Подробное изложение теории NP-полных задач с многочисленными примерами дано в книге М. Гэри и Д. Джонсона «Вычислительные машины и труднорешаемые задачи».— М.: Мир, 1982.— 416 с. (*Прим. перев.*)

Таблица 1.1

Результаты для задач минимизации $\omega = \max\{f_i\}$

Задача	Сложность задачи	m	$\{\tau_i\}$ *)	$\prec$	Дисциплина обслуживания	Ресурсы: s	Ресурсы: $\{m_i\}$	Ссылки
1	$O(n)$	—	равные	лес	без прерываний	0		[65]
2	$O(n^2)$	2	равные	—	без прерываний	0		[24]
3	не определена	фиксировано, $m \geqslant 3$	равные	—	без прерываний	0		
4	NP-полна	—	равные	—	без прерываний	0		[143]
5	NP-полна	фиксировано, $m \geqslant 2$	$\tau_i = 1$ или 2 для всех i	—	без прерываний	0		[143]
6	NP-полна	фиксировано, $m \geqslant 2$	—	ø	без прерываний	0		
7	$O(n \log_2 n)$	—	—	лес	с прерываниями	0		[109]
8	$O(n^2)$	2	—	—	с прерываниями	0		[109]
9	не определена	фиксировано, $m \geqslant 3$	—	—	с прерываниями	0		
10	NP-полна	—	—	—	с прерываниями	0		(см. гл. 4)
11	$O(n^3)$	2	равные	ø	без прерываний	—	—	[54]
12	NP-полна	фиксировано, $m \geqslant 2$	равные	лес	без прерываний	1	—	[54]
13	NP-полна	фиксировано, $m \geqslant 2$	равные	—	без прерываний	1	$m_1 = 1$	(см. гл. 4)
14	NP-полна	фиксировано, $m \geqslant 3$	равные	ø	без прерываний	1	—	[54]
15	$O(n \log_2 n)$	2	конвейер		без прерываний	0		[83]
16	NP-полна	фиксировано, $m \geqslant 3$	»		без прерываний	0		[57]
17	NP-полна	фиксировано, $m \geqslant 2$	система с последовательными приборами		без прерываний	0		[57]
18	NP-полна	фиксировано, $m \geqslant 3$	—	ø	Σf_i является минимальной	0		[32]

*) Везде, кроме задач 15—17, процессоры предполагаются идентичными.

параметром, так как ее конкретные значения не оказывают влияния на алгоритм. Частичное упорядочение является свободным параметром для этой задачи, однако не разрешены прерывания и не должны использоваться дополнительные ресурсы. Здесь и везде далее n предполагается параметром, принимающим любые допустимые значения.

2. В некоторых позициях столбца, в котором указана сложность, стоит запись «не определена»; это означает, что вопрос о сложности соответствующей задачи не решен.

3. Для каждой из задач, в которой запрещены прерывания и для которой известны оптимальные алгоритмы полиномиальной сложности, существует оптимальный алгоритм упорядочения с помощью списка приоритетов.

4. Важным моментом, касающимся NP-полных задач, приведенных в табл. 1.1, является то, что они представляют собой простейшие случаи, для которых доказана NP-полнота. Читатель может и должен сделать соответствующие выводы для более общих постановок. Снятие любого из ограничений на параметры исходной задачи, например рассмотрение неидентичных процессоров, непустого отношения предшествования, дополнительных ресурсов и т. д., очевидно приводит к задаче не менее сложной, чем исходная.

Одно важное наблюдение следует из сравнения строк 4 и 6. Задача 6 является NP-полной, следовательно, как легко видеть, такую же сложность имеет задача, в которой m произвольно. Обратное не всегда верно. Задача 4 относится к NP-полным, однако неизвестно, является ли таковой соответствующая ей задача 3 для любого фиксированного $m \geqslant 3$ (более того, были потрачены значительные усилия на отыскание эффективного алгоритма решения этой задачи при $m=3$).

5. Мы не беремся утверждать, что для всех полиномиальных алгоритмов, приведенных в таблице, указанная там сложность является минимальной. Однако, как показывает анализ, проведенный в последующих главах, во многих случаях имеются практически неопровержимые доказательства минимальности этих оценок. Например, алгоритмы, которые производят упорядочение времен выполнения, должны по меньшей мере иметь сложность $O(n \log_2 n)$ при условии, что в начальный момент все задания располагались в произвольном порядке. Алгоритмы, анализирующие структуру отношения предшествования, должны иметь сложность по меньшей мере $O(n^2)$, так как в графе, задающем порядок следования заданий, может быть $O(n^2)$ ребер (см., например, задачу 2). Как показано в гл. 2,

сложность зависит также и от способа представления исходных данных. Например, если граф задан в виде списка ребер, содержащего избыточные транзитивные ребра, сложность задачи может значительно возрасти. Задача 2 представляет собой соответствующий пример, так как указанная сложность достигается при отсутствии транзитивных ребер; заметим, что лучший из известных на сегодняшний день алгоритмов удаления транзитивных ребер из орграфа [2] имеет сложность $O(n^{2,8})$. Таким образом, если не делать предположения об отсутствии транзитивных ребер, сложность задачи 2 будет определяться сложностью этого алгоритма.

Алгоритмы решения задач 1, 2, 7 и 8 основываются, по существу, на нахождении критического пути в графе (они названы в гл. 2 *уровневыми* алгоритмами), при этом дисциплина назначения на каждом шаге представляет собой последовательный выбор из заданий, оставшихся на высшем уровне. (В действительности в задаче 2 к этому добавляется определение еще одной характеристики заданий, которая по отношению к графу предшествования является локально вычисляемой.) Ясно, что сложность таких алгоритмов определяется сложностью поиска критических путей в графе. Отметим, что обобщения «первого порядка» этих задач являются *NP*-полными, как показано в задачах 4, 5 и 10.

Задача 15 решается, по существу, путем соответствующего упорядочения заданий, следовательно, ее сложность определяется сложностью сортировки. Это самый ранний (1954 г.) из результатов, полученных в области минимизации длины расписаний. За исключением задачи 1 (1961 г.) и частных случаев задачи 16 (рассмотренных в гл. 2), остальные результаты получены после 1968 г.

Конвейерная задача, которая упоминается в строках 15 и 16, состоит в следующем. Каждая система заданий представляет собой набор n/m цепочек длины m, которые в литературе обычно называются работами, при этом имеется условие, что i-е задание в цепочке выполняется на процессоре P_i (n кратно m). В терминах величин $[\tau_{ij}]$, введенных в основной модели, столбцы $km+i$, $k=0, 1, \ldots, n/m-1$, соответствуют заданиям, которые должны выполняться на P_i, причем $\tau_{j,\ km+i}=\infty$ для всех $j\neq i$. Величины T_{km+i}, $i=1, 2, \ldots, m$, будут соответствовать последовательности заданий в работе k.

Задача 17 — это так называемая «простая задача с последовательными приборами», которая состоит в следую-

щем. Так же как и в конвейерной задаче, отношение $\prec$ задает набор цепочек произвольной длины, называемых работами (в общей задаче распределения работ отсутствует ограничение на тип отношения $\prec$). Каждая работа J_i характеризуется последовательностью пар (a_{ij}, P_{ij}), где a_{ij} обозначает время выполнения j-го задания в работе J_i, а P_{ij} — процессор, на котором это задание должно выполняться.

Общая задача является NP-полной для всех $m \geqslant 2$. В гл. 4 это не доказано, но в настоящем пункте мы опишем технику, применяемую для получения такого результата. Существует, однако, интересный частный случай, который мы здесь рассмотрим, когда имеется алгоритм с полиномиальной сложностью для $m=2$.

Каждая работа J_i содержит одну операцию для каждой машины (т. е. любая J_i представляет собой двухзвенную цепочку), при этом либо вначале выполняется задание на процессоре P_1, а затем на P_2, либо вначале — на P_2, а затем на P_1. Пусть (x_i, y_i) — времена выполнения заданий, составляющих J_i. Построим два упорядоченных набора C_1 и C_2 таких, что все работы J_i, в которых задание вначале выполняется на P_1 (соответственно на P_2), а затем на P_2 (соответственно на P_1), являются элементами C_1 (соответственно C_2), причем если (x_i, y_i) и (x_j, y_j) оба принадлежат C_1, то из $\min\{x_i, y_j\} < \min\{x_j, y_i\}$ следует, что J_i предшествует J_j в C_1 (это правило используется в задаче 15). Упорядочим таким же образом C_2. В соответствии с этим упорядочением назначаем на процессор P_1 задания вначале из C_1, а затем из C_2, выдерживая при этом ограничения на порядок следования. Аналогично, назначаем на P_2 задания вначале из C_2, а затем из C_1. Легко показать (см., например, теорему 2.12), что такая процедура назначения приводит к расписанию минимальной длины. Отметим, что x_i или y_i могут равняться нулю. Это позволяет рассматривать работы, состоящие из одного задания, либо для одной машины, либо для другой.

Труднорешаемость задач с дополнительными ресурсами становится ясной из рассмотрения задач 12—14. Задача 11, которая в этой книге не обсуждается, может рассматриваться как применение задачи о максимальном паросочетании [38]. За время, не превосходящее $O(n^3)$, можно построить неориентированный граф G такой, что $(T_i, T_j) \in G$ тогда и только тогда, когда T_i и T_j совместимы, т. е. когда $R_k(T_i) + R_k(T_j) \leqslant m_k$ для всех k, а значит, T_i и T_j могут выполняться параллельно. Максимальное паросочетание в

графе G дает кратчайшее расписание и может быть найдено за время $O(n^3)$.

Задача 18 обсуждается в § 1.5. Для задачи 7, в которой $<=\varnothing$, имеется алгоритм со сложностью $O(n)$, который будет рассмотрен в данной главе. В гл. 2 также обсуждаются эти задачи в связи с изучением способов количественного сравнения различных дисциплин упорядочения.

В заключение данного пункта кратко рассмотрим недавние результаты для задач, аналогичных рассмотренным, но с крайними сроками. Рассмотрим задачу распределения заданий равной длительности между двумя идентичными процессорами в случае произвольного частичного упорядочения и различных крайних сроков. В случае, когда частичное упорядочение задано в виде транзитивно замкнутого графа, известен способ [55], который позволяет за время $O(n^2)$ определить, существует ли расписание, при котором удовлетворяются все крайние сроки, и если оно существует, то позволяет его найти. Если желательно знать, существует ли расписание, при котором нарушается не более k крайних сроков, где k — фиксированное число, то это можно определить за время $O(n^{k+2})$. Однако задача о существовании расписания, при котором нарушается не более k крайних сроков, где k — переменное, является NP-полной даже для системы с одним процессором.

1.4.2. Результаты для задачи о среднем времени прохождения. В гл. 3 рассмотрены недавно полученные результаты, обобщающие результаты из предыдущего пункта для однопроцессорной системы. Вопросы, связанные с дополнительными ресурсами, не рассматриваются ввиду отсутствия соответствующих результатов. Различные виды дисциплины обслуживания заданий также не рассматриваются, так как во всех задачах предполагается составление расписаний без прерываний, причем по той же причине, что и в предыдущем случае. Напомним, впрочем, что возможность прерываний не приводит к улучшениям в случае идентичных процессоров, неотрицательных весов и пустого отношения предшествования. Замечания 1, 2, 4 и 5, сделанные по поводу табл. 1.1, относятся также и к табл. 1.2, которая содержит основные результаты гл. 3.

В задаче 1 табл. 1.2 существование алгоритма, имеющего полиномиальную сложность, показано для наиболее общего случая независимых заданий и равных стоимостей. Ранее были известны некоторые частные случаи, например алгоритм SPT, рассмотренный в § 1.3. Результаты, полученные для задачи 1, основаны на том, что она пред-

Таблица 1.2

Результаты для задачи минимизации $\sum_i \omega_i f_i$

Задача	Сложность задачи	Параметры				Ссылки
		$<$	$\{\omega_i\}$	$\{\tau_{ij}\}$	m	
1	$O(n^3)$	ø	равные	—	—	[12, 72]
2	NP-полна	набор цепочек	равные	идентичные процессоры	—	[18]
3	$O(n^2)$	лес	—	идентичные процессоры	1	[71]; см. также [48, 134]
4	не определена	—	—	идентичные процессоры	1	
5	NP-полна	ø	—	идентичные процессоры	фиксировано, $m \geqslant 2$	[11]
6	NP-полна	конвейерная система			фиксировано, $m \geqslant 2$	[57]

ставлена как частный случай общей транспортной задачи.

При решении задачи 3 используется локально вычисляемая функция стоимости (критерий упорядочения), сопоставленная каждой вершине (поддереву) леса. Эта функция в действительности является обобщением функции τ_i/ω_i, упомянутой в теореме 1.1. Использование упорядочения в соответствии со значениями вычисленной стоимости приводит к оптимальному расписанию путем рекурсивного выявления поддеревьев леса, которые должны выполняться следующими.

Задача 6 табл. 1.2 связана с проблемой конвейерной обработки, которой уделялось значительное внимание. Эта NP-полная задача используется в гл. 6 для иллюстрации перечислительных и приближенных методов составления расписаний.

За полиномиальное время может быть также решена задача, не охваченная нашей основной моделью, но являющаяся естественным обобщением задачи 3. Здесь снова имеется лес и однопроцессорная система, но каждое дерево леса, называемое работой, рассматривается как дерево решений в следующем смысле. Имеется произвольное (дис-

кретное) распределение вероятностей на множестве вершин, которое служит для принятия решения о том, какое задание (вершина) в данной работе должно выполняться следующим. Таким образом, выполнение работы состоит из последовательности, или цепочки, заданий, начинающихся в корне дерева и оканчивающихся в конечной вершине, при этом цепочка может быть прервана только в моменты окончания входящих в нее заданий. Данная последовательность и общее время ее выполнения являются случайными, следовательно, проблема заключается в минимизации ожидаемого значения среднего взвешенного времени завершения. Правило, в соответствии с которым принимается решение о том, из какой работы выбирать на выполнение следующее задание, представляет собой дальнейшее развитие идей, содержащихся в задаче 3. Вычислительная сложность существующих алгоритмов решения этой задачи определяется как $O(n^2)$.

Задача минимизации среднего значения задержки (см. предыдущий пункт) также относится к представленным в табл. 1.2, причем директивные сроки выступают в качестве параметра. В [14] доказано, что задача минимизации среднего взвешенного запаздывания является NP-полной для всех $m \geqslant 1$. Интересной нерешенной проблемой остается определение сложности задачи о минимизации средней задержки (при одинаковых весах) в случае $m=1$.

1.4.3. Сложность задач упорядочения. В гл. 4 развит формальный подход к определению сложности, который использован для того, чтобы прямо или косвенно доказать NP-полноту широкого класса различных задач упорядочения. Отличительной особенностью принятого там подхода является то, что в нем не используется в явном виде абстрактная теория вычислений, которая, собственно, и породила понятие сложности. Это сделано для того, чтобы привлечь внимание широкого круга читателей и предложить механизм, который может быть использован в практических задачах.

Изложение начинается с описания модели вычислительной машины, на которой реализуются алгоритмы решения комбинаторных задач; модель выбрана так, чтобы упростить анализ и сравнение сложности алгоритмов. Определяется понятие полиномиальных недетерминированных алгоритмов; они соответствуют задачам, которые решаются путем перебора, описываемого деревом поиска полиномиальной глубины. По определению существование такого рода алгоритма является необходимым для NP-полноты за-

дачи упорядочения. Далее обсуждаются тонкости, необходимые для проникновения в суть данной модели и выяснения ее ограничений. Вводится понятие полиномиальной сводимости комбинаторных задач, которым пользуются, когда хотят доказать, что та или иная задача имеет полиномиальное решение тогда и только тогда, когда некоторая NP-полная задача имеет такое решение.

Здесь уместно напомнить комментарий 4 к табл. 1.1. В нем говорится, что читатель получит возможность выявлять NP-полноту новых задач, пользуясь результатами гл. 4. Например, в гл. 4 показано, что для NP-полной задачи 6 из табл. 1.1 полиномиальный алгоритм существует тогда и только тогда, когда такой алгоритм существует для классической задачи с крайними сроками. Напомним, что в этой задаче задан общий для всех (независимых) заданий крайний срок d и спрашивается, существует ли такое расписание для m идентичных процессоров, при котором крайний срок d не нарушается ни на одном из процессоров? Доказанная ниже NP-полнота этой задачи непосредственно влечет за собой NP-полноту общей задачи с дополнительными ресурсами ($s \geqslant 1$) и задачи об упаковке в контейнеры, рассмотренных в гл. 5.

Опишем неформальный, но эффективный способ сведения комбинаторных задач на примере нескольких задач, не исследованных строго в гл. 4. Рассмотрим в качестве основы простую версию задачи о ранце, которая состоит в следующем. Пусть задан набор n целых положительных чисел $X=\{a_1, \ldots, a_n\}$ и целое число b; требуется узнать, существует ли подмножество элементов из X, сумма которых точно равна b? Другими словами, имеет ли уравнение $\sum_{i=1}^{n} c_i a_i = b$ решение при условии, что c_i принимают значение 0 или 1? Как хорошо известно, эта задача является NP-полной [86]. Мы используем этот факт для исследования сложности следующих задач упорядочения.

1. Рассмотрим задачу 7 из табл. 1.1 при $<=\varnothing$, которая, как легко установить, может иметь $O(n!)$ решений. Предположим, что в столбец «Дисциплина обслуживания» мы включили требование минимальности числа прерываний. Назовем такую задачу $PM(m)$-задачей для m процессоров.

Пример для случая $m=2$ и $\max\{\tau_i\} < \sum_{i=1}^{n} \tau_i/2$ показан на рис. 1.5, *а*. Заметим, что к такому же результату приводит и любой другой порядок выполнения этих зада-

ний; достаточно лишь в случае необходимости организовать прерывание последнего задания, выполняемого на P_1 (первого задания, выполняемого на P_2). Следовательно, в данном случае достаточно одного прерывания, а его

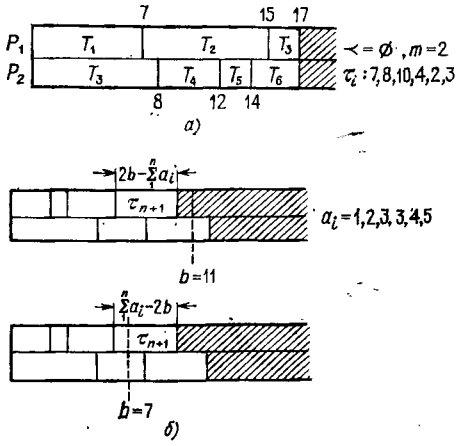


Рис. 1.5.

необходимость определяется существованием подмножества $\{\tau_i\}$, в котором сумма элементов точно равна $\sum_{i=1}^{n} \tau_i/2$. Таким образом, задача о ранце может быть сведена к PM (2)-задаче следующим образом. Пусть b и $\{a_i\}_{i=1}^n$ суть параметры задачи о ранце. Рассмотрим параметры PM (2)-задачи $\{\tau_i\}_{i=1}^{n+1}$, где $\tau_i = a_i$, $1 \leqslant i \leqslant n$, и $\tau_{n+1} = \left|2b - \sum_{i=1}^{n} a_i\right|$. Легко видеть, что данная PM (2)-задача имеет решение, при котором отсутствуют прерывания, тогда и только тогда, когда имеет решение исходная задача о ранце. На рис. 1.5, *б* представлен соответствующий пример. Очевидно, ввиду простоты сведения $\left(\text{вычисление } \left|2b - \sum_{i=1}^{n} a_i\right|\right)$ мы вправе ожидать, что PM (2)-задача имеет по крайней мере такую же сложность, как и задача о ранце. Такое же утверждение мы можем сделать относительно общего случая $m \geqslant 3$, так как можно добавить $m-2$ задания длины $\sum_{i=1}^{n} \tau_i/2$ и тем самым свести PM (2)-задачу к PM (m)-задаче.

2. Рассмотрим задачу 16 из табл. 1.1 и предположим, что работы (т. е. трехзвенные цепочки) описываются тройками (x_i, y_i, z_i), $1 \leqslant i \leqslant n$, где x_i (соответственно y_i и z_i) есть время выполнения первого (соответственно второго и третьего) задания на процессоре P_1 (соответственно на P_2 и P_3). Покажем, каким образом можно доказать NP-полноту этой задачи.

По заданным b и $\{a_i\}$ из задачи о ранце построим вход задачи 16 для $m=3$ в виде $(0, a_1, 0), \ldots, (0, a_n, 0)$,

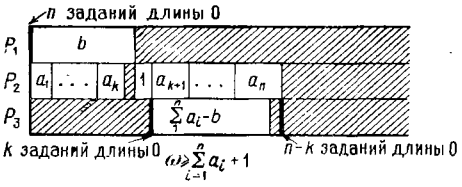


Рис. 1.6.

$\left(b, 1, \sum_{i=1}^{n} a_i - b\right)$. На рис. 1.6 представлен пример расписания. Заметим, что длина кратчайшего расписания не может быть меньше чем $\sum_{i=1}^{n} a_i + 1$. Но, как было показано, мы можем определить, является ли расписание оптимальным, только в том случае, если ответим на вопрос: существует ли набор величин a_i, сумма которых точно равна b? Следовательно, с помощью простого правила мы можем трансформировать входные данные задачи о ранце в исходные данные конвейерной задачи ($m=3$) таким образом, что оптимальное решение конвейерной задачи будет найдено в том и только в том случае, когда исходная задача о ранце имеет решение. Как и ранее, не представляет трудности сформулировать общую задачу для $m>3$ — это делается так же, как для $m=3$. Следовательно, мы вправе ожидать, что данная задача имеет по крайней мере такую же сложность, как и задача о ранце.

Такой же способ доказательства легко может быть применен к задаче 17. Читателям предлагается это проделать, учитывая отсутствие «упорядочения процессоров», имеющегося в конвейерной задаче. Важно отметить, что задача о ранце может быть решена за время, полиномиально зависящее от b (это не противоречит NP-полноте задачи о ранце, так как b может быть выражено с помощью $\log_2 b$ двоичных разрядов). Таким образом, если параметр, по отно-

шению к которому рассматривается сложность, есть сумма времен выполнения заданий, то из приведенных выше рассуждений не вытекает, что такие задачи труднорешаемы. Например, задача 6 из табл. 1.1 полиномиально разрешима для фиксированного $m \geqslant 2$ с точки зрения этого менее строгого показателя, характеризующего вход задачи. Однако задачи 16 и 17 остаются NP-полными даже в этом случае [57].

Как легко заметить, мы показали лишь, что рассмотренные задачи являются, по крайней мере, такими же сложными, как NP-полные задачи; мы не проверили, что они не являются более трудными. Полные доказательства NP-полноты мы оставляем тем, кто уже изучил гл. 4. Здесь необходимо отметить, однако, что общая схема метода сведения не разработана (в табл. 1.1 и 1.2 имеются задачи, для которых вопрос о сложности остается открытым) и что оценка сложности иногда оказывается очень трудоемкой задачей, как это подробно показано в гл. 4.

Мы завершаем обсуждение проблемы сложности рекомендацией, чтобы читатель, столкнувшийся с какой-либо задачей, самым пристальным образом исследовал все ее особенности и ограничения. Имейте в виду: NP-полнота общей задачи, которой соответствует данная конкретная задача, не должна убивать надежду найти эффективный оптимальный алгоритм. Специфические ограничения в конкретной задаче могут позволить построить алгоритм, сложность которого будет находиться в пределах полиномиальной зависимости от основных параметров этой задачи.

1.4.4. Оценки характеристик алгоритмов составления расписаний. Изучение вопросов сложности довольно ясно показывает, что наибольшего прогресса в области теории расписаний следует ожидать от исследования эвристических (приближенных) алгоритмов и разработки эффективных процедур перебора. В гл. 5, содержание которой мы здесь рассмотрим, основное внимание сосредоточено на простых процедурах (обычно имеющих сложность не более $O(n^2)$) и оценках их характеристик в наихудших случаях по сравнению с оптимальными алгоритмами. Вначале оценки характеристик устанавливаются для произвольных (неупорядоченных или случайных) правил составления расписаний; это делается для того, чтобы показать важность разработки эффективных эвристических алгоритмов и выявить комбинаторную структуру задач. Затем предлагаются легко реализуемые эвристические алгоритмы, из анализа оценок которых становится ясно, что даже простые эври-

стические правила могут обеспечивать адекватное решение многих NP-полных задач составления расписаний.

Мы ограничиваемся рассмотрением списочных (приоритетных) правил назначения без прерываний, при которых процессор никогда не остается свободным, если имеется задание, готовое к выполнению. Как уже было отмечено, это ограничение не приводит к ухудшению расписаний при условии независимости заданий и идентичности процессоров; впрочем, его нетрудно обосновать и в общем случае. Мы уже показали, что существующие оптимальные алгоритмы без прерываний могут рассматриваться как списочные правила составления расписаний. В такой форме представимы простейшие легко реализуемые эвристические алгоритмы, и, более того, трудно себе представить, что существуют их эффективные модификации, которые уже не укладываются в рамки списочных правил.

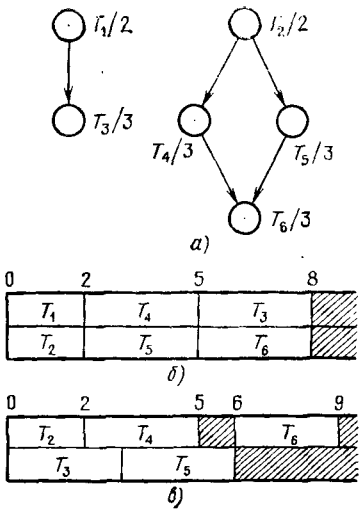


Рис. 1.7.

Исследование начинается с весьма подробно обсуждаемого примера анализа общей задачи минимизации длины расписания при условиях отсутствия дополнительных ресурсов, идентичности процессоров и произвольных m, $\{\tau_i\}$ и $<$. Рассматривается ряд аномальных случаев, показывающих, что при *уменьшении* времени выполнения заданий, *увеличении* количества процессоров и *устранении* ограничений на порядок следования длина расписания может *увеличиваться*. Аномальный случай, когда исключение задания из системы приводит к увеличению длины расписания, изображен на рис. 1.7. Анализ показывает, что длина нового расписания, полученного в результате таких изменений, не может быть более чем в $1+(m-1)/m'$ раз больше первоначальной, где $m' \geqslant m$ есть новое число процессоров. Кроме того, показано, что эта оценка является наилучшей в том смысле, что она достижима (по крайней мере, асимптотически).

В гл. 5 подробно рассмотрены алгоритмы упорядочения на основе критического пути и их возможности, при этом

приведено много примеров, служащих дополнением к материалу гл. 2. В частности, приводится полный анализ эффективности алгоритма LPT (критического пути) при независимых заданиях. В табл. 1.3 представлено большинство результатов, о которых пойдет речь в дальнейшем. (Если не оговорено противное, все оценки длин расписаний, указанные в данном пункте, являются наилучшими.) Отметим, что решение задачи 1 из табл. 1.3 вытекает из анализа влияния аномалий на оценки длин, о котором говорилось ранее.

Таблица 1.3

Оценки для списочных расписаний

Задача	Оценка	Параметры		Ссылки
		$L(\omega')$	$\prec$	
1	$\omega'/\omega \leqslant 2-\frac{1}{m}$	—	—	[45]
2	$\omega'/\omega \leqslant \frac{4}{3}-\frac{1}{(3m)}$	LPT (CP)	$\varnothing$	[46]
3	$\omega'/\omega \leqslant 1+\frac{1-1/m}{1+\lfloor k/m \rfloor}$	первые k элементов дают оптимальное расписание для k наибольших заданий	$\varnothing$	[46]
4	$\omega'/\omega \leqslant 1+(m-1)\frac{\max\{\tau_i\}}{\sum_{i=1}^{n}\tau_i}$	—	$\varnothing$	[46]
		CP	дерево	[85]
5	$\omega'/\omega \leqslant 2-1/m$	CP	—	см. § 5.2

Примечание. Оценка для задачи 3 неулучшаема при $k=0 \pmod m$.
Обозначения:
ω — длина оптимального списочного расписания;
CP — критический путь;
LPT — обслуживание в порядке наибольших времен выполнения;
$L(\omega')$ — список, порождающий ω';
$\lfloor x \rfloor$ — наибольшее целое число, не превосходящее x.

Затем рассматриваются задачи составления расписаний в предположении, что имеются дополнительные ресурсы ($s \geqslant 1$). Основной целью является нахождение оценок в наихудшем случае для произвольных расписаний. Значение этих результатов заключается в том, что они позволяют

проникнуть в суть этих трудных задач, а сами оценки показывают, что необходимы дальнейшие исследования в области эвристических алгоритмов. Значительное внимание уделено эффективности простых эвристических алгоритмов, в основном для случая, когда все $\tau_i = 1$. Основные результаты (в виде наилучших оценок) приведены в табл. 1.4; их обоснование, за исключением первого, неожиданно оказывается весьма трудной задачей.

Таблица 1.4

Оценки для задач с дополнительными ресурсами

Задача	Оценка	Параметры			Ссылки
		s	m	$\prec$	
1	$\omega'/\omega \leqslant m$	1	—	—	[49, 50]
2	$\omega'/\omega \leqslant s+1$	—	$m \geqslant n$	$\varnothing$	[49, 50]
3	$\omega'/\omega \leqslant \min\left\{\frac{(m+1)}{2}, s+2-\frac{(2s+1)}{m}\right\}$	—	$m \geqslant 2$	$\varnothing$	[50]

Обозначения:
ω' соответствует произвольному L;
ω соответствует оптимальному L.

В заключение главы рассмотрены оценки для задачи об упаковке в контейнеры, являющейся частным случаем рассмотренной ранее модели с дополнительными ресурсами, в которой $s=1$, $\prec = \varnothing$, все $\tau_i = 1$ и $m \geqslant n$. Подчеркнем значение данной задачи, сформулировав ее в виде задачи с крайними сроками, как показано на рис. 1.4. В дальнейшем приведены результаты для таких задач, в которых ограничения $s=1$ и $m \geqslant n$ сняты. Для простой задачи об упаковке в контейнеры рассмотрены четыре эвристических алгоритма; они описываются в терминах задачи с крайними сроками следующим образом.

Заданный произвольный список заданий последовательно просматривается, при этом используются следующие правила.

1. Назначение на первый подходящий процессор (FF : first-fit), при котором каждое задание назначается на процессор с минимальным номером из числа тех, которые подходят для этой цели (т. е. при таком назначении крайний срок не нарушается).

2. Назначение на лучший из подходящих процессоров (BF: best-fit), при котором каждое задание назначается на процессор, имеющий минимальное суммарное неиспользованное время (в пределах крайнего срока).

3. Назначение по убыванию на первый подходящий процессор (FFD: first-fit-decreasing), при котором назначение происходит, как в п. 1, но список заданий предварительно упорядочивается по невозрастанию времен выполнения.

4. Назначение по убыванию на лучший из подходящих процессоров (BFD: best-fit-decreasing), при котором действия остаются такими же, как в п. 3, с той разницей, что вместо первого подходящего процессора выбирается лучший.

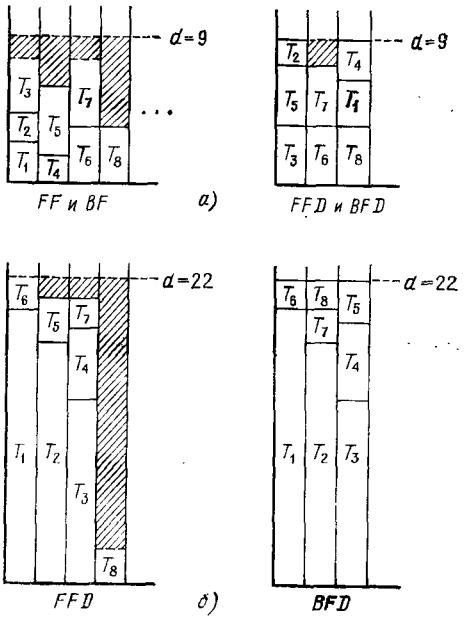


Рис. 1.8. Примеры решения задачи об упаковке в контейнеры. *а)* Пример эффекта упорядочения. Списки: (T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8) и (T_3, T_5, T_6, T_7, T_8, T_1, T_4, T_2), τ_i=3, 1, 4, 2, 4, 4, 4, 4. *б)* Пример, демонстрирующий преимущества методов BF и BFD: τ_i=20, 18, 14, 5, 3, 2, 2, 2.

Пример применения этих алгоритмов приведен на рис. 1.8. В табл. 1.5а сведены основные результаты для данных эвристических алгоритмов, а в табл. 1.5б представлены до-

Таблица 1.5а

Оценки для задачи об упаковке в контейнеры

Асимптотические оценки	Параметры		Ссылки
	$I(\alpha)$	A	
$R_A(\alpha)=17/10$	(1/2, 1)		[143, 52, 82]
$R_A(\alpha)=1+\lfloor\alpha^{-1}\rfloor^{-1}$	(0, 1/2]	FF и BF	[52, 82]
$R_A(\alpha)=11/9$	(1/2, 1]		
$R_A(\alpha)=71/60$	(8/29, 1/2]		[82, 77]
$R_A(\alpha)=7/6$	(1/4, 8/29]	FFD и BFD	
$R_A(\alpha)=23/20$	(1/5, 1/4]		
$R_A(\alpha)=1+\frac{(k-1)}{k(k-1)}$ *), $k=\lfloor\alpha\rfloor^{-1}$	(0, 1/4]		[78, 82]

*) Гипотеза.

Таблица 1.5б

Оценки для обобщенных задач

Асимптотические оценки	Параметры				Ссылки
	A	s	$<$	m	
$27/10-37/(10m)<R_A<27/10-24/(10m)$	—	1	ø	—	[91]
$R_A=2-2/m$	FFD	1	ø	$m\geqslant 2$	[91]
$R_A=s+7/10$	—	—	ø	$m\geqslant n$	[51]
$R_A=(17/10)s+1$	CP	—	—	$m\geqslant n$	[51]

полнительные результаты для некоторых случаев, когда ограничения на параметры ослаблены. Оценки для исходной задачи об упаковке в контейнеры асимптотически наилучшие. В терминах задачи составления расписаний при наличии дополнительных ресурсов эти оценки более точно вычисляются как функции наибольшей потребности в ресурсах в предположении, что объем дополнительного ресурса $m_1=1$ и что любая его доля может потребоваться при выполнении данного задания. В частности, пусть

$\omega_A(L)$ обозначает длину расписания (ей соответствует количество контейнеров), требуемую для назначения заданий из списка L в соответствии с правилом A. В табл. 1.5а символ A обозначает одно или несколько из рассмотренных выше правил, при этом количество процессоров (контейнеров) равно числу единиц времени, а времена выполнения заданий (как на рис. 1.8) рассматриваются как потребности в ресурсе. Пусть $\omega^*(L)$ обозначает минимальную длину расписания, которая требуется для выполнения заданий из L. В табл. 1.5а приведены значения для асимптотической оценки

$$R_A(\alpha) = \overline{\lim_{\omega^*(L)\to\infty}} \frac{\omega_A(L)}{\omega^*(L)},$$

где L пробегает все списки, для которых $\max\{R(T_i)\} \leqslant \alpha \in I(\alpha)$ и $I(\alpha)$ есть интервал (0, 1). В табл. 1.5б представлены результаты для смежных задач в виде асимптотических оценок, которые не являются функциями от $\max\{R(T_i)\}$.

Получение результатов, представленных в табл. 1.5, потребовало больших усилий. Для того чтобы привести доказательства всех этих результатов, пришлось бы удвоить объем книги. Все же те доказательства, которые удалось привести в полном объеме или кратко, должны облегчить изучение подобных комбинаторных задач.

Подход к решению комбинаторных задач, развитый в гл. 5, в которой проводится исследование многообещающих и быстродействующих эвристических алгоритмов для наихудших с точки зрения эффективности случаев, относится к сравнительно недавним достижениям, и он оказался плодотворным в других областях, например, при динамическом распределении памяти (см. гл. 5). Как показано в § 1.5, этот подход применим и к другим задачам упорядочения и может оказаться полезным при решении общей задачи составления расписаний. Например, можно легко показать, что произвольное расписание для конвейерной задачи может быть длиннее оптимального примерно в m раз. Представляет интерес исследование простых и удобных в применении эвристических алгоритмов и определение границ их эффективности. Это позволит определить, могут ли характеристики этих алгоритмов в наихудшем случае быть доведены до «приемлемого» уровня. Оказывается, что часто довольно трудно найти столь же простые хорошие эвристические алгоритмы, как рассмотренные в гл. 5. Имеется недавний обзор [34], в котором сравниваются эвристические алгоритмы для конвейерной задачи. Однако ясно, что не

обязательно иметь наилучшие (т. е. достижимые) оценки, и это обстоятельство может существенно облегчить математическую работу.

Возможно, наиболее ранней работой по определению границ эффективности алгоритмов составления расписаний является [39], в которой вычислены границы для среднего взвешенного времени прохождения. В частности, для $<=\varnothing$, $s=0$ и одинаковых процессоров можно показать, что

$$\max\left\{\bar{\omega}(n),\ \frac{\omega(1)}{m}+\frac{m-1}{2m}\bar{\omega}(n)\right\}\leqslant$$
$$\leqslant\bar{\omega}(m)<\frac{1}{m}\bar{\omega}(1)+\frac{m-1}{m}\bar{\omega}(n),$$

где $\bar{\omega}(k)$ есть среднее взвешенное время прохождения для k процессоров. Заметим, что $\bar{\omega}(1)$ и $\bar{\omega}(n)$ могут быть легко рассчитаны для набора $\{\omega_i\}$ стоимостей пребывания.

1.4.5. Итеративные и перечислительные методы составления расписаний. В соответствии с выбранной последовательностью изложения книга завершается главой, содержание которой так или иначе связано с предыдущими главами и которая посвящена практическим методам решения возникающих непосредственно на практике задач. Главной целью шестой главы является описание принципов создания эффективных точных и приближенных (или эвристических) алгоритмов в соответствии со следующими требованиями.

1. Формулировка должна быть достаточно общей, чтобы охватывать по возможности все практические приложения задач упорядочения.

2. При развитии того или иного подхода необходимо постоянно учитывать соотношение между вычислительными затратами и качеством получаемых решений.

3. Средства анализа должны обеспечивать возможность формального доказательства корректности алгоритмов, разработанных на основе метода ветвей и границ, и позволять изучать характеристики алгоритмов при изменении параметров задачи.

Проблема рассматривается в абстрактном виде, не зависящем от конкретных приложений, и, так же как в предыдущих главах, подвергается формальному анализу. Такие ограничения обусловлены не только общими принципами построения книги; они являются неизбежными, если принять во внимание следующие соображения: наличие

большого количества литературы, посвященной перечислительным и приближенным алгоритмам решения конкретных *NP*-полных задач, и тот факт, что эти частные приложения содержат очень много специальных методов, эффективность которых базируется на исследовании особенностей конкретных задач. Однако модели, общая методика и принципы, в соответствии с которыми решаются эти прикладные задачи, хорошо иллюстрируются при детальном изучении частной задачи о конвейере.

В шестой главе методы подразделяются на точные, чисто эвристические и с гарантированной точностью (т. е. такие, с помощью которых можно разработать и использовать эффективный алгоритм получения решения с заданной заранее точностью). Главные усилия направлены прежде всего на обобщение полученных ранее результатов и воплощение их в полностью параметризованной модели метода ветвей и границ, которая, в частности, включает в себя метод динамического программирования. Те, кто знаком с методом ветвей и границ, смогут оценить возможности построенной модели, ознакомившись со следующим перечнем ее параметров.

1. *Правило ветвления*, которое определяет процесс ветвления по отношению к дереву поиска; назначение правила заключается в разбиении пространства решений на непересекающиеся подмножества, которые либо подвергаются дальнейшему исследованию, либо отбрасываются.

2. *Правило выбора*, позволяющее выбрать следующую вершину дерева, с которой должно начаться ветвление.

3. *Характеристическая функция*, которая служит для отбрасывания тех поддеревьев полного дерева поиска, которые не содержат искомого решения.

4. *Отношение доминирования вершин*, используемое ниже, в п. 7.

5. *Функция нижней оценки*, которая ставит в соответствие каждому частичному решению нижнюю оценку его стоимости, служащую оценкой всех полных решений, получаемых из данного частичного решения.

6. *Верхняя граница стоимости*, которая равна стоимости некоторого полного решения, известного в начальный момент, или бесконечности, если ни одного решения вначале не известно.

7. *Правила исключения*, которые совместно с отношением доминирования и верхней границей стоимости служат для отбрасывания вновь возникающих или уже известных вершин на дереве поиска.

8. *Допуск* — действительное число на интервале [0, 1], которое представляет собой заданное максимальное относительное отклонение оптимального значения стоимости от стоимости приемлемого решения.

9. *Вектор предельных ресурсов*, компоненты которого представляют собой границы величин времени работы алгоритма, объема памяти для хранения активных вершин и непосредственных преемников ветвящихся вершин.

Доказательство корректности общего алгоритма проводится параллельно с рассмотрением изменений (в некоторых случаях аномальных) характеристик при изменении параметров. Затем дано описание эвристического алгоритма поиска в локальной окрестности и метода ветвей и границ с ограниченным возвратом, которые с успехом применяются на практике, при этом вновь обсуждается важный вопрос о соотношении между стоимостью вычислений и достигаемой точностью решения.

Далее в главе показано применение перечислительных и приближенных методов к нетривиальной конвейерной задаче, в которой требуется минимизировать среднее время прохождения (задача 6 из табл. 1.2). Представлены численные результаты и сделанные на их основании выводы.

Глава заканчивается обсуждением вопроса о близкой связи между подходами к построению переборных или приближенных методов с помощью динамического программирования и с помощью метода ветвей и границ. Подробно рассмотрен пример, иллюстрирующий эту связь.

1.5. Другие задачи упорядочения

В данном параграфе мы рассмотрим некоторые задачи упорядочения, имеющие непосредственное отношение к задачам, о которых идет речь в последующих главах, но отличающиеся от них. Начнем с теоретических результатов, связанных с задачей организации работы таких систем реального времени с крайними сроками, в которых осуществляется циклическое выполнение заданий. Затем рассмотрим задачи упорядочения, в которых используется более одной меры качества. Примером служит задача, занимающая 18-ю строку в табл. 1.1.

В заключение приведем некоторые замечания по общим вопросам, которые непосредственно примыкают к рассматриваемым в последующих главах, но не включены в них.

1.5.1. Однопроцессорное расписание для циклической системы заданий с крайними сроками. Существует очень много важных прикладных задач, возникающих в системах управления, диспетчирования и других системах реального времени, в которых выполнение заданий представляет собой вычисления, повторяемые периодически. При этом каждая реализация вычислений имеет фиксированный крайний срок, до которого она должна быть завершена. Крайний срок для данной реализации не может превосходить момента времени, в который потребуется начать очередную реализацию вычислений. Далее мы рассмотрим модель, в которой несколько заданий такого типа делят между собой время одного процессора и при этом должны соблюдаться крайние сроки (так называемый жесткий режим реального времени). Результаты, полученные в данной области, невелики, однако проводимые исследования, обусловленные важностью этих задач, несомненно, увенчаются многими новыми достижениями, подобными описанным в данной книге.

Определим формально модель рассматриваемой задачи как систему

$$\mathscr{S} = (\{T_i\},\ \{\tau_i\},\ \{I_i\},\ \{d_i\},\ \{t_{0i}\}),$$

где T_i, $1 \leqslant i \leqslant n$, есть задание, состоящее из периодической последовательности заявок на выполнение вычислений. При этом выполняются следующие условия.

1. Заявка на k-ю реализацию поступает к моменту $t_{0i} + (k-1)I_i$, $k=1, 2, \ldots$; I_i называется периодом T_i.

2. Для k-й реализации требуется τ_i единиц машинного времени.

3. Для k-й реализации установлен крайний срок $t_{0i} + (k-1)I_i + d_i$, $0 < \tau_i \leqslant d_i \leqslant I_i$, до наступления которого она должна быть завершена.

Рис. 1.9. Пример расписания с крайними строками: $I_1=10$, $I_2=6$, $I_3=15$, $d_1=6$, $d_2=3$, $d_3=11$, $t_{01}=0$, $t_{02}=4$, $t_{03}=9$, $\tau_1=4$, $\tau_2=2$, $\tau_3=1$; T_{ij} обозначает j-ю реализацию вычисления T_i.

На рис. 1.9 приведен пример такого расписания. Будем говорить, что алгоритм A допускает систему $\mathscr{S}$, если по-

рождаемое им расписание (содержащее в случае необходимости прерывания) выполнения заданий на одном процессоре таково, что реализуются все вычисления и не нарушаются крайние сроки.

Введем для произвольного алгоритма функцию

$$\mu_i(t) = \begin{cases} 1, & \text{если вычисление } T_i \text{ происходит в момент } t, \\ 0 & \text{в противном случае.} \end{cases}$$

Для допустимого расписания $\sum_{i=1}^{n} \mu_i(t) \leqslant 1$ для всех t.

Расписание полностью определено, если заданы функции $\mu_i(t)$ для всех i и всех $t \geqslant \min\{t_{0i}\}$. Мы говорим, что задание T_i является *активным* в момент t, если для некоторого $k \geqslant 0$ справедливо $t_{0i}+kI_i \leqslant t \leqslant t_{0i}+kI_i+d_i$ и

$$\int_{t_{0i}+kI_i}^{t} \mu_i(t)\,dt < \tau_i.$$

Алгоритм относительной срочности (RU — relative urgency), аналогичный алгоритму резервных времен из теоремы 1.3, назначает на процессор в каждый момент времени t такое активное задание, крайний срок которого ближе всего к t, т. е. для некоторого i

$$\mu_i(t) = \begin{cases} 1, & \text{если } t_{0i}+k_iI_i+d_i-t \leqslant t_{0j}+k_jI_j+d_j-t \\ & \text{для всех активных заданий } T_j, \\ 0 & \text{в противном случае,} \end{cases}$$

где

$$k_r = \left\lceil \frac{t-t_{0r}}{I_r} \right\rceil, \quad 1 \leqslant r \leqslant n.$$

Некоторая неопределенность возникает тогда, когда несколько заданий оказываются равноправными с точки зрения критерия RU. Будем пока предполагать, что этого не происходит, а позже вернемся к данному вопросу. Теперь мы хотим показать, что алгоритм RU является оптимальным в том смысле, что если система $\mathscr{S}$ допускается некоторым алгоритмом A, то она должна допускаться и алгоритмом RU.

Теорема 1.4 [97]. *Алгоритм* RU *является оптимальным* *).

Доказательство. Проведем доказательство от противного, т. е. предположим, что существует система $\mathscr{S}$, которую допускает некоторый алгоритм A, но не допускает

*) Частный случай этой задачи ранее был рассмотрен в [103].

RU-алгоритм. Нашей первой задачей будет описание таких RU-расписаний для системы $\mathscr{S}$, в которых нарушаются директивные сроки. Для этого необходимо ввести ряд определений.

Будем считать без потери общности, что $\min\{t_{0i}\}=0$, и предположим, что t_2 — самый ранний из нарушенных крайних сроков. Пусть $W_e(t)$ есть объем работы, выполненной процессором при исполнении заданий из $\mathscr{S}$ к моменту t, т. е.

$$W_e(t)=\sum_{i=1}^{n}\int_0^t \mu_i(x)\,dx.$$

Пусть $W_r(t)$ — объем работы, требуемый для выполнения тех же заданий, т. е.

$$W_r(t)=\sum_{i=1}^{n}\left[\frac{t-t_{0i}}{l_i}\right]\tau_i.$$

Определим t_0 как наибольшее время, не превышающее t_2, при котором $W_e(t)=W_r(t)$. Тогда мы можем высказать следующие утверждения.

1. $0\leqslant t_0<t_2$, так как $W_e(0)=W_r(0)=0$ и $W_e(t_2)<W_r(t_2)$.

2. Все реализации заданий, заявки на выполнение которых поступили к моменту t_0, завершены, так как $W_e(t_0)=W_r(t_0)$.

3. На интервале $(t_0, t_2]$ процессор должен быть полностью загружен RU-алгоритмом, так как для $t\in(t_0, t_2]$ $W_r(t)>W_e(t)$.

Введем величину t_1, представляющую собой последний момент времени, не превышающий t_2, когда процессор выполняет вычисление, входящее в систему $\mathscr{S}$, крайний срок которого превышает t_2, при условии, что такое вычисление имеется; в противном случае $t_1=t_0$. Мы утверждаем, что ни одно вычисление, заявка на которое поступила до момента t_1, не производилось в любой момент времени, находящийся в интервале $(t_1, t_2]$. Поскольку для случая $t_1=t_0$ это утверждение тривиально, предположим, что $t_1>t_0$. Пусть T обозначает реализацию задания, которое выполняется на интервале $(t_1, t_2]$, но заявка на которое поступила до момента t_1.

По определению t_1, крайний срок для T должен предшествовать t_2. Но, по определению RU-алгоритма, вычисление, выполняющееся в момент t_1, должно иметь крайний срок меньший, чем у задания T, следовательно, меньший, чем t_2.

Это противоречит определению t_1 и доказывает справедливость сформулированного утверждения.

Таким образом, интервал $(t_1, t_2]$ обладает следующими свойствами.

1. Вычисление может выполняться в интервале $(t_1, t_2]$ только в том случае, если заявка на него поступила в данном интервале и его крайний срок не превышает t_2.

2. Если заявка на вычисление поступила в интервале $(t_1, t_2]$ и если его крайний срок строго меньше t_2, то этот крайний срок будет соблюден.

3. Существует хотя бы одно вычисление, заявка на которое поступила в интервале $(t_1, t_2]$, у которого крайний срок равен t_2 и этот срок не соблюдается.

Остается сделать небольшой шаг к доказательству теоремы. Пусть C_1 есть набор вычислений, заявки на которые поступили в интервале $(t_1, t_2]$ и которые имеют крайние сроки, строго меньшие t_2. Пусть, далее, C_2 есть непустой набор вычислений, заявки на которые поступили в этом же интервале и которые имеют крайний срок, равный t_2. Если обозначить через W_1 время, выделенное RU-алгоритмом вычислениям из C_1, то получим

$$W_1 = t_2 - t_1 - \delta,$$

где δ есть время, выделенное на интервале $(t_1, t_2]$ вычислениям из набора C_2. Объем W_2 работы, проделанной на интервале $(t_1, t_2]$ с помощью гипотетического алгоритма A, должен быть больше, чем W_1 (по определению C_1 и C_2). Следовательно, мы можем записать $W_2 = W_1 + \delta'$, где δ' есть время, которое было затребовано и, следовательно, выделено для выполнения вычислений из C_2 и, возможно, также и для какой-либо другой работы. Поскольку алгоритм A допускает $\mathscr{S}$, мы имеем $\delta' > \delta$ и

$$W_2 = t_2 - t_1 + \delta' - \delta > t_2 - t_1.$$

Полученное противоречие доказывает теорему.

Теперь вернемся к вопросу о равноправных заданиях при применении RU-алгоритма. Формальное решение этой проблемы, согласующееся с RU-критерием, заключается в том, чтобы все задания с ближайшим общим крайним сроком делили время процессора между собой на равные части, осуществляя тем самым режим «разделения времени». То есть, если T_i представляет собой одно из k заданий (вычислений) с общим ближайшим крайним сроком, то считаем, что $\mu_i(t) = 1/k$ в течение всего времени, пока данная ситуация имеет место. С практической точки зрения, не нарушая

оптимальности, можно заранее рассчитать самый ранний момент времени, при котором возникнет такая ситуация, разделить найденный интервал на k равных частей и распределить эти части между «равноправными» заданиями.

Большой интерес представляет определение необходимых и достаточных условий, которым должны удовлетворять параметры системы, чтобы она допускалась оптимальным (в указанном выше смысле) RU-алгоритмом. Ясно, что наилучшее из того, что можно предпринять,— это постоянно держать процессор загруженным, а это дает необходимое условие $\sum_{i=1}^{n} \tau_i/I_i \leqslant 1$. Достаточное условие в общем случае не найдено. Однако для интересного частного случая, когда $d_i=I_i$, $1\leqslant i\leqslant n$, указанное выше необходимое условие является также достаточным. (Отметим, что данный частный случай соответствует таким прикладным задачам, в которых вычисления производить никогда не поздно, лишь бы их окончание происходило раньше, чем начнется следующий цикл.) Действительно, рассмотрим такой алгоритм разделения времени A, который в моменты времени $t_{0i}+kI_i$ ($1\leqslant i\leqslant n$; $k=0, 1, 2, \ldots$) выделяет для реализации вычисления T_i кванты времени, равные τ_i/I_i. Очевидно, каждый крайний срок будет соблюден в точности, так как $d_i=I_i$. Более того, поскольку мы предполагаем, что $\sum_{i=1}^{n} \tau_i/I_i \leqslant 1$, то, следовательно, алгоритм A допускает рассматриваемую систему. Принимая во внимание, что RU-алгоритм пригоден для всех систем, для которых пригоден алгоритм A, получаем следующую теорему.

Теорема 1.5. *RU-алгоритм будет допускать систему* $\mathscr{S}$, *в которой* $d_i=I_i$, $1\leqslant i\leqslant n$, *тогда и только тогда, когда* $\sum_{i=1}^{n} \tau_i/I_i \leqslant 1$.

Пусть по-прежнему $\tau_i/I_i\leqslant 1$; тогда естественное обобщение предшествующей модели на случай $m\geqslant 2$ процессоров приводит аналогичным образом к тому, что условие $\sum_{i=1}^{n} \tau_i/I_i \leqslant m$ является необходимым и достаточным для существования допускающего алгоритма.

Вопрос об использовании процессора при произвольных правилах упорядочения или распределениях приоритетов для заданий был исследован лишь для частного случая $t_{0i}=t_{0j}$ и $d_i=I_i$ для всех i и j [103]. Будем говорить, что систе-

ма полностью использует процессор при данном алгоритме диспетчирования, если увеличение любых значений τ_i приводит к тому, что система перестает быть допустимой. При сформулированных предположениях показано, что существуют системы из n заданий, полностью использующие процессор, при этом коэффициент загрузки (доля времени, когда процессор занят) не превышает $n(2^{1/n}-1)$.

1.5.2. Упорядочение с использованием вторичных показателей эффективности. Желательно знать такие характеристики расписаний, как длина, среднее время выполнения, среднее число обслуживаемых заданий, максимальное и среднее запаздывание, хотя используются они обычно в различных целях. С другой стороны, при планировании вычислительного процесса или при распределении работ может возникнуть необходимость использовать более одного показателя эффективности. Например, с точки зрения пользователя ЭВМ очень часто желательно иметь минимальное среднее время завершения (или ожидания), тогда как с точки зрения руководства вычислительного центра желательно иметь расписания с минимальной длиной, чтобы достигнуть максимальной пропускной способности. Несмотря на важность этой проблемы, ей посвящено небольшое количество работ. В данном пункте мы рассмотрим три таких работы, две из которых совсем недавние. Во всех работах предполагается $s=0$.

1.5.2.1. *Директивные сроки и (как вторичный показатель) среднее время прохождения.* Предположим, что $<$ пусто, $w_i=1$ для всех i, и пусть заданы директивные сроки d_i, как в § 1.2. В сравнительно ранней работе [136] рассматривается задача минимизации среднего времени завершения для однопроцессорной системы, если известно, что все директивные сроки могут быть соблюдены (т. е. максимальное запаздывание равно 0). Теорема 1.2 дает простую (имеющую сложность $O(n \log_2 n)$) процедуру определения, является ли это возможным. В этом случае описываемый далее алгоритм состоит из SPT-упорядочения, осуществляемого так, чтобы сохранить выполнение директивных сроков [136].

Для набора заданий $\mathscr{T}$ алгоритм заключается в следующем. Вначале нужно найти в $\mathscr{T}$ наибольшее задание T, директивный срок которого не превышает величину $\sum_{i=1}^{n} \tau_i$ (длина расписания для $\mathscr{T}$), и поставить это задание последним в очереди на выполнение (на n-е место). Затем нужно найти наибольшее задание в системе $\mathscr{T}-\{T\}$, директивный

срок которого не превышает $\sum_{i=1}^{n} \tau_i - \tau$ (длина расписания для $\mathscr{T} - \{T\}$), и назначить его на $(n-1)$-е место. Процедура рекурсивно продолжается до тех пор, пока останется только одно задание. Таким образом, мы имеем относительно простой полиномиальный алгоритм, минимизирующий среднее время выполнения при соблюдении директивных сроков. Оптимальность доказывается непосредственно путем перестановки двух заданий, как это показано в § 1.2.

1.5.2.2. *Длина расписания и (как вторичный показатель) среднее время прохождения.* В рассматриваемой здесь задаче $<=\varnothing, m>1$, процессоры идентичны, а величины стоимостей пребывания единичные. Последовательность, порождаемая алгоритмом SPT, является оптимальной по критерию среднего времени; поэтому можно ожидать, что последовательность, порождаемая алгоритмом LPT, должна давать по этому критерию очень плохой результат (действительно, для $m=1$ — наихудший). С другой стороны, расписание, порождаемое LPT, должно быть в среднем короче, чем расписание, порождаемое SPT. (Это утверждение согласуется с табл. 1.6.) Одно из решений возникшей дилеммы

Таблица 1.6

Оценки для SPT-упорядочения

Оценки	Параметры		Ссылки
	$L(\omega)$	$L(\omega')$	
$\omega'/\omega \leqslant 2 - 1/m$	LPT	SPT	[11]
$\omega'/\omega \leqslant 3/2$	SPT	$\overline{\text{SPT}}$	[32]
$\omega'/\omega \leqslant (5m-2)/(4m-1)$	$\overline{\text{SPT}}$	LPT	[12]
$\omega'/\omega \leqslant (5m-4)/(4m-3)$	$\overline{\text{SPT}}$	SPT_L	[32]

очень простое и заключается в том, чтобы построить расписание, обратное LPT, называемое RPT-расписанием. При этом длина расписания остается неизменной, но на каждом отдельном процессоре задания выполняются в порядке неубывания. Пример приведен на рис. 1.10.

Известно, что можно подобрать примеры, в которых отношение $\bar{\omega}_{\text{LPT}}/\bar{\omega}_{\text{SPT}}$ становится сколь угодно большим, в то время как для RPT выполняется оценка $\bar{\omega}_{\text{RPT}}/\bar{\omega}_{\text{SPT}} \leqslant m$,

являющаяся наилучшей в том смысле, что она может быть достигнута с любой степенью близости [11]. Однако результаты моделирования показывают, что эта оценка весьма неинформативна, так как для очень широкого класса распределений времен выполнения заданий среднее время

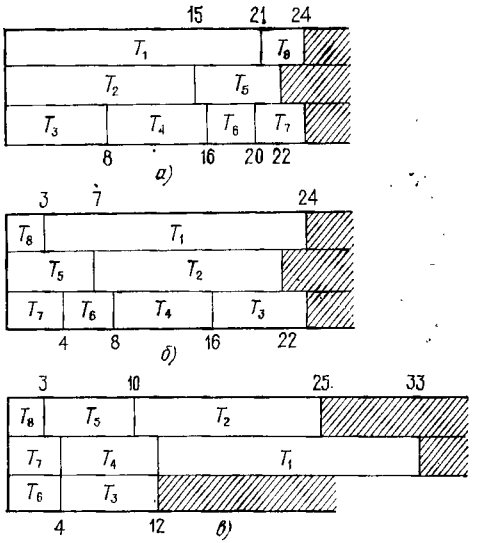


Рис. 1.10. Пример сравнения (*а*) LPT-, (*б*) RPT- (обратного LPT) и (*в*) SPT-расписаний.

прохождения RPT-расписания всегда оказывалось лишь на несколько процентов хуже оптимального.

1.5.2.3. *Среднее время прохождения и (как вторичный показатель) длина расписания.* Снова предполагаем, что $<=\varnothing$, $m>1$, процессоры идентичны, а величины стоимостей пребывания единичные. Теперь мы обратимся к задаче нахождения расписания с минимальной длиной, при котором минимизируется среднее время прохождения. Вначале убедимся, что для произвольного m существует большое количество расписаний, минимизирующих среднее время прохождения (все они называются SPT-расписаниями). Заметим, что среднее время прохождения для системы $\mathscr{T}$ из n заданий может быть записано как

$$\bar{\omega} = \frac{1}{n} \sum_{i=1}^{n} k_i \tau_i, \tag{9}$$

где k_i есть число, на единицу большее, чем количество заданий, следующих за T_i на данном процессоре. Данная формула отражает тот факт, что время выполнения T_i (для каждого i) входит как в его собственное время завершения, так и во времена завершения всех заданий, следующих за T_i на данном процессоре. Теперь предположим, что n кратно m (этого всегда можно достичь, добавив необходимое количество заданий с нулевым временем выполнения). Назовем m заданий, которые выполняются j-ми по счету на соответствующих процессорах, заданиями *ранга* j. Если расписание минимизирует среднее время прохождения, то задания ранга 1 оказываются самыми короткими, задания ранга 2—чуть длиннее и так далее. Из (9) видно, что в SPT-расписании в случае m процессоров мы можем любым способом переставить m заданий одного ранга, не изменяя $\bar{\omega}$. Это означает, что при такой перестановке мы не выходим за рамки SPT-расписаний. Таким образом, существует $O(m!^{n/m})$ SPT-расписаний.

Строгое (списочное) SPT-расписание образуется при упорядочении списка заданий по неубыванию времен выполнения и является самым длинным из всех SPT-расписаний (оно называется $\overline{\text{SPT}}$-расписанием), при этом наибольшие задания каждого ранга оказываются на одном и том же процессоре. Таким образом, задача заключается в том, чтобы найти такой набор перестановок, при котором достигается минимальная длина SPT-расписания (он называется $\underline{\text{SPT}}$-расписанием). К сожалению, эта задача *NP*-полна, как указано в строке 18 табл. 1.1. Изучался эвристический алгоритм решения этой задачи, заключающийся в том, что задания назначаются на выполнение в порядке убывания рангов, т. е. задания ранга j назначается после заданий ранга $j-1$. При этом LPT-правило локально применяется к соседним рангам и $i < j$ влечет, что все задания ранга i имеют меньшее время выполнения, чем все задания ранга j. Таким образом, после того, как распределены задания ранга $j-1$, назначаются задания ранга j по одному на каждый процессор. Назначение осуществляется в LPT-порядке с учетом времени завершения заданий ранга $j-1$. Назовем это SPT_L-правилом. В табл. 1.6 собраны известные результаты по наилучшим оценкам для правил $\underline{\text{SPT}}$, LPT, $\overline{\text{SPT}}$, и SPT_L.

Доказательство результатов, приведенных в первой и второй строках, относительно простое, а для получения третьей и четвертой строк требуется большое искусство.

Пример, приведенный на рис. 1.11, позволяет проникнуть в комбинаторную сущность этих задач, показывая возможность достижения или асимптотического приближения к двум последним границам.

1.5.2.4. *Другие критерии эффективности.* Другим первичным или вторичным критерием может быть среднее

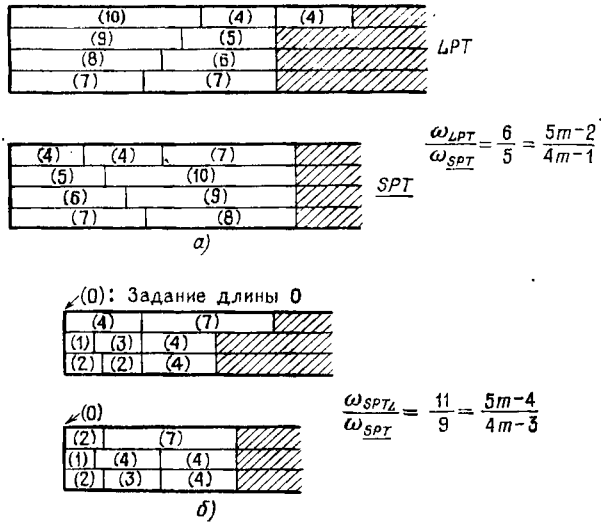


Рис. 1.11.

число заданий $\overline{N}$, находящихся на обслуживании. Как было показано в § 1.2, для системы с n заданиями имеем $\overline{N} = n\bar{\omega}/\omega$. При $m = 1$ $\overline{N}$ минимизируется SPT-алгоритмом. Однако, как показано на рис. 1.12, для $m > 1$ существуют списочные расписания, минимизирующие $\overline{N}$ и не являющиеся SPT-расписаниями. В действительности для $m \geqslant 1$ и $< = \varnothing$ можно показать, что строгое SPT-расписание может оказаться в $2m/(m+1)$ раз хуже оптимального по значению критерия $\overline{N}$ [26]. Устремляя k к ∞ и ε к 0 при постоянном $k\varepsilon$, убеждаемся (см. рис. 1.12,*б*), что эта оценка является наилучшей при $m = 2$. (Пример легко распространяется на произвольные m.) Рассмотрение примеров приводит к мысли о том, что оценка $2m/(m+1)$ является не очень информативной. Однако проблема нахождения расписания, минимизирующего $\overline{N}$, как легко показать, относится к числу *NP*-полных; поэтому оценка

эффективности эвристических алгоритмов с помощью моделирования окажется значительно более дорогостоящим делом, чем упомянутое ранее сравнительное изучение средних времен прохождения для правил SPT и RPT.

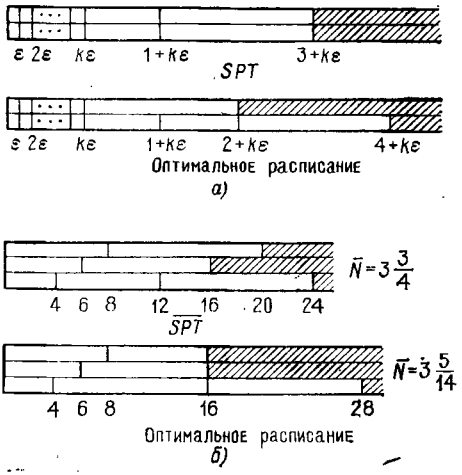


Рис. 1.12.

В связи с задачами, рассматриваемыми в данном параграфе, уместно упомянуть так называемые таймерные задачи, или задачи управления процессом обслуживания, которые не охватываются материалом этой книги. Для задач такого типа в первую очередь важно найти хотя бы одно правильное решение, а не заниматься оптимизацией. Мы имеем в виду прежде всего проблемы тупиков, взаимного исключения, детерминированности и синхронизации. Материалы по этим вопросам можно найти в [22, 17]. На практике решение данных проблем нужно обеспечить в первую очередь, после чего можно улучшать среднюю или максимальную длительность обслуживания. Заметим тем не менее, что решение трех последних из названных задач управления процессом обслуживания непосредственно связано с правильностью разработки спецификаций системы заданий и, следовательно, с точки зрения нашего исследования может считаться очевидным.

Задача о тупиковых ситуациях, в свою очередь, приводит к обобщению модели дополнительных ресурсов, рассмотренной в гл. 5. В частности, в такой обобщенной модели считается, что задание может захватывать и освобождать дополнительные ресурсы в произвольный момент своего

выполнения. При этом естественные ограничения состоят в том, что (а) заявки на дополнительные ресурсы в любой момент времени не превышают объемов этих ресурсов в системе и (б) все ресурсы, используемые заданием, освобождаются перед его завершением. Отметим, что если ресурсы не являются делимыми (не могут быть изъяты у задания, которое в данный момент ими пользуется), то может возникнуть ситуация, называемая тупиковой: два или более активных задания для своего продолжения требуют дополнительных ресурсов, но, ввиду ограниченного их количества, ни одно из заданий не может выполняться.

Учет этих факторов требует существенного обобщения базовой модели, которая и без того весьма сложна. В настоящее время имеется лишь небольшое число результатов, относящихся к задачам, в которых рассматриваются вторичные критерии эффективности, однако важность этих проблем несомненно послужит стимулом для дальнейшего развития исследований в этой области.

1.5.3. Заключительные замечания. Как можно заметить, предмет данной книги относится, в основном, к области комбинаторики. Однако структура задач и методы, используемые для их решения, существенно выходят за узкие рамки детерминированных задач упорядочения. Это подтверждается материалами гл. 6, в которой развиты перечислительные и приближенные методы, являющиеся общими для комбинаторных задач. Результаты в области сложности, изложенные в гл. 4, хотя и ориентированы на задачи составления расписаний, но могут также применяться при решении многих других задач назначения, размещения или распределения, возникающих в приложениях исследования операций. Подход к решению комбинаторных задач, развитый в гл. 5, может применяться в других областях, где есть основания считать, что очень быстрые эвристические алгоритмы могут оказаться практически эффективными. Краткое обсуждение этого вопроса содержится в гл. 5, предшествуя анализу задачи об упаковке в контейнеры. Результаты гл. 2 и 3, а также некоторых частей гл. 4 и 5, по-видимому, наиболее тесно связаны собственно с теорией расписаний. Но и при решении других комбинаторных задач читатель может руководствоваться общими принципами и техникой доказательств, которые с успехом применяются в детерминированной теории расписаний.

С математической точки зрения книга не является легкой для чтения. При этом, как мы отметили ранее, в нее было невозможно включить доказательства всех результатов.

Хотя доказательства, приведенные или намеченные в общих чертах в книге, обычно содержат тонкие моменты, следует иметь в виду, что абсолютно строгое доказательство часто сводится к трудоемкому перебору различных случаев. Как бы то ни было, математическая сторона книги верно отражает природу исследуемой области. Она также частично объясняет попытку дать в этой главе как можно более полный обзор известных на сегодня результатов по теории расписаний; его цель заключается в том, чтобы ознакомить студентов и исследователей с современным состоянием специальных разделов теории расписаний без предварительного изучения большого количества литературы.

В качестве последнего замечания укажем, что теория массового обслуживания представляет собой дополнительное средство для изучения систем обслуживания. Она является дополнительным средством в том смысле, что имеет дело, в основном, с анализом поведения заданных алгоритмов в стохастической среде, т. е. когда времена обслуживания и моменты поступления заявок являются случайными величинами, подчиняющимися заданным законам распределения вероятностей. Как будет видно из дальнейшего материала, наши усилия сосредоточены на синтезе или построении оптимальных или эвристических алгоритмов, анализ которых, в основном, ограничивается лишь определением их сложности и построением оценок для критериев эффективности. Модели, успешно изучаемые в теории массового обслуживания, отличаются существенно более простой структурой, что необходимо ввиду математических трудностей. Действительно, если бы не трудности анализа, то практически более полезным подходом к решению наших задач был бы подход с позиций вероятностных моделей. Используя эти модели, можно определить влияние случайных факторов на поведение задания или заявки в вычислительной или любой другой обслуживающей системе.

Читателю будет нетрудно представить себе множество обобщений и расширений материала данной книги. Подходы, развитые в гл. 5 и 6, открывают обширное поле для исследования. Например, важным расширением рассмотренного в книге круга задач явились бы задачи, в которых учитывались бы стоимости прерываний и стоимости подготовки работ. Только в очень редких случаях последовательности заданий, составляющих работы, являются независимыми, и тогда указанные факторы могут быть учтены путем простого увеличения времени выполнения. Так же как и Конвей, Максвелл и Миллер [28] в 1967 г., мы должны

ожидать, что успех этой книги будет зависеть от того, насколько она послужит стимулом для решения множества все еще не решенных проблем в теории расписаний.

1.6. Обозначения

В дальнейшем используются следующие обозначения:

$\mathcal{T}$ — множество заданий;

n — количество элементов в $\mathcal{T}$;

T_i — i-е задание $\mathcal{T}$; T, T' и т. п. также используются для обозначения заданий из $\mathcal{T}$;

$<$ — антирефлексивное отношение частичного порядка, заданное на $\mathcal{T}$. Типичный элемент $<$ записывается как (T_i, T_j) и означает, что T_i должно быть выполнено до того, как может начаться выполнение T_j. Как описано в § 1.2, $<$ часто представляется в виде ациклического ориентированного графа (орграфа);

P — набор процессоров или машин;

m — количество элементов в P;

P_i — i-й процессор в P;

τ_{ij} — время выполнения T_j на P_i;

τ_j — время выполнения T_j при идентичных P_i;

ω_j — стоимость пребывания T_j в системе.

Следующие символы используются только в гл. 1, 4 и 5:

$\mathcal{R}$ — множество видов ресурсов (отличных от процессоров);

s — число элементов в $\mathcal{R}$;

R_i — i-й тип ресурса в $\mathcal{R}$;

m_i — общий объем ресурса типа R_i;

$R_i(T_j)$ — объем ресурса типа R_i, требуемый в процессе выполнения T_j;

$r_i(t)$ — объем ресурса типа R_i, используемый при выполнении заданий в момент t;

ω — длина расписания или максимальное время прохождения (часто снабжается индексами для того, чтобы указать то или иное правило построения расписания).

Другие обозначения характеристик расписаний определяются в тексте (например, среднее время завершения). Читатель должен, однако, иметь в виду, что в гл. 3 и 4 используется ненормализованное среднее взвешенное время завершения (т. е. только взвешенная сумма времен завершения или $n\bar{\omega}$, с учетом определения $\bar{\omega}$).

Булевские операторы $+$ и $\neg$ используются в гл. 4 в обычном понимании как «дизъюнкция» и «отрицание» соответственно; между членами, стоящими вплотную, подразу-

мевается оператор конъюнкции. Символ τ используется в гл. 5 для обозначения «функции времени выполнения», т. е. $\tau(T_i)=\tau_i$. Символ $\triangleq$ используется в гл. 6 как эквивалент «равно по определению».

Система заданий представляется различными способами. Запись в виде пятерки объектов, как в § 1.2, непосредственно не используется, так как упрощенные варианты общей модели, рассматриваемые в гл. 2—6, задаются в более сокращенной записи. В этом плане, хотя слово «работа» иногда используется как синоним слова «задание», тем не менее в §§ 3.5—3.8 и в § 6.4 оно имеет специальное значение — как некоторая подсистема заданий. Значительная часть необходимых обозначений кратко определяется в начале соответствующих глав или непосредственно перед их использованием. Все обозначения, не представленные в этой главе, даются по мере их использования.

ГЛАВА 2

АЛГОРИТМЫ ПОСТРОЕНИЯ РАСПИСАНИЙ МИНИМАЛЬНОЙ ДЛИНЫ

2.1. Введение

Рассмотрим систему заданий $(\mathscr{T}, <)$, где $\mathscr{T}$ — множество заданий, которые нужно распределить между m процессорами, а $<$ — частичное упорядочение на $\mathscr{T}$, задающее ограничения на порядок выполнения заданий. Последнее означает, что если для заданий T и T' из $\mathscr{T}$ имеет место $T < T'$, то выполнение задания T должно быть завершено до того, как начнется выполнение задания T'. В общем случае времена выполнения заданий из $\mathscr{T}$ произвольны. Длиной некоторого расписания для $(\mathscr{T}, <)$ называется время, необходимое для выполнения всех заданий.

Нас интересуют эффективные алгоритмы нахождения расписаний минимальной длины. С нашей точки зрения, эффективным алгоритмом считается такой, который дает расписание минимальной длины для системы, состоящей из n заданий, за время, ограниченное величиной n^c, где c — константа. Такой алгоритм называется *полиномиальным*.

Расписание минимальной длины позволяет не только завершать выполнение всех заданий как можно раньше, но и обеспечивает максимальное использование ресурсов системы. Длина расписания зависит от общего времени выполнения всех заданий и общего времени незанятости процессоров. Минимизация длины расписания минимизирует время простоя, что в свою очередь приводит к максимальному использованию процессоров. Этот критерий эффективности применяется в тех случаях, когда порядок выполнения заданий не имеет значения. Если важна последовательность выполнения заданий, то более подходящим может быть критерий типа «среднее время обслуживания» (см. гл. 3).

Мы собираемся рассматривать расписания двух типов. В расписаниях *с прерываниями* допускается прерывание выполнения задания и последующее возобновление с точки прерывания. В расписаниях *без прерываний* выполнение задания, раз начавшись, происходит до завершения. На рис. 2.1 приведены примеры расписаний обоих типов. Здесь

требуется составить расписание выполнения системы из трех независимых заданий, каждое из которых имеет длину 2, на двух процессорах. Расписание без прерываний занимает 4 единицы времени. Минимальное расписание с прерываниями занимает 3 единицы; экономия составляет 25%.

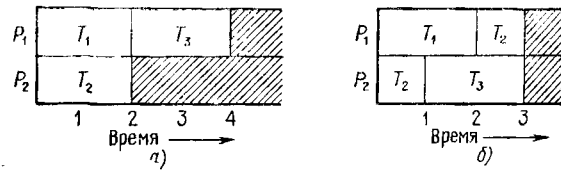


Рис. 2.1.

Платой за выигрыш является снятие задания с обслуживания (до его завершения) и сохранение информации о нем в течение всего времени ожидания. Если в состав оборудования вычислительной системы входит внешний накопитель типа быстродействующего барабана, то обмен информацией между основной памятью и накопителем осуществляется относительно эффективно. В этом случае стоимостью прерывания можно пренебречь. Даже если эта стоимость оказывается заметной, все равно, стремление быстрее завершить выполнение системы заданий может сделать критерий минимизации длины расписания превалирующим над другими соображениями.

Известно очень мало полиномиальных алгоритмов нахождения расписаний минимальной длины даже для тех случаев, когда наложены строгие ограничения на частичное упорядочение и времена выполнения заданий. Для начала мы будем считать, что все процессоры идентичны, следовательно, время выполнения задания T на всех процессорах одно и то же. В сущности, имеются два случая, для которых известны полиномиальные алгоритмы: (а) когда граф системы заданий является деревом и (б) когда в системе имеются только два процессора.

Если требуется построить расписание без прерывания, то в обоих случаях появляется дополнительное ограничение: все задания имеют одинаковое время выполнения.

Как показано в гл. 4, большинство проблем, связанных с построением расписаний, попадает в класс комбинаторных задач, для которых в течение длительного времени предпринимаются попытки найти решение, менее трудоемкое по сравнению с перебором. Задачи этого класса называют *NP*-

полными, полиномиально полными или просто полными. Интересное свойство данного класса состоит в том, что наличие полиномиального решения для одного представителя класса влечет за собой наличие такого решения для остальных задач данного класса. Размер этого класса и характер содержащихся в нем задач таковы, что мы пессимистически относимся к возможности найти полиномиальное решение для какой-либо полной задачи.

Некоторые результаты, показывающие, что задачи расписаний являются полными, содержатся в гл. 4. Часть из них имеет отношение к предмету нашего обсуждения и приведена ниже. В каждом из перечисленных случаев определение расписания минимальной длины без прерываний представляет собой полную задачу.

1. Времена выполнения заданий произвольные; имеется два или более процессоров. (Результат остается в силе, даже если все задания независимы.)

2. Времена выполнения заданий равны одной или двум единицам, частичное упорядочение является произвольным, имеется два или более процессоров.

Первый результат говорит о том, что для получения полиномиального алгоритма нельзя допускать произвольные значения времен выполнения. Если эти времена не произвольные, то их значения должны выбираться из некоторого набора допустимых величин. Второй результат приводит к необходимости рассмотрения систем с одинаковыми временами выполнения заданий, по крайней мере в случае произвольного частичного упорядочения.

При изучении систем заданий, в которых все задания имеют одинаковые времена выполнения, можно без потери общности считать, что все эти времена равны единице. В этом случае мы имеем так называемую UET-систему (unit-execution-time). В § 2.2 рассматривается алгоритм, имеющий почти линейную сложность, который для UET-системы $(\mathscr{T}, <)$, состоящей из n задач, строит расписание минимальной длины без прерываний при условии, что граф системы представляет собой дерево. Произвольное частичное упорядочение может быть представлено в виде ориентированного ациклического графа *). Параграф 2.3 посвящен орграфам. Для случая $m=2$ рассмотрен оптимальный (т. е. определяющий кратчайшее расписание без прерываний) алгоритм, имеющий сложность $O(n^2)$. Для $m=3, 4, 5, \ldots$ вопрос о

*) В дальнейшем для этого класса графов будем употреблять термин *орграф*. (*Прим. перев.*)

существовании оптимального полиномиального алгоритма остается открытым. Более детально этот вопрос рассмотрен в гл. 4.

В § 2.6 мы увидим, что алгоритмы без прерываний, описанные в §§ 2.2, 2.3, естественным образом приводят к алгоритмам с прерываниями. Рассмотрим систему заданий $(\mathscr{T}, <)$ с произвольными временами выполнения. Мы можем расщепить каждое задание на одинаковые части и получить цепочку заданий с равными временами выполнения. Благодаря этому получаем, с точностью до некоторых деталей, которые рассмотрены в § 2.6, UET-систему. Неудивительно поэтому, что единственными случаями, для которых известны полиномиальные алгоритмы нахождения расписаний минимальной длины с прерываниями, являются следующие: (а) граф $(\mathscr{T}, <)$ представляет собой дерево, или (б) имеются только два процессора. Частный случай системы с независимыми заданиями рассмотрен в § 2.5.

До сих пор мы ограничивались рассмотрением идентичных процессоров. В § 2.8 предполагается, что задан набор из m процессоров $P=\{P_1, P_2, \ldots, P_m\}$, имеющих быстродействия $b_1 \geqslant b_2 \geqslant \ldots \geqslant b_m$. Задание, требующее τ единиц времени, на процессоре P_j выполняется за τ/b_j единиц времени. Оказывается, что алгоритм с прерываниями, рассмотренный в § 2.6, может быть непосредственно распространен на набор процессоров P. Новый алгоритм является оптимальным для независимых заданий.

В заключение данной главы мы рассматриваем конвейерную задачу. В этом случае набор заданий состоит из цепочек длины m, в которых задания, стоящие на i-м месте, должны выполняться на процессоре P_i. Цепочки заданий можно интерпретировать следующим образом: первое задание соответствует процессу ввода, который должен быть осуществлен с устройства чтения с перфокарт, второе соответствует счету на процессоре, и третье соответствует выводу на печатающее устройство. Мы предлагаем алгоритм для двухпроцессорной конвейерной задачи. Для $m=3$, 4, ... определение расписания минимальной длины в конвейерной задаче является полной задачей.

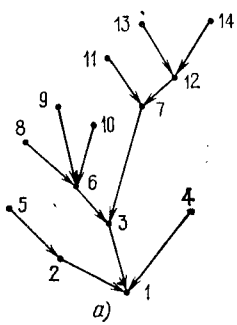
2.2. Системы заданий с древовидной структурой

В данном параграфе мы рассмотрим систему заданий $(\mathscr{T}, <)$, граф которой представляет собой лес. Каждое задание имеет единичное время выполнения. Имеется m идентичных процессоров, m произвольно. Опишем почти

линейный по сложности алгоритм определния расписания минимальной длины без прерываний.

Поскольку все задания имеют одинаковые времена, а прерывания запрещены, то удобно вместо непрерывного времени ввести дискретное. Выполнение расписания начинается в момент времени 0. Для всех целых i i-я единица времени (i-й интервал) представляет собой интервал $(i-1, i)$.

Рассмотрим дерево на рис. 2.2. На пути от вершины 14 к конечной вершине расположены вершины 14, 12, 7, 3 и 1.



а)

14	12	8	6	3	1
13	10	7	4		
11	9	5	2		

б)

Рис. 2.2.

Поскольку $14 < 12 < \ldots < 1$, то, независимо от количества процессоров, потребуется по меньшей мере 5 единиц времени, чтобы выполнить все задания. При наличии большого числа процессоров оптимальная стратегия будет заключаться в том, чтобы начать с наиболее удаленных заданий и продвигаться к конечной вершине. Как станет ясно, такая стратегия дает неплохие результаты при любом количестве процессоров.

Назовем уровнем вершины x в орграфе максимальное число вершин (включая x), находящихся на пути от x к конечной вершине. В лесу имеется ровно один такой путь. Конечное задание имеет уровень 1. Назовем задание готовым, если выполнены все его предшественники. Мы можем показать, что следующая стратегия приводит к оптимальному расписанию для леса при любом количестве процессоров.

Уровневая стратегия: *всякий раз, когда процессор освобождается, на него назначается невыполненное, но готовое задание из числа имеющих высший уровень (находящихся дальше всего от конечной вершины). Полу-*

чаемое таким образом расписание называется уровневым расписанием.

Вначале ограничим наше обсуждение рассмотрением деревьев; как будет показано, последующий переход к лесу является тривиальным.

Пример 1. Применение уровневой стратегии для трех процессов к системе заданий, изображенной на рис.2.2, позволяет построить соответствующее расписание. Вначале назначаются задания 14 и 13, имеющие уровень 5. Задание 12 уровня 4 должно ожидать окончания выполнения заданий 14 и 13, поэтому на третий процессор в первую единицу времени назначается задание 11. Задания 12, 10 и 9 уровня 4 выполняются в течение второй единицы времени. Читателю предлагается убедиться, что получаемое расписание является действительно уровневым.

Расписание, приведенное на рис. 2.2, иллюстрирует важное свойство расписаний минимальной длины без прерываний. Задание 1, будучи единственной конечной вершиной, должно ожидать окончания всех других заданий, а для того, чтобы на трех процессорах выполнить без прерываний 13 заданий, предшествующих заданию 1, требуется по меньшей мере 5 единиц времени. Таким образом, длина любых расписаний для данной системы заданий должна быть по крайней мере равна 6. Другими словами, в любом расписании для дерева в последнюю единицу времени может выполняться только одно задание — конечное. Расписание оказывается оптимальным, если в нем нет простоев во все интервалы времени, предшествующие предпоследнему.

Теорема 2.1. *Пусть граф* UET-*системы* $(\mathscr{T}, <)$ *является деревом; тогда уровневая стратегия обеспечивает построение расписания минимальной длины без прерываний.*

Доказательство. Доказательство теоремы строится по следующему плану.

1. Предположим, что уровневая стратегия не оптимальна. Тогда существует наименьшая (по количеству заданий) система $(\mathscr{T}, <)$, для которой уровневое расписание S имеет не минимальную длину.

2. Пусть ω есть длина S. Поскольку S не оптимально, в интервале $\omega-2$ должен быть свободный процессор.

3. Рассмотрим систему заданий $(\mathscr{T}', <')$, образованную из $(\mathscr{T}, <)$ путем исключения конечной вершины и всех ее непосредственных предшественников. Уровневое расписание S' для $(\mathscr{T}', <')$ может быть получено из S путем замены исключенных заданий на пустые интервалы.

4. Длина S' равна $\omega-2$.

5. Превратим лес $(\mathscr{T}', <')$ в дерево $(\mathscr{T}'', <'')$, добавив задание T_r, являющееся непосредственным преемником всех терминальных вершин $(\mathscr{T}', <')$. Длина уровневого расписания S'' для $(\mathscr{T}'', <'')$ равна $\omega-1$.

6. Поскольку система $(\mathscr{T}'', <'')$ была образована из $(\mathscr{T}, <)$ путем исключения двух уровней и добавления одного, система $(\mathscr{T}'', <'')$ должна иметь по крайней мере на одно задание меньше, чем $(\mathscr{T}, <)$. Но уровневое расписание S'' для $(\mathscr{T}'', <'')$ не может быть оптимальным. Таким образом, мы нашли систему, меньшую чем $(\mathscr{T}, <)$, для которой уровневая стратегия не является оптимальной, что противоречит первому предположению. Следовательно, теорема верна.

Перед тем как приступить к осуществлению данного плана, отметим, что при уровневой стратегии вершины на каждом уровне могут назначаться в различном порядке. Таким образом, расписание, полученное с помощью этой стратегии, не является единственным. Стратегия будет считаться неоптимальной, если найдется по крайней мере одно уровневое расписание, которое не будет оптимальным.

Пусть $(\mathscr{T}, <)$ — система с минимальным числом заданий из тех, для которых уровневое расписание превосходит по длине оптимальное. Обозначим через n число заданий в этой системе, через ω — длину ее уровневого расписания S, через ω_0 — длину ее оптимального расписания.

Поскольку число выполняемых заданий зафиксировано, длина расписания S для $(\mathscr{T}, <)$ зависит от количества незанятых периодов в S. В соответствии с обсуждением, предшествовавшим формулировке теоремы, на некотором процессоре должен быть пустой единичный интервал t, $t \leqslant \omega-2$.

Поскольку в интервале t имеется свободный процессор, из определения уровневой стратегии следует, что ни одно из заданий, выполняемых после t, не может быть готово в интервале t. Поэтому все задания, выполняемые после t, должны иметь предшественника, выполняемого на некотором процессоре в течение t. Поскольку граф $(\mathscr{T}, <)$ есть дерево, каждое задание имеет, самое большее, одного непосредственного преемника; таким образом, количество заданий, выполняемых в интервале $t+1$, не может превысить числа заданий, выполняемых в интервале t. Из этого следует, что после t всегда имеется свободный процессор, в частности, должен быть свободный процессор и в интервале $\omega-2$.

Конечное задание T из $(\mathscr{T}, <)$ выполняется в интервале ω. Поэтому непосредственный предшественник R вер-

шины T должен выполняться на некотором процессоре в единицу времени $\omega-1$. Поскольку на интервале $\omega-2$ имеется свободный процессор, то предшественник Q задания R должен выполняться в единицу времени $\omega-2$.

Рассмотрим систему заданий $(\mathscr{T}', <')$, образованную путем исключения конечной вершины и всех ее непосредственных предшественников. Мы утверждаем, что расписание S', образованное путем замены исключенных заданий пустыми интервалами, является уровневым расписанием для $(\mathscr{T}', <')$.

Конечное задание T выполняется на последнем интервале; следовательно, оно является преемником всех заданий и замена его пустым интервалом не может изменить уровневую структуру остальной системы. Теперь рассмотрим непосредственного предшественника U задания T, который выполняется в течение некоторой единицы времени t. Единственными заданиями, которые могли бы выполняться в течение t после исключения U, являются задания того же уровня либо более низкого. Если имеются задания более высокого уровня, которые выполняются после t, они должны были быть рассмотрены перед U и признаны не готовыми к выполнению. Поскольку все задания того же или более низкого уровня, чем U, исключены, ни одно задание не может заполнить пустой интервал, образовавшийся после исключения U.

Задание Q уровня 3 выполняется в интервале $\omega-2$. Таким образом, не существует интервал, предшествующий $\omega-2$, в котором выполняются только задания уровня 2. Следовательно, длина S' равна $\omega-2$. Поскольку уровни всех заданий в $(\mathscr{T}' <')$ на 2 ниже, чем уровни соответствующих значений в $(\mathscr{T}, <)$, S' является уровневым расписанием для $(\mathscr{T}', <')$.

Поскольку в формулировке теоремы речь идет о деревьях, превратим $(\mathscr{T}', <')$ в дерево путем добавления нового задания T_r, являющегося непосредственным преемником всех терминальных вершин системы $(\mathscr{T}', <')$. Обозначим новую систему заданий $(\mathscr{T}'', <'')$. Уровневое расписание S'' для $(\mathscr{T}'', <'')$ может быть образовано из S' путем назначения задания T_r на некоторый процессор в интервале $\omega-1$. Заметим, что T_r не может выполняться до интервала $\omega-1$, потому что Q, предшествующее T_r, выполняется в $\omega-2$. Таким образом, длина расписания S'' равна $\omega-1$. Кроме того, $(\mathscr{T}'', <'')$ содержит не более $n-1$ задания.

В оптимальном расписании для $(\mathscr{T}, <)$ конечное задание может выполняться только на интервале ω_0, а на интер-

вале ω_0—1 могут выполняться только его непосредственные предшественники. Следовательно, исключение конечного задания и его непосредственных предшественников приводит к минимальному расписанию, длина которого не больше чем ω_0—2. Добавление нового задания T_r, как это сделано ранее, приводит к системе заданий, для которой минимальная длина расписания не превышает ω_0—1.

Из $\omega_0 < \omega$ получаем $\omega_0 - 1 < \omega - 1$, и, следовательно, мы нашли меньшую систему заданий, чем $(\mathscr{T}, <)$, для которой уровневая стратегия не является оптимальной,— противоречие. Теорема доказана.

Следствие 2.1. *Пусть граф* UET-*системы является лесом; тогда уровневая стратегия строит расписание минимальной длины без прерываний.*

Доказательство. Если уровневая стратегия не строит расписание минимальной длины для $(\mathscr{T}, <)$, она не может построить расписание минимальной длины для дерева $(\mathscr{T}', <')$, образованного путем добавления нового задания, являющегося непосредственным преемником терминальных заданий из $(\mathscr{T}, <)$. К системе $(\mathscr{T}', <')$ применима теорема 2.1.

Мы показали, что уровневая стратегия приводит к оптимальному результату, а теперь должны разработать алгоритм составления расписания. Алгоритм, который мы рассмотрим, приписывает приоритеты всем заданиям. Задания более высоких уровней имеют более высокий приоритет. В дальнейшем наше обсуждение относится к лесам.

Поскольку уровни отсчитываются от конечных вершин, то, начиная с конечных вершин, последовательно просматриваются все вершины на одном уровне, затем на следующем и так далее, при этом каждой вершине приписывается *метка*. Для системы из n заданий метка вершины представляет собой целое число от 1 до n.На рис. 2.2 конечная вершина помечена меткой 1, вершины уровня 2 имеют метки 2, 3 и 4 и так далее до 13 и 14, которыми помечены два задания уровня 5.

После того, как процесс расстановки меток завершен, составляется список заданий в порядке убывания меток. Как и в гл. 1, этот список используется для составления расписаний следующим образом.

Составление списочного расписания: *всякий раз, когда процессор освобождается, список просматривается слева направо и первое еще не выполненное готовое задание назначается на процессор.*

Расписание, показанное на рис. 2.2, *б*, представляет собой списочное расписание для системы заданий, изображенной на рис. 2.2, *а*, в котором список упорядочен по убыванию меток. Отметим, что на первом интервале задание 12 не готово к выполнению, так как до него должны быть выполнены задания 14 и 13. Поэтому задание 11, следующее по списку после задания 12, назначается на третий процессор. Аналогично, на третьем интервале задание 6 не готово и на третий процессор назначается задание 5.

Алгоритм 2.1. Пусть граф системы $(\mathscr{T}, <)$ является лесом. Вначале каждому заданию из $\mathscr{T}$ приписывается метка. Имеется функция *) $L_t(\cdot)$, которая ставит в соответствие заданиям числа из набора $\{1, 2, \ldots, n\}$. Метки затем используются для построения списка для $(\mathscr{T}, <)$. Этот список приводит к расписанию минимальной длины без прерываний для системы $(\mathscr{T}, <)$ на m процессорах.

Временно будем считать, что $(\mathscr{T}, <)$ имеет одну конечную вершину.

1. Если T есть конечная вершина $(\mathscr{T}, <)$, пометим T меткой 1 (т. е. $L_t(T)=1$).

2. Пусть метки $1, 2, \ldots, j-1$ уже расставлены. Если T есть задание, непосредственный преемник которого имеет минимальную метку, положим $L_t(T)=j$. (Отметим, что процесс расстановки меток переходит от уровня к уровню, приписывая последовательные значения меток заданиям одного уровня.)

3. Используем список $(T_n, T_{n-1}, \ldots, T_1)$, в котором $L_t(T_i)=i$, $1 \leqslant i \leqslant n$, для построения расписания.

Расписание для леса $(\mathscr{T}, <)$ может быть получено путем добавления фиктивного задания, являющегося непосредственным преемником всех конечных вершин $(\mathscr{T}, <)$, и последующего применения пунктов 1—3 к новой системе заданий.

Расстановка меток, показанная на рис. 2.2, *а*, выполнена с использованием алгоритма 2.1.

Теорема 2.2. *Пусть граф* UET-*системы* $(\mathscr{T}, <)$ *представляет собой лес; тогда алгоритм* 2.1 *строит для* $(\mathscr{T}, <)$ *расписание минимальной длины без прерываний.*

Доказательство. Из следствия 2.1 вытекает, что нужно только показать, что алгоритм 2.1 строит уровневое расписание для $(\mathscr{T}, <)$. Из описания шага 3 видно,

*) Мы использовали нижний индекс t, чтобы отличить функцию расстановки меток для деревьев от функций, выполняющих аналогичную роль в других разделах.

что алгоритм расставляет метки от уровня к уровню. Читателю предоставляется самому завершить доказательство.

Один из путей может заключаться в том, чтобы показать с помощью индукции по j, что когда метка j приписывается заданию T, то на уровнях, меньших уровня T, нет ни одного непомеченного задания.

В этом параграфе мы показали, что при оптимальной стратегии составления расписаний для UET-лесов в первую очередь назначаются на выполнение задания наиболее высоких уровней. Рассмотренный алгоритм помечает задания от уровня к уровню так, что, чем выше уровень задания, тем большая метка ему приписывается. Структура данных для алгоритма обсуждается в § 2.4. Там же показано, что с помощью функции $L_l(\cdot)$ процесс составления оптимального расписания может быть завершен за почти линейное время.

В рассматриваемых совокупностях деревьев (лесах) каждое задание может иметь несколько непосредственных предшественников, но непосредственный преемник задания всегда единственный. Представляет также интерес противоположная ориентация ограничений предшествования, а именно: завершение некоторого задания может позволить начать выполнение более чем одного задания, но непосредственный предшественник задания всегда единственный. Такая система заданий называется *антилесом* *). Неформально, антилес может быть построен, если у леса изменить на обратную ориентацию всех ребер. Результаты настоящего параграфа также применимы к антилесам. Для этого просто игнорируется направление ребер и строится уровневое расписание. Для антилеса расписание следует читать наоборот, т. е. справа налево. Утверждения такого рода можно сделать для многих алгоритмов, рассматриваемых в данной главе (см. также § 2.7).

2.3. Двухпроцессорные расписания для систем заданий с произвольным частичным упорядочением

В настоящее время не известны полиномиальные алгоритмы составления расписаний для UET-систем при фиксированном количестве процессоров m, если $m>2$. Для $m=2$ мы в этом параграфе рассмотрим алгоритм, имеющий сложность $O(n^2)$. Здесь применяется подход, аналогичный

*) Граф такой системы называют также *выходящим деревом*, в противоположность термину *входящее дерево*, употребляемому для обозначения графа, рассматривавшегося выше. (*Прим. перев.*)

тому, который был использован для лесов, а именно: каждому заданию приписываются метки, определяющие их приоритеты, на основе которых строится список, используемый для составления расписания. Для того чтобы алгоритм мог работать, граф UET-системы не должен иметь транзитивных ребер.

В предыдущем параграфе было установлено, что стратегия «первым идет самый высокий уровень» является оптимальной для лесов. Поскольку деревья являются частным случаем орграфов, следует ожидать, что и для произвольных орграфов уровни играют определенную роль при построении расписаний. Было бы хорошо, если бы мы смогли снова расставить приоритеты по уровням, после чего пришлось бы только заботиться о порядке, в котором должны выполняться задания одного уровня. Поэтому сформулируем желательное свойство процесса расстановки меток $L(\cdot)$ для орграфов.

Свойство 1. *Если T имеет более высокий уровень, чем T', то $L(T)>L(T')$.*

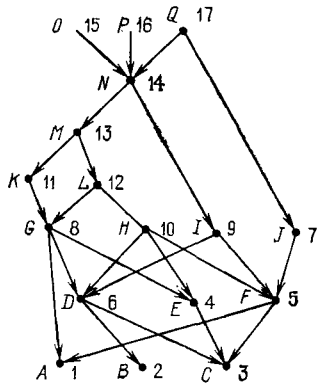
Помня о том, что мы собираемся строить расписание для двух процессоров, заметим, что если на каждом уровне имеется четное число заданий, то все задания одного уровня будут завершаться до начала заданий следующего уровня, и, таким образом, свойство 1 является достаточным для того, чтобы расписание было оптимальным.

Для того чтобы выявить ряд факторов, существенных для оптимальной расстановки меток, рассмотрим систему заданий на рис. 2.3. На самый высокий уровень (7) мы поместили три задания: O, P и Q. Таким образом, во вторую единицу времени нам придется искать задание, готовое к выполнению на более низком уровне. Поскольку Q имеет, кроме N, непосредственного преемника J, приоритет Q должен быть выше, чем у O и P. Мы приходим к следующему свойству.

Свойство 2. *Если множество непосредственных преемников задания T содержит в себе множество непосредственных преемников T', то $L(T)>L(T')$.*

Свойство 2 не является достаточным для оптимальности расписания, однако продолжим рассуждения. Поскольку в системе имеется 17 заданий, метка 17 приписывается заданию Q, а метки 15 и 16 — заданиям O и P соответственно. Единственным заданием уровня 6 является N, и оно получает метку 14. Мы поместили на уровень 6 только одно задание, чтобы продемонстрировать тот факт, что и в оптимальном расписании могут встречаться простои процессора

Свойств 1 и 2 оказывается достаточно до тех пор, пока мы не достигнем уровня 3, на котором нужно поместить задания G, H, I и J. Поскольку $G \prec A$ и $H \prec F$, задания G и H неразличимы по свойству 2. Отметим, что A имеет уровень 1, а F имеет уровень 2. Поскольку F (как имеющее



а)

17	15	14	13	12	10	6	4	3	
16	7		9	11	8	5	2	1	

б)

Рис. 2.3.

более высокий уровень, чем A), возможно, будет выполняться раньше, чем A, присвоим H большую метку, чем G.

Выясняются два взаимосвязанных обстоятельства: (а) нам необходим механизм для сравнения преемников заданий, и (б) при сравнении необходимо принимать в расчет приоритеты преемников. Поскольку приоритеты задаются метками, нам, похоже, удастся использовать метки при сравнении. Если мы собираемся использовать метки, то нужно вначале помечать преемников, а затем предшественников. Оказывается, существует оптимальный алгоритм поуровневой расстановки меток.

Алгоритм расстановки меток, который мы используем, начинается с приписывания метки 1 одному из конечных заданий. Предположим, что метки 1, 2, . . ., j—1 уже расставлены. Поскольку метки определяют приоритеты заданий, метка j приписывается такому заданию, которое имеет наинизший приоритет из множества заданий, готовых

к помечиванию. Для того чтобы выявить такое задание, сравним множества непосредственных преемников заданий, готовых к помечиванию. Для каждого задания построим убывающую последовательность, используя метки его непосредственных преемников. Наименьшая (лексикографически) последовательность определяет задание, которое помечается меткой j.

Более строго, пусть $l=(n_1, n_2, \ldots, n_k)$ и $l'=(n'_1, n'_2, \ldots, n'_{k'})$—две последовательности целых чисел, $k \geqslant 0$, $k' \geqslant 0$. Когда $k=0$, последовательность l пустая. Будем говорить, что l *лексикографически меньше*, чем l' (записывается $l \leqslant l'$), если: (1) обе последовательности совпадают до j-й позиции (для некоторого j), однако $n_j < n'_j$ (т. е. существует такое j, $1 \leqslant j \leqslant k$, что для всех i, $1 \leqslant i < j$, $n_i = n'_i$ и $n_j < n'_j$) или (2) две последовательности совпадают, но l более короткая (т. е. $k < k'$ и $n_i = n'_i$ для всех i, $1 \leqslant i \leqslant k$).

Следующий алгоритм определяет процесс расстановки меток, описанный выше неформально.

Алгоритм 2.2. Пусть $(\mathscr{T}, <)$—система из n заданий с единичными временами выполнения. Функция $L(\cdot)$ ставит в соответствие каждому заданию из $\mathscr{T}$ некоторую метку из набора $\{1, 2, \ldots, n\}$. Метки затем используются для построения списка, с помощью которого формируется расписание минимальной длины без прерываний на двух процессорах для $(\mathscr{T}, <)$.

1. Припишем метку 1 одному из конечных заданий.

2. Пусть метки 1, 2,. . ., j—1 уже ра,сставлены. Пусть S есть набор непомеченных заданий, не имеющих непомеченных преемников. Выберем элемент S который должен быть помечен меткой j.

Для каждой вершины x из S определим $l(x)$ следующим образом. Пусть $y_1, y_2, \ldots, y_k$ — непосредственные преемники x. Тогда $l(x)$ представляет собой убывающую последовательность целых чисел, которая образуется путем упорядочения набора $\{L(y_1), L(y_2), \ldots, L(y_k)\}$. Пусть x есть элемент S такой, что для всех x' из S выполняется $l(x) \leqslant l(x')$. Тогда положим $L(x)=j$.

3. Когда все задания помечены, полученный список $(T_n, T_{n-1}, \ldots, T_1)$, в котором для всех i $(1 \leqslant i \leqslant n)$ $L(T_i)=i$, определяет расписание для системы заданий.

Поскольку каждое задание выполняется за единичное время, процессоры 1 и 2 оказываются свободными одновременно. Будем считать, что вначале занимается процессор 1, а затем 2.

Для доказательства оптимальности алгоритма 2.2 для двух процессоров необходимо установить взаимосвязь между системой заданий и соответствующим расписанием. Основная идея доказательства заключается в следующем.

Определим множества F_k, F_{k-1}, . . ., F_0, $F_i \subseteq \mathscr{T}$, $k \geqslant i \geqslant 0$, *такие, что все задания* F_i *предшествуют всем заданиям* F_{i-1}. *Все задания в* F_i *должны, таким образом, быть завершены до того, как может начаться выполнение любого задания из* F_{i-1}. *Множества* F_i *обладают тем свойством, что любое расписание, при котором они выполняются последовательно, имеет по крайней мере такую же длину, как и расписание, порожденное алгоритмом* 2.2 (см. рис. 2.4).

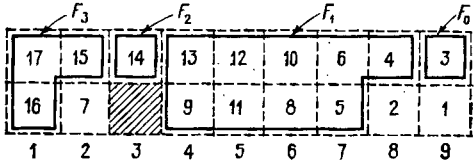


Рис. 2.4.

Мы хотим дать как можно более наглядное определение множества F_i. Рассмотрим задание 14, выполняющееся в третью единицу времени в расписании, показанном на рис. 2.3, *б*. Поскольку второй процессор в это время не занят, то задание 14 должно быть предшественником всех других заданий, выполняемых после него. Таким образом, как следует из определения списочного расписания, все остальные задания в третью единицу времени были просмотрены и среди них не нашлось готовых к выполнению.

Однако мы не можем разделить задания на два множества S_1 и S_2 таких, чтобы S_1 содержало все задания, выполняющиеся не позже задания 14, а S_2 содержало все остальные задания, надеясь, что все задания из S_1 предшествуют всем заданиям из S_2. Задание 7, выполняемое во вторую единицу времени, не предшествует никакому заданию с уровнем выше чем 7. Рассмотрим задания, которые выполняются не по порядку, и покажем, что, если не принимать в расчет такие задания, как 7, можно определить множества F_k, F_{k-1}, . . ., F_0.

Для заданного расписания будем обозначать через $\lambda(T)$ единицу времени, в течение которой выполняется задание T.

Соглашение о том, что вначале занимается процессор 1, а затем процессор 2, приводит к полезному свойству расписаний.

Лемма 2.1. *Пусть S есть списочное расписание, построенное для системы заданий $(\mathscr{T}, \prec)$ при помощи процесса расстановки меток в соответствии с алгоритмом 2.2. Если задание T выполняется на процессоре 1, то для всех $T \in \mathscr{T}$ можно утверждать, что если T' выполняется не раньше, чем T, то метка T' меньше метки T. Иными словами, условие $\lambda(T) \leqslant \lambda(T')$ влечет $L(T) > L(T')$.*

Доказательство. Поскольку задание T назначено на процессор 1 в течение $\lambda(T)$, задание T имеет наибольшую метку из числа заданий, готовых к выполнению на этот момент времени. Все задания, не готовые к выполнению, должны ожидать выполнения какого-либо из готовых. Из процесса расстановки пометок L следует, что для всех U, U' таких, что $U \prec U'$, должно быть справедливо $L(U) > L(U')$. Лемма доказана.

Для того чтобы показать, что алгоритм 2.2 является оптимальным, мы исследуем расписания, порождаемые им, аналогично тому, как это делается в обсуждении, предшествующем лемме 2.1.

Пусть S—списочное расписание для системы $(\mathscr{T}, \prec)$, построенное методом расстановки пометок в соответствии с алгоритмом 2.2. Определим *последовательные множества* F_k, F_{k-1}, ..., F_0, где k зависит от расписания, такие, что каждое задание из F_i является предшественником всех заданий из F_{i-1}. Последовательные множества зависят от заданий D_k, D_{k-1}, ..., D_0 и J_k, J_{k-1}, ..., J_0, где D_i «замыкает» группу заданий из последовательного множества F_{i-1}, а J_i «возглавляет» эту группу. Как D_i, так и J_i выполняются на одном и том же единичном интервале. Рис. 2.5 иллюстрирует следующие определения.

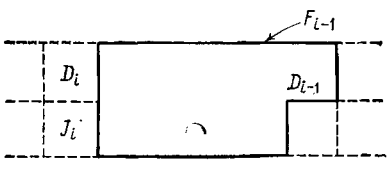


Рис. 2.5.

1. Пусть задания T и T' (T' может быть и периодом простоя) выполняются на последнем единичном интервале в S на процессорах 1 и 2 соответственно. Заметим, что задание T имеет большую метку, чем T'. Тогда D_0—это задание T, а J_0—это задание T'.

2. Предположим, что D_{k-1}, D_{k-2}, ..., D_0 и J_{k-1}, J_{k-2}, ..., J_0 уже определены. Определим J_k как (возможно пустое) задание T такое, что:

а) $L(T) < L(D_{k-1})$, где пустые задания считаются имеющими метку 0;

б) $\lambda(T) < \lambda(D_{k-1})$ и $\lambda(T)$—максимальное из всех λ удовлетворяющих этому неравенству.

Неформально, T «возглавляет» D_{k-1} и является ближайшим к D_{k-1} заданием, обладающим этим свойством. Теперь D_k—попросту задание, выполняемое на процессоре 2 на том же единичном интервале, что и J_k; $\lambda(D_k) = \lambda(J_k)$. Из леммы 2.1 следует, что задания $D_k, D_{k-1}, \ldots, D_0$ выполняются на процессоре 1, а задания $J_k, J_{k-1}, \ldots, J_0$— на процессоре 2.

3. Пусть $D_k, D_{k-1}, \ldots, D_0$ уже определены, а D_{k+1} не существует. Для всех i, $k > i \geqslant 0$, F_i есть множество всех заданий, выполняемых после D_{i+1}, но перед D_i, а также само D_i. Следовательно, мы имеем

$$F_i = \{T \mid \lambda(D_{i+1}) < \lambda(T) < \lambda(D_i)\} \cup \{D_i\}.$$

Поскольку D_{k+1} не существует, F_k определяется отдельно:

$$F_k = \{T \mid \lambda(T) < \lambda(D_k)\} \cup \{D_k\}.$$

Следующие две леммы показывают, что для всех i, $k \geqslant i \geqslant 1$, каждое задание из F_i является предшественником любого задания из F_{i-1}.

Лемма 2.2. *Для любого* i, $k \geqslant i \geqslant 1$, $D_i < T$ *для всех* T *из* F_{i-1}.

Доказательство. Как и на рис. 2.5, задания D_i и J_i выполняются на одном и том же интервале. Напомним, что задание, обозначенное J_i, получило это «звание» потому, что $L(J_i) < L(D_{i-1})$, и J_i является ближайшим к D_{i-1} заданием с таким свойством. Из этого следует, что для всех других заданий T из F_{i-1} имеем $L(T) \geqslant L(D_{i-1})$. Поскольку $L(J_i) < L(D_{i-1})$, то из транзитивности получаем, что $L(J_i) < L(T)$. Из определения списочного расписания следует, что, поскольку каждое задание T из F_{i-1} имеет большую метку, чем J_i, задание T рассматривалось перед тем, как назначать на процессор J_i. Следовательно, задание не могло быть готовым к выполнению в этом единичном интервале. Поэтому D_i должно быть предшественником T. Следует заметить, что D_i не обязано быть непосредственным предшественником T.

Доказательство следующей леммы основывается на изучении причины, по которой одно задание получает большую метку, чем другое. Рассмотрим задания 5 и 4 на рис. 2.4. Из леммы 2.2 мы знаем, что задание 4 предшествует заданию. 3. Следовательно, при составлении списка непосредственных преемников в соответствии с шагом 2 алгоритма

2.2 первый элемент в $l(4)$ должен был быть 3. Поскольку задание 5 имеет большую метку, чем 4, первым элементом в $l(5)$ должно быть, по крайней мере, тоже 3. Вообще говоря, есть две возможности — 3 и 4,— однако в обоих случаях задание 5 является предшественником задания 3.

Лемма 2.3. *Для всех* i, $k \geqslant i \geqslant 1$, *если* T *принадлежит* F_i, *а* T' *принадлежит* F_{i-1}, *то* $T < T'$.

Доказательство. Предположим, что лемма неверна. Тогда для некоторого i, $k \geqslant i \geqslant 1$, найдется задание T из F_i такое, что оно не предшествует заданию T' из F_{i-1}. Допустим временно, что T не имеет преемников в F_i.

Из леммы 2.2 следует, что D_i предшествует T'. Из определения F_i должно следовать, что T имеет большую метку, чем D_i, ибо в противном случае T было бы кандидатом на место J_{i+1} и не смогло бы входить в F_i. Из правила расстановки меток вытекает, что $l(D_i) \leqslant l(T)$.

Пусть $l(D_i) = (n_1, n_2, \ldots, n_r)$. Докажем, что несколько первых элементов $l(D_i)$ являются метками заданий из F_{i-1}. Из леммы 2.1 следует, что D_{i-1} имеет метку большую, чем все задания, выполняемые после D_{i-1}. По определению все другие задания в F_{i-1} имеют метки большие, чем D_{i-1}. Таким образом, элементы последовательного множества F_{i-1} имеют наибольшие метки из всех заданий, выполняемых после $\lambda(D_i)$. Теперь ясно, что наибольшие элементы $l(D_i)$ должны набираться из заданий множества F_{i-1}. В частности, если I есть максимальный набор заданий в F_{i-1}, не имеющих предшественников

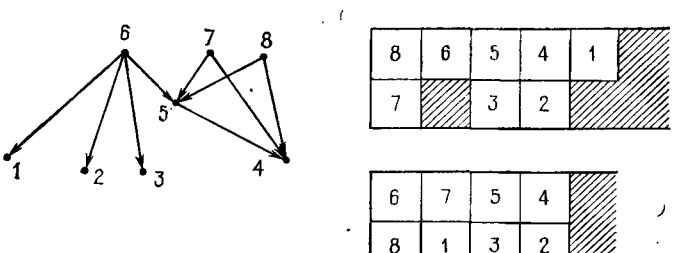


Рис. 2.6.

в F_{i-1}, то первые $|I|$ элементов множества $l(D_i)$ должны быть метками заданий, входящих в I. Последнее утверждение следует из введенного в гл. 1 предположения, что граф системы заданий $(\mathscr{T}, <)$ не имеет транзитивных ребер. Как показано на рис. 2.6, в случае, когда

граф содержит транзитивные ребра, алгоритм 2.2 не является оптимальным.

Мы выбрали T таким образом, чтобы оно не имело преемников в F_i. Для завершения доказательства заметим, что для того, чтобы $l(D_i)$ было лексикографически меньше, чем $l(T)$, T должно быть непосредственным предшественником каждого элемента из I. Таким образом, T должно быть предшественником всех элементов множества F_{i-1}, что является противоречием.

Наконец, последний случай, когда T должно иметь преемника в F_i, следует из транзитивности, так как T будет предшествовать всем заданиям, которым предшествует его преемник. Рано или поздно мы найдем такого преемника T в F_i, который сам не имеет преемников в F_i. Лемма доказана.

Теорема 2.3. *Пусть* $(\mathscr{T}, <)$ — UET-*система; тогда алгоритм* 2.2 *генерирует расписание минимальной длины без прерываний для* $(\mathscr{T}, <)$ *на двух процессорах.*

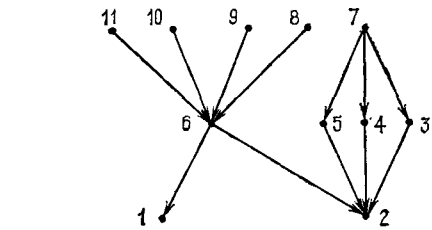
Доказательство. Пусть S — списочное расписание, построенное алгоритмом 2.2 для $(\mathscr{T}, <)$. Пусть в S имеется $k+1$ последовательных множеств $F_k, F_{k-1}, \ldots, F_0$. Из леммы 2.3 следует, что каждое задание в F_k является предшественником всех заданий в F_j для всех j, $k>j$. Таким образом, все задания из F_k должны быть завершены перед тем, как могут начать выполняться задания из F_j. Поскольку каждое последовательное множество имеет нечетное количество заданий, обозначим количество заданий в F_i через $2n_i-1$. Ясно, что для выполнения всех заданий, входящих в F_i, потребуется n_i единиц времени. Следовательно, любое расписание для $(\mathscr{T}, <)$ будет иметь длину, по крайней мере, в $\sum_{i=0}^{k} n_i$ единиц времени. Таким образом, если расписание S имеет длину $\sum_{i=0}^{k} n_i$, то оно должно быть оптимальным.

Леса являются частным случаем орграфов. Читатель может убедиться, что функция расстановки меток L назначает метки от уровня к уровню. Таким образом, возможно, хотя и нецелесообразно, применять алгоритм 2.2 для построения оптимальных расписаний для лесов на m процессорах.

Следует еще раз подчеркнуть, что алгоритм 2.2 позволяет построить оптимальное расписание только в том случае, когда граф, описывающий систему заданий, не содержит транзитивных ребер. Тем, кто в этом сомневается, рекомен-

дуем взглянуть на рис. 2.6. Однако, как нетрудно убедиться, транзитивные ребра могут быть удалены со сложностью $O(n^{2,8})$ в наихудшем случае [2].

Процесс расстановки пометок L приводит к оптимальному расписанию для двух процессоров. Естественно задать вопрос, будет ли он хорош или плох для большего



а)

11	8	6	3	2
10	7	5	1	
9		4		

б)

7	5	4	2
11	9	3	1
10	8	6	

в)

Рис. 2.7.

числа процессоров. Как показано на рис. 2.7, в случае трех процессоров расписание, составленное путем расстановки меток, имеет длину 5, а оптимальное расписание имеет длину 4. В самом деле, если на рис. 2.7 удалить задания 1 и 2 и для такой системы расставить метки, которые будут «неправильными», то списочное расписание, полученное для этого случая, будет иметь длину 4, а оптимальное расписание будет иметь длину 3.

Для случая шести процессоров рассмотрим систему заданий, изображенную на рис. 2.8. Пусть функция L расставляет метки в следующем порядке: A_{18}, B_1,. . ., B_{18}, C_1,. . ., C_{18}, D_1. В списочном расписании, основанном на таких метках, потребуется один единичный интервал для каждого уровня, содержащего два задания, и два единичных интервала для каждого уровня, содержащего $m+2$ задания, где $m=6$. Следовательно, длина расписания ω равна 11. Оптимальное расписание требует $\omega_0=8$ единичных интервалов. На каждом процессоре наблюдается такая же картина, следовательно, ω/ω_0 равно $(5k+1)/(3k+2)$ и стремится к

5/3. Для m процессоров подобная конструкция приводит к отношению $[(m-1)k+1]/(mk/2+2)$, которое при увеличении k стремится к $2-2/m$.

Из рис. 2.8 становится очевидным, что процесс расстановки меток при большом числе процессоров может давать совершенно неудовлетворительные результаты. Почему же он является оптимальным для двух процессоров? На интуитивном уровне это можно объяснить следующим образом.

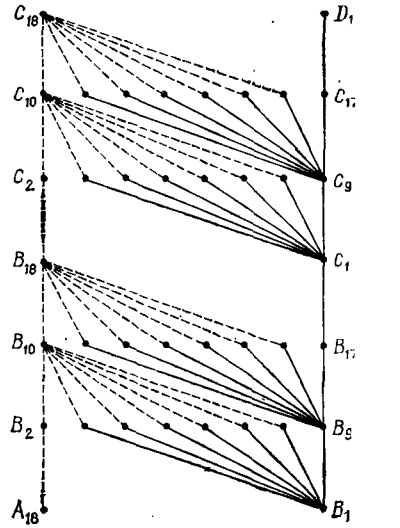


Рис. 2.8. Путем повторения картины можно построить систему заданий, для которой отношение длины расписания, полученного с помощью функции расстановки меток L, к длине оптимального расписания достигает $2-2/6=5/3$. Для произвольного m аналогичная конструкция приводит к системе заданий, для которой это отношение достигает $2-2/m$. Некоторые ребра для ясности даны штриховыми линиями. Все ребра направлены вниз.

Рассмотрим последовательные множества, построенные для двухпроцессорного варианта. Для m процессоров мы можем определить аналоги последовательных множеств, отыскивая единичные интервалы, в течение которых $m-1$ процессоров либо свободны, либо выполняют «возглавляющие» задания. Для этих новых последовательных множеств мы можем доказать лемму, аналогичную лемме 2.3, показав, что все задания из одного последовательного множества должны быть завершены перед тем, как может начаться выполнение заданий следующего последовательного множества. Поскольку последовательные множества должны выполняться друг за другом, эффективность процесса расстановки меток зависит от того, как спланировать выполнение заданий каждого из последовательных множеств. Из определения последовательного множества следует, что для любого единичного интервала, в течение которого выполняются задания этого множества, по крайней мере два процессора заняты выполнением заданий этого множества. Если процессоров всего два, получаем оптимальное расписание.

В [106] показано, что если ω — длина расписания, основанного на расстановке меток с использованием функции L, и ω_0 — длина оптимального расписания, то $\omega/\omega_0 \leqslant 2 - 2/m$. Как показывает рис. 2.8, оценка $2-2/m$ неулучшаема.

2.4. Реализация алгоритмов без прерываний

Временная сложность алгоритмов 2.1 и 2.2 зависит от структуры данных, используемой для представления графов, и от механизма расстановки меток и построения списочного расписания. Матричное представление содержит $O(n^2)$ элементов, следовательно, требуется $O(n^2)$ времени, чтобы выяснить, какие элементы матрицы являются ненулевыми. Поэтому представление леса в виде матрицы нецелесообразно, так как он содержит $O(n)$ ребер.

В этом параграфе мы вначале рассмотрим линейный алгоритм расстановки меток в дереве. При распространении алгоритма на произвольные орграфы его временная сложность возрастает до $O(n^2)$. В худшем случае орграф может содержать $O(n^2)$ ребер, но часто число ребер оказывается относительно небольшим. Алгоритм расстановки меток, время работы которого линейно зависит от числа ребер, можно найти в [130]. Чуть позже мы приведем почти линейный алгоритм построения расписания по заданным меткам.

Для алгоритма 2.1 удобно представлять данные в виде списка непосредственных предшественников каждой вершины. В дополнение к нему нужно иметь очередь Q заданий, готовых к помечиванию. Алгоритм состоит из следующих шагов.

1. Начиная с пустой очереди Q, поместим в эту очередь все конечные вершины леса. В результате в очереди будут все задания уровня 1. Ни одно задание еще не помечено.

2. Предположим, что метки $1, 2, \ldots, j-1$ уже расставлены. Возьмем из очереди задание T и дадим ему метку j. Поместим всех непосредственных предшественников T в конец очереди Q. Останов, если очередь пуста.

Легко показать, что с помощью данной процедуры расстановка меток в лесе происходит от уровня к уровню и в соответствии с алгоритмом 2.1. Кроме того, выполнение шагов займет линейное от числа вершин время.

Для орграфов мы также можем использовать список непосредственных предшественников и очередь заданий, готовых к помечиванию. Различие заключается в том, что

помечивание какого-либо задания не делает его непосредственных предшественников сразу готовными к помечиванию. В примере, показанном на рис. 2.9, задание A получает метку 1 в тот момент, когда F и G все еще имеют непомеченных преемников.

Рассматривая реализацию алгоритма 2.2, отметим, что в этом случае можно обойтись без составления списка меток в убывающем порядке. Мы проиллюстрируем этот момент, рассматривая процесс расстановки меток для системы заданий на рис. 2.9. Предположим, что метки от 1 до 4 уже расставлены. Поскольку вершины D и F еще не помечены, о списках для вершин G и H на этом этапе можно сказать лишь то, что $l(G)$ окончится метками 4,1, а $l(H)$ окончится меткой 4. Причина этого в том, что все метки, расставленные позже, будут больше 4. Вместо того, чтобы прослеживать изменение списков для G и H, нам достаточно лишь заметить, что список H теперь лексикографически меньше, чем список G. Когда ставится метка 5, список H становится больше, чем список G, поскольку 5 является непосредственным преемником H, но не G. Если бы 5 было непосредственным преемником как G, так и H, относительное упорядочение их списков осталось бы тем же самым.

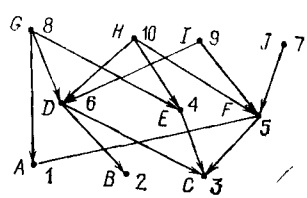


Рис. 2.9.

Алгоритм 2.2 может быть реализован с использованием очереди заданий, готовых к помечиванию, и списка LEX, который упорядочивает остальные задания по отношению к меткам, которые уже были приписаны. Пусть LEX_i представляет собой список после того, как метка i приписана. Когда метка $i+1$ приписывается некоторой вершине x, строится новый список LEX_{i+1}, который формируется из LEX_i путем просмотра LEX_i от наименьшего элемента к наибольшему и отбора непосредственных предшественников x. У этих заданий максимальная метка в их частичной последовательности равна $i+1$, поэтому они окажутся последними элементами списка LEX_{i+1}. Мы не заботились специально о тех предшественниках x, которых не было в LEX_i, и не говорили ничего о заданиях, готовых к помечиванию. Следующий пример иллюстрирует эту процедуру.

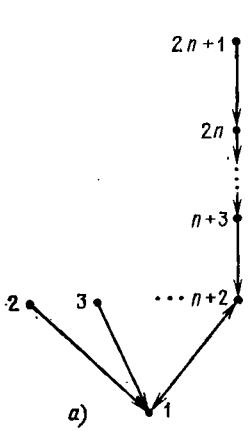
Пример 2. Рассмотрим систему заданий на рис. 2.9. Очередь заданий, готовых к помечиванию, и список LEX

в процессе расстановки меток имеют следующие состояния:

Назначаемая метка	1	2	3	4	5	6
Очередь LEX	*BC* *GF*	*C* *GFD*	*EFD* *G*	*FD* *HG*	*DJ* *GIH*	*JGIH* —

Метки с 7 по 10 назначаются заданиям J, G, I и H по мере того, как они выбираются из очереди.

Мы предоставляем читателю проделать процедуру модификации LEX, воспользовавшись предыдущим обсуждением. Далее рассмотрим вопрос о временной сложности процесса расстановки меток в целом. Если ответ на вопрос, является ли y непосредственным предшественником x, может быть получен за постоянное *) время, то для каждой помечаемой вершины построение модифицированного множества LEX для системы с n заданиями требует не более n шагов. При этом временная сложность составляет $O(n^2)$. Упомянутый вопрос может быть решен с помощью построенной заранее 0—1-матрицы, описывающей отношение предшествования.



После того, как в системе заданий расставлены все метки, следующий шаг заключается в построении по ним расписания. Как показано на рис. 2.10, составление списочного расписания в том виде, в котором мы его рассматривали ранее, в некоторых случаях может оказаться неэффектив-

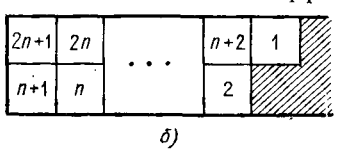


Рис. 2.10.

ным. Рассмотрим список заданий, с помощью которого строится расписание для системы, показанной на рис. 2.10. Если список просматривается слева направо до встречи с заданием, готовым к выполнению, то такой последователь-

*) Не зависящее от n. (*Прим. перев.*)

ный просмотр пройдет через $n-1$, $n-2$, . . ., 1 заданий соответственно. Построение расписания, таким образом, займет $O(n^2)$ шагов.

Мы покажем, что можно построить списочное расписание за почти линейное время. Алгоритм основан на двух обстоятельствах: 1) каждое задание имеет единичное время выполнения, и 2) в списке, с которым мы работаем, предшественники появляются раньше, чем их преемники: в списке $(T_n, T_{n-1}, \ldots, T_1)$ все предшественники задания T_i встречаются раньше T_i.

Неформально говоря, алгоритм строит расписание путем формирования последовательности расписаний для списков (T_n), (T_n, T_{n-1}) и так далее до тех пор, пока не будет построено расписание для полного списка $(T_n, T_{n-1}, \ldots, T_1)$.

Рассмотрим систему заданий на рис. 2.3. Расписания, соответствующие некоторым частичным спискам для данной системы заданий, показаны на рис. 2.11. Взяв списки (17) и (17, 16), мы получим расписания, представленные на рис. 2.11, *а*, *б*, где задания 16 и 17 назначены на процессоры 1 и 2 соответственно. Следующим рассматривается список (17, 16, 15). Задание 15 готово к выполнению уже на первом единичном интервале, но в это время отсутствует свободный процессор. Поэтому задание 15 назначается на

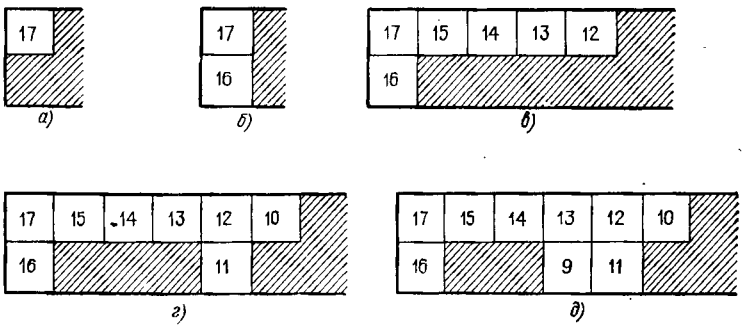


Рис. 2.11.

выполнение в следующем временном интервале. Как показывает следующая лемма, существует простой алгоритм построения расписания для нарастающего списка.

Лемма 2.4. *Пусть* S_{i+1} *— списочное расписание для* $(T_n, T_{n-1}, \ldots, T_{i+1})$; *тогда списочное расписание* S_i *для* $(T_n, T_{n-1}, \ldots, T_i)$ *может быть получено из* S_{i+1} *следующим образом. Пусть* t *— первый единичный интервал вре-*

мени, к началу которого выполнены все предшественники T_i. Назначение T_i на процессор осуществляется в течение первого единичного интервала u, $u \geqslant t$, на котором процессор оказывается свободным.

Доказательство. Для всех единичных интервалов, предшествующих t, задание T_i, по утверждению леммы, не готово к выполнению. Следовательно, даже если до него и доходит просмотр списка, оно не может быть назначено на выполнение. В процессе построения расписания S_i T_i может быть достигнуто в следующие после t интервалы, если имеется свободный процессор и нет других заданий, подлежащих назначению. Ясно, что такая ситуация соответствует первому после t свободному единичному интервалу в расписании S_{i+1}. Следовательно, лемма верна.

Лемма 2.4 дает нам надежду использовать повторяющиеся просмотры списка таким образом, чтобы при просмотрах определялись единичные временные интервалы, в течение которых процессор простаивает. Мы покажем, что с помощью эффективного алгоритма объединения непересекающихся множеств можно исключить повторные просмотры интервалов, в течение которых все процессоры заняты.

Предположим, что на интервале i все процессоры заняты и остаются занятыми до интервала j. Тогда для всех t, $i \leqslant t \leqslant j$, первым интервалом, на котором процессор свободен, является j. Соберем все такие t в набор с именем j. Вообще, мы выбираем такие имена наборов, что если t входит в набор u, то u и есть первый временной интервал, совпадающий с t или стоящий после него, на котором процессор свободен. Для того чтобы построить расписание, как это намечено в лемме 2.4, мы должны иметь возможность выявлять набор, в который входит свободный интервал, и определять первый интервал, на котором задание становится готовым к выполнению.

Алгоритм 2.3. Пусть D — орграф, в котором метки расставлены с помощью алгоритма 2.2, или дерево, в котором метки расставлены с помощью алгоритма 2.1. Настоящий алгоритм строит списочное расписание для системы заданий, представленной графом D, с помощью списка заданий, упорядоченного по убыванию меток.

В качестве структуры данных мы будем использовать множества, определенные для каждого единичного интервала. Вначале $SET(t)=\{t\}$ для каждого единичного интервала.

1. Все начальные задания из D объявляются готовыми в единичном интервале 1.

2. Для всех заданий x из D выполняется шаг 3 в порядке убывания меток.

3. Рассмотрим задание x. Пусть x готово к выполнению в единичном интервале t. Если t находится в $SET(u)$, назначаем x на процессор в единичном интервале u. Если на интервале u нет свободного процессора, рассмотрим интервал $u+1$. Пусть $u+1$ входит в $SET(v)$. Объединим $SET(u)$ и $SET(v)$ и назовем новый набор $SET(v)$.

Рассмотрим все y, являющиеся непосредственными преемниками x. Если все предшественники y уже рассмотрены, говорят, что y становится *готовым* в интервале $u+1$.

Для того чтобы доказать корректность алгоритма 2.3, мы вначале должны показать, что корректно производятся действия с наборами.

Лемма 2.5. *Пусть на некотором этапе интервал t входит в набор $SET(u)$. Тогда u является первым единичным интервалом, превосходящим t, на котором процессор оказывается свободным.*

Доказательство. Вначале интервал t входит в $SET(t)$ и все процессоры свободны. Предположим, что на некотором интервале u все процессоры загружены. Тогда u и $u+1$ имеют в качестве первого свободного единичного интервала один и тот же интервал. Если $u+1$ входит в $SET(v)$, то алгоритм объединяет $SET(u)$ и $SET(v)$ и называет новый набор $SET(v)$. Читателю предлагается построить более строгое доказательство, основывающееся на количестве выполнений шага 3 алгоритма.

Теорема 2.4. *Алгоритм* 2.3 *строит списочное расписание с помощью списка заданий, упорядоченного по убыванию меток.*

Доказательство. При просмотре списка задания встречаются в порядке убывания меток; поэтому, когда мы рассматриваем задание x, все предшественники x уже были назначены на выполнение. Таким образом, единичный интервал, на котором x становится готовым, известен. Из леммы 2.5 следует, что алгоритм правильно определяет следующий свободный единичный интервал. Поскольку расписание наращивается в соответствии с леммой 2.4, теорема должна быть верна.

Мы не вдаемся в подробности того, как осуществляются действия с наборами, отсылая читателя к [142, 75]. Заметим, что шаг 3 алгоритма 2.3 применяется к каждому ребру в системе заданий ровно один раз. Число операций с наборами пропорционально числу заданий n. В [142] показано,

что n операций с наборами могут быть выполнены за время, почти линейно зависящее от n. Более точно, n операций могут быть выполнены за время $O(n\alpha(n))$, где $\alpha(n)$ — функция, обратная функции Аккермана.

Следующая таблица дает представление о том, как медленно растет $\alpha(n)$:

n	$\alpha(n)$	n	$\alpha(n)$
255	1	65536	3
256	2	$2^{65536}-1$	3
65535	2	2^{65536}	4

Пусть $A(i, j)$ — целочисленная функция, определяемая следующим образом:

$$A(0, j) = 2j,$$
$$A(i, 0) = 0, \quad A(i, 1) = 2,$$
$$A(i, j) = A(i-1, A(i, j-1)) \quad \text{при} \quad i \geqslant 1, \ j \geqslant 2.$$

Тогда

$$\alpha(n) = \min\{i \mid A(i, 3) > \lfloor \log_2 n \rfloor\}.$$

2.5. Прерывания при независимых заданиях

В предыдущих параграфах предполагалось, что выполнение задания, раз начавшись, происходит до его завершения. В § 2.1 мы отметили, что на существование полиномиального алгоритма составления расписаний без прерываний можно надеяться только в том случае, когда все задания имеют одинаковые времена выполнения. В двух частных случаях — когда система заданий представляет собой лес и когда имеются только два процессора — мы получили полиномиальные алгоритмы нахождения расписаний минимальной длины.

Обратимся теперь к расписаниям с прерываниями. Поскольку в этом случае не требуется, чтобы задание выполнялось до конца, времена выполнения могут считаться произвольными. Оказывается, что известные полиномиальные алгоритмы с прерываниями очень тесно связаны с алгоритмами без прерываний, рассмотренными в §§ 2.2, 2.3. Кроме того, мы должны рассмотреть случай, когда задания независимы, поскольку теперь времена выполнения считаются произвольными. Для независимых заданий существует простой алгоритм определения расписания мини-

мальной длины с прерываниями. Пусть независимые задания $T_1, T_2, \ldots, T_n$ с временами выполнения $\tau_1, \tau_2, \ldots, \tau_n$ должны быть выполнены на m процессорах. Расписание наименьшей длины соответствует ситуации, когда отсутствуют простои процессоров. Длина такого расписания есть $\omega = X/m$, где $X = \sum_{i=1}^{n} \tau_i$. Мы увидим позднее, что расписание такой длины действительно может быть составлено в том случае, когда время выполнения каждого задания не превышает X/m. Если одно из заданий оказывается более длинным, чем X/m, то ω равна длине наибольшего задания.

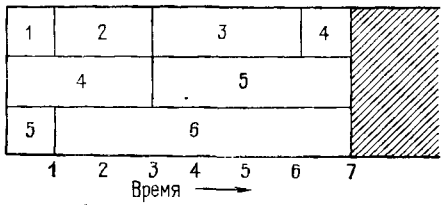


Рис. 2.12.

Работа алгоритма заключается, в общих чертах, в следующем. Вначале вычисляем ω так, как в предыдущем абзаце. Для системы заданий, изображенной на рис. 2.12, $\omega = 7$. Назначаем задания на первый процессор до тех пор, пока одно из них не превысит срок ω, означающий конец расписания. Назначаем остаток на следующий процессор, начиная с момента времени 0. На рис. 2.12 показано, что задание 4 не укладывается в интервал (6,7) и его оставшаяся часть переносится на процессор P_2. Заметим, что интервалы выполнения одного задания на двух процессорах не перекрываются, так как, согласно нашему выбору, величина ω, по крайней мере, не меньше, чем длина самого длинного задания.

Алгоритм 2.4. Пусть $(\mathscr{T}, <)$ — система, состоящая из независимых заданий $T_1, T_2, \ldots, T_n$ с временами выполнения $\tau_1, \tau_2, \ldots, \tau_n$. Приведенный ниже алгоритм определяет расписание минимальной длины с прерываниями для системы заданий, выполняемой на m процессорах. Время работы алгоритма, очевидно, составляет $O(n)$.

1. Пусть $X = \sum_{i=1}^{n} \tau_i$, и пусть τ есть длина самого длинного задания в системе; длина будущего расписания ω определяется выражением $\omega = \max\{X/m, \tau\}$.

2. Вначале задания не назначены ни на один из процессоров. Применим шаг 3 к заданиям T_1, T_2,. . ., T_n.

3. Пусть задания T_1, T_2,. . ., T_{i-1} уже назначены на процессоры P_1, P_2,. . ., P_{j-1} на интервале (0, ω) и на процессор P_j — на интервале (0, t) для некоторого $t<\omega$. Рассмотрим задание T_i. Если $t+\tau_i\leqslant\omega$, назначим T_i на P_j на интервале (t, $t+\tau_i$). В противном случае назначим T_i на P_j на интервале (t, ω) и на P_{j+1} на интервале (0, $\tau_i-(\omega-t)$).

На рис. 2.1, *б* приведен пример расписания, составленного в соответствии с алгоритмом 2.4. Другой пример приведен на рис. 2.12. Теперь мы покажем, что алгоритм 2.4 действительно определяет оптимальное расписание.

Т е о р е м а 2.5. *Алгоритм* 2.4 *строит расписание минимальной длины с прерываниями для системы независимых заданий, выполняемых на* m *процессорах.*

Д о к а з а т е л ь с т в о. Ясно, что в лучшем из всех мыслимых расписаний все задания выполняются за время $\omega=\max\{X/m$, длина самого длинного задания$\}$, где X есть сумма времен выполнения всех заданий. Из описания шага 3 алгоритма следует, что в каждый момент времени на один и тот же процессор назначается не более одного задания. Поскольку ω превышает длину самого длинного задания в системе, то в каждый момент времени одно и то же задание назначается не более чем на один процессор. Ни одно задание не назначается на время более позднее, чем ω. Более того, поскольку ωm заведомо не меньше X, имеется возможность разместить все задания внутри интервала (0, ω). Следовательно, расписание, определяемое алгоритмом 2.4, является оптимальным.

С каждым прерыванием связаны затраты, пусть даже и небольшие, поэтому нас интересует наибольшее количество прерываний, которое может встретиться в расписании минимальной длины. В гл. 1 было показано, что определение расписания минимальной длины с наименьшим количеством прерываний является достаточно сложной задачей, выходящей за рамки нашего обсуждения. Здесь мы ограничимся определением верхней оценки для количества прерываний. По сравнению с оценкой алгоритм 2.4 дает хорошие результаты.

Говорят, что задание T *прервано* в момент t, если в момент t выполнение T на некотором процессоре приостановлено, но при этом еще не завершено. Говорят, что задание является *активным* в момент t, если в этот момент какой-либо процессор выполняет T. Если задание начинает выпол-

няться на процессоре P в момент t и выполняется непрерывно на P до момента t', где t' максимально, интервал (t, t') называется *активным периодом* для задания на процессоре P.

Теорема 2.6. *Пусть S — расписание, построенное с помощью алгоритма 2.4 для системы из независимых заданий, выполняемых на m идентичных процессорах. Тогда S имеет не более $m-1$ прерываний. Более того, оценка $m-1$ для числа прерываний неулучшаема в том смысле, что существует такая система заданий, для которой всякое расписание минимальной длины содержит не менее $m-1$ прерываний.*

Доказательство. Из алгоритма 2.4 следует, что каждое задание имеет не более двух активных периодов. Когда задание имеет два активных периода, первый из них начинается в момент времени 0 на некотором процессоре, а второй период оканчивается в конце расписания на процессоре с меньшим номером. Таких заданий может быть не более $m-1$, и каждое из них прерывается один раз. Следовательно, расписание, определяемое алгоритмом, содержит не более $m-1$ прерываний.

Теперь мы покажем, что оптимальное расписание для системы, состоящей из $m+1$ заданий длины m, будет иметь по меньшей мере $m-1$ прерываний. Предположим, что для такой системы может быть составлено оптимальное расписание, содержащее меньше чем $m-1$ прерываний. Тогда в расписании будет не меньше двух процессоров, на которых

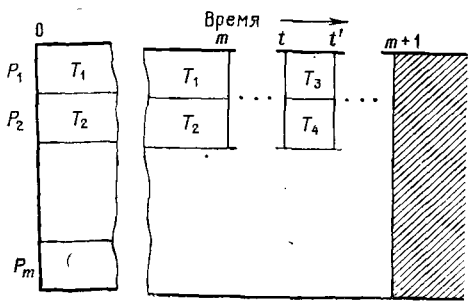


Рис. 2.13.

нет ни одного прерывания. Можно считать без потери общности, что это P_1 и P_2. Поскольку все задания имеют длину m, на интервале $(0, m)$ эти процессоры выполняют некоторые задания T_1 и T_2. Из алгоритма 2.4 следует, что оптимальное расписание для данной системы заданий не имеет

простоев и его длина равна $m+1$. Таким образом, P_1 и P_2 должны выполнять задания до момента $m+1$. Пусть для некоторых t и t', таких, что $m \leqslant t < t' \leqslant m+1$, в интервале (t, t') на P_1 выполняется T_3, а на P_2 — T_4 (см. рис. 2.13). Заметим, что возобновление выполнения T_3 и T_4 на P_1 и P_2 не считается прерыванием. В течение интервала (t, t') на оставшихся $m-2$ процессорах могут выполняться только задания T_5, T_6, . . ., T_{m+1}. Следовательно, мы имеем только $m-3$ задания, которые должны быть назначены на $m-2$ процессора в интервале (t, t'), т. е. в расписании должны быть простои. Следовательно, такое расписание неоптимально, что и доказывает теорему.

2.6. Обобщения алгоритмов без прерываний

Рассмотрим систему заданий $(\mathscr{T}, <)$, приведенную на рис. 2.14,*а*. Оптимальное расписание без прерываний для этой системы имеет длину 9 (см. рис. 2.14,*в*). Множество

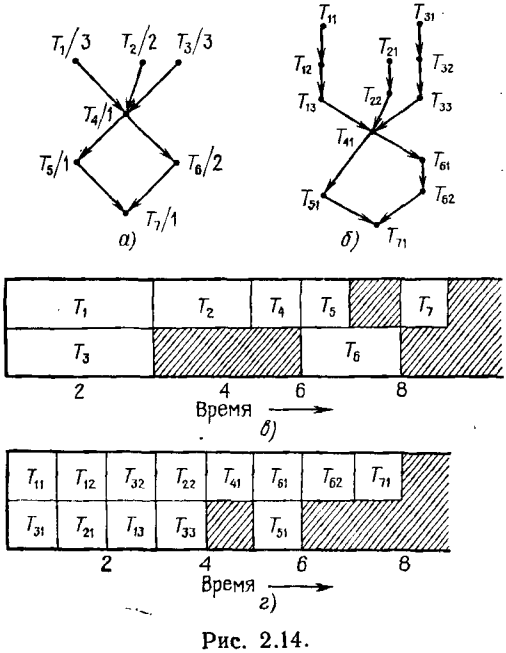


Рис. 2.14.

заданий $\{U_1, U_2, \ldots, U_k\}$ назовем цепочкой, если $U_1 < U_2 < \ldots < U_k$. Посмотрим, что получится, если расщепить каждое задание из $(\mathscr{T}, <)$ на цепочки заданий еди-

ничной длины (см. рис. 2.14, *б*). Оптимальное расписание без прерываний для новой системы заданий имеет длину 8 (см. рис. 2.14,*г*). Таким образом, расщепление каждого задания на цепочки заданий меньшего размера приводит, по крайней мере в данном примере, к лучшему приближению без прерываний оптимального расписания с прерываниями.

Алгоритмы, строящие расписания без прерываний, рассмотренные в предыдущих параграфах данной главы, применимы к системам заданий с одинаковыми временами выполнения. Мы сможем использовать эти алгоритмы для получения приближенных расписаний, если удастся превратить задания произвольной длины в цепочки, состоящие из заданий одинаковой длины. Поэтому сделаем следующее предположение: пусть $(\mathscr{T}, <)$ есть система заданий с временами выполнения $\tau_1, \tau_2, \ldots, \tau_n$. Если существует действительное положительное число w, такое, что времена выполнения всех заданий являются целыми кратными w, то задания из $\mathscr{T}$ называют *соизмеримыми*. Мы предполагаем, что все рассматриваемые системы состоят из соизмеримых заданий. Это предположение не слишком ограничительно, так как любые действительные числа могут быть с любой степенью точности приближены рациональными числами. Кроме того, числа, которые вводятся в вычислительную машину, ввиду ограниченной точности их представления являются соизмеримыми.

Пусть G—ориентированный ациклический граф, представляющий систему заданий $(\mathscr{T}, <)$. Вершина x_i в G представляет задание T_i длины τ_i для всех i, $1 \leqslant i \leqslant n$. Учитывая тесную взаимосвязь между системой заданий и графом, мы используем термины «вершина» и «задание» как синонимы. Более того, мы иногда говорим о графе G так, как будто это есть система заданий, для которой нужно составить расписание. Определим G_w как граф, получающийся из G заменой вершины T_i цепочкой T_{i1}, $T_{i2}, \ldots, T_{in_i}$, где $n_i = \tau_i/w$. Предшественники T_i становятся предшественниками T_{i1}, а преемник T_i становится преемником T_{in_i} (см. рис. 2.14). После этого каждая вершина в G_w представляет задание длины w, что позволяет применить к G_w алгоритмы составления расписаний без прерываний, рассмотренные ранее в этой главе. Вершины в G_w могут быть еще раз расщеплены и представлены в виде цепочек заданий одинаковой длины. Определим $G_{w/k}$ как граф, получающийся из G_w заменой каждой

вершины k-звенной цепочкой заданий одинаковой длины. При увеличении k можно ожидать, что оптимальное расписание без прерываний для $G_{w/k}$ становится все ближе к оптимальному расписанию с прерываниями для G.

Пусть $\omega(G, m)$ и $\omega'(G, m)$ — длины оптимального расписания без прерываний и оптимального расписания с прерываниями для системы, представленной графом G и выполняемой на m одинаковых процессорах.

Теорема 2.7. *Пусть G — граф, представляющий систему заданий $(\mathscr{T}, <)$, состоящую из заданий $T_1, T_2, \ldots, T_n$ с временами выполнения $\tau_1, \tau_2, \ldots, \tau_n$, w — положительное действительное число, которому пропорциональны все времена выполнения. Тогда $\omega'(G, m) \leqslant \omega(G_{w/k}, m) \leqslant \omega'(G, m) + C/k$, $k = 1, 2, \ldots$, где C — константа, зависящая только от G.*

Доказательство. Пусть S — оптимальное расписание с прерываниями для G. Поскольку G содержит n заданий, в расписании будет не более n моментов завершения заданий. Пусть этими моментами будут $t_1 \leqslant t_2 \leqslant \ldots \leqslant t_n$. Заметим, что t_i не обязаны различаться, потому что в одно и то же время могут быть завершены несколько заданий. Пусть t_0 — начало выполнения системы.

Рассмотрим интервалы $I_i = (t_{i-1}, t_i)$ для всех i, $1 \leqslant i \leqslant n$. Поскольку ни одно задание не завершается внутри интервала I_i, все задания, выполняющиеся на нем, должны быть независимы. Поскольку S может быть любым оптимальным расписанием с прерываниями, то в интервале I_i может произойти произвольное число прерываний. Пусть в интервале I_i выполняются задания $T_{i1}, T_{i2}, \ldots, T_{in_i}$, и пусть λ_j есть время, в течение которого на этом интервале выполнялось задание T_{ij}. Рассматривая T_{ij} как задания длины λ_j, мы можем перераспределить интервал I_i с помощью алгоритма 2.4. Обозначим через S' расписание, полученное в результате перераспределения во всех интервалах. Перераспределение позволяет нам ограничить число активных периодов каждого задания внутри интервала. Заметим, что длина расписания S' такая же, как длина S. Пусть $S'(t, t')$ означает часть расписания S', заключенную между моментами t и t'. Для любого i, $1 \leqslant i \leqslant n$, $S'(t_{i-1}, t_i)$ соответствует интервалу I_i в расписании S.

Поскольку нет никаких ограничений на моменты возникновения прерываний, в общем случае длина активного периода для данного задания не будет кратной w/k. Можно построить новое расписание S'', в котором каждый активный период является кратным w/k. Вдобавок S'' может быть

построено из n сегментов S''_i, $1 \leqslant i \leqslant n$, конкатенация которых дает S''. Для всех i, $1 \leqslant i \leqslant n$, построим S''_i следующим образом: удлиним каждый активный период в $S'(t_{i-1}, t_i)$ так, чтобы он стал кратным w/k, а в конец расписания добавим фиктивные задания так, чтобы все процессоры окончили работу в одно и то же время.

Мы утверждаем, что S'' не более чем на $2n^2w/k$ длиннее расписания S' и, следовательно, S. Рассмотрим $S'(t_{i-1}, t_i)$, соответствующую интервалу I_i в S. В худшем случае в этом интервале присутствуют все задания. Из алгоритма 2.4 следует, что у каждого задания в $S'(t_{i-1}, t_i)$ может быть не более двух активных периодов. Поэтому длина сегмента S'' превосходит длину $S'(t_{i-1}, t_i)$ не более чем на $2nw/k$. Поскольку в S'' содержится не более n сегментов, то длина S'' не более чем на $2n^2w/k$ превышает длину S и S'.

Длина каждого активного периода в S'' кратна w/k. Следовательно, мы можем использовать S'' для построения расписания без прерываний для $G_{w/k}$. Поскольку при построении S'' времена выполнения заданий были увеличены, для получения расписания без прерываний для $G_{w/k}$ нам, возможно, придется допустить простои. Ясно, что оптимальное расписание без прерываний для $G_{w/k}$ не может быть длиннее S''. Из этого следует, что $\omega(G_{w/k}, m) \leqslant \omega'(G, m) + 2n^2w/k$. Нижняя оценка теоремы очевидна, так как любое расписание без прерываний является частным случаем расписания с прерываниями.

Из теоремы 2.7 следует, что всякий раз, когда мы можем построить оптимальное расписание без прерываний для системы, состоящей из заданий с одинаковыми временами выполнения, мы можем построить расписание с прерываниями для системы заданий с произвольными временами выполнения, которое будет сколь угодно близким к оптимальному. Немногие случаи, для которых известны алгоритмы построения расписаний без прерываний, рассмотрены в §§ 2.2, 2.3, а именно деревья при произвольном числе процессоров и произвольные частичные упорядочения при двух процессорах.

Рассмотрим алгоритм расстановки меток, приведенный в § 2.3, который был использован для построения расписания при произвольном частичном упорядочении и двух процессорах. В нем метки расставлялись от уровня к уровню. Как следует из § 2.2, уровневая стратегия строит расписания для деревьев от уровня к уровню. По мере того, как задания расщепляются на цепочки более мелких заданий, количество уровней увеличивается и в пределе становится

бесконечным. При этом даже системы заданий с произвольным отношением предшествования могут выполняться уровневыми алгоритмами.

Переработаем поуровневую процедуру выполнения $G_{w/k}$ в расписание для G. Уровни определяются по отношению к конечной вершине, поэтому нам необходима мера расстояния от задания до конечной вершины, которая была бы выражена в терминах времен выполнения заданий, следующих за рассматриваемым. Задания одного и того же уровня обслуживаются одинаково. Для того чтобы наглядно показать, как это получается, введем понятие *разделение процессора*. Если процессоров оказывается больше, чем заданий, то каждое задание получает процессор в собственное пользование. Иначе задания должны делить доступные процессоры между собой. Таким образом, каждое задание получает α «процессоров», где $0<\alpha\leqslant 1$. Вместо того, чтобы говорить, что задание T имеет время выполнения τ, сопоставим T *требование на обслуживание*, составляющее τ единиц процессорного времени. Пример системы заданий, расписание для которой составлено с использованием разделения процессоров, приведен на рис. 2.15. В случае отсутствия прерываний задание T, назначенное на α «процессоров», будет выполняться в течение τ/α единиц времени, причем $\tau/\alpha \geqslant \tau$, так как $0<\alpha\leqslant 1$. Таким образом, α может рассматриваться как скорость, с которой выполняется T.

Пусть $(\mathscr{T}, <)$—система, состоящая из заданий T_1, T_2, ..., T_n с требованиями на обслуживание $\tau_1, \tau_2, \ldots, \tau_n$; $T_{i1}, T_{i2}, \ldots, T_{in_i}$—цепь в $\mathscr{T}$. Говорят, что цепь *начинается* с задания T_{i1}; *длина* цепи есть $\sum_{j=1}^{n_i} \tau_{ij}$. Уровень задания T в $\mathscr{T}$ равен длине максимальной цепи, начинающейся с задания T. Заметим, что данное определение согласуется с определением уровней в UET-системах.

Алгоритм 2.5. Пусть $(\mathscr{T}, <)$—система, состоящая из соизмеримых заданий. Данный алгоритм вначале определяет для этой системы расписание с разделением процессоров, а затем преобразует его в расписание с прерываниями.

1. Пусть имеется a доступных процессоров, и пусть на верхнем уровне находится b заданий. Если $b>a$, выделим a/b процессоров каждому из b заданий. В противном случае выделим по одному процессору на каждое задание. Если при этом остаются свободные процессоры, рассмотрим задания со следующего по высоте уровня и так далее.

2. Перераспределяем все процессоры среди невыполненных заданий, если происходит одно из следующих событий:

Событие 1: завершено выполнение задания.

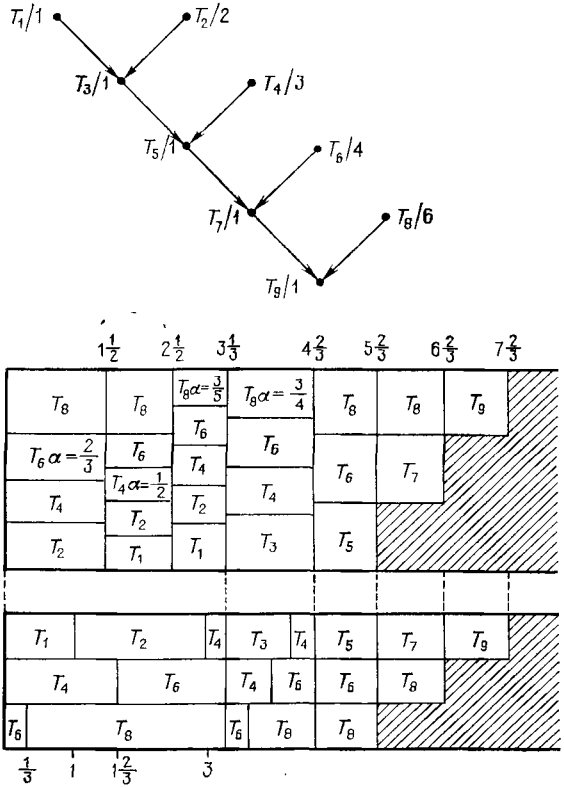


Рис. 2.15.

Событие 2: достигнута точка, в которой продолжение работы с прежним распределением процессоров приводит к тому, что задания более низкого уровня будут обслуживаться с большей скоростью, чем задания более высокого уровня.

3. Пусть $t_1 \leqslant t_2 \leqslant \ldots \leqslant t_n$ — моменты завершения заданий в расписании, построенном к концу выполнения шага 2. Заметим, что в моменты $t_1, t_2, \ldots, t_n$ происходит событие 1. Пусть t_0 — момент начала работы. Ясно, что все задания, выполняемые в интервале (t_{i-1}, t_i), для всех i, $1 \leqslant i \leqslant n$,

должны быть независимыми. Для построения расписания с прерываниями для нашей системы применим алгоритм 2.4 к порциям заданий, выполняемым внутри каждого такого интервала.

Пример 3. Рассмотрим систему заданий на рис. 2.15. Для нахождения расписания минимальной длины с прерываниями для данной системы заданий воспользуемся алгоритмом 2.5.

Для начала заметим, что из готовых заданий T_8 имеет уровень 7, T_2, T_4 и T_6 имеют уровень 6 и T_1 имеет уровень 5. Согласно шагу 1, заданию T_8 предоставляется целиком процессор. Задания T_2, T_4 и T_6 находятся на следующем по высоте уровне, и каждое из них получает по 2/3 процессора. По истечении 3/2 единиц времени задания T_2, T_4 и T_6 отработают по единице процессорного времени каждое и будут настигнуты на уровне 5 заданием T_1; возникнет ситуация 2.

Теперь рассмотрим уровни невыполненной части системы заданий. Задание T_8 при переназначении получает высший уровень $5\frac{1}{2}$, и, поскольку оно по-прежнему единственно на этом уровне, ему предоставляется процессор целиком. Задания T_1, T_2, T_4 и T_6, находящиеся на уровне 5, должны разделить между собой оставшиеся два процессора. В момент времени $2\frac{1}{2}$ задание T_8 находится на уровне $4\frac{1}{2}$ и по-прежнему не выполнено. Задания T_1, T_2, T_4 и T_6 выполнялись со скоростью 1/2 процессора и также достигли уровня $4\frac{1}{2}$, так что опять произойдет событие 2. Теперь задания T_1, T_2, T_4, T_6 и T_8 находятся на одном и том же уровне и распределяют между собой поровну три процессора до тех пор, пока в момент $3\frac{1}{3}$ не завершится выполнение T_1 и T_2; произойдет событие 1.

Когда расписание с разделением процессоров найдено, остается построить расписание с прерываниями. Отметим, что первыми завершены в момент времени $3\frac{1}{3}$ задания T_1 и T_2. Перераспределяя интервалы между временами окончания заданий, мы получим расписание с прерываниями, показанное на рис. 2.15.

Пусть G—граф системы заданий $(\mathscr{T}, <)$ и времена выполнения всех заданий из $\mathscr{T}$ кратны w. Рассмотрим уровневое расписание для $G_{w/k}$. Мы ожидаем, что по мере увеличения k уровневое расписание все больше и больше приближается к расписанию с разделением процессоров,

порождаемому алгоритмом 2.5. Таким образом, оба расписания в пределе имеют одну и ту же длину. Для деревьев уровневое расписание является оптимальным. Поэтому, как следует из теоремы 2.7, в пределе расписание с разделением процессоров имеет такую же длину, как оптимальное расписание с прерываниями. В действительности справедливо более сильное утверждение.

Теорема 2.8. *Алгоритм* 2.5 *строит оптимальное расписание для системы заданий* $(\mathscr{T}, <)$, *если граф* $(\mathscr{T}, <)$ *представляет собой лес или если расписание строится для двух процессоров.*

Доказательство. Мы наметим общий путь доказательства, детали которого можно найти в [110, 111].

1. В расписании с разделением процессоров в каждом интервале между событиями каждое задание выполняется в течение рационального отрезка времени.

2. В процессе доказательства утверждения 1 мы можем определить, с какой точностью нужно расщеплять задания. Поскольку каждое задание выполняется в течение рационального времени, мы можем найти такое k, чтобы уровневое расписание для $G_{\omega/k}$ «укладывалось» в интервалы между событиями.

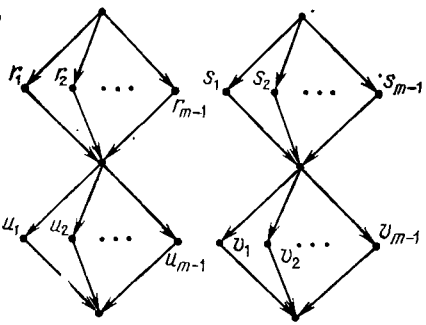


Рис. 2.16. Пример уровневого расписания с прерываниями, имеющего длину $2(3-2/m)+1$, в то время как оптимальное расписание без прерываний имеет длину $2(2)+2$. При добавлении к системе заданий новых уровней отношение длин уровневого и оптимального расписаний приближается к $3/2-1/m$.

3. Из § 2.2 следует, что уровневое расписание является оптимальным для деревьев. Можно показать, что для орграфов и четных k уровневое расписание является оптимальным для $G_{\omega/k}$ при двух процессорах.

4. Заметим теперь, что дальнейшее дробление $G_{\omega/k}$ не может привести к более короткому расписанию. В частности, для всех j $\omega(G_{\omega/k}, m) = \omega(G_{\omega/kj}, m)$. Из теоремы 2.7 следует, что $\omega(G_{\omega/kj}, m)$ достигает длины оптимального расписания с прерываниями для G. Из этого вытекает, что величина $\omega(G_{\omega/k}, m)$ должна быть равна $\omega'(G, m)$. Поскольку расписание с разделением процессоров имеет такую же длину, как и $\omega(G_{\omega/k}, m)$, теорема 2.8 верна.

Как и в случае, когда прерывания запрещены, уровневый алгоритм не приводит к оптимальному расписанию для орграфов, если число процессоров больше двух. Пример изображен на рис. 2.16. В недавней работе [106] показано, что верхняя оценка отношения длины уровневого расписания к длине оптимального расписания с прерываниями равна $2-2/m$.

2.7. Преимущества прерываний

На рис. 2.1 показано, что для данной системы заданий расписание с прерываниями может оказаться короче, чем расписание минимальной длины без прерываний. В этом параграфе мы сравним длины расписаний с прерываниями и без прерываний.

Кроме расписаний общего вида, имеется специальный класс расписаний, представляющий интерес. Это расписания, называемые списочными; они строятся путем назначения приоритетов заданиям с последующим назначением заданий на процессоры в порядке этих приоритетов. Существенная разница между расписанием без прерываний общего вида и списочным расписанием заключается в том, что в последнем случае процессор не может простаивать при наличии заданий, готовых к выполнению. Как мы увидим в данном параграфе это ограничение приводит к тому, что оптимальное списочное расписание может быть длиннее, чем общее расписание без прерываний. Свойства списочных расписаний более подробно исследованы в гл. 5.

Пусть $\omega_L(G, m)$ — длина кратчайшего списочного расписания *) для системы заданий, определяемой графом G, на m идентичных процессорах. Напомним, что в § 2.6 были введены величины $\omega(G, m)$ и $\omega'(G, m)$, представляющие собой длины кратчайших расписаний без прерываний и с прерываниями соответственно. Когда ясно, о каких G и m идет речь, мы будем опускать их и писать просто ω_L, ω, ω'.

Можно утверждать, что $\omega \leqslant \omega_L$, поскольку в списочных расписаниях задания выполняются без прерываний. Более того, класс расписаний без прерываний содержится в классе расписаний с прерываниями, и поэтому $\omega' \leqslant \omega$. Можно доказать, что $\omega_L/\omega' \leqslant 2-1/m$. Из $\omega \leqslant \omega_L$ теперь немедленно следует, что $\omega/\omega' \leqslant 2-1/m$. Пример, изображенный на рис. 2.17, показывает, что предельное значение $2-1/m$ может быть достигнуто величиной ω_L/ω' с любой степенью

*) Кратчайшего по всем возможным спискам.

точности. Для ω/ω' столь точная оценка отсутствует. В примере, который мы можем построить, это отношение равно $2m/(m+1)$, что немного меньше, чем $2-1/m$.

Для того чтобы доказать приведенные выше оценки, нам необходима лемма, которая доказывается в гл. 5.

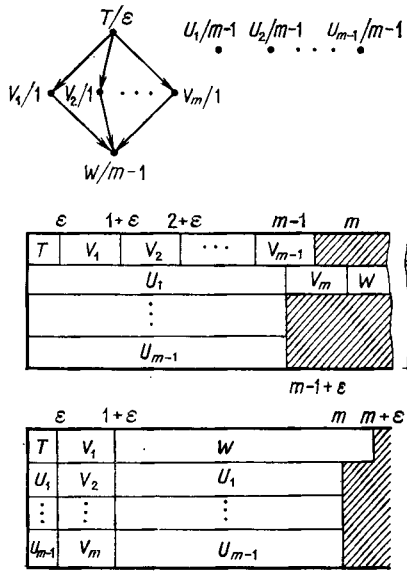


Рис. 2.17.

Лемма 2.6. *Пусть S — списочное расписание на m процессорах для системы заданий $(\mathscr{T}, <)$, I — суммарное время простоя процессоров в расписании S. Тогда в $(\mathscr{T}, <)$ существует цепочка заданий, длина которой не меньше $I/(m-1)$.*

Доказательство. См. теорему 5.1.

Лемма 2.7. *Пусть ω_L и ω' — длины кратчайшего списочного расписания и кратчайшего расписания с прерываниями для системы $(\mathscr{T}, <)$ на m идентичных процессорах. Тогда $\omega_L/\omega' \leqslant 2-1/m$.*

Доказательство. Пусть S — кратчайшее списочное расписание (длины ω_L) на m процессорах для системы заданий $(\mathscr{T}, <)$. Если I есть суммарное время простоя в S, то, как следует из леммы 2.6, в $(\mathscr{T}, <)$ существует цепочка заданий длины, не меньшей $I/(m-1)$. Таким образом, $\omega' \geqslant I/(m-1)$, что дает $I \leqslant \omega'(m-1)$.

Пусть X — сумма времен выполнения всех заданий в $\mathscr{T}$. Ни одно расписание, независимо от наличия прерываний, не может обеспечить ничего лучшего, чем полностью устранить простои процессоров. Поэтому $\omega' \geqslant X/m$, откуда $X \leqslant m\omega'$.

Из определения I и X имеем

$$\omega_L = (I + X)/m.$$

Подставляя оценки для I и X через ω', получаем

$$\omega_L \leqslant ((m-1)\,\omega' + m\omega')/m,$$

откуда немедленно следует, что $\omega_L/\omega' \leqslant 2 - 1/m$.

Как уже было отмечено, система заданий, изображенная на рис. 2.17, показывает, что величина $2-1/m$ является наилучшей оценкой для ω_L/ω'. На этом рисунке также дано сравнение списочного расписания и расписания общего вида без прерываний. Рассмотрим систему заданий на рис. 2.17. Предположим, что задание T выполняется на процессоре 1; при этом процессоры от 2 до m простаивают на интервале $(0, \varepsilon)$. В момент ε процессоры могут начать выполнение $V_1, V_2, \ldots, V_m$. В момент $1+\varepsilon$ процессоры могут начать выполнение W и $U_1, U_2, \ldots, U_{m-1}$, окончив выполнение всей системы заданий в момент $m+\varepsilon$. Таким образом, как видно из этого примера, ω_L/ω стремится к $2-1/m$.

Объединим все полученные оценки в следующей теореме.

Т е о р е м а 2.9. *Пусть* ω_L, ω *и* ω' — *длины кратчайших расписаний для системы* $(\mathscr{T}, <)$ *на* m *идентичных процессорах — соответственно списочного, без прерываний и с прерываниями. Тогда*:

1) $\omega' \leqslant \omega \leqslant \omega_L$;

2) $\omega_L/\omega' \leqslant 2 - 1/m$;

3) $\omega_L/\omega \leqslant 2 - 1/m$;

4) *существуют системы заданий, для которых* ω_L/ω' *и* ω_L/ω *стремятся к* $2-1/m$;

5) *существует система заданий, для которой* $\omega/\omega' = 2m/(m+1)$.

Д о к а з а т е л ь с т в о. Единственным утверждением, которое не вытекает из предыдущего обсуждения, является 5). Рассмотрим систему, состоящую из $m+1$ одинаковых заданий длины m каждое. Кратчайшее расписание без прерываний имеет длину $2m$; кратчайшее расписание с прерываниями имеет длину $m+1$.

Рассмотрим систему заданий на рис. 2.17. Пусть $(\mathscr{T}, <^R)$ получается из нее изменением ориентации всех дуг, задающих ограничения предшествования. Независимо от используемого списка, списочное расписание для $(\mathscr{T}, <^R)$ имеет длину $m+\varepsilon$. Таким образом, изменение ориентации дуг приводит к более короткому списочному расписанию. Такая аномалия не возникает в расписании без прерываний общего вида.

Пусть S—расписание для системы заданий $(\mathscr{T}, <)$. Обозначим через $(\mathscr{T}, <^R)$ систему заданий, образованную из $(\mathscr{T}, <)$ изменением ориентации всех ребер в графе предшествования $(\mathscr{T}, <)$ (т. е. $T <^R T'$ тогда и только тогда, когда $T' < T$). Пусть S^R есть расписание, *обратное* к S (зеркальное изображение: момент завершения S является начальным моментом в S^R, и последовательность выполнений заданий в S^R определяется путем чтения S в обратном направлении по оси времени). Заметим, что в S^R процессор может оставаться свободным, даже если в $(\mathscr{T}, <^R)$ имеются готовые к выполнению задания.

Теорема 2.10. *Пусть S есть расписание с прерываниями или расписание без прерываний общего вида для системы $(\mathscr{T}, <)$. S является кратчайшим расписанием для $(\mathscr{T}, <)$ тогда и только тогда, когда S^R—кратчайшее расписание для $(\mathscr{T}, <^R)$.* (Эта теорема неприменима к списочным расписаниям.)

Доказательство. В силу симметрии нам нужно доказать только необходимость. Вначале заметим, что S^R действительно расписание для $(\mathscr{T}, <^R)$, так как в нем соблюдаются ограничения предшествования и каждое задание выполняется в течение положенного времени. Предположим, что S—кратчайшее расписание для $(\mathscr{T}, <)$, но S^R не является таковым для $(\mathscr{T}, <^R)$. Тогда существует расписание S', более короткое, чем S^R, которое оказывается кратчайшим для $(\mathscr{T}, <^R)$. Поскольку $<=(<^R)^R$, $(S')^R$ должно быть расписанием для $(\mathscr{T}, <)$. Но $(S')^R$ короче, чем S, что противоречит минимальности S. Теорема доказана.

Интересный случай, не охватываемый теоремой 2.10, встречается, когда $(\mathscr{T}, <)$ представляет собой UET-лес. Несмотря на то, что при построении оптимального расписания с помощью алгоритма 2.1 используется список, теорема 2.1 утверждает, что S является кратчайшим из всех расписаний без прерываний. Таким образом, аргументы, аналогичные использованным при доказательстве теоремы 2.10, применимы к UET-лесам и антилесам.

2.8. Процессоры с разным быстродействием

До сих пор в данной главе мы рассматривали процессоры, имеющие одинаковые вычислительные возможности. В общем случае процессоры могут выполнять задания с разной скоростью. Даже если аппаратная часть процессоров одинакова, процессор P_1 может часть своего времени затрачивать, например, на обслуживание удаленного терминала. Если доля времени, затрачиваемого процессором на выполнение других функций, относительно постоянна, то такой процессор может рассматриваться как имеющий меньшее быстродействие. И наконец, всегда имеется возможность, что в систему будет включен быстрый процессор.

Итак, мы имеем набор из m процессоров $P=\{P_1, P_2, \ldots, P_m\}$, быстродействия которых суть $b_1 \geqslant b_2 \geqslant \ldots \geqslant b_m$. Для обозначения этого набора используем запись $P=(b_1, b_2, \ldots, b_m)$. В этом параграфе мы рассмотрим задачу определения кратчайшего расписания с прерываниями для множества независимых заданий. Наличие признаков «трудной» комбинаторной задачи заставляет отбросить попытку поиска для этой задачи расписания минимальной длины без прерываний. Рассмотрим набор независимых заданий, имеющих потребности в ресурсе *) $\tau_1 \geqslant \tau_2 \geqslant \ldots \geqslant \tau_n$. Так же как и для процессоров, запись $\mathcal{T}=(\tau_1, \tau_2, \ldots, \tau_n)$ обозначает набор заданий и их потребности в ресурсе. Задание T_i выполняется на процессоре P_j со скоростью b_j.

Предположим, мы хотим исполнить $\mathcal{T}=(8, 7)$ на $P=(2, 1)$. Для выполнения задания T_1 на процессоре P_1 требуется 4 единицы времени. Если T_1 назначить на P_1, а

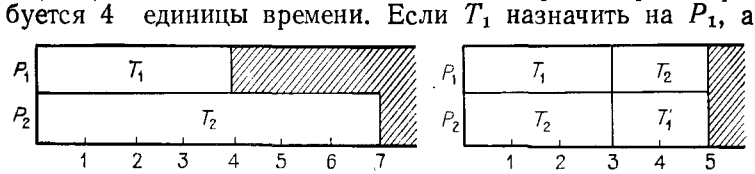


Рис. 2.18.

T_2 на P_2, как показано на рис. 2.18, *а*, то получим расписание длиной 7. Более короткое расписание можно получить, если замедлить выполнение T_1 и ускорить выполнение T_2, как показано на рис. 2.18, *б*. Представляет интерес частный случай, когда $P=(b_1, b_2, \ldots, b_m)$ и имеется m заданий с

*) Под ресурсом здесь понимается процессорное время. (*Прим. перев.*)

одинаковыми требованиями τ на ресурс. Чтобы все задания были завершены в одно и то же время, необходимо, чтобы они, в некотором смысле, выполнялись с одинаковой средней скоростью. Здесь снова может быть использован подход, показанный на рис. 2.18, *б*.

Мы начинаем обсуждать совместные возможности набора процессоров, поэтому введем $B_j = \sum_{i=1}^{j} b_i$ для всех j, $1 \leqslant j \leqslant m$.

Аналогично, положим $X_j = \sum_{i=1}^{j} \tau_i$ для всех j, $1 \leqslant j \leqslant m$.

Используя введенные обозначения, можно сказать, что алгоритм 2.4 из § 2.5, определявший расписания для независимых заданий, выполняемых на идентичных процессорах, строил расписание длиной $\omega = \max\{\tau_1, X_n/B_m\}$. Заметим, что X_n/B_m соответствует расписанию, в котором отсутствуют простои, и является нижней оценкой длины расписания. Однако, как показано на рис. 2.18, если процессоры идентичны, при выборе аналога величины τ_1 следует быть осторожным.

Рассмотрим систему $\mathcal{T} = (\tau_1, \tau_2, \ldots, \tau_n)$, которую нужно исполнить на наборе процессоров $P = (b_1, b_2, \ldots, b_m)$. Каждое задание в каждый момент времени может выполняться не более чем на одном процессоре, поэтому длина расписания будет не меньше τ_1/b_1 единиц времени. Задание T_1 на любом другом процессоре будет выполняться дольше. Если второй процессор имеет значительно меньшее быстродействие, чем первый, то отношение τ_2/b_2 может быть больше, чем τ_1/b_1. Как и в примере на рис. 2.18, мы можем усреднить выполнение двух заданий. Общий потребный ресурс есть $X_2 = \tau_1 + \tau_2$. Суммарное быстродействие двух процессоров есть $B_2 = b_1 + b_2$. Таким образом, длина усредненного расписания должна быть не менее X_2/B_2 единиц времени. В общем случае мы должны рассматривать $\max_j X_j/B_j$ в качестве нижней оценки длины расписания, поскольку τ_3/b_3 может быть больше, чем X_2/B_2, и так далее.

Алгоритм, который мы опишем, строит расписание, имеющее длину

$$\omega = \max\left\{ \max_{1 \leqslant j \leqslant m} X_j/B_j,\ X_n/B_m \right\}.$$

Из предыдущих рассуждений следует, что значение ω является оптимальным. Алгоритм представляет собой незначительную модификацию алгоритма 2.5 из § 2.6, который

строил расписание методом разделения процессоров для деревьев и m идентичных процессоров.

Алгоритм 2.6. Рассмотрим систему независимых заданий $\mathcal{T}=(\tau_1, \tau_2, \ldots, \tau_n)$, которые нужно распределить между процессорами $P=(b_1, b_2, \ldots, b_m)$. Напомним, что $\tau_1 \geqslant \tau_2 \geqslant \ldots \geqslant \tau_n$ и $b_1 \geqslant b_2 \geqslant \ldots \geqslant b_m$. Данный алгоритм определяет кратчайшее расписание с прерываниями для выполнения системы заданий $\mathcal{T}$ на процессорах P. Алгоритм может быть также использован для построения расписаний выполнения произвольных систем заданий.

1. Уровнем задания в произвольный момент времени является размер требования на ресурс у невыполненной части задания. Пусть j — количество неназначенных процессоров, а k — число заданий, имеющих наиболее высокий уровень. Если $k \leqslant j$, назначаем k заданий таким образом, чтобы они выполнялись с одинаковой скоростью на k наиболее быстрых процессорах. В противном случае назначим k заданий на j процессоров. Если остаются свободные процессоры, рассматриваем задания более низкого уровня и так далее.

2. Перераспределяем все процессоры среди невыполненных заданий при возникновении одиого из следующих событий:

Событие 1: завершено выполнение задаиия.

Событие 2: достигнута точка, в которой продолжение работы с прежним распределением процессоров приводит к тому, что задания более низкого уровня будут обслуживаться с большей скоростью, чем задания более высокого уровня.

3. Для того чтобы построить расписание с прерываниями из расписания с разделением процессоров, произведем переназначение заданий на интервалах, заключенных между каждой парой событий. Если k заданий разделяли между собой k процессоров, назначим задания таким образом, чтобы они выполнялись на k процессорах с той же средней скоростью (см. рис. 2.19).

Пусть k заданий разделяли j процессоров, $k>j$; обозначим через τ потребность каждого из k заданий в ресурсе на рассматриваемом интервале. Пусть b обозначает быстродействие самого медленного из j процессоров. Если τ/b меньше длины интервала, то задания можно назначать в соответствии с алгоритмом 2.4, не принимая в расчет разное быстродействие процессоров. В противном случае разделим интервал на k равных частей. Распределим k заданий таким образом, чтобы каждое из них встречалось ровно на j

интервалах каждый раз на разных процессорах. Необходимость распределения четырех заданий между тремя процессорами возникает на интервале (6, 10) на рис. 2.19. Этот рисунок дает пример расписания, построенного с помощью алгоритма 2.6.

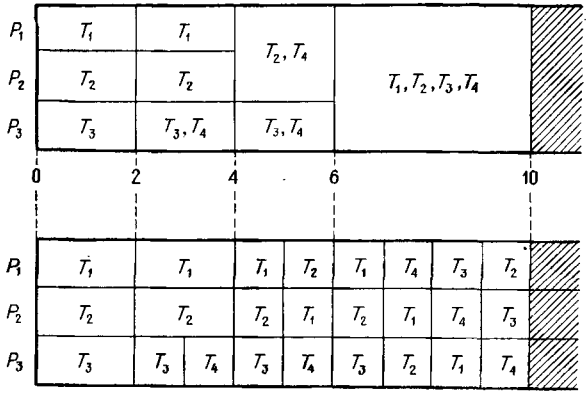


Рис. 2.19. Расписание для системы заданий $\mathscr{T}=(41, 25, 13, 11)$ и $P=(6, 2, 1)$.

Теорема 2.11. *Алгоритм* 2.6 *строит расписания минимальной длины с прерываниями для системы независимых заданий* $\mathscr{T}=(\tau_1, \tau_2, \ldots, \tau_n)$, *выполняемой набором процессоров* $P=(b_1, b_2, \ldots, b_m)$.

Доказательство. Если в расписании с разделением процессоров, построенном с помощью алгоритма, нет простоев, оно является оптимальным и его длина есть $\omega=X_n/b_m$.

Предположим теперь, что P_i — самый быстрый из процессоров, имеющих простои. В этом случае последнее задание, завершаемое на P_i, имеет более низкий уровень, чем задания, которые уже завершены на $P_1, P_2, \ldots, P_{i-1}$. Как следует из алгоритма, с течением времени уровень выполняемых заданий уменьшается. Следовательно, $P_1, P_2, \ldots, P_{i-1}$ выполняли $T_1, T_2, \ldots, T_{i-1}$ с самого начала расписания. Поскольку процессоры $P_1, P_2, \ldots, P_{i-1}$ не имеют простоев, задания $T_1, T_2, \ldots, T_{i-1}$ завершаются в одно и то же время. Отсюда следует, что длина расписания равна X_{i-1}/B_{i-1}. Заметим, что если какое-либо из заданий $T_1, T_2, \ldots, T_{i-1}$ будет назначено на более медленный, чем P_{i-1}, процессор, то длина расписания только увеличится.

Легко видеть, что расписание с разделением процессоров преобразуется в допустимое расписание с прерываниями. Поэтому алгоритм действительно строит оптимальное расписание с прерываниями для $\mathscr{T}$ на P.

Отметим, что алгоритм 2.6 может применяться также к деревьям и орграфам. Можно ожидать, что для двух процессоров он строит оптимальное расписание, поскольку простои могут возникать только на медленном процессоре. В случае деревьев и трех или более процессоров алгоритм 2.6 не оптимален. Рассмотрим дерево, в котором задания расположены на трех уровнях, как показано на рис. 2.20, а набор процессоров таков, что $b_1 = \sqrt{m} + 1$ и $b_i = 1$ для всех i, $2 \leqslant i \leqslant m$. Пусть имеется $\sqrt{m}$ заданий $V_1, V_2, \ldots, V_{\sqrt{m}}$, непосредственно предшествующих конечному заданию. На рис. 2.20 $m=9$, а значит, $\sqrt{m}=3$. Времена выполнения заданий выбраны так, чтобы расписания были такими, как на рис. 2.20. По мере увеличения m отношение длины уровневого расписания к длине оптимального расписания стремится к 2.

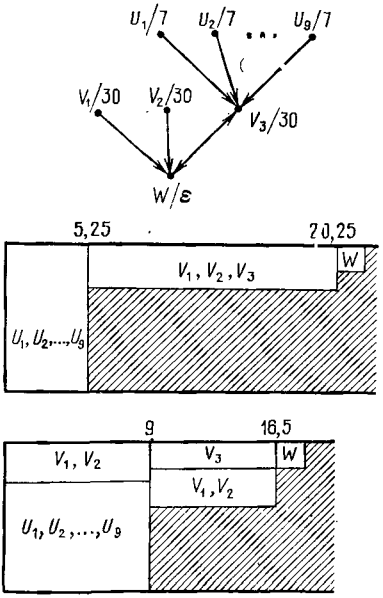


Рис. 2.20.

2.9. Конвейерная задача

Во время пребывания в вычислительной системе программа обрабатывается различными устройствами. Все программы проходят фазы ввода, исполнения и вывода; поэтому система заданий может быть представлена в виде цепочек, состоящих из m заданий, в которых i-е задание обрабатывается на устройстве P_i. Определение расписания минимальной длины в данной ситуации составляет предмет так называемой конвейерной задачи. Мы дадим алгоритм отыскания расписания минимальной длины для $m=2$. Для бо́льших значений m задача является NP-полной (см. гл. 1).

Для двухпроцессорной задачи рассмотрим систему заданий $(\mathscr{T}, <)$, состоящую из n цепочек $C_1, C_2, \ldots, C_n$, каждая из которых состоит из двух заданий A_i и B_i, $A_i < B_i$.

Задание A_i должно выполняться на процессоре P_1 в течение α_i единиц времени, а затем, после завершения A_i, на процессоре P_2 в течение β_i единиц времени должно выполняться B_i. Пример приведен на рис. 2.21.

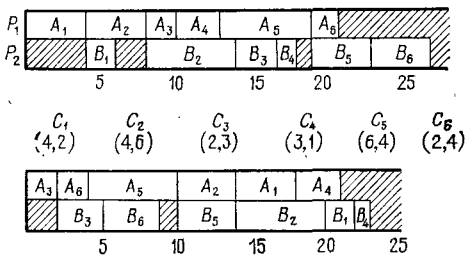


Рис. 2.21. Два конвейерных расписания для цепочек $C_1, C_2, \ldots, C_6$. Оптимальный порядок — 362514.

На этом рисунке задания $B_1, B_2, \ldots, B_n$ выполняются на P_2 в том же порядке, в котором $A_1, A_2, \ldots, A_n$ выполняются на P_1. Мы покажем, что и в общем случае можно ограничиться рассмотрением расписаний, в которых если A_i выполняется перед A_j, то B_i также предшествует B_j.

Пусть S — расписание на двух процессорах для набора цепочек $C_1, C_2, \ldots, C_n$. Скажем, что цепочка C_i *предшествует* цепочке C_j тогда и только тогда, когда A_i выполняется раньше, чем A_j и B_i — раньше, чем B_j.

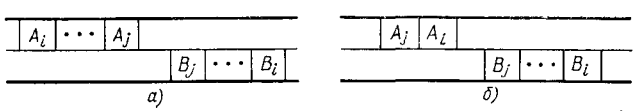


Рис. 2.22.

Лемма 2.8. *Пусть S — расписание на двух процессорах для набора цепочек $C_1, C_2, \ldots, C_n$; тогда можно построить расписание не длиннее, чем S, в котором для каждой пары цепочек C_i и C_j либо C_i предшествует C_j, либо C_j предшествует C_i.*

Доказательство. Предположим, что лемма неверна. Тогда существуют две цепочки C_i и C_j, подобные тем, что показаны на рис. 2.22, *а*, такие, что A_i выполняется перед A_j, но B_j выполняется перед B_i. Пусть B_j и B_i явля-

ются последними такими заданиями на P_2. Очевидно, мы можем отложить выполнение A_i так, чтобы оно непосредственно следовало после A_j; при этом все задания, находящиеся между A_i и A_j (включая A_j), будут выполнены на α_i единиц времени раньше, как показано на рис. 2.22, *б*. Преобразование расписания, при котором задания на процессоре P_1 будут выполнены раньше, не нарушает ограничений предшествования между заданиями в цепочке. Более того, A_i все еще выполняется перед B_i, и длина расписания не увеличилась. Доказательство леммы получается теперь индукцией по числу перестановок, необходимых для получения нужного расписания.

С помощью леммы 2.8 мы показали, что можно ограничиться расписаниями, в которых задания $A_1, A_2, \ldots, A_n$ на процессоре P_1 выполняются в том же порядке, что и задания $B_1, B_2, \ldots, B_n$ на процессоре P_2. Нам осталось лишь определить порядок, в соответствии с которым должны выполняться цепочки $C_1, C_2, \ldots, C_n$.

Для того чтобы наметить подход к решению задачи, рассмотрим две цепочки C_i и C_j. Если C_i выполняется перед

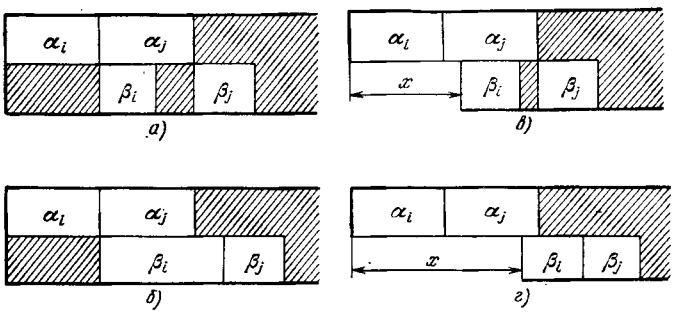


Рис. 2.23.

C_j, как показано на рис. 2.23, то длина промежутка времени ω_{ij}, в течение которого выполняется по крайней мере одна из цепочек C_i или C_j, есть

$$\omega_{ij} = \alpha_i + \max\{\alpha_j, \beta_i\} + \beta_j.$$

Аналогично, для ω_{ji}, имеющей тот же смысл для расписания, в котором C_j выполняется перед C_i, получаем

$$\omega_{ji} = \alpha_j + \max\{\alpha_i, \beta_j\} + \beta_i.$$

Следующая лемма показывает, что условием, при котором ω_{ji} короче, чем ω_{ij}, является $\min\{\alpha_j, \beta_i\} < \min\{\alpha_i, \beta_j\}$.

Лемма 2.9. *Пусть S — расписание на двух процессорах для набора цепочек $C_1, C_2, \ldots, C_n$. Предположим, что в S C_i выполняется непосредственно перед C_j. Если* $\min\{\alpha_j, \beta_i\} \leqslant \min\{\alpha_i, \beta_j\}$, *то расписание, полученное путем перемены порядка, в котором выполняются C_i и C_j, не длиннее S.*

Доказательство. Вначале предположим, что C_i и C_j — первые две цепочки, выполняемые в S. Процессор P_1 освобождается по истечении $\alpha_i+\alpha_j$ единиц времени, независимо от порядка, в котором выполняются C_i и C_j. Таким образом, нужно рассматривать только время, когда освободится процессор P_2.

Пусть ω_{ij} и ω_{ji} — величины, определенные выше. Мы должны показать, что $\omega_{ij} \geqslant \omega_{ji}$.

Из выражения $\min\{\alpha_j, \beta_i\} \leqslant \min\{\alpha_i, \beta_j\}$ получаем $\max\{-\alpha_j, -\beta_i\} \geqslant \max\{-\alpha_i, -\beta_j\}$. Прибавляя к обеим частям $\alpha_i+\alpha_j+\beta_i+\beta_j$, получаем $\alpha_i+\beta_j+\max\{\beta_i, \alpha_j\} \geqslant \alpha_j+\beta_i+\max\{\beta_j, \alpha_i\}$.

В левой и правой частях этого неравенства стоят выражения для ω_{ij} и ω_{ji}, откуда получаем $\omega_{ij} \geqslant \omega_{ji}$, что и требовалось доказать.

Если C_i и C_j не являются первыми цепочками в расписании, то, как показано на рис. 2.23, может существовать временной интервал x такой, что выполнение B_i не может начаться раньше, чем через x единиц времени после начала A_i. Если интервал x невелик, то B_j завершается через ω_{ij} единиц времени после того, как начнется A_i. Если интервал x большой, то B_j завершится через $x+\beta_i+\beta_j$ единиц времени.

Аналогично, если C_j выполняется перед C_i, то B_i заканчивается через $\max\{\omega_{ji}, x+\beta_i+\beta_j\}$ единиц. Таким образом, даже в том случае, когда две цепочки не являются первыми в расписании, перестановка C_j и C_i местами не может удлинить расписание.

Как и следовало ожидать, условие, полученное в лемме 2.9, является основой алгоритма определения расписаний минимальной длины.

Теорема 2.12. *Рассмотрим систему, состоящую из n цепочек $C=(A_i, B_i)$, $1 \leqslant i \leqslant n$, в двухпроцессорной конвейерной задаче. Пусть α_i и β_i — времена выполнения A_i и B_i на процессорах P_1 и P_2 соответственно. Если* $\min\{\alpha_i, \beta_j\} < \min\{\alpha_j, \beta_i\}$, *то можно построить расписание минимальной длины, в котором C_i выполняется перед C_j.*

Доказательство. Пусть S — кратчайшее расписание для данной системы. Из леммы 2.8 следует, что

можно без ограничения общности считать, что для всех цепочек C_i и C_j, если A_i выполняется перед A_j, то B_i выполняется перед B_j.

Если в S найдутся две соседние цепочки, для которых условия теоремы не выполняются, то мы можем на основании леммы 2.9 переставить их местами. Нам осталось просто показать, что перестановки приводят к искомому расписанию.

С помощью условия $\min\{\alpha_i, \beta_j\} < \min\{\alpha_j, \beta_i\}$ мы можем частично упорядочить цепочки. Зафиксировав правило разрешения конфликтов, можно получить полностью упорядоченную последовательность. Можно показать, что перестановка цепочек в соответствии с полным упорядочением превращает S в расписание, удовлетворяющее условиям теоремы.

Двухпроцессорная конвейерная задача, которую мы рассмотрели, не охватывает все три фазы исполнения программ, а именно: ввод, вычисления, вывод. Хотя трехпроцессорная конвейерная задача является NP-полной, существует частный случай, относительно которого можно сделать важные утверждения. Пусть теперь цепочка C_i состоит из трех задач, имеющих времена выполнения α_i, β_i и γ_i. Рассмотрим, например, систему ввода-вывода ($I/0$), в которой считается, что в процессе вывода имеется некоторое время простоя.

В частности, если $\min\{\alpha_i, \alpha_j\} \geqslant \max\{\beta_i, \beta_j\}$ или если $\min\{\gamma_i, \gamma_j\} \geqslant \max\{\beta_i, \beta_j\}$, то мы можем использовать условие

$$\min\{\alpha_i+\beta_i,\ \beta_j+\gamma_j\} < \min\{\alpha_j+\beta_j,\ \beta_i+\gamma_i\}$$

для того, чтобы поместить C_i перед C_j.

Экспериментальные исследования [28] показали, что это условие дает хорошие эвристики и для других систем.

Применение теоремы 2.12 к простой задаче конвейерного типа рассмотрено в п. 1.4.1.

Библиографическая справка

Уровневая стратегия, рассмотренная в § 2.2, разработана Т. Ху [65]. Алгоритмы определения кратчайших расписаний без прерываний на двух процессорах для UET-систем разработаны Фужи, Касами и Ниномия [43], Мураокой [116], а также Коффманом и Грэхемом [24]. В [43] для системы заданий построен неориентированный граф G. Присутствие ребра (T, T') в G означает, что ни одно из отношений $T \prec T'$ и $T' \prec T$ не выполнено. Расписание для двух процессоров определяется с помощью максимального паросочетания в G. Алгоритм не удается обобщить для получения эвристических правил в случае трех и более

процессоров. В [116] предлагается подход, при котором вначале рассматриваются «компактные» графы — системы заданий, в которых для каждого задания T сумма длины максимального пути, ведущего от начальной вершины к T, и уровня T является постоянной. Затем алгоритм распространяется на системы заданий общего вида. Описание довольно длинное. Алгоритм 2.2 изложен в соответствии с [24], так как процесс расстановки меток, приводящий к оптимальному расписанию для двух процессоров, может быть тотчас же превращен в эвристический алгоритм для большого числа процессоров.

Другое доказательство алгоритма Коффмана — Грэхема дано в [108]. Вообще говоря, расписания минимальной длины, построенные с помощью алгоритмов 2.1 и 2.2, не являются единственными. Попытка описать класс всех оптимальных расписаний на двух процессорах предпринята Шиндлером и Симонсмейером [137].

Расписания с прерываниями для независимых заданий, рассмотренные в § 2.5, излагаются по работам Мак-Нотона [109] и Роткопфа [119]. Понятия разделения процессора и уровневых расписаний развиты Мюнтцем и Коффманом в [110, 111], откуда взяты теоремы 2.7 и 2.8. Сравнение длин расписаний с прерываниями и без прерываний проведено в [24].

Задачи с процессорами, имеющими разное быстродействие, рассматривались разными авторами. Работа Байера [6] посвящена исследованию расписаний без прерываний для двух различных процессоров. Им получен алгоритм определения расписаний минимальной длины для UET-деревьев в случае, когда один процессор вдвое быстрее другого. Лю и Лю [101] предложили алгоритм определения расписания минимальной длины с прерываниями для независимых заданий, выполняемых на вычислительной системе, в которой один из процессоров — быстрый. Ими же получено выражение для минимальной длины расписания, использованное в теореме 2.11 (см. также [102]). Материал § 2.8 излагается в соответствии с [74].

Решение двухпроцессорной конвейерной задачи впервые получено Джоисоиом [83]; см. также работу Джексона [80].

ГЛАВА 3

КРИТЕРИЙ СРЕДНЕГО ВЗВЕШЕННОГО ВРЕМЕНИ ПРОХОЖДЕНИЯ

3.1. Введение

В этой главе мы рассмотрим задачи составления расписаний, в которых в качестве критерия стоимости выступает взвешенная сумма времен окончания всех обслуживаемых заданий, т. е. среднее взвешенное время завершения (прохождения) заданий *).

Мы рассмотрим три аспекта этой проблемы.

1. В первой модели рассматривается один процессор и система заданий, в которой с каждым заданием связано время обработки и стоимость его пребывания в системе. Мы представим эффективные алгоритмы определения оптимальных расписаний для частного случая, когда ограничения предшествования имеют вид леса.

2. Во второй модели вводится понятие *работы*, которая представляет собой упорядоченную последовательность заданий, генерируемую вероятностным методом. Задача заключается в том, чтобы составить расписание выполнения совокупности работ на одной машине. Времена выполнения и стоимости заданы в виде случайных величин, а критерием качества является математическое ожидание суммы взвешенных времен окончания всех заданий, побывавших на обслуживании.

3. В последней модели мы рассмотрим задачу при наличии более одной машины. Сами машины не обязательно являются одинаковыми. Мы положим значения стоимостей равными единице и покажем, как эту задачу построения расписания свести к транспортной задаче.

Вопросы сложности, относящиеся к перечисленным задачам, рассмотрены в гл. 1 и 4.

---

*) Заметим, что в отличие от определения $\bar{\omega}$, данного в гл. 1, взвешенная сумма не делится на количество заданий n. Легко видеть, что результаты, полученные для нашего более простого определения, применимы к определению $\bar{\omega}$ из гл. 1.

3.2. Модель

В задачу $\mathcal{P}$ составления расписания входят следующие компоненты.

1. Система заданий $(\mathcal{T}, <)$, где $\mathcal{T}$ есть индексированное множество из n заданий, $n \geqslant 0$, а $<$—отношение частичного порядка, заданное на $\mathcal{T}$. Набор индексов $\mathcal{T}$ обозначается $I(\mathcal{P})$. Для каждого j из $I(\mathcal{P})$ T_j есть элемент $\mathcal{T}$.

2. Положительное число τ_j, для каждого j из $I(\mathcal{P})$ обозначающее потребность задания T_j во времени обслуживания.

3. Действительное число w_j для каждого j из $I(\mathcal{P})$, называемое стоимостью пребывания T_j в системе. На числа w_j не накладываются никакие ограничения, в частности, они могут быть как положительными, так и отрицательными.

Поскольку мы рассматриваем лишь один процессор, можно ограничиться расписаниями, задаваемыми перестановкой индексов заданий. Говорят, что перестановка $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_n$ элементов $I(\mathcal{P})$ *совместима* с $<$ (или с $\mathcal{P}$), если $T_j < T_{j'}$ влечет $k < k'$, где $\alpha_k = j$ и $\alpha_{k'} = j'$.

Перестановка $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_n$ элементов $I(\mathcal{P})$ определяет расписание естественным образом: задание T_{α_1} выполняется первым, задание T_{α_2}—вторым, T_{α_3}—третьим и так далее. Среднее взвешенное время завершения для расписания, определяемого перестановкой α, дается выражением

$$\mathrm{mwft}(\alpha) = \sum_{j=1}^{n} w_{\alpha_j} \left(\sum_{i=1}^{j} \tau_{\alpha_i} \right).$$

Перестановка α (последовательность, расписание) называется *оптимальной*, если α совместима с $\mathcal{P}$ и при этом достигается минимум mwft среди всех перестановок, совместимых с $\mathcal{P}$.

При отсутствии ограничений предшествования можно легко построить оптимальное расписание. Пусть $\rho_j = w_j/\tau_j$ для всех j из $I(\mathcal{P})$.

Теорема 3.1 [136]. *Пусть $\mathcal{P}$—задача упорядочения с $< = \varnothing$. Тогда перестановка $\alpha = \alpha_1, \ldots, \alpha_n$ оптимальна по отношению к $\mathcal{P}$ в том и только в том случае, когда $\rho_{\alpha_1} \geqslant \rho_{\alpha_2} \geqslant \ldots \geqslant \rho_{\alpha_n}$.*

Доказательство. Данный результат вытекает из рассмотрения произвольной перестановки $\beta = \beta_1, \ldots, \beta_n$ и подсчета изменения mwft при перестановке местами двух соседних элементов β_j и β_{j+1}. Оказывается, это приводит

к уменьшению mwft тогда и только тогда, когда $\rho_{\beta_j} < \rho_{\beta_{j+1}}$, и не влияет на результат тогда и только тогда, когда $\rho_{\beta_j} = \rho_{\beta_{j+1}}$. Отсюда вытекает утверждение теоремы. Более детальное доказательство приведено в § 1.3.

Идея об использовании отношений ρ_j может быть распространена на случай, когда $\prec$ является произвольным отношением предшествования. Это приводит к интересному описанию оптимальных расписаний [134, 135] и к эффективным алгоритмам построения оптимальных расписаний для задач с некоторыми видами ограничений предшествования [71, 134, 135].

В следующем параграфе мы определим функцию ρ, которая является обобщением отношений ρ_j. Функция ρ задается на непустых подмножествах индексов заданий и используется для введения упорядочения на множестве наборов заданий. Мы введем также понятия начального множества и ρ-максимального множества индексов заданий. ρ-максимальные множества играют важную роль, поскольку, как показано в следующем параграфе, индексы в таком множестве должны появляться в виде непрерывной последовательности во всех оптимальных перестановках. Наконец, мы дадим «двойственную» формулировку в терминах конечных множеств и ρ-минимальных множеств, которая используется при доказательстве оптимальности алгоритмов составления расписаний.

3.3. Предварительные замечания

В данном параграфе мы дадим более общее определение отношениям ρ_j, введенным в предыдущем параграфе, задав их на непустых подмножествах множества индексов.

Пусть $U \subseteq I(\mathcal{P})$ и $U \neq \varnothing$. Определим $\tau(U)$, $w(U)$ и $\rho(U)$ следующим образом:

$$\tau(U) = \sum_{j \in U} \tau_j, \quad w(U) = \sum_{i \in U} w_j,$$

$$\rho(U) = \frac{w(U)}{\tau(U)}.$$

Набор $U \subseteq I(\mathcal{P})$ называется *начальным множеством* по отношению к $\mathcal{P}$, если выполняются условия:

1) $U \neq \varnothing$;

2) если $T_j \prec T_{j'}$, то $j \notin U$ влечет $j' \notin U$.

Например, для системы заданий, изображенной на рис. 3.1, начальными множествами среди других являются $\{1\}$, $\{1, 2, 3\}$, $\{1, 2, 4\}$ и $\{1, 2, 3, 4, 6\}$. Пусть $\mathcal{Y}$ обозна-

чает класс начальных множеств (по отношению к $\mathscr{P}$). Определим число ρ^* как

$$\rho^* = \max_{U \in \mathscr{Y}} \rho(U).$$

Мы говорим, что множество $U \subseteq I(\mathscr{P})$ является *ρ-максимальным множеством* по отношению к $\mathscr{P}$, если выполняются условия:

1) $U \in \mathscr{Y}$;
2) $\rho(U) = \rho^*$;
3) если $V \in \mathscr{Y}$; $\rho(V) = \rho^*$ и $V \subseteq U$, то $V = U$.

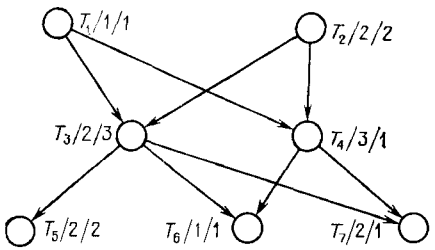


Рис. 3.1. Задача составления расписания. Обозначения: $T_j/\tau_j/\omega_j$.

Значение ρ^* для задачи, изображенной на рис. 3.1, составляет 6/5 и достигается на $U = \{1, 2, 3\}$; U является ρ-максимальным множеством.

Очень часто в задаче упорядочения $\mathscr{P}$ нам приходится ограничиваться рассмотрением подмножества заданий и вычислять ρ-максимальные множества с учетом этого ограничения. Положим $U \subseteq I(\mathscr{P})$ и определим $\mathscr{T}/U$ и $</U$ как

$$\mathscr{T}/U = \{T_j \downarrow T_j \in \mathscr{T}, \quad j \in U\},$$
$$</U = \{(T_i, T_j) \downarrow (T_i, T_j) \in <, \quad i \in U, \quad j \in U\}.$$

Задача упорядочения $\mathscr{P}/U$ получается из $\mathscr{P}$, если ограничиться системой заданий $(\mathscr{T}/U, </U)$, где $(\mathscr{T}, <)$ есть система заданий для $\mathscr{P}$.

В связи с этим мы можем также рассмотреть перестановки подмножеств множества $I(\mathscr{P})$. Пусть $U \subseteq I(\mathscr{P})$ и $\alpha = \alpha_1, \alpha_2, \ldots, \alpha_k$ есть перестановка подмножества множества $I(\mathscr{P})$. Тогда α/U будет обозначать перестановку подмножества множества U, образованного из тех элементов α, которые соответствуют элементам U. Например, если $U = \{2, 3, 4, 5\}$ и $\alpha = 1, 2, 8, 5, 6, 3$, то $\alpha/U = 2, 5, 3$. Пусть λ обозначает пустую перестановку; тем самым для произвольной перестановки α имеем $\alpha/\varnothing = \lambda$.

Ограничение перестановок является коммутативным. Так, если U и V — произвольные подмножества множества $I(\mathcal{P})$ и α — произвольная перестановка подмножества элементов $I(\mathcal{P})$, то $\alpha/U/V = \alpha/V/U = \alpha/U \cap V$.

Пусть α и β — две перестановки. Тогда композицией α, β перестановок α и β называется последовательность элементов α, следом за которыми стоят элементы β. Предположим, что $\alpha=2, 3, 7$ и $\beta=4, 5, 6, 1$; тогда $\alpha, \beta=2, 3, 7, 4, 5, 6, 1, \beta$, $\alpha=4, 5, 6, 1, 2, 3, 7$ и $\alpha, \lambda=\lambda, \alpha=2, 3, 7$.

В выражениях вида $\alpha/U, \beta/V$ оператор $/$ выполняется раньше композиций перестановок, так что $\alpha/U, \beta/V=\alpha/U, (\beta/V)$.

Теперь мы дадим ряд лемм, описывающих свойства функции ρ и ρ-максимальных множеств, которые потребуются для дальнейшего исследования.

Если $U \subseteq I(\mathcal{P})$, то обозначим через $\bar{U}$ множество $I(\mathcal{P})-U$. Если α — перестановка, совместимая с $\mathcal{P}$, то $\mathrm{mwft}(\alpha/U)$ есть среднее взвешенное время прохождения α/U относительно $\mathcal{P}/U$.

Лемма 3.1. *Пусть $\mathcal{P}$ — задача упорядочения и α — перестановка $I(\mathcal{P})$, совместимая с $\mathcal{P}$. Если U — такое подмножество $I(\mathcal{P})$, что $\alpha=\alpha/U, \alpha/\bar{U}$, то*

$$\mathrm{mwft}(\alpha) = \mathrm{mwft}(\alpha/U) + \mathrm{mwft}(\alpha/\bar{U}) + w(\bar{U})\,\tau(U).$$

Доказательство. Положим $k=|U|$. Имеем

$$\begin{aligned}
\mathrm{mwft}(\alpha) &= \sum_{j=1}^{n} w_{\alpha_j}\left(\sum_{i=1}^{j}\tau_{\alpha_i}\right) = \\
&= \sum_{j=1}^{k} w_{\alpha_j}\left(\sum_{i=1}^{j}\tau_{\alpha_i}\right) + \sum_{j=k+1}^{n} w_{\alpha_j}\left(\sum_{i=1}^{j}\tau_{\alpha_i}\right) = \\
&= \mathrm{mwft}(\alpha/U) + \sum_{j=k+1}^{n} w_{\alpha_j}\left(\sum_{i=k+1}^{j}\tau_{\alpha_i} + \sum_{i=1}^{k}\tau_{\alpha_i}\right) = \\
&= \mathrm{mwft}(\alpha/U) + \mathrm{mwft}(\alpha/\bar{U}) + w(\bar{U})\,\tau(U).
\end{aligned}$$

Из леммы 3.1 следует, что если α оптимальна относительно $\mathcal{P}$ и $\alpha=\alpha/U, \alpha/\bar{U}$, то α/U оптимальна относительно $\mathcal{P}/U$, а $\alpha/\bar{U}$ оптимальна относительно $\mathcal{P}/\bar{U}$. Обратное не обязательно верно.

Лемма 3.2. *Пусть $\mathcal{P}$ — задача упорядочения, U, U_1 и U_2 — подмножества $I(\mathcal{P})$ такие, что $U=U_1 \cap U_2$, $U_1 \cap U_2=\varnothing$, $U_1 \neq \varnothing$ и $U_2 \neq \varnothing$. Тогда*

$$\rho(U) = \frac{\tau(U_1)}{\tau(U)}\,\rho(U_1) + \frac{\tau(U_2)}{\tau(U)}\,\rho(U_2).$$

Доказательство. Имеем

$$\rho(U)=\frac{w(U)}{\tau(U)}=\frac{w(U_1)+w(U_2)}{\tau(U)}=\frac{\tau(U_1)}{\tau(U)}\rho(U)+\frac{\tau(U_2)}{\tau(U)}\rho(U_2).$$

Лемма 3.2 утверждает, что $\rho(U)$ представляет собой выпуклую комбинацию $\rho(U_1)$ и $\rho(U_2)$, т. е. $\rho(U)=\eta\rho(U_2)+(1-\eta)\rho(U_2)$, где $0\leqslant\eta\leqslant 1$. Поэтому, если $\rho(U_1)\leqslant\rho(U_2)$, то $\rho(U_1)\leqslant\rho(U)\leqslant\rho(U_2)$; если $\rho(U_2)\leqslant\rho(U_1)$, то $\rho(U_2)\leqslant\rho(U)\leqslant\rho(U_1)$; наконец, если $\rho(U_1)=\rho(U_2)$, то $\rho(U)=\rho(U_1)$.

Лемма 3.3. (Перестановка соседних элементов.) *Пусть $\mathcal{P}$ — задача упорядочения, а α и β — две перестановки $I(\mathcal{P})$, совместимые с $\mathcal{P}$, такие, что $\alpha=\alpha/U,\alpha/U_1,\alpha/U_2,\alpha/W$ и $\beta=\alpha/U,\alpha/V_2,\alpha/V_1,\alpha/W$, где U, V_1, V_2 и W — непересекающиеся множества, $V_1\neq\varnothing$ и $V_2\neq\varnothing$. Тогда* $\mathrm{mwft}(\alpha)\leqslant\mathrm{mwft}(\beta)$ *в том и только в том случае, когда* $\rho(V_1)\geqslant\rho(V_2)$.

Доказательство. Легко выразить $\mathrm{mwft}(\beta)$ через $\mathrm{mwft}(\alpha)$, а именно:

$$\mathrm{mwft}(\beta)=\mathrm{mwft}(\alpha)+w(V_1)\tau(V_2)-w(V_2)\tau(V_1)=$$
$$=\mathrm{mwft}(\alpha)+\tau(V_2)\tau(V_1)[\rho(V_1)-\rho(V_2)].$$

Отсюда получаем

$$\mathrm{mwft}(\beta)-\mathrm{mwft}(\alpha)=\tau(V_2)\tau(V_1)[\rho(V_1)-\rho(V_2)],$$

что доказывает лемму, так как $\tau(V_2)\tau(V_1)>0$.

Лемма 3.4. *Пусть U и V — два различных ρ-максимальных множества относительно $\mathcal{P}$. Тогда $U\cap V=\varnothing$.*

Доказательство. Пусть $W=U\cap V$ и при этом $W\neq\varnothing$. W является начальным множеством, и, поскольку U является ρ-максимальным, мы заключаем на основании леммы 3.2, что $\rho(W)<\rho^*$. Следовательно, в силу леммы 3.2 $\rho(U-W)>\rho^*$. Еще раз используя лемму 3.2, можем записать $\rho(U\cup V)=\eta\rho(U-W)+(1-\eta)\rho(V)$, где $0<\eta<1$, откуда следует, что $\rho(U\cup V)>\rho^*$. Получили противоречие, так как $U\cup V$ — начальное множество.

Следующие леммы показывают, что изучение ρ-максимальных множеств позволяет глубже понять структуру оптимального расписания.

Лемма 3.5. *Если U является ρ-максимальным множеством относительно $\mathcal{P}$, то U содержится в виде непрерывной последовательности во всех оптимальных перестановках для $\mathcal{P}$.*

Доказательство. Вначале покажем, что U должно появиться в виде непрерывной последовательности во всех оптимальных перестановках для некоторой системы

$\mathcal{P}'$, выбранной так, что оптимальные перестановки относительно $\mathcal{P}$ образуют подмножество оптимальных перестановок относительно $\mathcal{P}'$. Из этого будет вытекать доказательство леммы.

Положим

$$\prec' = \prec - \{(T_i, T_j) \mid (T_i, T_j) \in \prec,\ T_i \in U,\ T_j \notin U\}.$$

Определение $\mathcal{P}'$ точно такое же, как $\mathcal{P}$, с той разницей, что $\prec$ заменяется на $\prec'$. Ясно, что все перестановки, совместимые с $\mathcal{P}$, являются совместимыми с $\mathcal{P}'$. Позже мы покажем, что существует оптимальная перестановка относительно $\mathcal{P}'$, которая совместима с $\mathcal{P}$. Таким образом, все оптимальные перестановки для $\mathcal{P}$ будут оптимальными для $\mathcal{P}'$.

Пусть α — оптимальная перестановка относительно $\mathcal{P}'$. Покажем, что U появляется в виде непрерывной последовательности в α. Предположим обратное, т. е. что α имеет вид

$$\alpha = \alpha/V_1, \alpha/U_1, \alpha/V_2, \alpha/U_2, \ldots, \alpha/V_r, \alpha/U_r, \alpha/V_{r+1},$$

где все множества V_i, U_i попарно не пересекаются, $r>1$, все непусты, за исключением, возможно, V_1 и V_{r+1}, и при этом $U = \bigcup_{i=1}^{r} U_i$.

Положим $k = \min \{j \mid \rho(U_j) \geqslant \rho(U_i)$ для всех $1 \leqslant i \leqslant r\}$. Величина k должна быть больше 1, так как, в противном случае, на основании леммы 3.2 мы получим противоречие с ρ-максимальностью U. Таким образом, мы можем записать $\rho(U_{k-1}) < \rho(U_k)$. Более того, $\rho(V_k) \geqslant \rho(U_k)$, поскольку если предположить обратное, то с помощью леммы 3.3 можно показать, что α не оптимальна относительно $\mathcal{P}$. Принимая во внимание эти обстоятельства и учитывая лемму 3.3, можно поменять местами α/U_{k-1} и α/V_k в α, получив при этом новую перестановку α', такую, что $\mathrm{mwft}(\alpha') < \mathrm{mwft}(\alpha)$. Это противоречит оптимальности α относительно $\mathcal{P}'$, и мы можем сделать вывод о том, что U должно появиться в виде непрерывной последовательности во всех оптимальных перестановках для $\mathcal{P}'$.

Остается показать, что существует оптимальная перестановка в $\mathcal{P}'$, которая является оптимальной также и для $\mathcal{P}$. Пусть α есть оптимальная перестановка относительно $\mathcal{P}'$ и $\alpha = \alpha/V, \alpha/U, \alpha/W$. Если $V = \varnothing$, то утверждение доказано. Поэтому предположим, что $V \neq \varnothing$. Ясно, что $\alpha' = \alpha/U, \alpha/V, \alpha/W$ совместима с $\mathcal{P}$ и $\mathcal{P}'$. Из леммы 3.3 и оптимальности α мы знаем, что $\rho(V) \geqslant \rho(U)$. Будем

считать, что $\rho(V) > \rho(U)$. На основании леммы 3.2 имеем $\rho(U \cup V) > \rho(U)$. Но это невозможно, поскольку $U \cup V$ является начальным множеством относительно $\mathcal{P}$. Следовательно, $\rho(V) = \rho(U)$ и согласно лемме 3.3 $\mathrm{mwft}(\alpha') = \mathrm{mwft}(\alpha)$.

Лемма 3.6. *Если множество U ρ-максимально относительно $\mathcal{P}$, то существует оптимальная перестановка α для $\mathcal{P}$ такая, что $\alpha = \alpha/U, \alpha/\bar{U}$.*

Доказательство. См. доказательство леммы 3.5.

Лемма 3.7. *Если перестановка α оптимальна относительно $\mathcal{P}$, то α может быть представлена в виде $\alpha = \alpha/U, \alpha/\bar{U}$, где U — некоторое множество, ρ-максимальное относительно $\mathcal{P}$.*

Доказательство. Пусть U есть ρ-максимальное множество, которое раньше всех встречается в α. Тогда на основании леммы 3.5 мы можем записать α как α/V, α/U, α/W. Если V непусто, то оно является начальным множеством, а значит, $\rho(V) \leqslant \rho(U)$. Используя лемму 3.3, можно исключить случай $\rho(V) < \rho(U)$. Следовательно, $\rho(V) = \rho(U)$ и V либо само ρ-максимально, либо содержит ρ-максимальное множество. Это противоречит нашему выбору U; следовательно, $V = \varnothing$.

Результаты этого параграфа могут быть представлены в двойственной форме следующим образом.

Множество $U \subseteq I(\mathcal{P})$ называется *конечным* множеством относительно $\mathcal{P}$, если выполняются условия:

1) $U \neq \varnothing$;

2) если $T_j < T_{j'}$, то $j \notin U$.

Пусть $\mathcal{F}$ означает класс конечных множеств относительно $\mathcal{P}$. Определим ρ_* как

$$\rho_* = \min_{U \in \mathcal{F}} \rho(U).$$

Скажем, что U является *ρ-минимальным* множеством относительно $\mathcal{P}$, если выполняются условия:

1) $U \in \mathcal{F}$;

2) $\rho(U) = \rho^*$;

3) если $V \in \mathcal{F}$, $\rho(V) = \rho^*$ и $V \subseteq U$, то $V = U$.

Все леммы данного параграфа, относящиеся к ρ-максимальным множествам, имеют соответствующие двойственные формулировки в терминах ρ-минимальных множеств. В дополнение к ним имеем следующую лемму.

Лемма 3.8. *Пусть U — ρ-максимальное, а V — ρ-минимальное множество относительно $\mathcal{P}$. Если $U \neq V$, то $U \cap V = \varnothing$.*

Доказательство. Пусть $W = U \cap V$. Предположим, что $W \neq \varnothing$. Тогда

$$\rho(U) = \eta\rho(U - W) + (1 - \eta)\rho(W),$$

где $0 < \eta < 1$, и

$$\rho(V) = \gamma\rho(V - W) + (1 - \gamma)\rho(W),$$

где $0 < \gamma < 1$. Поскольку W содержится как в начальном, так и в конечном множествах, множества $U - W$ и $V - W$ являются, соответственно, начальным и конечным. Отсюда получим $\rho(U - W) < \rho^*$, $\rho(V - W) > \rho_*$. Таким образом, $\rho(W) > \rho^* \geqslant \rho_* > \rho(W)$. Мы вынуждены заключить, что $W = \varnothing$. При доказательстве мы предполагали, что как $U - V$, так и $V - U$ непусты. Доказательство проходит и в том случае, когда одно из них пусто.

3.4. Алгоритмы

Отношение предшествования $\prec$ называется *входящим лесом*, если каждая вершина в графе предшествования имеет не более одного непосредственного преемника. Аналогично, $\prec$ является *выходящим лесом*, если каждая вершина в графе предшествования имеет не более одного непосредственного предшественника. В данном параграфе мы рассмотрим эффективные алгоритмы отыскания оптимальных расписаний для задач, в которых ограничения предшествования являются входящими либо выходящими лесами. Вначале рассматривается задача с входящим лесом, а затем для получения алгоритмов составления расписаний в случае выходящего леса используются соображения двойственности. Мы также представим алгоритм для случая, когда задача составления расписания может быть разделена на две или более «независимых» задачи.

Пусть $\mathcal{P}$—задача составления расписания такая, что $\prec$ является входящим лесом. С помощью леммы 3.2 и определения ρ-максимального множества можно показать, что для нахождения ρ-максимального множества относительно $\mathcal{P}$ нет необходимости рассматривать все начальные множества $\mathcal{P}$; в действительности нам нужно рассмотреть только n из них, где n есть число заданий.

Обозначим через H_j множество, состоящее из индекса j и индексов i всех заданий, для которых $T_i \prec T_j$.

Лемма 3.9. *Пусть $\mathcal{P}$—задача составления расписания для входящего леса и U—ρ-максимальное множество*

относительно $\mathcal{P}$. *Тогда существует индекс* $j \in I(\mathcal{P})$ *такой, что* $U = H_j$.

Доказательство. Пусть l есть индекс из U, соответствующий такому заданию T_l, что индексы преемников T_l не входят в U. Мы утверждаем, что $U = H_l$. Очевидно, $H_l \subseteq U$. Пусть $V = U - H_l$, и предположим, что $V \neq \varnothing$. Утверждается, что V — начальное множество. Допустим обратное. Тогда $H_l < V$*). Но это невозможно, потому что в U нет преемников T_l, а также потому, что, как следует из свойства входящего леса и определения H_l, ни один из элементов H_l не имеет преемников в U. Следовательно, V является начальным множеством и U представляет собой объединение двух непересекающихся начальных множеств. Это противоречит ρ-максимальности U, а значит, $V = \varnothing$.

Неформально говоря, построение оптимальной перестановки в задаче $\mathcal{P}$ составления расписания для входящего леса может быть осуществлено следующим образом. Каждому заданию T_j ставится в соответствие число $\rho(H_j)$, а затем определяется индекс l такой, что $\rho(H_l) \geqslant \rho(H_j)$ для $j \in I(\mathcal{P})$, причем $\rho(H_l) > \rho(H_j)$ для всех индексов $j \neq l$. Поскольку T_l является предшественником всех заданий, индексы которых входят в H_l, существует оптимальная относительно $\mathcal{P}$ перестановка α, такая, что

$$\alpha = \alpha/(H_l - \{l\}), l, \alpha/(I(\mathcal{P}) - H_l).$$

Таким образом, наша задача может быть «разложена» на две меньшие, а именно: $\mathcal{P}/(H_l - \{l\})$ и $\mathcal{P}/(I(\mathcal{P}) - H_l)$.

Алгоритм 3.1.

Вход: $\mathcal{P}$ — задача составления расписания для входящего леса.

Выход: оптимальное расписание для $\mathcal{P}$.

Метод:

procedure TRF($\mathcal{P}$)
begin
if $\mathcal{P}$ пусто **then return** λ
else
begin
Пусть l есть индекс задания из $\mathcal{P}$ такой, что H_l является ρ-максимальным множеством для $\mathcal{P}$
$U := H_l - \{l\}$; $V := I(\mathcal{P}) - H_l$

---

*) Пусть U и V — подмножества $I(\mathcal{P})$. Мы записываем $U < V$, если существует пара заданий T_i и T_j такая, что $T_i < T_j$, $i \in U$ и $j \in V$.

return TRF ($\mathcal{P}/U$),l,TRF ($\mathcal{P}/V$)
end
end TRF

Процедура TRF является рекурсивной, она имеет в качестве единственного входного параметра задачу упорядочения; результатом ее выполнения оказывается оптимальная перестановка по отношению к входной задаче. Выполнение оператора **return** ⟨выражение⟩, встречающегося в процедуре, заключается в вычислении ⟨выражения⟩ и возврате в точку вызова процедуры с вычисленным значением в качестве результата. Результат выполнения процедуры TRF, вообще говоря, не единствен, потому что может существовать более одного ρ-максимального множества. Однако независимо от выбора такого множества TRF дает оптимальную перестановку.

Теорема 3.2. *Пусть $\mathcal{P}$—задача упорядочения для входящего леса. Перестановка α оптимальна для $\mathcal{P}$ в том и только в том случае, когда она может быть получена в результате применения процедуры* TRF *к $\mathcal{P}$.*

Доказательство. Проведем доказательство методом индукции по n, где n—число заданий в $\mathcal{P}$. Справедливость теоремы для $n=1$ очевидна. Пусть n—целое число, большее 1. Предположим, что теорема верна для всех задач составления расписаний, количество заданий в которых меньше n. Пусть $\mathcal{P}$—задача, содержащая n заданий, α—перестановка, полученная в результате применения процедуры TRF. Мы можем представить α в виде $\alpha=\alpha/U,l,\alpha/V$, где U, V и l определены с помощью TRF, а $U\cup\{l\}$ есть ρ-максимальное множество. Из леммы 3.6 следует, что существует оптимальная перестановка $\beta=\beta/(U\cup\{l\}),\beta/V$. Поскольку T_l является преемником всех заданий, индексы которых находятся в U, мы можем записать $\beta=\beta/U,l,\beta/V$, где β/U—оптимальная перестановка для $\mathcal{P}/U$, а β/V—оптимальная перестановка для $\mathcal{P}/V$. По предположению индукции α/U и α/V являются оптимальными перестановками для $\mathcal{P}/U$ и $\mathcal{P}/V$ соответственно. Поэтому α есть оптимальная перестановка для $\mathcal{P}$.

Пусть β—произвольная оптимальная перестановка для $\mathcal{P}$. На основании леммы 3.7 мы можем записать β в виде $\beta=\beta/U,\beta/\bar{U}$, где U является ρ-максимальным множеством относительно $\mathcal{P}$. Из леммы 3.9 следует, что существует индекс l такой, что $H_l=U$. Поэтому TRF может сгенерировать перестановку α вида TRF ($\mathcal{P}/U-\{l\}$),l,TRF ($\mathcal{P}/\bar{U}$). Очевидно, β может быть записана в виде $\beta=\beta/(U-\{l\})$,

$l, \beta/\bar{U}$. Теперь, используя предположение индукции, мы заключаем, что процедура TRF строит оптимальную перестановку β.

Легко посчитать асимптотическую оценку времени счета при разумном применении TRF. Пусть T_j—задание из $\mathcal{P}$, $T_{i_1}, \ldots, T_{i_k}$—его непосредственные предшественники. Имеем

$$\rho(H_j) = \frac{w(H_j)}{\tau(H_j)} = \frac{w_j + \sum_{p=1}^{k} w(H_{i_p})}{\tau_j + \sum_{p=1}^{k} \tau(H_{i_p})}.$$

Мы можем проходить граф предшествования $<$ таким образом, чтобы для каждого задания T_j величины w и τ для множеств $H_{i_1}, \ldots, H_{i_k}$ вычислялись до посещения T_j [87]. Поэтому для всех заданий в $\mathcal{P}$ величины $\rho(H_j)$ могут быть вычислены за время $O(n)$. Эта процедура определяет положение по крайней мере одного задания в оптимальной последовательности, и, поскольку потребуется не более $O(n)$ времени, чтобы определить положение каждого из оставшихся заданий, временная сложность процедуры TRF в худшем случае определяется как $O(n^2)$.

Пусть теперь $\mathcal{P}$—задача составления расписания для выходящего леса и K_j есть множество, состоящее из индекса j и всех индексов i таких, что $T_j < T_i$. Следующая лемма показывает, что задачи составления расписаний для выходящих лесов являются двойственными по отношению к задачам для входящих лесов.

Лемма 3.10. *Пусть $\mathcal{P}$—задача составления расписания для выходящего леса, U—множество, ρ-минимальное относительно $\mathcal{P}$. Тогда $U = K_j$ для некоторого $j \in I(\mathcal{P})$.*

Доказательство. Двойственно доказательству, данному для леммы 3.9.

Алгоритм 3.2.

Вход: $\mathcal{P}$—задача составления расписания для выходящего леса.

Выход: оптимальное расписание для $\mathcal{P}$.

Метод:

procedure IRF($\mathcal{P}$)
begin
if $\mathcal{P}$ пусто **return** λ
else
begin

Пусть l есть индекс задания из $\mathcal{P}$ такой, что K_l является ρ-минимальным множеством для $\mathcal{P}$

$U := K_l - \{l\}$; $V := I(\mathcal{P}) - K_l$
return IRF$(\mathcal{P}/V)$, l, IRF$(\mathcal{P}/U)$
end
end IRF

Теорема 3.3. *Пусть $\mathcal{P}$ — задача составления расписания для выходящего леса. Перестановка α оптимальна для $\mathcal{P}$ в том и только в том случае, когда она может быть получена в результате применения процедуры* IRF *к $\mathcal{P}$.*

Доказательство. Двойственно доказательству теоремы 3.2.

Рассмотрим задачу $\mathcal{P}$ составления расписания и предположим, что существует разбиение $I(\mathcal{P})$ на два непустых множества U_1 и U_2 таких, что $\prec = \prec/U_1 \cup \prec/U_2$. Тогда мы будем писать $\mathcal{P} = \mathcal{P}_1 \| \mathcal{P}_2$, где $\mathcal{P}_i = \mathcal{P}/U_i$, $i = 1, 2$. Неформально говоря, систему $\mathcal{P}$ можно рассматривать как состоящую из двух независимых (параллельных) систем заданий $\mathcal{P}_1$ и $\mathcal{P}_2$.

Следующие леммы показывают некоторые важные свойства такой декомпозиции и образуют основу для доказательства корректности последующего алгоритма.

Лемма 3.11. *Пусть $\mathcal{P}$ — задача составления расписания такая, что $\mathcal{P} = \mathcal{P}_1 \| \mathcal{P}_2$. Если множество U ρ-максимально (минимально) относительно $\mathcal{P}$, то U является ρ-максимальным (минимальным) либо для $\mathcal{P}_1$, либо для $\mathcal{P}_2$. Более того, если V_1 и V_2 — ρ-максимальные (минимальные) множества для $\mathcal{P}_1$ и $\mathcal{P}_2$ соответственно, то по крайней мере одно из них является ρ-максимальным (минимальным) множеством для $\mathcal{P}$.*

Доказательство. Эту лемму легко доказать на основе леммы 3.2.

Лемма 3.12. *Пусть $\mathcal{P}$ — задача составления расписания такая, что $\mathcal{P} = \mathcal{P}_1 \| \mathcal{P}_2$, U_1 и U_2 — множества индексов для $\mathcal{P}_1$ и $\mathcal{P}_2$ соответственно. Если перестановка α оптимальна относительно $\mathcal{P}$, то α/U_1 и α/U_2 оптимальны относительно $\mathcal{P}_1$ и $\mathcal{P}_2$ соответственно. Более того, если α_1 и α_2 оптимальны для $\mathcal{P}_1$ и $\mathcal{P}_2$, то существует оптимальная относительно $\mathcal{P}$ перестановка α такая, что $\alpha/U_1 = \alpha_1$ и $\alpha/U_2 = \alpha_2$.*

Доказательство. Доказательство проведем методом индукции по n — числу заданий в $\mathcal{P}$. Для $n = 1$ утвержде-

ние, очевидно, верно. Пусть $n > 1$ и α—перестановка, оптимальная относительно $\mathcal{P}$. На основании леммы 3.7 мы можем представить α в виде $\alpha = \alpha/U, \alpha/\overline{U}$, где U — множество, ρ-максимальное для $\mathcal{P}$ и, в силу предыдущей леммы, ρ-максимальное либо для $\mathcal{P}_1$, либо для $\mathcal{P}_2$ (скажем, для $\mathcal{P}_1$). По предположению индукции α/U_2 оптимальна относительно $\mathcal{P}_2$, а $\alpha/(U_1 - U)$—относительно $\mathcal{P}_1/(U_1 - U)$. На основании леммы 3.6 теперь заключаем, что последовательность $\alpha/U, \alpha/(U_1 - U) = \alpha/U_1$ является оптимальной относительно $\mathcal{P}_1$.

Пусть α_1 и α_2—перестановки, оптимальные относительно $\mathcal{P}_1$ и $\mathcal{P}_2$ соответственно. Без потери общности предположим, что $\alpha_1 = \alpha_1/U, \alpha_1/(U_1 - U)$ и U ρ-максимально относительно $\mathcal{P}$. Из леммы 3.6 следует, что существует оптимальная относительно $\mathcal{P}$ перестановка α такая, что

$$\alpha = \alpha/U, \alpha/(I(\mathcal{P}) - U) = \alpha_1/U, \alpha/(I(\mathcal{P}) - U).$$

Используя индуктивное предположение, мы теперь получаем, что $\alpha/(I(\mathcal{P}) - U)/U_2 = \alpha_2$ и $\alpha/(I(\mathcal{P}) - U)/U_1 = \alpha_1/(U_1 - U)$.

Лемма может быть легко обобщена на случай, когда $\mathcal{P}$ можно «разложить» более чем на две задачи. В следующем алгоритме принцип декомпозиции используется для упрощения поиска оптимальной перестановки.

Алгоритм 3.3.

Вход: задача составления расписания $\mathcal{P}$ такая, что $\mathcal{P} = \mathcal{P}_1 \| \mathcal{P}_2 \| \ldots \| \mathcal{P}_k$, где $k > 1$.

Выход: α—перестановка, оптимальная относительно $\mathcal{P}$.

Обозначения: $<_i$ обозначает отношение предшествования для $\mathcal{P}_i$, $1 \leqslant i \leqslant k$.

Метод:

procedure PAL
begin
for $i := 1$ **until** k **do**
begin
$\alpha :=$ оптимальная перестановка относительно $\mathcal{P}_i$
$<_i := <_i \cup \{(T_r, T_s) \mid r$ появляется в α раньше, чем $s\}$
end
$< := \bigcup_{i=1}^{k} <_i$; $\alpha := \mathrm{TRF}(\mathcal{P})$
end PAL

Из предыдущих лемм следует, что PAL строит оптимальную перестановку для $\mathcal{P}$ и, более того, что любая

оптимальная перестановка для $\mathscr{P}$ может быть получена в результате выполнения процедуры PAL.

Теорема 3.4. *Пусть $\mathscr{P}$—задача составления расписаний такая, что $\mathscr{P}=\mathscr{P}_1\|\mathscr{P}_2\|\ldots\|\mathscr{P}_k$, где $k>1$. Тогда α является оптимальной перестановкой относительно $\mathscr{P}$ в том и только в том случае, когда она может быть получена в результате применения процедуры* PAL.

3.5. Модель со случайными величинами

В этом параграфе мы обобщим предыдущие результаты, полученные для детерминированных цепочек заданий, путем введения вероятностной модели, описывающей времена выполнения, стоимости пребывания заданий в системе, в также последовательности заданий, составляющих каждую работу. В частности, каждая работа может быть представлена в виде дерева решений, вершины которого соответствуют заданиям. Для каждого задания T имеется произвольное, но известное совместное распределение вероятностей, описывающее время выполнения, стоимость пребывания в системе и выбор задания, если таковое имеется, которое должно выполняться следующим после T. Таким образом, выполнение работы состоит из выполнения некоторой цепочки заданий в дереве, начинающейся в корневом задании и оканчивающейся в некотором его преемнике.

Единственными точками, в которых выполнение работы может быть прервано, а процессор назначен на выполнение другой работы, являются моменты завершения заданий. Среднее время завершения расписания для совокупности независимых работ (так называемой системы работ) вычисляется как взвешенная сумма времен завершения заданий. Нашим основным результатом является эффективный алгоритм, выбирающий в каждый момент завершения задания работу, которая должна быть назначена на процессор, причем таким образом, чтобы математическое ожидание среднего взвешенного времени завершения всей системы заданий было минимальным [10].

В *систему работ* $\mathscr{Y}$ входят следующие компоненты.

1. Система заданий $(\mathscr{T}, <)$, где $\mathscr{T}$—индексированное множество из n заданий, $n\geqslant 0$, а $<$ обозначает частичное упорядочение на $\mathscr{T}$. Множество индексов $\mathscr{T}$ обозначается через $I(\mathscr{Y})$, и каждому j из $I(\mathscr{Y})$ соответствует T_j в $\mathscr{T}$. Тип частичного упорядочения ограничен *выходящим лесом* (напомним, что $<$ является выходящим лесом, если каждая

вершина в графе предшествования, соответствующем $\prec$, имеет не более одного непосредственного предшественника).

2. Каждому j в $I(\mathcal{Y})$ сопоставлена случайная величина S_j, принимающая положительные значения; она определяет время, необходимое для выполнения T_j. Обозначим через τ_j математическое ожидание S_j.

3. Каждому j в $I(\mathcal{Y})$ сопоставлена случайная величина C_j, которая задает стоимость пребывания задания T_j в системе. Диапазон изменения C_j не ограничен. Обозначим через w_j математическое ожидание C_j.

4. Каждому j в $I(\mathcal{Y})$ сопоставлена случайная величина D_j, которая принимает значения, соответствующие индексам непосредственных преемников T_j, либо нуль (нуль не является индексом какого-либо задания). Мы накладываем ограничение, что если T_i является непосредственным преемником T_j, то $P\{D_j=i\}>0$, т. е. вероятность выполнения всех заданий отлична от нуля.

5. Для каждого j в $I(\mathcal{Y})$ задано трехмерное распределение вероятностей F_j, которое определяет совместное поведение совокупности величин S_j, C_j и D_j. Для всех i и j тройки чисел (S_i, C_i, D_i) и (S_j, C_j, D_j) независимы.

Система работ с одним начальным заданием называется *работой*. Непустая система работ $\mathcal{Y}$ может быть представлена в виде

$$\mathcal{Y}=J_1\|\ldots\|J_r,$$

где J_i—работа, $1\leqslant i\leqslant r$.

Пусть $T_{j_1},\ldots,T_{j_r}$—начальные задания в $J_1,\ldots,J_r$ соответственно. Процессор должен быть выделен для выполнения одного из заданий $T_{j_1},\ldots,T_{j_r}$. Если на процессор в момент времени $t\geqslant 0$ назначено выполнение задания T_{j_l}, являющегося начальным в работе J_l, то процессор не может быть перераспределен до момента $t+S_{j_l}$. Завершение задания T_{j_l} дает в среднее взвешенное время завершения вклад величины $(t+S_{j_l})C_{j_l}$. Величина D_{j_l} определяет «подработу» работы J_l, которая должна заменить J_l в $\mathcal{Y}$, а именно: если $D_{j_l}=0$, то J_l должна быть удалена из $\mathcal{Y}$; если же $D_{j_l}=i\neq 0$, то J_l/K_i заменяет J_l в $\mathcal{Y}$ (напомним, что K_i представляет собой множество индексов, включающее i и все индексы j такие, что $T_i\prec T_j$).

Введем функцию переходов δ, которая имеет в качестве аргументов систему работ $\mathcal{Y}$, начальное задание T_j в $\mathcal{Y}$ и величину i, где i есть либо нуль, либо индекс непо-

средственного преемника T_j. Значением δ является (возможно, пустая) система работ $\delta(\mathcal{Y}, T_j, i)$, определяемая следующим образом: если $i=0$, то $\delta(\mathcal{Y}, T_j, i)=\mathcal{Y}/(I(\mathcal{Y})-K_j)$; если же $i\neq 0$, то $\delta(\mathcal{Y}, T_j, i)=\mathcal{Y}/(I(\mathcal{Y})-(K_j-K_i))$.

На рис. 3.2 изображена система работ, состоящая из двух работ J_1 и J_2. Начальное задание T_1 работы J_1 имеет два возможных «исхода»: $D_1=2$ или $D_1=3$. Случайные величины S_1, C_1 и D_1 распределены следующим образом:

$$\mathrm{P}\{S_1=1,\ C_1=1/2,\ D_1=2\}=p,$$
$$\mathrm{P}\{S_1=2,\ C_1=1,\ \ D_1=3\}=1-p,$$

где $0\leqslant p\leqslant 1$. Распределение остальных переменных имеет вид

$$\mathrm{P}\{D_i=0\}=1 \quad \text{для} \quad i=2,\ 3,\ 4,$$
$$\mathrm{P}\{S_i=1\}=1 \quad \text{для} \quad i=2,\ 3,\ 4,$$
$$\mathrm{P}\{C_i=1\}=1 \quad \text{для} \quad i=2,\ 4,$$
$$\mathrm{P}\{C_3=5\}=1.$$

Следовательно, работа J_1 имеет две возможные реализации, а именно:

$$T_1/1/(1/2) \quad \text{и затем} \quad T_2/1/1$$

или

$$T_1/2/1 \quad \text{и затем} \quad T_2/1/5,$$

где обозначения даны в следующем порядке: задание /время выполнения/стоимость. Первая реализация имеет вероятность p, а вторая — вероятность $1-p$. Работа J_2 имеет ровно одну реализацию $T_4/1/1$.

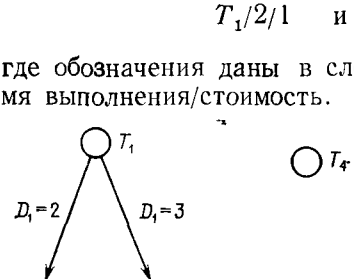


Рис. 3.2.

Если в момент $t=0$ назначить на процессор задание T_1, то имеются только две возможности:

1. T_1 завершается в момент $t=1$, вклад в среднее взвешенное время составляет 1/2, результирующая система работ имеет вид $\delta(\mathcal{Y}, T_1, 2)=\mathcal{Y}/\{2,\ 4\}$.

2. T_1 завершается в момент $t=2$, вклад в среднее взвешенное время составляет 2, результирующая система работ имеет вид $\delta(\mathcal{Y}, T_1, 3)=\mathcal{Y}/\{3,\ 4\}$.

Правило упорядочения ξ представляет собой целочисленную функцию, которая ставит в соответствие каждой не-

пустой системе работ $\mathcal{Y}$ начальное задание $\xi(\mathcal{Y})$ этой системы.

Величина среднего взвешенного времени завершения при использовании в системе $\mathcal{Y}$ правила упорядочения ξ в момент $t \geqslant 0$ определяется с помощью процедуры mwft, примененной к ξ, $\mathcal{Y}$ и t.

Алгоритм 3.4.

Вход: правило упорядочения ξ, система работ $\mathcal{Y}$ и неотрицательное число t.

Выход: величина среднего взвешенного времени для решающего правила ξ, примененного к $\mathcal{Y}$, в предположении, что процессор недоступен до момента $t \geqslant 0$.

Метод:

procedure mwft (ξ, $\mathcal{Y}$, t)
begin
if $\mathcal{Y}$ пуста **then return** 0
else
begin
Пусть j таково, что $T_j = \xi(\mathcal{Y})$; $(\tau, w, i) = (S_j, C_j, D_j)$
return $(t+\tau)w + \text{mwft}(\xi, \delta(\mathcal{Y}, T_j, i), t+\tau)$
end
end mwft

Оператор присвоения $(\tau, w, i) := (S_j, C_j, D_j)$ выполняется путем генерирования случайных величин S_j, C_j и D_j в соответствии с F_j и присвоения значений этих величин переменным τ, w, i соответственно. Таким образом, $\text{mwft}(\xi, \mathcal{Y}, t)$ представляет собой случайную величину, значения которой зависят от исходов экспериментов, производимых в процессе выполнения алгоритма 3.4.

Обозначим через $\overline{\zeta}$ среднее значение случайной величины ζ. Мы говорим, что правило упорядочения ξ является *оптимальным*, если для всех правил упорядочения ξ' и всех систем работ $\mathcal{Y}$ имеет место соотношение

$$\overline{\text{mwft}}(\xi, \mathcal{Y}, 0) \leqslant \overline{\text{mwft}}(\xi', \mathcal{Y}, 0).$$

Пусть $\mathcal{Y}$ — система работ, изображенная на рис. 3.2, а ξ — такое правило упорядочения, что $\xi(\mathcal{Y}) = T_1$, $\xi(\mathcal{Y}/\{2, 4\}) = T_4$ и $\xi(\mathcal{Y}/\{3, 4\}) = T_3$. Легко видеть, что $\text{mwft}(\xi, \mathcal{Y}, 0)$ принимает два возможных значения в зависимости от исхода D_1. Эти два значения суть $1/2 + 2 + 3 = 11/2$ и $2 + 15 + 4 = 21$, причем первое из них реализуется с вероятностью p, а второе — с вероятностью $1-p$. Поэтому $\overline{\text{mwft}}(\xi, \mathcal{Y}, 0) = 21 - (31/2)\,p$.

Существует лишь одна иная стратегия, которую нужно рассмотреть, а именно правило упорядочения ξ', назначающее вначале T_4 вместо T_1. В этом случае также имеются только два возможных исхода, и мы получаем $\overline{\text{mwft}}(\xi', \mathcal{Y}, 0) = 24 - 19p$. Из этого делаем вывод о том, что ξ дает лучший результат при $0 \leqslant p \leqslant 6/7$, а ξ' оказывается лучшим, когда $6/7 \leqslant p \leqslant 1$.

3.6. Ранговая функция

Целью данного параграфа является определение ранговой функции ρ, которая ставит в соответствие каждой работе J положительное число $\rho(J)$, называемое рангом J. Свойство этой функции заключается в том, что для получения оптимального расписания необходимо назначать на процессор начальное задание из той работы, которая имеет наибольший ранг. В следующем параграфе мы докажем, что расписание, построенное на основе стратегии «наибольший ранг — первым», является оптимальным и, наоборот, что все оптимальные правила упорядочения соответствуют этой стратегии.

Пусть J есть работа, T_j — начальное задание в J и U — подмножество индексов заданий в J. Определим случайные величины $s(J, U)$ и $c(J, U)$ следующим образом:

$$s(J, U) = \begin{cases} 0, & \text{если } j \notin U, \\ S_j + s(\delta(J, T_j, D_j), U), & \text{если } j \in U, \end{cases}$$

$$c(J, U) = \begin{cases} 0, & \text{если } j \notin U, \\ C_j + c(\delta(J, T_j, D_j), U), & \text{если } j \in U. \end{cases}$$

Пусть $\tau(J, U)$ и $w(J, U)$ обозначают средние значения $s(J, U)$ и $c(J, U)$ соответственно.

Пусть U — начальное множество относительно J. Определим $\rho(J, U)$ как

$$\rho(J, U) = w(J, U)/\tau(J, U).$$

Рангом работы J назовем величину

$$\rho(J) = \max\{\rho(J, U) \mid U \text{ является начальным множеством относительно } J\}.$$

Мы говорим, что множество $U \subseteq I(J)$ является ρ-максимальным относительно J, если:

1) U — начальное множество относительно J;

2) $\rho(J, U) = \rho(J)$;

3) из того, что V является начальным множеством относительно J, $V \subseteq U$ и $\rho(J, V) = \rho(J)$, следует, что $V = U$.

В оставшейся части параграфа мы рассмотрим некоторые свойства ранговой функции ρ.

Пусть T_j — задание в J, а $T_{i_1}, \ldots, T_{i_k}$ — последовательность заданий в J такая, что $T_{i_{l+1}}$ является непосредственным преемником T_{i_l} для $l = 1, \ldots, k-1$, T_{i_1} является начальным заданием J и $T_{i_k} = T_j$. Определим p_j — вероятность «достижения» T_j — как

$$p_j = \prod_{l=1}^{k-1} \mathrm{P}\{D_{i_l} = i_{l+1}\}.$$

Произведение пустого множества сомножителей ($k-1=0$) равно 1. Из данного нами определения работы следует, что $p_j > 0$ для всех $j \in I(J)$.

Лемма 3.13. *Пусть J есть работа и U — начальное множество относительно J. Тогда*

$$\tau(J, U) = \sum_{j \in U} p_j \tau_j,$$

$$w(J, U) = \sum_{j \in U} p_j w_j.$$

Доказательство. Доказательство проводится индукцией по числу заданий в U. База индукции (при $|U| = 1$) получается легко, так как $\tau(J, U) = \tau_j$ и $w(J, U) = w_j$, где T_j — начальное задание J.

Пусть U — начальное множество в J такое, что $|U| \geqslant 2$, и предположим, что лемма верна для любой работы и соответствующего ей начального множества с меньшим чем $|U|$ числом элементов.

Пусть T_i — начальное задание в J и $T_{i_1}, \ldots, T_{i_k}$ — непосредственные преемники T_i. Положим $U_{i_l} = K_{i_l} \cap U$ для $l = 1, \ldots, k$. Множество U_{i_l} является либо пустым, либо начальным для J/K_{i_l}, $l = 1, \ldots, k$. Мы можем записать $\tau(J, U)$ в виде

$$\tau(J, U) = \tau_i + \sum_{l=1}^{k} \mathrm{P}\{D_i = i_l\} \tau(J/K_{i_l}, U_{i_l}).$$

Применяя предположение индукции к $\tau(J/K_{i_l}, U_{i_l})$ и используя определение величин p_j, нетрудно увидеть, что

$$\tau(J, U) = \sum_{j \in U} p_j \tau_j.$$

Аналогично доказывается выражение для $w(J, U)$. Определим величины $\tau(U)$, $w(U)$ и $\rho(U)$, где U — произвольное непустое подмножество индексов работы J:

$$\tau(U) = \sum_{j \in U} p_j \tau_j, \quad w(U) = \sum_{j \in U} p_j w_j,$$

$$\rho(U) = w(U)/\tau(U).$$

Лемма 3.14. *Пусть J — работа, U — непустое подмножество $I(J)$. Если l — такой индекс в $I(J)$, что $U \subseteq K_l$, то*

$$\tau(U) = p_l \sum_{j \in U} p'_j \tau_j, \quad w(U) = p_l \sum_{j \in U} p'_j w_j,$$

где вероятность p_l вычислена для работы J, а p'_j, $j \in U$, — для работы J/K_l.

Доказательство. Доказательство непосредственно следует из определения упомянутых величин.

Следующая лемма связывает ранг некоторой «подработы» данной работы с рангом самой работы. Мы используем этот результат в § 3.7.

Лемма 3.15. *Пусть U есть ρ-максимальное множество относительно работы J, а l — такой индекс в U, что T_l не является начальным заданием J. Тогда $\rho(J/K_l) > \rho(J)$.*

Доказательство. Пусть $U_1 = U - K_l$ и $U_2 = K_l \cap U$. Применяя лемму 3.2, получаем

$$\rho(J) = \eta\rho(U_1) + (1-\eta)\rho(U_2),$$

где $0<\eta<1$. Поскольку U является начальным множеством в J, мы имеем $\rho(U_1)<\rho(J)$; поэтому $\rho(U_2)>\rho(J)$.

Поскольку $\rho(J/K_l) \geqslant \rho(J/U_2, U_2)$, остается показать, что $\rho(J/U_2, U_2) = \rho(U_2) = w(U_2)/\tau(U_2)$, где $w(U_2)$ и $\tau(U_2)$ определены для работы J, а это прямо следует из леммы 3.14.

3.7. Оптимальные расписания

Пусть ξ — правило упорядочения, а $\mathcal{Y} = J_1 \| \ldots \| J_r$ — непустая система, состоящая из работ J_i, $i = 1, \ldots, r$. Пусть l — такой индекс, что $\xi(\mathcal{Y})$ является начальным заданием работы J_l. Скажем, что $\xi(\mathcal{Y})$ назначено по наибольшему рангу, если $\rho(J_l) \geqslant \rho(J_i)$ для $i = 1, \ldots, r$. Правило упорядочения ξ называется LR-*правилом*

упорядочения *), если для всех систем работ $\mathcal{Y}$ $\xi(\mathcal{Y})$ назначено по наибольшему рангу.

Теорема 3.5. *Правило упорядочения* ξ *оптимально тогда и только тогда, когда оно является* LR-*правилом.*

Доказательство. Правило упорядочения ξ назовем *n-оптимальным*, если для всех правил упорядочения ξ' и всех систем работ $\mathcal{Y}$ из $|J(\mathcal{Y})| \leqslant n$ следует $\overline{\text{mwft}}(\xi, \mathcal{Y}, 0) \leqslant \overline{\text{mwft}}(\xi', \mathcal{Y}, 0)$. Правило упорядочения ξ называется *n*-LR-*правилом*, если для всех систем работ $\mathcal{Y}$, таких, что $|I(\mathcal{Y})| \leqslant n$, $\xi(\mathcal{Y})$ назначено по наибольшему рангу. Для того чтобы доказать теорему, мы покажем для каждого $n \geqslant 1$, что ξ является n-оптимальным правилом упорядочения в том и только в том случае, когда оно является n-LR-правилом. Доказательство проводится методом индукции по n; для $n=1$ результат очевиден.

Пусть $n \geqslant 2$; предположим, что существуют n-оптимальное правило упорядочения ξ и система работ $\mathcal{Y} = J_1 \| \ldots \| J_r$ такие, что $|I(\mathcal{Y})| = n$, а $\xi(\mathcal{Y})$ назначено не по наибольшему рангу. Пусть $T_{j_1} = \xi(\mathcal{Y})$ есть начальное задание работы J_{i_1}. По предположению $r \geqslant 2$, и, следовательно, существует начальное задание T_{j_2} работы J_{i_2}, которое назначено LR-правилом (относительно $\mathcal{Y}$), при этом $\rho(J_{i_1}) > \rho(J_{i_2})$. Наша цель теперь — получить противоречие, построив правило упорядочения ξ' такое, что $\overline{\text{mwft}}(\xi', \mathcal{Y}, 0) < \overline{\text{mwft}}(\xi, \mathcal{Y}, 0)$.

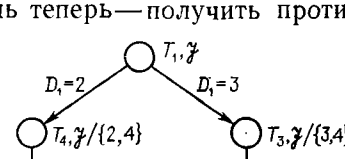


Рис. 3.3.

Представим процесс применения ξ к $\mathcal{Y}$ в виде помеченного дерева E**). Каждая вершина v из E имеет две метки: $T(v)$ (задание) и $\mathcal{Y}(v)$ (непустая система работ), связанные соотношением $T(v) = \xi(\mathcal{Y}(v))$. Корнем дерева является v_0, и $\mathcal{Y}(v_0) = \mathcal{Y}$. У каждой вершины имеется по одному поддереву для каждой непустой системы работ, которые получаются из $\mathcal{Y}(v)$ после назначения $T(v)$ на процессор.

Дерево, изображенное на рис. 3.3, иллюстрирует правило упорядочения, примененное к задаче, представленной на

---

*) LR — largest rank — наибольший ранг. (*Прим. перев.*)

**) *Дерево* E представляет собой конечное множество элементов, называемых *вершинами*, в котором имеется выделенный элемент, называемый *корнем*, и разбиение $E_1, \ldots, E_k$ множества остальных элементов (если таковые имеются) на $k \geqslant 1$ деревьев. Деревья $E_1, \ldots, E_k$ называются *поддеревьями* с данным корнем.

рис. 3.2. Здесь использовано одно из правил упорядочения, рассмотренных в предыдущем параграфе.

Если V есть подмножество вершин E, положим

$$I(V) = \{i \mid v \in V, \quad T(v) = T_i\}.$$

Пусть A является *максимальным* подмножеством вершин E, т. е.:

1) v_0 находится в A;

2) если $v \in A$, $v \neq v_0$, то непосредственный предшественник v находится в A;

3) если $v \in A$, то $T(v)$ принадлежит J_{t_1}.

Ясно, что множество A единственно и $I(A)$ является начальным множеством в J_{t_1}.

Вершина v называется *непосредственным преемником* подмножества вершин V, если v не принадлежит V и непосредственный предшественник v содержится в V.

Пусть $v_1, \ldots, v_a$—непосредственные преемники A. Пусть также $\mathcal{Y}_i = \mathcal{Y}(v_i)$ для $i = 1, \ldots, a$. Поскольку $|I(\mathcal{Y}_i)| \leqslant n-1$, мы можем воспользоваться предположением индукции для того, чтобы заключить, что назначения, произведенные при помощи ξ в $\mathcal{Y}$, осуществлены в соответствии с LR-правилом, более того, мы можем изменять эти назначения, сохраняя указанное свойство, не оказывая при этом влияние на среднее взвешенное время прохождения.

Пусть E_i—поддерево дерева E с корнем v_i, соответствующее $\mathcal{Y}_i$, $i = 1, \ldots, a$ (см. рис. 3.4). Поскольку все

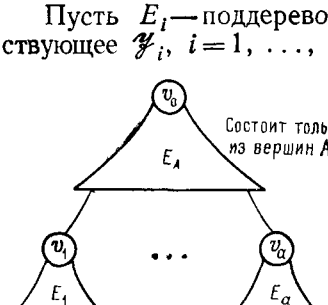


Рис. 3.4.

Рис. 3.5.

назначения в A произведены в J_{t_1}, $\rho(J_{t_2})$—наибольший ранг среди всех работ в $\mathcal{Y}_i$, $i = 1, \ldots, a$. Пусть U есть ρ-максимальное множество для J_{t_2}. Используя предположение индукции и лемму 3.15, мы можем представить E_i в виде, изображенном на рис. 3.5, где $I(B) = U$. Мы не показываем зависимость B от i, поскольку, на

основании леммы 3.15, назначения заданий из J_{l_2}, индексы которых входят в U, произведены LR-правилом. Следовательно, в качестве начальной «структуры» E_i, $i=1, \ldots, a$, может быть взято E_B.

На рис. 3.6 представлена структура E и структура E', получающегося из E путем «перестановки». Мы не будем

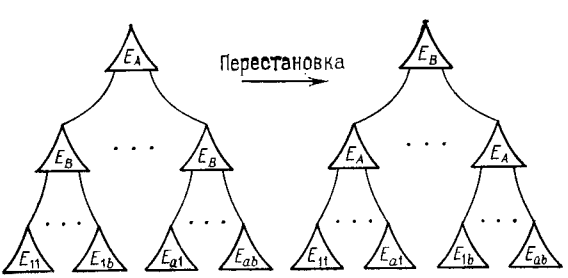


Рис. 3.6.

описывать все детали перестановки; идея ее заключается просто в том, что задания с индексами из $I(B)$ выполняются раньше заданий с индексами из $I(A)$. Метки $\mathcal{Y}(\cdot)$ вершин из E_A и E_B в дереве E' отличаются от аналогичных меток в E из-за изменения порядка выполнения. Пусть ξ' обозначает правило упорядочения, изображенное с помощью E'. Легко вычислить изменение математического ожидания среднего взвешенного времени прохождения при переходе от ξ к ξ':

$$\overline{\mathrm{mwft}}(\xi', \mathcal{Y}, 0) = \overline{\mathrm{mwft}}(\xi, \mathcal{Y}, 0) + \\ + \overline{s(J_{l_2}, I(B))\, c(J_{l_1}, I(A))} - \overline{s(J_{l_1}(A))\, c(J_{l_2}, I(B))}.$$

Принимая во внимание независимость $s(J_{l_2}, I(B))$ и $c(J_{l_1}, I(A))$, а также независимость $s(J_{l_1}, I(A))$ и $c(J_{l_2}, I(B))$, получим

$$\overline{\mathrm{mwft}}(\xi', \mathcal{Y}, 0) = \overline{\mathrm{mwft}}(\xi, \mathcal{Y}, 0) + \tau(J_{l_2}, I(B))\, w(J_{l_1}, I(A)) - \\ - \tau(J_{l_1}, I(A))\, w(J_{l_2}, I(B)) = \\ = \overline{\mathrm{mwft}}(\xi, \mathcal{Y}, 0) + \gamma(\rho(J_{l_1}, I(A)) - \rho(J_{l_2})),$$

где $\gamma > 0$. Поскольку $\rho(J_{l_2}) > \rho(J_{l_1}) \geqslant \rho(J_{l_1}, I(A))$, мы получаем противоречие.

Для того чтобы завершить доказательство, мы должны показать, что если $I(\mathcal{Y}) = n$, то все n-LR-расписания для $\mathcal{Y}$ имеют одну и ту же стоимость. С этой целью может быть использован предыдущий метод «перестановки», с по-

мощью которого одно n-LR-расписание сводится к другому n-LR-расписанию, и при этом математическое ожидание среднего взвешенного времени прохождения не увеличится. Детали доказательства мы опускаем.

3.8. Алгоритмы для моделей, содержащих случайные величины

В данном параграфе мы рассмотрим вычислительные аспекты применения LR-правила упорядочения. Алгоритм 3.5 является LR-алгоритмом упорядочения; он имеет на входе систему работ $\mathcal{Y}$ и определяет последовательность назначений и среднее взвешенное время завершения. Этот алгоритм имеет такую же структуру, как и алгоритм 3.4, однако здесь мы можем быть уверены, что все назначения произведены по LR-правилу.

Алгоритм 3.5.

Вход: $\mathcal{Y} = J_1 \| \ldots \| J_r$.

Выход: последовательность LR-назначений и среднее взвешенное время завершения получившегося расписания.

Метод:

1. **procedure** LR ($\mathcal{Y}$)
2. **begin**
3. $t := 0$
4. mwft: $= 0$
5. $l[i] :=$ индекс начального задания J_i для $i = 1, \ldots, r$
6. **while** не выполняется ($l[i] = 0$ для $i = 1, \ldots, r$) **do**
7. **begin**
8. Пусть j таково, что $l[j] \neq 0$ и $\rho(J_j/K_{l[j]}) \geqslant \rho(J_i/K_{l[i]})$ для всех $1 \leqslant i \leqslant r$ и $l[i] \neq 0$
9. $U :=$ ρ-максимальное множество относительно $J_j/K_{l[j]}$
10. **while** $l[j] \in U$ **do**
11. **begin**
12. $(\tau, w, d) := (S_{l[j]}, G_{l[j]}, D_{l[j]})$
13. mwft: $=$ mwft $+ (t + \tau) w$
14. $t := t + \tau$
15. $l[j] := d$
16. **end**
17. **end**
18. **return** mwft
19. **end** LR

Как и в алгоритме 3.4, LR ($\mathcal{Y}$) является случайной величиной, конкретные значения которой зависят от выбора j в строке 8 и от результата операции присвоения в строке 12. Из сравнения строк 13 и 14 процедуры

LR ($\mathscr{Y}$) с оператором **return** в процедуре mwft будет следовать, что эта случайная величина равна среднему взвешенному времени завершения для построенного расписания.

Теорема 3.6. *Пусть* $\mathscr{Y}$ *— система работ,* ξ *— правило упорядочения. Тогда* $\overline{\mathrm{LR}}(\mathscr{Y}) \leqslant \overline{\mathrm{mwft}}(\xi, \mathscr{Y}, 0)$.

Доказательство. Данный результат является прямым следствием теоремы 3.5 и леммы 3.15.

В строке 8 процедуры LR определяется работа с наибольшим рангом, что может оказаться обременительным в вычислительном аспекте, если не принять некоторых мер предосторожности. Это относится и к определению ρ-максимального множества U в строке 9. Предварительное применение к каждой работе приведенного ниже алгоритма дает информацию, которая может быть использована для эффективного выполнения вычислений, соответствующих строкам 8 и 9 процедуры LR.

Пусть J есть работа. Скажем, что разбиение $B_1, \ldots, B_q$ множества $I(J)$ является *полным* относительно J, если для каждого i в $\{1, \ldots, q\}$ выполняются следующие условия.

1. B_i содержит единственный элемент, скажем l, такой, что если T_j является предшественником T_l в J, то j не является элементом B_i. Индекс l называется *начальным индексом* B_i.

2. Если j — индекс из B_i, не являющийся начальным, то j' находится в B_i, где $T_{j'}$ — непосредственный предшественник T_j.

3. Если l есть начальный индекс B_i, то B_i является ρ-максимальным множеством относительно J/K_l.

Не составляет труда показать, что полное разбиение относительно J существует и является единственным. Следующий алгоритм описывает стратегию определения полного разбиения работы.

Алгоритм 3.6.

Вход: работа J.

Выход: полное разбиение $B_1, \ldots, B_q$ относительно J и числа $R_1, \ldots, R_q$ такие, что $R_i = \rho(J/K_{l_i})$, где l_i есть начальный индекс набора B_i для $i = 1, \ldots, q$.

Метод:

1. **procedure** RANK
2. **begin**
3. $q := 0$
4. $U := I(J)$

5. **while** $U \neq \varnothing$ **do**
6. **begin**
7. Пусть l—такой индекс в U, что $\rho(K_l \cap U) \leqslant \rho(K_i \cap U)$ для всех i в U и $\rho(K_l \cap U) < \rho(K_j \cap U)$ для всех j в $K_l \cap U$, $j \neq i$
8. $q := q+1$
9. $B_q := K_l \cap U$
10. $R_q := \rho(K_l \cap U)$
11. $U := U - K_l$
12. **end**
13. **end** RANK

Лемма 3.16. *Пусть* $B_1, \ldots, B_q$ *и* $R_1, \ldots, R_q$ *получены в результате применения процедуры* RANK *к работе* J. *Тогда* $B_1, \ldots, B_q$—*полное разбиение* J, *и если* l_i—*начальный индекс множества* B_i, *то* $\rho(J/K_{l_i}) = R_i$.

Доказательство. Естественными орудиями доказательства этого утверждения являются лемма 3.8 и метод индукции. В целом процедура стандартная, и мы оставляем детали доказательства заинтересованному читателю.

В предположении, что для каждого задания T_j в J известны числа τ_j, w_j и $P\{D_j = i\}$ для каждого i такого, что T_i является непосредственным преемником T_j, прямая реализация алгоритма 3.6 будет иметь сложность $O(n^2)$ в наихудшем случае, где $n = |I(J)|$. Основная идея заключается в расчете величин $\tau'_j = p_j \tau_j$ и $w'_j = p_j w_j$ для каждого j в $I(J)$. Это может быть проделано путем однократного просмотра графа предшествования J и занимает $O(n)$ времени. Процедура минимизации в строке 7 также может быть выполнена путем просмотра графа предшествования J снизу вверх, а именно: если T_j есть задание с непосредственными преемниками $T_{i_1}, \ldots, T_{i_k}$ и для T_{i_l}, $l = 1, \ldots, k$, известны $\tau(K_{i_l})$ и $w(K_{i_l})$, мы можем записать

$$\rho(K_j) = \frac{w(K_j)}{\tau(K_j)} = \frac{w'_j + \sum_{l=1}^{k} w(K_{i_l})}{\tau'_j + \sum_{l=1}^{k} \tau(K_{i_l})}.$$

Поэтому, если мы посетим задания $T_{i_1}, \ldots, T_{i_k}$ перед посещением T_j, нам удастся вычислить $\rho(K_j)$ на основе локальной информации.

Поскольку строка 7 выполняется за время $O(n)$, то общая временная сложность алгоритма 3.6 в наихудшем случае составляет $O(n^2)$.

Начального вычисления произведений $p_j w_j$ и $p_j \tau_j$ можно избежать, если заметить, что при расчете отношений $w(K_j)/\tau(K_j)$ происходит сокращение, описанное в лемме 3.14. Поэтому, если вместо $w(K_j)$ и $\tau(K_j)$ мы вычислим $\tilde{w}(K_j) = (1/p_j)\, w(K_j)$ и $\tilde{\tau}(K_j) = (1/p_j)\, \tau(K_j)$, искомые отношения не изменятся:

$$\rho(K_j) = \frac{w(K_j)}{\tau(K_j)} = \frac{\tilde{w}(K_j)}{\tilde{\tau}(K_j)}.$$

Оказывается, $\tilde{w}$ и $\tilde{\tau}$ могут быть получены без выполнения начального вычисления величин $\tilde{w}_j$ и $\tilde{\tau}_j$ для каждого задания T_j, а именно:

$$\tilde{w}(K_j) = w_j + \sum_{l=1}^{k} P\{D_j = i_l\}\, \tilde{w}(K_{i_l}),$$

$$\tilde{\tau}(K_j) = \tau_j + \sum_{l=1}^{k} P\{D_j = i_l\}\, \tilde{\tau}(K_{i_l}).$$

Вернемся теперь к процедуре LR и используем полные разбиения для получения эффективной реализации LR-правила упорядочения. Пусть J — работа и $\mathcal{B}(J) = \{(B_1, R_1), \ldots, (B_q, R_q)\}$; здесь $B_1, \ldots, B_q$ — полное разбиение J и $R_i = \rho(J/K_{l_i}) = \rho(B_i)$, где l_i есть начальный индекс множества B_i для $i = 1, \ldots, q$.

Лемма 3.17. *Пусть $\mathcal{Y} = J_1 \| \ldots \| J_2$ — система работ, а $l_1, \ldots, l_r$ — значения программных переменных $l[1], \ldots, l[r]$ непосредственно перед очередным выполнением строки* 6 *в процедуре* LR *при ее применении к $\mathcal{Y}$. Если $l_i \neq 0$, то существует пара $(B, R) \in \mathcal{B}(J_i)$ такая, что l_i является начальным индексом в B.*

Лемма 3.17 гарантирует, что информация в $\mathcal{B}(J_1), \ldots, \mathcal{B}(J_r)$ является достаточной (т. е. она может быть использована для эффективного выполнения шагов 8 и 9 в процедуре LR). В самом деле, мы предварительно определяем все ранги и ρ-максимальные множества, которые потребуются при выполнении LR($\mathcal{Y}$). Очевидно, что если ранги работ хранятся в очереди с приоритетами, то определение j в строке 8 может быть выполнено за фиксированное время. Организация очереди с приоритетами может быть осуществлена за $O(\log_2 r)$ шагов при каждом обращении к очереди [3]. Поэтому в предположении, что мы имеем $\mathcal{B}(J_1), \ldots, \mathcal{B}(J_r)$, LR-правило упорядочения может быть реализовано за $O(n \log_2 r)$ единиц времени, где $n = |I(\mathcal{Y})|$, r — число работ в $\mathcal{Y}$.

3.9. Составление расписаний для многопроцессорных систем

В данном параграфе мы рассмотрим детерминированные задачи составления расписаний для нескольких процессоров. Для того чтобы сделать проблему решаемой, введем два ограничения: 1) задания независимым (т. е. $<=\varnothing$), и 2) все стоимости пребывания заданий в системе равны 1.

Таким образом, описание задачи упорядочения состоит из матрицы времен выполнения. Пусть $[\tau_{ij}]$ обозначает матрицу размером $m \times n$, где τ_{ij} есть время выполнения задания T_j на процессоре i для $1 \leqslant j \leqslant n$ и $1 \leqslant i \leqslant m$ [72, 11, 12].

Расписание ξ состоит из разбиения множества $\{1, \ldots, n\}$ на m блоков $B_1, \ldots, B_m$, некоторые из которых могут быть пустыми, и перестановок α_i элементов B_i для $i=1, \ldots, m$. Блок B_i содержит индексы тех заданий, которые должны выполняться на процессоре i, а α_i определяет порядок выполнения заданий на i-м процессоре, $1 \leqslant i \leqslant m$.

Рассмотрим пример, в котором $n=8$, $m=3$, а $[\tau_{ij}]$ имеет вид

$$[\tau_{ij}] = \begin{bmatrix} 2 & 3 & 1 & 4 & 6 & 5 & 2 & 3 \\ 1 & 4 & 1 & 5 & 5 & 4 & 3 & 4 \\ 3 & 2 & 3 & 2 & 3 & 5 & 1 & 2 \end{bmatrix}.$$

Расписание ξ состоит из блоков $B_1=\{3, 5, 8\}$, $B_2=\varnothing$ и $B_3=\{1, 2, 4, 6, 7\}$ и перестановок $\alpha_1=3, 8, 5$, $\alpha_2=\lambda$ и $\alpha_3=7, 4, 2, 6, 1$. Для этого расписания имеем

$$\mathrm{mwft}(\xi) = \sum_{j=1}^{n} f_j(\xi) = 47.$$

В следующих параграфах мы покажем, как свести такую задачу упорядочения к задаче о потоке минимальной стоимости [42], и дадим эффективный алгоритм, входом которого является матрица $[\tau_{ij}]$, а выходом — оптимальное расписание ξ^*.

3.10. Сведéние

Пусть $[\tau_{ij}]$ — $m \times n$-матрица времен выполнения, ξ — некоторое расписание. Предположим, что, согласно ξ, задание T_j должно выполняться на i-м процессоре и что имеется ровно k, $0 \leqslant k < n$, заданий, которые выполняются на этом процессоре следом за T_j. Величина τ_{ij} вносит вклад во время завершения задания T_j и тех k заданий, которые

выполняются следом за T_j. Учитывая это, мы запишем mwft(ξ) таким образом, чтобы выделить члены, зависящие от τ_{ij}, а именно:

$$\text{mwft}(\xi) = \\ = (k+1)\,\tau_{ij} + \text{другие слагаемые, не содержащие } \tau_{ij}.$$

Если T_j выполняется на i-м процессоре последним, то коэффициент при τ_{ij} равен 1; если T_j выполняется на i-м процессоре предпоследним, то коэффициент равен 2, и так далее. Это позволяет переформулировать нашу задачу следующим образом.

Образуем матрицу C размером $m' \times n$, имеющую вид

$$C = \begin{bmatrix} [1\tau_{ij}] \\ [2\tau_{ij}] \\ \cdots \\ [n\tau_{ij}] \end{bmatrix},$$

где $[r\tau_{ij}]$ обозначает матрицу, получающуюся из $[\tau_{ij}]$ умножением каждого ее элемента на r. Элементы C обозначаются c_{ij}, $1 \leqslant i \leqslant m'$, $1 \leqslant j \leqslant n$, $m' = mn$.

Неформально говоря, элементы матрицы $[1\tau_{ij}]$ представляет собой возможные вклады в mwft при условии, что задания выполняются последними на соответствующих процессорах, элементы $[2\tau_{ij}]$ представляют собой возможные вклады в mwft при условии, что эти задания выполняются предпоследними, и так далее.

Используя матрицу C в качестве матрицы стоимости, сформулируем задачу о потоке минимальной стоимости в сети, где (i, j)-й элемент матрицы C определяет стоимость перемещения единицы потока (чего-либо) из точки x_i в точку y_j. После постановки этой задачи мы покажем, каким образом оптимальный поток определяет оптимальное расписание.

В транспортной сети N [42], изображенной на рис. 3.7, каждая дуга имеет единичную пропускную способность в направлении стрелки, за исключением обратной дуги, которая имеет пропускную способность n единиц. Пусть (z, z') — дуга в N и $c(z, z')$ обозначает стоимость переноса единицы потока от z к z'. Мы имеем $c(x, y) = c_{ij}$ для $1 \leqslant i \leqslant m'$ и $1 \leqslant j \leqslant n$, а все другие стоимости равны нулю.

Поток в N определяется заданием функции h, которая ставит в соответствие дугам сети N числа из множества $\{0, 1, \ldots, \}$. Пусть (z, z') — дуга в N и $h(z, z')$ обозначает

поток от z к z'. Поток называется *допустимым*, если для всех дуг (z, z') в N величина $h(z, z)$ меньше или равна пропускной способности дуги (z, z') и поток сохраняется в каждой вершине N. *Величина* h допустимого потока равна $h(x_0, y_0)$.

Допустимый поток h называется *оптимальным*, если его величина равна n, а его стоимость

$$\text{cost}(h) = \sum_{i=1}^{m'} \sum_{j=1}^{n} c(x_i, y_j)\, h(x_i, y_j)$$

достигает минимума по всем допустимым потокам величины n.

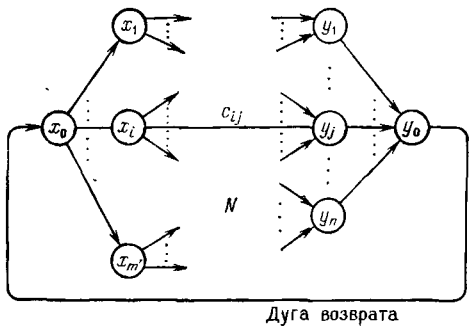


Рис. 3.7.

Расписание ξ легко представить в виде допустимого потока в сети N. Для всех заданий T_j, если задание T_j назначено на i-й процессор и имеется k заданий, выполняемых на i-м процессоре после T_j, положим $h(x_{i'}, y_j) = h(x_0, x_{i'}) = 1$, где $i' = km + i$. Положим также $h(y_0, x_0) = n$ и $h(y_j, y_0) = 1$ для $1 \leqslant j \leqslant n$. Все остальные переменные положим равными нулю. Легко видеть, что поток h является допустимым и что $\text{mwft}(\xi) = \text{cost}(h)$.

С другой стороны, если $\tau_{ij} > 0$ для $1 \leqslant i \leqslant m$ и $1 \leqslant j \leqslant n$, то любой оптимальный поток может быть интерпретирован как оптимальное расписание. Если $h(x_{i'}, y_j) = 1$, то T_j назначается на процессор i, а после него на этом процессоре выполняется еще k заданий, где i и k определяются следующей системой уравнений:

$$\begin{aligned} &i' = km + i, \\ &0 \leqslant k < n, \\ &1 \leqslant i \leqslant m. \end{aligned} \qquad (1)$$

Уравнения (1) однозначно определяют k и i. Вклад задания T_j в mwft составляет $(k+1)\,\tau_{ij}$ и в точности совпадает с коэффициентом при $h(x_{i'}, y_j)$ в выражении для стоимости. Нетрудно показать, что расписание, полученное таким путем из оптимального потока h, составляется хорошо в том смысле, что h определяет k дополнительных заданий, следующих за T_j. Это вытекает из ограничений на величины τ_{ij}. Если существует задание T_j с $\tau_{ij}=0$ для некоторого i, то T_j может быть распределено на i-й процессор в момент $t=0$ без внесения вклада в mwft для расписания. Таким образом, мы можем игнорировать все такие задания в любых алгоритмах упорядочения.

Теорема 3.7. *Пусть $\mathcal{P}$ есть задача упорядочения, определяемая $m\times n$-матрицей $[\tau_{ij}]$ времен выполнения, причем $\tau_{ij}>0$ для $1\leqslant i\leqslant m$ и $1\leqslant j\leqslant n$. Если h — оптимальный поток в сети N, соответствующей $\mathcal{P}$, то расписание ξ, определяемое из h с помощью системы уравнений* (1), *оптимально для $\mathcal{P}$.*

3.11. Многопроцессорные алгоритмы

Допустимый поток h в N называется *экстремальным*, если он имеет минимальную стоимость среди всех допустимых потоков величины $h(x_0, y_0)$. Хорошо известный метод отыскания оптимального потока заключается в последовательном построении экстремальных потоков $h_0, \ldots, h_p$ таких, что величина h_p равна p, $0\leqslant p\leqslant n$ [42]. Начальный поток h_0 имеет нулевые значения на всех дугах N. Предположим, что мы определили h_p для некоторого p, меньшего чем n.

Для того чтобы найти h_{p+1}, мы выделим путь из x_0 в y_0 в N, вдоль которого поток может быть увеличен на одну единицу и такой, что стоимость единичного увеличения потока является минимальной среди стоимостей всех таких путей. Этот путь называется *увеличивающим путем минимальной стоимости* и может быть найден с помощью алгоритма определения кратчайшего пути. Известно, что если h_p является экстремальным и h_{p+1} получается при единичном увеличении потока вдоль увеличивающего пути с минимальной стоимостью, то h_{p+1} является экстремальным [42].

Пусть $X=\{x_1, \ldots, x_{m'}\}$ и $Y=\{y_1, \ldots, y_n\}$. *Увеличивающий путь* относительно допустимого потока h определяется как последовательность различных вершин $\mathrm{v}=x_0, x_{i_1}, y_{j_1}, \ldots, x_{i_r}, y_{j_r}, y_0$, удовлетворяющая следующим

условиям:

$$r \geqslant 1; \quad x_{i_k} \in X \text{ и } y_{j_k} \in Y \text{ для } 1 \leqslant k \leqslant r;$$
$$h(x_0, x_{i_1}) = h(y_{j_r}, y_0) = 0;$$
$$h(x_{i_k}, y_{j_k}) = 0 \text{ для } 1 \leqslant k \leqslant r;$$
$$h(x_{i_{k+1}}, y_{j_k}) = 1 \text{ для } 1 \leqslant k \leqslant r.$$

Следующий алгоритм, на вход которого поступает увеличивающий путь $\nu = z_1, z_2, \ldots, z_r$, определяет новый поток h', отличающийся от исходного h тем, что величина потока увеличена на единицу вдоль ν.

Алгоритм 3.7.

Вход: допустимый поток h и увеличивающий путь $\nu = z_1, \ldots, z_r$ относительно h.

Выход: допустимый поток h, полученный из исходного при единичном увеличении потока вдоль ν.

Метод:

procedure AUGMENT FLOW
begin
$h(y_0, x_0) := h(y_0, x_0) + 1$
$h(z_{r-1}, y_0) := 1$
$h(x_0, z_2) := h(z_2, z_3) := 1$
$k := 4$
while $k \leqslant r-2$ **do**
begin
$h(z_k, z_{k-1}) := 0$
$h(z_k, z_{k+1}) := 1$
$k := k + 2$
end
end AUGMENT FLOW

Легко видеть, что поток, получаемый в результате выполнения процедуры AUGMENT FLOW, является допустимым. Обозначив этот поток через h', а первоначальный — через h, получим

$$\text{cost}(h') = \text{cost}(h) + \sum_{k \in E} c(z_k, z_{k+1}) - \sum_{k \in O} c(z_{k+1}, z_k),$$

где $E = \{2, 4, \ldots, r-2\}$, $O = \{3, 5, \ldots, r-3\}$. Положительные слагаемые получаются в результате увеличения потока от нуля до единицы в дугах (x, y), где $x \in X$, $y \in Y$ и $h(x, y) = 0$. Отрицательные слагаемые соответствуют дугам, в которых мы уменьшаем поток от единицы до нуля.

Увеличивающий путь минимальной стоимости можно отыскать, применяя алгоритм пометок для определения

кратчайшего пути в сети N; при этом функция расстояния α выбирается так, чтобы выделять в N только увеличивающие пути. Пусть $X_1=\{x_i | h(x_i, y_j)=1$ для некоторого $j\}$ и $Y_1=\{y_j | h(x_i, y_j)=1$ для некоторого $i\}$. Пусть функция π ставит в соответствие каждой вершине некоторое неотрицательное целое число. Функция расстояния d определяется следующим образом:

$$d(z, z') = \begin{cases} \pi(z)+c(z, z')-\pi(z'), & z \in X,\ z' \in Y \text{ и } h(z, z')=0, \\ \pi(z)-c(z', z)-\pi(z'), & z' \in X,\ z \in Y \text{ и } h(z', z)=1, \\ \pi(x_0)-\pi(z'), & z=x_0 \text{ и } z' \in X-X_1, \\ \pi(z)-\pi(y_0), & z \in Y-Y_1 \text{ и } z'=y_0, \\ \infty & \text{в остальных случаях.} \end{cases}$$

Функция π введена для того, чтобы обеспечить неотрицательность d. Пусть $\nu=z_1, \ldots, z_r$ — увеличивающий путь относительно h, и пусть h' — поток, получившийся в результате применения процедуры AUGMENT FLOW к ν и h. Длина ν дается выражением

$$l(\nu)=\sum_{i=2}^{r} d(z_{i-1}, z_i).$$

Легко показать, что $l(\nu)=\pi(x_0)-\pi(y_0)+\text{cost}\ (h')-\text{cost}\ (h)$. Следовательно, увеличивающий путь минимальной длины соответствует увеличивающему пути минимальной стоимости. Таким образом, для нахождения кратчайшего пути, увеличивающего поток, мы можем использовать алгоритм расстановки меток. Как было отмечено ранее, роль функции π заключается в обеспечении неотрицательности d, что позволяет нам использовать эффективный алгоритм определения кратчайшего пути.

Алгоритм 3.8 в основном совпадает с алгоритмом, предложенным Эдмондсом и Карпом [40] для определения потоков минимальной стоимости.

Алгоритм 3.8.

Вход: сеть N, соответствующая задаче составления расписания $\mathcal{P}$.

Выход: оптимальный поток h.

Структура данных: X_1 и Y_1 являются множествами, h определено на $X \times Y$, π определена на $X \cup Y \cup \{x_0, y_0\}$, σ и β определены на $X_1 \cup Y \cup \{y_0\}$.

Метод:

1. **procedure** MFT
2. **begin**

```
3. X_1:=∅; Y_1:=∅
4. for each u∈X∪Y∪{x_0, y_0} do π(u):=0
5. for each (x, y)∈X×Y do h(x, y):=0
6. p:=0
7. while p<n do
8. begin
9. call FIND PATH
10. call UPDATE FLOW
11. p:=p+1
12. end
13. end MFT
```

Подпрограммы, использованные в MFT, реализованы алгоритмами 3.9—3.11.

Алгоритм MFT определяет поток только в дугах (x, y), где $x \in X$ и $y \in Y$. Величины потоков в дугах (x_0, x) и (y, y_0) для $x \in X$ и $y \in Y$ легко получаются из h, а поток в обратной дуге (y_0, x_0) равен p. В дальнейшем, говоря о потоке h, вычисленном процедурой MFT, мы подразумеваем, что он уже расширен указанным образом на все дуги в N.

Корректность MFT основана на следующей теореме.

Теорема 3.8. [42]. *Пусть h — экстремальный поток в N и ν — увеличивающий путь минимальной стоимости относительно h. Тогда поток h', полученный в результате применения процедуры* AUGMENT FLOW *к h и к ν, является экстремальным.*

Алгоритм 3.9.

```
procedure FIND PATH
begin
call INITIALIZE LABELS
S:=Y
while S≠∅ do
begin
Выбор y*∈S такого, что σ(y*)≤σ(y) для всех y∈S
S:=S—{y*}
if y*∈Y_1 then
begin
Пусть x* таково, что h(x*, y*)=1
σ(x*):=σ(y*)
β(x*):=y*
for each y∈S do
if σ(x*)+d(x*, y)<σ(y) then
begin
σ(y):=σ(x*)+d(x*, y)
```

$\beta(y) := x^*$
end
end
else
if $\sigma(y^*) + d(y^*, y_0) < \sigma(y_0)$ **then**
begin
$\sigma(y_0) := \sigma(y^*) + d(y^*, y_0)$
$\beta(y_0) := y^*$
end
end
end FIND PATH

Алгоритм 3.10.

procedure INITIALIZE LABELS
begin
for each $y \in Y$ **do**
begin
$\sigma(y) := \min_{x \in (X - X_1)} d(x, y)$
$\beta(y) := x$, где x таково, что $d(x, y) = \sigma(y)$
end
$\sigma(y_0) := \infty$
end INITIALIZE LABELS

Алгоритм 3.11.

procedure UPDATE FLOW
begin
$y := \beta(y_0)$
$x := \beta(y)$
while $x \in X_1$ **do**
begin
$y' := \beta(x)$
$h(x, y) := 1$
$h(x, y') := 0$
$y := y'$
$x := \beta(y)$
end
$h(x, y) := 1$
$Y_1 := Y_1 \cup \{\beta(y_0)\}$
for each $u \in X_1 \cup Y \cup \{y_0\}$ **do** $\pi(u) := \pi(u) + \sigma(u)$
$X_1 := X_1 \cup \{x\}$
end UPDATE FLOW

Оператор 5 процедуры MFT присваивает нулевые значения h, что соответствует экстремальному потоку величины $p=0$ (строка 6). Индуктивное предположение, используемое в строке 7, таково.

П1. *Поток h является экстремальным потоком величины p.*

Другим ключевым фактором, определяющим корректность MFT, является функция расстояния d. Вначале (строка 4) π принимает значение 0 во всех вершинах N. Это означает, что для всех $x \in X$ и $y \in Y$ $d(x, y)=c(x, y)$ и $d(x_0, x)=d(y, y_0)=0$. Индуктивные предположения относительно d, используемые в строке 7 MFT, заключаются в следующем.

П2. *Для всех $x \in X$ и $y \in Y$, если $h(x, y)=0$, то $d(x, y) \geqslant 0$, и если $h(x, y)=1$, то $d(x, y)=0$.*

П3. *Для всех $x \in (X-X_1)$ и $y \in (Y-Y_1)$ $d(x_0, x)=0$ и $d(y, y_0) \geqslant 0$.*

Процедура FIND PATH — это алгоритм определения кратчайшего пути методом расстановки пометок, который позволяет определить кратчайшие пути (относительно d) из x_0 во все вершины множества $X \cup Y \cup \{y_0\}$. Поскольку $d(x_0, x)$ равно нулю для всех $x \in (X-X_1)$, процедура MFT записывает информацию о пути в виде пометок σ и β, которые приписываются только вершинам, входящим в $X_1 \cup Y \cup \{y_0\}$. Так же как и в отношении функции h, мы полагаем, что σ и β распространены на вершины, входящие в $(X-X_1) \cup \{x_0\}$, следующим образом: для всех $x \in (X-X_1) \cup \{x_0\}$ $\sigma(x)=0$ и $\beta(x)=x_0$.

Процедура FIND PATH — не что иное, как хорошо известный метод нахождения кратчайшего пути при неотрицательной функции расстояния. Мы использовали свойства d (П2 и П3) для ускорения процесса расстановки пометок. Первая стадия этого процесса заключается в выполнении процедуры INITIALIZE LABELS. Поскольку $d(x_0, x)=0$ для всех $x \in X - X_1$, мы знаем, что вершины в $X-X_1$ будут просмотрены первыми. Легко видеть, что после просмотра всех вершин в $X-X_1$ мы имеем для всех $y \in Y$

$$\sigma(y) = \min_{x \in X-X_1} d(x, y)$$

и $\beta(y)=x$, где x таково, что $d(x, y)=\sigma(y)$. Более того, все вершины в Y являются кандидатами на просмотр.

С другой стороны, мы использовали преимущества П2 для осуществления двойных шагов помечивания, а именно: если y^* входит в Y_1, то мы знаем, что $d(y^*, x^*)=0$ и что x^* является единственной вершиной, достижимой из y^* относительно d. Поэтому мы немедленно полагаем $\sigma(x^*)=\sigma(y^*)$ и $\beta(x^*)=y^*$ и переходим к просмотру x^*. Подведем итоги предыдущего обсуждения в следующей лемме.

Лемма 3.18. *Если перед вызовом процедуры* FIND PATH *выполняются условия* П2, П3 *и* $p<n$, *то после ее завершения переменные* σ *и* β *удовлетворяют следующим условиям.*

1. *Для всех вершин* u *в* N, *отличных от* x_0, $\sigma(u)$ *есть длина кратчайшего пути (относительно* d*) из* x_0 *в* u.

2. *Для всех вершин* u *в* N, *отличных от* x_0, $\beta(u)$ *является непосредственным предшественником вершины, находящейся на кратчайшем пути (относительно* d*) из* x_0 *в* u.

Процедура UPDATE FLOW использует данные о кратчайшем пути для увеличения потока вдоль увеличивающего пути минимальной стоимости, а также готовит переменные h, X_1, Y_1 и π для использования в следующем цикле оператора **while** в MFT. Легко видеть, что при доказанных свойствах σ и β (лемма 3.18) процедура UPDATE FLOW правильно изменяет поток h, что подтверждает индуктивное предположение П1. Предположения П2 и П3 остаются справедливыми для нового потока благодаря соответствующему определению π, осуществленному в конце процедуры UPDATE FLOW.

Корректность процедуры MFT следует из теоремы 3.8 и справедливости индуктивных предположений П1, П2 и П3.

Теорема 3.9. *Процедура* MFT *правильно вычисляет оптимальный поток* h *в сети* N, *соответствующей задаче составления расписания* $\mathcal{P}$.

Временная сложность MFT в худшем случае может быть определена как $O(mn^2+n^3)$. Происхождение слагаемого n^3 очевидно, а слагаемое mn^2 появляется за счет процедуры INITIALIZE LABELS. На первый взгляд, минимизация для каждого $y \in Y$ по всем x, входящим в $X-X_1$, требует $O(mn^2)$ времени и, поскольку эта минимизация встречается n раз, то общее время всей процедуры составляет $O(mn^3)$. Однако мы можем использовать преимущества структуры матрицы C для того, чтобы снизить общую стоимость процедуры INITIALIZE LABELS до $O(mn^2)$. Идея заключается в том, чтобы заметить, что $c_{ij}<c_{i+m,\,j}$, вследствие чего минимизация для каждого y может быть ограничена рассмотрением только m различных вершин в $X-X_1$.

Алгоритм MFT может быть использован для построения оптимального расписания в случае, когда каждый процессор становится доступным с некоторой задержкой. Пусть $\tau_1, \ldots, \tau_m$ — набор из m неотрицательных чисел, и допустим, что i-й процессор не может быть назначен на выполнение какого-либо задания до момента τ_i для $i=1, \ldots, m$.

Следовательно, в любом заданном расписании завершение каждого задания, выполняемого на i-м процессоре, будет задержано на τ_i единиц времени.

Обозначим через $[\tau_i]$ матрицу размера $m \times n$, имеющую вид

$$[\tau_i] = \begin{bmatrix} \tau_1 & \tau_1 & \dots & \tau_1 \\ \tau_2 & \tau_2 & \dots & \tau_2 \\ \dots & \dots & \dots & \dots \\ \tau_m & \tau_m & \dots & \tau_m \end{bmatrix},$$

и пусть D есть $m' \times n$-матрица вида

$$D = \begin{bmatrix} [\tau_i] \\ [\tau_i] \\ \dots \\ [\tau_i] \end{bmatrix}.$$

Если в качестве матрицы стоимостей транспортной сети N взять матрицу $C+D$ (вместо C), то оптимальный поток для N определит оптимальное расписание относительно $[\tau_{ij}]$ и задержек $\tau_1, \dots, \tau_m$.

3.12. Специальные случаи

Во многих случаях нет необходимости решать задачу в такой общей постановке, как в предыдущем параграфе. Более часто может встречаться ситуация, когда m процессоров, в основном, имеют одинаковые характеристики, за исключением их относительного быстродействия; таковы, например, печатающие устройства. Один из способов построения модели в этой ситуации состоит в том, чтобы поставить в соответствие каждому процессору положительное число $\gamma_i \geqslant 1$ и каждому заданию T_j — время выполнения $\tau_j > 0$. Время, необходимое i-му процессору для обслуживания задания T_j, равно $\gamma_i \tau_j$ [28].

Очевидно, что, обозначив $\tau_{ij} = \gamma_i \tau_j$, мы можем использовать MFT для определения оптимального расписания. Однако более экономный метод определения оптимального расписания базируется на следующей лемме.

Лемма 3.19. *Пусть* $a_1 \geqslant a_2 \geqslant \dots \geqslant a_n$ *и* $b_1, \dots, b_n$ — *две последовательности. Перестановка* $\alpha = \alpha_1, \dots, \alpha_n$ *минимизирует величину*

$$\sum_{i=1}^{n} a_i b_{\alpha_i}$$

в том и только в том случае, когда $b_{\alpha_1} \leqslant b_{\alpha_2} \leqslant \dots \leqslant b_{\alpha_n}$.

Доказательство. Рассмотрите перестановку местами двух членов последовательности.

Построим матрицу Γ размера $m \times n$:

$$\Gamma = \begin{bmatrix} \gamma_1 & 2\gamma_1 & 3\gamma_1 & \dots & n\gamma_1 \\ \gamma_2 & 2\gamma_2 & 3\gamma_2 & \dots & n\gamma_2 \\ \dots & \dots & \dots & \dots & \dots \\ \gamma_m & 2\gamma_m & 3\gamma_m & \dots & n\gamma_m \end{bmatrix}.$$

Обозначим (i, j)-й элемент Γ через γ_{ij}, и пусть $\gamma_{i_1 j_1}, \dots, \gamma_{i_{m'} j_{m'}}$ есть такая последовательность элементов матрицы Γ ($m' = mn$), что

$$\gamma_{i_1 j_1} \leqslant \gamma_{i_2 j_2} \leqslant \dots \leqslant \gamma_{i_{m'} j_{m'}}.$$

Предположим также, что величины τ удовлетворяют условию

$$\tau_1 \geqslant \tau_2 \geqslant \dots \geqslant \tau_n.$$

«Объединим в пару» τ_k с $\gamma_{i_k j_k}$ для $k = 1, \dots, n$. Если τ_k объединено с $\gamma_{i_k j_k}$, то T_k назначается на процессор i_k и имеется $j_k - 1$ заданий, следующих за T_k на процессоре i_k.

Доказательство корректности такого подхода основано на том, что некоторые расписания могут быть интерпретированы как объединение в пары величин τ с элементами матриц Γ. Более того, лемма 3.19 дает способ оптимального объединения в пары, которое связывает τ_k с $\gamma_{i_k j_k}$. Наконец, мы видим, что это оптимальное объединение в пары дает правильное расписание, поскольку величины γ_{ij} появляются в последовательности $\gamma_{i_1 j_1}, \dots, \gamma_{i_{m'} j_{m'}}$ раньше, чем $\gamma_{i, j+1}$.

Используя эффективный алгоритм организации очередей с приоритетами [3], мы можем построить алгоритм для составления оптимальных расписаний, временная сложность которого для наиболее трудного случая определяется как $O(n \log n + n \log m + m)$. Здесь $O(n \log n)$ представляет собой стоимость сортировки времен выполнения, а $O(n \log m + m)$ есть стоимость использования очереди с приоритетами для получения величии $\gamma_{i_1 j_1}, \dots, \gamma_{i_n j_n}$.

Рассматривая еще более частный случай, возьмем значения γ_i равными друг другу. Это соответствует случаю идентичных машин, для которого оптимальные расписания известны под общим названием «первым обслуживается имеющий минимальное время обслуживания» (SPT). Класс SPT-расписаний довольно широк, и их свойства, связанные с максимальным временем завершения заданий, хорошо изучены. Обзор результатов в этой области можно найти в п. 1.5.2.3 гл. 1.

ГЛАВА 4

СЛОЖНОСТЬ ЗАДАЧ УПОРЯДОЧЕНИЯ

4.1. Введение

В области теории расписаний, так же как и во многих других областях, мы сталкиваемся с многочисленными задачами, которые могут быть решены относительно легко, в то время как другие похожие задачи оказываются чрезвычайно трудными. Например, как показано в гл. 2, оптимальное расписание на двух процессорах для системы из n заданий с единичными временами выполнения может быть получено за $O(n^2)$ шагов. Однако оказывается, что такая же задача с тремя процессорами требует для своего решения времени, экспоненциально зависящего от n. Это означает, что мы должны испытать практически все расписания, прежде чем найдем наилучшее.

В сущности, в настоящее время отсутствуют способы доказательства того, что для решения задачи упорядочения на трех процессорах для системы заданий с единичными временами выполнения или других задач такого типа действительно требуется экспоненциальное время. Однако мы можем сделать следующее: показать, что существует большой класс задач, называемых NP-полными, которые либо все разрешимы за полиномиальное время, либо все неразрешимы за полиномиальное время. В класс NP-полных задач входят многие хорошо известные задачи [86], такие, как задача коммивояжера или общая задача упорядочения n работ на m процессорах, для которых математики и специалисты в области информатики десятилетиями безуспешно ищут решения, требующие менее чем экспоненциального времени. Таким образом, имеются серьезные основания считать, что ни одна из NP-полных задач неразрешима за полиномиальное время.

Вследствие этого становится важным уметь определять, является ли задача NP-полной, потому что в случае положительного ответа нам, скорее всего, понадобится эвристический метод, позволяющий получить приближения к

оптимальному решению. Неудивительно поэтому, что эвристические методы для задач упорядочения привлекли к себе такое внимание (см. гл. 5 и 6). Если же задача разрешима за полиномиальное время, то мы, скорее всего, попытаемся найти точное решение. Известно, что решить задачу, сложность которой определяется как $O(n^{10})$, в действительности не легче, чем задачу сложности $O(2^n)$, по крайней мере для разумных значений n. Однако сложность большинства встречающихся на практике задач, разрешимых за полиномиальное время, оказывается, растет как $O(n^2)$ или в крайнем случае как $O(n^3)$.

Таким образом, чисто эмпирически мы можем считать задачи, разрешимые за полиномиальное время, легкорешаемыми и искать для них точные решения, а NP-полные или еще более сложные задачи — относить к классу труднорешаемых, не подлежащих точному решению. Как читатель, наверное, догадывается, существует обширная область задач, NP-полнота (или еще большая сложность) которых не установлена, а полиномиальные алгоритмы решения также неизвестны. Примерами являются трехпроцессорная задача упорядочения с единичными временами выполнения заданий, а также задача составления расписания по критерию минимума среднего взвешенного времени прохождения при произвольном отношении предшествования (см. табл. 1.1 и 1.2).

4.2. Задачи и полиномиальная сводимость

В качестве отправной точки определим формально задачу как вопрос, на который следует ответить «да» или «нет» *). Обычно вопрос имеет «параметры», т. е. свободные переменные. Совокупность выбранных значений параметров называется входом задачи. Размер входа определяется как длина строки, используемой для представления значений параметров. Алфавит, используемый в строках, описывающих входы, должен быть фиксированным для данной задачи и не зависеть от входов.

Пример 1. Мы можем сформулировать общую задачу составления расписаний следующим образом: «Существует ли расписание на m процессорах для множества заданий $T_1, T_2, \ldots, T_n$ с отношением предшествования $<$ и временами выполнения $\tau_1, \tau_2, \ldots, \tau_n$ такое, что его время за-

*) Такие задачи называются обычно задачами распознавания. (*Прим. перев.*)

вершения не превосходит ω?» Параметрами здесь являются n, m, ω, $\prec$, τ_1, τ_2, . . ., τ_n. Входы этой задачи удобно задавать с помощью следующих пяти символов: 0, 1, открывающая и закрывающая скобки и запятая. Числа n, m и ω могут быть представлены в двоичной записи. Разделенные запятыми, они образуют начало строки. Затем следуют пары вида (i, j), где i и j являются двоичными числами. Пара (i, j) присутствует в строке в том и только в том случае, когда, в соответствии с отношением предшествования $\prec$, задание T_i предшествует заданию T_j. Наконец, записываем в двоичном представлении n времен выполнения $\tau_1, \tau_2, \ldots, \tau_n$, разделенных запятыми.

Для примера положим $n=5$, $m=2$, $\omega=8$, $\prec$ зададим как $1\prec4\prec5$ и $2\prec3$, а $\{\tau_i\}$ определим таблицей

i	1	2	3	4	5
τ_i	4	5	4	2	1

Тогда строка, задающая этот вход, есть

m n ω $1\prec4$ $4\prec5$ $2\prec3$ τ_1 τ_2 τ_3 τ_4 τ_5
101,10,1000(1,100)(100,101)(10,11)100,101,100,10,1

Ответ в этом случае гласит: «Нет».

Естественно, возникает вопрос, не теряем ли мы в общности, формулируя задачу в виде вопроса, требующего ответа «да» или «нет», по сравнению с постановкой задачи оптимизации. Другими словами, можно предположить, что задача, сформулированная в примере 1 в виде вопроса, требующего ответа «да» или «нет», может быть решена за полиномиальное время, а соответствующая оптимизационная задача вида «заданы n, m, $\prec$ и τ; каково минимальное значение t, при котором задача распознавания со входом n, m, t, $\prec$, τ_1, . . ., τ_n имеет ответ «да»?» может не иметь полиномиального решения. Оказывается, в этом случае такая ситуация невозможна; невозможна она и для любой другой известной NP-полной задачи.

Для доказательства этого утверждения предположим, что существует полиномиальный алгоритм решения нашей задачи упорядочения, заданной в виде задачи распознавания. Пусть алгоритм требует времени $p(s)$ на входе длины s.

Для минимизации t при заданных n, m $<$ и наборе τ*) мы могли бы использовать следующий алгоритм. Во-первых, если $m=0$, то такое t не существует, и задача решена. Если $m\geqslant 1$, то минимальное значение t не превосходит $\sum_{i=1}^{n}\tau_i$. Поскольку в строке, представляющей рассматриваемый вход оптимизационной задачи, записаны в двоичном представлении все τ_i, мы видим, что длина входа s по меньшей мере равна $\sum_{i=1}^{n}\log_2\tau_i$. Отсюда следует, что оптимальное значение t не превышает 2^s.

Наш алгоритм будет обращаться к гипотетическому алгоритму решения задачи распознавания s раз, каждый раз с другим значением t. Мы начинаем с получения ответа для входа n, m, 2^{s-1}, $<$, τ_1, . . ., τ_n задачи распознавания. Если получен ответ «да», то решаем задачу для входа n, m, 2^{s-2}, $<$, τ_1, . . ., τ_n; если ответ «нет», то решаем задачу для входа n, m, $3(2^{s-2})$, $<$, τ_1, . . ., τ_n. Будем продолжать в том же духе, при каждом обращении сокращая вдвое область, в которой обязательно находится оптимальное значение t.

Длины строк, представляющих входы каждой из рассмотренных задач распознавания, не превосходят $2s+2$, поскольку эти строки получаются из строк, представляющих входы оптимизационной задачи, добавлением числа t. Двоичное представление t требует не более $s+1$ битов плюс запятая, поскольку $t\leqslant 2^s$. Таким образом, каждое обращение к гипотетическому алгоритму для задачи распознавания потребует времени не более чем $p(2s+2)$, а поскольку количество обращений не превосходит s, то общее время составит не более чем $sp(2s+2)$. Время работы нашего алгоритма есть это время плюс время, необходимое для составления строк, участвующих в различных обращениях, и принятия решения о том, какой вход рассматривать следующим. Поскольку вычисления, необходимые для этого, просты, читатель может убедиться в том, что эта процедура требует не более cs^2 шагов, где c — некоторая постоянная. Таким образом, наш алгоритм затрачивает время $sp(2s+2)+cs^2$. Поскольку p — полином, полиномом будет и $sp(2s+2)+cs^2$. Следовательно, наш алгоритм также полиномиален.

---

*) Будем считать, что величины n, m, $<$, τ_1, . . ., τ_n в оптимизационной постановке задачи упорядочения закодированы так же, как и в примере 1, а ω опущена.

Предыдущее доказательство является хорошим примером, иллюстрирующим общий принцип, используемый в теории NP-полных задач,— полиномиальное сведение одной задачи к другой. Вход одной задачи (в данном случае — оптимизации) был преобразован в один или несколько входов другой задачи (упорядочения в форме задачи распознавания), и для решения этих задач был применен гипотетический алгоритм решения второй задачи. Полученные решения были использованы для решения первой задачи. Существенным моментом являлось то, что вход первой задачи был преобразован во входы второй задачи быстро — за полиномиальное время; следовательно, входы второй задачи оказались не слишком длинными, во всяком случае не вышли за пределы, определяемые полиномом от длины первоначального входа. Таким образом, мы приходим к следующему определению.

Мы говорим, что задача $\mathcal{P}_1$ *полиномиально сводится* к задаче $\mathcal{P}_2$, и записываем это как $\mathcal{P}_1 \propto \mathcal{P}_2$, если задача $\mathcal{P}_1$ может быть решена алгоритмом A, имеющим следующие свойства:

1. A вызывает подпрограмму A' для решения задачи $\mathcal{P}_2$ нуль или более раз.

2. Исключая время, затраченное на обращение к A', алгоритм A затрачивает время, полиномиально зависящее от длины того входа $\mathcal{P}_1$, который является входом для A.

Сделаем следующие замечания относительно полиномиальной сводимости. Если $\mathcal{P}_1 \propto \mathcal{P}_2$ и для $\mathcal{P}_2$ есть полиномиальный алгоритм, то такой же алгоритм имеется и для $\mathcal{P}_1$. Для уяснения этого обстоятельства предположим, что алгоритм A' требует $p_1(n)$ единиц времени. Алгоритм A затрачивает время $p_2(n)$, исключая обращения к A', где p_2—некоторый полином. Тогда, очевидно, происходит не более $p_2(n)$ вызовов A', и ни один из аргументов при обращении к A' не имеет длину, превышающую $p_2(n)$, потому что больший аргумент просто не может быть записан за время $p_2(n)$. Таким образом, каждый вызов A' занимает не более $p_1(p_2(n))$ единиц времени. Поскольку таких вызовов не более $p_2(n)$, общее время, затрачиваемое алгоритмом A при длине входа n, не превосходит $p_2(n) + p_2(n)\, p_1(p_2(n))$, а это—полином, если полиномами являются p_2 и p_1. Заметим, что время, затрачиваемое алгоритмом A, может быть значительно больше времени, затрачиваемого алгоритмом A', но оно тем не менее остается полиномиальным, если таковым является время, затрачиваемое A'.

В дальнейшем мы покажем, что существует много разных NP-полных задач $\mathcal{P}$, среди них и различные задачи упорядочения, которые имеют то свойство, что любая задача из огромного класса задач (называемого **NP**) полиномиально сводится к $\mathcal{P}$. Таким образом, если бы некоторая NP-полная задача имела полиномиальный алгоритм, то такой алгоритм имели бы все задачи в **NP**.

4.3. Модель вычислительной машины

Теперь нам следует выбрать модель вычислительной машины. Мы построим модель, которая является простой, но достаточно общей, чтобы читатель убедился, что с ее помощью можно моделировать любые реальные машины. Начнем с того, что любая вычислительная машина, включая все ее регистры, устройства управления, а также основную и внешнюю память, может быть представлена в любой момент времени в виде последовательности битов, называемой состоянием. Конечное фиксированное количество битов служит для представления содержимого основной памяти и всех цепей, обычно называемых «управлением». Только число битов, хранящихся на внешних накопителях (например, на лентах), может рассматриваться как потенциально неограниченное, поскольку, хотя каждая лента имеет конечную емкость, программа может обращаться к бесконечному числу лент; все ленты, когда-либо использованные программой, могут считаться частью состояния, так как даже если какая-то лента в данный момент снята с устройства, то она может быть вновь поставлена позднее по требованию программы. Введем ряд допущений относительно работы машины.

1. Состояние машины изменяется в фиксированные моменты времени, называемые *шагами*. В промежутках между шагами состояние остается неизменным.

2. Значение каждого бита после выполнения шага зависит от значений некоторого фиксированного набора битов непосредственно перед шагом.

3. При любом целом n число битов, используемых программой, когда ее вход имеет длину n, не зависит от самой вводимой информации. Более того, существует константа c такая, что если программа выполняется за t шагов, то в машине может быть использовано не более ct битов.

Условие 1 не требует пояснений. Условие 2 сначала может показаться не очевидным. Начнем объяснение с того, что обозначим каждый бит, находящийся на внешнем на-

копителе, таком, как лента, в соответствии с положением этого бита относительно считывающей головки накопителя. Так, если лента сдвигается влево на один шаг, то значение бита зависит только от значения на предыдущем шаге его первого соседа по дорожке. Таким образом, в общем случае значение бита на внешнем накопителе может быть определено путем обследования конечной основной памяти, чтобы увидеть, требуется ли перемещение накопителя (в случае непрерывно вращающихся диска или барабана в этом нет необходимости), и определения значения соседнего бита, если такое перемещение требуется. Значения всех битов в основной памяти и устройстве управления вычислительной машины зависят только от них самих и от значений тех битов на внешних накопителях, которые в данный момент находятся под считывающими головками. Заметим, что объем основной памяти хотя и велик, но всегда ограничен.

Условие 3 еще более тонко. Мы можем представить себе вырожденную программу, которая при считывании входа в виде n десятичных цифр требует, чтобы был установлен i-й пакет дисков, где i есть введенное десятичное число. Поскольку i изменяется в диапазоне от 0 до $10^n - 1$, число битов памяти, используемой при некотором входе, имеющем длину n, определяется экспонентой от n, хотя программа требует очень малого времени для своего выполнения,— разумеется, полиномиально (в действительности линейно) зависящего от n.

Мы хотели бы узнать, всегда ли, когда программа выполняется за полиномиальное время, объем памяти также подчиняется полиномиальной зависимости. К счастью, работу любой полиномиальной программы на входе длины n мы можем промоделировать за время, не превосходящее квадрата времени ее выполнения, использовав при этом фиксированный набор ячеек памяти. Количества времени и памяти, затрачиваемые моделирующей программой, могут достигать квадрата времени выполнения моделируемой программы, но если последняя полиномиальна по времени, то первая будет полиномиальной и по времени, и по памяти. Здесь мы не будем вдаваться в детали моделирования. Дальнейшие сведения можно найти в статье Кука и Решкова [31] и книге Ахо, Хопкрофта и Ульмана [3]. Читатель и сам может убедиться в том, что только в вырожденных случаях может появиться потребность использовать много различных непересекающихся областей памяти при разных входах одинаковой длины.

4.4. Недетерминированные вычисления

Выбрав подходящую модель детерминированной вычислительной машины, мы теперь добавим в нее чисто абстрактное свойство — недетерминированность. Представим себе, что в обычной машине имеется команда

CHOICE L1, L2.

В результате выполнения этой команды одновременно образуются две копии машины, находящиеся в ее текущем состоянии. Затем каждая копия изменяет свое состояние: одна — выполняя следующей команду L1, другая — команду L2. Таким образом, после n-кратного выполнения команды CHOICE образуются 2^n копий машины, причем все они могут быть в разных состояниях.

Недетерминированная машина может быть использована (разумеется, абстрактно) для решения задач распознавания, если мы объявим, что ответ, выданный машиной, будет считаться как «да» в том случае, если одна или более копий дают этот ответ, независимо от того, сколько копий выдает ответ «нет».

Мы можем абстрактно определить время, затрачиваемое недетерминированной машиной, как длину самой длинной последовательности изменений состояния.

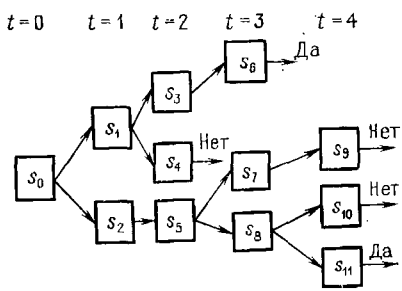


Рис. 4.1.

Пример 2. Рассмотрим работу недетерминированной вычислительной машины, изображенную на рис. 4.1. В момент времени $t=0$ машина начинает работу, имея состояние s_0. Она выполняет команду CHOICE и образует две копии в состояниях s_1 и s_2 в момент $t=1$. В состоянии s_2 машина выполняет команду, отличную от CHOICE, переходя в состояние s_5. В противоположность этому, в состоянии s_1 команда CHOICE выполняется, и двумя новыми состояниями оказываются s_3 и s_4. В s_4 нет следующего состояния, и вырабатывается ответ «нет» (т. е. не печатается ответ «да»). Из состояния s_3 происходит переход к s_6 в момент $t=3$. На последнем шаге, вероятно, печатается «да», потому что для s_6 показан этот ответ. Мы теперь знаем, что эта недетерминированная машина дает ответ «да» для

любого входа в s_0. Мы можем проследить изменения состояния до момента $t=4$, когда обнаруживается, что все копии вышли на оператор вывода и дальнейшие переходы невозможны. Это означает, что все копии машины остановились. Мы говорим, таким образом, что вычислительная машина, начав работу в состоянии s_0, затрачивает 4 единицы времени.

Следует подчеркнуть, что время, «затрачиваемое» недетерминированной машиной, может быть намного меньше времени, необходимого для ее детерминированного моделирования. Как было отмечено, недетерминированная машина, затрачивающая время n, может достигнуть (при условии, что выполняются только команды CHOICE) 2^n различных состояний. Обычное моделирование такого устройства на реальной машине займет по меньшей мере 2^n единиц времени.

Таким образом, представляется, что существует по крайней мере некоторое количество задач, которые могут быть решены на недетерминированной машине за полиномиальное время, но требуют экспоненциального времени на реальной машине, хотя никто не смог доказать это утверждение (или, наоборот, показать, что недетерминированные машины можно эффективно моделировать на реальных машинах).

Мы можем определить класс **NP** как совокупность всех задач, которые могут быть решены на недетерминированной машине за полиномиальное время. Определим **P** как класс задач, решаемых детерминированной машиной за полиномиальное время. Назовем задачу $\mathcal{P}$ *NP-полной*, если любая задача из **NP** полиномиально сводится к $\mathcal{P}$.

Из только что данного определения следует, что если некоторая *NP*-полная задача имеет полиномиальный детерминированный алгоритм, то все задачи в **NP** также имеют такой алгоритм (т. е. **P** = **NP**). Таким образом, если одна *NP*-полная задача имеет полиномиальное решение, то и все остальные — тоже. Поскольку в класс *NP*-полных задач входят такие задачи, как задача коммивояжера, задача о ранце, общая задача составления расписания, и ни для одной из них не найден полиномиальный алгоритм (и притом не из-за недостатка попыток), мы имеем основания предполагать, что все *NP*-полные задачи требуют при детерминированном подходе более чем полиномиального времени, а в действительности, по-видимому, требуют экспоненциального времени.

Пример 3. Покажем, что общая задача составления расписания входит в **NP**. В действительности задача яв-

ляется NP-полной, хотя для того, чтобы это показать, требуется приложить дополнительные усилия. Алгоритм для недетерминированной машины может быть построен следующим образом. Допустим, что нам дано n заданий $T_1, T_2, \ldots, T_n$, которые должны быть упорядочены на m процессорах, и что ограничения предшествования $<$ заданы в виде списка пар $T_i < T_j$. Предположим также, что времена выполнения $\tau_1, \tau_2, \ldots, \tau_n$ заданы в виде списка двоичных чисел. Наконец, будем полагать, что предельное время ω также задано в двоичном представлении.

1. Многократно применяя операцию CHOICE, назначим каждому заданию T_i процессор P_{j_i} и время начала выполнения $t_i < \omega$. Тем самым мы угадываем расписание.

2. Для каждых k и l, $1 \leqslant k < l \leqslant n$, убедимся в том, что либо $P_{j_k} \neq P_{j_l}$, либо $t_k + \tau_k \leqslant t_l$, либо $t_l + \tau_l \leqslant t_k$. Тем самым мы убеждаемся в том, что никакие два задания не назначены на один и тот же процессор в одно и то же время.

3. Для каждого k, $1 \leqslant k \leqslant n$, проверим, что выполняется $t_k + \tau_k \leqslant \omega$. Тем самым мы убеждаемся в том, что все работы завершаются до ω-го шага по времени.

Для того чтобы определить время, затрачиваемое приведенным недетерминированным алгоритмом, установим нижнюю границу длины входа N, предполагая, что вход закодирован так, как в примере 1. Поскольку ω, n, m и значения τ должны быть представлены в двоичной записи и разделены запятыми и поскольку имеется n времен выполнения, мы видим, что

$$N \geqslant n + \log_2 m + \log_2 \omega + \sum_{i=1}^{n} \log_2 \tau_i. \qquad (1)$$

Оценим теперь сверху время T, затрачиваемое алгоритмом. Будем с явным запасом считать, что для записи сложения или вычитания k-разрядных чисел требуется k единиц времени. Тогда шаг 1 требует не более $O(n(\log_2 \omega + \log_2 m))$ времени для записи номера процессора и времени начала выполнения каждого из n заданий. Заметим, что к окончанию выполнения шага 1 существует огромное число копий машины. Однако длина любого пути в дереве копий (как на рис. 4.1) сравнительно невелика.

На шаге 2 рассматривается не более n^2 (в действительности $n(n-1)/2$) пар (k, l). Арифметические действия по выполнению трех проверок на шаге 2 требуют не более

$O(\log_2\tau_k+\log_2\tau_l+\log_2\omega+\log_2 m)$ времени. Таким образом, легко показать, что общее время, затрачиваемое на выполнение шага 2, составляет не более $O\left(n^3\left(n+\log_2\omega+\right.\right.$ $\left.\left.+\log_2 m+\sum_{i=1}^{n}\log_2\tau_i\right)\right)$. Наконец, шаг 3 занимает, очевидно, $O\left(n\log_2\omega+\sum_{i=1}^{n}\log_2\tau_i\right)$ времени.

Следовательно, доминирующим является время, затрачиваемое на выполнение шага 2, и мы получаем

$$T\leqslant cn^2\left(n+\log_2\omega+\log_2 m+\sum_{i=1}^{n}\log_2\tau_i\right), \qquad (2)$$

где c — некоторая константа. Из (1) и (2) получаем $T\leqslant cN^3$. Таким образом, выражение (1) дает нижнюю оценку длины входа, а (2) — верхнюю оценку времени работы, и мы можем быть уверены в том, что время работы нашей недетерминированной программы решения общей задачи составления расписаний определяется полиномом не выше третьей степени от длины входа. Следовательно, общая задача составления расписаний входит в класс **NP**.

4.5. *NP*-полная задача

Теперь мы можем показать, что *NP*-полной является *задача о выполнимости булевских выражений*, которая для заданного булевского выражения состоит в следующем: существует ли такая подстановка значений истинности 0 и 1 (ложь и истина) в пропозициональные переменные, при которой выражение оказывается истинным?

Напомним, что для доказательства *NP*-полноты задачи о выполнимости необходимо показать, что наличие для нее детерминированного полиномиального алгоритма влечет за собой наличие детерминированного полиномиального алгоритма для любой задачи $\mathcal{P}$, входящей в **NP**. Это означает, что $\mathcal{P}$ полиномиально сводится к задаче о выполнимости. Заметим, что полиномиальное время, необходимое для сведения $\mathcal{P}$ к задаче о выполнимости, не обязано быть одинаковым для всех $\mathcal{P}$.

Теорема 4.1. *Задача о выполнимости булевских выражений является NP-полной.*

Доказательство. Заметим сначала, что задача о выполнимости входит в **NP**. Используя процедуру

CHOICE, мы можем сгенерировать все возможные наборы значений истинности логических переменных. Для каждого из 2^n способов приписывания переменным значений «истина» или «ложь» создается одна копия машины. На это требуется только $O(n)$ шагов, поскольку процесс является недетерминированным. Теперь значение булевского выражения может быть вычислено каждой копией машины на основе своего набора значений истинности. Читатель должен без труда построить детерминированный алгоритм, осуществляющий это вычисление за $O(N^2)$ шагов, где N — длина исходного булевского выражения (заметим, что $n \leqslant N$) *). На самом деле существует алгоритм со сложностью $O(N)$. Таким образом, недетерминированная машина может дать ответ на вопрос, является ли входное выражение выполнимым за время, полиномиально зависящее от длины этого выражения. Заметим, что недетерминированная машина дает ответ «да», если хотя бы одна ее копия дает такой ответ. Это свойство является принципиальным, так же как способность машины порождать свои копии.

Теперь мы должны проделать трудную часть работы, показав, что любая задача $\mathscr{P}$, входящая в **NP**, полиномиально сводится к задаче о выполнимости. Напомним сформулированные в § 4.3 допущения, связанные с нашей моделью машины. Предположим, что нам дана недетерминированная машина, в которую загружена программа решения задачи $\mathscr{P}$, и что эта программа затрачивает время $p_{\mathscr{P}}(n)$ при длине входа n, где $p_{\mathscr{P}}$ — некоторый полином. Допустим, что для решения задачи о выполнимости задан гипотетический полиномиальный детерминированный алгоритм A, скажем, со сложностью $p(n)$. Тогда мы можем построить следующий полиномиальный детерминированный алгоритм $A_{\mathscr{P}}$ для решения задачи $\mathscr{P}$: для любого заданного входа I, имеющего длину n, $A_{\mathscr{P}}$ строит за полиномиальное время булевское выражение B_I, которое является выполнимым в том и только в том случае, когда недетерминированная машина дает ответ «да» при входе I. Выражение B_I будет иметь порядка $[p_{\mathscr{P}}(n)]^2$ переменных X_{ij},

*) Мы подразумеваем здесь, что «длина» булевского выражения есть число символов $+$, $\cdot$, $\neg$, скобок и пропозициональных переменных в строке, определяющей выражение. В действительности, поскольку все задачи должны быть записаны в некотором фиксированном алфавите, пропозициональные переменные должны быть занумерованы и закодированы в двоичной или иной форме. Таким образом, мы можем даже показать, что если в выражении встречается n переменных, то $n \log_2 n \leqslant N$.

$1 \leqslant i \leqslant cp_{\mathcal{P}}(n)$, $0 \leqslant t \leqslant p_{\mathcal{P}}(n)$, где c—верхняя граница отношения числа использованных битов к числу выполненных шагов. Постоянная c существует согласно допущениям нашей модели. Интуитивно ясно, что $X_{it}=1$ в том и только в том случае, когда i-й бит, используемый программой, равен 1 в момент t. Заметим, что, поскольку программа недетерминированно выполняется в течение не более $p_{\mathcal{P}}(n)$ шагов, нам нужно заботиться только о тех моментах, когда $0 \leqslant t \leqslant p_{\mathcal{P}}(n)$. Мы также предполагаем, что на каждом шаге используется не более c битов, при этом число шагов не превышает $p_{\mathcal{P}}(n)$. Таким образом, имеется не более $cp_{\mathcal{P}}(n)$ битов, которые следует принимать во внимание. Из наших допущений следует, что машина использует одни и те же биты при любом входе длины n.

Теперь выражение B_I может быть записано в виде булевского произведения («и») трех выражений C, D и E, которые утверждают следующее.

1. Выражение C утверждает, что начальная конфигурация машины является корректной. Это означает, что на вход подано I, программа присутствует в машине и счетчик команд (адресов ячеек) указывает на первую команду, а все неиспользуемые разряды установлены в 0. Таким образом, C просто есть булевское произведение переменных X_{i0} или $\neg X_{i0}$ для всех i, $1 \leqslant i \leqslant cp_{\mathcal{P}}(n)$, и C имеет длину $O(p_{\mathcal{P}}(n))$.

2. Выражение D означает, что в каждый момент $t \geqslant 1$ содержимое каждого бита i такое, каким оно должно быть, исходя из значений в момент $t-1$ фиксированного числа битов, от которых зависит данный i-й бит. Если команда, выполненная машиной в момент $t-1$, не является командой CHOICE, то значение бита i в момент t однозначно определяется содержимым соответствующих битов в момент $t-1$. Если же в момент $t-1$ выполняется команда CHOICE, то изменяется только счетчик адресов, и в момент t имеются два разрешенных состояния фиксированного числа битов этого счетчика.

Таким образом, мы можем записать D в виде произведения выражений D_t для $1 \leqslant t \leqslant p_{\mathcal{P}}(n)$ и D_{it} для всех i и t, $1 \leqslant t \leqslant p_{\mathcal{P}}(n)$, $1 \leqslant i \leqslant cp_{\mathcal{P}}(n)$. При этом D_{it} утверждает, что либо в момент $t-1$ выполняется команда CHOICE, либо нет, и тогда X_{it} сохраняет свое значение в момент t; D_t утверждает, что либо в момент $t-1$ выполняется команда, отличная от CHOICE, либо выпол-

няется CHOICE, и тогда счетчик адресов в момент t имеет одно из двух возможных значений, а все остальные биты имеют в момент t те же значения, которые они имели в момент $t-1$. В соответствии с нашими допущениями о машине каждое из выражений D_{it} может быть записано в виде строки фиксированной длины, не зависящей от n. Таким образом, их общая длина составляет $O\left(p^2_{\mathcal{P}}(n)\right)$, и мы можем записать их за это время с помощью очевидного алгоритма. Каждое выражение D_t требует строки длиной $O\left(p_{\mathcal{P}}(n)\right)$ и может быть легко записано. Для этого вполне достаточно $O\left(p^2_{\mathcal{P}}(n)\right)$ времени и памяти. Следовательно, все выражение D может быть записано за время (и в памяти), определяемое как $O\left(p^2_{\mathcal{P}}(n)\right)$.

3. Выражение E утверждает, что последовательность конфигураций, описываемых переменными X_{it}, заканчивается печатью «да». Без потери общности предположим, что имеется определенный бит j, который устанавливается в 1 в том и только в том случае, когда вырабатывается ответ «да». Таким образом, E является попросту суммой выражений X_{jt} для $0 \leqslant t \leqslant p_{\mathcal{P}}(n)$. Ясно, что длина E есть $O\left(p_{\mathcal{P}}(n)\right)$.

Легко проверить, что «да» является ответом в задаче $\mathcal{P}$ при входе I в том и только в том случае, когда выражение $B_I = C \cdot D \cdot E$ выполнено. Выполняющий набор значений истинности может быть легко определен по вычислению, приводящему к печати ответа «да», и такое вычисление может быть легко определено по набору значений величин X_{it}, при котором выполняется B_I. Мы убедились в том, что B_I имеет длину $c_1 p^2_{\mathcal{P}}(n)$, где c_1 — некоторая константа, и что B_I может быть построено по I за время $O\left(p^2_{\mathcal{P}}(n)\right)$. Теперь мы можем применить гипотетический алгоритм A к B_I и определить, выполнимо ли B_I. Поскольку A требует $p\left(c_1 p^2_{\mathcal{P}}(n)\right)$ времени при входе B_I, то весь этот алгоритм требует $O\left(p^3_{\mathcal{P}}(n) + p\left(c_1 p^2_{\mathcal{P}}(n)\right)\right)$ времени и является детерминированным, потому что A и способ построения B_I по I являются детерминированными. Поскольку p и $p_{\mathcal{P}}$ представляют собой полиномы, то время выполнения $A_{\mathcal{P}}$, очевидно, полиномиально зависит от n, т. е. от длины входа I.

Таким образом, мы показали, что произвольная задача $\mathcal{P}$, входящая в **NP**, полиномиально сводится к задаче о выполнимости.

В действительности мы можем показать большее, чем утверждается в теореме 4.1. Определим *задачу о выполнимости конъюнктивной нормальной формы* (КНФ) как вопрос, является ли выполнимым булевское выражение, представленное в конъюнктивной нормальной форме (произведение сумм литералов*)). Определим *задачу о k-выполнимости* как вопрос, является ли выполнимым булевское выражение в КНФ, содержащее точно k литералов в каждом сомножителе (выражение в k-КНФ).

Пример 4. Выражение $(x+y+z)(\neg x+w+\neg z)$ записано в 3-КНФ. Выражение $(x+y+z)(\neg w+\neg x)(z+\neg y)$ записано в КНФ, но не в k-КНФ ни при каком k.

Нетрудно показать, что задача о 2-выполнимости относится к классу **P**; в действительности ее можно решить детерминированным алгоритмом за линейное время. Однако оказывается, что ситуация меняется при $k \geqslant 3$, поскольку мы можем доказать следующее.

Теорема 4.2. *Задача о 3-выполнимости является NP-полной.*

Доказательство. Вначале заметим, что выражение B_I, построенное при доказательстве теоремы 4.1, представлено почти в КНФ. В самом деле, выражение C представлено в КНФ, поскольку оно является произведением многих «сумм», состоящих из одного литерала. Выражение E представлено в КНФ потому, что оно является «произведением» одной суммы литералов. Выражение D является произведением выражений D_t и D_{it}, введенных при доказательстве теоремы 4.1. Каждое из этих выражений имеет фиксированную длину, не зависящую от длины входа I. Следовательно, используя законы булевой алгебры, мы можем перевести это выражение в КНФ, возможно увеличив при этом его длину в постоянное число раз, но не более. Таким образом, доказательство теоремы 4.1 может быть использовано также для того, чтобы показать NP-полноту задачи о КНФ-выполнимости, и мы можем считать этот результат доказанным.

Теперь для того, чтобы доказать NP-полноту задачи о 3-выполнимости, покажем, что КНФ-выполнимость может быть сведена к ней. Предположим, что нам задано выраже-

*) Литералом является переменная или отрицание переменной (т. е. x или $\neg x$).

ние $E=E_1, E_2, \ldots, E_k$ в КНФ, где каждое из E_i является суммой литералов. Мы можем построить новое выражение F в 3-КНФ такое, что F является выполнимым в том и только в том случае, когда E является выполнимым. Время, затрачиваемое на построение F, будет полиномиальным по отношению к длине E, что даст полиномиальное сведение КНФ-выполнимости к задаче о 3-выполнимости.

Исследуем отдельно каждый элемент E_i в зависимости от того, сколько литералов он имеет.

Случай 1. E_i имеет один литерал x. Введем две новые переменные y и z и заменим E_i на $(x+y+z)(x+y+\neg z)\cdot(x+\neg y+z)(x+\neg y+\neg z)$. Тогда подстановка значений исходных переменных (включая x) делает E_i истинным в том и только в том случае, когда существует такая подстановка значений исходных переменных и переменных y и z (для переменных y и z подойдет любая подстановка), что модифицированное выражение оказывается истинным.

Случай 2. E_i имеет два литерала. Этот случай аналогичен случаю 1.

Случай 3. E_i имеет три литерала. Здесь не требуется никакой модификации.

Случай 4. E_i имеет $k>3$ литералов, скажем $E_i=x_1+x_2+\ldots+x_k$. Введем новые переменные $y_1, y_2, \ldots, y_{k-3}$ и заменим E_i на $(x_1+x_2+y_1)(x_3+\neg y_1+y_2)\cdot(x_4+\neg y_2+y_3)\ldots(x_{k-2}+\neg y_{k-4}+y_{k-3})(x_{k-1}+x_k+\neg y_{k-3})$. Если $k=4$, то замена для E_i есть $(x_1+x_2+y_1)(x_3+x_4+\neg y_1)$. Читателю предоставляется проверить, что для всех наборов значений истинности, которые приводят к выполнению E_i, и только для них, существуют наборы значений истинности для всех переменных, включая все y_i, которые делают истинным построенное выражение.

Определим F как произведение выражений, построенных из всех выражений E_i. Заметим, что ни в одном из случаев длина модифицированного выражения не превышает длину исходного более чем в 32 раза *), при этом F легко может быть построено по E, разумеется за полиномиальное относительно длины E время. Таким образом, КНФ-выполнимость полиномиально сведена к 3-выполнимости.

---

*) В предположении, что каждая переменная считается одним символом. Если мы закодируем имена переменных в двоичном коде, мы должны будем включить множитель в виде логарифма от длины входа, но сведение все равно остается полиномиальным.

4.6. *NP*-полнота задачи составления расписаний

Теперь мы готовы к тому, чтобы полиномиально свести задачу о 3-выполнимости к общей задаче составления расписаний, показав тем самым, что последняя является *NP*-полной.

Теорема 4.3. *Задача составления расписаний является NP-полной.*

Доказательство. Мы сейчас покажем, как построить за полиномиальное время из любого булевского выражения E, заданного в 3-КНФ, вход S задачи упорядочения такой, что выполнение всех заданий из S может завершиться до установленного срока в том и только в том случае, когда E является выполнимым. Вход S будет иметь пустое отношение предшествования и всего лишь два процессора. Времена выполнения заданий в S будут огромны, а именно экспоненциальны от длины E. Однако, поскольку времена выполнения представлены в двоичной записи, длина представления S будет все еще полиномом от длины E, и S может быть записана за время, которое является полиномом от длины E.

Вначале положим $E=E_1 E_2 \ldots E_k$ и пусть $E_j=(x_j+y_j+z_j)$, где каждое x_j, y_j и z_j является литералом (т. е. переменной или ее отрицанием). Пусть в выражении E встречаются p переменных, скажем, $v_1, v_2, \ldots, v_p$. Построим множество заданий, время выполнения каждого из которых является $(k+p+1)$-разрядным десятичным числом. Младшие разряды соответствуют суммам E_i, а старшие разряды, за исключением самого старшего, соответствуют переменным. Назначение самого старшего разряда мы объясним позднее.

1. Для каждой переменной v_i, $1 \leqslant i \leqslant p$, построим задания T_i и T_i' с временами выполнения τ_i и τ_i'. Десятичные представления τ_i и τ_i' имеют 0 во всех разрядах, за исключением следующих:

а) в каждом представлении имеется 1 в $(i+1)$-м разряде слева (в разряде, соответствующем v_i);

б) в представлении τ_i имеется 1 в j-м разряде справа (в разряде, соответствующем E_j), если подстановка $v_i=1$ делает E_j истинным, т. е. если один из литералов x_j, y_j или z_j представляет собой v_i;

в) в представлении τ_i' имеется 1 в j-м разряде справа, если одно из x_j, y_j, z_j есть $\neg v_i$ (т. е. подстановка $v_i=0$ делает E_i истинным).

2. Для каждого j, $1 \leqslant j \leqslant k$, построим два задания U_j и U_j'. Пусть времена выполнения U_j и U_j' суть $\sigma_j=\sigma_j'=10^{j-1}$,

т. е. в σ_j и σ'_j все разряды равны 0, кроме единственной единицы в разряде, соответствующем E_j.

3. Пусть V_0 — задание с временем выполнения $\rho_0 = \sum_{i=k}^{p+k} 10^i$.

Тем самым ρ_0 имеет 1 в самом левом разряде и во всех разрядах, соответствующих переменным, и 0 — в разрядах, соответствующих всем E_j.

4. Пусть V_1 — задание с временем выполнения $\rho_1 = \sum_{i=0}^{p+k} 10^i$.

Задания	Времена выполнения	1	p	k
T_i, T'_i	τ_i, τ'_i	1		1 ... 1 ... 11
U_j, U'_j	σ_j, σ'_j			1
V_0	ρ_0	1	11 ... 1	
V_1	ρ_1	1	11 ... 1	11 ... 1
	ω	1	22 ... 2	33 ... 3

Рис. 4.2.

Распределение ненулевых цифр для времен выполнения различных заданий представлено на рис. 4.2.

Для завершения описания задачи упорядочения S положим

$$\omega = 10^{p+k} + 2\sum_{i=k}^{p+k-1} + 3\sum_{i=0}^{k-1} 10^i.$$

Распределение цифр ω также представлено на рис. 4.2. Заметим, что сумма всех времен выполнения в точности равна 2ω; следовательно, ответ «да» для S будет в том и только в том случае, когда времена выполнения могут быть разделены на два равных набора. Мы можем показать, что S имеет ответ «да» в том и только в том случае, когда E является выполнимым.

Достаточность. Предположим, что E выполнено, если в v_i подставлено значение a_i (0 или 1) для каждого i. Основываясь на этом наборе значений, выберем множество заданий, общее время выполнения которых в точности равно ω. Эти задания будут выполняться на одном процессоре в любом порядке, а оставшиеся работы будут выполняться на другом процессоре, и таким образом будет соблюден срок ω.

Вначале выберем T_i, если $a_i = 1$, и T_i', если $a_i = 0$. Поскольку E выполняется при заданном наборе значений, мы знаем, что в десятичном представлении суммы времен выполнения заданий, выбранных таким способом, в первых k разрядах справа стоят цифры 1, 2 или 3. Ни один нуль здесь появиться не может. Поэтому мы можем выбрать еще от нуля до двух заданий U_j и U_j' для каждого j, а это означает, что сумма времен выполнения будет иметь цифры 3 в первых k разрядах справа. Заметим также, что поскольку мы выбрали T_i или T_i', но не оба сразу, в крайнем слева разряде суммы времен выполнения имеем 0, а в каждом из p следующих разрядов имеем 1. Следовательно, выбирая теперь V_0, мы получаем задания с общим временем выполнения, в точности равным ω.

Необходимость. Вначале заметим, что V_0 и V_1 должны выполняться на разных процессорах, поскольку их суммарное время выполнения превышает ω. Рассмотрим другие задания, выполняемые на том же процессоре (скажем, P_1), что и V_0. Чтобы быть уверенным в том, что сумма времен выполнения всех задач, выполняемых на P_1, будет иметь 2 в разрядах со второго по $(p+1)$-й, нужно назначить на P_1 в точности одно из заданий T_i и T_i' для каждого i. Сумма времен выполнения выбранных заданий должна иметь по крайней мере одну единицу в каждом из k крайних справа разрядов. Это объясняется тем, что если бы в j-м разряде справа не было единицы, то даже выполнение обоих заданий U_j и U_j' на P_1 не обеспечило бы наличия тройки в этом разряде у общего времени выполнения для процессора P_1.

Теперь мы можем легко построить набор значений переменных, приводящий к выполнимости E. Конкретно, подставим в x_j значение 1, если T_j выполняется процессором P_1, и подставим 0, если T_j' выполняется процессором P_1. Предыдущий анализ показывает, что при такой подстановке каждое E_j получает значение 1.

На самом деле мы доказали большее, чем утверждает теорема 4.3, поскольку не использовали ограничений предшествования и рассматривали только два процессора.

Следствие 4.1. *Задача упорядочения, в которой отсутствуют ограничения предшествования и имеются только два процессора, все еще остается NP-полной.*

Более абстрактно, можно сказать следующее.

Следствие 4.2. *Для заданного списка целых чисел задача определения того, можно ли этот список разбить на две части таким образом, чтобы сумма чисел в одной части*

оказалась бы в точности равна сумме чисел в другой части, является NP-полной.

Следует отметить, что в предыдущем доказательстве *NP*-полноты задачи упорядочения важную роль играли большие значения времен выполнения. Нам могут возразить, сказав, что действительный «размер» задачи зависит не от длины строки, в которой записаны времена выполнения, а от суммы величин этих времен выполнения. Для случая фиксированного числа процессоров и отсутствия ограничений предшествования читатель без особого труда найдет алгоритм, который требует время, пропорциональное сумме времен выполнения (но экспоненциальное от суммы длин десятичных представлений времен выполнения).

Данная ситуация, бесспорно, выявляет скрытый изъян теории *NP*-полных задач. Способность различных аспектов задачи сделать эту задачу трудной не всегда пропорциональна длине записи, описывающей тот или иной аспект. В рассмотренном нами случае интуитивно чувствуется, что из-за своей длины числа, записанные в десятичном виде, создают гораздо большие трудности в проведении анализа, чем другие аспекты задачи упорядочения. Например, запись ограничений предшествования требует по крайней мере нескольких символов для каждого ограничения; таким образом, представление ограничений предшествования будет иметь относительно большую длину при меньшем вкладе в сложность задачи.

К счастью, для задач составления расписаний мы можем показать, что и в частном случае, когда времена выполнения равны 2 или даже 1 (но при наличии ограничений предшествования), они все еще остаются *NP*-полными. Следовательно, в некотором смысле структура ограничений предшествования вбирает в себя всю комбинаторную сложность, которая содержалась в больших временах выполнения.

4.7. *NP*-полнота задачи минимизации среднего взвешенного времени прохождения

Перед тем как перейти к результатам, упомянутым ранее, рассмотрим другой тип задач упорядочения, в которых величины времен выполнения способствуют возникновению *NP*-полноты.

Задача минимизации среднего взвешенного времени прохождения (поставленная в виде задачи распознавания) состоит в следующем. Заданы набор из m процессоров для выполнения n заданий с временами выполнения $\tau_1, \tau_2, \ldots$

$\ldots, \tau_n$, *веса* $w_1, w_2, \ldots, w_n$, *предельная стоимость* L и *ограничения* предшествования $\prec$. Спрашивается, существует ли такое расписание, что $\sum_{i=1}^{n} w_i f_i \leqslant L$, где f_i есть время завершения i-го задания.

Теорема 4.4. *Задача минимизации среднего взвешенного времени прохождения является NP-полной.*

Доказательство. Мы можем полиномиально свести к этой задаче задачу, упомянутую в следствии 4.2 теоремы 4.3. Последняя задача является NP-полной и заключается в ответе на вопрос, может ли заданный список целых чисел $a_1, a_2, \ldots, a_n$ быть разбит на два подмножества S_1 и S_2, обладающих следующими свойствами:

1) $S_1 \cup S_2 = \{1, 2, \ldots, n\}$;
2) $S_1 \cap S_2 = \varnothing$;
3) $\sum_{i \in S_1} a_i = \sum_{i \in S_2} a_i$.

По заданному входу $a_1, a_2, \ldots, a_n$ этой задачи *(задачи о разбиении)* мы можем за полиномиальное время построить вход задачи минимизации среднего взвешенного времени прохождения следующим образом. Пусть имеются два процессора P_1 и P_2, пустое отношение предшествования и n заданий $T_1, T_2, \ldots, T_n$ таких, что время выполнения и вес i-го задания равны a_i. Положим предельное время L равным $\frac{1}{4}\left[\sum_{i=1}^{n} a_i\right]^2 + \frac{1}{2}\sum_{i=1}^{n} a_i^2$.

Предположим, что для этого множества заданий имеется некоторое расписание, в котором процессор P_1 последовательно выполняет задания с временами выполнения (и, следовательно, весами) $\sigma_1, \sigma_2, \ldots, \sigma_j$, а процессор P_2 выполняет остальные задания с временами выполнения $\rho_1, \rho_2, \ldots, \rho_k$ (в указанном порядке). Заметим, что существует однозначное соответствие между набором всех a_i и объединением всех σ_i и ρ_i. Среднее взвешенное время прохождения для приведенного расписания есть

$$F = \sum_{i=1}^{l} \sigma_i \sum_{l=1}^{i} \sigma_l + \sum_{i=1}^{k} \rho_i \sum_{l=1}^{i} \rho_l, \tag{3}$$

поскольку i-е задание, выполняемое на P_1, имеет вес σ_i и время окончания $\sum_{l=1}^{i} \sigma_l$ (для P_2 — аналогично). Алгебран-

ческие преобразования над (3) дают

$$F=\frac{1}{2}\left[\sum_{i=1}^{j}\sigma_i\right]^2+\frac{1}{2}\left[\sum_{i=1}^{k}\rho_i\right]^2+\frac{1}{2}\left[\sum_{i=1}^{n}a_i^2\right]. \qquad (4)$$

Пусть $A=\sum_{i=1}^{n}a_i$ и $B=\sum_{i=1}^{n}a_i^2$. Тогда предельное время L равно $A^2/4+B^2/2$. Положим $A_1=\sum_{i=1}^{j}\sigma_i$. Тогда $\sum_{i=1}^{k}\rho_i=A-A_1$, и (4) может быть представлено в виде

$$F=\frac{1}{2}[A_1^2+(A-A_1)^2+B]=L+\left(\frac{A}{2}-A_1\right)^2. \qquad (5)$$

Поскольку из (5) следует, что $F>L$, за исключением случая $A_1=A/2$, при котором $F=L$, мы видим, что ответ для построенного входа задачи минимизации среднего взвешенного времени гласит «да» в том и только в том случае, когда ответ для входа $a_1, a_2, \ldots, a_n$ задачи о разбиении есть «да». Следовательно, имея решение задачи о разбиении, мы можем выполнять соответствующую половину заданий на процессоре P_1 в любом порядке. Наоборот, имея решение задачи минимизации среднего взвешенного времени прохождения, мы можем поместить величины a_i, соответствующие заданиям, выполняемым на процессоре P_1, в один набор S_1, а остальные поместить в S_2, получив при этом решение задачи о разбиении.

Следствие 4.3. *Задача минимизации среднего взвешенного времени прохождения с двумя процессорами при отсутствии ограничений предшествования является NP-полной.*

4.8. *NP*-полнота задачи составления расписаний при единичных временах выполнения

Определение. *Задача составления расписаний при единичных временах выполнения* (UET-*задача*) в форме задачи распознавания определяется следующим образом: существует ли расписание для n заданий на m процессорах с $\tau_i=1$, $1\leqslant i\leqslant n$, предельным временем ω и ограничениями предшествования $<$?

Легче оперировать с модифицированной версией этой задачи, в которой в каждый момент времени оказывается доступным разное количество процессоров.

Модифицированная UET-*задача* состоит в том, чтобы определить, существует ли расписание для n заданий с

$\tau_i=1$, $1\leqslant i\leqslant n$, заданным предельным временем ω, ограничениями предшествования $<$ и с $m(i)$ процессорами, доступными в момент времени i, $0\leqslant i\leqslant\omega$ *).

Вначале мы покажем, что модифицированная UET-задача является NP-полной (путем полиномиального сведения к ней задачи о 3-выполнимости). Затем покажем, что UET-задача упорядочения является NP-полной, сведя к ней модифицированную UET-задачу

Лемма 4.1. *Модифицированная* UET-*задача является NP-полной.*

Доказательство. Сведем задачу 3-выполнимости к модифицированной UET-задаче. Пусть $E=E_1\ldots E_k$ — булевское выражение в 3-КНФ, и пусть $F_j=x_j+y_j+z_j$, $1\leqslant j\leqslant k$, где x_j, y_j и z_j являются литералами. Пусть v_1, v_2, . . ., v_p— переменные, входящие в E. Построим вход модифицированной UET-задачи следующим образом. Заданиями являются:

1) T_{ij} и $\overline{T}_{ij}$, $1\leqslant i\leqslant p$, $0\leqslant j\leqslant p$;

2) S_i и $\overline{S}_i$, $1\leqslant i\leqslant p$;

3) D_{ij}, $1\leqslant i\leqslant k$, $1\leqslant j\leqslant 7$.

Задания T и S представляют переменные, а задания D описывают множители, входящие в выражение для E.

Ограничения предшествования заданы следующим образом.

1. $T_{ij}<T_{i,j+1}$ и $\overline{T}_{ij}<\overline{T}_{i,j+1}$, $1\leqslant i\leqslant p$, $0\leqslant j<p$.

2. $T_{i,i-1}<S_i$ и $\overline{T}_{i,i-1}<\overline{S}_i$, $1\leqslant i\leqslant p$.

3. Рассмотрим D_{ij}, и пусть $a_1a_2a_3$—двоичное представление j (заметим, что случай $a_1=a_2=a_3=0$ встретиться не может). Напомним, что E_i есть $x_i+y_i+z_i$. Пусть литералы x_i, y_i и z_i представляют собой переменные v_{r_1}, v_{r_2} и v_{r_3} либо их отрицания. Положим $T_{r_1,p}<D_{ij}$, если x_i есть v_{r_1} (а не $\neg v_{r_1}$) и $a_1=1$ или $x_i=\neg v_{r_1}$ и $a_1=0$. Иначе, положим $\overline{T}_{r_1,p}<D_{ij}$. Аналогично, $T_{r_2,p}<D_{ij}$, если $y_i=v_{r_2}$ и $a_2=1$ или $y_i=\neg v_{r_2}$ и $a_2=0$; иначе, $\overline{T}_{r_2,p}<D_{ij}$. Наконец, $T_{r_3,p}<D_{ij}$, если $z_i=v_{r_3}$ и $a_3=1$ или $z_i=\neg v_{r_3}$ и $a_3=0$; иначе, $\overline{T}_{r_3,p}<D_{ij}$.

---

*) Процессор, доступный в момент времени i, может быть использован для выполнения задания, начинающегося в момент i и завершающегося в момент времени $i+1$.

Срок завершения $\omega = p+3$. Число процессоров $m(i)$, доступных в момент i, определяется следующим образом:

$$
\begin{aligned}
m(0) &= p, \\
m(1) &= 2p+1, \\
m(i) &= 2p+2, \quad 2 \leqslant i \leqslant p, \\
m(p+1) &= k+p+1, \\
m(p+2) &= 6k.
\end{aligned}
$$

Мы можем показать, что построенная модифицированная UET-задача имеет решение в том и только в том случае, когда имеет решение исходная задача о 3-выполнимости. Неформально, идея доказательства состоит в том, что мы можем считать, что v_i (или $\neg v_i$) принимает значение «истина» в том и только в том случае, когда T_{i0} (или соответственно $\overline{T}_{i0}$) выполняется, начиная с момента времени 0. Мы увидим, что наличие S и $\overline{S}$ приводит к тому, что ровно одно из заданий T_{i0} и $\overline{T}_{i0}$ начинает выполняться в момент времени 0, а другое выполняется, начиная с момента времени 1. Тогда требование, чтобы $k+m+1$ заданий выполнялись, начиная с момента времени $p+1$, равносильно требованию того, чтобы для каждого i существовал один момент j такой, что D_{ij} может выполняться, начиная с этого времени (таких моментов не может быть больше одного). Но это условие эквивалентно заявлению, что сумма членов, представленных с помощью D_i, имеет значение «истина», когда те из v_i и $\overline{v}_i$, для которых соответствующие T_{i0} и $\overline{T}_{i0}$ выполнялись в момент времени 0, получили значение «истина».

Вначале покажем, что в любом решении модифицированной UET-задачи выполнение обоих заданий T_{i0} и $\overline{T}_{i0}$ для любого i не может начинаться в момент 0. Предположим противное. Тогда, поскольку $i_0 = p$, должно найтись j такое, что ни T_{j0}, ни $\overline{T}_{j0}$ не начинают выполняться в момент времени 0. В таком случае ни S_j, ни $\overline{S}_j$ не начнут выполняться в момент j или ранее, потому что, например, перед S_j должны выполняться одно за другим задания $T_{j0}, T_{j1}, \ldots, T_{j,j-1}$. Таким образом, общее число заданий, которые могут быть выполнены, начиная с момента j или до него, определяется следующим образом:

1) не более $p(2j+1)$ заданий типа T и $\overline{T}$, а именно $T_{i0}, T_{i1}, \ldots, T_{ij}$, если T_{i0} выполнялось в момент времени 0, и $\overline{T}_{i0}, \overline{T}_{i1}, \ldots, \overline{T}_{i,j-1}$ в противном случае;

2) не более $2(j-1)$ заданий типа S, а именно S_1, $\overline{S}_1$, S_2, $\overline{S}_2$, ..., S_{j-1}, $\overline{S}_{j-1}$.

Следовательно, общее число заданий, которые могут быть начаты к моменту времени j, не превосходит $2pj+2j+p-2$. Однако для $1 \leqslant j \leqslant p$ имеем

$$\sum_{i=0}^{j} m(i) = 3p+1+(j-2)(2p+2) = 2pj+2j+p-1.$$

Мы заключаем, что в любом решении нашей модифицированной UET-задачи только одно из заданий T_{i0} и $\overline{T}_{i0}$ может начинаться в момент времени 0. Более того, зная, какие из заданий T_{i0} и $\overline{T}_{i0}$ выполняются в момент времени 0, мы можем точно определить задания, которые будут выполняться, начиная с каждого момента между 1 и p. А именно, если T_{i0} начинало выполняться в момент 0, то в момент j нужно начинать выполнение T_{ij}; в противном случае нужно выполнять $T_{i,j-1}$. Кроме того, мы должны выполнять S_j (соответственно $\overline{S}_j$), начиная с момента времени j, если T_{j0} (соответственно $\overline{T}_{j0}$) выполнялось, начиная с момента 0, и должны выполнять S_{j-1} (соответственно $\overline{S}_{j-1}$) в момент j, если T_{j0} (соответственно $\overline{T}_{j0}$) выполнялось, начиная с момента 1.

Начиная с момента $p+1$, мы можем выполнять p оставшихся заданий типа T и $\overline{T}$ и одно оставшееся из S или $\overline{S}$. Поскольку $m(p+1) = p+k+1$, то для получения решения мы должны иметь возможность выполнить k заданий типа D. Заметим, что для каждой пары D_{ij} и $D_{ij'}$, $j \neq j'$, существует по крайней мере одно l такое, что $T_{lp} < D_{ij}$ и $\overline{T}_{lp} < D_{ij'}$, или наоборот. Поскольку мы уже доказали, что ровно одно из заданий T_{lp} и $\overline{T}_{lp}$ может начаться в момент p или ранее, из этого следует, что для каждого i не более одного задания из $D_{i1}, D_{i2}, \ldots, D_{i7}$ может выполняться, начиная с момента $p+1$.

Кроме того, если мы подставили значение «истина» в v_j (соответственно $\neg v_j$) в том и только в том случае, когда T_{j0} (соответственно $\overline{T}_{j0}$) выполнялось, начиная с момента времени 0, то мы можем выполнить одно из заданий $D_{i1}, D_{i2}, \ldots, D_{i7}$, начиная с момента $p+1$, в том и только в том случае, когда E_i при такой подстановке значений переменных принимает значение «истина». Мы получаем, что решение нашей модифицированной UET-задачи сущест-

вует тогда и только тогда, когда первоначальное произведение сумм является выполнимым.

Пример 5. Рассмотрим булевское выражение $(x_1 + \neg x_2 + x_3) \cdot (\neg x_1 + \neg x_3 + x_4)$. Здесь $k = 2$, $p = 4$, $E = (x_1 + \neg x_2 + x_3)$, $E_2 = (\neg x_1 + \neg x_3 + x_4)$ и $m(0), \ldots, m(6)$ есть, соответственно, 4, 9, 10, 10, 10, 7, 12. Одно из возможных решений соответствующей модифицированной

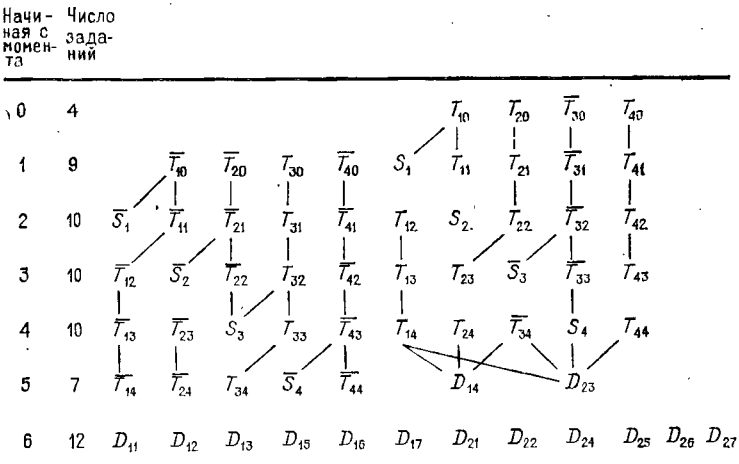


Рис. 4.3.

UET-задачи, в основу которого положена подстановка значения «истина» в x_1, x_2, $\neg x_3$ и x_4, показано на рис. 4.3. Линии представляют отношение $<$, за исключением последней строки, где необходимые многочисленные линии опущены.

Теорема 4.5. UET-*задача упорядочения является NP-полной.*

Доказательство. Построим полиномиальное сведение модифицированной UET-задачи к UET-задаче. Конструкция очень проста. Если в модифицированной задаче указано предельное время ω и в момент i доступны $m(i)$ процессоров, $0 \leqslant i \leqslant \omega$, то положим $m = 1 + \max m(i)$. Затем для $0 \leqslant i < \omega$ введем $m - m(i)$ новых заданий R_{i1}, R_{i2}, $\ldots$, $R_{i, m-m(i)}$. Пусть $R_{i-1, j} < R_{ik}$ для всех $0 < i < \omega$ и любых j и k. Тогда, если исходная модифицированная UET-задача имеет решение, то будет существовать расписание и для системы, дополненной новыми заданиями и использующей в каждый момент времени m процессоров, и наоборот.

Следствие 4.4. *Задача составления расписаний с прерываниями является NP-полной.*

Доказательство. Легко убедиться, что если происходит любое прерывание, то выполнение системы заданий модифицированной UET-задачи, построенной в лемме 4.1, не может завершиться вовремя. Следовательно, для построенного входа расписание с прерываниями существует тогда и только тогда, когда заданное булевское выражение является выполнимым.

4.9. Составление расписаний для двух процессоров и времен выполнения, равных 1 и 2

В предыдущем параграфе мы видели, что задача, являющаяся частным случаем общей задачи составления расписаний, может быть NP-полной даже тогда, когда не используется возможность назначения больших времен выполнения. Числовыми параметрами UET-задачи были только предельное время и число процессоров. В действительности возможность двоичного кодирования параметров m и ω UET-задачи не имеет существенного значения.

Читатель может проверить процедуру сведения, использованную в лемме 4.1 и теореме 4.5, в случае, когда m, числа $m(i)$ и ω представлены в унарном коде, и убедиться в том, что сложность этой процедуры по-прежнему полиномиально зависит от длины входа.

Теперь мы рассмотрим NP-полную задачу, являющуюся частным случаем задачи упорядочения, в которой число процессоров и предельное время не выступают как параметры, а времена выполнения, хотя и являются параметрами, ограничены значениями 1 и 2. NP-полнота этой задачи ясно показывает, что комбинаторная структура ограничений предшествования является достаточным фактором, чтобы сделать задачу трудной; при этом нам не придется опираться на сложность, присущую большим числам.

Задача $S12$ состоит в определении того, имеет ли множество из n заданий с τ_i, равными 1 или 2 для $1 \leqslant i \leqslant n$, и с наибольшим временем завершения

$$\omega = \frac{1}{2} \sum_{i=1}^{n} \tau_i$$

расписание на двух процессорах. Заметим, что если такое расписание существует, то в нем нет простоев.

Для того чтобы доказать NP-полноту задачи $S12$, нужно несколько усилить теорему 4.5.

Лемма 4.2. UET-*задача упорядочения с* $n=\omega m$ *(т. е. с таким ограничением, чтобы в решении не было простоев) является NP-полной.*

Доказательство. Анализ доказательств леммы 4.1 и теоремы 4.5 показывает, что в каждом случае, если решение существует, то в нем нет простоев. Поэтому мы могли бы с самого начала ограничить соответствующим образом определение UET-задачи и модифицированной UET-задачи, и все доказательства остались бы в силе.

Теорема 4.6. *Задача S*12 *является NP-полной.*

Доказательство. На основании леммы 4.2 для доказательства достаточно свести UET-задачу упорядочения с $n=\omega m$ к задаче $S12$. Предположим, что имеется вход UET-задачи, в котором заданы предельное время ω, число процессоров m и отношение предшествования $<$ на множестве $\mathscr{T}$ из ωn заданий. Построим следующий вход задачи $S12$.

Заданиями являются:

1) X_i, $0\leqslant i<\omega'$, где $\omega'=(4m+1)\omega$;
2) Y_{ij}, $0\leqslant i<\omega$, $0\leqslant j\leqslant m$;
3) T и T' для каждого T в $\mathscr{T}$.

Время выполнения равно 2 для T, входящих в $\mathscr{T}$, и 1 для всех остальных заданий.

Отношение $<'$ описывается следующим образом:

а) $X_i<'X_{i+1}$, $0\leqslant i<\omega'-1$;

б) $X_u<'Y_{ij}<'X_{u+2}$, где $u=(4m+1)i+2j-1$ (в случае $i=j=0$, когда $u=-1$, мы имеем только отношение $Y_{00}<'X_1$);

в) $T'<'T$ для всех T из $\mathscr{T}$;

г) $S<'T'$ для S и T из $\mathscr{T}$ тогда и только тогда, когда $S<T$.

Предельное время завершения — ω', определенное в п. 1).

Вначале отметим, что, как следует из а), для того, чтобы уложиться в предельное время, в каждый момент один процессор должен быть занят выполнением заданий типа X. Можно положить, что процессор P_1 полностью выделяется для этой цели. Из условия б) следует, что задания типа Y должны выполняться на процессоре P_2 в специальные моменты времени, как показано на рис. 4.4.

Это означает, что, передвигаясь по оси времени, мы получаем чередование *разрывов*, в которых каждый второй

единичный интервал свободен для P_2, и *полос*, которые представляют собой $2m$ последовательных свободных интервалов для P_2. Поскольку в нашей задаче задания T из $\mathcal{T}$ требуют двух единиц времени каждое, ясно, что они должны выполняться только в полосах. Поскольку имеется ωm таких заданий, они должны полностью заполнить полосы. Это означает, что задания T' должны выполняться только в разрывах.

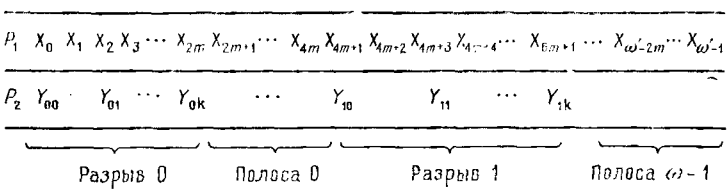


Рис. 4.4.

Вследствие этого, если задания S и T выполняются в одной полосе, то отношение $S<T$ невозможно. Действительно, если бы такое случилось, то, поскольку мы имеем $S<'T'<'T$, задание T' также должно было бы выполняться в этой полосе, что противоречит только что сделанному выводу. Таким образом, если наша задача $S12$ имеет решение, то мы можем найти решение исходной UET-задачи, начиная в момент времени i выполнение в точности тех заданий, которые выполнялись в i-й полосе.

Наоборот, если мы имеем решение исходной UET-задачи, то мы можем найти решение построенной задачи S12, выполняя T' в i-м разрыве, а T — в i-й полосе всякий раз, когда в исходной задаче T начинает выполняться в момент времени i.

4.10. Составление расписаний с ограничениями на ресурсы

Если в задачу упорядочения ввести ограничения на ресурсы, то мы можем показать ее NP-полноту даже при более строгих условиях, чем те, которые были приняты в предыдущих двух параграфах.

Задача упорядочения с ограничениями на ресурсы включает множество из n заданий с заданными временами выполнения, m процессоров, предельное время ω и ограничения предшествования $<$, т. е. все как обычно, однако вводится еще множество $\mathcal{R}=\{R_1, R_2, \ldots, R_s\}$ *ограничений ресур-*

сов и *предельные ресурсы* m_i, $1 \leqslant i \leqslant s$. Каждое R_i ставит в соответствие заданиям неотрицательные числа, означающие объем ресурса i-го типа, требуемый для выполнения задания *). Задача заключается в определении того, существует ли такое расписание, укладывающееся в предельное время, которое в процессе выполнения заданий требует в каждый момент времени не более m_i единиц i-го ресурса для всех i.

С помощью методики, использованной в примере 3 для задачи без ограничений на ресурсы, легко показать, что общая задача составления расписаний с ограничениями на ресурсы принадлежит **NP**. А именно, угадываем расписание и проверяем, что оно укладывается в предельное время и в ограниченные ресурсы. Здесь мы приведем две сильные теоремы об *NP*-полных вариантах задачи составления расписаний с ограничениями на ресурсы, доказанные Гэри и Джонсоном [54].

Теорема 4.7. *Задача упорядочения на двух процессорах, при единичных временах выполнения, одном ресурсе и ограничениях предшествования в виде леса является NP-полной.*

Теорема 4.8. *Задача упорядочения на трех процессорах, при единичных временах выполнения, одном ресурсе и пустом отношении предшествования является NP-полной.*

Хотя мы не доказываем ни одну из этих теорем, мы можем легко доказать утверждение, аналогичное теореме 4.7.

Теорема 4.9. *Задача упорядочения на двух процессорах, при единичных временах выполнения и одном ресурсе с предельным значением* 1 *является NP-полной.*

Доказательство. Способом, аналогичным использованному при доказательстве теоремы 4.6, мы можем полиномиально свести UET-задачу с $\omega m = n$ к сформулированному варианту задачи упорядочения с ограничениями на ресурсы. Вместо того, чтобы заставлять задания с временами исполнения, равными 2, выполняться в полосах, мы можем заставить выполняться в полосах те задания, которые не требуют единичного ресурса.

Формально, предположим, что задан вход UET-задачи, а именно: множество $\mathscr{T}$, состоящее из n заданий, отношение

*) Типичными видами ресурсов могут быть память и устройства ввода-вывода. Процессоры не рассматриваются здесь как ресурсы, хотя это и можно было сделать.

предшествования $\prec$, предельное время завершения ω и m процессоров, где $n=\omega m$. Построим вход UET-задачи с двумя процессорами и одним ресурсом следующим образом. Берется последовательность, состоящая из $2\omega m$ заданий $X_0X_1, \ldots, X_{2\omega m-1}$. Кроме того, каждому заданию T из $\mathscr{T}$ соответствуют два задания T и T' в новой системе. Ограничения предшествования $\prec'$ заданы следующим образом:

1) $X_i \prec' X_{i+1}$, $0 \leqslant i < 2\omega m-2$;
2) $T' \prec' T$ для всех T из $\mathscr{T}$;
3) $S \prec' T$ тогда и только тогда, когда $S \prec T$ для S и T из $\mathscr{T}$.

Предельное время $\omega' = 2\omega m$.

Наконец, имеется один ресурс с предельной величиной 1. Задания T из $\mathscr{T}$ требуют по единице этого ресурса. Задания T' (для T из $\mathscr{T}$) не требуют никаких ресурсов. Задания типа X_i, у которых индекс i находится в диапазонах от 0 до $m-1$, от $2m$ до $3m-1$, от $4m$ до $5m-1$ и т. д., требуют по единице ресурса. Остальным заданиям ресурс не требуется.

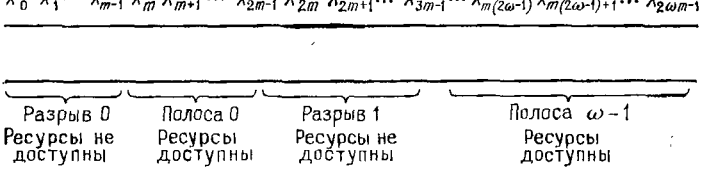


Рис. 4.5.

Как и в теореме 4.6, мы можем считать, что если расписание существует, то один процессор P_1 используется для всех заданий типа X. Тогда задания типа T и T' могут выполняться на P_2. Однако наличие ресурса заставляет нас выполнять задания типа T в полосах, определяемых интервалами от m до $2m$, затем от $3m$ до $4m$ и т. д., как показано на рис. 4.5. Дальнейший ход доказательства такой же, как в теореме 4.6. Все задания типа T должны выполняться в полосах, и при этом никакие два задания, выполняемые в одной и той же полосе, не могут быть связаны отношением $\prec$. Таким образом, расписание для исходной UET-задачи существует тогда и только тогда, когда существует решение построенной задачи, что и завершает полиномиальное сведение.

Библиографическая справка

Родоначальником концепции *NP*-полных задач является Кук [30]. Он также показал *NP*-полноту 3-КНФ задачи. Им были доказаны теоремы 4.1 и 4.2. Большой вклад в доказательство широты класса *NP*-полных задач внес Карп [86]; см. также книгу Ахо, Хопкрофта и Ульмана [3], содержащую обзор результатов в этой области.

Теорема 4.3, утверждающая, что общая задача упорядочения является *NP*-полной, имеется в неявном виде в работе Карпа [86]. Теорема 4.4 о минимуме взвешенного времени завершения взята из работы Бруно, Коффмана и Сети [11]. Теоремы 4.5 и 4.6 об UET-упорядочении и об упорядочении при временах выполнения 1 и 2 взяты из работы Ульмана [143]. Теоремы 4.7 и 4.8 об упорядочении при ограничениях на ресурсы получены Гэри и Джонсоном [54]. Недавние работы [14, 55, 57] содержат дополнительные результаты, касающиеся *NP*-полноты задач упорядочения, многие из которых упомянуты в § 1.4.

ГЛАВА 5

ОЦЕНКИ ХАРАКТЕРИСТИК АЛГОРИТМОВ СОСТАВЛЕНИЯ РАСПИСАНИЙ

В этой главе мы исследуем поведение ряда алгоритмов составления мультипроцессорных расписаний с ограничениями на ресурсы при наиболее неблагоприятных условиях и рассмотрим многочисленные важные частные случаи. Модель и система обозначений описаны в гл. 1. В § 5.1 исследована довольно неожиданная зависимость длины списочного расписания ω от различных параметров задачи — даже в случае отсутствия ограничений на дополнительные ресурсы. В §§ 5.2, 5.3 исследованы характеристики алгоритмов составления расписаний с помощью критического пути. В § 5.4 выведены верхние оценки характеристик для расширенной модели, в которой имеются ограничения на дополнительные ресурсы. В § 5.5 рассмотрены эвристические алгоритмы решения задачи о минимизации числа процессоров, требуемых для выполнения системы заданий без нарушения заданных крайних сроков, известной как задача об упаковке в контейнеры. Наконец, в § 5.6 представлены верхние оценки для ряда смежных задач. Вопрос об определении сложности рассматриваемых в данной главе задач был исследован в предыдущей главе.

5.1. Аномалии многопроцессорных расписаний

Мы начнем с примера, который иллюстрирует на основе системы, состоящей из одиночных заданий, аномальные эффекты, возникающие при варьировании *любого* из параметров, включая и список приоритетов.

Пример 1. Граф $G(<, \tau)$ системы заданий изображен на рис. 5.1, *а*. Зафиксировав параметры, рассмотрим эффекты, сопровождающие изменение списка приоритетов, увеличение числа процессоров, уменьшение времени выполнения и ослабление ограничений предшествования, которые изображены на рис. 5.2.—5.5. Эталоном для сравнения служит оптимальное расписание, представленное на рис. 5.1, *б*.

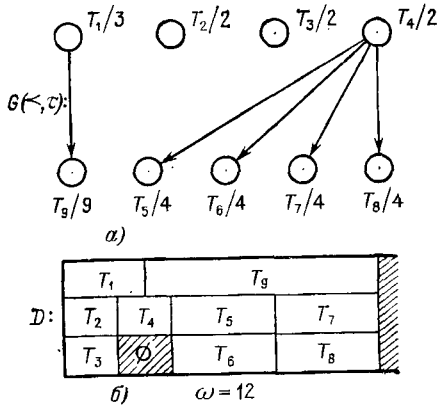


Рис. 5.1. Система заданий (*а*) и оптимальное расписание (*б*); $m=3$, $L=(T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9)$. На рис. 5.2.— 5.5 показано, как изменяется расписание при изменении L, m, τ и $\prec$.

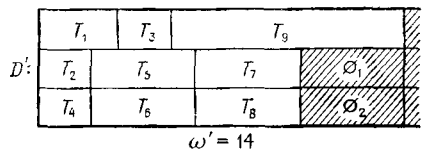


Рис. 5.2. Изменение списка приоритетов. Здесь $L'=(T_1, T_2, T_4, T_5, T_6, T_3, T_9, T_7, T_8)$.

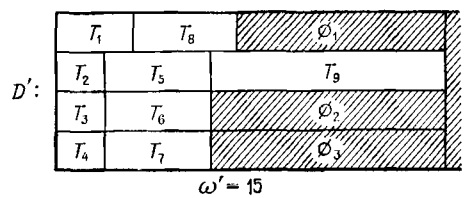


Рис. 5.3. Эффект увеличения числа процессоров: m заменено на $m'=4$.

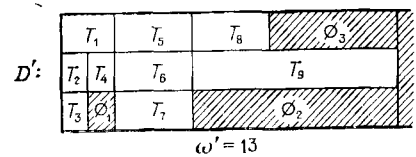


Рис. 5.4. Эффект уменьшения времен выполнения: τ заменены на $\tau'=\tau-1$.

Заметим, что хотя для случаев, изображенных на рис. 5.3, 5.4 и 5.5, интуитивно можно предположить уменьшение ω, однако на самом деле оказывается, что в этих случаях ω *увеличивается*.

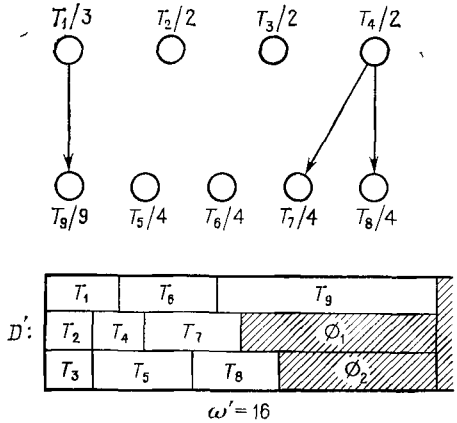


Рис. 5.5. Эффект ослабления ограничений предшествования: $\prec$ заменено на $\prec' = \prec - \{(T_4, T_5), (T_4, T_6,)\}$; $m=3$, $L=(T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9)$.

Следующий пример показывает, что такое увеличение ω не обязательно происходит вследствие плохого выбора списка приоритетов L, а является в действительности свойством, присущим самой модели.

Пример 2. На рис. 5.6, *а* изображена система заданий и указан список приоритетов; предполагается, что

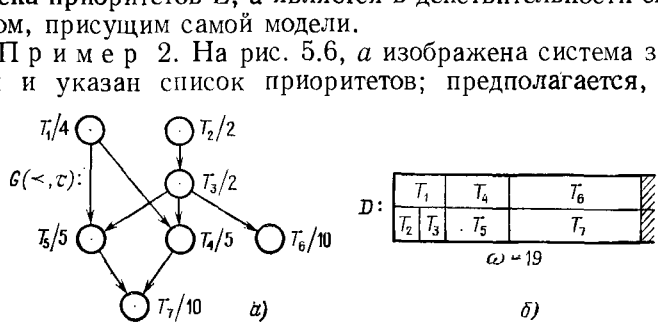


Рис. 5.6.

$m=2$. Оптимальное расписание дано на рис. 5.6, *б*. Теперь рассмотрим такую же систему заданий, но времена выполнения заменим на $\tau' = \tau - 1$. Мы обнаружим, что *независимо от того, каков список приоритетов*, для новой системы мы не можем получить расписание, длина которого была бы меньше, чем у изображенного на рис. 5.7.

Теперь выведем общую верхнюю оценку относительного изменения длины расписания при изменении одного или более параметров задачи. Предположим, нам задано множество $\mathscr{T}$ заданий, которые мы выполняем дважды. В первый раз мы используем времена выполнения τ, частичное

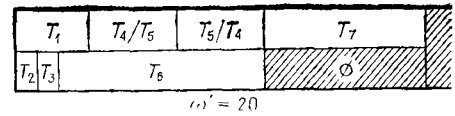


Рис. 5.7.

упорядочение $\prec$, список приоритетов L и вычислительную систему, состоящую из m идентичных процессоров. Во второй раз мы используем времена выполнения $\tau' \leqslant \tau$, частичное упорядочение $\prec' \subseteq \prec$, список приоритетов L' и вычислительную систему, состоящую из m' идентичных процессоров. Соответствующие времена завершения обозначим через ω и ω'.

Теорема 5.1 [45]. *При введенных предположениях имеем*

$$\omega'/\omega \leqslant 1 + (m-1)/m'.$$

Доказательство. Рассмотрим временную диаграмму D', полученную в результате выполнения заданий T_i из $\mathscr{T}$ при втором наборе параметров. Определим разбиение интервала $[0, \omega')$ на два подмножества A и B следующим образом:

$$A = \{t \in [0, \omega') \mid \text{все процессоры заняты в момент } t\},$$
$$B = [0, \omega') - A.$$

Заметим, что как A, так и B являются объединениями непересекающихся полуоткрытых интервалов. Пусть T_{j_1} обозначает задание, выполнение которого в D' завершается в момент ω' (т. е. такое, что $f_{j_1} = \omega'$). Тогда существуют две возможности.

1. Если s_{j_1} — время начала выполнения задания T_{j_1} — является внутренней точкой B, то согласно определению B имеется некоторый процессор P_i, который для некоторого $\varepsilon > 0$ свободен на интервале $[s_{j_1} - \varepsilon, s_{j_1})$. Единственная ситуация, при которой это может произойти, состоит в том, что для некоторого T_{j_2} выполняется $T_{j_2} \prec' T_{j_1}$ и $f_{j_2} = s_{j_1}$.

2. С другой стороны, предположим, что s_{j_1} не является внутренней точкой B. Более того, предположим, что $s_{j_1} \neq 0$. Пусть $x_1 = \sup\{x \mid x < s_{j_1}, x \in B\}$; если указанное множество пусто, полагаем $x_1 = 0$. Из определения A и B видно, что

$x_1 \in A$ и для некоторого $\varepsilon > 0$ P_i является свободным на интервале $[x_1 - \varepsilon, x_1)$. Но опять это может произойти только потому, что существует некоторое задание $T_{j_2} \prec' T_{j_1}$, которое выполняется в течение этого интервала времени.

Таким образом, мы видим, что *либо* существует задание $T_{j_2} \prec' T_{j_1}$ такое, что $y \in [f_{j_2}, s_{j_1})$ влечет $y \in A$, *либо* $x < s_{j_1}$ влечет $x \in A$ или $x < 0$.

Мы можем индуктивно повторять эту процедуру, формируя T_{j_3}, T_{j_4}, ..., до тех пор, пока не достигнем задания T_{j_r}, для которого $x < s_{j_r}$ влечет либо $x \in A$, либо $x < 0$. Следовательно, мы показали существование цепочки заданий

$$T_{j_r} \prec' T_{j_{r-1}} \prec' \ldots \prec' T_{j_2} \prec' T_{j_1} \tag{1}$$

такой, что в D' в любой момент $t \in B$ выполняется некоторое задание T_{j_k}. Из этого следует, что

$$\sum_{\theta' \in D'} \tau'(\theta') \leqslant (m'-1) \sum_{k=1}^{r} \tau'_{j_k}, \tag{2}$$

где сумма, стоящая слева, взята по всем пустым заданиям *) θ' в D'.

Но из (1) и предположения, что $\prec' \subseteq \prec$, имеем

$$T_{j_r} \prec T_{j_{r-1}} \prec \ldots \prec T_{j_2} \prec T_{j_1}. \tag{3}$$

Значит,

$$\omega \geqslant \sum_{k=1}^{r} \tau_{j_k} \geqslant \sum_{k=1}^{r} \tau'_{j_k}. \tag{4}$$

Следовательно, из (2) и (4) получаем

$$\omega' = \frac{1}{m} \left\{ \sum_{k=1}^{n} \tau'_k + \sum_{\theta' \in D'} \tau'(\theta') \right\} \leqslant \frac{1}{m'} (m\omega + (m'-1)\omega). \tag{5}$$

Отсюда получаем

$$\omega'/\omega \leqslant 1 + (m-1)/m', \tag{6}$$

и теорема доказана.

Следующие примеры показывают, что оценка теоремы 5.1 не только является наилучшей, но и может на самом деле быть достигнута (асимптотически) при варьировании любого из параметров.

Пример 3. В этом примере L изменяется, $\prec$ пусто й m произвольно. Задания и времена выполнения заданы

*) То есть по всем интервалам простоя процессоров. (*Прим. перев.*)

следующим образом:

$$T_1/1,\ T_2/1,\ \ldots,\ T_{m-1}/1,\ T_m/m-1,\ T_{m+1}/m-1,\ \ldots$$
$$\ldots,\ T_{2m-2}/m-1,\ T_{2m-1}/m.$$

На рис. 5.8 представлены первый из используемых списков и получившееся в результате его применения расписание.

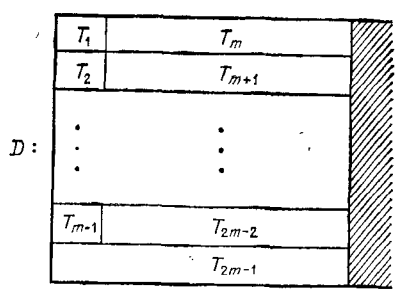


Рис. 5.8. Оптимальное расписание; $L=(T_1,\ T_2,\ \ldots,\ T_{m-1},\ T_{2m-1},\ T_m,\ T_{m+1},\ \ldots,\ T_{2m-2})$.

Второй список и получившееся более длинное расписание даны на рис. 5.9.

Как можно убедиться,

$$\omega'/\omega = 2-1/m.$$

Пример 4. В этом примере уменьшаются величины τ. Здесь, так же как и в остальной части главы, ε обозна-

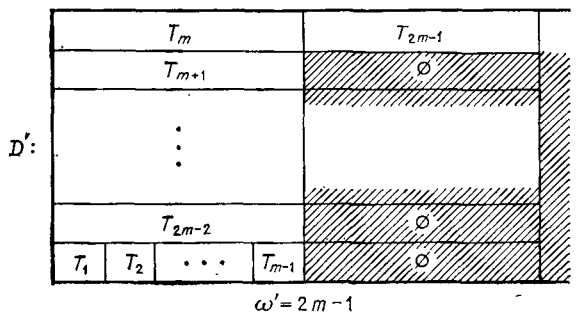
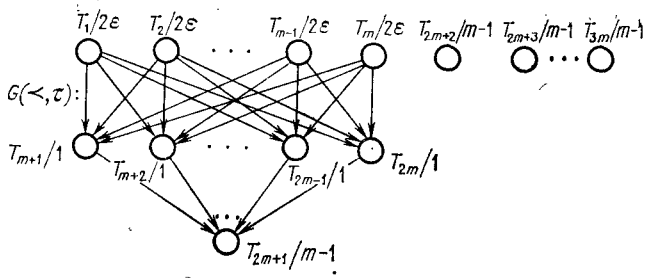


Рис. 5.9. Плохое расписание; $L'=(T_m,\ T_{m+1},\ \ldots,\ T_{2m-2},\ T_1,\ T_2,\ \ldots$
$\ldots,\ T_{m-1},\ T_{2m-1})$.

чает достаточно малое положительное число. Система заданий и соответствующее оптимальное расписание изображены на рис. 5.10. На рис. 5.11 показан эффект, возникающий при



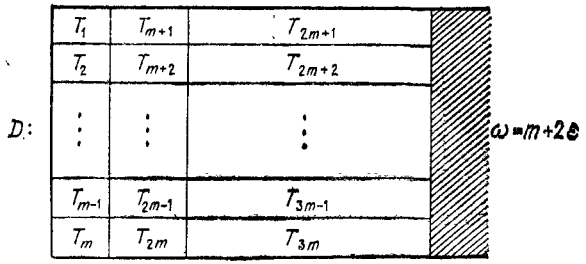


Рис. 5.10.

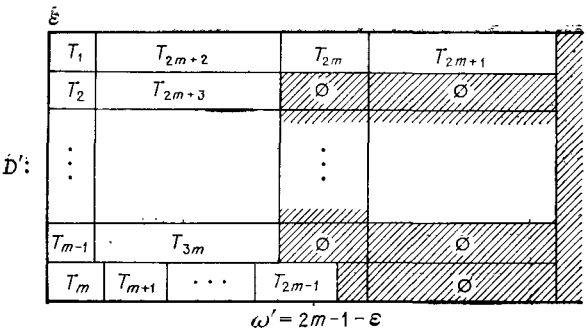


Рис. 5.11.

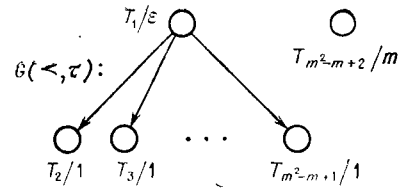
изменении времен выполнения следующим образом:

$$\tau_i' = \begin{cases} \tau_i - \varepsilon & \text{для } 1 \leqslant i \leqslant m-1, \\ \tau_i & \text{в противном случае.} \end{cases}$$

Сравнивая рисунки, находим, что

$$\frac{\omega'}{\omega} = \frac{2m-1+\varepsilon}{m+2\varepsilon} \to 2 - \frac{1}{m} \quad \text{при } \varepsilon \to 0.$$

Пример 5. В этом примере мы ослабим ограничения предшествования $\prec$. Сравним систему заданий и оптималь-



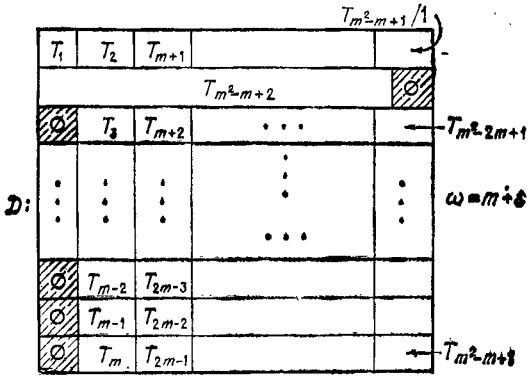


Рис. 5.12.

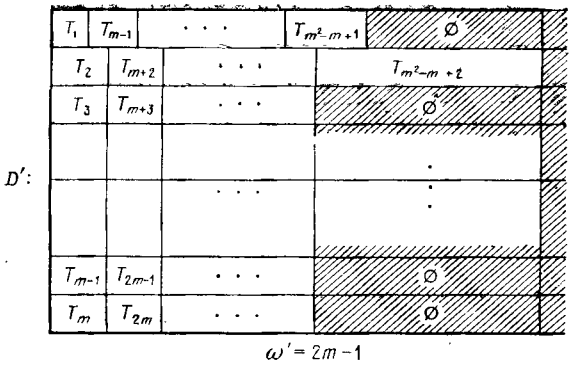


Рис. 5.13.

ное расписание (рис. 5.12) с тем, что получается, когда ограничения предшествования устранены (рис. 5.13). Из этих диаграмм получаем, что

$$\frac{\omega'}{\omega} = \frac{2m-1}{m+\varepsilon} \to 2 - \frac{1}{m} \text{ при } \varepsilon \to 0.$$

Пример 6. Наконец, рассмотрим, что происходит при увеличении числа процессоров. Предположим, задана система заданий, представленная на рис. 5.14, *а*, оптимальное расписание для которой при наличии m процессоров

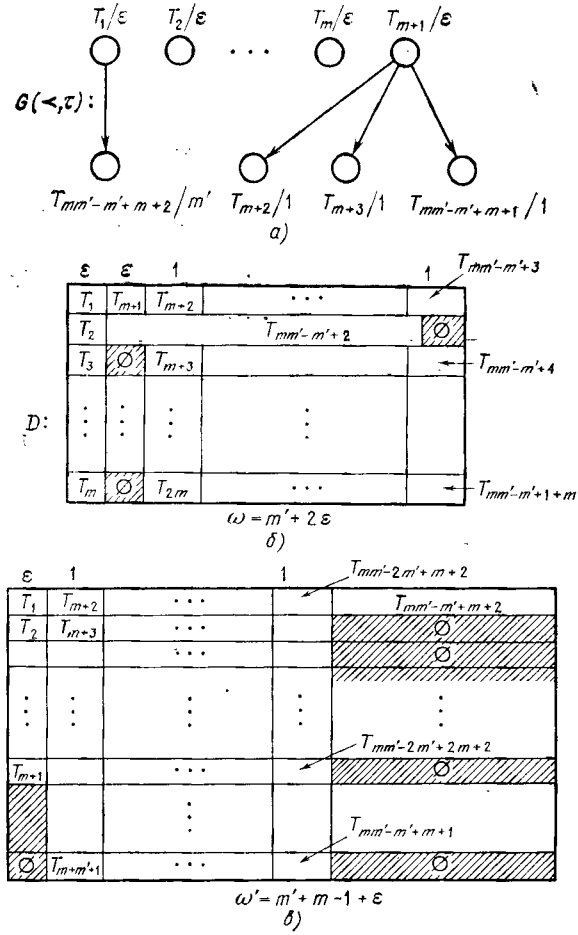


Рис. 5.14.

изображено на рис. 5.14, *б*. Теперь положим $m' > m$. Мы получаем более длинное расписание, показанное на рис. 5.14, *в*. Вычисляя отношение длин расписаний, получим

$$\frac{\omega'}{\omega} = \frac{m' + m - 1 + \varepsilon}{m' + 2\varepsilon} \to 1 + \frac{m-1}{m'} \quad \text{при} \quad \varepsilon \to 0.$$

Аналогичный пример существует и для случая $m' < m$, но мы его здесь не приводим.

Следующий пример, предложенный Кауфманом [84], показывает, что оценка (6) может быть достигнута при изменении L даже в том случае, когда $\prec$ представляет собой лес и все $\tau_i = 1$.

Пример 7. Рассмотрим граф, изображенный на рис. 5.15, *а*, и предположим, что все задания имеют единичные времена выполнения. Оптимальное расписание и соответствующий список показаны на рис. 5.15, *б*. Если изменить список так, как показано на рис. 5.16, то расписание окажется длиннее. Очевидно,

$$\omega'/\omega = 2 - 1/m.$$

Наконец, дадим пример, в котором снова достигается оценка $2-1/m$ при изменении L, на этот раз для случая, когда $\prec$ пусто и

$$\max\{\tau_i\}/\min\{\tau_i\} \leqslant 4.$$

Неизвестно, является ли константа 4 лучшей из возможных. Во всяком случае, мы видим, что достижимость наихудшего результата не зависит от наличия заданий с большим разбросом времен выполнения.

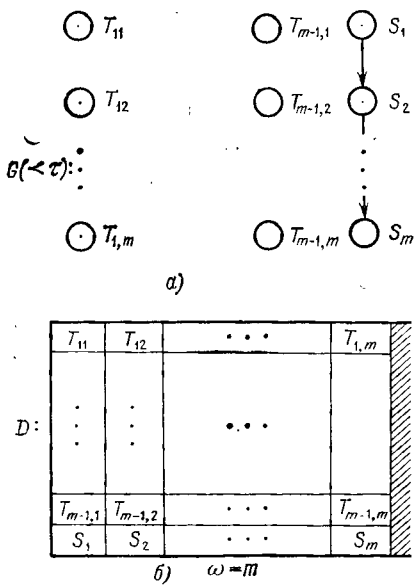


Рис. 5.15. Система заданий и оптимальное расписание;
$L = (T_{11}, \ldots, T_{m-1,1}, S_1, T_{12}, \ldots, T_{m-1,2}, S_2, T_{21}, S_m)$.

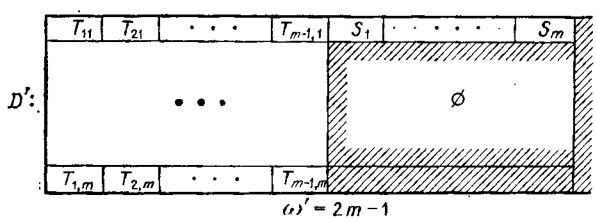


Рис. 5.16. Очень плохое расписание;
$L = (T_{11}, T_{12}, \ldots, T_{1,m}, \ldots, T_{m-1,1}, T_{m-1,m}, S_1, \ldots, S_m)$.

Пример 8. Пусть $\prec$ пусто, и предположим, что максимум отношения времен выполнения не превышает 4. Рассмотрим три случая в зависимости от величины $m \pmod 4$. Для $m=2r$ положим, что набор $\tau=(\tau_1, \ldots, \tau_{2m+1})$ задан в виде

$(r, r, r+1, r+1, \ldots, 2r-2, 2r-2, 2r-1, 2r-1,$
$3r-2, 3r-2, 3r-3, 3r-3, \ldots, 2r-1, 2r-1, 4r).$

Тогда с помощью соответствующего списка $L=(T_1, \ldots, T_{2m+1})$ получим расписание, показанное на рис. 5.17, *а*.

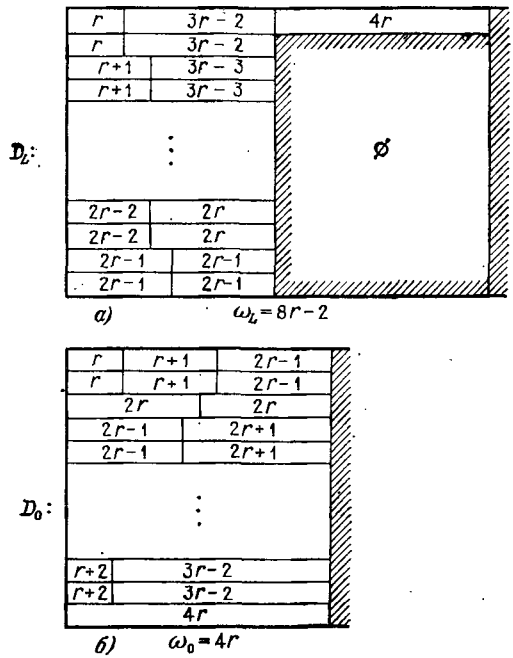


Рис. 5.17.

Поскольку для оптимального расписания, изображенного на рис. 5.17, *б*, $\omega_0=m$, получаем

$$\frac{\omega_L}{\omega_0} = 2 - \frac{1}{2r} = 2 - \frac{1}{m}.$$

Для $m=4r+1$ пусть набор τ задан в виде

$(2r+1, 2r+1, 2r+1, 2r+1, 2r+3, 2r+3, 2r+3,$
$2r+3, \ldots, 4r-1, 4r-1, 4r-1, 4r-1, 4r, 6r-1, 6r-1,$
$6r-1, 6r-1, 6r-3, 6r-3, 6r-3, 6r-3, \ldots, 4r+1,$
$4r+1, 4r+1, 4r+1, 4r, 8r+2).$

В этом случае, используя список $L=(T_1, \ldots, T_{2m+1})$, получим расписание, изображенное на рис. 5.18, *а*, в то время как оптимальное расписание имеет вид, изображенный на

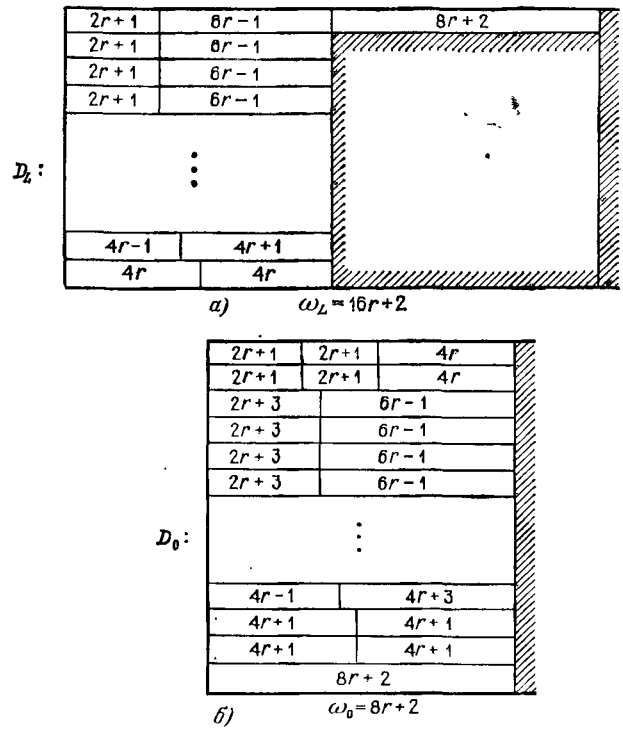


Рис. 5.18.

рис. 5.18, *б*. Снова получаем

$$\frac{\omega_L}{\omega_0}=2-\frac{1}{4r+1}=2-\frac{1}{m}.$$

Наконец, пусть $m=4r+3$ и набор τ задан в виде

$(r+1,\ r+1,\ r+1,\ r+1,\ r+2,\ r+2,\ r+2,$
$r+2,\ \ldots,\ 2r+1,\ 2r+1,\ 2r+1,\ 3r+1,\ 3r+1,\ 3r+1,$
$3r+1,\ 3r,\ 3r,\ 3r,\ \ldots,\ 2r+2,\ 2r+2,\ 2r+2,\ 2r+2,$
$2r+1,\ 2r+1,\ 2r+1,\ 4r+3).$

Расписание, составленное при этих параметрах с помощью списка $L=(T_1, \ldots, T_{2m+1})$, представлено на

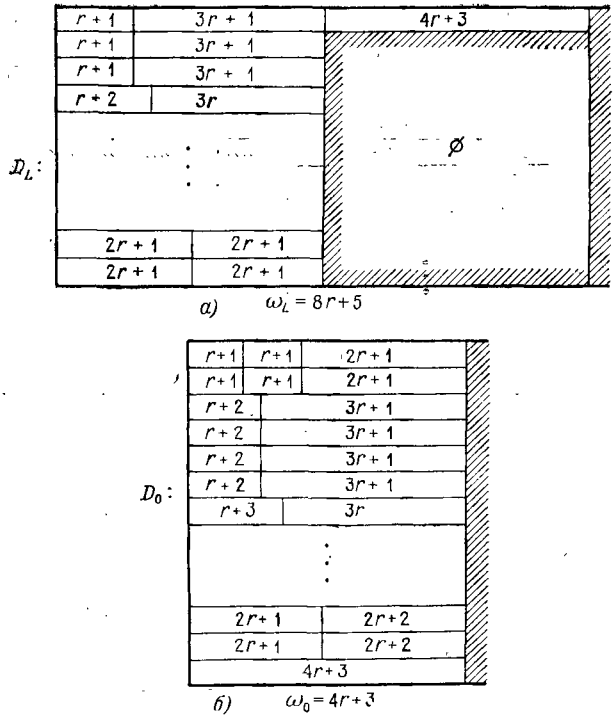


Рис. 5.19.

рис. 5.19, *а*. Оптимальное расписание изображено на рис. 5.19, *б*, и оно показывает, что

$$\frac{\omega_L}{\omega_0} = 2 - \frac{1}{4r+3} = 2 - \frac{1}{m}.$$

5.2. Оценки для независимых заданий без дополнительных ресурсов

В этом параграфе мы исследуем частный случай, когда отношение частичного порядка $<$ пусто и, как и ранее, $s=0$ (т. е. отсутствуют ограничения, связанные с дополнительными ресурсами). Как показано в примере 3, неудачный выбор списка L может привести к наихудшему времени завершения ω, а именно $\omega=(2-1/m)\omega_0$. Однако если при формировании L принять небольшие меры предосторожности, то оценка для ω может быть значительно улучшена.

Рассмотрим в качестве первого алгоритма составления

расписания для системы $\mathscr{T}$ один из вариантов алгоритма упорядочения *по критическому пути* (см. гл. 1). В этом случае список L составляется просто путем расположения T_i в порядке убывания величин τ_i. Обозначим соответствующее время завершения через ω_{CP} (CP — critical path).

Т е о р е м а 5.2 [46]. *Если* $\prec$ *пусто, то*

$$\frac{\omega_{CP}}{\omega_0} \leqslant \frac{4}{3} - \frac{1}{3m}. \tag{7}$$

Более того, существуют примеры, в которых достигается равенство.

Д о к а з а т е л ь с т в о. Предположим, что $\mathscr{T} = \{T_1, \ldots, T_n\}$ — набор заданий, с временами выполнения τ_i, такой, что (7) не выполняется. Более того, мы можем предположить, что T_i пронумерованы так, что $\tau_1 \geqslant \tau_2 \geqslant \ldots \geqslant \tau_n$. Для $m=1$ теорема, очевидно, верна. Поэтому мы можем предположить, что $m \geqslant 2$ и что n есть минимальное значение, для которого теорема не выполняется.

Вначале заметим, что, как следует из определения ω_{CP}, порядок, в котором выполняются задания, в точности соответствует тому, который имеется в списке $L = (T_1, T_2, \ldots, T_n)$. Предположим, что на соответствующей временной диаграмме D_L имеется задание T_r такое, что $r < n$ и $f_r = \omega_{CP}$. Если мы рассмотрим подмножество $\mathscr{T}' = \{T_1, \ldots, T_r\}$ и список $L' = (T_1, \ldots, T_r)$, то увидим, что для $\mathscr{T}'$ время завершения ω' при использовании L' в точности равно ω_{CP}. С другой стороны, оптимальное значение ω'_0 для $\mathscr{T}'$, очевидно, удовлетворяет условию $\omega'_0 \leqslant \omega_0$. Следовательно, мы имеем

$$\frac{\omega'}{\omega'_0} \geqslant \frac{\omega_{CP}}{\omega_0} > \frac{4}{3} - \frac{1}{3m};$$

поэтому $\mathscr{T}'$ образует меньший контрпример для данной теоремы, что противоречит минимальности n. Следовательно, мы можем предположить, что $f_k < \omega_{CP}$ для $k < n$.

Ясно, что

$$\omega_0 \geqslant \frac{1}{m} \sum_{i=1}^{n} \tau_i. \tag{8}$$

Кроме того, имеем

$$\sum_{i=1}^{n-1} \tau_i \geqslant m s_n, \quad \omega_L = s_n + \tau_n, \tag{9}$$

где s_n обозначает время начала выполнения T_n, поскольку ни один процессор не простаивал до того, как начало

выполняться T_n. Следовательно, мы можем записать

$$\frac{\omega_L}{\omega_0} = \frac{s_n + \tau_n}{\omega_0} \leqslant \frac{\tau_n}{\omega_0} + \frac{1}{m\omega_0} \sum_{i=1}^{n-1} \tau_i =$$

$$= \frac{(m-1)\,\tau_n}{m\omega_0} + \frac{1}{m\omega_0} \sum_{i=1}^{n} \tau_i \leqslant \frac{(m-1)\,\tau_n}{m\omega_0} + 1.$$

Поскольку по предположению для $\mathscr{T}$ (7) не выполняется, получаем

$$1 + \frac{(m-1)\,\tau_n}{m\omega_0} \geqslant \frac{\omega_L}{\omega_0} > \frac{4}{3} - \frac{1}{3m},$$

откуда

$$\frac{(m-1)\,\tau_n}{m\omega_0} > \frac{1}{3} - \frac{1}{3m} = \frac{m-1}{3m},$$

или

$$\tau_n > \omega_0/3. \qquad (10)$$

Поэтому, если (7) не верно, то в оптимальном решении (с временной диаграммой D_0) ни один процессор не может выполнять больше двух заданий.

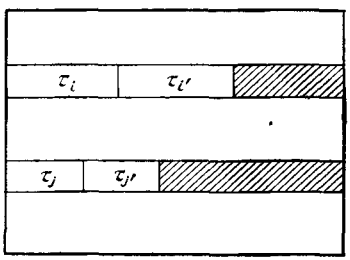


Рис. 5.20.

Предположим, что диаграмма, изображенная на рис. 5.20, представляет собой фрагмент D_0, в котором $\tau_i > \tau_j$, $\tau_{i'} > \tau_{j'}$. Если поменять местами $\tau_{i'}$ и $\tau_{j'}$ (получив диаграмму, изображенную на рис. 5.21), то новое время завершения ω' в D_0', очевидно, будет удовлетворять неравенству $\omega' \leqslant \omega_0$. Аналогично, если встречается конфигурация, показаиная на рис. 5.22, где $\tau_i > \tau_j$, то в ней перемещение $T_{i'}$ с процессора, выполняющего T_i, на процессор, выполняющий T_j, не может привести к увеличению времени завершения.

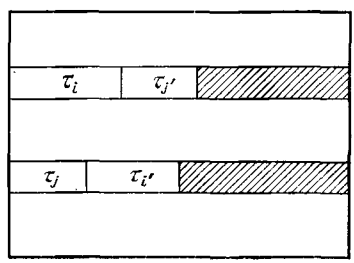


Рис. 5.21.

Назовем каждое из двух вышеописанных действий операцией *типа I*.

Под операцией *типа II* на D_0 мы подразумеваем преобразование любой «инверсии» типа той, что показана на

рис. 5.23, в «нормализованную» форму, изображенную на рис. 5.24. Очевидно, такая операция не меняет ω_0.

Для произвольной временной диаграммы D определим функцию $\mathscr{S}(D)$ следующим образом. Пусть τ_i^* обозначает

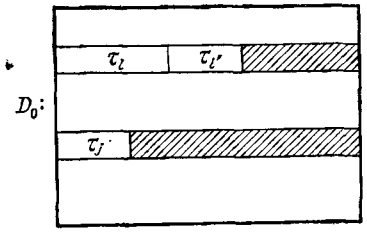


Рис. 5.22.

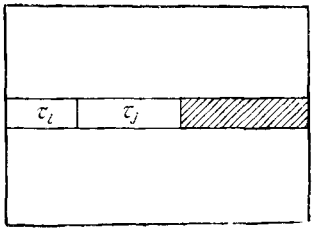


Рис. 5.23.

наименьшее время t такое, что для всех $t' \geqslant t$ процессор P_i находится в D в состоянии простоя; тогда

$$\mathscr{S}(D) = \sum_{1 \leqslant i < j \leqslant m} |\tau_i^* - \tau_j^*|.$$

Нетрудно убедиться в следующем:

1) если D' получено из D с помощью операции типа I, то $\mathscr{S}(D') < \mathscr{S}(D)$;

2) если D' получено из D с помощью операции типа II, то $\mathscr{S}(D') = \mathscr{S}(D)$.

Теперь, начиная с D_0, будем применять все возможные операции типов I и II до тех пор, пока в получившейся в результате временной диаграмме D^* не останется внутренних конфигураций, к которым эти правила можно было бы применить. Существование такой диаграммы D^* вытекает из следующих соображений: (а) имеется конечное число возможных вариантов назначений n заданий на m процессоров, (б) между любыми двумя операциями типа I может быть выполнено лишь конечное число операций типа II, и (в) вследствие условия 1) может быть выполнено лишь конечное число операций типа I. Из этого следует, что в D^* для любой конфигурации, подобной изображенной на рис. 5.25, имеем:

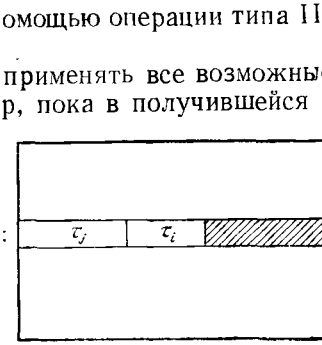


Рис. 5.24.

$$\tau_i > \tau_j \text{ влечет } \tau_{j'} \geqslant \tau_{i'},\ \tau_i \leqslant \tau_k \text{ и } \tau_i > \tau_{i'}. \qquad (11)$$

Таким образом, с помощью соответствующего переназначения процессоров для D^* мы можем довести D^* до вида, изображенного на рис. 5.26, где

$$\tau_{k_1} \geqslant \tau_{k_2} \geqslant \ldots \geqslant \tau_{k_s},$$

$$\tau_{i_1} \geqslant \tau_{i_2} \geqslant \ldots \geqslant \tau_{i_t}$$

и, на основании (11),

$$\tau_{i'_1} \leqslant \tau_{i'_2} \leqslant \ldots \leqslant \tau_{i'_t}.$$

Но (11) влечет также $\tau_{k_s} \geqslant \tau_{i_1}$ и $\tau_{i_t} \geqslant \tau_{i'_t}$. Следовательно, объединяя эти неравенства, получим

$$\tau_{k_1} \geqslant \ldots \geqslant \tau_{k_s} \geqslant \tau_{i_1} \geqslant \ldots \geqslant \tau_{i_t} \geqslant \tau_{i'_t} \geqslant \ldots \geqslant \tau_{i'_1}. \quad (12)$$

Поскольку ни одна из операций, совершенных над D_0, не приводит к увеличению ω_0, то ввиду оптимальности ω_0 время завершения $\overline{D}$ должно также быть равным ω_0. Но

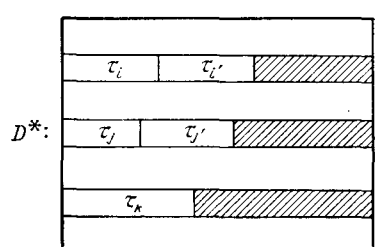


Рис. 5.25.

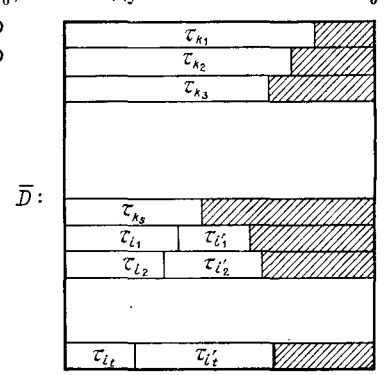


Рис. 5.26.

$\overline{D}$ теперь стала очень похожей на временную диаграмму D_L, которая получена с помощью списка $L=(T_1, \ldots, T_n)$, упорядоченного по убыванию длительностей (с точностью до перенумерации заданий равной длины). Фактически, единственное отличие D_L от $\overline{D}$ может возникнуть только при назначении задания $T_{i'_k}$, выполняемого на своем процессоре вторым. Точнее, оно может появиться только в том случае, если для некоторой пары T_{i_k}, $T_{i'_k}$ возникнет ситуация, изображенная на рис. 5.27, где $\tau_{i_k} + \tau_{i'_k} \leqslant \tau_{i_r}$ для $r < k$. В этом случае в D_L задание $T_{i'_r}$, имеющее длину $\tau_{i'_r}$, должно быть назначено на P_j вместо P_i. Однако если бы

эта ситуация была возможной, то мы получили бы в $\overline{D}$ картину, показанную на рис. 5.28. Следовательно, было бы возможно переместить $T_{i'_r}$ с P_i на P_j. Кроме того,

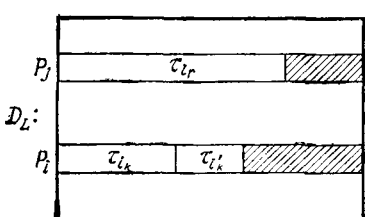


Рис. 5.27.

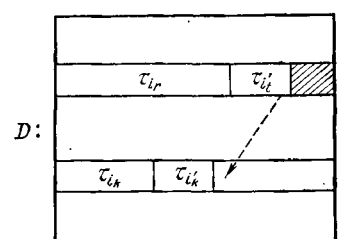


Рис. 5.28.

поскольку время завершения не увеличивается, оно по-прежнему равно ω_0. Но это — противоречие, поскольку мы получили оптимальное решение, в котором на один процессор назначены три задания. Отсюда заключаем, что D_L и $\overline{D}$ изоморфны (в очевидном смысле) и $\omega_0 = \omega_{CP}$. Но это противоречит гипотезе, что $\omega_{CP}/\omega_0 > 4/3 - 1/(3m)$; следовательно, (7) доказано.

Чтобы убедиться в том, что в (7) возможно равенство, рассмотрим множество заданий, имеющих длины

$$(\tau_1, \ldots, \tau_n) = (2m-1,\ 2m-1,\ 2m-2,\ 2m-2,\ \ldots,\ m+1,\ m+1,\ m,\ m,\ m),$$

где $n = 2m+1$, т. е.

$$\tau_k = 2m - \lfloor (k+1)/2 \rfloor,$$
$$1 \leqslant k \leqslant 2m, \text{ и } \tau_{2m+1} = m.$$

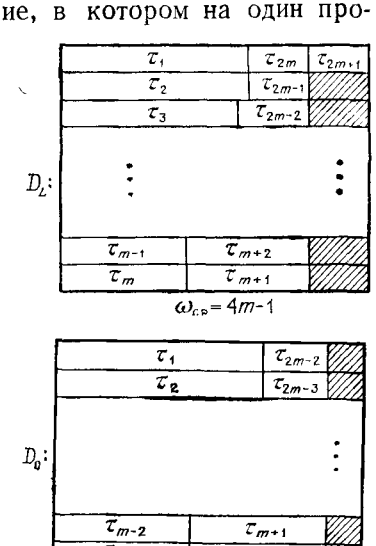


Рис. 5.29.

Соответствующие временные диаграммы для ω_{CP} и ω_0 приведены на рис. 5.29. Можно убедиться в том, что

$$\frac{\omega_L}{\omega_0} = \frac{4}{3} - \frac{1}{3m},$$

что и требовалось доказать.

Теперь мы докажем следующий интуитивно ясный результат: по мере того, как отношение наибольшего времени выполнения одного задания к сумме времен выполнения всех заданий стремится к нулю, отношение ω/ω_0 стремится к 1.

Теорема 5.3. *Пусть* $<$ *пусто*; *тогда*

$$\frac{\omega}{\omega_0} \leqslant 1 + (m-1)\frac{\max\limits_i \tau_i}{\sum\limits_i \tau_i}. \tag{13}$$

Доказательство. Положим $\tau^* = \max\limits_i \tau_i$. Как следует из правил, в соответствии с которыми работает система, ни один процессор не простаивает до момента $\omega - \tau^*$. Поскольку по меньшей мере один процессор оказывается занятым в течение всего времени ω, мы видим, что

$$\sum_i \tau_i \geqslant \omega + (m-1)(\omega - \tau^*).$$

Следовательно,

$$\omega_0 \geqslant \frac{1}{m}(\omega + (m-1)(\omega - \tau^*)),$$

т. е.

$$\frac{\omega}{\omega_0} \leqslant 1 + \frac{(m-1)}{m}\frac{\tau^*}{\omega_0} \leqslant 1 + (m-1)\frac{\max\limits_i \tau_i}{\sum\limits_i \tau_i},$$

и теорема доказана.

Следующий результат помогает получить количественную оценку зависимости между стоимостью составления «частично» оптимального расписания и соответствующим уменьшением отношения ω/ω_0.

Теорема 5.4 [46]. *Пусть* $<$ *пусто и* L_k *— список из* k *заданий* T_i, *имеющих наибольшее* τ_i, $1 \leqslant i \leqslant k$, *который является оптимальным для этого набора заданий. Построим список* $L(k)$ *путем добавления оставшихся заданий в произвольном порядке, и пусть* $\omega(k)$ *обозначает время завершения при использовании* $L(k)$. *Тогда*

$$\frac{\omega(k)}{\omega_0} \leqslant 1 + \frac{1 - 1/m}{1 + \lfloor k/m \rfloor}. \tag{14}$$

Эта оценка является наилучшей при $k \equiv 0 \pmod m$.

Доказательство. Пусть ω_k — время завершения списка L_k. Если $\omega(k) = \omega_k$, то $\omega(k) = \omega_0$, и теорема справедлива. Поэтому можно предполагать, что $\omega(k) > \omega_k$.

Можно также предполагать, что $n>k$. Пусть $\tau^* = \max\limits_{k+1\leqslant j\leqslant n} \tau_j$. Как и в доказательстве предыдущей теоремы, при использовании списка $L(k)$ ни один процессор не может простаивать до момента $\omega(k) - \tau^*$. Следовательно,

$$\sum_{j=1}^{n} \tau_j \geqslant m(\omega(k) - \tau^*) + \tau^*,$$

откуда

$$\omega_0 \geqslant \frac{1}{m}\sum_{j=1}^{n} \tau_j \geqslant \omega(k) - \left(\frac{m-1}{m}\right)\tau^*.$$

Существуют по меньшей мере $k+1$ заданий, имеющих длину не менее τ^*. Следовательно, какой-то процессор должен выполнить не менее $1+\lfloor k/m \rfloor$ из этих «длинных» заданий. Из этого следует

$$\omega_0 \geqslant (1 + \lfloor k/m \rfloor)\,\tau^*.$$

Объединяя предыдущие неравенства, получим

$$\omega(k) \leqslant \omega_0 + \left(\frac{m-1}{m}\right)\tau^* \leqslant \omega_0\left(1 + \frac{1-1/m}{1+\lfloor k/m \rfloor}\right),$$

что доказывает (14).

Чтобы убедиться в том, что оценка (14) неулучшаема при $k\equiv 0 \pmod m$, рассмотрим следующий пример.

Пример 9. Определим τ_i для $1\leqslant i\leqslant k+1+m(m-1)$ следующим образом:

$$\tau_i = \begin{cases} m & \text{для } 1\leqslant i\leqslant k+1, \\ 1 & \text{для } k+2\leqslant i\leqslant k+1+m(m-1). \end{cases}$$

Для этого набора времен выполнения и списка приоритетов

$$L(k) = (T_1, \ldots, T_k, T_{k+2}, \ldots, T_{k+1+m(m-1)}, T_{k+1})$$

получаем $\omega(k) = k+2m-1$. Поскольку $\omega_0 = k+m$, то

$$\frac{\omega(k)}{\omega_0} = \frac{k+2m-1}{k+m} = 1 + \frac{1-1/m}{1+\lfloor k/m \rfloor},$$

так как $k\equiv 0 \pmod m$, и равенство в (14) достигнуто. Это доказывает теорему 5.4.

При достаточно больших k, мы, таким образом, получаем лучшую верхнюю оценку для $\omega(k)$, чем дает (7) для $\omega_{\text{ср}}$. Однако поиск оптимального списка для k первых заданий при больших k сам по себе является трудной задачей. Вместо этого при больших k можно попытаться начинать со списка, *близкого* к оптимальному.

Пусть L'_k обозначает список, сформированный из k наибольших значений τ_i, такой, что для некоторого $\alpha \geqslant 0$ выполняется

$$\omega'_k/\omega_0 \leqslant 1+\alpha,$$

и пусть $L'(k)$ образуется из L'_k путем добавления оставшихся заданий. Тогда предыдущие аргументы могут быть использованы для доказательства следующего обобщения теоремы 5.4.

Теорема 5.4′. *Для L'_k, определенного выше, справедливо соотношение*

$$\frac{\omega'(k)}{\omega_0} \leqslant 1+\max\left\{\alpha, \quad \frac{m-1}{m\,[(k+1)/m]}\right\}. \qquad (15)$$

Более специфический метод нахождения списка приоритетов, близкого к оптимальному, рассмотрен в [73]. Здесь вместо значения величины ω_0 используется ее предполагаемое значение ω'_0 и частичное решение строится следующим образом. Задания упорядочиваются в порядке невозрастания величин τ_i. Первое (наибольшее) задание назначается первым в числе выполняемых на первом процессоре. В общем случае i-е задание назначается очередным заданием на процессоре с наименьшим номером, на котором оно может быть выполнено без нарушения крайнего срока ω'_0. Процесс останавливается, когда появляется задание, которое невозможно назначить ни на один процессор, не нарушив при этом крайнего срока. Пусть L_1 обозначает список заданий, назначенных к моменту остановки описанного процесса, и L_2 обозначает оставшуюся часть списка. Пусть L'_1 — перестановка элементов L_1, которая, будучи использованной в качестве списка приоритетов, дает такое же расписание, как построенное выше, и пусть L' — список, полученный из L'_1 путем добавления к нему L_2. Доказан следующий результат.

Теорема [73]. *Для L' и ω'_0, определенных выше, выполнение условия $\omega'_0 \geqslant \omega_0$ влечет*

$$\omega_{L'}/\omega'_0 \leqslant 5/4.$$

Это выражение непосредственно не дает оценку для $\omega_{L'}/\omega_0$. Однако, чем ближе предполагаемое значение ω'_0 к ω_0, тем лучше будет отношение $\omega_{L'}/\omega_0$. Более того, можно использовать повторные применения алгоритма для получения более хороших предполагаемых значений с помощью методов двоичного поиска. Пусть $L'(j)$ обозначает список, полученный в результате j-й итерации. Имеет место следующий результат.

Теорема [73]. *Справедливо неравенство*

$$\frac{\omega_{L'(j)}}{\omega_0} \leqslant \frac{5}{4}\left(1+\frac{1}{2^{j-1}}\right).$$

Это выражение будет давать лучшую, чем (7), оценку для ω_{CP} при $m \geqslant 5$ и $j \geqslant 8$.

Могут, однако, возразить, что метод составления расписания с помощью нахождения критического пути для независимых заданий (т. е. пустого $<$) дает для других показателей эффективности лучшие оценки, чем (7). В частности, естественно рассматривать в качестве такого показателя величину $\omega^*(L)$ (см. [33]), которая для данного списка L определяется выражением

$$\omega^*(L) = \sum_{i=1}^{m} \omega_i^2(L),$$

где $\omega_i(L)$ — время, за которое процессор P_i обслуживает задания, упорядоченные с помощью списка L. Этот показатель эффективности используется при изучении сегментации массивов во внешней памяти с целью минимизации времени доступа. Доказан следующий результат.

Теорема [33]. *Справедливо неравенство*

$$\omega^*_{CP}/\omega^*_0 \leqslant 25/24.$$

С другой стороны, в [33] приводится пример, для которого

$$\omega^*_{CP}/\omega^*_0 \geqslant 37/36 - 1/(36m).$$

Сложность доказательства не позволила нам включить его в книгу.

В другой задаче распределения внешней памяти рассмотрен сходный показатель эффективности, а именно минимизируется функция $(m-2)/2+(m/2)\omega^*$. Для этого случая доказано неравенство [21]

$$\frac{(m-2)/m+\omega^*_{CP}}{(m-2)/m+\omega^*_0} \leqslant 1+\frac{1}{16(m-1)},$$

которое снова показывает увеличение эффективности СР-назначений при использовании квадратичных критериев.

Наконец, СР-упорядочение применялось также для получения достаточно «коротких» расписаний, в которых обеспечивается минимальное среднее время завершения (см. гл. 3). В частности, предположим, что назначение независимых заданий осуществляется следующим образом. Первыми назначаются m наименьших заданий (ранга 1) по одному на каждый процессор. Следующие m наименьших заданий (ранга 2) назначаются следом по одному на процессор по

мере завершения заданий ранга 1, причем первыми назначаются наибольшие задания. Этот процесс продолжается для последующих рангов до тех пор, пока все задания не будут назначены и на каждом процессоре не окажутся задания всех рангов (предполагается, что n кратно m). Для критерия «первым обслуживается наибольшее задание» (СР-критерий) можно показать [32], что оценка

$$\omega'_{СР}/\omega'_0 \leqslant (5m-4)/(4m-3)$$

оптимальна; здесь штрих означает, что рассматриваются только те расписания, которые минимизируют среднее время прохождения.

5.3. Замечания о составлении расписаний методом критического пути

Как показывает пример 7, даже в том случае, когда G представляет собой дерево и все $\tau_i = 1$, возможна ситуа-

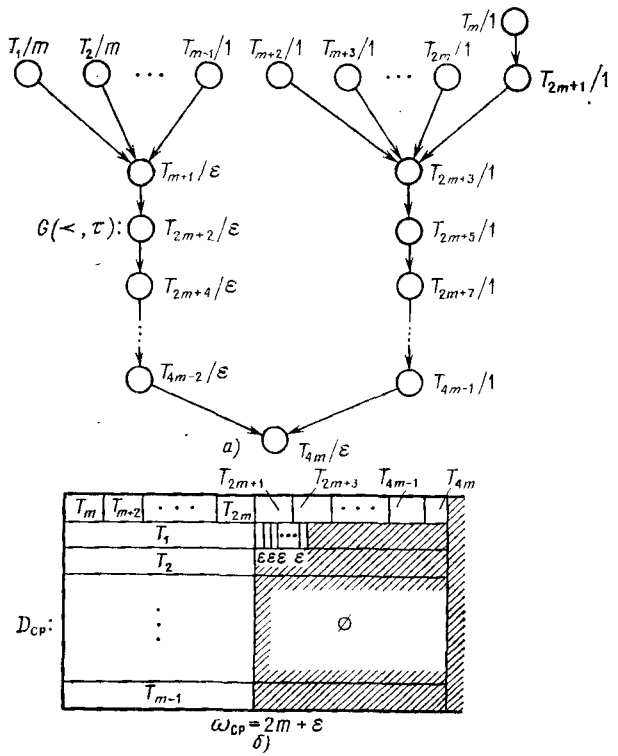


Рис. 5.30.

ция, в которой $\omega_L/\omega_0 = 2 - 1/m$ при достаточно плохом списке L. Если при этом используется метод составления расписаний с помощью критического пути, то, как известно, $\omega_{CP} = \omega_0$ (см. гл. 2). Следующий пример, предложенный Грэхемом [61], показывает, что если допустить произвольные τ_i, то даже в случае, когда $<$ — дерево, отношение ω_{CP}/ω_0 может быть очень близким к 2.

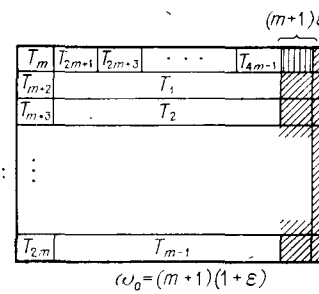


Рис. 5.31.

Пример 10. Рассмотрим дерево и расписание, составленное методом критического пути, представленные на рис. 5.30, *а* и *б* соответственно. Оптимальный список может привести к расписанию, представленному на рис. 5.31. Эти диаграммы показывают, что

$$\frac{\omega_{CP}}{\omega_0} = \frac{2m+\varepsilon}{(m+1)(1+\varepsilon)};$$

в пределе при $\varepsilon \to 0$ получаем $2 - 2/(m+1)$. Предположительно, это асимптотически наихудшее поведение ω_{CP} в случае, когда $<$ является деревом.

Кауфман показал [85], что если $<$ является деревом, то

$$\frac{\omega_{CP}}{\omega_0} \leqslant 1 + (m-1)\frac{\max\limits_i \tau_i}{\sum\limits_i \tau_i};$$

этот результат аналогичен оценке теоремы 5.3 для пустого $<$ и произвольного списка L.

Однако, как показывает следующий пример, при частичном порядке общего вида и произвольных значениях τ_i расписание, составленное методом критического пути, может оказаться наихудшим из возможных.

Пример 11. Пусть имеется граф, изображенный на рис. 5.32, *а*. В результате применения метода критического пути получим расписание, показанное на рис. 5.32, *б*. Оптимальный список имеет вид

$$L = (\varepsilon_1, \ldots, \varepsilon_m, U_1, \ldots, U_m, T_1, \ldots, T_m).$$

Соответствующее ему расписание представлено на рис. 5.33. Очевидно, что

$$\frac{\omega_{\text{СР}}}{\omega_0} = \frac{2m-1-2\varepsilon}{m} \to 2-\frac{1}{m} \quad \text{при} \quad \varepsilon \to 0.$$

Это, как мы уже видели, соответствует наихудшему случаю, какой только может иметь место для списочного рас-

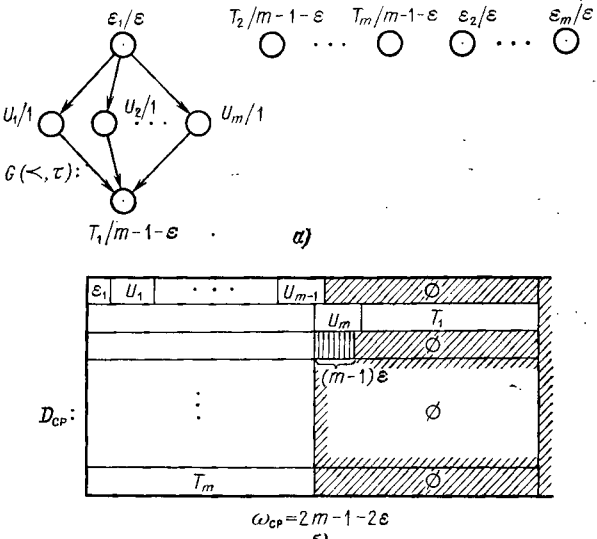


Рис. 5.32.

писания. Заметим, что этот пример также показывает, что если в качестве очередного задания, подлежащего выполнению, выбирается то, которое имеет наибольшее значение суммы $\sum_{T \prec T_i} \tau_i$, то величина $\bar{\omega}/\omega_0$ может быть сколь угодно близкой к $2-1/m$; здесь $\bar{\omega}$ — длина соответствующего расписания.

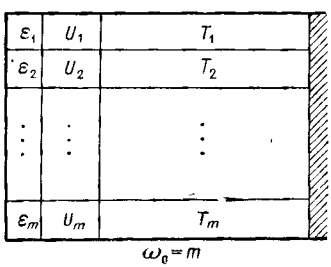


Рис. 5.33.

Если допустить произвольное частичное упорядочение $\prec$, но теперь положить все τ_i равными 1, то оценка для наихудшего варианта расписания, составленного мето-

дом критического пути, задается выражением [27]

$$\frac{\omega_{CP}}{\omega_0} \leqslant 2 - \frac{1}{m-1}, \quad m \geqslant 3. \tag{16}$$

В [27] приведены также примеры, из которых следует, что эта оценка является лучшей из возможных. Рассмотрим такой пример для $m=3$.

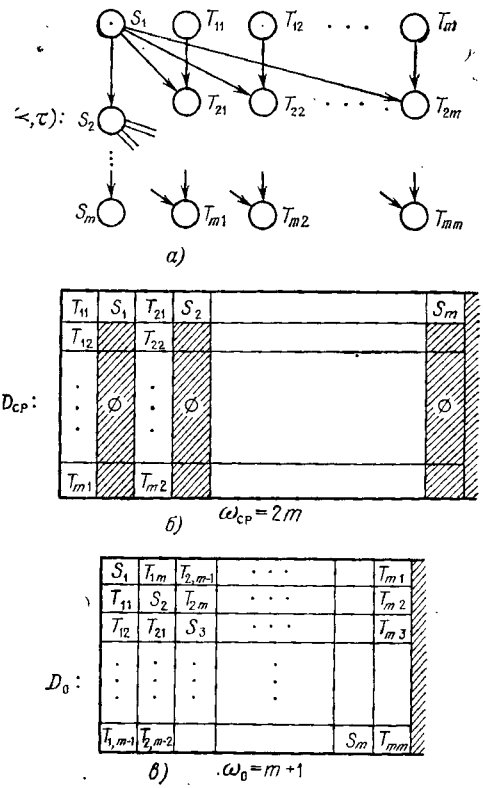


Рис. 5.34.

Пример 12. Пусть в графе, изображенном на рис. 5.34, *а*, все $\tau_i=1$. Один из возможных списков, составленных методом критического пути (в котором осуществлен наихудший из возможных способов разрешения конфликтных ситуаций), имеет вид $L=(T_1, T_2, T_3, \ldots, T_{12})$. Соответствующее расписание показано на рис. 5.34, *б*. Оптимальное расписание, которому соответствует список $L_0=(T_4, T_8, T_1, T_2, T_3, T_5, T_6, T_7, T_9, T_{10}, T_{11}, T_{12})$, приве-

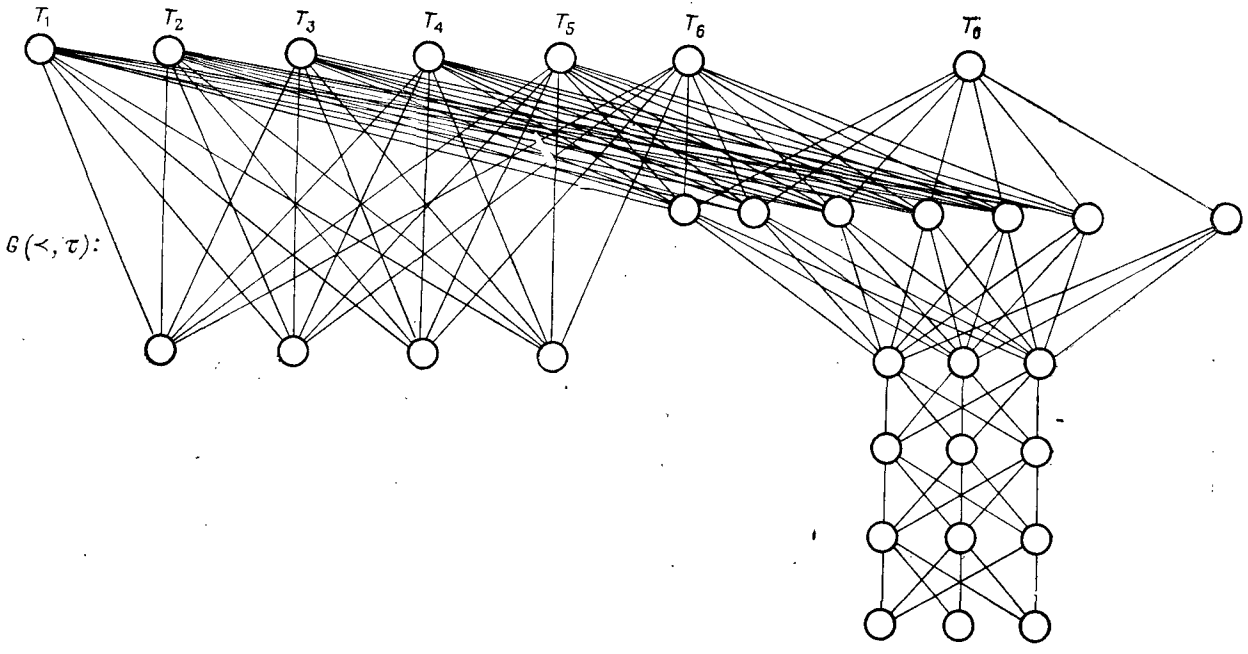


Рис. 5.35.

дено на рис. 5.34, *в*. Сравнивая два расписания, находим

$$\omega_{СР}/\omega_0 = 3/2,$$

что в точности совпадает с оценкой (16) для $m=3$.

Наконец, приведем несколько неожиданный пример, показывающий, что даже если все $\tau_i=1$, то может не найтись списка L, который бы обеспечивал оптимальное выполнение $\mathscr{T}$ как при $m=2$, так и при $m=3$.

Пример 13. Рассмотрим граф, изображенный на рис. 5.35, в котором все τ_i предполагаются равными 1. В любом оптимальном списке для $m=2$ T_0 должно предшествовать *некоторым* заданиям T_i, $1\leqslant i\leqslant 6$. В любом оптимальном списке для $m=3$ *все* T_i, $1\leqslant i\leqslant 6$, должны предшествовать T_0!

5.4. Составление расписаний при наличии многих ресурсов

В этом параграфе мы рассмотрим общий случай, когда число типов ресурсов s отлично от нуля. Напомним, что $\mathscr{R}=\{R_1, \ldots, R_s\}$ обозначает множество ресурсов, $R_i(T_j)$ обозначает объем ресурса типа R_i, необходимый в течение всего времени выполнения задания T_j, $r_i(t)$ обозначает (для фиксированного расписания) общий объем ресурса типа R_i, используемый в момент t. Как и следовало ожидать, включение в задачу дополнительных ресурсов приводит к тому, что величина ω/ω_0 может стать намного больше, чем прежде. Это ясно видно из следующей теоремы и соответствующего примера.

Теорема 5.5 [49]. *Для $s=1$, произвольных L, $\prec$, τ и m выполняется условие*

$$\omega/\omega_0 \leqslant m. \tag{17}$$

Доказательство. Справедливость (17) доказывается очень просто. Для этого достаточно лишь заметить, что

$$\omega \leqslant \sum_{i=1}^{n} \tau_i \leqslant m\omega_0,$$

поскольку при использовании списка L до момента ω все время имеется хотя бы один работающий процессор, а число одновременно занятых процессоров никогда не превышает m.

Для того чтобы убедиться в том, что (17) является наилучшей возможной оценкой, рассмотрим следующий пример.

Пример 14. Пусть $\mathcal{T}=\{T_1, \ldots, T_m, T_1', \ldots, T_m'\}$, $s=1$, $R_1(T_i)=1/m$, $R_1(T_i')=1$, $1\leqslant i\leqslant m$, $\tau_i=1$, $\tau_i'=\varepsilon>0$.

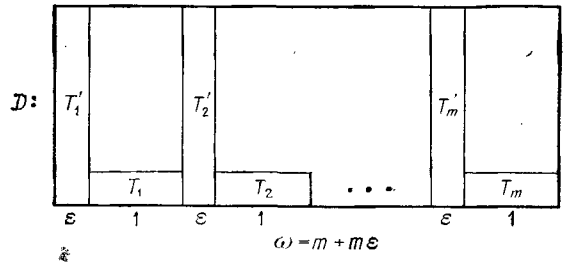


Рис. 5.36.

Пусть $\prec$ определено так, что $T_i'\prec T_i$ для $1\leqslant i\leqslant m$. Для списков

$$L=(T_1, \ldots, T_m, T_1', \ldots, T_m'),$$
$$L'=(T_1' \ldots, T_m', T_1, \ldots, T_m)$$

получаем расписания, изображенные на рис. 5.36 и 5.37 соответственно. Сравнивая эти расписания, получаем $\omega/\omega'=(m+m\varepsilon)/(1+m\varepsilon)\longrightarrow m$ при $\varepsilon\longrightarrow 0$.

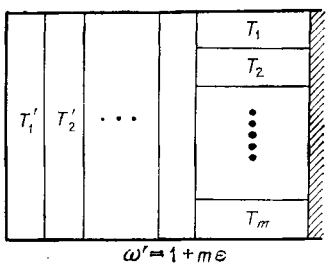


Рис. 5.37.

Более интересная оценка дается следующей теоремой.

Теорема 5.6 [50]. *Если множество $\prec$ пусто, $m\geqslant n$, s, L, τ — произвольные, то*

$$\omega/\omega_0\leqslant s+1.$$

Доказательство. Вначале докажем ряд предварительных результатов.

Пусть G — граф с множеством вершин $V=V(G)$ и множеством ребер $E=E(G)$. *Правильной разметкой* λ графа G назовем функцию λ: $V\to[0, \infty)$, удовлетворяющую условию:

$$\lambda(a)+\lambda(b)\geqslant 1 \text{ для всех } e=\{a, b\}\in E. \qquad (18)$$

Определим величину $\tau^*(G)$ следующим образом:

$$\tau^*(G)=\inf_{\lambda}\sum_{v\in V}\lambda(v),$$

где нижняя грань берется по всем правильным разметкам λ графа G.

Лемма 5.1. *Для любого графа G существует правильная разметка $\lambda : V \to \{0, 1/2, 1\}$ такая, что*

$$\tau^*(G) = \sum_{v \in V} \lambda(v).$$

Доказательство. Для случая, когда G является двудольным графом (т. е. G не имеет нечетных циклов), хорошо известная теорема Кенига утверждает, что количество ребер в максимальном паросочетании равно мощности минимального множества вершин G, инцидентных каждому ребру G. Таким образом, для двудольного графа G существует правильная разметка $\lambda : V \to \{0, 1\}$ такая, что

$$\tau^*(G) = \sum_{v \in V} \lambda(v).$$

Для произвольного графа G построим двудольный граф G_B следующим образом. Для каждой вершины $v \in V(G)$ образуем две вершины v_1, $v_2 \in V(G_B)$; для каждого ребра $\{u, v\} \in E(G)$ образуем два ребра $\{u_1, v_2\}$, $\{u_2, v_1\} \in E(G_B)$. Нетрудно убедиться, что $\tau^*(G_B) = 2\tau^*(G)$, и более того, если $\lambda_B : V(G_B) \to \{0, 1\}$ является правильной разметкой G_B, то разметка $V(G) \to \{0, 1/2, 1\}$, определенная формулой $\lambda(v) = \frac{1}{2}(\lambda(v_1) + \lambda(v_2))$, является правильной разметкой G.

Для положительных целых чисел k и s обозначим через $G(k, s)$ граф, в котором множество вершин есть $\{0, 1, \ldots, (s+1)k-1\}$, а множество ребер состоит из всех пар $\{a, b\}$ таких, что $|a-b| \geqslant k$.

Лемма 5.2. *Предположим, что $G(k, s)$ разбит на s остовных подграфов H_i, $1 \leqslant i \leqslant s$. Тогда*

$$\max_{1 \leqslant i \leqslant s} \{\tau^*(H_i)\} \geqslant k. \tag{19}$$

Доказательство. Предположим, что лемма не верна, т. е. существует разбиение $G(k, s)$ на подграфы H_i, $1 \leqslant i \leqslant s$, такое, что $\tau^*(H_i) < k$ для $1 \leqslant i \leqslant s$. В этом случае, на основании леммы 5.1, для каждого i существует правильная разметка $\lambda_i : V(H_i) \to \{0, 1/2, 1\}$ такая, что

$$\sum_{v \in V(H_i)} \lambda_i(v) = \tau^*(H_i) < k. \tag{20}$$

Пусть $A = \{a_1 < \ldots < a_p \mid \lambda_i(a_j) \leqslant 1/2$ для всех i, $1 \leqslant i \leqslant s\}$, и пусть $\hat{\tau} = \sum_{i=1}^{s} \tau^*(H_i)$.

Имеется три случая.

1. $p \leqslant k$. В этом случае имеем

$$\hat{\tau} \geqslant k(s+1) - p \geqslant k(s+1) - k = ks,$$

что противоречит (20).

2. $k < p < 2k+1$. Для каждого ребра $\{a_j, a_{j+k}\}$, $1 \leqslant j \leqslant p-k$, должно существовать i такое, что $\lambda_i(a_j) + \lambda_i(a_{j+k}) \geqslant 1$. Следовательно,

$$\hat{\tau} \geqslant k(s+1) - p + (p-k) = ks,$$

что снова противоречит (20).

3. $p \geqslant 2k+1$. Вначале заметим, что для каждой вершины $v \in V(G(k, s))$ существует i такое, что $\lambda_i(v) \geqslant 1/2$. Действительно, предположим, что $\lambda_i(v) = 0$ для $1 \leqslant i \leqslant s$. Должно существовать a_j такое, что $|a_j - v| \geqslant k$. Но поскольку $\lambda_i(a_j) \leqslant 1/2$ для всех i, то $\lambda_i(a_j) + \lambda_i(v) \leqslant 1/2$ для всех i, что является противоречием.

Для каждого i обозначим через n_i число вершин v таких, что $\lambda_i(v) = 1$. Тогда имеем

$$|\{v \mid \lambda_i(v) > 0\}| \leqslant 2k - 1 - n_i,$$

поскольку в противном случае мы получили бы

$$\sum_{v \in V(H_i)} \lambda_i(v) \geqslant n_i \cdot 1 + (2k - 2n_i) \cdot \frac{1}{2} = k,$$

что противоречит (20). Следовательно,

$$\sum_{i=1}^{s} |\{v \mid \lambda_i(v) > 0\}| \leqslant (2k-1)s - \sum_{i=1}^{s} n_i. \tag{21}$$

Пусть q обозначает число вершин v таких, что существует только одно i, для которого $\lambda_i(v) > 0$. Тогда имеем

$$\sum_{i=1}^{s} |\{v \mid \lambda_i(v) > 0\}| \geqslant 2(k(s+1) - q) + q. \tag{22}$$

Объединяя (21) и (22), получим

$$q \geqslant 2k + s + \sum_{i=1}^{s} n_i. \tag{23}$$

Разумеется, мы можем без потери общности положить, что если $\lambda_i(v) = 1$, то $\lambda_j(v) = 0$ для всех $j \neq i$. Поэтому на основании определения n_i должно существовать по меньшей мере $2k+s$ вершин, скажем $b_1 < \ldots < b_{2k+s}$, таких, что

$$\sum_{i=1}^{s} \lambda_i(b_j) = \frac{1}{2},$$

т. е. для каждого b_j существует единственная разметка λ_i такая, что $\lambda_i(b_j)=1/2$ и $\lambda_l(b_j)=0$ для всех $l\neq i$. Поэтому, если $|b_j-b_l|_i\geqslant k$, то для некоторого i $\lambda_i(b_j)=\lambda_i(b_l)=1/2$. Поскольку $|b_1-b_{2k+s}|\geqslant k$, то существует i_0 такое, что $\lambda_{i_0}(b_1)=\lambda_{i_0}(b_{2k+s})=1/2$. Но по той же причине $\lambda_{i_0}(b_{k+j})=\lambda_{i_0}(b_1)=1/2$ и $\lambda_{i_0}(b_{2k+s})=\lambda_{i_0}(b_j)=1/2$ для $1\leqslant j\leqslant k+s$. Поэтому

$$\tau^*(H_{i_0})=\sum_{v\in V(H_{i_0})}\lambda_{i_0}(v)\geqslant(2k+s)\cdot\frac{1}{2}\geqslant k,$$

что является противоречием. Таким образом, доказательство леммы 5.2 завершено.

Напомним, что когда система заданий $\mathcal{S}$ выполняется с помощью фиксированного списка L, то s_i обозначает момент начала выполнения T_i. Как следует из определения списочного расписания, каждое s_i является суммой подмножества величин τ_j.

Без потери общности можно положить $\omega_0=1$. Предположим теперь, что $\omega>s+1$. Кроме того, предположим, что для некоторого k каждое τ_i может быть записано в виде $\tau_i=l_i/k$, где l_i — положительное целое число. Поскольку $\tau_i\leqslant\omega_0=1$, то $l_i\leqslant k$. Далее, все $r_i(t)$, $1\leqslant i\leqslant s$, постоянны на каждом интервале $[l/m,\ (l+1)/m]$ и их значения равны $r_i(l/m)$. Важно отметить, что поскольку $<$ пусто и $m\geqslant n$, то для $t_1,\ t_2\in[0,\ \omega)$ при $t_2-t_1\geqslant 1$ должно выполняться

$$\max_{1\leqslant i\leqslant s}\{r_i(t_1)+r_i(t_2)\}>1.$$

В противном случае любое задание, выполняемое в момент t_2, должно было бы выполняться в момент t_1 или ранее. Поэтому для каждого i, $1\leqslant i\leqslant s$, можно определить граф H_i следующим образом:

$$V(H_i)=\{0,\ 1,\ \ldots,\ (s+1)k-1\};$$

$\{a,\ b\}$ является ребром H_i тогда и только тогда, когда

$$r_i(a/k)+r_i(b/k)>1. \tag{24}$$

Заметим, что если $|a-b|\geqslant k$, то $\{a,\ b\}$ является ребром по крайней мере одного из H_i. Поэтому нетрудно видеть, что

$$G(k,\ s)\subseteq\bigcup_{1\leqslant i\leqslant s}H_i.$$

Из (24) следует, что разметка λ_i: $V(H_i)\to[0,\ \infty)$, задаваемая формулой $\lambda_i(a)=r_i(a/k)$, является правильной разметкой графа H_i. Поскольку $G\subseteq G'$ влечет $\tau^*(G)\leqslant\tau^*(G')$ и

условие (24) для r_i является строгим неравенством, то на основании леммы 5.2 получаем

$$\max_i \left\{ \sum_{l=0}^{(s+1)k-1} r_i\left(\frac{l}{k}\right) \right\} = \max_i \left\{ \sum_{v \in V(H_i)} \lambda_i(v) \right\} > \max_i \{\tau^*(H_i)\} \geqslant k.$$

С другой стороны, должно выполняться условие

$$\frac{1}{k} \sum_{l=0}^{(s+1)k-1} r_i\left(\frac{l}{k}\right) \leqslant \int_0^\infty r_i(t)\,dt \leqslant 1, \quad 1 \leqslant i \leqslant s,$$

откуда

$$\sum_{l=0}^{(s+1)k-1} r_i\left(\frac{l}{k}\right) \leqslant k, \quad 1 \leqslant i \leqslant s.$$

Поскольку это является противоречием, теорему 5.6 можно считать доказанной для случая, когда $\tau_i = l_i/k$ (при положительных целых k и l_i). Разумеется, из этого немедленно следует, что теорема 5.6 верна, когда все τ_i рациональны. Доказательство теоремы может быть завершено с помощью следующей леммы.

Лемма 5.3. *Пусть* $\tau = (\tau_1, \ldots, \tau_n)$ — *последовательность положительных действительных чисел. Тогда для любого* $\varepsilon > 0$ *существует последовательность* $\tau' = (\tau'_1, \ldots, \tau'_n)$ *такая, что:*

1) $|\tau'_i - \tau_i| < \varepsilon$ *для* $1 \leqslant i \leqslant n$;

2) *для всех* $A, B \subseteq \{1, \ldots, n\}$

$$\sum_{a \in A} \tau_a \leqslant \sum_{b \in B} \tau_b \Leftrightarrow \sum_{a \in A} \tau'_a \leqslant \sum_{b \in B} \tau'_b;$$

3) *все* τ_i *являются положительными рациональными числами.*

Замечание. Значение условия 2) заключается в том, что оно гарантирует одинаковый для τ_i и τ'_i порядок следования заданий T_i при использовании списка L. Поэтому, если при выполнении системы заданий $\mathscr{T}$ используется список L (один раз при временах выполнения τ_i, другой раз при временах τ'_i), то соответствующие значения ω и ω' удовлетворяют условию

$$|\omega - \omega'| \leqslant n\varepsilon.$$

Следовательно, если имеется система заданий $\mathscr{T}$, в которой $\omega/\omega_0 > s+1$ и при этом некоторые τ_i являются иррацио-

нальными, то можно построить другую систему $\mathscr{T}^*$, в которой иррациональные величины τ_i будут заменены рациональными τ_i', а соответствующие значения ω^* и $\omega^{*\prime}$ будут удовлетворять условиям

$$|\omega - \omega^*| \leqslant n\varepsilon, \quad |\omega' - \omega^{*\prime}| \leqslant n\varepsilon.$$

Таким образом, если ε достаточно мало, то по-прежнему $\omega^*/\omega^{*\prime} > s+1$. Однако это противоречило бы тому, что уже доказано.

Лемма 5.3 вытекает из следующего несколько более общего результата. Приведенное здесь доказательство получено Хваталом [20].

Лемма 5.3′. *Пусть $\mathscr{S}$ обозначает конечную систему неравенств вида*

$$\sum_{i=1}^{n} a_i x_i \geqslant a_0 \quad \text{или} \quad \sum_{i=1}^{n} a_i x_i > a_0,$$

где a_k — рациональные числа. Если $\mathscr{S}$ имеет действительное решение $(x_1, \ldots, x_n)$, то для любого $\varepsilon > 0$ $\mathscr{S}$ имеет рациональное решение $(x_1', \ldots, x_n')$, для которого $|x_i - x_i'| < \varepsilon$ для всех i.

Доказательство. Проведем доказательство методом индукции по n. Для $n = 1$ результат очевиден. Теперь допустим, что $\mathscr{S}$ является системой неравенств с $n > 1$ переменными, разрешимой в действительных числах. $\mathscr{S}$ естественно распадается на две части: $\mathscr{S}_1$ — подмножество неравенств, содержащих x_n, и $\mathscr{S}_0$ — оставшиеся неравенства системы $\mathscr{S}$. Каждое неравенство в $\mathscr{S}_1$ может быть записано в одной из следующих четырех форм:

а) $$\alpha_0 + \sum_{i=1}^{n-1} \alpha_i x_i \leqslant x_n;$$

б) $$\alpha_0 + \sum_{i=1}^{n-1} \alpha_i x_i < x_n;$$

в) $$\beta_0 + \sum_{i=1}^{n-1} \beta_i x_i \geqslant x_n;$$

г) $$\beta_0 + \sum_{i=1}^{n-1} \beta_i x_i > x_n.$$

Для каждой пары неравенств, одно из которых типа а), а другое — типа в), рассмотрим неравенство

д) $$\alpha_0 + \sum_{i=1}^{n-1} \alpha_i x_i \leqslant \beta_0 + \sum_{i=1}^{n-1} \beta_i x_i.$$

Аналогично, для сочетаний {а), г)}, {б), в)} и {б), г)} получим неравенства

$$\text{е)} \qquad \alpha_0 + \sum_{i=1}^{n-1} \alpha_i x_i < \beta_0 + \sum_{i=1}^{n-1} \beta_i x_i.$$

Пусть $\mathscr{S}^*$—множество всех неравенств типов д), е), которые получены из $\mathscr{S}_1$. Так как по предположению система $\mathscr{S} = \mathscr{S}_0 \cup \mathscr{S}_1$ имеет действительное решение $(x_1, \ldots, x_n)$, то и $\mathscr{S}_0 \cup \mathscr{S}^*$ имеет действительное решение $(x_1, \ldots, x_{n-1})$. Но $\mathscr{S}_0 \cup \mathscr{S}^*$ включает только $n-1$ переменных, так что на основании предположения индукции $\mathscr{S}_0 \cup \mathscr{S}^*$ имеет рациональное решение $(x_1', \ldots, x_{n-1}')$, для которого $|x_i - x_i'| < \varepsilon'$ для всех i и любого заданного $\varepsilon' > 0$. Подставляя x_i' в а), б), в) и г), получим систему неравенств

$$\text{ж)} \qquad a' \leqslant x_n, \quad b' < x_n, \quad c' \geqslant x_n, \quad d' > x_n,$$

где a', b', c', d' являются рациональными числами.

Поскольку x_i удовлетворяют неравенствам д) и е), получаем $a' \leqslant c'$, $b' < c'$, $a' < d'$, $b' < d'$. Таким образом, для любого $\varepsilon > 0$, если ε' выбрано достаточно малым, существует рациональное число x_n', удовлетворяющее ж), для которого $|x_n - x_n'| < \varepsilon$. Доказательство завершено. Это одновременно доказывает лемму 5.3 и теорему 5.6.

Следующий пример показывает, что верхняя оценка, полученная в предыдущей теореме, не может быть улучшена.

Пример 15. Пусть $\mathscr{T} = \{T_1, T_2, \ldots, T_{s+1}, T_1', T_2', \ldots, T_{sN}'\}$, $m \geqslant s(N+1)+1 = n$ и $<$ пусто. Положим

$$\tau_i = 1, \quad 1 \leqslant i \leqslant s+1, \quad \tau_1' = 1/N, \quad 1 \leqslant i \leqslant sN;$$

$$R_i(T_i) = 1 - 1/N, \quad R_i(T_j) = 1/sN, \quad j \neq i, \quad 1 \leqslant i \leqslant s;$$

$$R_i(T_j') = 1/N, \quad 1 \leqslant j \leqslant sN, \quad 1 \leqslant i \leqslant s;$$

$$L = (T_1, T_1', \ldots, T_N', T_2, T_{N+1}', \ldots, T_{2N}', T_3, \ldots, T_{k+1}, T_{k+1}', T_{kN+2}', \ldots, T_{(k+1)N}', T_{k+2}, \ldots, T_{sN}', T_{s+1});$$

$$L' = (T_1', T_2', \ldots, T_{sN}', T_1, T_2, \ldots, T_{s+1}).$$

Легко убедиться, что для этого примера

$$\omega = s+1, \quad \omega' = 1 + s/N.$$

Таким образом, ω/ω', а значит, и ω/ω_0 сколь угодно близки к $s+1$ при больших N.

Последний результат этого параграфа точно показывает, какое влияние оказывает ограничение количества процессоров на величину отношения ω/ω_0 (т. е. здесь не предполагается, что $m \geqslant n$).

Теорема 5.7 [50]. *В случае, когда* $<$ *пусто,* $m \geqslant 2$, s, L, τ — *произвольные, справедлива оценка*

$$\frac{\omega}{\omega_0} \leqslant \min\left\{\frac{m+1}{2},\ s+2-\frac{2s+1}{m}\right\}. \qquad (25)$$

Доказательство. Необходимо доказать две основных леммы, каждая из которых дает оценку сверху для ω/ω_0. Каждая из этих оценок является наилучшей при некоторых значениях s и m. Пусть X представляет собой объединение непересекающихся интервалов из $[0, \omega)$. Обозначим через $\mu(X)$ сумму длин этих интервалов.

Лемма 5.4. *Если* $<$ *пусто и* s, L, τ, m — *произвольные, то*

$$\omega/\omega_0 \leqslant (m+1)/2.$$

Доказательство. Пусть $I=\{t \mid |f(t)|=1\}$, где $f(t)$, напомним, определено как подмножество заданий T_i, которые находятся на обслуживании в момент t (рассматривается фиксированный список L). Вначале покажем, что

$$\mu(I) \leqslant \omega_0. \qquad (26)$$

Рассмотрим множество заданий T, определяемое как

$$T = \bigcup_{t \in I} f(t).$$

Для любой пары заданий T_i, T_j, принадлежащих T, должно существовать некоторое число k, $1 \leqslant k \leqslant s$, такое, что

$$R_k(T_i) + R_k(T_j) > 1,$$

так как в противном случае одно из этих заданий должно было бы начаться раньше (кроме случая $m=1$, для которого лемма очевидна). Но из этого следует, что в оптимальном расписании никакие два задания, принадлежащие T, не могут выполняться одновременно. Поэтому получаем

$$\omega_0 \geqslant \sum_{T_i \in T} \tau_i \geqslant \mu(I),$$

что доказывает (26).

Для того чтобы завершить доказательство леммы 5.4, заметим, что по крайней мере два процессора должны работать во все моменты времени $t \in \bar{I}=[0, \omega)-I$. Поэтому

$$m\omega_0 \geqslant \sum_{i=1}^{n} \tau_i \geqslant 2\mu(\bar{I}) + \mu(I) = 2\omega - \mu(I) \geqslant 2\omega - \omega_0,$$

и окончательно получаем $(m+1)\omega_0 \geqslant 2\omega$.

Лемма 5.5. *Если* $<$ *пусто,* $m \geqslant 3$ *и* s, L, τ — *произвольные, то*

$$\frac{\omega}{\omega_0} \leqslant s + 2 - \frac{2s+1}{m}. \tag{27}$$

Доказательство. Предположим, что существует пример, противоречащий лемме. На основании леммы 5.3 можно считать, что все τ_i рациональны, т. е. существует положительное целое число k такое, что для каждого i, $1 \leqslant i \leqslant n$, существует целое число l_i такое, что $\tau_i = l_i/k$. Без потери общности можно также положить $\omega_0 = 1$. Таким образом, $\omega > s+2-(2s+1)/m$ и каждое l_i удовлетворяет условию $1 \leqslant l_i \leqslant k$.

Рассмотрим работу системы, использующей список L. Как и раньше, положим $I = \{t \in [0, \omega) \mid |f(t)| = 1\}$, $I' = \{t \in [0, \omega) \mid |f(t)| = n\}$. Пусть также $\overline{I} = [0, \omega) - I$. Из доказательства леммы 5.4 следует, что $\mu(I) \leqslant 1$. Поскольку по крайней мере два процессора находятся в рабочем состоянии в каждый момент времени $t \in \overline{I}$, получим

$$m \geqslant \sum_{i=1}^{n} \tau_i \geqslant m \cdot \mu(I') + \mu(I) + 2(\omega - \mu(I) - \mu(I')) \geqslant$$

$$\geqslant (m-2)\mu(I') + 2\omega - 1,$$

откуда

$$\mu(I') \leqslant (m+1-2\omega)/(m-2). \tag{28}$$

Поскольку $\omega > s+2-(2s+1)/m$, получим

$$\mu(\overline{I}) = \omega - \mu(I') \geqslant \omega - \frac{m+1-2\omega}{m-2} >$$

$$> s+2-\frac{2s+1}{m} - \frac{m+1-2(s+2-(2s+1)/m)}{m-2} = s+1. \tag{29}$$

Теперь заметим, что для любых t_1, $t_2 \in \overline{I}$, удовлетворяющих условию $t_2 - t_1 \geqslant 1$, должно существовать i, $1 \leqslant i \leqslant s$, такое, что

$$r_i(t_1) + r_i(t_2) > 1. \tag{30}$$

В противном случае обслуживание некоторого задания, выполняемого в момент времени t_2, должно было бы начаться в момент t_1 или раньше. Напомним, что $\overline{I}$ представляет собой совокупность интервалов, каждый из которых имеет вид $[l/k, (l+1)/k]$ для некоторого целого числа l, и введем набор целых чисел $a_0 < a_1 < \ldots < a_p$ таких, что

$$\overline{I} = \{[a_i/k, (a_i+1)/k) : 0 \leqslant i \leqslant p\}.$$

Заметим, что неравенство (29) влечет неравенство $p \geqslant (s+1)k$. Для каждого i, $1 \leqslant i \leqslant s$, построим граф H_i, в котором множество вершин $V(H_i)$ определяется как

$$V(H_i) = \{0, 1, 2, \ldots, (s+1)k-1\},$$

а множество ребер определяется следующим образом: $\{u, v\}$ является ребром в H_i тогда и только тогда, когда

$$r_i(a_u/k) + r_i(a_v/k) > 1.$$

Заметим, что условие $|u-v| \geqslant k$ влечет условие $|a_u - a_v| \geqslant k$, из которого на основании (30) вытекает, что $\{u, v\}$ является ребром хотя бы одного из H_i, $1 \leqslant i \leqslant s$. Поэтому нетрудно убедиться, что $G(k, s) \subseteq \bigcup_i H_i$. С помощью тех же рассуждений, которые применялись в доказательстве теоремы 5.6, можно показать, что для некоторого i, $1 \leqslant i \leqslant s$,

$$\int_0^\omega r_i(t)\,dt > 1,$$

а это противоречит предположению о том, что $\omega_0 = 1$. Таким образом, лемма 5.5 доказана. Доказательство теоремы 5.7 вытекает из лемм 5.4 и 5.5.

Теперь рассмотрим примеры, показывающие, что оценка, определяемая теоремой 5.7, является наилучшей из возможных. Эти примеры несколько сложнее предыдущих. Проверку правильности приведенных выражений для ω и ω' мы оставляем читателю.

Пример 16. Рассмотрим три случая.

1) Пусть $2 \leqslant m \leqslant s+1$, где $\mathscr{T} = \{T_0, T_1, T_2, \ldots, T_{m-1}, T_1', T_2', \ldots, T_{m-1}'\}$, $\prec$ пусто, s произвольно и, кроме того,

$$\tau_0 = 1, \quad \tau_j = \tau_j' = 1/2, \quad 1 \leqslant j \leqslant m-1;$$

$$R_i(T_0) = 1/(2m), \qquad 1 \leqslant i \leqslant s;$$
$$R_i(T_i) = R_i(T_i') = 1/2, \qquad 1 \leqslant i \leqslant m-1;$$
$$R_i(T_j) = R_i(T_j') = 1/(2m), \quad i \neq j, \ 1 \leqslant i \leqslant s, \ 1 \leqslant j \leqslant m-1;$$
$$L = (T_1, T_1', T_2, T_2', \ldots, T_{m-1}, T_{m-1}', T_0);$$
$$L' = (T_0, T_1, T_2, \ldots, T_{m-1}, T_1', T_2', \ldots, T_{m-1}').$$

Тогда

$$\omega = (m+1)/2, \ \omega_0 = \omega' = 1.$$

2) Пусть $s+1 < m \leqslant 2s+1$. Для достаточно малого $\varepsilon > 0$ и произвольного положительного целого k определим

$\varepsilon_i = \varepsilon(m-1)^{i-2k}$, $1 \leqslant i \leqslant 2k$. Положим

$$\mathscr{T} = \{T_0\} \cup \{T_{ij} \mid 1 \leqslant i \leqslant m-1,\ 1 \leqslant j \leqslant k\} \cup \cup \{T'_{ij} \mid 1 \leqslant i \leqslant m-1,\ 1 \leqslant j \leqslant k\}.$$

Далее, $<$ пусто, s произвольно и, кроме того,

$$\tau_0 = 2k, \quad \tau_{ij} = \tau'_{ij} = 1, \quad 1 \leqslant i \leqslant m-1, \quad 1 \leqslant j \leqslant k;$$
$$R_i(T_0) = \varepsilon_1, \quad 1 \leqslant i \leqslant s;$$
$$R_i(T_{ij}) = 1-(m-1)\varepsilon_{2j-1}, \quad 1 \leqslant i \leqslant s, \quad 1 \leqslant j \leqslant k;$$
$$R_l(T_{ij}) = \varepsilon_{2j-1}, \quad l \neq i, \quad 1 \leqslant l \leqslant s, \quad 1 \leqslant i \leqslant m-1, \quad 1 \leqslant j \leqslant k;$$
$$R_i(T'_{s+l,\,j}) = 1-(m-1)\varepsilon_{2j}, \quad 1 \leqslant i \leqslant m-s-1, \quad 1 \leqslant j \leqslant k;$$
$$R_l(T'_{ij}) = \varepsilon_{2j}, \quad l \neq i-s, \quad 1 \leqslant l \leqslant s, \quad 1 \leqslant i \leqslant m-1, \quad 1 \leqslant j \leqslant k;$$
$$L = (A_1, A_2, \ldots, A_k, A'_1, A'_2, \ldots, A'_{k-1}, A_0),$$

где

$$A_i = (B_{1i}, B_{2i}, \ldots, B_{si}), \quad 1 \leqslant i \leqslant k;$$
$$B_{ji} = (T_{ji}, T'_{ji}), \quad 1 \leqslant i \leqslant k, \quad 1 \leqslant j \leqslant s;$$
$$A'_i = (B'_{1i}, B'_{2i}, \ldots, B'_{m-1,\,i}), \quad 1 \leqslant i \leqslant k-1;$$
$$B'_{ji} = (T'_{s+j,\,i}, T_{s+j,\,i+1}), \quad 1 \leqslant i \leqslant k-1, \quad 1 \leqslant j \leqslant m-1;$$
$$A_0 = (T_0, T_{s+1,\,1}, T_{s+2,\,1}, \ldots, T_{m-1,\,1}, T'_{s+1,\,k}, T'_{s+2,\,k}, \ldots, T'_{m-1,\,k});$$
$$L' = (C_0, C_1, C'_1, C_2, C'_2, \ldots, C_k, C'_k),$$
$$C_0 = (T_0);$$
$$C_i = (T_{1i}, T_{2i}, \ldots, T_{m-1,\,i}), \quad 1 \leqslant i \leqslant k;$$
$$C'_i = (T'_{1i}, T'_{2i}, \ldots, T'_{m-i,\,i}), \quad 1 \leqslant i \leqslant k.$$

Тогда

$$\omega = k(m+1)-(m-s-1), \quad \omega' = \omega_0 = 2k,$$

а значит,

$$\frac{\omega}{\omega_0} = \frac{m+1}{2} - \frac{m-s-1}{2k} \to \frac{m+1}{2} \quad \text{при} \quad k \to \infty.$$

3) Пусть $m > 2s+1$. Для достаточно малого $\varepsilon > 0$ и произвольного целого числа k' положим $k = k'm$ и определим $\varepsilon_i = \varepsilon(m-1)^{i-k}$, $1 \leqslant i \leqslant k$. Пусть

$$\mathscr{T} = \{T_0\} \cup \{T_{ij} \mid 1 \leqslant i \leqslant m-1, \quad 1 \leqslant j \leqslant k\},$$

$<$ пусто, s произвольно и, кроме того,

$$\tau_0 = k, \quad \tau_{ij} = 1, \quad 1 \leqslant i \leqslant m-1, \quad 1 \leqslant j \leqslant k;$$
$$R_i(T_0) = \varepsilon_1, \quad 1 \leqslant i \leqslant s;$$
$$R_i(T_{ij}) = 1-(m-1)\varepsilon_j, \quad 1 \leqslant i \leqslant s, \quad 1 \leqslant j \leqslant k;$$
$$R_l(T_{ij}) = \varepsilon_j, \quad l \neq i, \quad 1 \leqslant l \leqslant s, \quad 1 \leqslant i \leqslant m-1, \quad 1 \leqslant j \leqslant k;$$
$$L = (A_1, A_2, \ldots, A_{m-2s-1}, B_1, B_2, \ldots, B_s, C),$$

где

$$A_i = (T_{2s+i,1}, T_{2s+i,2}, \ldots, T_{2s+i,k}), \quad 1 \leqslant i \leqslant m-2s-1;$$
$$B_i = (T_{i1}, T_{s+i,2}, T_{i2}, T_{s+i,3}, \ldots, T_{i,k-1}, T_{s+1,k}), \quad 1 \leqslant i \leqslant s;$$
$$C = (T_0, T_{s+1,1}, T_{s+2,1}, \ldots, T_{2s,1}, T_{1k}, T_{2k}, \ldots, T_{sk});$$
$$L' = (T_0, D_1, D_2, \ldots, D_k),$$
$$D_l = (T_{1l}, T_{2l}, \ldots, T_{m-1,l}), \quad 1 \leqslant l \leqslant k.$$

Тогда получим

$$\omega = (s+2)k'm - (2s+1)k' - s, \quad \omega' = \omega_0 = k'm,$$

а значит,

$$\frac{\omega}{\omega_0} = s+2-\frac{2s+1}{m}-\frac{s}{k'm} \to s+2-\frac{2s+1}{m} \text{ при } k' \to \infty.$$

5.5. Задача об упаковке в контейнеры

В этом параграфе *) мы рассмотрим важный частный случай, когда $s=1$, $<$ пусто, все $\tau_i=1$ и $m \geqslant n$. Эта задача известна в литературе под названием задачи об упаковке в контейнеры из-за очевидной эквивалентности проблемы составления расписания для этого случая и задачи «упаковки» совокупности «весов» $(R_1(T_1), \ldots, R_1(T_n))$ в минимальное число «контейнеров» (т. е. единичных временных интервалов) единичной емкости (т. е. максимальный используемый ресурс равен 1) таким образом, чтобы содержимое ни одного контейнера не имело бы вес, превышающий 1.

Как и в гл. 1, отметим, что эта задача является двойственной для задачи минимизации количества процессоров, требуемых для выполнения заданий, имеющих общий крайний срок ω. Без потери общности можно предположить, что времена выполнения нормированы таким образом, что

*) Содержание и способ изложения материала основаны на [82].

$\omega=1$. В этом случае процессоры становятся «свободным» ресурсом, а ресурс «время выполнения» ограничен величиной $\omega=1$. В дальнейшем оказывается удобным принять эту точку зрения, согласно которой наша задача рассматривается как частный случай задачи из предыдущего параграфа.

Рассматриваемая задача эквивалентна также одномерной задаче раскроя (разновидности задачи о рюкзаке); поэтому эффективные алгоритмы определения оптимальных или близких к оптимальным вариантов упаковки (т. е. расписаний) имеют очевидное практическое применение, например, при составлении таблиц, распределении файлов, нарезании колец (формирование колец разной ширины из материала, имеющего одну стандартную ширину), решении задачи оптимизации длины кабеля и вообще таких задач, в которых несколько «кусков» различной «длины» должны быть образованы из кусков, имеющих стандартную «длину».

В данном параграфе мы несколько изменим обозначения для того, чтобы они соответствовали стандартным.

1. Вес $R_1(T_i)$ обозначен через a_i. Можно считать, что $0<a_i\leqslant 1$.

2. Последовательность весов обозначена через $L=(a_1, a_2, \ldots, a_n)$.

3. Минимальное количество контейнеров, в которые могут быть упакованы элементы множества L, обозначено через L^*.

4. Контейнер i, обозначенный через B_i, соответствует временному интервалу $[i-1, i)$.

Существует четыре алгоритма упаковки в контейнеры, которые в основном составят предмет нашего исследования. В каждом из этих алгоритмов вес a_1 загружается первым, а вес a_k загружается раньше, чем a_{k+1}, для всех $k\geqslant 1$.

1. Загрузка в первый подходящий контейнер (или FF: first-fit). Каждый вес a_k помещается в контейнер, имеющий минимальный индекс из числа тех, которые подходят для размещения a_k.

2. Загрузка в лучший из подходящих контейнеров (или BF: best-fit). Каждый вес a_k помещается в такой контейнер, который после помещения в него a_k будет иметь наименьший неиспользованный объем.

3. Загрузка в первый подходящий контейнер по убыванию весов (или FFD: first-fit-decreasing). Множество L представляется в виде списка, упорядоченного по невозрастанию весов, и к нему применяется правило FF.

4. Загрузка в лучший из подходящих контейнеров по убыванию весов (или BFD: best-fit-decreasing). То же, что и в п. 3, но с заменой FF на BF.

Количества контейнеров, требуемых в каждом из этих алгоритмов при работе со списком L, обозначаются через $FF(L)$, $BF(L)$, $FFD(L)$ и $BFD(L)$ соответственно.

Начнем с простого примера, иллюстрирующего виды списков, для которых FF и BF дают неудачные решения.

Пример 17. Пусть n делится на 18, и пусть δ удовлетворяет условию $0<\delta<1/84$. Определим список $L=(a_1, a_2, \ldots, a_n)$ следующим образом:

$$a_i = \begin{cases} 1/6-2\delta & \text{для } 1 \leqslant i \leqslant n/3, \\ 1/3+\delta & \text{для } n/3 < i \leqslant 2n/3, \\ 1/2+\delta & \text{для } 2n/3 < i \leqslant n. \end{cases}$$

Очевидно, $L^*=n/3$, так как элементы можно загрузить оптимально, помещая по одному элементу каждого типа в каждый контейнер. Однако, как легко проверить, оба алгоритма упаковки (FF и BF), примененные к списку L, приведут к такому варианту упаковки, когда имеется $n/18$ контейнеров, каждый из которых содержит по шесть элементов размером $1/6-2\delta$, $n/6$ контейнеров, каждый из которых содержит по два элемента размером $1/3+\delta$ и $n/3$ контейнеров, каждый из которых содержит единственный элемент размером $1/2+\delta$. Два варианта упаковки показаны на рис. 5.38. Таким образом, получаем

$$\frac{FF(L)}{L^*} = \frac{BF(L)}{L^*} = \frac{n/18+n/6+n/3}{n/3} = \frac{5}{3}.$$

Несколько модифицировав список L, описанный в примере 16, можно получить еще худший результат.

Теорема 5.8. *Для любого $k \geqslant 1$ существует список L, для которого $L^*=k$ и*

$$FF(L) = BF(L) > \frac{17}{10} L^* - 8.$$

Доказательство. Как и в предыдущем примере, элементы L принадлежат трем группам, и их размеры приблизительно равны 1/6, 1/3 и 1/2. Количество элементов во всех группах одинаково. В списке L элементы первой группы предшествуют элементам второй группы, которые в свою очередь предшествуют элементам третьей группы.

Пусть n—положительное число, делящееся на 17, и пусть δ выбрано так, что $0 < \delta < 18^{-n/17}$. Первая группа разбита на $n/17$ блоков по 10 чисел в каждом. Обозна-

чим 10 чисел в i-м блоке первой группы через $a_{0,i}, a_{1,i}, \ldots, a_{9,i}$. Эти числа определены следующим образом:

$$a_{0i} = 1/6 + 33\delta_i,$$
$$a_{1i} = 1/6 - 3\delta_i,$$
$$a_{2i} = a_{3i} = 1/6 - 7\delta_i,$$
$$a_{4i} = 1/6 - 13\delta_i,$$
$$a_{5i} = 1/6 + 9\delta_i,$$
$$a_{6i} = a_{7i} = a_{8i} = a_{9i} = 1/6 - 2\delta_i,$$
$$\delta_i = \delta \cdot 18^{(n/17)-i} \quad \text{для} \quad 1 \leqslant i \leqslant n/17.$$

Предположим, что первые $10n/17$ элементов списка L суть $(a_{01}, a_{11}, \ldots, a_{91}, a_{02}, a_{12}, \ldots, a_{92}, \ldots)$. Заметим, что

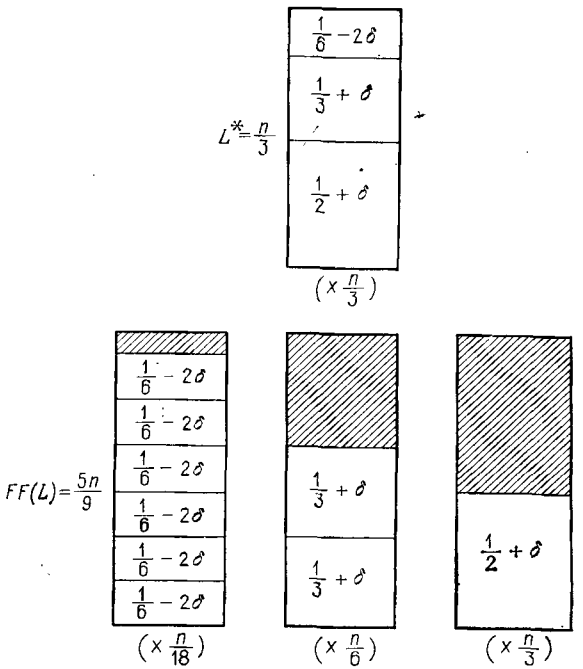


Рис. 5.38.

$a_{0i} + a_{1i} + \ldots + a_{4i} = 5/6 + 3\delta_i$ и $a_{5i} + a_{6i} + \ldots + a_{9i} = 5/6 + \delta_i$. Следовательно, для всех i при использовании алгоритма FF либо BF первые пять элементов i-го блока займут $(2i-1)$-й контейнер, а последние пять элементов i-го блока займут $(2i)$-й контейнер. Для того чтобы убедиться в этом, нам необходимо только заметить, что a_{4i} (наименьший элемент

в блоке i) не будет загружен ни в один из предыдущих контейнеров из-за того, что даже наименее заполненный из них $(2i-2)$-й содержит в сумме $5/6+\delta_{i-1}=5/6+18\delta_i$. Кроме того, наименьший элемент из набора $a_{5i}, a_{6i}, \ldots, a_{9i}$ имеет величину $1/6-2\delta_i$ и не будет загружен в $(2i-1)$-й контейнер, наполнение которого составляет $5/6+3\delta_i$. Следовательно, $n/17$ блоков в первой группе должны занять $2n/17$ контейнеров.

Обратимся теперь ко второй группе. Элементы этой группы равны примерно $1/3$, и они снова разбиваются на $n/17$ блоков, по 10 элементов в каждом. Элементы этих блоков обозначим через $b_{0i}, b_{1i}, \ldots, b_{9i}$. В списке L все эти элементы идут вслед за a_{ji} и образуют последовательность $(b_{01}, b_{11}, \ldots, b_{91}, b_{02}, b_{12}, \ldots, b_{92}, \ldots)$. Значения b_{ji} определены следующим образом:

$$
\begin{aligned}
b_{0i} &= 1/3+46\delta_i,\\
b_{1i} &= 1/3-34\delta_i,\\
b_{2i} &= b_{3i} = 1/3+6\delta_i,\\
b_{4i} &= 1/3+12\delta_i,\\
b_{5i} &= 1/3-10\delta_i,\\
b_{6i} &= b_{7i} = b_{8i} = b_{9i} = 1/3+\delta_i.
\end{aligned}
$$

Элементы i-го блока занимают контейнеры от $(2n/17)+5i-4$ до $(2n/17)+5i$, причем b_{0i} упаковано вместе с b_{1i}, b_{2i} — вместе с b_{3i} и так далее. Чтобы в этом убедиться, рассмотрим наполнение пяти контейнеров, занятых i-м блоком; оно выглядит следующим образом: $2/3+12\delta_i$, $2/3+12\delta_i$, $2/3+2\delta_i$, $2/3+2\delta_i$ и $2/3+2\delta_i$.

Следовательно, вес $b_{5i}=1/3-10\delta_i$ не может быть помещен ни в один из первых двух контейнеров, а вес $b_{1i}=1/3-34\delta_i$ не может быть помещен ни в один из контейнеров, содержащих предыдущие блоки, поскольку все они наполнены по крайней мере до уровня $2/3+2\delta_{i-1}=2/3+36\delta_i$. Следовательно, $n/17$ блоков второй группы занимают $5n/17$ контейнеров.

Третья группа состоит из $10n/17$ элементов, каждый из которых имеет вес $1/2+\delta$. Очевидно, каждый из них занимает один контейнер. Таким образом, список L исчерпан. Общее число контейнеров, потребовавшихся при загрузке алгоритмами FF или BF, в точности равно n.

Однако элементы списка L могут быть упакованы в $(10n/17)+1$ контейнеров следующим образом. Все контейнеры, кроме двух, содержат по одному весу $1/2+\delta$. Остав-

шееся пространство в каждом из таких контейнеров заполнено одной из трех следующих комбинаций:

1) $a_{ji}+b_{ji}$ для некоторых i, j таких, что $2\leqslant j\leqslant 9$, $1\leqslant i\leqslant n/17$;

2) $a_{0i}+b_{1i}$ для некоторых, i, $1\leqslant i\leqslant n/17$;

3) $a_{1i}+b_{0,\,i+1}$ для некоторых i, $1\leqslant i\leqslant n/17$.

Остаются b_{01}, $a_{1,\,(n/17)}$ и один вес $1/2+\delta$, которые легко могут быть загружены в два оставшихся контейнера. Таким образом, мы показали, что $L^*\leqslant 1+10n/17$, поэтому

$$\frac{FF(L)}{L^*}\geqslant\frac{17n}{10n+17}>\frac{17}{10}-\frac{2}{L^*},$$

и аналогично

$$\frac{BF(L)}{L^*}>\frac{17}{10}-\frac{2}{L^*}.$$

Чтобы получить значения L^*, не сравнимые с 1 по модулю 10, мы можем сформировать список L' путем присоединения к L k элементов, каждый из которых имеет размер 1, где k — фиксированное число, не большее 9. В этом случае предыдущие рассуждения приводят к выражениям $FF(L')=FF(L)+k$ и $L'^*=L^*+k_1$, поэтому

$$\frac{FF(L')}{L'^*}\geqslant\frac{17}{10}-\frac{8}{L'^*},$$

причем такая оценка имеет место и для $BF(L')/L'^*$. Это доказывает теорему 5.8.

Теперь покажем, что примеры, построенные в предыдущем доказательстве, соответствуют наихудшему случаю и что число 17/10 является асимптотической нижней оценкой для отношений $FF(L)/L^*$ и $BF(L)/L^*$ при больших L.

Теорема 5.9 [144, 50]. *Для всех списков L справедливы следующие оценки:*

$$FF(L)\leqslant\frac{17}{10}L^*+2,$$

$$BF(L)\leqslant\frac{17}{10}L^*+2.$$

Доказательство. Используем только два следующих свойства алгоритмов FF и BF.

1. Ни один элемент не загружается в пустой контейнер до тех пор, пока не будет установлено, что этот элемент не может быть загружен ни в один из частично заполненных контейнеров.

2. Если имеется единственный непустой контейнер с наименьшим уровнем, то ни один элемент не может быть в него помещен, пока не будет установлено, что этот элемент

не может быть помещен ни в один из контейнеров с меньшим номером. (Под *уровнем* контейнера понимается просто суммарный содержащийся в нем вес.)

Определим отображение W: $[0, 1] \to [0, 1]$ следующим образом (см. рис. 5.39):

$$W(\alpha) = \begin{cases} 6\alpha/5 & \text{для } 0 \leqslant \alpha \leqslant 1/6, \\ 9\alpha/5 - 1/10 & \text{для } 1/6 < \alpha \leqslant 1/3, \\ 6\alpha/5 + 1/10 & \text{для } 1/3 < \alpha \leqslant 1/2, \\ 1 & \text{для } 1/2 < \alpha \leqslant 1. \end{cases}$$

Утверждение 1. *Пусть некоторый контейнер содержит элементы* $b_1, b_2, \ldots, b_k$; *тогда*

$$\sum_{i=1}^{k} W(b_i) \leqslant \frac{17}{10}.$$

Доказательство. Если $b \leqslant 1/2$, то $W(b)/b \leqslant 3/2$. Экстремум отношения достигается только при $b = 1/3$, а в остальных случаях оно имеет меньшую величину. Таким образом, утверждение очевидно, если в наборе нет элемента b_i, превышающего 1/2. Если же такой элемент есть, можно считать, что это b_1, и тогда нужно показать, что если

$$\sum_{i=2}^{k} b_i < 1/2,$$

то

$$\sum_{i=2}^{k} W(b_i) \leqslant \frac{7}{10}.$$

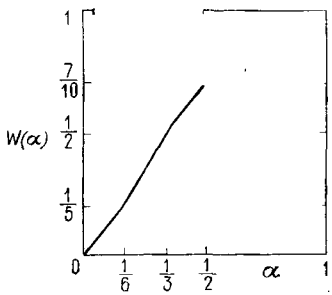


Рис. 5.39.

Следует отметить, что поскольку наклон графика $W(b)$ одинаков в интервалах $[0, 1/6]$ и $[1/3, 1/2]$, то любой элемент b_i из второго интервала может быть без потери общности заменен двумя элементами $1/3$ и $b_i - 1/3$. Поэтому предположим, что $b_i \leqslant 1/3$ для $2 \leqslant i \leqslant k$. Кроме того, если b_j и b_j, оба меньше или равны 1/6, то они могут быть объединены в один элемент, даже при этом $\sum_i W(b_i)$ не уменьшится (но может в действительности увеличиться). Следовательно, можно предположить, что в интервале $(0, 1/6]$ находится, самое большее, один из элементов b_i, $i \geqslant 2$, а остальные находятся в интервале $(1/6, 1/3]$.

Это предположение дает возможность свести доказательство к рассмотрению четырех случаев:

1) $k=2$, $b_2 \leqslant 1/3$;
2) $k=3$, $1/6 < b_2 \leqslant b_3 \leqslant 1/3$;
3) $k=3$, $b_2 \leqslant 1/6 < b_3 \leqslant 1/3$;
4) $k=4$, $b_2 \leqslant 1/6 < b_3 \leqslant b_4 \leqslant 1/3$.

В случае 1) немедленно получаем $W(b_2) \leqslant 7/10$, так как $b_2 \leqslant 1/2$. В случае 2) $W(b_2)+W(b_3)=(9/5)(b_2+b_3)-1/5 \leqslant \leqslant 9/5 \cdot 1/2 - 1/5 = 7/10$, поскольку $b_2+b_3 \leqslant 1/2$. В случае 3) $W(b_2)+W(b_3)=(6/5)b_2+(9/5)b_3-1/10 \leqslant 1/5+3/5-1/10 = =7/10$. И наконец, в случае 4) $W(b_2)+W(b_3)+W(b_4) \leqslant \leqslant (6/5)b_2+(9/5)(b_3+b_4)-1/5=(9/5)(b_2+b_3+b_4)-(3/5)b_2 - -1/5 \leqslant 9/10-1/5=7/10$, поскольку $b_2+b_3+b_4 \leqslant 1/2$.

Введем понятие *крупности* контейнера, означающее наибольшее число α такое, что некоторый контейнер с меньшим номером наполняется до уровня $1-\alpha$. Крупность первого контейнера равна 0.

Утверждение 2. *Предположим, что контейнеры заполнены в соответствии с алгоритмом* FF *или* BF *и некоторый контейнер B имеет крупность* α. *Тогда любой элемент, находящийся в B, который был помещен в него перед тем, как B оказался заполненным больше чем наполовину, превышает* α.

Доказательство. До тех пор, пока контейнер не оказался заполненным до уровня, превышающего 1/2, он либо пуст, либо является единственным непустым контейнером самого низкого уровня (в соответствии со свойством 1 алгоритма загрузки). При этом согласно ограничениям 1 и 2 любой элемент, помещенный в контейнер, должен оказаться неподходящим для любого другого контейнера с меньшим номером, следовательно, должен превышать α.

Утверждение 3. *Пусть контейнер, имеющий крупность* $\alpha < 1/2$, *заполнен элементами* $b_1 \geqslant b_2 \geqslant \ldots \geqslant b_k$ *посредством* FF-*упаковки* (BF-*упаковки*). *Если* $\sum_{i=1}^{k} b_i \geqslant 1-\alpha$, *то* $\sum_{i=1}^{k} W(b_i) \geqslant 1$.

Доказательство. Если $b_1 > 1/2$, то справедливость утверждения следует немедленно, так как $W(b_1)=1$. Поэтому предположим, что $b_1 \leqslant 1/2$. Если $k \geqslant 2$, то второй элемент, помещенный в контейнер, был помещен до того, как контейнер оказался заполненным больше чем наполовину. Следовательно, согласно утверждению 2, по меньшей мере два элемента превышают α. В частности, должно вы-

полняться $b_1 \geqslant b_2 > \alpha$. Рассмотрим несколько случаев в зависимости от диапазона значений α.

Случай 1. $\alpha \leqslant 1/6$. Тогда $\sum_{i=1}^{k} b_i \geqslant 1-\alpha \geqslant 5/6$. Поскольку $W(\beta)/\beta \geqslant 6/5$ в диапазоне $0 \leqslant \beta \leqslant 1/2$, немедленно получаем

$$\sum_{i=1}^{k} W(b_i) \geqslant \frac{6}{5} \cdot \frac{5}{6} = 1.$$

Случай 2. $1/6 \leqslant \alpha \leqslant 1/3$. Рассмотрим варианты в зависимости от значений k.

1) $k=1$. Здесь должно быть $1-\alpha \leqslant 1/2$ или $\alpha \geqslant 1/2$, поскольку $b_1 \leqslant 1/2$, но это противоречит предположению, что $\alpha \leqslant 1/3$.

2) $k=2$. Если как b_1, так и b_2 оба больше или равны 1/3, то $W(b_1)+W(b_2) \geqslant (6/5 \cdot 1/3 + 1/10) \cdot 2 = 1$. Если оба меньше 1/3, то $b_1+b_2 < 2/3 < 1-\alpha$, что противоречит нашей гипотезе. Если $b_1 \geqslant 1/3$ и $b_2 < 1/3$, то, поскольку b_1 и b_2 должны быть больше, чем α, получаем $\alpha < b_2 < 1/3 \leqslant b_1 \leqslant 1/2$. Следовательно, $W(b_1)+W(b_2) = (9/5)b_1 - 1/10 + (6/5)b_2 + 1/10 = (6/5)(b_1+b_2) + (3/5)b_1$. Поскольку $b_1+b_2 \geqslant 1-\alpha$ и $b_1 > \alpha$, получаем $W(b_1)+W(b_2) \geqslant (6/5)(1-\alpha) + (3/5)\alpha = 1 + (1/5 - (3/5)\alpha) \geqslant 1$, так как $\alpha \leqslant 1/3$.

3) $k \geqslant 3$. Как и в предыдущем случае, если два элемента b_i больше или равны 1/3, результат следует немедленно. Если $b_1 \geqslant 1/3 > b_2 \geqslant \alpha$, то

$$W(b_1) + W(b_2) + \sum_{i=3}^{k} W(b_i) \geqslant$$

$$\geqslant \frac{6}{5} b_1 + \frac{1}{10} + \frac{9}{5} b_2 - \frac{1}{10} + \frac{6}{5} \sum_{i=3}^{k} b_i =$$

$$= \frac{6}{5} \sum_{i=1}^{k} b_i + \frac{3}{5} b_2 \geqslant \frac{6}{5}(1-\alpha) + \frac{3}{5}\alpha = 1 + \frac{1}{5} - \frac{3}{5}\alpha \geqslant 1.$$

Если $1/3 > b_1 \geqslant b_2 > \alpha$, то

$$W(b_1) + W(b_2) + \sum_{i=3}^{k} W(b_i) \geqslant \left(\frac{9}{5}\right)(b_1+b_2) - \frac{1}{5} + \frac{6}{5} \sum_{i=3}^{k} b_i \geqslant$$

$$\geqslant \left(\frac{6}{5}\right)(1-\alpha) + \left(\frac{3}{5}\right)(2\alpha) - \frac{1}{5} = 1 + \frac{6}{5}\alpha - \frac{6}{5}\alpha = 1.$$

Случай 3. $1/3<\alpha<1/2$. При $k=1$ имеем $b_1\geqslant 1-\alpha>1/2$, так что $W(b_1)=1$. Если $k\geqslant 2$, то $b_1\geqslant b_2>1/3$, и результат следует немедленно.

Утверждение 4. *Если контейнер, имеющий крупность* $\alpha<1/2$, *заполнен элементами* $b_1\geqslant\ldots\geqslant b_k$ *и* $\sum_{i=1}^{k} W(b_i)=1-\beta$, *где* $\beta>0$, *то имеет место один из двух случаев*:

1) $k=1$ *и* $b_i\leqslant 1/2$;

2) $\sum_{i=1}^{k} b_i\leqslant 1-\alpha-\frac{5}{9}\beta$.

Доказательство. Если $k=1$ и $b_1>1/2$, то не может быть $\beta>0$. Поэтому, если условие 1) не выполняется, мы можем считать, что $k\geqslant 2$; отсюда так же, как и при доказательстве предыдущего утверждения, получаем $b_1\geqslant b_2\geqslant\alpha$. Пусть $\sum_{i=1}^{k} b_i=1-\alpha-\gamma$. Тогда можно построить контейнер, заполненный элементами $b_3, b_4, \ldots, b_k$ и двумя другими элементами δ_1 и δ_2, выбранными таким образом, что $\delta_1+\delta_2=b_1+b_2+\gamma$, $\delta_1\geqslant b_1$, $\delta_2\geqslant b_2$ и при этом ни δ_1, ни δ_2 не превышает $1/2$. Из доказательства утверждения 3 и того факта, что δ_1 и δ_2 превышают α, следует, что

$$\sum_{i=3}^{k} W(b_i)+W(\delta_1)+W(\delta_2)\geqslant 1.$$

Но поскольку наклон W в диапазоне $[0, 1/2]$ не превышает $9/5$, получаем $W(\delta_1)+W(\delta_2)\leqslant W(b_1)+W(b_2)+(9/5)\gamma$. Поэтому $\gamma\geqslant(5/9)\delta$, и условие 2) выполняется.

Теперь мы готовы к тому, чтобы завершить доказательство теоремы 5.9. Пусть $L=(a_1, a_2, \ldots, a_n)$ и $\overline{W}=\sum_{i=1}^{n} W(a_i)$. Согласно утверждению 1 $(17/10)L^*\geqslant\overline{W}$.

Предположим, что в алгоритме FF (BF) контейнеры $B_1', B_2', \ldots, B_k'$ таковы, что в каждом из них находится хотя бы один элемент и для каждого из них $\sum_{j} W(a_j)=1-\beta_i$, $\beta_i>0$, где j пробегает все элементы в контейнере B_i'. Мы полагаем, что из условия $1\leqslant i<j\leqslant k$ вытекает, что в первоначальной нумерации контейнеров B_i' имеет меньший номер, чем B_j'. Пусть γ_i обозначает крупность B_i'. Поскольку B_i' не содержит ни одного элемента, превышающего $1/2$, для всех γ_i должно выполняться $\gamma_i<1/2$. На основании утверждения 4 и определения крупности

получим

$$\gamma_i > \gamma_{i-1} + \frac{5}{9}\beta_{i-1}, \quad 1 < i \leqslant k.$$

Поэтому

$$\sum_{i=1}^{k-1} \beta_i \leqslant \frac{9}{5} \sum_{i=2}^{k} (\gamma_i - \gamma_{i-1}) = \frac{9}{5}(\gamma_k - \gamma_1) \leqslant \frac{9}{5} \cdot \frac{1}{2} < 1.$$

Поскольку β_k не может превышать 1, то

$$\sum_{i=1}^{k} \beta_i \leqslant 2.$$

Применяя утверждение 3, получим

$$FF(L) \leqslant \overline{W} + 2 \leqslant 1.7L^* + 2,$$
$$BF(L) \leqslant \overline{W} + 2 \leqslant 1.7L^* + 2,$$

что завершает доказательство теоремы.

Если список $L = (a_1, \ldots, a_n)$ таков, что для некоторого $\alpha \leqslant 1/2$ все a_i меньше или равны α, то наихудший случай для обоих алгоритмов загрузки не является столь экстремальным. В частности, имеет место следующий результат.

Теорема 5.10 [52]. *Для любого положительного числа $\alpha \leqslant 1/2$ положим $k = [\alpha^{-1}]$. Тогда справедливы следующие утверждения.*

1. *Для каждого $l \geqslant 1$ существует список $L = (a_1, \ldots, a_n)$, в котором все $a_i \in (0, \alpha]$ и $L^* = l$, такой, что*

$$FF(L) \geqslant \left(\frac{k+1}{k}\right) L^* - \frac{1}{k}.$$

2. *Для любого списка $L = (a_1, \ldots, a_n)$, в котором все $a_i \in (0, \alpha]$, имеем*

$$FF(L) \leqslant \left(\frac{k+1}{k}\right) L^* + 2.$$

Утверждения 1 и 2 выполняются также для алгоритма BF.

Доказательство. Вначале опишем способ построения списка L, в котором ни один элемент не превышает α и при этом

$$\frac{FF(L)}{L^*} = \frac{BF(L)}{L^*} = \frac{k+1}{k} - \frac{1}{kL^*}.$$

Пусть l — произвольное положительное число. Список L составляется из элементов, которые все очень близки к $1/(k+1)$. Участвуют два типа элементов, определяемые

следующим образом:

$$b_j = \frac{1}{k+1} - k^{2j+1}\delta, \quad j = 1, 2, \ldots, i-1,$$

$$a_{1j} = a_{2j} = \ldots = a_{kj} = \frac{1}{k+1} + k^{2j}\delta, \quad j = 1, 2, \ldots, l,$$

где $\delta > 0$ — достаточно малое число. В списке L элементы типа a располагаются в порядке невозрастания, а элементы типа b располагаются в порядке строгого возрастания и распределены так, что между каждой парой b_j и b_{j-1} последовательных элементов типа b оказывается ровно k элементов типа a. Список L полностью характеризуется тем свойством, что элемент b_{l-1} оказывается во второй позиции. Мы предоставляем читателю убедиться, что

$$FF(L) = BF(L) = \left\lfloor \frac{l(k+1)-1}{k} \right\rfloor.$$

Легко видеть, что элементы L могут быть оптимально упакованы путем загрузки b_j, a_{1j}, a_{2j}, ..., a_{kj} в один контейнер для каждого $j = 1, \ldots, l-1$ и загрузки a_{1l}, a_{2l}, ..., a_{kl} в один дополнительный контейнер. Это дает $L^* = l$. Тогда получим

$$\frac{FF(L)}{L^*} = \frac{BF(L)}{L^*} \geqslant \frac{l(k+1)-1}{kl} = \frac{k+1}{k} - \frac{1}{kL^*}.$$

Так же легко может быть получена верхняя оценка. Предположим, что список L не содержит элементов, превышающих $1/k$, где k — целое число.

Рассмотрим процесс упаковки L с помощью алгоритма FF. Каждый контейнер, кроме, быть может, последнего, содержит по меньшей мере k элементов. Предположим, что каждый из контейнеров B_i и B_j $(i < j < FF(L))$ содержит элементы, составляющие в сумме меньше чем $k/(k+1)$. Тогда, поскольку B_j содержит k элементов, B_j должен содержать элемент, размер которого меньше чем $1/(k+1)$. Но этот элемент должен был бы поместиться в B_i и поэтому не может быть помещен в B_j алгоритмом FF; получаем противоречие. Таким образом, все контейнеры, за исключением, быть может, двух, должны быть заполнены не менее чем до уровня $k/(k+1)$. Следовательно, обозначив сумму всех элементов в L через $w(L)$, получим

$$L^* \geqslant w(L) \geqslant \frac{k}{k+1}(FF(L) - 2),$$

и, окончательно,

$$FF(L) \leqslant \frac{(k+1)\,L^*}{k} + 2.$$

Аналогичный, но несколько более сложный способ доказательства может быть применен для алгоритма BF. Таким образом, теорема доказана.

Из теоремы 5.10 немедленно следует, что если $\alpha \leqslant 1/2$ и $L=(a_1,\ldots,a_n)$, причем все $a_i \in (0,\ \alpha]$, то для всех $\varepsilon>0$

$$\frac{FF\ (L)}{L^*} \leqslant 1+[\alpha^{-1}]^{-1}+\varepsilon$$

при условии, что L^* достаточно велико. Такая же оценка справедлива для BF.

Наиболее глубокие из известных результатов, относящихся к задаче упаковки в контейнеры, получены для алгоритмов загрузки FFD и BFD. Для того чтобы описать эти результаты, положим

$$R_{\mathrm{FFD}}(\alpha) = \overline{\lim_{L^*\to\infty}}\ \mathrm{FFD}\,(L)/L^*,$$

где L пробегает все списки, в которых все элементы a_i удовлетворяют условию $a_i \in (0,\ \alpha]$.

Например, из предыдущих результатов вытекает, что

$$R_{\mathrm{FF}}(\alpha) = \begin{cases} 17/10 & \text{при } \alpha \in (1/2,\ 1], \\ 1+\lfloor \alpha^{-1} \rfloor^{-1} & \text{при } \alpha \in (0,\ 1/2]. \end{cases}$$

Теорема 5.11 [77]. *Справедливо равенство*

$$R_{\mathrm{FFD}}(\alpha) = \begin{cases} 11/9 & \textit{при } \alpha \in (1/2,\ 1], \\ 71/60 & \textit{при } \alpha \in (8/29,\ 1/2], \\ 7/6 & \textit{при } \alpha \in (1/4,\ 8/29], \\ 23/20 & \textit{при } \alpha \in (1/5,\ 1/4]. \end{cases}$$

Единственное известное доказательство теоремы 5.11, принадлежащее Д. Джонсону [77, 82], чрезвычайно остроумно и довольно сложно; оно занимает более 100 страниц. Нет нужды говорить, что ограниченный объем книги не позволяет привести его здесь. Наиболее трудная часть доказательства, а именно установление верхней оценки для $R_{\mathrm{FFD}}(\alpha)$, базируется на такой же стратегии, которая была использована для получения верхних оценок для FF и BF. Она заключается в том, что вводится весовая функция, которая ставит в соответствие элементам списка L действительные числа или веса, зависящие от их размеров, таким образом, чтобы соблюдались следующие условия.

1. Общий вес всех элементов, содержащихся в списке L, отличается от общего числа контейнеров, используемых в конкретном алгоритме загрузки (например, FF или FFD), не более чем на некоторую абсолютную постоянную c.

2. Общий вес любого правильно упакованного контейнера должен быть меньше некоторой фиксированной постоянной c'.

Для FF мы имели $c'=17/10$ и $c=2$; для FFD можно выбрать $c'=11/9$ и $c=4$.

Как и для случая FF и BF, оценки, установленные в теореме 5.11 для FFD, справедливы также и для BFD. Однако между алгоритмом FFD и алгоритмом BFD существует некоторая асимметрия, как показывает следующий результат (который также используется для доказательства равенства $R_{\mathrm{BFD}}(1)=11/9$).

Теорема 5.12 [52, 82]. *Для всех списков* $L=(a_1,\ldots,a_n)$ *имеют место соотношения*

1) $BFD(L)\leqslant FFD(L)$, *если все* $a_i\in[1/6,\ 1]$;

2) $BFD(L)=FFD(L)$, *если все* $a_i\in[1/5,\ 1]$.

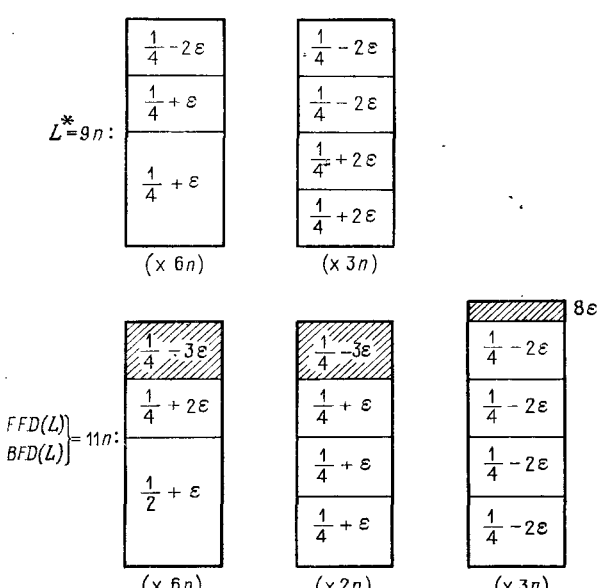
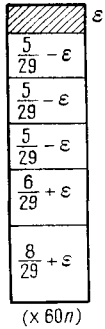


Рис. 5.40.

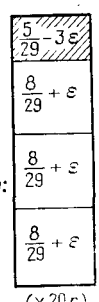
Даже для этой теоремы доказательство занимает около 15 страниц и здесь не приводится.

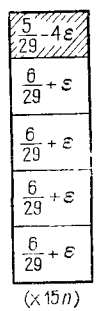
Возможно, что полная формулировка теоремы 5.11 заключается в следующем [78].



Оптимальная упаковка $L^* = 60$

Упаковка посредством алгоритма *FFD*. $FFD(L) = 71n$:





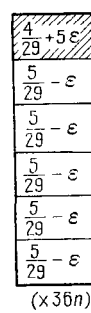
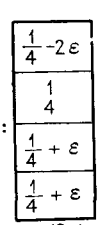
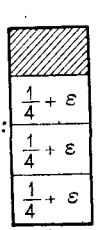


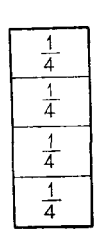
Рис. 5.41.

$L^* = 12n$:



$FFD(L) = 14n$:





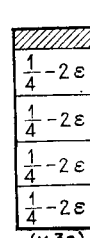


Рис. 5.42.

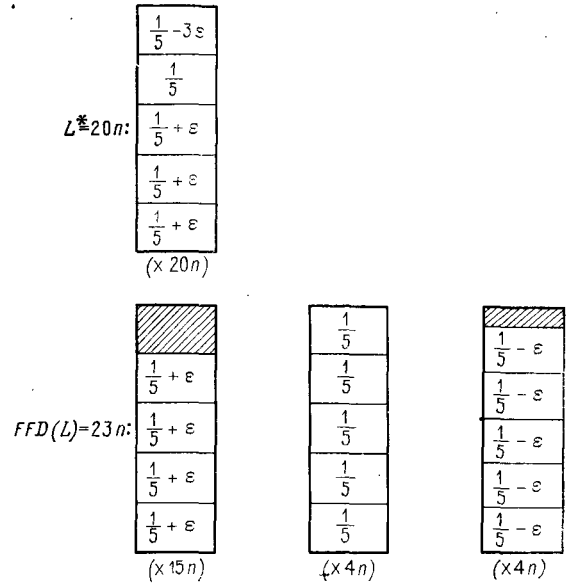


Рис. 5.43.

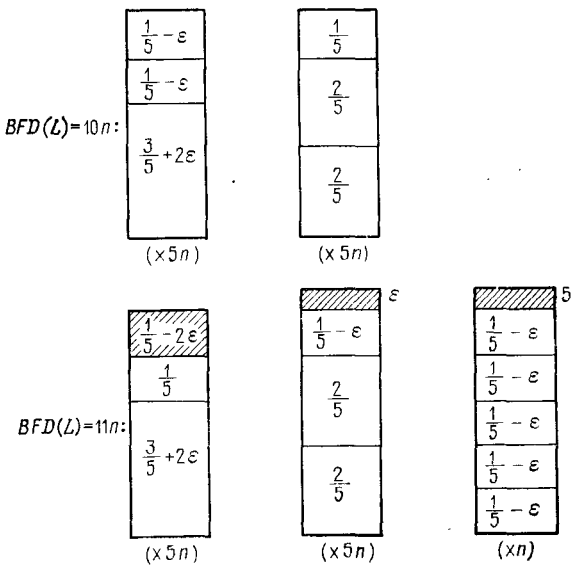


Рис. 5.44.

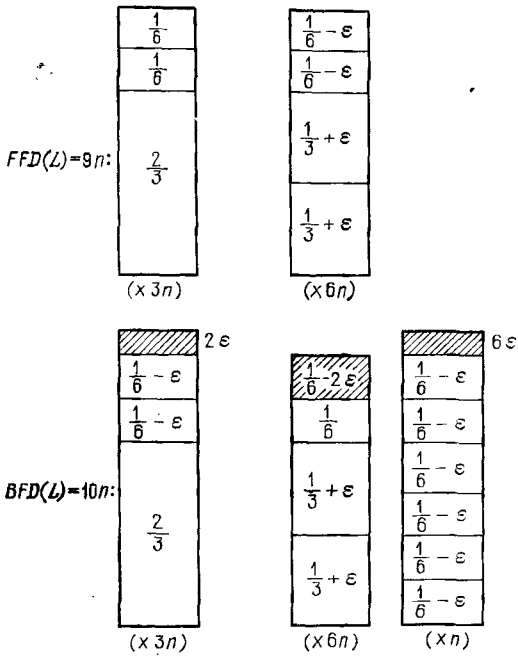


Рис. 5.45.

Гипотеза. *Справедливо равенство*

$$R_{\mathrm{FFD}}(\alpha) = \begin{cases} 11/9 & \text{при } \alpha \in (1/2,\ 1], \\ 71/60 & \text{при } \alpha \in (8/29,\ 1/2], \\ 7/6 & \text{при } \alpha \in (1/4,\ 8/29], \\ 1 + \dfrac{k-2}{k\,(k-1)} & \text{при } \alpha \leqslant \dfrac{1}{4},\ k = \lfloor \alpha^{-1} \rfloor . \end{cases}$$

Приведенные значения $R_{\mathrm{FFD}}(\alpha)$ при $\alpha \leqslant 1/4$ представляют собой верхние оценки, которые могут быть получены путем прямого обобщения примеров, изображенных на рис. 5.42, 5.43.

На рис. 5.40—5.45 приведены говорящие сами за себя примеры, показывающие, что пределы изменения a_i, указанные в теоремах 5.11 и 5.12, неулучшаемы.

Одна из причин, по которой доказательства большого числа результатов, относящихся к задаче об упаковке в контейнеры, оказываются чрезвычайно сложными, заключается в том, что существуют примеры, подобные приведенному ниже (найденному Сильвией Галац [66]). В этом при-

мере рассматриваются список L и его подсписок L', $L' \subseteq L$, для которых $FFD(L') > FFD(L)$. Такое положение может доставить большие неприятности при попытке построить доказательство методом индукции.

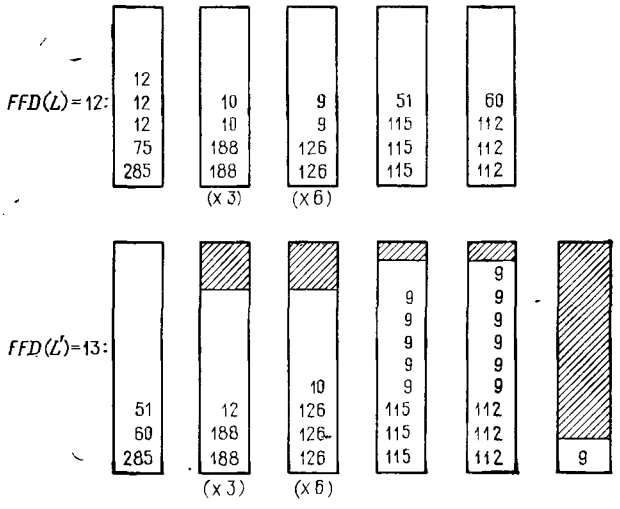


Рис. 5.46.

Пример 17. Пусть

$$L = (285,\ 188\,(\times 6),\ 126\,(\times 18),\ 115\,(\times 3),\ 112\,(\times 3),\ 75,\ 60,\ 51,\ 12\,(\times 3),\ 10\,(\times 6),\ 9\,(\times 12)),$$

$$L' = L - \{75\},$$

где $a\,(\times b)$ означает b копий a. Контейнеры имеют емкость 396. Упаковки, полученные с помощью алгоритма FFD для L и L', показаны на рис. 5.46.

Довольно полное обсуждение многочисленных других алгоритмов упаковки в контейнеры, а также анализ их поведения в среднем (в отличие от поведения в худшем) можно найти в докторской диссертации Д. Джонсона [77, 78].

5.6. Оценки для некоторых других задач

Последний параграф главы мы посвятим описанию некоторых недавних результатов, относящихся к оценке отношения ω/ω_0 при других значениях параметров s, L, $<$, τ и m.

При условии отсутствия ограничений на количество процессоров задача составления расписания с параметрами: $s=1$, все $\tau_i=1$, $<$ пусто, m и L произвольные — совпадает с задачей упаковки в контейнеры, в которой мы получили для ω/ω_0 асимптотическую оценку 17/10. Когда число процессоров m фиксировано (оно может быть много меньше, чем число заданий n), то это эквивалентно условию, чтобы каждый контейнер содержал не более m элементов. Для этого случая справедлив результат, полученный Краузе.

Теорема [91]. *При* $s=1$, $\tau_i=1$, *пустом* $<$, *произвольных* m *и* L *справедливы следующие утверждения:*

1) $(\omega-2)/\omega_0<27/10-24/(10m)$;

2) *существуют примеры, для которых* $\omega/\omega_0\geqslant 27/10-(37)/(10\ m)$.

Следовательно, наличие ограничения на количество процессоров увеличивает рассматриваемое отношение примерно на 1.

Если задания в L расположены в порядке убывания требований на размер ресурса, то списочное расписание оказывается в точности таким, которое получается путем применения алгоритма FFD к соответствующей задаче упаковки в контейнеры, в которой введено ограничение, что ни один контейнер не может содержать больше чем m элементов. Время завершения в расписании обозначено здесь через ω_{FFD}.

Теорема [91]. *Пусть* $s=1$, $<$ *пусто, все* $\tau_i=1$, m *произвольно; тогда справедлива оценка*

$$(\omega_{FFD}-1)/\omega_0\leqslant 2-2/m \quad при \quad m\geqslant 2.$$

Теперь обратимся к ряду интересных результатов, касающихся задач с многими ресурсами (т. е. таких, в которых s может иметь произвольное значение). Быть может, наиболее поразительный из них заключается в следующем.

Теорема 5.13 [51]. *Пусть* $<$ *пусто, все* $\tau_i=1$, $m\geqslant n$ *и* s *и* L *произвольны; тогда справедлива оценка*

$$\omega\leqslant(s+7/10)\,\omega_0+5/2. \tag{31}$$

Более того, коэффициент при ω_0 *является неулучшаемым.*

Разумеется, при $s=1$ эта оценка в точности совпадает с оценкой теоремы 5.9 (с несколько меньшим постоянным слагаемым). В действительности доказательство теоремы 5.13 методом индукции основано на использовании теоремы 5.9 для базы индукции.

Сравнивая этот результат с результатом теоремы 5.6, мы видим, что ограничение, заключающееся в равенстве 1 всех τ_i, позволяет усилить верхнюю оценку отношения ω/ω_0 от $s+1$ до $s+7/10$. Еще раньше Яо было показано [146], что асимптотическая оценка для ω/ω_0 не может превышать $s+17/20$.

Как можно заподозрить на основании теоремы 5.5, введения ограничения, что все $\tau_i=1$, при сохранении общего вида упорядочения $<$ недостаточно для того, чтобы помешать отношению ω/ω_0 сильно увеличиваться. Наиболее сильный из известных в настоящее время результатов, относящихся к этому вопросу, заключается в следующем.

Теорема 5.14 [51]. *Если все* $\tau_i=1$, $m\geqslant n$, s, $<$, L— *произвольные, то справедливы следующие утверждения:*

1) $\dfrac{\omega}{\omega_0}\leqslant\dfrac{1}{2}s\omega_0+\dfrac{1}{2}s+1$;

2) *существуют примеры, для которых*

$$\frac{\omega}{\omega_0}\geqslant\frac{1}{2}s\omega_0+\frac{1}{2}s+1-\frac{2s}{\omega_0}.$$

Если расписание составляется методом критического пути, то оценка величины ω/ω_0 для наихудшего случая значительно улучшается. В частности, может быть доказан следующий результат [51].

Теорема 5.15. *Если все* $\tau_i=1$, $m\geqslant n$, s *и* $<$ — *произвольные, то справедливы следующие утверждения:*

1) $\dfrac{\omega_{CP}}{\omega_0}>\dfrac{17}{10}s+1$;

2) *для любого* $\varepsilon>0$ *существуют примеры, для которых*

$$\frac{\omega_{CP}}{\omega_0}>\frac{17}{10}s+1-\varepsilon.$$

Напомним, что в частном случае $s=0$, даже если имеются ограничения на число процессоров (т. е. допускается $m<n$), Ху показал, что $\omega_{CP}=\omega_0$ (см. гл. 2).

Имеется несколько естественных обобщений рассмотренной модели системы с дополнительными ресурсами, исследование которых только начинается. Эти обобщения допускают различное быстродействие процессоров (см. гл. 2), невозможность выполнять некоторые задания на некоторых процессорах, а также наличие контейнеров различного объема. Однако ограниченный объем книги не позволяет нам здесь рассмотреть эти вопросы.

ГЛАВА 6

ПЕРЕЧИСЛИТЕЛЬНЫЕ И ИТЕРАТИВНЫЕ АЛГОРИТМЫ

6.1. Введение

В этой главе мы рассмотрим практические методы решения таких задач составления расписаний, для которых отсутствуют полиномиальные алгоритмы нахождения оптимальных расписаний. Как показано в гл. 4, общая задача упорядочения выполнения работ и, в частности, конвейерная задача попадают в эту категорию. Разумеется, термин «практический» является весьма расплывчатым; в некоторых случаях мы можем потребовать, чтобы алгоритм, работающий в реальном времени, производил упорядочение двух десятков работ за считанные миллисекунды. В других случаях расчет оптимального или субоптимального расписания за несколько минут оказывается вполне допустимым.

Мы можем разбить вычислительные методы, предназначенные для составления расписаний (или решения других комбинаторных оптимизационных задач), на классы в соответствии с тем, предназначены ли они для определения оптимального решения или только приближенного решения. Назовем методы первого класса *точными*, а методы второго класса — *приближенными*. Мы также рассмотрим третий важный класс методов, а именно таких, которые позволяют найти приближенное решение с допустимым гарантированным отклонением от оптимального решения. Мы их назовем методами с *гарантированной точностью*. Когда мы сталкиваемся с задачей, такой большой и сложной, что поиск оптимального решения оказывается неосуществимым, хорошим выходом является нахождение решения, качество которого было бы известным. Если данные содержат неопределенность или критерий стоимости допускает некоторый произвол, то решение с гарантированной точностью может оказаться фактически таким же хорошим, как и точное.

Все точные методы, предназначенные для решения общих задач, рассматриваемых в этой главе, включают в том или ином виде перебор, так как невозможно избежать взрывоподобного увеличения размерности пространства решений (по крайней мере, сейчас мы не знаем, как это сделать). Идея заключается в том, чтобы осуществлять перебор эффективно, используя при этом как можно больше информации для сокращения дерева поиска. Здесь мы обсудим общую схему такого перебора с возвратом, включающую все, что в литературе обычно называют алгоритмами ветвей и границ. В настоящей главе представлено обобщение подходов, изложенных в [88, 94, 96].

Приближенные алгоритмы — это любые схемы, которые могут привести к получению допустимого решения с низкой стоимостью. Плодотворный подход часто формируется при исследовании частных случаев, и правило, дающее оптимальный результат в простых случаях, такое, как «первой назначается работа с наименьшей длительностью», может быть перенесено на сложные ситуации. Несмотря на кажущуюся уязвимость этого подхода, в последние десятилетия появилось несколько общих методов, которые представляются эффективными для широкого круга задач. Одним из них является *поиск в локальной окрестности*, другим — *метод ветвей и границ без возвратов*. Мы рассмотрим оба этих подхода совместно с экспериментальными результатами для среднего времени завершения в конвейерной задаче для двух машин.

В заключение мы обсудим подход, в котором сочетаются точный и приближенный методы, а именно применяется комбинация точного и приближенного алгоритмов для более быстрого получения точного решения или для нахождения решения, обладающего заданной точностью. Подход, гарантирующий заданную точность, может рассматриваться как метод доказательства теорем в очень частном случае, когда допустимое расписание предъявляется вместе с доказательством того, что его стоимость лежит в некоторых границах по отношению к оптимальной.

При создании эффективного алгоритма для задачи, являющейся слишком большой и сложной для точного решения, разработчик в основном полагается на интуицию и знание особенностей задачи. В алгоритмах ветвей и границ это отражается на построении процедур вычисления нижней границы (оценки), правила доминирования и расчета верхней границы (оценки). Нижняя оценка сложности для метода ветвей и границ невозможна, так как если бы была из-

вестна идеальная нижняя граница и идеальный эвристический алгоритм определения начального решения, то работа была бы закончена, не начавшись.

В следующем параграфе вводится общая схема для алгоритмов ветвей и границ. Она включает ситуации переполнения памяти, превышения времени счета, а также остановки при достижении заданной точности. За этим следуют доказательство корректности и ряд недавних результатов, касающихся зависимости потребностей в вычислительных ресурсах от правил отсечения, определения допустимых решений, от правила доминирования, нижней оценки и заданной степени точности. Далее мы рассмотрим приближенные алгоритмы, включающие эвристические процедуры и поиск в локальной окрестности. В заключение представлены результаты расчетов для двухмашинной конвейерной задачи, которая упоминалась выше. Описаны характеристики точных, приближенных и гарантирующих заданную точность алгоритмов. Особое внимание уделено извлечению из результатов машинных расчетов наиболее полезной информации — в виде допустимых решений и нижних оценок оптимальных решений.

6.2. Алгоритмы ветвей и границ для задач о перестановках

6.2.1. Предпосылки. Под методами ветвей и границ подразумевают алгоритмы перебора, которые появились не так давно как фундаментальный общий метод нахождения оптимальных решений дискретных оптимизационных задач. Методика ветвей и границ быстро развилась, и сейчас полный список работ, связанных с ней, насчитывает несколько сотен наименований. Характерными примерами являются работы, посвященные конвейерным задачам и общей задаче составления расписаний [8, 76, 128, 129], задаче коммивояжера [68], общим квадратичным задачам о назначении [53, 118] и задачам целочисленного программирования [58, 60]. Хотя метод ветвей и границ был развит и обобщен многими авторами [1, 7, 60, 107, 113, 120, 123], очень немногие из них рассматривали зависимость относительных потребностей в вычислительных ресурсах от выбора параметров алгоритма. Одним из исключений является недавняя работа Фокса и Шрейджа [44], которые теоретически сравнили относительное число вершин, обследуемых алгоритмами ветвей и границ, для задачи целочисленного

программирования при трех различных стратегиях ветвления (правилах выбора следующей вершины).

Хотя алгоритмы ветвей и границ обычно являются более эффективными, чем полный перебор, все же их требования в вычислительных ресурсах растут как экспоненты или полиномы высокой степени от размера задачи n. В такой ситуации реальные задачи, как правило, не могут быть решены точно. Недавние результаты в области вычислительной сложности дают большие основания считать (хотя это еще не доказано), что потребности в вычислительных ресурсах для нахождения точных решений большого числа таких комбинаторных задач не могут быть ограничены полиномом от длины строки входных данных [11, 86, 143]. Такие задачи называются *NP-полными* (см. гл. 4).

Во многих практических ситуациях не обязательно получить точное решение. Приемлемыми могут считаться такие решения, о которых известно, что их стоимость отличается от оптимальной на несколько процентов. Проблемы получения решений, стоимость которых отличалась бы от оптимального значения не более чем на заданное число процентов, привлекли сравнительно небольшое внимание [53, 96, 90]. Похожий подход заключается в нахождении решений с помощью таких эвристических алгоритмов, для которых вычислены границы характеристик для наихудшего случая [36, 49, 46] (см. гл. 5). Эти границы являются фиксированными. Поскольку они должны быть справедливы во всех случаях (включая вырожденные), то часто они оказываются весьма широкими. В методах, рассматриваемых в настоящей главе, допуск, определяющий величину максимального допустимого отклонения, может быть задан произвольно. Точность, определяемая этим допуском, в принципе достижима, но это может потребовать больше вычислительных ресурсов, чем выделено. Важную роль при этом играет получаемое эвристическими методами субоптимальное решение, которое дает начальную верхнюю границу. Алгоритм ветвей и границ может либо проверить, что данное решение удовлетворяет заданному допуску, либо построить новое решение, которое бы ему удовлетворяло.

В следующих двух пунктах мы рассмотрим общую схему классификации алгоритмов ветвей и границ для решения задачи о перестановках, основанную на девятке параметров (B_p, S, E, F, D, L, U, BR, RB), где B_p — правило ветвления, или разбиения, S — правило выбора следующей вершины, E — набор правил исключения вершин, F —ха-

рактеристическая функция, позволяющая обнаруживать вершины, не обеспечивающие допустимого решения, D — отношение доминирования для вершин, L — функция нижней оценки вершин, U — верхняя оценка стоимости решения, BR — заданная точность решения, RB — предельное количество имеющихся ресурсов, включая время вычислений и память. Общая схема метода в терминах задачи целочисленного программирования, недавно предложенная Джоффрионом и Марстеном [58], согласуется с нашим подходом, но является менее подробной. Мы покажем, каким образом девятка параметров может быть использована для описания класса обычных алгоритмов ветвей и границ для общей задачи о перестановках, включая общую задачу составления расписаний [25, 128], пространство допустимых решений которой является подмножеством множества всех перестановок n объектов. В рамках этой основной схемы мы опишем ряд теоретических результатов, касающихся поведения чисел порождаемых вершин и активных вершин как функций от выбора девятки параметров. Как мы увидим, эти результаты подтверждают одни наши интуитивные представления и опровергают другие.

6.2.2. Предварительные определения и обозначения.

1. *Общая задача о перестановках* размерности n является комбинаторной оптимизационной задачей, определяемой тройкой $(\mathscr{P}, X, f)$, элементы которой обозначают следующее:

а) $\mathscr{P}$ — *пространство решений*; $\mathscr{P} \subseteq \mathscr{P}_n \triangleq \{$перестановка из n объектов$\} = \{\pi\}$;

б) X — *пространство параметров*; каждая точка $x \in X$ представляет собой допустимый набор «данных» задачи;

в) f — *функция стоимости*; $f\colon \mathscr{P} \times X \to \mathbf{R}$, где $f(\pi, x)$ — стоимость решения π при значении параметра x.

2. *Глобально оптимальное решение* для значения параметра x представляет собой решение $\pi^* \in \mathscr{P}$ такое, что $f(\pi^*, x) \leqslant f(\pi, x)$ для всех $\pi \in \mathscr{P}$.

3. Для заданного множества $M = \{i_1, i_2, \ldots, i_j\} \subseteq N \triangleq \{1, 2, \ldots, n\}$ π^M означает перестановку на множестве M:

а) если $M = N$, то π^M называется *полной перестановкой на* N;

б) если $M \subseteq N$, то π^M называется *частичной перестановкой на* N;

в) если $M = \varnothing$, то $e \triangleq \pi^{\varnothing}$.

4. Пусть задано $\pi_r^M \triangleq (r_1, r_2, \ldots, r_j)$. Тогда:

а) $|\pi_r^M| \triangleq$ {размерность частичной перестановки π_r^M} $= |M| = j$;

б) $\overline{M} \triangleq N - M$;

в) если $l \in \overline{M}$, то полагаем $\pi_r^M \circ l \triangleq (r_1, r_2, \ldots, r_j, l)$;

г) если π_s^K — перестановка множества $K \subseteq \overline{M}$ и $\pi_s^K = (s_1, s_2, \ldots, s_k)$, то полагаем $\pi_r^M \circ \pi_s^K \triangleq (\ldots((\pi_r^M \circ s_1) \circ s_2) \ldots \ldots \circ s_k)$;

д) $\{\pi_r^M \circ\} \triangleq \{\pi_r^M \circ \pi_s^{\overline{M}} \mid \pi_s^{\overline{M}}$ есть некоторая перестановка на $\overline{M}\} \triangleq \{$*продолжение* $\pi_r^M\} \triangleq \{$множество всех полных перестановок, начинающихся с частичной перестановки $\pi_r^M\}$.

5. Для заданного множества Y частичных перестановок на N

$$\{Y \circ\} \triangleq \{\cup \{\pi_y^K \circ\} \mid \pi_y^K \in Y\} \triangleq \{\textit{продолжение } Y\}.$$

6. Для заданного π_r^M, $M \subseteq N$:

а) *потомком* π_r^M называется любая частичная перестановка $\pi_s^P = \pi_r^M \circ \pi_t^K$ при $K \subseteq \overline{M}$;

б) *непосредственным потомком* (*сыном*) π_r^M называется любой потомок $\pi_x^P = \pi_r^M \circ \pi_t^K$ такой, что K является одноэлементным множеством и $K \subseteq \overline{M}$ (т. е. $\pi_x^P = \pi_r^M \circ l$ для некоторого $l \in \overline{M}$);

в) *предком* π_r^M называется любая перестановка π_q^L такая, что $\pi_r^M = \pi_q^L \circ \pi_t^K$, $K \neq \varnothing$;

г) *непосредственным предком* (*отцом*) π_r^M называется предок π_q^L такой, что K является одноэлементным множеством, т. е. если $\pi_r^M = (r_1, r_2, \ldots, r_j)$, то отцом π_r^M является $\pi_q^M = (r_1, r_2, \ldots, r_{j-1})$.

7. Между множеством частичных решений задачи о перестановках $(\mathcal{P}, x, f)$ и множеством частичных перестановок на N имеется взаимно однозначное соответствие. Каждое частичное решение может быть затем представлено в виде соответствующей частичной перестановки. Термины *частичное решение* и *частичная перестановка* используются как синонимы.

6.2.3. Девятка (B_p, S, E, F, D, L, U, BR, RB). Класс алгоритмов ветвей и границ, обычно используемых для решения общей задачей о перестановках $(\mathcal{P}, X, f)$, может быть описан с помощью девятки объектов (B_p, S, E, F, D, L, U, BR, RB). Это является обобщением подхода, рассмотренного в [94]. Девятка параметров определяется следующим образом.

1. *Правило ветвления* B_p описывает процесс ветвления в задаче о перестановках. Целью ветвления является раз-

биение множества полных перестановок $\mathcal{P}_n$ на непересекающиеся подмножества. Эти подмножества представляются в виде вершин. Каждой вершине поставлена в соответствие перестановка $\pi_y^{M_y}$ (метка), заданная на множестве $M_{y_{M_y}}$ для $M_y \subseteq N = \{1, 2, \ldots, n\}$. Вершина, помеченная $\pi_y^{M_y}$, представляет собой множество полных перестановок $\{\pi_y^{M_y} \circ\}$. Частичная перестановка $\pi_y^{M_y}$ является предком каждой перестановки, входящей в множество $\{\pi_y^{M_y} \circ\}$. Ветвление в вершине $\pi_b^{M_b} = (b_1, b_2, \ldots, b_k)$ представляет собой процесс разбиения множества $\{\pi_b^{M_b} \circ\}$ на непересекающиеся подмножества: $\{\pi_b^{M_b} \circ\} = \bigcup\limits_{l \in \overline{M}_b} \{(\pi_b^{M_b} \circ l) \circ\}$.

Назовем $\pi_b^{M_b}$ *вершиной ветвления*. Из сказанного следует, что $\{(\pi_b^{M_b} \circ i) \circ\} \cap \{(\pi_b^{M_b} \circ j) \circ\} = \varnothing$ для $i, j \in \overline{M}_b$, $i \neq j$. Вершины $\pi_b^{M_b} \circ l$, $l \in \overline{M}_b$, являются сыновьями вершины ветвления $\pi_b^{M_b}$. Эти вершины называются *порожденными при шаге ветвления* $\pi_b^{M_b}$. Для того чтобы упростить это громоздкое обозначение, вместо $\pi_y^{M_y}$ используется запись π_y, в которой множество M_y подразумевается.

2. *Правило выбора* S служит для выбора следующей вершины ветвления π_b из текущего множества активных вершин. В процессе выполнения алгоритма (B_p, S, E, F, D, L, U, BR, RB) вершина π_y называется *текущей активной* вершиной в том и только в том случае, когда она порождена, но еще не исключена и не подвергнута ветвлению. Сыновья вершины ветвления π_b порождаются в лексикографическом порядке. Сын $\pi_b \circ l$, $l \in \overline{M}_b$, добавляется к множеству текущих активных вершин тогда (но не только тогда), когда: 1) он имеет допустимое продолжение, т. е. $\{(\pi_b \circ l) \circ\} \cap \mathcal{P} \neq \varnothing$; 2) он не исключается путем применения одного из правил исключения вершин E и 3) он не потерян в результате превышения заранее заданного объема множества текущих активных вершин. Алгоритм всегда начинает работу с выбора $e = \pi^{\varnothing}$ в качестве первой вершины ветвления (можно также считать, что e — первая порожденная вершина). Работа алгоритма завершается, когда следующая вершина ветвления представляет собой полное решение.

Для иллюстрации одного из общеупотребительных правил исключения рассмотрим следующий пример.

Дано: множество независимых заданий $\mathcal{T} = \{T_1, T_2, T_3\}$, имеющих времена выполнения $\tau_1 = 3$, $\tau_2 = 2$ и $\tau_3 = 1$.

Найти: расписание π^* выполнения заданий на одном процессоре, которое минимизирует сумму времен завершения заданий.

В этой задаче $\mathcal{P} = \mathcal{P}_3$, $x \in X$—вектор времен выполнения, $x = (\tau_1, \tau_2, \tau_3)$, а функция стоимости $f(\pi, x)$ представляет собой сумму времен завершения при данном расписании $\pi \in \mathcal{P}$ и временах выполнения $x \in X$. Например,

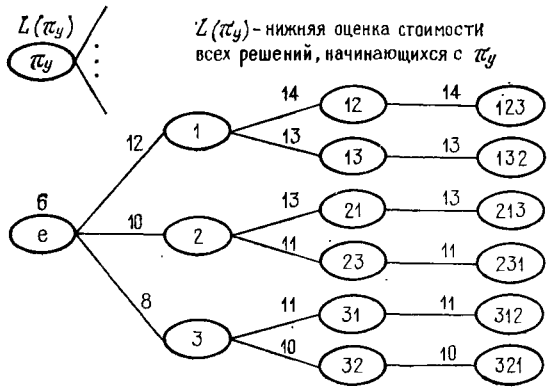


Рис. 6.1.

при $\pi = (1, 2, 3)$ T_1 завершается в момент времени 3, T_2—в момент времени $3+2=5$ и T_3—в момент $5+1=6$. Следовательно, $f(\pi, x) = 14$. На рис. 6.1 показано полное дерево перебора.

а) Правило $S = \mathrm{LLB}$ (*least-lower-bound*). Выбирается текущая активная вершина π_a, которая имеет наименьшую нижнюю оценку стоимости $L(\pi_a)$. В случае, когда несколько вершин оказываются равноправными по этому критерию, то выбирается либо первая порожденная вершина, ($\mathrm{LLB_{FIFO}}$), либо последняя, ($\mathrm{LLB_{LIFO}}$). В обоих случаях порядок, в котором вершины подвергаются ветвлению, определяется функцией нижней оценки L. На рис. 6.2 показан порядок, в котором вершины из примера, изображенного на рис. 6.1, выбираются для ветвления при использовании правил LLB.

б) Правило S=FIFO (*first-in first-out*). Выбирается та текущая активная вершина, которая была порождена первой. Этот метод поиска *в ширину* не зависит от функции нижней оценки L. На рис. 6.3 показан пример дерева, вершины которого пронумерованы в соответствии с правилом FIFO.

в) Правило S=LIFO (*last-in first-out*). Выбирается та текущая активная вершина, которая была порождена последней, и пропускаются активные вершины, которые со-

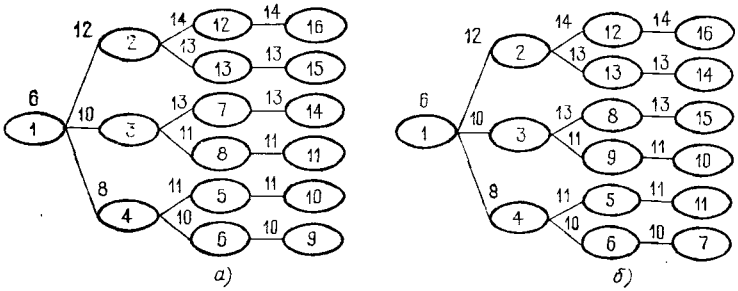


Рис. 6.2. Процесс перебора с использованием правил LLB. *а*) LLB_{FIFO}; *б*) LLB_{LIFO}.

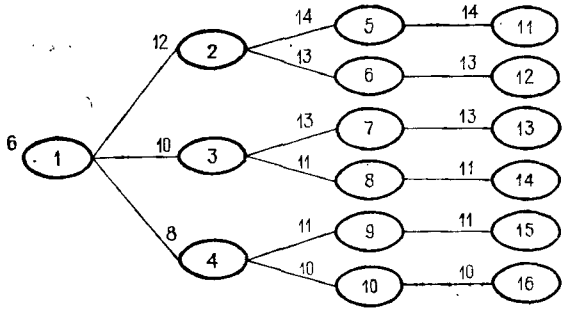


Рис. 6.3.

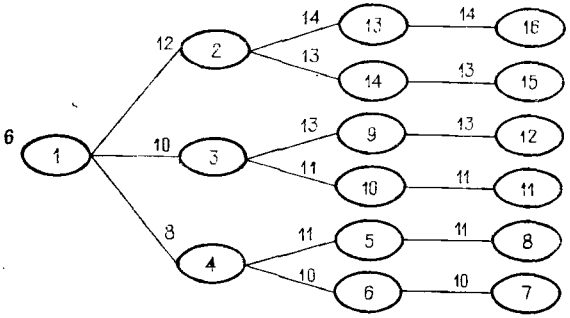


Рис. 6.4.

держат полные решения, до тех пор, пока нет незавершенных активных вершин. Получаем поиск *в глубину*, при котором порядок выбора вершин снова не зависит от функции нижней оценки L (см. рис. 6.4).

г) Правило $S = \mathrm{DF/LLB}$ (*depth-first/least-lower-bound*). Из множества последних порожденных активных сыновей выбирается сын с наименьшей нижней оценкой, но при этом пропускаются множества сыновей, представляющих собой полные решения, до тех пор, пока отсутствуют незавершенные активные вершины. В случае равнозначности вершин по этому критерию выбирается сын, который был порожден первым, т. е. $\mathrm{DF/LLB_{FIFO}}$, или последним, т. е. $\mathrm{DF/LLB_{LIFO}}$. На рис. 6.5 показано наше дерево, вершины которого пронумерованы в соответствии с правилом DF/LLB (при отсутствии случаев равенства).

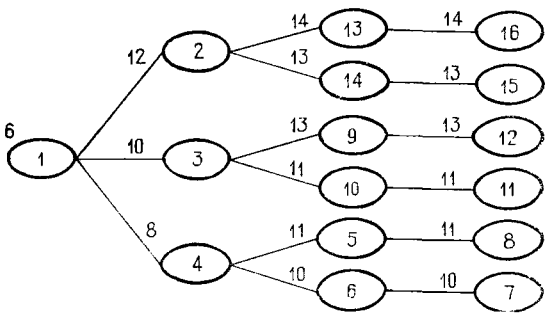


Рис. 6.5.

3. *Характеристическая функция* F используется для исключения тех частичных перестановок, о которых известно, что они имеют продолжения на множестве допустимых решений $\mathcal{P}$. Конкретно, функция F ставит в соответствие каждой частичной перестановке из $\{\pi_y \mid \{\pi_y \circ\} \in \mathcal{P}_n\}$ число 0 или 1 следующим образом:

а) $F(\pi_y) = 1$ для любой частичной перестановки π_y, если $\{\pi_y \circ\} \cap \mathcal{P} \neq \varnothing$;

б) $F(\pi_y^N) = 1$ для любой полной перестановки π_y^N тогда и только тогда, когда $\pi_y^N \in \mathcal{P}$.

Заметим, что наше определение не требует, чтобы $F(\pi_y) = 0$, когда $\{\pi_y \circ\} \cap \mathcal{P} = \varnothing$, если только π_y не является полной перестановкой. Это отражает то, что при порождении π_y бывает нелегко определить, что она не имеет допустимого продолжения. Множество всех частичных перестановок π_y, для которых $F(\pi_y) = 1$, обозначается буквой P. Это есть множество возможных частичных решений.

4. *Отношение доминирования* D представляет собой бинарное отношение, определенное на множестве частичных

решений задачи $(\mathcal{P}, x, f)$. Для любого частичного решения (вершины) π_y такого, что $\{\pi_y \circ\} \cap \mathcal{P} \neq \varnothing$, $\hat{\pi}_y^N$ будет обозначать полное решение с минимальной стоимостью, начинающееся с π_y. Это означает, что $\hat{\pi}_y^N \in \{\pi_y \circ\} \cap \mathcal{P}$ и $f(\hat{\pi}_y^N, x) = \min\{f(\pi_w, x) \mid \pi_w \in \{\pi_y \circ\} \cap \mathcal{P}\}$. $\mathcal{D}$ является транзитивным бинарным отношением на множестве частичных решений таким, что $\pi_y \mathcal{D} \pi_z$ тогда и только тогда, когда:

а) $\{\pi_y \circ\} \cap \mathcal{P} \neq \varnothing$ и $\{\pi_z \circ\} \cap \mathcal{P} \neq \varnothing$;

б) $f(\hat{\pi}_y^N, x) \leqslant f(\hat{\pi}_z^N, x)$.

Отношение D является подмножеством $\mathcal{D}$, обладающим следующими свойствами:

а) (Транзитивность) $\pi_w D \pi_y$ и $\pi_y D \pi_z$ только тогда, когда $\pi_w D \pi_z$.

б) (Согласованность) $\pi_y D \pi_z$ только тогда, когда $L(\pi_y) \leqslant L(\pi_z)$.

Поскольку $D \subseteq \mathcal{D}$, то это сразу означает, что $\pi_y D \pi_z$ только в том случае, когда $\hat{\pi}_y^N$ — потомок с минимальной стоимостью вершины π_y — имеет не большую стоимость, чем $\hat{\pi}_z^N$ — потомок с минимальной стоимостью вершины π_z. Когда $\pi_y D \pi_z$, то говорят, что π_y *доминирует* над π_z. Это похоже на метод, предложенный Шварцем для конвейерной задачи составления расписания [141]. Обычно предполагают, что D определено так, чтобы содержать по крайней мере все пары $\pi_y D \pi_z$, для которых $L(\pi_y) \leqslant L(\pi_z)$ и $M_y = N$. При использовании алгоритма ветвей и границ пары вида (π_y, π_z) проверяются на предмет принадлежности к D. Следовательно, при выборе отношения D обычно приходится идти на компромисс между сильным отношением и таким, принадлежность к которому легко проверяется.

5. *Функция нижней оценки* L ставит в соответствие каждому частичному решению $\pi_y \in \mathrm{P}$ действительное число $L(\pi_y)$, представляющее собой нижнюю оценку стоимости для всех полных решений, содержащихся в множестве $\{\pi_y \circ\} \cap \mathcal{P}$. Требуется, чтобы для всех π_y, $\pi_z \in \mathrm{P}$ L имела следующие свойства:

а) если π_z — потомок π_y, то $L(\pi_z) \geqslant L(\pi_y)$;

б) для каждого полного решения π_y^N $L(\pi_y^N) = f(\pi_y^N, x)$.

При выборе L обычно принимается компромиссное решение между строгой функцией нижней оценки, т. е. такой, которая вычисляет нижнюю оценку $L(\pi_y)$ так, что ее значение оказывается очень близким к действительному зна-

чению $f(\hat{\pi}_y^N, x)$ для потомка с минимальной стоимостью, и такой функцией нижней оценки, которая была бы легко вычислимой.

Рассмотрим задачу, изображенную на рис. 6.1. В этом случае частичное решение $\pi_y^M = (y_1, y_2, \ldots, y_j)$ представляет собой такую последовательность, в которой задание T_{y_1} выполняется первым, за ним следует T_{y_2} и так далее. Нижняя оценка суммы времен завершения всех заданий, начинающихся с частичного решения π_y^M, была вычислена следующим образом:

$$L(\pi_y) = \sum_{y_i \in M_y} \{\text{время завершения задания } T_{y_i}\} + \\ + \sum_{y_i \in \overline{M}y} (\{\text{время завершения задания } T_{y_i}\} + \tau_{y_i}).$$

6. *Верхняя оценка стоимости* U представляет собой стоимость $f(\pi_u^N, x)$ некоторого полного решения π_u^N, известного в начале выполнения алгоритма. Если не известно ни одного полного решения, то полагают $U = \infty$, т. е. U больше любого возможного значения стоимости. На каждом этапе алгоритма известное решение с наименьшей стоимостью обозначается через $\hat{\pi}_u^N$, а его стоимость—через $\hat{U} = f(\hat{\pi}_u^N, x)$. Показано, что начальный выбор решения, верхняя оценка стоимости которого близка к оптимальному значению, может существенно сократить общий объем вычислений [96].

7. *Правила исключения* E представляют собой множество правил использования функции доминирования D и верхней оценки стоимости U с целью вывода из числа активных (исключения) вновь порожденных и текущих активных вершин. Длязаданного алгоритма (B_p, S, E, F, D, L, U, BR, RB) пусть π_b—текущая вершина ветвления, $\pi_b \circ l$—ее непосредственный потомок в множестве допустимых частичных решений P и π_a—член текущего активного множества на шаге ветвления π_b, $\pi_b \neq \pi_a$. Множество E представляет собой подмножество набора из следующих трех правил.

а) U/DBAS (проверяется доминирование потомков вершины ветвления над текущими активными вершинами по критерию верхней оценки). Если $L(\pi_b \circ l) > \hat{U}$, то все члены $\{(\pi_b \circ l) \circ\} \cap \mathcal{P}$ имеют бо́льшую стоимость, чем решение $\hat{\pi}_u^N$ с верхней оценкой $\hat{U} \triangleq f(\hat{\pi}_u^N, x)$. В этом случае $\pi_b \circ l$ исключается сразу после порождения, не будучи

занесенным в число активных. Если $L(\pi_a) > \hat{U}$, то все члены $\{\pi_a \circ\} \cap \mathcal{P}$ имеют бо́льшую стоимость, чем решение $\hat{\pi}_u^N$ (верхняя оценка), и π_a удаляется (исключается из активного множества).

б) AS/DB (проверяется доминирование множества активных вершин над потомками вершины ветвления). Каждая текущая активная вершина проверяется на доминирование по отношению к каждому потомку вершины ветвления. Если $\pi_a D(\pi_b \circ l)$, то $\pi_b \circ l$ исключается после порождения до занесения в число активных.

в) DB/AS (проверяется доминирование потомков вершины ветвления над множеством текущих активных вершин). Каждый потомок текущей вершины ветвления проверяется на доминирование по отношению к каждой текущей активной вершине. Если $(\pi_b \circ l) D \pi_a$, то π_a удаляется (исключается) из множества текущих активных вершин.

Если E содержит правила AS/DB и DB/AS, то множество активных вершин может зависеть от порядка, в котором эти правила применяются. Мы полагаем, что AS/DB применяется раньше, чем DB/AS.

8. *Допуском* BR называется действительное число, $0 \leqslant BR \leqslant 1$, представляющее собой желаемую величину максимального относительного отклонения оптимальной стоимости от стоимости приемлемого решения. На каждой стадии алгоритма $\hat{U}$ представляет собой стоимость лучшего из известных полных решений, а $\hat{L}$ — известную нижнюю оценку стоимости оптимального решения. При $(\hat{U} - \hat{L})/\hat{U} \leqslant BR$ имеем

$$(1 - BR)\,\hat{U} \leqslant \hat{L}.$$

Следовательно, стоимость $f(\pi^*, x)$ оптимального решения π^* удовлетворяет неравенству

$$(1 - BR)\,\hat{U} \leqslant f(\pi^*, x) \leqslant \hat{U}.$$

Если BR выбрано равным 0, то лучшее из текущих решений $\hat{\pi}_u^N$ считается приемлемым только тогда, когда оно является оптимальным; если же BR выбрано равным 0,05, то решение считается приемлемым, если оптимальная стоимость оказывается ниже полученной верхней оценки $\hat{U}$ не более чем на 5%. Опыт практических расчетов показывает, что если допуск задается в пределах от 5 до 10%, то требуемый объем вычислений может быть существенно сокращен [96].

9. *Предельное количество ресурсов RB* представляет собой вектор, компонентами которого являются верхние границы общего времени, отведенного для расчетов, и объем памяти для хранения активных вершин и непосредственных потомков вершины ветвления. В частности, $RB=$ $=$ (TIMELIMIT, MAXSZAS, MAXSZDB), где TIMELIMIT обозначает заранее заданное максимальное время счета, MAXSZAS$\geqslant 1$ — максимальный разрешенный размер множества активных вершин и MAXSZDB$\geqslant 1$ — максимальный разрешенный размер множества потомков каждой вершины ветвления. Когда требуемое время счета превышает TIMELIMIT, выполнение алгоритма останавливается и выводится последнее состояние процесса поиска. Если превышается предел по памяти, то теряются одно или несколько частичных решений, потомки которых могли бы стать оптимальными решениями. Нижняя оценка стоимости всех потомков устанавливается с учетом наименьшей нижней оценки вершин, отброшенных в результате переполнения. Эта оценка обозначается $\hat{L}_{\mathrm{OFS}}$. Нижняя оценка оптимального решения $\hat{L}$ не может превышать $\hat{L}_{\mathrm{OFS}}$.

6.2.4. Описание параметров и блок-схема алгоритма. Для подробного описания алгоритма $BB=(B_p, S, E, F, D, L, U, BR, RB)$ введены следующие обозначения:

$\mathrm{P} \triangleq \{\pi_y^{M_y} \mid M_y \subseteq N$ и $F(\pi_y)=1\}$ — множество всех допустимых частичных решений для $(\mathcal{P}, x, f)$;

$\mathrm{BFS}(\pi_b)$ — множество предыдущих вершин ветвления перед началом шага ветвления π_b;

$\mathrm{DB}(\pi_b) \subseteq \{(\pi_b^{M_b} \circ l) \mid l \in \bar{M}\}$ — множество допустимых непосредственных потомков, порожденных в процессе шага ветвления π_b и не отброшенных в результате переполнения;

$\mathrm{AS}(\pi_b)$ — множество текущих активных вершин перед началом шага ветвления π_b;

$\mathrm{OFS}(\pi_b)$ — множество всех вершин, которые вызвали переполнение либо $\mathrm{AS}(\pi_y)$, либо $\mathrm{DB}(\pi_y)$ для некоторого $\pi_y \in \mathrm{BFS}(\pi_b)$ (множество всех вершин, вызвавших переполнение в результате начала шага ветвления π_b);

$\mathrm{ES}(\pi_b)$ — множество допустимых вершин, исключенных в процессе шага ветвления π_b одним из правил исключения, входящих в E;

$\hat{U}(\pi_b)$ — стоимость решения, имеющего минимальную верхнюю оценку стоимости перед началом шага ветвления π_b;

$\hat{L}_{\mathrm{AS}}(\pi_b)$ — нижняя оценка стоимости всех решений, вы-

численная по полному множеству активных вершин в начале шага ветвления π_b;

$\hat{L}_{OFS}(\pi_b)$ — нижняя оценка стоимости всех решений, вычисленная по полному множеству вершин, вызывающих переполнение, в начале шага ветвления π_b (равна ∞, если $OFS \neq \varnothing$);

$\hat{L}(\pi_b) \triangleq \min\{\hat{L}_{AS}(\pi_b), \hat{L}_{OFS}(\pi_b)\}$ — нижняя оценка стоимости оптимального решения в начале шага ветвления π_b.

С помощью введенных обозначений правила исключения E могут быть представлены в одном из двух видов: (1) любое π_z из множества B исключается, если $\hat{U}(\pi_b) < \hat{L}(\pi_z)$, или (2) любое π_z из B исключается, если существует некоторое π_y из множества A такое, что $\pi_y D \pi_z$. На рис. 6.6 представлено полное множество правил исключения, где $A \rightarrow B$ представляет правило E_i.

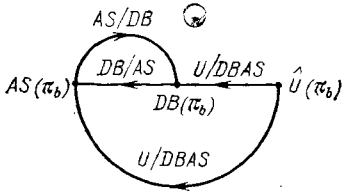


Рис. 6.6.

Блок-схема общего алгоритма ветвей и границ $BB = (B_p, S, E, F, D, L, U, BR, RB)$ изображена на рис. 6.7 в обозначениях, предложенных Бруно и Штиглицем [19]. При использовании алгоритма BB множество BFS вершин, подлежащих ветвлению, и множество OFS вершин, вызывающих переполнение, не будут запоминаться. Они показаны на схеме только для наглядности. В этом случае $\hat{L}_{OFS}$ изменяется всякий раз, когда вершина, имеющая меньшую нижнюю оценку стоимости, чем текущее значение $\hat{L}_{OFS}$, вызывает переполнение AS или DB. На рис. 6.8 представлена детальная блок-схема, описывающая процесс непосредственного применения правил исключения. Можно использовать и более эффективные в вычислительном аспекте методы, но простота данного способа придает ему значительные преимущества.

6.2.5. Доказательство корректности. В данном пункте доказывается ряд лемм, с помощью которых устанавливается, что общий алгоритм ветвей и границ $(B_p, S, E, F, D, L, U, BR, RB)$ завершается (1) нахождением решения, удовлетворяющего допуску BR при условии, что не превышается предельное количество требуемых ресурсов, или (2) нахождением допуска на величину оптимального решения, даже если превышается предельное количество ресурсов. Начнем с доказательства того факта, что объединение

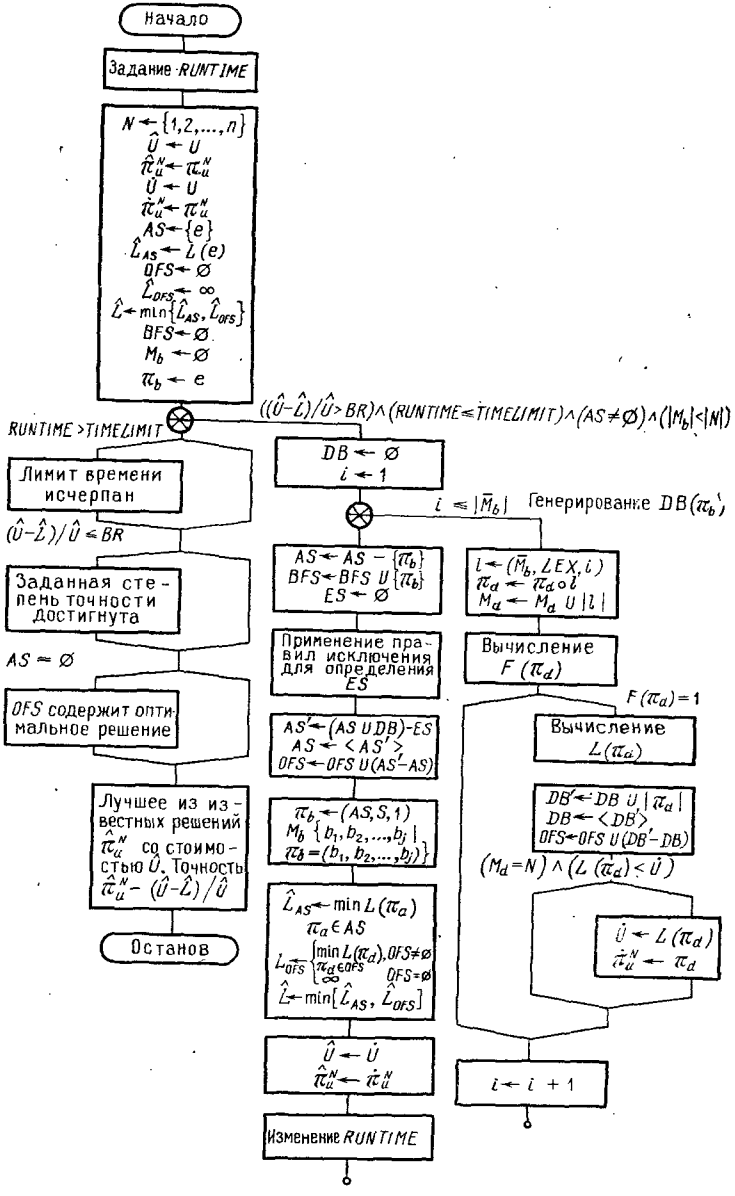

Начало
Задание RUNTIME
$N \leftarrow \{1,2,\ldots,n\}$
$\hat{U} \leftarrow U$
$\hat{\pi}_u^N \leftarrow \pi_u^N$
$\dot{U} \leftarrow U$
$\dot{\pi}_u^N \leftarrow \pi_u^N$
$AS \leftarrow \{e\}$
$\hat{L}_{AS} \leftarrow L(e)$
$OFS \leftarrow \varnothing$
$\hat{L}_{OFS} \leftarrow \infty$
$\hat{L} \leftarrow \min\{\hat{L}_{AS}, \hat{L}_{OFS}\}$
$BFS \leftarrow \varnothing$
$M_b \leftarrow \varnothing$
$\pi_b \leftarrow e$
$((\hat{U}-\hat{L})/\hat{U} > BR) \wedge (RUNTIME \leq TIMELIMIT) \wedge (AS \neq \varnothing) \wedge (|M_b| < |N|)$
$RUNTIME > TIMELIMIT$
Лимит времени исчерпан
$(\hat{U}-\hat{L})/\hat{U} \leq BR$
Заданная степень точности достигнута
$AS = \varnothing$
OFS содержит оптимальное решение
Лучшее из известных решений $\hat{\pi}_u^N$ со стоимостью $\hat{U}$. Точность $\hat{\pi}_u^N - (\hat{U}-\hat{L})/\hat{U}$
Останов
$DB \leftarrow \varnothing$
$i \leftarrow 1$
$i \leq |\bar{M}_b|$
Генерирование $DB(\pi_b')$
$AS \leftarrow AS - \{\pi_b\}$
$BFS \leftarrow BFS \cup \{\pi_b\}$
$ES \leftarrow \varnothing$
Применение правил исключения для определения ES
$AS' \leftarrow (AS \cup DB) - ES$
$AS \leftarrow \langle AS' \rangle$
$OFS \leftarrow OFS \cup (AS' - AS)$
$\pi_b \leftarrow (AS, S, 1)$
$M_b \{b_1, b_2, \ldots, b_j \mid \pi_b = (b_1, b_2, \ldots, b_j)\}$
$\hat{L}_{AS} \leftarrow \min L(\pi_a)$
$\pi_a \in AS$
$\hat{L}_{OFS} \leftarrow \begin{cases} \min L(\pi_d), & OFS \neq \varnothing \\ \pi_d \in OFS \\ \infty, & OFS = \varnothing \end{cases}$
$\hat{L} \leftarrow \min\{\hat{L}_{AS}, \hat{L}_{OFS}\}$
$\hat{U} \leftarrow \dot{U}$
$\hat{\pi}_u^N \leftarrow \dot{\pi}_u^N$
Изменение RUNTIME
$l \leftarrow (\bar{M}_b, LEX, i)$
$\pi_d \leftarrow \pi_d \circ l$
$M_d \leftarrow M_d \cup |l|$
Вычисление $F(\pi_d)$
$F(\pi_d) = 1$
Вычисление $L(\pi_d)$
$DB' \leftarrow DB \cup |\pi_d|$
$DB \leftarrow \langle DB' \rangle$
$OFS \leftarrow OFS \cup (DB' - DB)$
$(M_d = N) \wedge (L(\pi_d) < \dot{U})$
$\dot{U} \leftarrow L(\pi_d)$
$\dot{\pi}_u^N \leftarrow \pi_d$
$i \leftarrow i + 1$

Рис. 6.7.

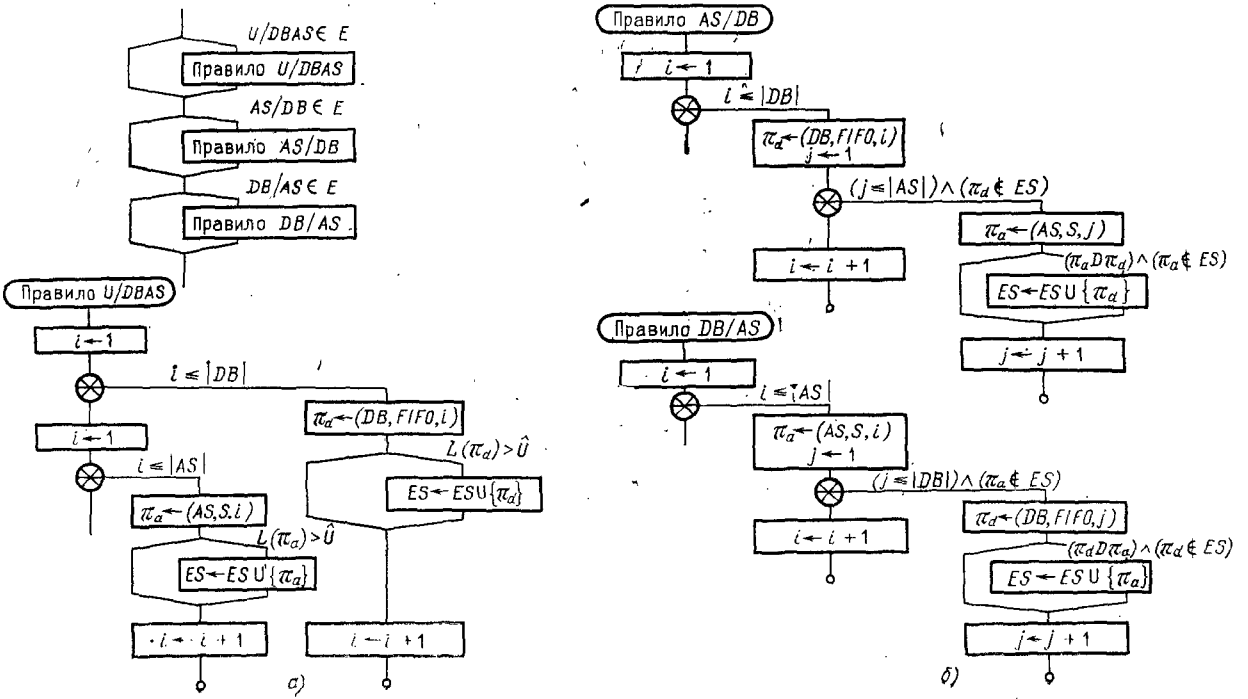

U/DBAS∈ E
Правило U/DBAS
AS/DB∈ E
Правило AS/DB
DB/AS∈ E
Правило DB/AS
Правило U/DBAS
i←1
i ≤ |DB|
i←1
i ≤ |AS|
πa←(AS,S.i)
L(πa)>Û
ES←ES∪{πa}
i←i+1
πa←(DB,FIFO,i)
L(πd)>Û
ES←ES∪{πd}
i←i+1
а)
Правило AS/DB
i←1
i ≤ |DB|
πa←(DB,FIFO,i)
j←1
(j≤|AS|)∧(πa∉ES)
πa←(AS,S,j)
(πaDπd)∧(πa∉ES)
ES←ES∪{πa}
j←j+1
i←i+1
Правило DB/AS
i←1
i ≤ |AS|
πa←(AS,S,i)
j←1
(j≤|DB|)∧(πa∉ES)
πd←(DB,FIFO,j)
(πdDπa)∧(πd∉ES)
ES←ES∪{πa}
j←j+1
i←i+1
б)

Рис. 6.8.

множества активных вершин и множества вершин, вызывающих переполнение, наверняка содержит оптимальное решение.

Лемма 6.1. *(Продолжение объединения активного множества и множества вершин, вызывающих переполнение, содержит оптимальное решение.)*

Пусть $BB = (B_p, S, E, F, D, L, U, BR, RB)$. *Если* π_b *является текущей вершиной ветвления, то существует вершина* $\pi_y^N \in \{[\mathrm{AS}(\pi_b) \cup \mathrm{OFS}(\pi_b)] \circ\}$ *такая, что* $\pi_y^N \in \mathcal{P}$ *и* $f(\pi_y^N, x) = f(\pi^*, x)$.

Доказательство. Применим индукцию на множестве $\mathrm{AS}(\pi_b) \cup \mathrm{OFS}(\pi_b)$ по вершинам с успешным исходом ветвления.

База индукции. Справедливость утверждения для первой вершины ветвления $e = \pi^{\varnothing}$ не вызывает сомнений, поскольку $\mathrm{AS}(e) \cup \mathrm{OFS}(e) = \{e\} \cup \varnothing = \{e\}$ и $\{e \circ\} = \{\pi^{\varnothing} \circ\} = \mathcal{P}_n \supseteq \mathcal{P}$.

Шаг индукции. Пусть гипотеза справедлива для вершины ветвления π_b и всех других вершин, подвергнутых ветвлению раньше. Покажем, что гипотеза справедлива для следующей вершины ветвления π_c. Блок-схема *BB* (рис. 6.7) показывает, что новое множество представляет собой старое множество, плюс новые допустимые потомки вершины ветвления, минус исключенные вершины и вершина ветвления:

$$\mathrm{AS}(\pi_c) \cup \mathrm{OFS}(\pi_c) = [(\mathrm{AS}(\pi_b) \cup \{(\pi_b \circ l) \mid l \in \overline{M}_b, \quad F(\pi_b \circ l) = 1\}) - (\mathrm{ES}(\pi_b) \cup \{\pi_b\})] \cup \mathrm{OFS}(\pi_b). \quad (1)$$

Мы должны показать, что $\{(\mathrm{AS}(\pi_c) \cup \mathrm{OFS}(\pi_c)) \circ\}$ содержит оптимальное решение.

Предположим, что $\pi_y \in \mathrm{AS}(\pi_b) \cup \mathrm{OFS}(\pi_b)$, но $\pi_y \notin \mathrm{AS}(\pi_c) \cup \mathrm{OFS}(\pi_c)$. Из (1) следует, что либо π_y была ранее исключена посредством E, при этом $\pi_y \in \mathrm{ES}(\pi_b)$, либо была подвергнута ветвлению, $\pi_y = \pi_b$.

Если π_y была исключена посредством E, то π_y была исключена посредством правил U/DBAS или DB/AS. Если она была исключена посредством U/DBAS, то $L(\pi_y) > \hat{U}(\pi_b)$, и, следовательно, ни один из членов $\{\pi_y \circ\}$ не является оптимальным решением. Если π_y была исключена посредством DB/AS, то $(\pi_b \circ l) D \pi_y$ для некоторого $l \in \overline{M}_b$. В этом случае π_y следовало бы заменить в объединении множества активных вершин и множества вершин, вызывающих переполнение, на $(\pi_b \circ l)$. Из определения D сле-

дует, что $\{(\pi_b \circ l) \circ\}$ содержит оптимальное решение, если его содержит $\{\pi_y \circ\}$.

Если $\pi_y = \pi_b$, то каждый непосредственный потомок $\pi_b — \pi_b \circ l$ для всех $l \in \bar{M}_b$ — находится в $\mathrm{AS}(\pi_c) \cup \mathrm{OFS}(\pi_c)$, если только он не имеет допустимого продолжения, $F((\pi_b \circ l)) = 0$, или не исключен посредством одного из следующих правил исключения E:

1) U/DBAS, $L((\pi_b \circ l)) > \hat{U}(\pi_b)$;
2) AS/DB, $\pi_a D(\pi_b \circ l)$, $\pi_a \in \mathrm{AS}(\pi_b)$.

Если $F((\pi_b \circ l)) = 0$, то в $\{(\pi_b \circ l) \circ\}$ не содержится оптимального решения, поскольку $\{(\pi_b \circ l) \circ\} \cap \mathcal{P} = \varnothing$. Если $(\pi_b \circ l)$ исключена посредством правила U/DBAS, то стоимость любого полного решения, содержащегося в $\{(\pi_b \circ l) \circ\} \cap \mathcal{P}$, превышает $\hat{U}(\pi_b)$ (стоимость решения $\hat{\pi}_u^N$, дающего верхнюю оценку). Правило AS/DB исключает потомков $(\pi_b \circ l)$ только в том случае, когда существует другая текущая активная вершина $\pi_z \in \mathrm{AS}(\pi_b) \cap \mathrm{AS}(\pi_c)$ такая, что $\pi_z D(\pi_b \circ l)$. Из определения D следует, что $\{\pi_z \circ\}$ содержит оптимальное решение в том случае, когда его содержит $\{(\pi_b \circ l) \circ\}$.

Мы заключаем, что для каждой вершины $\pi_y \in \in [\mathrm{AS}(\pi_b) \cup \mathrm{OFS}(\pi_b)] — [\mathrm{AS}(\pi_c) \cup \mathrm{OFS}(\pi_c)]$ либо π_y не имеет оптимального продолжения, либо существует вершина $\pi_z \in \mathrm{AS}(\pi_c) \cup \mathrm{OFS}(\pi_c)$ такая, что $\pi_z D \pi_y$, т. е. $f(\hat{\pi}_z^N, x) \leqslant \leqslant f(\hat{\pi}_y^N, x)$. Следовательно, $\{[\mathrm{AS}(\pi_c) \cup \mathrm{OFS}(\pi_c)] \circ\}$ содержит оптимальное решение в том случае, когда его содержит $\{[\mathrm{AS}(\pi_b) \cup \mathrm{OFS}(\pi_b)] \circ\}$.

Следствие 6.1. (*Продолжение активного множества содержит оптимальное решение, если требуемый объем памяти не превышает заранее заданного предела.*)

Пусть $BB = (B_p, S, E, F, D, L, U, BR, RB)$. *Если* π_b *является текущей вершиной ветвления и ни одна вершина не оказывается потерянной в результате превышения объема* AS *или* DB, *то существует вершина* $\pi_y^N \in \in \{\mathrm{AS}(\pi_b) \circ\}$ *такая, что* $\pi_y^N \in \mathcal{P}$ *и* $f(\pi_y^N, x) = f(\pi^*, x)$.

Доказательство. Справедливость следствия непосредственно вытекает из леммы 6.1, так как при условии, что объемы массивов AS или DB не превышены, $\mathrm{OFS}(\pi_b) = \varnothing$. Следовательно, $\mathrm{AS}(\pi_b) \cup \mathrm{OFS}(\pi_b) = \mathrm{AS}(\pi_b)$ для каждой вершины ветвления π_b.

Лемма 6.2. (*Когда потребности в ресурсах не превышают заранее установленных границ, то алгоритм завершится нахождением решения, стоимость которого удовлетворяет заданному допуску.*)

Пусть $BB=(B_p,\ S,\ E,\ F,\ D,\ L,\ U,\ BR,\ RB)$. *Если* RUNTIME $\leqslant$ TIMELIMIT, $|\{(\pi_b \circ l) \mid l \in \bar{M}_b, F(\pi_b \circ l)=1\}| \leqslant$ MAXSZDB и $|[\mathrm{AS}(\pi_b) \cup \mathrm{DB}(\pi_b)]-[\mathrm{ES}(\pi_b) \cup \{\pi_b\}]| \leqslant$ MAXSZAS *для всех* $\pi_b \in \mathrm{BFS}(\pi_t)$, *где* π_t *— вершина ветвления в момент окончания, то* BB *заканчивается получением решения* $\hat{\pi}_u^N$ *такого, что*

$$(1-BR)\,\hat{U}(\pi_t) \leqslant f(\pi^*,\ x) \leqslant \hat{U}(\pi_t) = f(\hat{\pi}_u^N,\ x).$$

Доказательство. Поскольку требуемый объем памяти не превышает заданного предела, то $\mathrm{OFS}(\pi_b)=\varnothing$; следовательно, $\hat{L}_{\mathrm{OFS}}(\pi_t)=\infty$. Поэтому $\hat{L}(\pi_t)=\min\{\hat{L}_{\mathrm{AS}}(\pi_t), \hat{L}_{\mathrm{OFS}}(\pi_t)\}=\hat{L}_{\mathrm{AS}}(\pi_t)=\min\{L(\pi_a) \mid \pi_a \in \mathrm{AS}(\pi_t)\}$. Функция L определена таким образом, что для всех π_y, $\pi_z \in \mathrm{P}$ она обладает следующими свойствами:

1) $L(\pi_z) \geqslant L(\pi_y)$, если π_z является потомком π_y;
2) $L(\pi_y^N)=f(\pi_y^N,\ x)$ для каждого полного решения π_y^N.

Из этого сразу следует, что $\hat{L}(\pi_t)$ является нижней оценкой для всех полных решений, содержащихся в множестве $\{\mathrm{AS}(\pi_t) \circ\}$. Поскольку, на основании следствия 6.1, множество $\{\mathrm{AS}(\pi_t) \circ\}$ обязательно содержит оптимальное решение, то $\hat{L}(\pi_t)$ является нижней оценкой оптимального решения, т. е. $\hat{L}(\pi_t) \leqslant f(\pi^*,\ x)$.

Ветвление закончится при $\pi_b=\pi_t$ в том и только в том случае, когда справедливо одно из следующих условий:

1) $(\hat{U}(\pi_b)-\hat{L}(\pi_b))/\hat{U}(\pi_b) \leqslant BR$;
2) RUNTIME $>$ TIMELIMIT;
3) $|M_b|=|N|$.

Предположим, что окончание наступает в результате выполнения условия 1). Тогда мы можем записать

$$\hat{U}(\pi_t)-\hat{L}(\pi_t) \leqslant BR \cdot \hat{U}(\pi_t),$$

или

$$(1-BR)\,\hat{U}(\pi_t) \leqslant \hat{L}(\pi_t).$$

Мы уже видели, что $\hat{L}(\pi_t) \leqslant f(\pi^*,\ x)$, и знаем, что $f(\pi^*,\ x) \leqslant f(\hat{\pi}_u^N,\ x)=\hat{U}(\pi_t)$. Следовательно,

$$(1-BR)\,\hat{U}(\pi_t) \leqslant f(\pi^*,\ x) \leqslant \hat{U}(\pi_t) = f(\hat{\pi}_u^N,\ x),$$

где $\hat{\pi}_u^N$ — решение, дающее верхнюю оценку в момент окончания алгоритма.

Теперь предположим, что окончание наступило в результате выполнения условия 2). Но это противоречит на-

шему предположению о том, что требуемые ресурсы не превышают заранее установленных пределов.

Наконец, предположим, что работа алгоритма *BB* заканчивается потому, что $|M_b| = |N|$. Это происходит в том случае, когда следующая вершина ветвления уже является полным решением. Покажем, что это может произойти, если только условие 1), $(\hat{U}(\pi_t) - \hat{L}(\pi_t))/\hat{U}(\pi_t) \leqslant BR$, также выполняется. Мы должны рассмотреть четыре случая, соответствующие разным правилам ветвления: LLB, FIFO, LIFO и DF/LLB.

Когда $S = \text{LLB}$, вершина ветвления π_b такова, что $\hat{L}(\pi_b) = \min\{L(\pi_a) \mid \pi_a \in \text{AS}(\pi_b)\}$. Поскольку $|M_b| = |N|$, то π_b должно быть полным решением со стоимостью $f(\pi_b, x) = L(\pi_b) = \hat{L}(\pi_b)$. Поскольку $\text{AS}(\pi_b)$ содержит оптимальное решение (следствие 6.1), мы заключаем, что $\pi_t = \pi_b$ должно быть оптимальным. Это означает, что $\hat{U}(\pi_t) = f(\pi_t, x)$ и $\hat{U}(\pi_t) - \hat{L}(\pi_t) = f(\pi_t, x) - f(\pi_t, x) = 0$. Следовательно, $(\hat{U}(\pi_t) - \hat{L}(\pi_t))/\hat{U}(\pi_t) = 0 \leqslant BR$.

Когда $S = \text{FIFO}$, все активные вершины размера k выбираются для ветвления раньше, чем любая активная вершина размера $k+1$. Следовательно, если $|M_b| = |N| = |\{1, 2, \ldots, n\}|$, то в $\text{AS}(\pi_b)$ отсутствуют активные вершины размера $n-1$ или менее. Поскольку $\text{AS}(\pi_b)$ должно содержать оптимальное решение и все вершины в $\text{AS}(\pi_b)$ являются полными, $\hat{L}(\pi_b) = \min\{L(\pi_a) \mid \pi_a \in \text{AS}(\pi_b)\} = f(\pi^*, x)$. Как и в предыдущем случае, при $\pi_b = \pi_t$ $(\hat{U}(\pi_t) - \hat{L}(\pi_t))/\hat{U}(\pi_t) = 0 \leqslant BR$.

Когда $S = \text{FIFO}$ или $S = \text{DF/LLB}$, все активные вершины размера $k < n$ выбираются для ветвления раньше, чем любая активная вершина размера n. Следовательно, если вершина π_b такова, что $|M_b| = |N|$, то $\text{AS}(\pi_b)$ должно состоять из полных решений. Так же, как и в предыдущем случае, это означает, что когда $\pi_t = \pi_b$, то $(\hat{U}(\pi_t) - \hat{L}(\pi_t))/\hat{U}(\pi_t) = 0 \leqslant BR$.

Лемма 6.3. *(Алгоритм ветвей и границ всегда дает нижнюю оценку стоимости оптимального решения; если известно некоторое полное решение, то он также дает величину отклонения стоимости этого решения от оптимальной.)*

Пусть $BB = (B_p, S, E, F, D, L, U, BR, RB)$. *Если* π_t *является вершиной ветвления в момент окончания работы BB, то стоимость оптимального решения* π^* *удовлетворяет неравенству* $\hat{L}(\pi_t) \leqslant f(\pi^*, x) \leqslant \hat{U}(\pi_t)$. *Если*

известно полное решение, то $\hat{\pi}_u^N$ *— полное решение с минимальной стоимостью и* $\hat{U}(\pi_t) = f(\hat{\pi}_u^N, x)$. *Выражение* $(\hat{U}(\pi_t) - \hat{L}(\pi_t))/\hat{U}(\pi_t)$ *определяет границу разности между стоимостью этого известного решения и стоимостью оптимального решения.*

Доказательство. Из определения функции нижней оценки L следует, что для каждого допустимого частичного решения $\pi_y \in \mathrm{P}$ имеет место $L(\pi_y) \leqslant L(\pi_y^N) = f(\pi_y^N, x)$ для всех продолжений $\pi_y^N \in \{\pi_y \circ\} \cap \mathcal{P}$. Из леммы 6.1 также известно, что $\mathrm{AS}(\pi_t) \cup \mathrm{OFS}(\pi_t)$ обязательно содержит оптимальное решение. Следовательно, имеем

$$\hat{L}(\pi_t) = \min\{\hat{L}_{\mathrm{AS}}(\pi_t), \quad \hat{L}_{\mathrm{OFS}}(\pi_t)\} = \\ = \min\{L(\pi_y) \mid \pi_y \in \mathrm{AS}(\pi_t) \subset \mathrm{OFS}(\pi_t)\}.$$

Используя предыдущее неравенство, можем записать

$$\hat{L}(\pi_t) \leqslant \\ \leqslant \min\{f(\pi_y^N, x) \mid \pi_y^N \in \{\pi_y \circ\} \cap \mathcal{P}, \ \pi_y \in \mathrm{AS}(\pi_t) \cup \mathrm{OFS}(\pi_t)\} = \\ = f(\pi^*, x).$$

Мы установили, что $\hat{L}(\pi_t) \leqslant f(\pi^*, x)$.

Покажем теперь, что $f(\pi^*, x) \leqslant \hat{U}(\pi_t)$ и $\hat{\pi}_u^N$ является известным решением с минимальной стоимостью $f(\hat{\pi}_u^N, x) = \hat{U}(\pi_t)$. Для начальной вершины ветвления $\pi_b = e$ в качестве $\hat{U}(e)$ принимается значение U, т. е. стоимость наилучшего решения π_u^N, известного к началу выполнения *BB* (U является одним из девяти параметров алгоритма). Если $U \neq \infty$, то в качестве $\hat{\pi}_u^N$ задается известное решение π_u^N; в противном случае $\hat{\pi}_u^N$ остается неопределенным до тех пор, пока не будет порождено некоторое решение. Если в течение текущего шага ветвления порождаются улучшенные решения, то лучшее из них — пусть это будет $\overset{*}{\pi}{}_u^N$ со стоимостью $f(\overset{*}{\pi}{}_u^N, x) = \overset{*}{U}$ — запоминается (см. рис. 6.7). Перед началом следующего шага ветвления $\hat{\pi}_u^N$ и $\hat{U}$ заменяются на $\overset{*}{\pi}{}_u^N$ и $\overset{*}{U}$, которые соответствуют лучшим из известных решений. Как $\hat{\pi}_u^N$, так и $\hat{U}$ остаются неизменными в течение всего шага ветвления, что облегчает анализ, проводимый в следующем пункте. Мы заключаем, что $f(\pi^*, x) \leqslant \hat{U}(\pi_t)$, и если полное решение известно в начале шага ветвления π_t, то оно есть $\hat{\pi}_u^N$, а его стоимость $f(\hat{\pi}_u^N, x) = \hat{U}(\pi_t)$.

Теперь мы можем записать

$$\hat{L}(\pi_t) \leqslant f(\pi^*, x) \leqslant \hat{U}(\pi_t) = f(\hat{\pi}_u^N, x),$$

и, следовательно,

$$\frac{f(\hat{\pi}_u^N, x) - f(\pi^*, x)}{f(\hat{\pi}_u^N, x)} \leqslant \frac{\hat{U}(\pi_t) - \hat{L}(\pi_t)}{\hat{U}(\pi_t)},$$

где $(\hat{U}(\pi_t) - \hat{L}(\pi_t))/\hat{U}(\pi_t)$ является границей разности между стоимостью лучшего из известных и стоимостью оптимального решений.

6.2.6. Теоретическое сравнение потребностей в вычислительных ресурсах. В этом п. рассматриваются относительные потребности алгоритмов ветвей и границ в ресурсах как функции параметров E, F, D, L, U и BR при фиксированных B_p, S и RB. Будем считать, что требуемые ресурсы (время и память) никогда не превышают предельного значения RB. Это соответствует заданию $RB \triangleq \infty$. В этом случае лемма 6.2 гарантирует, что выполнение алгоритма завершится нахождением решения, точность которого находится в пределах, определяемых величиной BR.

Сравнительный анализ, представленный в этом пункте, справедлив для любых мер оценки объема вычислений, лишь бы они являлись монотонными неубывающими функциями от размера множеств $\mathrm{BFST} \triangleq \mathrm{BFS}(\pi_t) \cup \{\pi_t\}$, $\mathrm{AS}(\pi_b)$ и $\mathrm{DB}(\pi_b)$ для всех $\pi_b \in \mathrm{BFST}$, где π_t — вершина ветвления в момент окончания работы алгоритма. Например, $|\cup \mathrm{DB}(\pi_b)|_{\pi_b \in \mathrm{BFS}(\pi_t)}$ — общее число генерируемых вершин — представляет собой монотонную неубывающую функцию от размера $\mathrm{DB}(\pi_b)$ для всех $\pi_b \in \mathrm{BFST}$. Эта величина может быть использована в качестве грубого показателя относительной потребности во времени. Максимальный размер множества активных вершин $\max\limits_{\pi_b \in \mathrm{BFST}} |\mathrm{AS}(\pi_b)|$ является монотонной неубывающей функцией от BFST и $\mathrm{AS}(\pi_b)$, $\pi_b \in \mathrm{BFST}$, и может служить мерой оценки требуемого максимального объема памяти.

Нас интересует проблема выбора параметров E_i, F_i, D_i, L_i, U_i и BR_i для $BB_1 = (B_p, S_1, E_1, F_1, D_1, L_1, U_1, BR_1, RB_1 = \infty)$ и $BB_2 = (B_p, S_2, E_2, F_2, D_2, L_2, U_2, BR_2, RB_2 = \infty)$, для которых выполняется следующая совокупность соотношений для всех $\pi_b \in \mathrm{BFST}_1$:

1) $\mathrm{BFS}_1(\pi_b) \subseteq \mathrm{BFS}_2(\pi_b)$;
2) $\mathrm{AS}_1(\pi_b) \subseteq \mathrm{AS}_2(\pi_b)$;
3) $\mathrm{DB}_1(\pi_b) \subseteq \mathrm{DB}_2(\pi_b)$;

4) $\hat{U}_1(\pi_b) \leqslant \hat{U}_2(\pi_b)$;

5) $\hat{L}_1(\pi_b) \geqslant \hat{L}_2(\pi_b)$.

В дополнение к этому будет также проверяться справедливость следующих соотношений для всех $\pi_b \in \mathrm{BFST}_1$:

6) если $\pi_y \in \mathrm{BFS}_2(\pi_b) - \mathrm{BFS}_1(\pi_b)$, то $L_1(\pi_y) > \hat{U}_1(\pi_b)$ либо $\{\pi_y\circ\} \cap \mathscr{P} = \varnothing$;

7) если $\pi_y \in \mathrm{AS}_2(\pi_b) - \mathrm{AS}_1(\pi_b)$, то $L_1(\pi_y) > \hat{U}_1(\pi_b)$ либо $\{\pi_y\circ\} \cap \mathscr{P} = \varnothing$.

Полный набор соотношений 1)—7) обозначается символом (*).

В процессе сравнительного анализа параметров используются следующие обозначения.

1. $E_2 \subseteq E_1$, если множество правил исключения E_2 является подмножеством (не обязательно собственным) множества правил E_1.

2. $F_1 \leqslant F_2$, если для каждой частичной перестановки (вершины) π_y такой, что $\{\pi_y\circ\} \subseteq \mathscr{P}_n$, $F_1(\pi_y) = 1$ только в том случае, когда $F_2(\pi_y) = 1$.

3. $D_2 \subseteq D_1$, если для каждой пары частичных решений π_y и π_z $\pi_y D_2 \pi_z$ только в том случае, когда $\pi_y D_1 \pi_z$.

4. $L_2 \leqslant L_1$, если для всех частичных решений π_y, для которых $F_1(\pi_y) = 1$, $L_2(\pi_y) \leqslant L_1(\pi_y)$.

5. $U_1 \leqslant U_2$, если стоимость начального решения, дающего верхнюю оценку для BB_1, меньше или равна стоимости начального решения, дающего верхнюю оценку для BB_2.

6. $BR_1 \geqslant BR_2$, если допуск на допустимое решение для BB_1 не меньше допуска на допустимое решение для BB_2.

Теперь сравним относительные потребности в вычислительных ресурсах для BB_1 и BB_2 при условии, что параметры E, F, D, L, U и BR связаны только что описанным способом.

Первая теорема показывает, что потребности в ресурсах не увеличатся, если к E добавить правило исключения U/DBAS. Мы приводим полное доказательство, поскольку оно хорошо иллюстрирует технику, применяемую для доказательства других теорем в этом параграфе.

Теорема 6.1. *(Потребности в вычислительных ресурсах не увеличиваются (но могут уменьшиться), если исключаются вершины, у которых нижняя оценка стоимости превышает верхнюю оценку стоимости.)*

Пусть $BB_1 = (B_p, S, E_1, F, D, L, U, BR, RB = \infty)$ *и* $BB_2 = (B_p, S, E_2, F, D, L, U, BR, RB = \infty)$. *Если* $E_1 = E_2 \cup \{U/\mathrm{DBAS}\}$, *то соотношения* (*) *выполняются для всех* $\pi_b \in \mathrm{BFST}_1$.

Доказательство. Теорема доказывается методом индукции по последовательным вершинам ветвления вплоть до окончания BB_1.

База индукции. Соотношения (*), очевидно, выполняются для начальной вершины ветвления $\pi_b = e = \pi^{\varnothing}$. В частности, справедливы следующие соотношения:

1) $\mathrm{BFS}_1(e) = \mathrm{BFS}_2(e) = \varnothing$;
2) $\mathrm{AS}_1(e) = \mathrm{AS}_2(e) = \{e\}$;
3) $\mathrm{DB}_1 = \mathrm{DB}_2 = \{l \mid l \in \overline{\varnothing} = N \text{ и } F(l) = 1\}$;
4) $\hat{U}_1(e) = \hat{U}_2(e) = U$;
5) $\hat{L}_1(e) = \hat{L}_2(e) = L(e)$;
6) $\mathrm{BFS}_2(e) - \mathrm{BFS}_1(e) = \varnothing$;
7) $\mathrm{AS}_2(e) - \mathrm{AS}_1(e) = \varnothing$.

Шаг индукции. В целях нумерации последовательных вершин ветвления положим, что π_k обозначает k-ю вершину ветвления в ситуации, когда полное дерево допустимых частичных решений P просмотрено с использованием правила S и вершины не исключены посредством какого бы то ни было правила исключения. Когда в BB_1 или BB_2 добавляются правила исключения, то π_k не будет подвергнута ветвлению, если она или кто-либо из ее предков были исключены из активного множества. Если π_k не входит в число активных, то она пропускается, все переменные остаются неизменными, и в качестве кандидата для ветвления выбирается следующая вершина π_{k+1}. Поскольку $S_1 = S_2 = S$ и $L_1 = L_2 = L$, последовательности ветвлений в BB_1 и BB_2 являются подпоследовательностями только что описанного порядка. Эти подпоследовательности отыскиваются путем рассмотрения только тех вершин, которые действительно подвергаются ветвлению. Докажем (*) методом индукции по вершинам π_k, являющимся последовательными кандидатами для ветвления, вплоть до окончания BB_1 (при $\pi_k = \pi_t$). В качестве предположения индукции примем, что система (*) справедлива для $\pi_b = \pi_k$. Покажем, что (*) справедлива для вершины π_{k+1}, являющейся следующим возможным кандидатом на ветвление. Правильность шага индукции проверяется путем рассмотрения следующих возможных случаев:

1) $\pi_k \in \mathrm{BFST}_1 \cap \mathrm{BFST}_2$;
2) $\pi_k \in \mathrm{BFST}_1 \cap \overline{\mathrm{BFST}_2}$;
3) $\pi_k \in \overline{\mathrm{BFST}_1} \cap \mathrm{BFST}_2$;
4) $\pi_k \in \overline{\mathrm{BFST}_1} \cap \overline{\mathrm{BFST}_2}$.

Пусть выполняется случай 1), $\pi_k \in \mathrm{BFST}_1 \cap \mathrm{BFST}_2$. Тогда π_k подвергается ветвлению в BB_1 и BB_2. В этом

случае мы можем записать $BFS_1(\pi_{k+1}) = BFS_1(\pi_k) \cup \{\pi_k\} \subseteq BFS_2(\pi_k) \cup \{\pi_k\} = BFS_2(\pi_{k+1})$, поскольку $BFS_1(\pi_k) \subseteq BFS_2(\pi_k)$ согласно предположению индукции (*1). Следовательно, мы доказали (*1) для π_{k+1}.

Для того чтобы показать справедливость соотношения (*2), $AS_1(\pi_{k+1}) \subseteq AS_2(\pi_{k+1})$, заметим, что, поскольку потребности в памяти никогда не выходят за пределы граничных значений RB, можно записать $AS_i(\pi_{k+1}) = (AS_i(\pi_k) \cup DB_i(\pi_k)) - ES_i(\pi_k)$ для $i = 1, 2$. Согласно предположению индукции (*2) $AS_1(\pi_k) \subseteq AS_2(\pi_k)$, а из (*3) $DB_1(\pi_k) \subseteq DB_2(\pi_k)$. Следовательно, $AS_1(\pi_{k+1}) \subseteq AS_2(\pi_{k+1})$, если $ES_1(\pi_k) \supseteq ES_2(\pi_k) \cap (AS_1(\pi_k) \cup DB_1(\pi_k))$. Мы покажем, что если $\pi_z \in ES_2(\pi_k) \cap (AS_1(\pi_k) \cup DB_1(\pi_k))$, то одновременно $\pi_z \in ES_1(\pi_k)$. Это доказывает (*2) для π_{k+1}. Поскольку $\pi_z \in ES_2(\pi_k) \cap (AS_1(\pi_k) \cup DB_1(\pi_k))$, вершина π_z должна была быть удалена в BB_2 посредством некоторого правила из набора $E_2 \subseteq E$. Рассмотрим теперь все возможные случаи и покажем, что π_z будет удалена также и в BB_1.

Если π_z была удалена в BB_2 посредством U/DBAS, то $E_1 = E_2$ и $AS_1(\pi_{k+1}) = AS_2(\pi_{k+1})$, поскольку $BB_1 = BB_2$. Из этого следует, что π_z должна быть удалена в BB_1.

Если π_z была удалена в BB_2 посредством правила AS/DB, то $\pi_y D \pi_z$ для некоторого $\pi_y \in AS_2(\pi_k)$. Поскольку согласно предположению индукции $AS_1(\pi_k) \subseteq AS_2(\pi_k)$, мы можем записать $AS_2(\pi_k) = (AS_2(\pi_k) - AS_1(\pi_k)) \cup AS_1(\pi_k)$. Если $\pi_y \in AS_1(\pi_k)$, то $\pi_y D \pi_z$, и π_z удаляется в BB_1 посредством правила AS/DB. Если $\pi_y \in (AS_2(\pi_k) - AS_1(\pi_k))$, то, согласно предположению индукции (*7), $L(\pi_y) = L_1(\pi_y) > \hat{U}(\pi_k)$. Поскольку $L(\pi_y) \leqslant L(\pi_z)$ при $\pi_y D \pi_z$ (условие согласованности D), мы видим, что $L(\pi_z) \geqslant L(\pi_y) > \hat{U}(\pi_k)$, и π_k должна быть удалена в BB_1 посредством правила U/DBAS.

Если π_z была удалена в BB_2 посредством правила DB/AS, то $\pi_d D \pi_z$ для некоторого $\pi_d \in DB_2(\pi_k)$. Но поскольку $DB_1(\pi_k) = DB_2(\pi_k)$ при $F_1 = F_2 = F$, то π_z должна быть удалена также и в BB_1.

Мы заключаем, что $\pi_y \in ES_2(\pi_k) \cap (AS_1(\pi_k) \cup DB_1(\pi_k))$ только в том случае, когда $\pi_y \in ES_1(\pi_k)$. Следовательно, условие (*2) выполняется для π_{k+1}.

Поскольку мы уже доказали, что $AS_1(\pi_{k+1}) \subseteq AS_2(\pi_{k+1})$, получаем, что $\pi_{k+1} \in BFST_1$ только в том случае, когда $\pi_{k+1} \in BFST_2$. Если $\pi_{k+1} \in BFST_1$, то $DB_1(\pi_{k+1}) = DB_2(\pi_{k+1})$, так как $F_1 = F_2 = F$. Если $\pi_{k+1} \notin BFST_1$, то $DB_1(\pi_{k+1}) = \varnothing \subseteq DB_2(\pi_{k+1})$. Это доказывает (*3) для π_{k+1}.

Для того чтобы доказать (*4), $\hat{U}_1(\pi_{k+1}) \leqslant \hat{U}_2(\pi_{k+1})$, используем предположение индукции $\hat{U}_1(\pi_k) \leqslant \hat{U}_2(\pi_k)$ и заметим, что множество непосредственных потомков вершины π_k составляют только те решения, которые были порождены между шагами π_k и π_{k+1}. Если $\pi_k \circ l$, $l \in \overline{M}_k$, является допустимым полным решением, то $F(\pi_k \circ l) = 1$ и $\pi_k \circ l \in \mathrm{DB}_1(\pi_k) = \mathrm{DB}_2(\pi_k)$. Любое решение, улучшающее верхнюю оценку, найденное на этапе π_k, найдется как в BB_1, так и в BB_2, и, следовательно, $\hat{U}_1(\pi_{k+1}) \leqslant \leqslant \hat{U}_1(\pi_k)$, $\hat{U}_2(\pi_{k+1}) \leqslant \hat{U}_2(\pi_k)$ и $\hat{U}_1(\pi_{k+1}) \leqslant \hat{U}_2(\pi_{k+1})$. Соотношение (*4) доказано для π_{k+1}.

Поскольку мы уже доказали (*3), $\mathrm{AS}_1(\pi_{k+1}) \subseteq \mathrm{AS}_2(\pi_{k+1})$, получаем, что $\hat{L}_{\mathrm{AS}_1}(\pi_{k+1}) \triangleq \min\{L(\pi_a) \mid \pi_a \in \mathrm{AS}_1(\pi_{k+1})\} \geqslant \geqslant \min\{L(\pi_a) \mid \pi_a \in \mathrm{AS}_2(\pi_{k+1})\} \triangleq \hat{L}_{\mathrm{AS}_2}(\pi_{k+1})$.

Так как потребности в памяти никогда не превышают предельного объема, то $\mathrm{OFS}_1(\pi_{k+1}) = \mathrm{OFS}_2(\pi_{k+1}) = \varnothing$ и $\hat{L}_{\mathrm{OFS}_1}(\pi_{k+1}) = \hat{L}_{\mathrm{OFS}_2}(\pi_{k+1}) = \infty$. Следовательно, $\hat{L}_1(\pi_{k+1}) \triangleq \triangleq \min\{\hat{L}_{\mathrm{AS}_1}(\pi_{k+1}), \hat{L}_{\mathrm{OFS}_1}(\pi_{k+1})\} \geqslant \min\{\hat{L}_{\mathrm{AS}_2}(\pi_{k+1}), \hat{L}_{\mathrm{OFS}_2}(\pi_{k+1})\} \triangleq \hat{L}_2(\pi_{k+1})$, что доказывает (*5) для π_{k+1}.

По предположению индукции (*6) для π_k, если $\pi_y \in \in \mathrm{BFS}_2(\pi_k) - \mathrm{BFS}_1(\pi_k)$, то $L_1(\pi_y) > \hat{U}_1(\pi_k)$ либо $\{\pi_y \circ\} \cap \cap \mathscr{P} = \varnothing$. Но поскольку $\pi_k \in \mathrm{BFST}_1 \cap \mathrm{BFST}_2$, получаем $\mathrm{BFS}_2(\pi_{k+1}) - \mathrm{BFS}_1(\pi_{k+1}) = \mathrm{BFS}_2(\pi_k) - \mathrm{BFS}_1(\pi_k)$. Аналогично, $\hat{U}_1(\pi_{k+1}) \leqslant \hat{U}_1(\pi_k)$. Следовательно, если $\pi_y \in \in \mathrm{BFS}_2(\pi_{k+1}) - \mathrm{BFS}_1(\pi_{k+1})$, то $L_1(\pi_y) > \hat{U}_1(\pi_{k+1})$ либо $\{\pi_y \circ\} \cap \mathscr{P} = \varnothing$.

Наконец, покажем, что выполняется (*7), а именно, если $\pi_y \in \mathrm{AS}_2(\pi_{k+1}) - \mathrm{AS}_1(\pi_{k+1})$, то $L_1(\pi_y) > \hat{U}_1(\pi_{k+1})$ либо $\{\pi_y \circ\} \cap \mathscr{P} = \varnothing$. Поскольку предельный объем ресурсов не превышается, мы можем записать $\mathrm{AS}_2(\pi_{k+1}) - \mathrm{AS}_1(\pi_{k+1}) = [(\mathrm{AS}_2(\pi_k) \cup \mathrm{DB}_2(\pi_k)) - \mathrm{ES}_2(\pi_k)] - [(\mathrm{AS}_1(\pi_k) \cup \cup \mathrm{DB}_1(\pi_k) - \mathrm{ES}_1(\pi_k)]$. Правая часть может быть переписана в следующем виде:

$$
\begin{aligned}
&= [(\mathrm{AS}_2 \cup \mathrm{DB}_2) \cap \overline{\mathrm{ES}}_2] \cap \overline{[(\mathrm{AS}_1 \cup \mathrm{DB}_1) \cap \overline{\mathrm{ES}}_1]} = \\
&= [(\mathrm{AS}_2 \cup \mathrm{DB}_2) \cap \overline{\mathrm{ES}}_2] \cap [(\overline{\mathrm{AS}}_1 \cap \overline{\mathrm{DB}}_1) \cup \mathrm{ES}_1] = \\
&= [(\mathrm{AS}_2 \cap \overline{\mathrm{ES}}_2) \cup (\mathrm{DB}_2 \cap \overline{\mathrm{ES}}_2)] \cap [(\overline{\mathrm{AS}}_1 \cap \overline{\mathrm{DB}}_1) \cup \mathrm{ES}_1] = \\
&= (\mathrm{AS}_2 \cap \overline{\mathrm{ES}}_2) \cap [(\overline{\mathrm{AS}}_1 \cap \overline{\mathrm{DB}}_1) \cup \mathrm{ES}_1] \cup \\
&\qquad \cup [(\mathrm{DB}_2 \cap \overline{\mathrm{ES}}_2) \cap (\overline{\mathrm{AS}}_1 \cap \overline{\mathrm{DB}}_1) \cup \mathrm{ES}_1] = \\
&= [(\mathrm{AS}_2 \cap \overline{\mathrm{AS}}_1 \cap \overline{\mathrm{ES}}_2 \cap \overline{\mathrm{DB}}_1) \cup (\mathrm{AS}_2 \cap \overline{\mathrm{ES}}_2 \cap \mathrm{ES}_1)] \cup
\end{aligned}
$$

$$\cup [(DB_2 \cap \overline{DB_1} \cap \overline{ES_2} \cap \overline{AS_1}) \cup (DB_2 \cap \overline{ES_2} \cap ES_1)] =$$
$$= [(AS_2 — AS_1) \cap (\overline{ES_2} \cap \overline{DB_1})] \cup [(AS_2 \cup DB_2) \cap (\overline{ES_2} \cap ES_1)] \cup$$
$$\cup [(DB_2 — DB_1) \cap (\overline{ES_2} \cap \overline{AS_1})].$$

Для $\pi_y \in AS_2(\pi_{k+1}) — AS_1(\pi_{k+1})$ покажем, что $L_1(\pi_y) > \hat{U}(\pi_{k+1})$ либо $\{\pi_y \circ\} \cap \mathcal{P} = \varnothing$ в каждом из трех случаев, определяемых последним выражением.

Во-первых, если $\pi_y \in [(AS_2(\pi_k) — AS_1(\pi_k)) \cap (\overline{ES_2}(\pi_k) \cap \cap \overline{DB_1}(\pi_k))]$, то $\pi_y \in (AS_2(\pi_k) — AS_1(\pi_k))$, и на основании предположения индукции (*7) получаем $L(\pi_y) = L_1(\pi_y) \geqslant \geqslant \hat{U}_1(\pi_k) \geqslant \hat{U}_1(\pi_{k+1})$ либо $\{\pi_y \circ\} \cap \mathcal{P} = \varnothing$.

Во-вторых, если $\pi_y \in [(AS_2(\pi_k) \cup DB_2(\pi_k)) \cap (\overline{ES_2}(\pi_k) \cap \cap ES_1(\pi_k))]$, то вершина π_y была удалена на шаге ветвления π_k в BB_1, но не в BB_2. Это могло бы случиться только тогда, когда π_y была удалена посредством правила $U/DBAS \in E_1$. В этом случае $L_1(\pi_y) = L(\pi_y) > \hat{U}_1(\pi_k) \geqslant \geqslant \hat{U}_1(\pi_{k+1})$, что доказывает (*7) для π_{k+1}.

Наконец, если $\pi_y \in [(DB_2(\pi_k) — DB_1(\pi_k)) \cap (\overline{ES_2}(\pi_k) \cap \cap \overline{AS_1}(\pi_k))]$, то $\pi_y \in (DB_2(\pi_k) — DB_1(\pi_k))$. Но $(DB_2(\pi_k) — — DB_1(\pi_k)) = \varnothing$, так как $F_1 = F_2 = F$, и, следовательно, последний случай оказывается невозможным. Тем самым мы завершили доказательство соотношений (*1)—(*7) для случая 1).

Теперь предположим, что выполняется случай 2), $\pi_k \in BFST_1 \cap \overline{BFST_2}$. Вершина π_k будет подвергнута ветвлению в BB_i в том и только в том случае, когда $\pi_k \in \in AS_i(\pi_k)$. Согласно предположению индукции (*2) $AS_1(\pi_k) \subseteq AS_2(\pi_k)$, и, следовательно, $\pi_k \in AS_2(\pi_k)$, если $\pi_k \in AS_1(\pi_k)$. Из этого следует, что $\pi_k \in BFST_1$ только тогда, когда $\pi_k \in BFST_2$. Следовательно, $BFST_1 \cap \overline{BFST_2} = \varnothing$, а это доказывает, что случай 2) невозможен.

Предположим, что выполняется случай (3), $\pi_k \in \overline{BFST_1} \cap \cap BFST_2$. Тогда $\pi_k \in AS_2(\pi_k) — AS_1(\pi_k)$. Мы снова должны доказать справедливость соотношений (*1)—(*7) для π_{k+1}. Поскольку $\pi_k \in \overline{BFST_1} \cap BFST_2$, то мы можем записать $BFS_1(\pi_{k+1}) = BFS_1(\pi_k)$ и $BFS_2(\pi_{k+1}) = BFS_2(\pi_k) \cup \{\pi_k\}$. Теперь из (*1) следует, что $BFS_1(\pi_{k+1}) = BFS_1(\pi_k) \subseteq \subseteq BFS_2(\pi_k) \subseteq BFS_2(\pi_k) \cup \{\pi_k\} = BFS_2(\pi_{k+1})$. Это доказывает (*1) для π_{k+1}.

Для того чтобы доказать (*2), $AS_1(\pi_{k+1}) \subseteq AS_2(\pi_{k+1})$, заметим, что $AS_1(\pi_{k+1}) = AS_1(\pi_k)$, так как π_k не подвергалась ветвлению в BB_1. Но $AS_2(\pi_{k+1}) = (AS_2(\pi_k) \cup$

$\cup \mathrm{DB}_2(\pi_k)) - \mathrm{ES}_2(\pi_k)$, так как $\pi_k \in \mathrm{BFST}_2$ и предельный объем памяти не превышается. Поскольку, согласно предположению индукции (*2), $\mathrm{AS}_1(\pi_k) \subseteq \mathrm{AS}_2(\pi_k)$, то $\mathrm{AS}_1(\pi_{k+1}) \subseteq \subseteq \mathrm{AS}_2(\pi_{k+1})$ при условии, что $\mathrm{ES}_2(\pi_k) \cap \mathrm{AS}_1(\pi_k) = \varnothing$. Предположим, что $\mathrm{ES}_2(\pi_k) \cap \mathrm{AS}_1(\pi_k) \neq \varnothing$. Тогда существует некоторая вершина $\pi_z \in \mathrm{ES}_2(\pi_k) \cap \mathrm{AS}_1(\pi_k)$, которая должна была быть удалена в BB_2 посредством какого-нибудь правила из E_2. Нужно рассмотреть три возможных случая для каждого из допустимых правил исключения, входящих в E_2.

Если π_z была исключена посредством правила U/DBAS, то $E_1 = E_2 \cup \{U/\mathrm{DBAS}\} = E_2$, и поэтому BB_1 и BB_2 оказались бы идентичными. Следовательно, $\overline{\mathrm{BFST}_1} \cap \mathrm{BFST}_2 = \varnothing$, что невозможно.

Если π_z была исключена посредством правила AS/DB, то $\pi_y D \pi_z$ для некоторых вершин $\pi_y \in \mathrm{AS}_2(\pi_k)$ и $\pi_z \in \mathrm{DB}_2(\pi_k)$. Следовательно, $\pi_z \notin \mathrm{AS}_2(\pi_k)$ и на основании предположения индукции (*2) получаем $\pi_z \notin \mathrm{AS}_1(\pi_k)$. Полученное противоречие доказывает, что этот случай невозможен.

Наконец, если π_z была исключена посредством правила DB/AS, то $\pi_z \in \mathrm{AS}_2(\pi_k) \cap \mathrm{AS}_1(\pi_k)$ и $\pi_d D \pi_z$ для некоторого $\pi_d \in \mathrm{DB}_2(\pi_k) = \{\pi_k \circ l \mid l \in \overline{M}_k$ и $F(\pi_k \circ l) = 1\}$. Вследствие монотонного неубывания функции нижней оценки L $L(\pi_d) \geqslant L(\pi_k)$, и из согласованности D получаем $L(\pi_d) \leqslant L(\pi_z)$. Но поскольку $\pi_k \in \mathrm{AS}_2(\pi_k) - \mathrm{AS}_1(\pi_k)$, то на основании предположения индукции (*7) получаем, что либо $L_2(\pi_k) = L(\pi_k) > \hat{U}_1(\pi_k)$, либо $\{\pi_k \circ\} \cap \mathcal{P} = \varnothing$. Вторая возможность может быть сразу исключена, поскольку из нее вытекает соотношение $\{\pi_d \circ\} \cap \mathcal{P} \leqslant \{\pi_k \circ\} \cap \cap \mathcal{P} = \varnothing$, что противоречит определению D и, следовательно, $\pi_d D \pi_z$. В случае реализации первой возможности видно, что $L(\pi_z) \geqslant L(\pi_d) \geqslant L(\pi_k) > \hat{U}_1(\pi_k)$. Из этого следует, что $\pi_z \notin \mathrm{AS}_1(\pi_k)$, так как в BB_1 она была бы отброшена посредством правила U/DBAS или правила DB/AS. Это снова приводит к противоречию, и этот случай также является невозможным. Поэтому (*2) выполняется для π_{k+1}.

Доказательство (*3) для π_{k+1} аналогично приведенному ранее для случая 1).

Доказательство (*4), $\hat{U}_1(\pi_{k+1}) \leqslant \hat{U}_2(\pi_{k+1})$, следует из того, что $\hat{U}_1(\pi_{k+1}) = \hat{U}_1(\pi_k)$ и любые новые решения $\pi_d \in \mathrm{DB}(\pi_k)$, которые порождаются в течение шага ветвления π_k в BB_2, должны иметь стоимость $L(\pi_d) \geqslant$ $\geqslant L(\pi_k)$. Согласно предположению индукции (*7) $L(\pi_k) =$

$= L_1(\pi_k) > \hat{U}_1(\pi_k)$ либо $\{\pi_k \circ\} \cap \mathcal{P} = \varnothing$. Следовательно, $\hat{U}_1(\pi_{k+1}) = \hat{U}_1(\pi_k) \leqslant \hat{U}_2(\pi_{k+1})$, что доказывает (*4) для π_{k+1}.

Утверждение (*5), $\hat{L}_1(\pi_{k+1}) \geqslant \hat{L}_2(\pi_{k+1})$, непосредственно вытекает из (*2), $AS_1(\pi_{k+1}) \subseteq AS_2(\pi_{k+1})$, как было доказано в случае 1).

На основании предположения индукции (*7) известно, что поскольку $\pi_k \in AS_2(\pi_k) - AS_1(\pi_k)$, то либо $L(\pi_k) = L_1(\pi_k) > \hat{U}_1(\pi_k)$, либо $\{\pi_k \circ\} \cap \mathcal{P} = \varnothing$. На основании предположения индукции (*6) известно, что если $\pi_y \in BFS_2(\pi_k) - BFS_1(\pi_k)$, то $L_1(\pi_y) > \hat{U}_1(\pi_k)$ либо $\{\pi_y \circ\} \cap \mathcal{P} = \varnothing$. Поскольку $BFS_2(\pi_{k+1}) - BFS_1(\pi_{k+1}) = (BFS_2(\pi_k) \cup \{\pi_k\}) - BFS_1(\pi_k)$ и $\hat{U}_1(\pi_k) = \hat{U}_1(\pi_{k+1})$ при $\pi_k \in \overline{BFST_1}$, можно заключить, что если $\pi_y \in BFS_2(\pi_{k+1}) - BFS_1(\pi_{k+1})$, то $L_1(\pi_y) > \hat{U}_1(\pi_{k+1})$ либо $\{\pi_y \circ\} \cap \mathcal{P} = \varnothing$. Это доказывает (*6) для π_{k+1}.

Нам осталось доказать (*7), а именно, если $\pi_y \in AS_2(\pi_{k+1}) - AS_1(\pi_{k+1})$, то либо $L_1(\pi_y) > \hat{U}_1(\pi_{k+1})$, либо $\{\pi_y \circ\} \cap \mathcal{P} = \varnothing$. Поскольку $\pi_k \notin BFST_1$, то $DB_1(\pi_k) = ES_1(\pi_k) = \varnothing$, и, следовательно, $AS_1(\pi_{k+1}) = AS_1(\pi_k)$. Для $AS_2(\pi_{k+1})$ имеем $AS_2(\pi_{k+1}) = (AS_2(\pi_k) \cup DB_2(\pi_k)) - ES_2(\pi_k)$, так как предельный объем памяти не превышается. Следовательно,

$$
\begin{aligned}
&AS_2(\pi_{k+1}) - AS_1(\pi_{k+1}) = \\
&\quad = [(AS_2(\pi_k) \cup DB_2(\pi_k)) - ES_2(\pi_k)] - AS_1(\pi_k) = \\
&\quad\quad = [(AS_2 \cup DB_2) \cap \overline{ES_2}] \cap \overline{AS_1} = \\
&\quad\quad = [(AS_2 \cap \overline{AS_1}) \cup (DB_2 \cap \overline{AS_1})] \cap \overline{ES_2} = \\
&\quad\quad\quad = [(AS_2(\pi_k) - AS_1(\pi_k)) \cup DB_2(\pi_k)] \cap \overline{ES_2}(\pi_k).
\end{aligned}
$$

Если $\pi_y \in AS_2(\pi_{k+1}) - AS_1(\pi_{k+1})$, то либо $\pi_y \in AS_2(\pi_k) - AS_1(\pi_k)$, либо $\pi_y \in DB_2(\pi_k)$. В первом случае предположение индукции (*7) для π_k гласит, что $L_1(\pi_y) > \hat{U}_1(\pi_k) = \hat{U}_1(\pi_{k+1})$ либо $\{\pi_y \circ\} \cap \mathcal{P} = \varnothing$. Во втором случае $\pi_y \in DB_2(\pi_k)$, где $\pi_k \in AS_2(\pi_k) - AS_1(\pi_k)$. Поскольку π_y является потомком π_k, заключаем, что $L(\pi_y) = L_1(\pi_y) \geqslant L_1(\pi_k) > \hat{U}_1(\pi_k)$ либо $\{\pi_y \circ\} \cap \mathcal{P} = \varnothing$.

Наконец, *предположим, что выполняется случай* 4), $\pi_k \in \overline{BFST_1} \cap \overline{BFST_2}$. В этом случае π_k не подвергается ветвлению ни в BB_1, ни в BB_2, и все параметры остаются неизменными до шага ветвления π_{k+1}. Индукция осуществляется тривиальным образом.

Следующая теорема подтверждает, что исключение частичных решений, не имеющих допустимых продолжений, не увеличит потребности в вычислительных ресурсах.

Теорема 6.2. *(Потребности в вычислительных ресурсах не увеличатся, если для исключения бесперспективных частичных решений используется более сильная характеристическая функция.)*

Пусть $BB_1 = (B_p, S, E, F_1, D, L, U, BR, RB)$ *и* $BB_2 = (B_p, S, E, F_2, D, L, U, BR, RB)$. *Если* $F_1 \leqslant F_2$, *то совокупность соотношений* (*) *выполняется для всех* $\pi_b \in \mathrm{BFST}_1$.

Доказательство. Формальное доказательство теоремы аналогично доказательству предыдущей теоремы. Заметим, что если $F_1(\pi_y) = 0$, то из определения F_1, соотношения $\{\pi_y \circ\} \cap \mathcal{P} = \varnothing$ и определения отношения доминирования D следует, что π_y не может быть использована для доминирования любого другого частичного решения. Если π_y порождена в BB_2 и $F_2(\pi_y) = 1$, то π_y не имеет допустимого продолжения и не может вытеснить другие частичные решения. Соотношения (*) справедливы для всех $\pi_b \in \mathrm{BFST}_1$, так как все частичные решения π_y в BB_2, для которых $F_1(\pi_y) = 0$, исключены в BB_1 без изменения других параметров.

Хотя предыдущие теоремы подтверждают наши интуитивные представления о полезности введения правила исключения U/DBAS и усиления характеристической функции, тем не менее можно построить противоречащий нашей интуиции пример, касающийся усиления отношения доминирования. Зная это, мы заключаем, что потребности в вычислительных ресурсах алгоритма (B_p, S, E, F, D, L, U, BR, RB) не обязательно являются монотонными невозрастающими функциями от отношения доминирования D. Мы приведем контрпример для некоторого набора параметров, но легко можно найти контрпримеры и для других наборов.

Теорема 6.3. *(Потребности в вычислительных ресурсах могут возрастать при использовании более сильного отношения доминирования.)*

Пусть $BB_1 = (B_p, S, E, F, D_1, L, U, BR, RB)$ *и* $BB_2 = (B_p, S, E, F, D_2, L, U, BR, RB)$. *Если* $D_2 \subseteq D_1$, *то соотношения* (*) *не обязательно выполняются для всех* $\pi_b \in \mathrm{BFST}_1$.

Доказательство. Построим пример, демонстрирующий, что соотношения (*) не обязательно являются справедливыми для всех $\pi_b \in \mathrm{BFST}_1$. Полное дерево поиска для этого примера приведено на рис. 6.9. Пространство решений $\mathcal{P} = \mathcal{P}_1$, и, следовательно, $F(\pi_y) = 1$ для всех π_y.

Нижняя оценка стоимости $L(\pi_y)$, сопоставленная каждой вершине π_y, записывается слева над соответствующей вершиной. Для упрощения записи вместо обозначения

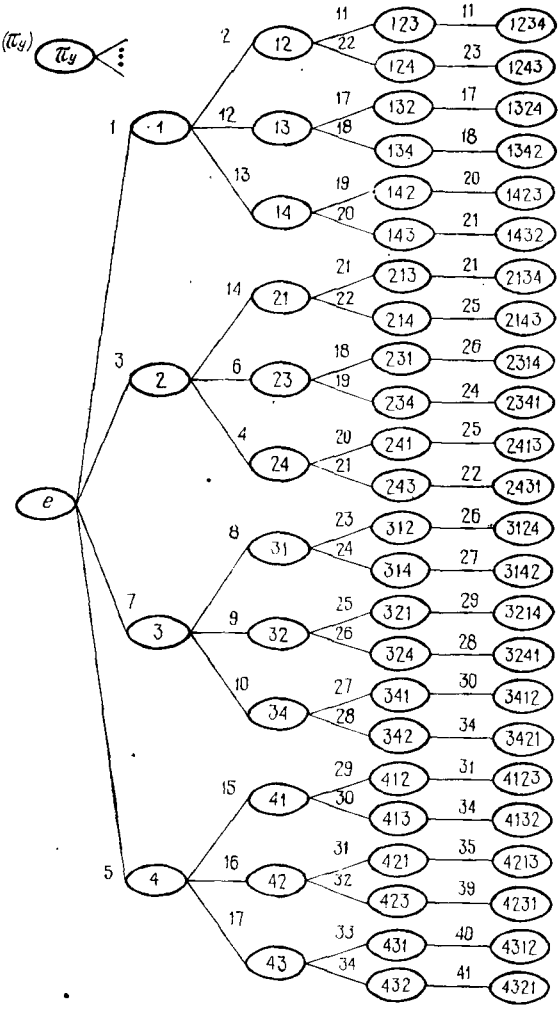


Рис. 6.9.

$\pi_a \triangleq (a_1, a_2, \ldots, a_j)$ используется $a_1, a_2, \ldots, a_j$, а члены отношения доминирования D_i, $\pi_y D_i \pi_z$ записываются в виде пары (π_y, π_z). Теперь положим $BB_1 = (B_p$, LLB, {DB/AS}, 1, D_1, L, ∞, 0, $\infty)$ и $BB_2 = (B_p$, LLB, {DB/AS}, 1, D_2, L,

∞, 0, ∞), где $D_2 = \{(24, 4), (23, 3)\} \cup \{(\pi_y, \pi_z) \mid L(\pi_y) < L(\pi_z), M_y = \{1, 2, 3, 4\}\}$ и $D_1 = \{(12, 2)\}$. Запись $U = \infty$ означает, что в начальный момент работы алгоритма не известно ни одного решения. Поскольку $BR = 0$, то алгоритм завершается при $\hat{L}(\pi_b) = \hat{U}(\pi_b)$.

Шаги BB_1 и BB_2 описываются в терминах упорядоченных множеств AS, DB и ES, при этом запись (A, R, j) означает j-й элемент множества A, упорядоченного с помощью правила R. Шаги алгоритмов BB_1 и BB_2 имеют вид как на таблицах (с. 282).

Заметим, что $\mathrm{BFST}_1 \not\subseteq \mathrm{BFST}_2$ и соотношения (*) не выполняются для всех $\pi_b \in \mathrm{BFST}_2$. В этом примере

$$\left| \bigcup_{\pi_b \in \mathrm{BFST}_1} \mathrm{DB}_1(\pi_b) \right| = 22 > \left| \bigcup_{\pi_b \in \mathrm{BFST}_2} \mathrm{DB}_2(\pi_b) \right| = 17$$

и

$$\max_{\pi_b \in \mathrm{BFST}_1} |\mathrm{AS}_1(\pi_b)| = |\mathrm{AS}_1(123)| = 13 > \max_{\pi_b \in \mathrm{BFST}_2} |\mathrm{AS}_2(\pi_b)| = |\mathrm{AS}_2(123)| = 9.$$

Следующие теоремы показывают, каким образом изменяются потребности в вычислительных ресурсах при улучшении функции нижней оценки и начального решения, дающего верхнюю оценку.

Теорема 6.4. *(При использовании правил выбора* FIFO *или* LIFO *потребности в вычислительных ресурсах не увеличиваются при использовании более точной функции нижней оценки и улучшенного начального решения, дающего верхнюю оценку.)*

Пусть $BB_1 = (B_p, S, E, F, D, L_1, U_1, BR, RB = \infty)$ *и* $BB_2 = (B_p, S, E, F, D, L_2, U_2, BR, RB = \infty)$. *Если* $S = \mathrm{FIFO}$ *или* $S = \mathrm{LIFO}$, *а* $L_1 \geqslant L_2$ *и* $U_1 \leqslant U_2$, *то соотношения* (*) *выполняются для всех* $\pi_b \in \mathrm{BFST}_1$.

Доказательство. Формальное доказательство этой теоремы приводится методом индукции так же, как в теореме 6.1. Заметим, что поскольку $S = \mathrm{FIFO}$ или $S = \mathrm{LIFO}$, то порядок ветвления одинаков для BB_1 и BB_2, так как он не зависит от функции нижней оценки. Для доказательства теоремы следует рассмотреть два различных случая: (1) $U/\mathrm{DBAS} \notin E$ и (2) $U/\mathrm{DBAS} \in E$. В первом случае $\mathrm{BFS}_1(\pi_b) = \mathrm{BFS}_2(\pi_b)$ и $\mathrm{AS}_1(\pi_b) = \mathrm{AS}_2(\pi_b)$ для всех $\pi_b \in \mathrm{BFST}_1$. В обоих случаях можно показать, что соотношения (*) выполняются для всех $\pi_b \in \mathrm{BFST}_1$. Алгоритм BB_1 завершится раньше, чем BB_2, если для последней

BB_1

π_b	($AS_1(\pi_b)$, LLB, j)	($DB_1(\pi_b)$, LEX, k)	($ES_1(\pi_b)$, FIFO, l)	$\hat{L}_1(\pi_b)$	$\hat{U}_1(\pi_b)$
e	(e)	(1, 2, 3, 4)	ø	0	∞
1	(1, 2, 4, 3)	(12, 13, 14)	(2)	1	∞
12	(12, 4, 3, 13, 14)	(123, 124)	ø	2	∞
4	(4, 3, 123, 13, 14, 124)	(41, 42, 43)	ø	5	∞
3	(3, 123, 13, 14, 41, 42, 43, 124)	(31, 32, 34)	ø	7	∞
31	(31, 32, 34, 123, 13, 14, 41, 42, 43, 124)	(312, 314)	ø	8	∞
32	(32, 34, 123, 13, 14, 41, 42, 43, 124, 312, 314)	(321, 324)	ø	9	∞
34	(34, 123, 13, 14, 41, 42, 43, 124, 312, 314, 321, 324)	(341, 342)	ø	10	∞
123	(123, 13, 14, 41, 42, 43, 124, 312, 314, 321, 324, 341, 342)	(1234)	(13, 14, 41, 42, 124, 312, 314, 321, 324, 341, 342)	11	∞
1234	(1234) $\pi^* = 1234$			11	11

BB_2

π_b	($AS_2(\pi_b)$, LLB, j)	($DB_2(\pi_b)$, LEX, k)	($ES_1(\pi_b)$, FIFO, l)	$\hat{L}_1(\pi_b)$	$\hat{U}_1(\pi_b)$
e	(e)	(1, 2, 3, 4)	ø	0	∞
1	(1, 2, 3, 4)	(12, 13, 14)	ø	1	∞
12	(12, 2, 4, 3, 13, 14)	(123, 124)	ø	2	∞
2	(2, 4, 3, 123, 13, 14, 124)	(21, 23, 24)	(3, 4)	3	∞
24	(24, 23, 123, 13, 14, 21, 124)	(241, 243)	ø	4	∞
23	(23, 123, 13, 14, 21, 241, 243, 124)	(231, 234)	ø	6	∞
123	(123, 13, 14, 21, 231, 234, 241, 243, 124)	1234	(13, 14, 21, 231, 34, 241, 43, 124)	11	∞
1234	(1234) $\pi^* = 1234$			11	11

вершины ветвления π_t в BB_1 выполняется $(\hat{U}_1(\pi_t) - L_1(\hat{\pi}_t))/\hat{U}_1(\pi_t) \leqslant BR < (\hat{U}_2(\pi_t) - \hat{L}_2(\pi_t))/\hat{U}_2(\pi_t)$.

Следующая теорема показывает, что, когда S=LLB или S=DF/LLB, при улучшении функции нижней оценки может произойти возрастание требований к вычислительным ресурсам. Такая ситуация возможна, однако в большинстве случаев все же следует ожидать уменьшения потребностей.

Теорема 6.5. *(При* S=LLB *или* S=DF/LLB *потребности в вычислительных ресурсах могут увеличиваться при использовании более точной функции нижней оценки.)*

Пусть $BB_1 = (B_p, S, E, F, D, L_1, U, BR, RB)$ *и* $BB_2 = (B_p, S, E, F, D, L_2, U, BR, RB)$. *Если* S=LLB *или* S = DF/LLB *и при этом* $L_1 \geqslant L_2$, *то соотношения* (*) *не обязательно выполняются для всех* $\pi_b \in \mathrm{BFST}_1$.

Доказательство. Рассмотрим пример, показывающий, что соотношения (*) не обязательно выполняются для всех $\pi_b \in \mathrm{BFST}_1$. Функции нижних оценок L_1 и L_2 определяются деревьями частичного перебора, изображенными на рис. 6.10. Мы исследуем случай, когда S=DF/LLB, но эта же задача может быть использована как контрпример и для S=LLB.

Пусть $BB_1 = (B_p, \mathrm{DF/LLB_{LIFO}}, \varnothing, 1, \varnothing, L_1, \infty, 0, \infty)$ и $BB_2 = (B_p, \mathrm{DF/LLB_{LIFO}}, \varnothing, 1, \varnothing, L_2, \infty, 0, \infty)$. Поскольку $BR = 0$ и $RB = \infty$, то алгоритмы завершатся получением оптимального решения, т. е. такого, для которого $\hat{L}_i(\pi_b) = \hat{U}_i(\pi_b)$. В алгоритмах BB_1 и BB_2 выполняются следующие шаги (см. таблицы на с. 285).

Мы видим, что $\mathrm{BFST}_1 \not\subseteq \mathrm{BFST}_2$ и соотношения (*) не выполняются для всех $\pi_b \in \mathrm{BFST}_1$. В частности, заметим, что

$$\left| \bigcup_{\pi_b \in \mathrm{BFST}_1} \mathrm{DB}_1(\pi_b) \right| = 19 > \left| \bigcup_{\pi_b \in \mathrm{BFST}_2} \mathrm{DB}_2(\pi_b) \right| = 10$$

и

$$\max_{\pi_b \in \mathrm{BFST}_1} |\mathrm{AS}_1(\pi_b)| = |\mathrm{AS}_1(124)| = 9 > \max_{\pi_b \in \mathrm{BFST}_2} |\mathrm{AS}_2(\pi_b)| = |\mathrm{AS}_2(123)| = 7.$$

Теорема 6.6. *(При* S=LLB *или* S=DF/LLB *потребности в вычислительных ресурсах не возрастут, если использовать лучшее начальное решение, дающее верхнюю оценку.)*

Заданы $BB_1 = (B_b, S, E, F, D, L, U_1, BR, RB{=}\infty)$ *и* $BB_2 = (B_b, S, E, F, D, L, U_2, BR, RB{=}\infty)$. *Если* S=LLB

или S=DF/LLB *и при этом* $U_1 \leqslant U_2$, *то соотношения* (*) *выполняются для всех* $\pi_b \in \mathrm{BFST}_1$.

Доказательство. Пусть π_k представляет собой k-ю вершину ветвления в ситуации, когда перебор полного дерева допустимых частичных решений осуществляется с

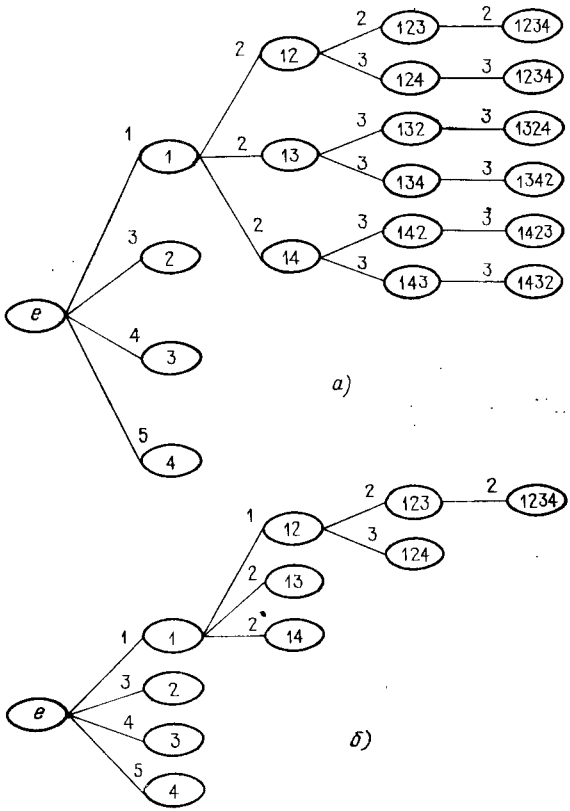


Рис. 6.10.

помощью правила S и функции нижней оценки L и при этом вершины не удаляются посредством какого бы то ни было правила. Поскольку $S=S_1=S_2$ и $L=L_1=L_2$, то последовательности ветвлений в BB_1 и BB_2 представляют собой подпоследовательности первоначально установленного порядка. Рассмотрим два различных случая: (1) $U/\mathrm{DBAS} \notin E$ и (2) $U/\mathrm{DBAS} \in E$. Доказательство соотношений (*) для каждого из этих случаев проводится так же, как доказательство теоремы 6.1. В первом случае мы фактически пока-

BB_1

π_b	$(AS_1(\pi_b)$, DF/LLB_{LIFO}, $i)$	$(DB_1(\pi_b)$, LEX, $k)$	$(ES_1(\pi_b)$, FIFO, $l)$	$\hat{L}_1(\pi_b)$	$\hat{U}_1(\pi_b)$
e	(e)	(1, 2, 3, 4)	ø	0	∞
1	(1, 2, 3, 4)	(12, 13, 14)	ø	1	∞
14	(14, 13, 12, 2, 3, 4)	(142, 143)	ø	2	∞
143	(143, 142, 13, 12, 2, 3, 4)	(1432)	ø	2	∞
142	(1432, 142, 13, 12, 2, 3, 4)	(1423)	ø	2	3
13	(1423, 1432, 13, 12, 2, 3, 4)	(132, 134)	ø	2	3
134	(134, 132, 1423, 1432, 12, 2, 3, 4)	(1342)	ø	2	3
132	(1342, 132, 1423, 1432, 12, 2, 3, 4)	(1324)	ø	2	3
12	(1324, 1342 1423, 1432, 12, 2, 3, 4)	(12[illegible], 124)	ø	2	3
124	(124, 123, 1324, 1342, 1423, 14[illegible]2, 2, 3, 4)	(1243)	ø	2	3
123	(1243, 123, 1324, 1342, 1423, 1432, 2, 3, 4)	(1234)	ø	2	3
1234	(1234, 1243, 1324, 1423, 1432, 2, 3, 4) $\pi^* = 1234$		ø	2	2

BB_2

π_b	$(AS_2(\pi_b)$, LLB_{LIFO}, $i)$	$(DB_2(\pi_b)$, LEX, $k)$	$(ES_2(\pi_b)$, FIFO, $l)$	$\hat{L}_2(\pi_b)$	$\hat{U}_2(\pi_b)$
e	(e)	(1, 2, 3, 4)	ø	0	∞
1	(1, 2, 3, 4)	(12, 13, 14)	ø	1	∞
12	(12, 14, 13, 2, 3, 4)	(123, 124)	ø	2	∞
123	(123, 14, 13, 124, 2, 3, 4)	(1234)	ø	2	∞
1234	(1234, 14, 13, 124, 2, 3, 4) $\pi^* = 1234$			2	2

зываем, что $BFS_1(\pi_b) = BFS_2(\pi_b)$ и $AS_1(\pi_b) = AS_2(\pi_b)$ для всех $\pi_b \in BFST_1$. Снова BB_1 закончит работу раньше, чем BB_2, если для последней вершины ветвления π_t в BB_1 выполняется

$$(\hat{U}_1(\pi_t) - \hat{L}_1(\pi_t))/\hat{U}_1(\pi_t) \leqslant BR < (\hat{U}_2(\pi_t) - \hat{L}_2(\pi_t))/\hat{U}_2(\pi_t).$$

Как и ожидалось, наша последняя теорема подтверждает, что можно уменьшить, но нельзя увеличить потребности в вычислительных ресурсах, если считать допустимыми решения, имеющие меньшую точность.

Теорема 6.7. *(Если увеличить допустимую разницу между стоимостью допустимых решений и стоимостью оптимального решения, то потребности в вычислительных ресурсах не увеличатся.)*

Пусть $BB_1=(B_p, S, E, F, D, L, U, BR_1, RB=\infty)$ *и* $BB_2=(B_p, S, E, F, D, L, U, BR_2, RB=\infty)$. *Если* $BR_1 \geqslant BR_2$, *то соотношения* (*) *выполняются для всех* $\pi_b \in \mathrm{BFST}_1$.

Доказательство. Алгоритмы BB_1 и BB_2 одинаковы во всем, за исключением того, что BB_1 может завершить свою работу раньше, чем BB_2, так как $BR_1 \geqslant BR_2$. В частности, очевидно, что для всех $\pi_b \in \mathrm{BFST}_1$ имеем:

1) $\mathrm{BFS}_1(\pi_b)=\mathrm{BFS}_2(\pi_b)$;
2) $\mathrm{AS}_1(\pi_b)=\mathrm{AS}_2(\pi_b)$;
3) $\mathrm{DB}_1(\pi_b)=\mathrm{DB}_2(\pi_b)$;
4) $\hat{U}_1(\pi_b)=\hat{U}_2(\pi_b)=\hat{U}(\pi_b)$;
5) $\hat{L}_1(\pi_b)=\hat{L}_2(\pi_b)=\hat{L}(\pi_b)$.

Более того, BB_1 закончит работу раньше, чем BB_2, если для последней в BB_1 вершины ветвления π_t выполняется условие $BR_1 \geqslant (\hat{U}(\pi_t)-\hat{L}(\pi_t))/\hat{U}(\pi_t) > BR_2$. Эти соотношения представляют собой частный случай (*).

Все предыдущие теоремы, кроме одной, касаются изменения только одного параметра. Однако эти результаты применимы также и к более общему случаю, когда мы хотим узнать, будут ли потребности в вычислительных ресурсах алгоритма $BB_1=(B_p, S_1, E_1, F_1, D_1, L_1, U_1, BR_1, RB_1)$ меньше или равны потребностям алгоритма $BB_2=(B_p, S_2, E_2, F_2, D_2, L_2, U_2, BR_2, RB_2)$. Например, если $S_1=S_2=S$, $E_1=E_2 \cup \{U/\mathrm{DBAS}\}$, $F_1 \leqslant F_2$, $D_1=D_2=D$, $L_1=L_2=L$, $U_1 \leqslant U_2$, $BR_1 \geqslant BR_2$ и $RB_1=RB_2=\infty$, то потребности в вычислительных ресурсах алгоритма BB_1 не превысят потребности BB_2. Это можно доказать следующим образом:

$$BB_1=(B_p, S, E_1, F_1, D, L, U_1, BR_1, \infty) \leqslant$$
$$\leqslant (B_p, S, E_2, F_1, D, L, U_1, BR_1, \infty) \leqslant \quad \text{(теорема 6.1)}$$
$$\leqslant (B_p, S, E_2, F_2, D, L, U_1, BR_1, \infty) \leqslant \quad \text{(теорема 6.2)}$$
$$\leqslant (B_p, S, E_2, F_2, D, L, U_2, BR_1, \infty) \leqslant \quad \text{(теорема 6.6)}$$
$$\leqslant (B_p, S, E_2, F_2, D, L, U_2, BR_2, \infty) = \quad \text{(теорема 6.7)}$$
$$= BB_2.$$

6.2.7. Комментарии к применению метода ветвей и границ. Для описания класса точных и приближенных алгоритмов

ветвей и границ мы использовали девятку параметров (B_p, S, E, F, D, L, U, BR, RB). Для этого класса алгоритмов было проведено теоретическое сравнение относительных потребностей в вычислительных ресурсах в зависимости от выбора параметров E, F, D, L, U и BR при фиксированных B_p, S и $RB=\infty$. Все результаты подытожены на рис. 6.11.

	Параметр	Соотношения(*)
1.	$E_1 = E_2 \cup$ U/DBAS	Истина (теорема 6.1)
2.	$F_1 \leqslant F_2$	Истина (теорема 6.2)
3.	$D_1 \supseteq D_2$	Ложь (теорема 6.3)
4.	$L_1 \geqslant L_2$; а) $S=$FIFO или $S=$LIFO	Истина (теорема 6.4)
	б) $S=$LLB или $S=$DF/LLB	Ложь (теорема 6.5)
5.	$U_1 \leqslant U_2$	Истина (теоремы 6.4 и 6.6)
6.	$BR_1 \geqslant BR_2$	Истина (теорема 6.7)

Рис. 6.11.

Теорема 6.1 показывает, что мы ничего не теряем при исключении текущих активных и только что порожденных вершин, стоимость которых превышает стоимость решения, дающего верхнюю оценку. Теорема 6.2 показывает, что потребности в вычислительных ресурсах не увеличатся, если будут исключены вершины, не имеющие допустимого продолжения. Потребности в ресурсах могут увеличиться при использовании более строгого отношения доминирования (теорема 6.3), но это происходит обычно в вырожденных случаях. Даже в том случае, когда соотношения (*) выполняются, нет никакой гарантии, что реальное время выполнения алгоритма не увеличится. Например, более сильное отношение доминирования может потребовать большего времени счета. Теоремы 6.4 и 6.5 утверждают, что соотношения (*) остаются в силе, когда более точная функция нижней оценки используется совместно с правилами FIFO и LIFO, но они не обязательно выполняются при использовании правил LLB или DF/LLB. На практике более точная функция нижней оценки обычно сокращает общий объем вычислений, но для того, чтобы выигрыш оказался реальным, это сокращение должно компенсировать увеличение затрат, связанных с вычислением более точной нижней оценки. Теоремы 6.4, 6.6 и 6.7 подтверждают естественное предположение о том, что лучшее начальное решение, дающее верх-

нюю оценку, и более широкие пределы допустимого отклонения от оптимальной точности не могут увеличить объем вычислений. Экспериментальные расчеты для конвейерной задачи показывают, что объем вычислений может быть значительно уменьшен при использовании хорошего решения, дающего верхнюю оценку, и малого ненулевого допустимого отклонения [96] (к чему мы еще вернемся в § 6.4).

Метод ветвей и границ, описанный в этом параграфе, тесно связан с динамическим программированием, но мы отложим обсуждение этой связи до § 6.5. В следующем параграфе дан обзор приближенных алгоритмов и представлены результаты практических расчетов для конвейерной задачи.

6.3. Приближенные алгоритмы

6.3.1. Метод ветвей и границ с ограниченными возвратами. При использовании метода ветвей и границ можно специально сократить объем используемой памяти (путем выбора MAXSZAS или MAXSZDB) при условии, что конечный результат не обязательно должен быть точным. Крайним случаем такого подхода является полный отказ от возвращений; для этого нужно просто продвигаться по ветви вглубь в направлении наименьшей нижней оценки вплоть до получения после n шагов полного решения [4]. Это соответствует MAXSZAS=MAXSZDB=1. Назовем такую процедуру методом ветвей и границ «без возвратов». Она требует вычисления нижних оценок для $O(n^2)$ вершин.

Поскольку размеры множеств вершин заранее ограничены, то лишняя информация просто отбрасывается. Однако обязательно хранится наименьшая нижняя оценка, связанная с отброшенными вершинами. Это позволяет получить нижнюю оценку решения в момент завершения. Такая нижняя оценка совместно с верхней оценкой, вычисленной для лучшего полного решения, используется для обеспечения точности решения.

Другим ограничивающим фактором может стать время вычислений. Если в ситуации, когда превышается отведенное время счета, мы имеем полное решение, то оно, совместно с наименьшей нижней оценкой для вершин из активного множества, определяет границы точности нашего решения.

6.3.2. Эвристические алгоритмы. Мы не можем сделать особых обобщений, касающихся эвристических конструк-

ций. Сталкиваясь с конкретной задачей, человек может обдумывать способ получения решения, обладающего невысокой стоимостью, очень мало заботясь при этом (или совсем не заботясь) о теоретическом обосновании. Например, при решении задачи коммивояжера можно использовать правило, согласно которому из i-го города коммивояжер направляется в ближайший непосещенный город. Этот так называемый метод «иди в ближний» в данном случае оказывается не очень эффективным, но он иллюстрирует произвольную природу такого рода конструкций.

Другой пример связан с эвристическими алгоритмами, рассмотренными в гл. 5. Здесь иногда можно найти оценки характеристик для наихудшего случая, но дальнейшее увеличение объема вычислений не приводит, вообще говоря, к улучшению оценки или к получению оптимального решения.

Мы снова подчеркиваем, однако, что эвристические процедуры при всей своей произвольности и слабости могут быть с успехом использованы в методе ветвей и границ как средство получения начальных решений и верхних оценок [67, 125].

6.3.3. Поиск в локальной окрестности. В последние 15 лет для приближенного решения большого количества комбинаторных задач успешно применялись совершенно различные подходы, из которых недавно возник общий метод. Мы называем этот метод *поиском в локальной окрестности* (LNS—local neighborhood search), хотя некоторые авторы называют его *естественным отбором* [35], *эвристическим программированием* [92, 93], *локальным поиском* [140] и *поиском в окрестности* [126, 145], не считая менее распространенных названий. Для данной задачи о перестановках размерности $n(\mathcal{P}, X, f)$ (см. п. 6.2.2) этот метод можно описать следующим образом [23, 92, 100, 104, 124, 122, 126, 140, 145].

1. Для каждой перестановки $\pi \in \mathcal{P}$ определяется подмножество $\mathcal{P}$, называемое *окрестностью* $N(\pi)$ перестановки π. Когда такая окрестность $N(\pi)$ определена для каждой перестановки $\pi \in \mathcal{P}$, мы говорим, что окрестность N определена для задачи.

2. Задан алгоритм I, который просто выбирает начальное решение $\pi_1 \in \mathcal{P}$. Часто алгоритм I рандомизируется для того, чтобы при последовательных обращениях к нему генерировались различные решения.

3. Задана стратегия Q, предназначенная для поиска $N(\pi)$ и выбора при заданном i некоторой перестановки

$\pi_{i+1} \in N(\pi_i)$ такой, что

$$f(\pi_{i+1}, x) < f(\pi_i, x),$$

если такая точка, улучшающая решение, существует в $N(\pi_i)$. Если для некоторого k $f(\pi_k, x) \leqslant f(\pi, x)$ для всех $\pi \in N(\pi_k)$, то π_k называют *локально оптимальной* в окрестности N.

Идея таких алгоритмов состоит в обследовании некоторой окрестности данного решения и в восприятии всех улучшений по мере их отыскания. Данный процесс продолжается до тех пор, пока все возможности по улучшению решения в данной окрестности не окажутся исчерпанными. Процедура может повторяться для различных начальных решений, после чего выбирается лучшее решение. Этот класс алгоритмов обозначается LNS (I, Q, N), где:

I — метод выбора начальных решений;

Q — стратегия поиска в окрестности и улучшения решения;

N — окрестность, подвергаемая обследованию с целью улучшения решения.

В качестве примера очень успешного применения такой методологии опишем метод, разработанный Чень Лином [100] для решения задачи коммивояжера, которая фактически может рассматриваться как задача составления расписания. Следуя Лину, назовем маршрут коммивояжера λ-оптимальным, если невозможно получить новый маршрут, имеющий меньшую стоимость, путем замены любых λ его дуг на другие λ дуг. Для любого маршрута π необходимо найти окрестность размерности $O(n^\lambda)$, где n — число городов. Лин генерирует λ-оптимальные маршруты путем выбора случайных начальных маршрутов и включения в них первых улучшений, которые отыскиваются при обследовании каждой окрестности. Он установил, что 3-оптимальные маршруты оказываются намного лучше 2-оптимальных в том смысле, что (1) все 3-оптимальные маршруты являются одновременно и 2-оптимальными, (2) средняя длина 3-оптимальных маршрутов значительно меньше, чем средняя длина 2-оптимальных, и (3) вероятность того, что 3-оптимальный маршрут окажется оптимальным, значительно выше, чем для 2-оптимального маршрута. Он также установил, что 4-оптимальные маршруты строить нецелесообразно, так как они требуют значительно большего времени для их построения, при этом вероятность оказаться оптимальными для них ненамного выше. Эта взаимосвязь между размером окрестности и качеством решения является по-

стоянным предметом обсуждения при использовании метода LNS. Лин утверждает, что выбор одного из двух алгоритмов A_1 или A_2 следует производить не только в зависимости от того, какой из них позволяет быстрее найти оптимальный маршрут, но также с учетом относительной длительности их работы. Таким образом, алгоритму A_1 может быть отдано предпочтение перед A_2 из-за того, что A_2 требует непропорционально большого времени, хотя и находит оптимальное решение более часто. Если имеется время для k_i экспериментов с алгоритмом A_i и если A_i находит оптимальный маршрут с вероятностью p_i, то следует выбирать такой алгоритм, для которого вероятность успеха

$$p_i^* = 1 - (1 - p_i)^{k_i}$$

максимальна.

Можно возразить, что выбор наилучшей окрестности может зависеть от размерности задачи n, пользуясь при этом следующими правдоподобными аргументами. Предположим, например, что имеется семейство окрестностей размерности $s = s(n, \lambda)$ и что вероятность нахождения оптимального решения есть $p = p(n, \lambda)$. Мы хотим минимизировать по λ вероятность неудачи

$$1 - p^* = (1 - p)^{t/cs} \cong e^{-tp/cs},$$

т. е., имея в своем распоряжении t секунд машинного времени, мы тратим cs секунд на определение локально оптимального решения, при этом значение p, удовлетворяющее уравнению, оказывается малым. (Заметим, что мы предположили пропорциональность общего времени счета размеру окрестности. Это представляется правильным для 3-оптимальности в задаче коммивояжера, но в общем случае неверно. Справедливость данного предположения существенно зависит от того, сколько шагов должно быть сделано от начальной точки до локального оптимума.) Таким образом, величина λ, которая максимизирует окончательную вероятность успеха, приблизительно равна величине, которая максимизирует отношение p/s, а она зависит, вообще говоря, от n. Лин эмпирически нашел, что для 3-оптимальности $p \cong 2^{-n/10}$. Считая разного рода догадки предметом дискуссии, положим $p(n, \lambda) = e^{-an/\lambda}$. Тогда, если $s = n^\lambda$, то оптимальное значение λ равно $(an/\ln n)^{1/2}$. Аналогичные рассуждения остаются в силе, если вместо вероятности нахождения оптимального решения рассматривать вопрос о близости стоимости полученного решения к оптимальной.

Для конкретной задачи построение LNS (I, Q, N) требует столь же тщательного выбора структуры N, как и ее размерности, причем это относится также к I и Q. В [95] предлагается подход, при котором N «изучается» экспериментально с помощью эволюционного процесса. В работе Крона обсуждается проблема определения «структуры» окрестности и детально описываются два случая, в которых метод LNS применялся для решения задач упорядочения; за подробностями читатель может обратиться к [92]. В качестве последнего комментария к проблеме окрестности отметим, что в [126, 145, 140] поднимается вопрос о том, когда поиски в окрестностях приводят к глобально оптимальным решениям; указано при этом, что точная окрестность для задачи коммивояжера должна иметь размерность $O(n!)$.

Метод выбора начальных решений I может быть либо случайным, либо конструктивным (к таким относятся метод ветвей и границ с ограниченным возвратом или эвристические конструкции), либо представлять собой комбинацию этих методов, получаемую рандомизацией конструктивной процедуры. Обычно целью рандомизации конструктивной процедуры является получение различных начальных решений, которые, отличаясь друг от друга, все имели бы достаточно низкую стоимость. Вопрос о том, является ли локальная оптимизация, осуществляемая из неслучайно выбранной точки, более предпочтительной, чем оптимизация из совершенно случайной точки, имеет большое значение, и ответ на него зависит от конкретной задачи. В задаче коммивояжера применение чисто случайной начальной точки дает, как нам кажется, лучший результат, хотя в одном из методов предлагается начинать с «хорошего» маршрута [138]. В других задачах выбор хорошего начального решения играет существенную роль, поскольку реальный размер окрестности оказывается слишком малым для получения хороших решений из совершенно случайных начальных точек [139, 62, 63]. Применительно к задаче о размещении багажа, как отмечают Хенен и Куртцберг [70], этот вопрос не решен. Интуитивно кажется, что, чем слабее окрестность в конкретной задаче, тем бо́льшую выгоду можно извлечь из правильного выбора начального решения.

В заключение общего обсуждения метода LNS рассмотрим проблему выбора Q, т. е. стратегии поиска в окрестности и усовершенствования решения. Представляется справедливым в общем случае утверждение, что нецелесообразно во что бы то ни стало исследовать всю окрестность с целью нахождения наилучшего усовершенствования (так называе-

мая стратегия наискорейшего спуска). Более разумным представляется способ, при котором первое же найденное улучшение вводится в решение. Это означает, что, как правило, нужно полностью исследовать только последнюю окрестность. Последний поиск, согласно Лину, называется «отметкой перед окончанием работы». Существуют два основных способа упорядочения процесса обследования окрестности: во-первых, окрестность может быть обследована случайным образом, и, во-вторых, окрестность может быть лексикографически упорядочена с помощью фиксированной нумерации. В последнем случае можно вводить лексикографию заново или продолжать ее циклически, начиная от точки, в которой был окончен предыдущий поиск в окрестности. Эти процедуры, согласно Крону [92], называются вариантами A и B соответственно. Возможное преимущество варианта B заключается в том, что он позволяет избегать возможных повторных выполнений неудачных преобразований, которые, скорее всего, останутся неудачными и после улучшения.

Подводя итог, отметим, что построение хорошего алгоритма LNS для конкретной задачи является отчасти искусством, отчасти наукой; изобретательность и глубина понимания задачи обычно с лихвой вознаграждаются. Следующий параграф посвящен применению рассмотренных ранее вычислительных методов к решению конвейерной задачи.

6.4. Алгоритмы решения конвейерной задачи

6.4.1. Предпосылки. Здесь мы приведем ряд результатов, связанных с практическим применением рассмотренных точных и приближенных методов к решению классической конвейерной задачи, состоящей в упорядочении выполнения n работ на двух машинах с целью минимизации среднего времени прохождения *). В этой задаче составления расписания предполагается, что каждая работа J_i, $1 \leqslant j \leqslant n$, состоит не более чем из двух различных заданий, причем каждое из этих заданий может быть выполнено только на одной из двух различных машин P_1 или P_2. Время, требуемое для выполнения работы J_j на машине i, обозначается через τ_{ij}. Для соблюдения характерного для конвейерной задачи условия, заключающегося в том, что все работы выполняются на машинах в одинаковом порядке,

*) Как указано в табл. 1.2, эта задача является NP-полной.

считаем, что каждая работа завершает свое выполнение на машине P_1 раньше, чем она начнет выполняться на машине P_2. Перестановочное расписание составлено, если обе машины выполняют работы в одинаковом порядке (определяемом перестановкой) и при этом отсутствуют бесполезные простои между операциями. Если в качестве меры эффективности расписания используется среднее время завершения работ, то, как хорошо известно, класс перестановочных расписаний обязательно содержит оптимальное решение [28]. Хотя критерий среднего времени завершения не привлек такого большого внимания, как критерий длины расписания (длительность, или максимальное время, завершения) [4, 28, 37, 41, 64, 112], Игнолл и Шрейдж [76] получили практические результаты для точного метода ветвей и границ, а Крон и Штиглиц [93] применили метод LNS к общей задаче с m машинами. Представленные здесь точные и приближенные алгоритмы для случая двух машин демонстрируют практическую эффективность объединения метода LNS и метода ветвей и границ для получения оптимальных решений и решений с гарантированной точностью.

6.4.2. Применение точного метода ветвей и границ. Для применения точных алгоритмов метода ветвей и границ к конвейерной задаче были выбраны следующие параметры.

1. $S=\mathrm{LLB}_{\mathrm{FIFO}}$ (наименьшая нижняя оценка). Выбирается текущая активная вершина с наименьшей нижней оценкой стоимости. В случае, если этому критерию удовлетворяет несколько вершин, то выбирается вершина, порожденная первой.

2. $E=\{U/\mathrm{DBAS},\ U/\mathrm{DB},\ \mathrm{AS/DB}\}$;

а) U/DBAS (доминирующие вершины выявляются среди потомков вершин ветвления и элементов активного множества с помощью верхней оценки стоимости). Исключаются те потомки активных вершин, нижняя оценка стоимости которых превышает текущую верхнюю оценку U;

б) U/DB (доминирующие вершины выявляются среди потомков вершин ветвления с помощью верхней оценки). Исключаются те потомки вершин ветвления, нижняя оценка стоимости которых превышает текущую верхнюю оценку U;

в) AS/DB (проверяется доминирование активных вершин над потомками вершин ветвления). Исключаются те потомки вершин ветвления, которые доминируются элементами активного множества.

3. $F(\pi_y)=1$ для всех π_y, следовательно, $\mathscr{P}=\mathscr{P}_n$.

4. $D = D_{1S}$ (по Игноллу и Шрейджу [76]). Пусть π_y и π_z — две вершины, представляющие частичные расписания для подмножества работ M_y, где $|M_y| = r$; пусть $F_{ij}(\pi)$ — время окончания работы J_j на машине P_i в частичном расписании π, и, наконец, пусть x_k означает k-ю работу в расписании π_x.

В этом случае $(\pi_y, \pi_z) \in D_{1S}$ (т. е. π_y доминирует над π_z) тогда и только тогда, когда

$$F_{2y_r}(\pi_y) \leqslant F_{2z_r}(\pi_z) \quad \text{и} \quad \sum_{i \in M_y} F_{2i}(\pi_y) \leqslant \sum_{i \in M_y} F_{2i}(\pi_z).$$

5. $L = L_{1S}$ (по Игноллу и Шрейджу [76]). Эта функция нижней оценки взята из [76]. В ней использованы следующие величины:

τ_{1i} — время выполнения работы J_i на машине 1;

τ_{2i} — время выполнения работы J_i на машине 2;

$F_{2i} = F_{2i}(\pi_t)$ — время завершения работы J_i на машине 2 в расписании π_t.

Для заданного частичного решения π_y, определенного на множестве работ M_y, где $|M_y| = r$, сумма времен завершения для любого расписания $\pi_t \in \{\pi_y \circ\}$ имеет вид

$$S = \sum_{i \in M_y} F_{2i} + \sum_{i \in \bar{M}_y} F_{2i}, \tag{2}$$

где F_{2i} во второй сумме вычисляется по расписанию π_t. Вторая сумма может быть ограничена снизу следующим образом: если бы каждая работа могла начинаться на машине P_2 сразу после ее завершения на машине P_1, то

$$\sum_{i \in \bar{M}_y} F_{2i} = S_1 = \sum_{k=r+1}^{n} [F_{1t_r} + (n-k+1)\,\tau_{1t_k} + \tau_{2t_k}].$$

Здесь работы в расписании π_t упорядочены по величине t_k. С другой стороны, если каждая работа на машине P_2 должна ожидать только окончания предыдущей работы на P_2, то

$$\sum_{i \in \bar{M}_y} F_{2i}(\pi_t) = S_2 =$$

$$= \sum_{k=r+1}^{n} [\max\{F_{2t_r}(\pi_t),\ F_{1t_r}(\pi_t) + \min_{i \in \bar{M}_y} \tau_{1i}\} + (n-k+1)\,\tau_{2t_k}].$$

Если хотя бы одно из двух предыдущих условий не выполняется, то вторая сумма в (2) может только увеличиться;

следовательно, имеем

$$S \geqslant \sum_{i \in M_y} F_{2i} + \max\{S_1, S_2\}.$$

Более того, S_1 минимизируется при выборе такого π_l, в котором работы τ_{1i_k}, $i_k \in \overline{M}_y$, располагаются в порядке возрастания, что приводит к $S_1 = \hat{S}_1$. S_2 также минимизируется при выборе такого π_l, в котором работы τ_{2i_k}, $i_k \in \overline{M}_y$, располагаются в порядке возрастания, что приводит к $S_2 = \hat{S}_2$. Следовательно,

$$S \geqslant \sum_{i \in M_y} F_{2i} + \max\{\hat{S}_1, \hat{S}_2\}.$$

Заметим, что, поскольку упорядочение τ_{1i_k} и τ_{2i_k} необходимо сделать только один раз, нижняя оценка может быть вычислена для π_y за время $O(n)$.

6. Пусть U обозначает верхнюю оценку, заданную в начале алгоритма. Она отличается от $\hat{U}(\pi_b)$ — верхней оценки, известной в процессе работы алгоритма в тот момент, когда π_b является текущей вершиной ветвления. Здесь имеются две возможности: $U = \infty$, если не задано никакого начального решения; $U < \infty$, если задано некоторое начальное решение.

7. Желая получить точное решение, полагаем $BR = 0$.

8. RB = (TIMELIMIT, MAXSZAS, MAXSZDB). В качестве предельного времени счета TIMELIMIT устанавливались различные значения при разных вариантах входных данных; эти значения указаны вместе с численными результатами. Величина MAXSZDB была взята равной n (чтобы DB никогда не переполнялось), за исключением случая, когда использовался алгоритм без возвратов, в котором MAXSZDB = 1. Наконец, MAXSZAS = 999, и вычисления прекращались, если размер AS превышал MAXSZAS.

Мы используем два следующих точных алгоритма ветвей и границ:

$BB_1 = (B_p,\ \mathrm{LLB_{FIFO}},\ \{\mathrm{AS/DB},\ U/\mathrm{DB}\},\ F = 1,\ D_{1S},\ L_{1S},\ U_1,\ BR = 0,\ RB)$,

$BB_2 = (B_p,\ \mathrm{LLB_{FIFO}},\ \{\mathrm{AS/DB}\},\ F = 1,\ D_{1S},\ L_{1S},\ U_2 = \infty,\ BR = 0,\ RB)$,

где U_1 — верхняя оценка, полученная способом, рассмотренным в § 6.3 и, более детально, в следующем пункте.

6.4.3. Применение метода ветвей и границ без возвратов. Дерево поиска при отсутствии возвратов может быть использовано для получения полного решения, стоимость которого служит затем верхней оценкой для всех остальных значений стоимости. Этот алгоритм формируется путем выбора следующих параметров:

$$S = \mathrm{LLB}_{\mathrm{FIFO}},\quad E = \varnothing,\quad F = 1,\quad D = \varnothing,\quad L = L_{\mathrm{IS}},\quad U = \infty,$$
$$RB = (\mathrm{TIMELIMIT},\ \mathrm{MAXSZAS},\ \mathrm{MAXSZDB}) = (\infty,\ 1,\ 1).$$

В этом случае потомок каждой вершины ветвления, который имеет наименьшую нижнюю оценку, становится следующей вершиной ветвления. Этот алгоритм обозначается BBLB(1); он аналогичен алгоритму, описанному в [4].

6.4.4. Применение метода поиска в локальной окрестности. Здесь мы рассмотрим следующие варианты.

1. а) I=UPRS (равномерный псевдослучайный выбор начального решения — uniform pseudorandom start); начальное решение генерируется с помощью случайной перестановки.

б) I=BBLB(1); результат однократного выполнения алгоритма BBLB(1) принимается за начальное решение.

2. Q=RFI (случайный поиск первого улучшения — random first improvement); окрестность исследуется случайным образом, и реализуется первое же найденное улучшение.

3. а) N=BSI (одиночная вставка назад — backward single insertion) представляет собой множество перестановок (решений), порожденных путем помещения работы, занимавшей i-ю позицию, сразу после работы, занимавшей j-ю позицию ($i<j$). Например, при $i=2$, $j=4$ решение *abcde* преобразуется в *acdbe*.

б) N=FSI (одиночная вставка вперед — forward single insertion) представляет собой множество перестановок, порожденных путем помещения работы, занимавшей i-ю позицию, сразу перед работой, занимавшей j-ю позицию ($i>j$). Например, при $i=4$, $j=2$ решение *abcde* превращается в *adbce*.

в) N=API2 (двойная перемена мест в соседних парах — adjacent pair interchange 2) представляет собой множество перестановок, порожденных следующим образом. В начале меняются местами работы, занимающие позиции i и $i+1$, затем, если задано второе число $j \neq i$, то меняются местами работы, занимающие позиции j и $j+1$. Например, при

$i=2$, $j=4$ решение *abcde* превращается в *acbed*. При $i=2$, $j=3$ *abcde* превращается в *acdbe*.

г) $N=\mathrm{BSI} \cup \mathrm{API2}$.

Все эти окрестности имеют размерность $O(n^2)$.

6.4.5. Числовые данные. Для проверки точных и приближенных алгоритмов использовалось двадцать пять наборов случайных величин, по пяти наборов на каждую задачу; размеры этих наборов $n=5$, 10, 15, 20 и 25. В качестве времен выполнения были приняты независимые выборки целых чисел, имеющих равномерное распределение $P\{\tau_{ij}=r\}=1/10$. Вместо вычисления среднего времени завершения, которое может не быть целым числом, мы использовали эквивалентную величину, равную сумме времен завершения.

6.4.6. Характеристики приближенных алгоритмов. В 25 исследованных задачах для построения субоптимальных решений использовался метод поиска в локальной окрестности и метод ветвей и границ без возвратов. В первой серии экспериментов алгоритм LNS использовался при $I=$ $=\mathrm{PRS}$, $Q=\mathrm{RFI}$ и окрестностях $N=\mathrm{BSI}$, FSI, API2 и $\mathrm{BSI} \cup$ $\cup$ API2.

В табл. 6.1 представлены лучшие решения, полученные с помощью каждой окрестности; там же указано, сколько раз получалось это решение и сколько было использовано различных начальных решений (т. е. в столбце «Количество» указано количество нахождений лучшего решения/количество использованных начальных решений). Решения, отмеченные звездочкой, являются оптимальными, что было доказано с помощью точного метода, а решения, набранные полужирным шрифтом, являются лучшими субоптимальными решениями для этих наборов данных. Задачи, имеющие размерность $n=5$, не включены в таблицу, так как они относительно просты и все окрестности в них часто дают оптимальное решение при любых наборах входных данных. Сравнение качества решений, проведенное на этом небольшом количестве наборов входных данных, показывает, что различия между окрестностями являются незначительными. Ни одна из окрестностей не оказалась лучше, чем другие, при всех наборах данных. В частности, интересно отметить, что $\mathrm{BSI} \cup \mathrm{API2}$ не является стабильно лучшей окрестностью, чем BSI или API2.

Для получения субоптимальных решений использовался также метод ветвей и границ без возвратов BBLB(1). Решения, генерируемые этим эвристическим алгоритмом, оказались, вообще говоря, не столь хорошими, как решения,

Таблица 61

Лучшие решения, полученные с помощью приближенных методов

n	Набор данных	Лучшее известное решение	LNS (PRS, RFI, N)								BBLB (1)	LNS (BBLB (1), RFI, N)			
			N=BSI		N=FSI		N=API2		N=BSIUAPI2			N=BSI		N=FSI	
			Лучшее решение	Количество	Лучшее решение	Количество	Лучшее решение	Количество	Лучшее решение	Количество	Решение	Лучшее решение	Количество	Лучшее решение	Количество
10	1	298*	299	2/10	298*	3/10	304	2/10	299	1/10	305	298*	6/10	298*	5/10
	2	326*	326*	3/10	326*	7/10	326*	5/10	326*	5/10	335	326*	9/10	326*	10/10
	3	375*	375*	4/10	375*	6/10	375*	3/10	375*	3/10	375*	375*	1/1	375*	1/1
	4	340*	340*	1/10	340*	5/10	340*	1/10	340*	5/10	340*	340*	1/1	340*	1/1
	5	291*	291*	1/10	291*	8/10	291*	4/10	291*	3/10	294*	291*	1/10	291*	10/10
15	1	598	605	2/10	599	1/10	602	3/10	598	1/10	667	603	3/10	600	5/10
	2	608*	608*	6/10	608*	8/10	608*	9/10	608*	8/10	609	608*	10/10	608*	10/10
	3	573	576	1/10	573	1/10	574	1/10	577	1/10	600	578	1/10	578	6/10
	4	713*	713*	8/10	713*	7/10	713*	7/10	713*	8/10	713*	713*	1/1	713*	1/1
	5	634*	639	1/10	636	1/10	642	1/10	637	1/10	638	638	1/1	638	1/1
20	1	954	971	1/10	956	1/10	958	1/10	955	1/10	956	956	1/1	954	10/10
	2	1161	1166	1/10	1173	1/10	1177	1/10	1170	2/10	1310	1161	1/10	1170	1/10
	3	957	958	2/10	958	1/10	962	1/10	957	1/10	991	957	1/10	958	4/10
	4	1015	1020	1/10	1018	1/10	1019	1/10	1016	1/10	1041	1015	1/10	1019	5/10
	5	881*	885	1/10	886	1/10	884	1/10	890	2/10	886	885	10/10	885	10/10
50	1	6295	6334	1/5	6327	1/5					7127	6295	1/5	6324	1/5
	2	5046	5151	1/5	5079	1/5					5877	5046	1/5	5068	1/5
	3	6300	6392	1/5	6455	1/5					6990	6300	1/5	6409	1/5
	4	6362	6519	1/5	6460	1/5					6389	6362	1/5	6382	1/5
	5	5726	5927	1/5	5949	1/3					6017	5726	1/5	5805	1/5

*—Звездочка означает оптимальное решение.

полученные методом поиска в локальной окрестности, но для их получения требовалось вдвое меньше времени. Следующий шаг состоял в использовании этих решений в качестве улучшенного начального решения для поиска в локальной окрестности: алгоритм LNS (BBLB (1), RFI, N) был проверен с двумя окрестностями N=BSI и N=FSI. Для небольших задач ($n \leqslant 15$) оказалось, что решения, получаемые на основе этих улучшенных начальных решений, были почти такими же, как и при использовании случайных начальных решений, однако в задачах большей размерности (n=20, 50) использование улучшенного начального решения обычно приводило к улучшению конечного результата. Кроме того, отметим, что генерирование решений методом BSI стабильно дает лучшие результаты, чем FSI, для задачи с 50 работами, но для задач меньшего размера качество этих процедур одинаково. Решения, сведенные в табл. 6.1, представляют собой оптимальные или лучшие из найденных решений для каждого набора данных.

При $n \geqslant 15$ среднее время, требуемое для получения решения с помощью LNS (UPRS, RFI, BSI), было больше, чем сумма времен, требуемых для выполнения BBLB (1) и LNS (BBLB (1), RFI, BSI). Это наводит на мысль, что в хорошем эвристическом алгоритме минимизации среднего времени завершения в конвейерных задачах большой размерности следует вначале получить улучшенное начальное решение с помощью метода ветвей и границ без возвратов, а затем попытаться улучшить решение с помощью поиска в локальной окрестности.

Среднее число соседних решений, которые необходимо проверить (выбрать и сравнить их стоимость со стоимостью базового решения) прежде, чем будет найден локальный оптимум, может быть использовано в качестве меры оценки времени, требуемого для реализации процедуры локального поиска. Оно также дает количество вычислений функции стоимости. В конвейерной задаче с n работами и m машинами каждое вычисление стоимости требует времени, линейно зависящего от произведения mn, поскольку времена завершения каждой из n работ на m машинах должны быть вычислены с помощью рекурсивной процедуры построения перестановочного расписания, выполняющейся до тех пор, пока для всех n работ не будет назначено время начала на каждой машине. Экспериментальные результаты для пяти вариантов задач показывают, что число соседних расписаний, проверяемых алгоритмами LNS (UPRS, RFI, N) и LNS (BBLB (1), RFI, N) для N=BSI, FSI, API2 и

BSI ∪ API2, растет как $n^{2,3} \div n^{2,5}$, а время счета — как $n^{3,3} \div n^{3,7}$. Таким образом, можно предположить, что число проверяемых соседних расписаний растет чуть быстрее, чем размер окрестности ($O(n^2)$). Алгоритм ветвей и границ без возвратов всегда генерирует $n(n-2)/2+1$ вершин, для каждой из которых необходимо вычислять функцию нижней оценки; следовательно, время счета растет приблизительно как n^3.

6.4.7. Характеристики точных алгоритмов и алгоритмов с гарантированной точностью. Метод ветвей и границ, использованный Игноллом и Шрейджем [76] для задачи минимизации среднего времени прохождения в двухмашинной задаче, эквивалентен описанному выше алгоритму BB_2, за исключением того, что они использовали LLB_{LIFO}, разрешая равноценные ситуации с помощью правила «последний пришел — первый обслужен» вместо использованного нами правила «первый пришел — первый обслужен». Как было установлено, эти изменения практически не оказывают влияния на результаты обсуждаемых экспериментов. Из анализа, проведенного в п. 6.2.6, следует, что потребности алгоритма в вычислительных ресурсах могут быть уменьшены, но не могут увеличиться при дополнении набора правил исключения E правилом U/DBAS. Кроме того, было показано, что потребности в вычислительных ресурсах являются монотонными невозрастающими функциями начальной верхней оценки стоимости U. Вообще, приводимый ниже набор экспериментальных данных показывает, что средняя величина потребностей в вычислительных ресурсах одной и той же конвейерной задачи, в которой в качестве критерия оценки выступает среднее время завершения, может быть значительно уменьшена с помощью добавления к E слегка ослабленной версии U/DBAS и использования хорошего субоптимального решения для начальной верхней оценки U.

В наших машинных экспериментах предполагалось, что верхняя оценка стоимости каждой вершины ветвления будет использоваться только для исключения потомков вершин ветвления, но не элементов текущего активного множества. Это новое правило обозначается U/DB. Правило U/DB слабее, чем U/DBAS, так как активная вершина π_a, не исключенная по критерию верхней оценки в момент порождения (т. е. когда $L(\pi_a) \not> U(A(\pi_a))$, где $A(\pi_a)$ обозначает отца вершины π_a), может удовлетворять неравенству $L(\pi_a) > U(\pi_b)$ для некоторой вершины π_b, подвергнутой ветвлению позднее. Если пренебречь повторной проверкой

		BB_2					BB_1			
n	Набор данных	S	Конечная нижняя оценка	Максимальное число активных вершин	Общее число порожденных вершин	RUNTIME (с)	Начальная верхняя оценка	S	Конечная нижняя оценка	Максимальное число активных вершин
10	1	R1	298	381	724	2.787	298	R1	298	50
	2	R1	326	132	178	.256	326	R1	326	12
	3	R1	375	61	83	.060	375	R1	375	4
	4	R1	340	54	69	.045	340	R1	340	2
	5	R1	291	456	1579	8.079	291	R1	291	95
15	1	EN	589	999	1819	17.062	603	ET	589	942
	2	R1	608	237	291	.743	608	R1	608	11
	3	EN	563	999	1716	16.829	578	EN	567	999
	4	R1	713	428	738	3.073	713	R1	713	29
	5	EN	629	999	2242	22.271	638	R1	624	167
20	1	EN	923	999	2494	26.631	956	ET	943	893
	2	EN	1122	999	1495	15.764	1161	EN	1122	999
	3	EN	940	999	1887	22.160	957	EN	944	999
	4	EN	976	999	1650	16.802	1015	EN	980	999
	5	EN	876	999	2462	27.367	885	R1	881	124
50	1	EN	5842	999	1130	10.882	6295	EN	5842	999
	2	EN	4716	999	1050	9.507	5046	EN	4716	999
	3	EN	5795	999	1070	10.516	6300	EN	5795	999
	4	EN	5945	999	1046	10.304	6362	EN	5946	999
	5	EN	5050	999	1136	10.390	5726	EN	5050	999

*) Получено на IBM 360/91. Транслятор с ФОРТРАНА H.

возможного доминирования над активным множеством при улучшении верхней оценки, то может оказаться, что часть бесполезных вершин остается в списке активных и эти вершины могут быть подвергнуты ветвлению. Однако в наших экспериментах начальная верхняя оценка часто оказывалась взятой по оптимальному решению, следовательно, и *U*/DB и *U*/DBAS исключали одни и те же вершины. Кроме того, поскольку мы используем только правило выбора LLB, то множество вершин, подвергнутых ветвлению, не может содержать вершины, имеющей нижнюю оценку, превышающую оптимальную стоимость (лемма 6, [94]). Поэтому использование *U*/DB вместо *U*/DBAS не может изменить сос-

Таблица 6.2

алгоритмов и алгоритмов с заданной точностью

			BB_1: BR=.05 (n=10, 15, 20), .10 (n=50)						
Общее число порожденных вершин	RUNTIME *) (с)	$\frac{\hat{U}-\hat{L}}{\hat{U}}$	Начальная верхняя оценка	S	Конечная нижняя оценка	Максимальное число активных вершин	Общее число порожденных вершин	RUNTIME *) (с)	$\frac{\hat{U}-\hat{L}}{\hat{U}}$
622	.319	.0	298	AB	285	33	75	.046	.045
109	.048	.0	326	AB	320	4	11	.013	.018
64	.030	.0	375	AB	367	1	11	.010	.021
43	.022	.0	340	AB	339	1	11	.011	.003
1366	1.026	.0	291	AB	282	6	11	.011	.031
5726	30.072	.023	603	AB	573	242	553	1.221	.050
168	0.089	.0	608	AB	606	2	16	.016	.033
4263	29.856	.019	578	AB	550	21	30	.026	.048
497	.264		713	AB	709	4	16	.016	.006
4605	3.872	.0	638	AB	607	49	153	.128	.049
16524	40.026	.014	956	AB	909	87	299	.320	.049
1738	15.842	.035	1161	AB	1111	17	21	.024	.043
6656	32.531	.014	957	AB	935	10	21	.023	.023
3244	23.156	.036	1015	AB	968	8	21	.022	.046
5415	4.436	.0	885	AB	858	4	21	.022	.031
1130	10.635	.065	6295	AB	5835	50	51	.114	.073
1050	9.260	.065	5046	AB	4716	50	51	.117	.065
1070	10.017	.080	6300	AB	5788	50	51	.117	.081
1079	9.920	.050	6362	AB	5925	43	51	.106	.069
1136	9.855	.118	5726	EN	5059	999	1136	10.333	.118

тав множества вершин, подвергнутых ветвлению. Теперь рассмотрим

$BB_2 = (B_p,\ \mathrm{LLB_{FIFO}},\ 1,\ \{\mathrm{AS/DB}\},\ D_{1S},\ L_{1S},\ \infty,\ BR,\ RB)$,

$BB_1 = (B_p,\ \mathrm{LLB_{FIFO}},\ 1,\ \{\mathrm{AS/DB},\ U/\mathrm{DB}\},\ D_{1S},\ L_{1S},\ U_1,\ BR,\ RB)$,

где U_1 — стоимость лучшего решения, полученного с помощью LNS(BBLB(1), RFI, BSI), и BR=0. Потребности в вычислительных ресурсах алгоритмов BB_1 и BB_2 отражены в табл. 6.2. Столбец S описывает условия, при которых алгоритм завершает работу для каждого набора данных: R1 означает, что $\hat{L}=\hat{U}$, т. е. доказана оптимальность реше-

ния, давшего верхнюю оценку; *ET* — что время счета равно или превышает TIMELIMIT; *EN* — что число активных вершин равно MAXSZAS. Если алгоритм завершается с превышением предельных ресурсов (*ET* или *EN* в колонке *S*), то в качестве минимальной нижней оценки стоимости заносится наименьшая нижняя оценка, взятая по активным вершинам. Если алгоритм завершается нахождением оптимального решения (R1 в колонке *S*), то последняя нижняя оценка является стоимостью этого оптимального решения π^*.

Как BB_1, так и BB_2 для всех задач, имеющих размер $n=10$, оканчиваются нахождением оптимального решения (R1), но анализ табл. 6.2 показывает, что потребности в вычислительных ресурсах у алгоритма BB_1 значительно меньше, чем у BB_2. Статистика размеров максимального множества активных вершин может быть принята в качестве оценки минимального объема памяти, требуемого для выполнения алгоритма. Средний объем требуемой памяти для задач данного размера n может быть найден путем усреднения этих данных. При $n=10$ средний объем памяти для алгоритма BB_1 составляет только 15% от требуемого объема памяти для алгоритма BB_2, а среднее время счета BB_1 составляет всего лишь 13% от времени счета BB_2. Если в оценку времени включить время получения начального решения, дающего верхнюю оценку (т. е. время, затраченное алгоритмом LNS (BBLB (1), RFI, BSI), считая, что он выполняет по 10 попыток для одного набора данных), то и тогда еще средний выигрыш составляет 76%. Улучшение характеристик по времени и по памяти возникло за счет того, что все потомки вершины π_d, имеющие нижнюю оценку стоимости $L(\pi_d)$, превышающую верхнюю оценку U, немедленно исключались посредством правила U/DB. Эти потомки никогда не стали бы активными, и для них не было необходимости проверять с помощью правила AS/DB, доминируют ли над ними текущие активные вершины.

Когда размер задачи превышает 10, предельные значения объема памяти и времени счета быстро превышаются. Однако поскольку потребности в вычислительных ресурсах алгоритма BB_1 меньше или равны потребностям BB_2, то BB_1, в целом, оканчивает работу, имея решения, более близкие к оптимальным. В некоторых случаях (задачи 15—5 и 20—5) BB_1 находит оптимальное решение, а BB_2 не успевает из-за превышения лимитов по ресурсам. Когда оба алгоритма заканчивают работу из-за превышения времени

счета, то конечная нижняя оценка, достигнутая BB_1, оказывается, по меньшей мере, такой же, а часто и превышает оценку, достигнутую алгоритмом BB_2. Поскольку мы используем правило ветвления LLB, то активная вершина, имеющая наименьшую нижнюю оценку стоимости, всегда является текущей вершиной ветвления. Эта нижняя оценка стоимости, обозначенная $\hat{L}$, совместно с окончательной верхней оценкой $\hat{U}$ может быть использована для оценки точности приближения к оптимальному решению. В табл. 6.2 погрешность $(\hat{U}-\hat{L})/\hat{U}$ меняется от 0 до 2,3% для задач, имеющих размер $n=15$, от 0 до 3,6% для задач, имеющих размер $n=20$, и от 5,0 до 11,8% для задач, имеющих размер $n=50$.

Эти экспериментальные данные показывают, что, прежде чем пытаться найти оптимальное решение с помощью метода ветвей и границ, следует найти хорошее субоптимальное решение. Использование правила исключения U/DB совместно с хорошим начальным решением, дающим верхнюю оценку, существенно уменьшает объем вычислений, и если алгоритм ветвей и границ превышает заданные предельные значения объема памяти и времени счета, то, тем не менее, известное решение, дающее верхнюю оценку, и наименьшая нижняя оценка, взятая по активным вершинам, позволяют с хорошей точностью оценить степень близости полученного решения к оптимальному.

6.4.8. Оценка точности субоптимальных решений. Когда нет необходимости находить оптимальное решение, можно получить еще больший выигрыш в вычислительных ресурсах. В этом пункте мы покажем, что стоимость субоптимального решения, полученного с помощью метода ветвей и границ, превышает оптимальное значение не более чем на заранее заданную величину. Этот метод достижения заданной точности был предложен другими авторами и рассмотрен в [60, 67, 53, 125]. Снова воспользуемся алгоритмом BB_1, но уже при заданном значении BR, $0 \leqslant BR < 1$. Случай, когда $BR=0$, был рассмотрен в предыдущем пункте. В табл. 6.2 даны потребности в вычислительных ресурсах алгоритма BB_1 при $BR=0{,}05$ для $n=10$, 15 и 20 и $BR=0{,}10$ для $n=50$; запись $S=AB$ означает, что алгоритм достиг заданной точности. В большинстве случаев требуемая точность достигается на первом шаге ветвления (общее число порожденных вершин в этом случае равно $n+1$), при этом время счета оказывается ничтожным по сравнению с требуемым для получения при помощи алгоритма ветвей и гра-

ниц оптимального решения. Это является результатом применения хорошего решения, дающего верхнюю оценку, и хорошей границы, вычисляемой с помощью функции нижней оценки L. Когда заданная степень точности устанавливается близкой к 0,01, то среднее количество затрачиваемых вычислительных ресурсов незначительно отличается от того, какое необходимо для получения оптимального решения, поскольку в этом случае оба способа обычно превышают заданные предельные значения вычислительных ресурсов. Однако в тех случаях, когда этого не происходит, достигается экономия.

6.4.9. Выводы относительно результатов экспериментов. Результаты машинных экспериментов для описанных алгоритмов решения конвейерной задачи позволяют сделать следующие выводы.

1. Потребности в вычислительных ресурсах алгоритмов нахождения оптимальных решений методом ветвей и границ могут быть сокращены путем использования хорошего начального решения, дающего верхнюю оценку, совместно с правилом доминирования, учитывающим верхнюю оценку, таким, как U/DB или U/DBAS.

2. Если необходимо получить точное решение задачи, то общий объем вычислений (т. е. затраты на получение начального решения, дающего верхнюю оценку, плюс затраты на выполнение оптимального алгоритма) может быть значительно уменьшен, если вначале найти хорошее субоптимальное решение.

3. Оптимальные алгоритмы ветвей и границ с точки зрения потребностей в вычислительных ресурсах могут быть не хуже приближенных алгоритмов для задач, размер которых не превышает 20.

4. Когда достаточно иметь субоптимальное решение, которое бы гарантированно удовлетворяло заданной границе точности, метод ветвей и границ может быть использован для проверки качества субоптимального решения или для генерирования другого субоптимального решения, обладающего желаемой точностью. Задание точности от 5 до 10% часто приводит к значительной экономии вычислительных ресурсов по сравнению с тем количеством ресурсов, которое требуется для нахождения оптимального решения (отклонение 0%).

5. Метод ветвей и границ без возвратов, используемый для получения начального решения, совместно с методом поиска в локальной окрестности дает хороший эвристический способ решения рассмотренной задачи.

6.5. Взаимосвязь между методом ветвей и границ и динамическим программированием

6.5.1. Предпосылки. Впервые метод динамического программирования для решения задач упорядочения был применен Хелдом и Карпом [68]. Позже Лоулер и Моор [105] нашли более совершенные алгоритмы для нескольких частных случаев. Не так давно Сахни [127] и Горовиц и Сахни [73] предложили сходные алгоритмы, основанные на идеях динамического программирования, для решения задачи упорядочения независимых заданий при различных предположениях относительно процессоров и функции стоимости. Мы не собираемся подробно обсуждать эти работы, а рассмотрим модельную задачу, которая показывает, как метод ветвей и границ связан с динамическим программированием.

6.5.2. Модельная задача. Рассмотрим задачу организации выполнения n независимых заданий на одном процессоре. Время выполнения задания T_i представлено неотрицательным целым числом τ_i. Кроме того, вводится понятие стоимости пребывания, или *потерь*, $w_i(t)$, где $w_i(t)$ является монотонной неубывающей функцией от t, которая обозначает стоимость завершения задания T_i в момент t. Цель заключается в том, чтобы найти расписание, при котором минимизируются суммарные потери по всем n заданиям. Оптимальное решение может быть представлено в виде перестановки π, где $\pi(j)=k$, если T_k должно быть выполнено j-м по счету. Хелд и Карп [68] вывели рекуррентные соотношения, которые позволяют решить задачу за $O(n2^n)$ шагов. Позднее Коглер [88] использовал такой же подход для создания эквивалентного алгоритма ветвей и границ, имеющего сложность $O(n2^n)$.

6.5.3. Частный случай применения метода динамического программирования. Лоулер и Моор [105] первыми установили, что когда

$$w_i(t)=\begin{cases} l_i, & t>d_i, \\ 0, & t\leqslant d_i \end{cases}$$

(т. е. когда с завершением T_i позже директивного срока d_i связана фиксированная величина потерь l_i), задача может быть решена путем оптимального разбиения заданий на два множества. Назовем ее задачей *составления расписания с директивными сроками и фиксированными потерями* (FLDS: fixed-loss due-date scheduling). Задания, входящие в первое множество J, все выполняются с соблюдением директивных сроков, а оставшиеся задания, образующие

множество $\bar{J}$, выполняются с нарушением директивных сроков. Множества J и $\bar{J}$ могут быть определены путем упорядочения заданий по значениям директивных сроков, причем в первую позицию помещается задание с наиболее ранним сроком, а к остальным применяется следующий алгоритм динамического программирования [105].

Пусть $f(j,\ t)$ обозначает минимальную сумму потерь первых j заданий при условии, что задание j завершено не позднее, чем t. Установим следующие граничные условия:

$$f(0,\ t)=0 \qquad \text{при} \quad t\geqslant 0;$$
$$f(j,\ t)=+\infty \quad \text{для} \quad j=1,\ 2,\ \ldots,\ n;\ t<0,$$

а затем решим рекуррентное уравнение

$$f(j,\ t)=\min\begin{cases} f(j,\ t-1), \\ w_j(t)+f(j-1,\ t-\tau_j), \\ l_j+f(j-1,\ t). \end{cases} \quad j=1,\ 2,\ \ldots,\ n;\ t\geqslant 0,$$

Поскольку выполнение всех заданий завершается за время $\omega=\sum_{i=1}^{n}\tau_i$, то это уравнение должно быть решено только для $0\leqslant t\leqslant\omega$. Задания, которым в результате решения уравнения будет выделено нулевое процессорное время, попадают в множество $\bar{J}$ запаздывающих заданий, в то время как остальные задания, образующие множество J, выполняются в порядке неубывания их директивных сроков. Задания, попавшие в $\bar{J}$, выполняются в любом порядке вслед за заданиями из J. Поскольку наше рекуррентное уравнение должно быть решено для n значений j для каждого t от 0 до ω, то решение может быть получено за $O(n\omega)$ шагов.

6.5.4. Представление пространства решений в виде двоичного дерева. Нетрудно понять и сопоставить с методом ветвей и границ некоторую модификацию этого алгоритма динамического программирования. Как и прежде, предположим, что задания перенумерованы в порядке неубывания директивных сроков (т. е. $d_1\leqslant d_2\leqslant\ldots\leqslant d_n$). Тогда все допустимые решения (разбиения) могут быть представлены в виде последовательностей литералов α_1, $\alpha_2,\ \ldots,\ \alpha_n$, где $\alpha=i$ или $\bar{i}$. Если $\alpha_1=i$, то $T_i\in J$ и задание T_i выполняется в указанном порядке и завершается с соблюдением директивного срока d_i. Если же $\alpha_i=\bar{i}$, то $T_i\in\bar{J}$ и задание T_i относится к запаздывающим. Все воз-

можные последовательности (решения) можно представить в виде листьев полного двоичного дерева высотой n. Например, для $n=4$ дерево имеет вид, изображенный на рис. 6.12, где корень обозначен через e, а остальным

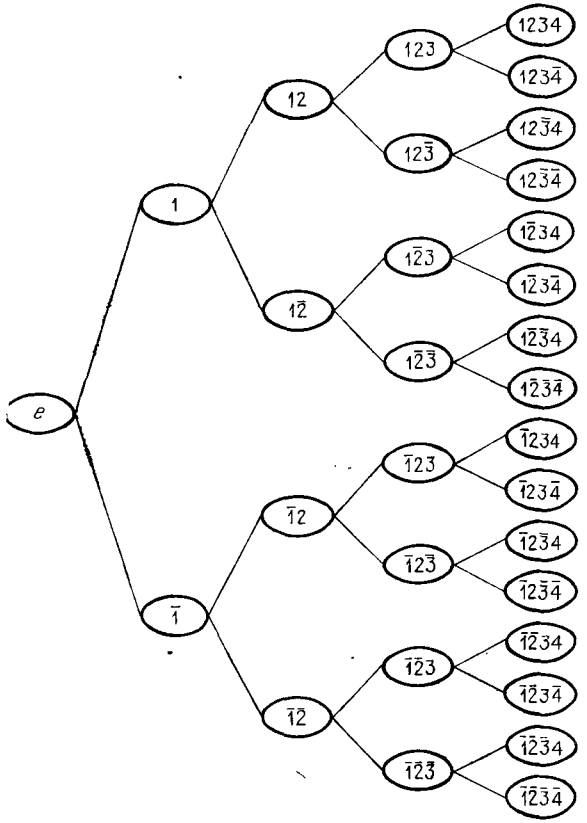


Рис. 6.12.

вершинам поставлены в соответствие последовательности вида $\alpha_1\alpha_2\ldots\alpha_r$. Лист $\bar{1}\ 2\ \bar{3}\ 4$ обозначает разбиение

$$J=\{T_2,\ T_4\},\quad \overline{J}=\{T_1,\ T_3\}.$$

Рассмотрим FLDS-задачу для 4 заданий, в которой исходные величины имеют следующие значения:

$$\begin{aligned} d_1&=2, \quad l_1=3, \quad \tau_1=2,\\ d_2&=4, \quad l_2=2, \quad \tau_2=3,\\ d_3&=5, \quad l_3=1, \quad \tau_3=2,\\ d_4&=6, \quad l_4=2, \quad \tau_4=3. \end{aligned}$$

Параметры d_i, l_i и τ_i представляют собой, соответственно, директивный срок, величину потерь и время выполнения задания T_i. Сопоставим каждой вершине (частичному разбиению) $\alpha_1\alpha_2\ldots\alpha_r$ двоичного дерева пару $(\bar{l}, \bar{\tau})$, где $\bar{l}$ обозначает сумму потерь, соответствующую заданиям, уже находящимся в $\bar{J}$, а $\bar{\tau}$ обозначает общее время выполнения заданий, уже находящихся в J, т. е.

$$\bar{l} = \sum_{\alpha_i = \bar{1}} l_i \quad \text{и} \quad \bar{\tau} = \sum_{\alpha_i = \bar{1}} \tau_i.$$

Эти пары используются как метки вершин. На рис. 6.13 представлен сокращенный вариант рис. 6.12, в котором

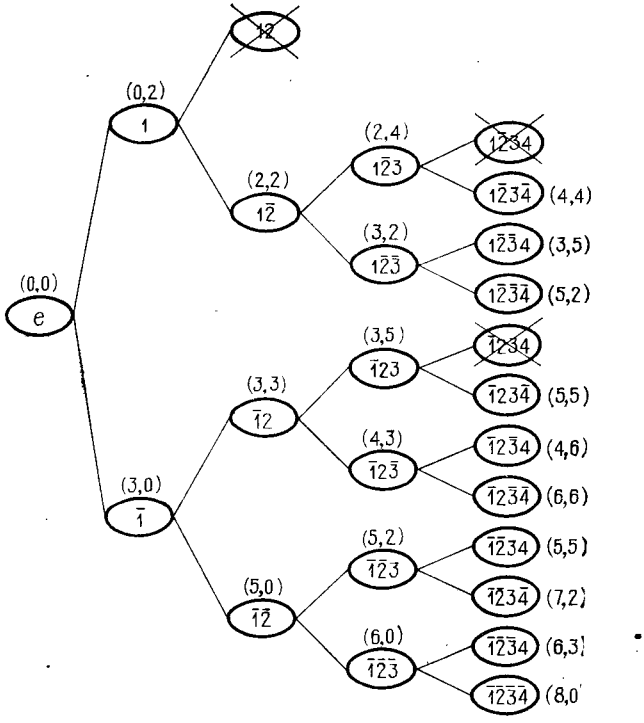


Рис. 6.13.

недопустимые частичные решения и их потомки перечеркнуты. Например, вершина 12 и ее потомки не являются допустимыми, так как задание T_2 не может быть выполнено в директивный срок, если ему предшествует задание T_1.

(Допустимость легко проверить; для этого нужно убедиться, что для каждого задания в J соблюдается его директивный срок.) Мы предполагаем, что все задания, находящиеся в $\bar{J}$, являются запаздывающими. В некоторых случаях это может быть достигнуто лишь путем вставки дополнительного времени простоя между заданиями, относящимися к J, и заданиями, относящимися к $\bar{J}$. Например, вершина $\bar{1}\ \bar{2}\ \bar{3}\ \bar{4}$ представляет решение, в котором все задания являются запаздывающими. Оптимальное расписание не требует наличия простоев между заданиями; следовательно, расписания, в которых имеются простои, не представляют интереса.

6.5.5. Алгоритм динамического программирования. Теперь найдем оптимальное решение с помощью алгоритма динамического программирования, подобного тому, который использовали Сахни [127] и Горовиц и Сахни [73]. Работа алгоритма заключается в вычислении последовательности множеств $S^{(0)}, S^{(1)}, \ldots, S^{(n)}$. Каждое множество $S^{(i)}$ представляет собой лексикографически упорядоченный набор пар $(\bar{l}, \bar{\tau})$, сопоставленных каждой вершине уровня i в соответствующем двоичном дереве; компоненты пары соответствуют суммарным потерям и суммарному времени выполнения.

FLDS-алгоритм динамического программирования

Вход: d_i, l_i, τ_i, $1 \leqslant i \leqslant n$, при этом $d_1 \leqslant d_2 \leqslant \ldots \leqslant d_n$.

Выход: расписание $\alpha_1\alpha_2\ldots\alpha_r$, которому соответствует минимальная сумма потерь.

Метод:

Шаг 1 [Начальная установка] $S^{(0)} \leftarrow \{(0, 0)\}$

Шаг 2 [Генерирование $S^{(1)}, S^{(2)}, \ldots, S^{(n)}$]

for $i = 1, 2, \ldots, n$ **do**

begin $V \leftarrow \varnothing$; $\bar{V} \leftarrow \varnothing$

for каждой пары $(\bar{l}, \bar{\tau})$ в $S^{(i-1)}$ **do**

begin

if $\bar{\tau} + \tau_i \leqslant d_i$ **then** $V \leftarrow V \cup \{(\bar{l}, \bar{\tau} + \tau_i)\}$ $\bar{V} \leftarrow \bar{V} \cup \{(\bar{l} + l_i, \bar{\tau})\}$

end

слияние V и $\bar{V}$ для получения $S(i)$

Если в процессе слияния мы вычисляли $(\bar{l}_\alpha, \bar{\tau}_\alpha)$ и $(\bar{l}_\beta, \bar{\tau}_\beta)$ такие, что $\bar{l}_\alpha \leqslant \bar{l}_\beta$ и $\bar{\tau}_\alpha \leqslant \bar{\tau}_\beta$, то $(\bar{l}_\beta, \bar{\tau}_\beta)$ исключается

end

Шаг 3 [Оптимальное решение]. Пара из $S^{(n)}$ с наименьшим значением $\bar{l}$ является меткой оптимального решения $\alpha_1\alpha_2\ldots\alpha_n$.

Этот алгоритм просматривает дерево частичных решений путем непосредственного перебора вершин в ширину (breadth-first). Метод, использованный в шаге 2 для слияния множеств и исключения пар, отличается от других общеупотребительных методов перебора. Заметим, что если вершинам $\alpha_1\alpha_2\ldots\alpha_r$ и $\beta_1\beta_2\ldots\beta_r$, соответствуют пары $(\bar{l}_\alpha, \bar{\tau}_\alpha)$ и $(\bar{l}_\beta, \bar{\tau}_\beta)$ такие, что $\bar{l}_\alpha \leqslant \bar{l}_\beta$ и $\bar{\tau}_\alpha \leqslant \bar{\tau}_\beta$, то лучшее полное решение, начинающееся с $\alpha_1\alpha_2\ldots\alpha_r$, является, по крайней мере, таким же хорошим, как и лучшее полное решение, начинающееся с $\beta_1\beta_2\ldots\beta_r$. Следовательно, частичное решение $\beta_1\beta_2\ldots\beta_r$ может быть исключено, поскольку над ним доминирует $\alpha_1\alpha_2\ldots\alpha_r$. Аналогично, для каждой вершины $\alpha_1\alpha_2\ldots\alpha_{i-1}$, которой сопоставлена пара $(\bar{l}, \bar{\tau})$ в $S^{(i-1)}$, если $\bar{\tau}+\tau_i > d_i$, то $\alpha_1\alpha_2\ldots\alpha_{i-1}$ не является допустимым и не включается в $S^{(i)}$. Правило исключения и проверка на допустимость обеспечивают, что ни одно из $S^{(i)}$ не содержит больше чем $M = \min\left\{\sum_{i=1}^{n} l_i, \sum_{i=1}^{n} \tau_i, d_n\right\}+1$ пар. Поскольку число вершин на уровне i в двоичном дереве равно, самое большее, 2^i, то другой границей размера $S^{(i)}$ является 2^n.

Для того чтобы показать, что элементы множеств V и $\bar{V}$ и объединенное множество $S^{(i)}$ генерируется в лексикографическом порядке, можно воспользоваться методом индукции. Поскольку ни V, ни $\bar{V}$ не содержат большего числа вершин, чем $S^{(i-1)}$, размер которого ограничен величиной $\min\{M, 2^n\}$, то и V и $\bar{V}$ могут быть порождены и слиты (шаг 2) для получения $S^{(i)}$ за $O\{\min[M, 2^n]\}$ шагов. Эта процедура повторяется n раз, прежде чем будет найдено оптимальное решение. Поэтому общее число шагов оценивается как $O\{\min[nM, n2^n]\}$. Заметим, что потребности в количестве шагов для наихудших вариантов выполнения алгоритма ограничены величинами nM и $n2^n$. Граница $n2^n$ не зависит от данных задачи, в то время как граница nM зависит от значений параметров l_i, τ_i и d_i. Если l_i, τ_i и d_i имеют заданные распределения, ограниченные величинами $\bar{L}$, $\bar{T}$ и $n\bar{D}$ соответственно, то $M \leqslant \min\{n\bar{L}, n\bar{T}, n\bar{D}\}$. В этом частном случае число шагов, необходимых для получения оптимального решения, есть $O(n^2)$.

Таблица 6.

Шаги алгоритма динамического программирования FLDS

i	V	$\overline{V}$	Исключенные двойки	S^i
0	—	—	—	((0, 0))
1	((0, 2))	((3, 0))	—	((0, 2)), (3, 0))
2	((3, 3))	((2, 2), (5, 0))	((3, 3))	((2, 2), (5, 0))
3	((2, 4), (5, 2))	((3, 2), (6, 0))	((5, 2))	((2, 4), (3, 2), (6, 0))
4	((3, 5), (6, 3))	(4, 4), (5, 2), (8, 0))	((6, 3))	((3, 5), (4, 4), (5, 2), (8, 0))

Оптимальное решение $(\overline{l}, \overline{\tau}) = (3, 5) \Rightarrow 1\overline{2}34$

В табл. 6.3 показаны шаги алгоритма динамического программирования FLDS, выполняемые при решении задачи с четырьмя заданиями. Заметим, что V, $\overline{V}$ и $S^{(i)}$ лексикографически упорядочены. Окончательное решение может быть найдено путем сопоставления каждой пары с соответствующей ей вершиной на двоичном дереве, изображенном на рис. 6.13.

6.5.6. Эквивалентный алгоритм ветвей и границ. Эквивалентный алгоритм ветвей и границ для задачи FLDS может быть описан с помощью девятки параметров $BB_{\text{FLDS}}=(B_b, \text{FIFO}, \{\text{AS/DB}, \text{DB/AS}\}, F_{\text{FLDS}}, D_{\text{FLDS}}, L_{\text{FLDS}}, \infty, 0, \infty)$. Параметры определяются следующим образом.

1. B_b представляет собой правило ветвления для задачи двоичного разбиения. Пусть π^r_α обозначает вершину типа $\alpha_1\alpha_2 \ldots \alpha_r$, где $\alpha_i=i$ или $\overline{i}$. Тогда $\{\pi^r_\alpha \circ\}$ представляет листья полного поддерева, имеющего π^r_α в качестве корня. Это поддерево соответствует множеству полных решений, имеющих в качестве общего предка частичное решение π^r_α. Правило ветвления B_b осуществляет разбиение множества $\{\pi^r_\alpha \circ\}$ на два непересекающихся подмножества $\{(\pi^r_\alpha \circ (r+1)) \circ\}$ и $\{(\pi^r_\alpha \circ (\overline{r+1})) \circ\}$. Это правило ветвления приводит к очевидной модификации общего алгоритма ветвей и границ для решения задач о перестановках, которая представлена на рис. 6.7. Каждая вершина π^r_α имеет теперь двух потомков $(\pi^r_\alpha \circ (r+1))$ и $(\pi^r_\alpha \circ (\overline{r+1}))$.

2. S=FIFO. Правило выбора FIFO приводит к тому, что двоичное дерево просматривается в ширину.

3. $E=\{\text{AS/DB}, \text{DB/AS}\}$. Данный набор правил исключения, используемых совместно с отношением доминирования

D_{FLDS}, обеспечивает исключение всех вершин, для которых существуют доминирующие вершины.

4. Характеристическая функция F_{FLDS} служит для исключения недопустимых частичных решений. В частности, для данной вершины $\pi'_\alpha = \alpha_1\alpha_2\ldots\alpha_r$

$$F_{\text{FLDS}}(\pi'_\alpha) \triangleq \begin{cases} 1, & \text{если для каждого } i \text{ при } \alpha_i = 1 \text{ задание } T_i \text{ завершается в соответствии с директивным сроком } d_i; \\ 0 & \text{в противном случае.} \end{cases}$$

5. Пусть заданы две вершины $\pi^i_\alpha = \alpha_1\alpha_2\ldots\alpha_j$ и $\pi^k_\beta = \beta_1\beta_2\ldots\beta_k$, которым соответствуют пары $(\bar{l}_\alpha, \bar{\tau}_\alpha)$, $(\bar{l}_\beta, \bar{\tau}_\beta)$. Правило доминирования D_{FLDS} определяется таким образом, что $(\pi^i_\alpha, \pi^k_\beta) \in D_{\text{FLDS}}$ в том и только в том случае, когда $j = k$, $\bar{l}_\alpha \leqslant \bar{l}_\beta$ и $\bar{\tau}_\alpha \leqslant \bar{\tau}_\beta$. Это эквивалентно условию, которое было использовано в методе динамического программирования для исключения пар.

6. Нижняя оценка стоимости $L_{\text{FLDS}}(\pi_\alpha)$, приписываемая каждой вершине π_α, определяется как стоимость частичного решения $\pi_\alpha = \alpha_1\alpha_2\ldots\alpha_r$, т. е. $L_{\text{FLDS}}(\pi_\alpha) \triangleq \sum_{\alpha_i=\bar{i}} l_i = \bar{l}_\alpha$.

7. $U = \infty$, поскольку предполагается, что нет ни одного заранее известного решения.

8. $BR = 0$, поскольку требуется оптимальное решение.

9. $RB = \infty$. Для того чтобы провести сравнение с методом динамического программирования, предполагается, что ресурсы по времени счета и по объему памяти не ограничены.

Этапы выполнения алгоритма BB_{FLDS} представлены в табл. 6.4. То обстоятельство, что в качестве примера также взято дерево, изображенное на рис. 6.13, позволяет сравнить эти этапы с этапами метода динамического программирования, представленными в табл. 6.3. Это сравнение показывает, что генерируются и исключаются одни и те же вершины, кроме последнего шага, на котором выполнение BB_{FLDS} завершается нахождением решения 1234 и проверкой его оптимальности. Единственное существенное различие состоит в том, что алгоритм динамического программирования генерирует все необходимые вершины на каждом уровне дерева до объединения и исключения, в то время как в алгоритме ветвей и границ каждая новая вершина проверяется сразу после ее генерирования. Поскольку для осуществления одного шага в алгоритме

Таблица 6.4

Шаги алгоритма *BB*FLDS

π_b	AS (π_b)	DB (π_b)	ES (π_b)	$\hat{L}$ (π_b)	$\hat{U}$ (π_b)
e	(e)	(1, $\bar{1}$)	ø	0	∞
1	(1, $\bar{1}$)	($1\bar{2}$)	ø	0	∞
$\bar{1}$	($\bar{1}$, $1\bar{2}$)	($\bar{1}2$, $\bar{1}\bar{2}$)	($\bar{1}2$)	2	∞
$1\bar{2}$	($1\bar{2}$, $\bar{1}\bar{2}$)	($1\bar{2}3$, $1\bar{2}\bar{3}$)	ø	2	∞
$\bar{1}2$	($\bar{1}2$, $1\bar{2}3$, $1\bar{2}\bar{3}$)	($\bar{1}23$, $\bar{1}2\bar{3}$)	($\bar{1}23$)	2	∞
$1\bar{2}3$	($1\bar{2}3$, $1\bar{2}\bar{3}$, $\bar{1}2\bar{3}$)	($1\bar{2}34$)	ø	2	∞
$1\bar{2}\bar{3}$	($1\bar{2}\bar{3}$, $\bar{1}2\bar{3}$, $1\bar{2}34$)	($1\bar{2}34$, $1\bar{2}\bar{3}4$)	ø	3	4
$\bar{1}2\bar{3}$	($\bar{1}2\bar{3}$, $1\bar{2}34$, $1\bar{2}\bar{3}4$, $1\bar{2}3\bar{4}$)	—	—	3	3

$$\frac{\hat{U}-\hat{L}}{\hat{U}} = 0 \Rightarrow \hat{\pi} \Rightarrow 1\bar{2}34 \text{—оптимальное решение.}$$

динамического программирования требуется меньшее общее число сравнений для проверки доминирования, то он является более эффективным. Общий алгоритм ветвей и границ также может быть модифицирован и доведен до этого же уровня эффективности.

6.5.7. Комментарии. Связь между методом ветвей и границ и динамическим программированием была продемонстрирована на примере модельной задачи. В общем случае для заданной задачи составления расписания и заданного алгоритма динамического программирования, предназначенного для ее решения, может быть также найден эквивалентный алгоритм ветвей и границ. Эта связь не должна вызывать удивление. В обоих случаях соответствующее пространство решений может быть представлено в виде конечного ориентированного дерева частичных решений, и задача заключается в локализации и извлечении оптимального решения, при этом полный перебор осуществляется лишь в небольшой части полного дерева. Применение метода ветвей и границ и динамического программирования к другим задачам составления расписаний рассмотрено в работах Горовица и Сахни [73], Коглера [89], Ринноу Кана и др. [121] и Сахни [127].

6.6. Заключительные замечания

Мы рассмотрели с определенной степенью детализации вычислительные методы практического решения задач составления расписаний, не поддающихся решению аналити-

ческими методами. Эти задачи составления расписаний попадают в категорию «больших» комбинаторных оптимизационных задач, многие из которых являются NP-полными и считаются традиционно трудноразрешимыми. Главное внимание в этой главе было уделено получению при заданном количестве вычислительных ресурсов наиболее полезной информации, необходимой для осуществления упорядочения заданий либо в виде оптимального расписания (когда это возможно), либо в виде расписания с обеспечением заданной точности относительно оптимального.

В этой главе обобщены работы многих исследователей, и некоторые из работ представлены в списке цитированной литературы. Развитие этих исследований представляется идущим в направлении «численного анализа» для комбинаторных задач — некоторой теории и методологии, которые позволят нам работать с переборными задачами и получать результаты, имеющие предсказуемое качество в зависимости от затраченного объема работы. Продолжая аналогию с задачами, в которых участвуют непрерывные величины, нельзя рассчитывать получить с помощью численных методов «точное» решение нелинейной оптимизационной задачи или даже задачи линейного программирования. Однако можно надеяться получить решения, обладающие достаточной точностью, с помощью математического аппарата, оперирующего числами конечной длины. С другой стороны, правомерна ли вообще постановка вопроса о получении «оптимального» решения большой NP-полной задачи?

Недавно был опубликован ряд результатов, относящихся к только что затронутым проблемам [127, 133, 56]. В частности, в [127] на основании того факта, что в задаче составления расписания с директивными сроками и фиксированными значениями потерь, которая была рассмотрена в § 6.5, время счета ограничено $O(nM)$, делается вывод о том, что для этой задачи (и ряда сходных задач) решения со степенью точности $BR=\varepsilon$ могут быть получены за время $O(n^2/\varepsilon)$. С другой стороны, в [133] показано, что существуют приближенные постановки NP-полных задач (более точно, P-полных, если следовать обозначениям, принятым в [133]), которые также являются NP-полными, а в [56] показано, что приближенная постановка задачи раскраски графа является в некотором смысле такой же трудной, как и сама задача раскраски графа. Таким образом, Сахни и Гонзалес [133] указывают, что некоторые «трудные» комбинаторные задачи являются «более трудными», чем другие.

С нашей точки зрения, имеется несколько областей для дальнейшего исследования. Прежде всего, представляется заслуживающим рассмотрения вопрос о сложности «приближенных» комбинаторных задач, т. е. задач, в которых требуется получение скорее хороших решений, чем оптимальных. К этой области относится общий вопрос о чувствительности решения таких задач к параметрам. Кажется, что разработка алгоритмов решения такого рода задач уже началась. Много усилий требует создание автоматических методов построения нижних оценок, нахождение хороших окрестностей для метода поиска в локальной окрестности и совершенствование метода ветвей и границ в целом.

1. Agin N. Optimum Seeking with Branch and Bound.— Management Science, 1966, **13**, № 4, p. 176—185.
2. Aho A. V., Garey M. R., Ullman J. D. The Transitive Reduction of a Directed Graph.— SIAM Journal on Computing, 1972, № 1, p. 131—137.
3. Aho A. V., Hopcroft J. E., Ullman J. D. The Design and Analysis of Computer Algorithms.— Reading, Mass.: Addison-Wesley, 1974. [Русск. перев.: Ахо А., Хопкрофт Дж., Ульман Дж. Построение и анализ вычислительных алгоритмов.— М.: Мир, 1979.]
4. Ashour S. An Experimental Investigation and Comparative Evaluation of Flow-Shop Scheduling Techniques.— Operations Research, 1970, **18**, № 3, p. 541—549.
5. Baer J. L. A Survey of Some Theoretical Aspects of Multiprocessing.— Computing Surveys, 1973, **5**, № 1, pp. 31—80.
6. Baer J. L. Optimal Scheduling on Two Processors of Different Speeds. Computer Architectures and Networks/E. Gelenbe and R. Mahl (Eds.).— North Holland, 1974, p. 27—45.
7. Balas E. A Note on the Branch-and-Bound Principle.— Operations Research, 1968, **16**, № 2, p. 442—444; Erratum, 1968, **16**, № 4, p. 886.
8. Balas E. Machine Sequencing via Disjunctive Graphs: An Implicit Enumeration Algorithm.— Operations Research, 1969, **17**, № 6, p. 941—957.
9. Baker K. Introduction to Sequencing and Scheduling.— N. Y.: John Wiley, 1974.
10. Bruno J., Coffman E. G., Jr., Johnson D. B. On Batch Scheduling of Jobs with Stochastic Service Times and Cost Structures on a Single Server. Technical Report № 154, Computer Science Department, Pennsylvania State University, August 1974.— Journal of Computer and System Sciences, 1976, **12**, № 3, p. 319—335.
11. Bruno J., Coffman E. G., Jr., Sethi R. Scheduling Independent Tasks to Reduce Mean Finishing Time.— Communications of the ACM, 1974, **17**, № 7, p. 382—387.
12. Bruno J., Coffman E. G., Jr., Sethi R. Algorithms for Minimizing Mean Flow Time. Proceedings.— IFIPS Congress, North Holland, August 1974, p. 504—510.
13. Bruno J., Hofri M. On Scheduling Chains of Jobs on One Processor with Limited Preemptions. Technical Report № 153, Computer Science Department, Pennsylvania State University, March 1974.— SIAM Journal on Computing, 1975, **4**, № 4, p. 478—490.

14. Brucker P., Lenstra J. K., Rinnooy Kan A. H. G. Complexity of Machine Scheduling Problems. Technical Report BW 43/75.— Mathematisch Centrum, Amsterdam, 1975 *).
15. Bowden E. K. Priority Assignment in a Network of Computers.— IEEE Transactions on Computers, 1969, **C-18**, № 11, p. 1021—1026.
16. Brucker P. On Hu's «Cutting the Longest Queue» Algorithm. Technical Report № 14, Series B.— University of Regensburg, West Germany, July 1973.
17. Brinch-Hansen P. Operating Systems Principles.— N. J.: Prentice-Hall, Englewood Cliffs, 1973.
18. Bruno J., Sethi R. On the Complexity of Mean Flow-Time Scheduling. Technical Report.— Computer Science Department, Pennsylvania State University, 1975.
19. Bruno J., Steiglitz K. The Expression of Algorithms by Charts.— Journal of the ACM, 1972, **19**, № 3, p. 517—525.
20. Chvatal V. (personal communication).
21. Cody R., Coffman E. G., Jr. Record Allocation for Minimizing Expected Retrieval Costs on Drum-Like Storage Devices.— Proceedings, 8th Annual Princeton Conference on Information Sciences and Systems, March 1973.
22. Coffman E. G., Jr., Denning P. J. Operating Systems Theory.— N. J.: Prentice-Hall, Englewood Cliffs, 1973.
23. Christofides N., Eilon S. Algorithms for Large-Scale Travelling Salesman Problems.— Operational Research Quarterly, 1972, 23, № 4, p. 511—518.
24. Coffman E. G., Jr., Graham R. L. Optimal Scheduling for Two Processor Systems.— Acta Informatica, 1972, **1**, № 3, p. 200—213.
25. Charlton J. M., Death C. C. A Method of Solution for General Machine-Scheduling Problems.— Operations Research, 1970, **18**, № 4, p. 689—707.
26. Coffman E. G., Jr., Labetoulle L. Deterministic Scheduling to Minimize Mean Number in System. Technical Report, Institut de Recherche d'Informatique et d'Automatique, Rocquencourt, France, 1975.
27. Chen N. F., Liu C. L. Technical Report, Computer Science Dept. University of Illinois, 1975.
28. Conway R. W., Maxwell W. L., Miller L. W. Theory of Scheduling.— Reading, Mass.: Addison-Wesley, 1967. [Русск. перев.: Конвей Р. В., Максвелл В. Л., Миллер Л. В. Теория расписаний.— М.: Наука, 1975.]
29. Coffman E. G., Jr. A Survey of Mathematical Results in Flow-Time Scheduling for Computer Systems. Proceedings, GI 73.— Hamburg: Springer-Verlag, 1973, p. 25—46. (Lect. Notes Comput. Sci., 1973, № 1, p. 25—46.)
30. Cook S. A. The Complexity of Theorem Proving Procedures. Proceedings, 3rd ACM Symposium on Theory of Computing.— 1971, p. 151—158. [Русск. перев.: Кук С. А. Сложность процедур вывода теорем.— В кн.: Кибернетический сборник. Новая серия. Вып. 12. Сборник переводов. М.: Мир, 1975, с. 5—15.]
31. Cook S. A., Rechkow R. A. Time-Bounded Random Access Machines.— Journal of Computer and System Sciences, 1973, № 4.

*) См. соответствующую работу: Lenstra J. K., Rinnooy Kan A. H. G., Brucker P. Complexity of Machine Scheduling Problems.— Ann. Discrete Math., 1977, № 1, p. 343—362.

32. Coffman E. G., Jr., Sethi R. Algorithms Minimizing Mean Flow-Time: Schedule-Length Properties. Technical Report, Computer Science Department, Pennsylvania State University, 1973.— Acta Informatica, 1976, **6**, № 1, p. 1—14.
33. Chandra A. K., Wong C. K. Worst-Case Analysis of a Placement Algorithm Related to Storage Allocation.— SIAM Journal on Computing, 1975, **4**, № 3, p. 249—263.
34. Dannenbring D. G. An Evaluation of Flow-Shop Heuristics. Technical Report.— University of North Carolina, 1974.
35. Dunham B., Fridshal D., Fridshal R., North J. H. Design by Natural Selection.— IBM Research Report № RC-476, June 1961.
36. Denning P. J., Graham G. S. A Note on Subexpression Ordering in the Execution of Arithmetic Expressions.— Communications of the ACM, 1973, **16**, № 11, p. 700—702; Erratum, 1974, **17**, № 8, p. 455.
37. Day J. E., Hottenstein M. P. Review of Sequencing Research.— Naval Research and Logistics Quarterly, 1970, **17**, № 1, p. 11—40.
38. Edmonds J. Paths, Trees, and Flowers.— Canadian Journal of Mathematics, 1965, **17**, № 3, p. 449—467.
39. Eastman W. L., Even S., Isaacs I. M. Bounds for the Optimal Scheduling of *n* Jobs on *m* Processors.— Management Science, 1964, **11**, № 2, p. 268—279.
40. Edmonds J., Karp R. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems.— Journal of the ACM, 1972, **19**, № 2, p. 248—264.
41. Elmaghraby S. E. The Machine Sequencing Problem — Review and Extensions.— Naval Research and Logistics Quarterly, 1968, **15**, № 2, p. 205—232.
42. Ford L. R., Fulkerson D. R. Flows in Networks.— N. J.: Princeton University Press, Princeton, 1962. [Русск. перев.: Форд Л. Р., Фалкерсон Д. Р. Потоки в сетях.— М.: Мир, 1966.]
43. Fujii M., Kasami T., Ninomiya K. Optimal Sequencing of Two Equivalent Processors.— SIAM Journal on Applied Mathematics, 1969, **17**, № 4, p. 784—789; Erratum, 1971, **20**, № 1, p. 141.
44. Fox B. L., Schrage L. E. The Value of Various Strategies in Branch-and-Bound (unpublished report), June 1972.
45. Graham R. L. Bounds for Certain Multiprocessing Anomalies.— Bell System Technical Journal, 1966, **45**, № 9, p. 1563—1581.
46. Graham R. L. Bounds on Multiprocessing Timing Anomalies.— SIAM Journal on Applied Mathematics, 1969, **17**, № 2, p. 416—429.
47. Graham R. L. Bounds on Multiprocessing Anomalies and Related Packing Algorithms.— Proceedings, AFIPS Conference, 1972, № 40, p. 205—217.
48. Garey M. R. Optimal Task Sequencing with Precedence Constraints.— Discrete Mathematics, 1973, № 4, p. 37—56.
49. Garey M. R., Graham R. L. Bounds on Scheduling with Limited Resources.— Operating Systems Review, 1973, 7, № 4, p. 104—111.
50. Garey M. R., Graham R. L. Bounds for Multiprocessing Scheduling with Resource Constraints.— SIAM Journal on Computing, 1975, **4**, № 2, p. 187—200.
51. Garey M. R., Graham R. L., Johnson D. S., Yao A. C. Resource Constrained Scheduling as Generalized Bin-Packing.— Journal of Combinatorial Theory, 1976, **21**, № 3.

52. Garey M. R., Graham R. L., Ullman J. D. Worst-Case Analysis of Memory Allocation Algorithms.— Proceedings, 4th Annual ACM Symposium on the Theory of Computing, 1972, p. 143.
53. Gilmore P. C. Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem.— SIAM Journal on Applied Mathematics, 1962, **10**, № 2, p. 305—313.
54. Garey M. R., Johnson D. S. Complexity Results for Multiprocessor Scheduling Under Resource Constraints.— Proceedings, 8th Annual Princeton Conference on Information Sciences and Systems, 1974, p. 168—182; SIAM Journal on Computing, 1975, **4**, № 4, p. 397—411.
55. Garey M. R., Johnson D. S. Deadline Scheduling of Equal Execution Time Tasks on Two Processors. Technical Report.— Murray Hill, N. J.: Bell Laboratories, 1975.
56. Garey M. R., Johnson D. S. The Complexity of Near-Optimal Graph Coloring. Technical Report.— Murray Hill, N. J.: Bell Laboratories, 1975; Journal of the ACM, 1976, № 23, p. 43—49.
57. Garey M. R., Johnson D. S., Sethi R. The Complexity of Flowshop and Jobshop Scheduling. Technical Report № 168, Computer Science Dept.— The Pennsylvania State University, 1975; Math. Oper. Res., 1976, **1**, № 2, p. 117—129.
58. Geoffrion A. M., Marsten R. E. Integer Programming Algorithms: A Framework and State of the Art Survey.— Management Science, 1972, **18**, № 9, p. 465—491.
59. Gapp W., Mankelar P. S., Mitten L. G. Sequencing Operations to Minimize In-Process Inventory Costs.— Management Science, 1965, № 11, p. 476—484.
60. Garfinkel R. S., Nemhauser G. L. A Survey of Integer Programming Emphasizing Computation and Relations among Models. Mathematical Programming/T. C. Hu and S. M. Robinson (Eds.) — New York: Academic Press, 1973, p. 77—156.
61. Graham G. S. (personal communication).
62. Gratzer F. J. Computer Solution of Large Multicommodity Flow Problems, Ph. D. thesis.— Electrical Engineering Department, Princeton University, 1970.
63. Gratzer F. J., Steiglitz K. A Heuristic Approach to Large Multicommodity Flow Problems. Proceedings of the Symposium on Computer-Communications Networks and Teletraffic, Microwave Research Institute Symposia Series, v. XXII.— New York: Brooklyn Polytechnic Press, 1972, p. 311—324.
64. Gupta J. N. D. Economic Aspects of Production Scheduling Systems.— Journal Operations Research Society of Japan, 1971, **13**, № 4, p. 169—193.
65. Hu T. C. Parallel Sequencing and Assembly Line Problems.— Operations Research, 1961, **9**, № 6, p. 841—848. [Русск. перев.: Ху Т. С. Параллельное упорядочивание и проблемы линии сборки.— В кн.: Кибернетический сборник. Новая серия. Вып. 4. Сборник переводов.— М.: Мир, 1967, с. 43—56.]
66. Halasz S. (personal communication).
67. Hillier F. S. A Bound-and-Scan Algorithm for Pure Integer Linear Programming with General Variables.— Operations Research, 1969, № 17, p. 638—679.
68. Held M., Karp R. A Dynamic Programming Approach to Sequencing Problems.— SIAM Journal on Applied Mathematics, 1962, **10**, № 1, p. 196—210. [Русск. перев.: Хелд М., Карп Р. М. Применение динамического программирования.—

В кн.: Кибернетический сборник. Вып. 9. Сборник переводов. М.: Мир, 1964, с. 202—218.
69. Held M., Karp R. The Traveling-Salesman Problem and Minimum Spanning Trees. Part II.— Mathematical Programming, 1971, 1, № 1, p. 6—25.
70. Hanan M., Kurtzberg J. M. Placement Techniques. Ch. 5. Design Automation of Digital Systems/M. A. Breuer (Ed.) — Englewood Cliffs, N. J.: Prentice-Hall, 1972.
71. Horn W. A. Single-Machine Job Sequencing with Treelike Precedence Ordering and Linear Delay Penalties.— SIAM Journal on Applied Mathematics, 1972, 23, № 2, p. 189—202.
72. Horn W. A. Minimizing Average Flow Time with Parallel Machines.—Operations Research, 1973, 21, № 3, p. 846—847.
73. Horowitz E., Sahni S. Exact and Approximate Algorithms for Scheduling Non-Identical Processors.— Technical Report, Computer Science Program, University of Southern California, 1974; Journal of the ACM, 1976, 23, № 2, p. 317—327.
74. Horvath E. C., Sethi R. Preemptive Schedules for Independent Tasks. Technical Report № 162.— Computer Science Department, The Pennsylvania State University, 1975.
75. Hopcroft J. E., Ullman J. D. Set Merging Algorithms.— SIAM Journal on Computing, 1973, 2, № 4, p. 294—303.
76. Ignall E., Schrage L. Application of the Branch and Bound Technique to Some Flow-Shop Scheduling Problems.— Operations Research, 1965, 13, № 3, p. 400—412.
77. Johnson D. S. Near-Optimal Bin-Packing Algorithms, Ph. D. thesis.— Electrical Engineering Department, Massachusetts Institute of Technology, 1974.
78. Johnson D. S. Fast Algorithms for Bin Packing.— Journal of Computer and System Sciences, 1974, 8, № 3, p. 272—314.
79. Johnson D. S. Fast Allocation Algorithms.— Proceedings, 13th Annual IEEE Symposium on Switching and Automata Theory, 1972, p. 144—154.
80. Jackson J. R. An Extension of Johnson's Results on Job-Lot Scheduling.— Naval Research and Logistics Quarterly 1956, 3, № 3, p. 201—203.
81. Jackson J. R. Scheduling a Production Line to Minimize Maximum Tardiness. Research Report № 43, Management Sciences Research Project.— University of California at Los Angeles, January 1955.
82. Johnson D. S., Demers A., Ullman J. D., Garey M. R., Graham R. L. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms.— SIAM Journal on Computing, 1974, № 3, p. 299—326.
83. Johnson S. M. Optimal Two-and-Three-Stage Production Schedules.— Naval Research and Logistics Quarterly, 1954, 1, № 1, p. 61—68. [Русск. перев.: Джонсон С. Оптимальное расписание для двух- и трехступенчатых процессов с учетом времени наладки.— В кн.: Кибернетический сборник. Новая серия. Вып. 1. Сборник переводов. М.: Мир, 1965, с. 78—86; см. также кн.: Календарное планирование.— М.: Прогресс, 1966, с. 33—41.]
84. Kaufman M. T. Anomalies in Scheduling Unit-Time Tasks.— Stanford Electronics Laboratory Technical Report № 34, 1972.
85. Kaufman M. T. An Almost-Optimal Algorithm for the Assembly Line Scheduling Problem.— IEEE Transactions on Computers, 1974, TC-74, № 11, p. 1169—1174.

86. Karp R. M. Reducibility Among Combinatorial Problems.— In: Complexity of Computer Computation/R. E. Miller and J. W. Thatcher (Eds.) New York: Plenum Press, 1972, p. 85—104. [Русск. перев.: Карп Р. М. Сводимость комбинаторных проблем.— В кн.: Кибернетический сборник. Новая серия. Вып. 12. Сборник переводов. М.: Мир, 1975, с. 16—38.]
87. Knuth D. E. The Art of Computer Programming. V. 1, Fundamental Algorithms.— Reading, Mass.: Addison-Wesley, 1968. [Русск. перев.: Кнут Д. Искусство программирования для ЭВМ. Т. 1. Основные алгоритмы.— М.: Мир, 1976.]
88. Kohler W. H. Exact and Approximate Algorithms for Permutation Problems, Ph. D. thesis.— Princeton University, Electrical Engineering Department, 1972.
89. Kohler W. H. A Preliminary Evaluation of the Critical Path Method for Scheduling Tasks on Multiprocessor Systems.— Tech nical Report NSF GK 37400-74-2, Department of Electrical and Computer Engineering, University of Massachusetts, 1974. IEEE. Transactions on Computers, 1975, **C-24**, № 12, p. 1235—1238.
90. Kolesar P. J. A Branch and Bound Algorithm for the Knapsack Problem.— Management Science, 1967, **13**, № 9, p. 723—735.
91. Krause K. L. Analysis of Computer Scheduling with Memory Constraints, Ph. D. thesis.— Purdue University, Computer Science Department, 1973.
92. Krone M. J. Heuristic Programming Applied to Scheduling Problems, Ph. D. thesis.— Princeton University; Electrical Engineering Department, 1970.
93. Krone M., Steiglitz K. Heuristic Programming Solution of a Flowshop Scheduling Problem.— Operations Research, 1974, **22**, № 3, p. 629—638.
94. Kohler W. H., Steiglitz K. Characterization and Theoretical Comparison of Branch-and-Bound Algorithms for Permutation Problems.— Journal of the ACM, 1974, **21**, № 1, p. 140—156.
95. Kohler W. H., Steiglitz K. Evolutionary Learning of Neighborhoods for Heuristic Programs and Application to a Sequencing Problem.— Proceedings, 9th Allerton Conference on Circuit and System Theory, October 1971, p. 377—389.
96. Kohler W. H., Steiglitz K. Exact, Approximate, and Guaranteed Accuracy Algorithms for the Flow-Shop Problem $n/2/F/\overline{F}$.— Journal of the ACM, 1975, **22**, № 1, p. 106—114.
97. Labetoulle J. Some Theorems on Real Time Scheduling.— In: Computer Architecture and Networks/E. Gelenbe and R. Mahl (Eds.).— North Holland, 1974, p. 285—298.
98. Lawler E. L. On Scheduling Problems with Deferral Costs.— Management Science, 1964, **11**, № 2, p. 280—288.
99. Lawler E. L. Optimal Sequencing of a Single Machine Subject to Precedence Constraints.— Management Science, 1973, **19**, № 5, p. 544—546.
100. Lin S. Computer Solutions to the Traveling Salesman Problem.— Bell System Technical Journal, 1965, **44**, № 10, p. 2245—2269.
101. Liu J. W. S., Liu C. L. Performance Analysis of Heterogeneous Multiprocessor Computing Systems. In: Computer Architecture and Networks/E. Gelenbe and R. Mahl (Eds.).— North Holland, 1974, p. 331—343.
102. Liu J. W. S., Liu C. L. Bounds on Scheduling Algorithms for Heterogeneous Computer Systems. Proceedings, IFIPS 74 Congress.— North Holland, August 1974, p. 349—353.

103. Liu C. L., Layland J. W. Scheduling Algorithms for Multiprogramming in a Hard Real-Time Environment.— Journal of the ACM, 1973, **20**, № 1, p. 46—61.
104. Lin S., Kernighan B. W. An Effective Heuristic for the Traveling-Salesman Problem.— Operations Research, 1973, **21**, № 2, p. 498—516.
105. Lawler E. L., Moore J. M. A Functional Equation and its Application to Resource Allocation and Sequencing Problems.— Management Science, 1969, **16**, № 1, p. 77—84.
106. Lam S., Sethi R. Worst-case Analysis of Two Scheduling Algorithms. Technical Report, Computer Science Department, Pennsylvania State University, 1975.— SIAM Journal of Computing, 1977, **6**, № 3, p. 518—536.
107. Lawler E. L., Wood D. E. Branch-and-Bound Methods: A Survey.— Operations Research, 1966, **14**, № 4, p. 699—719.
108. Misra J. Constructive Proofs of Two Scheduling Algorithms: Technical Report.— Gaithersburg: IBM, April 1974.
109. McNaughton R. Scheduling with Deadlines and Loss Functions.— Management Science, 1959, **6**, № 1, p. 1—12.
110. Muntz R. R., Coffman E. G., Jr. Preemptive Scheduling of Real Time Tasks on Multiprocessor Systems.— Journal of the ACM, 1970, **17**, № 2, p. 324—338.
111. Muntz R. R., Coffman E. G., Jr. Optimal Preemptive Scheduling on Two-Processor Systems.— IEEE Transactions on Computers, 1969, **C-18**, № 11, p. 1014—1020.
112. Mellor P. A Review of Job Shop Scheduling.— Operational Research Quarterly, 1966, **17**, № 2, p. 161—171.
113. Mitten L. G. Branch-and-Bound Methods: General Formulation and Properties.— Operations Research, 1970, **18**, № 1, p. 24—34.
114. Moore J. M. An *n* Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs.— Management Science, 1968, **15**, № 1, p. 102—109.
115. Munkres J. Algorithms for the Assignment and Transportation Problems.— SIAM Journal on Applied Mathematics, 1957, № 5, p. 32—38.
116. Muraoka Y. Parallelism, Exposure and Exploitation in Programs, Ph. D. thesis.— University of Illinois, Computer Science Department, 1971.
117. Nollemeier H. *E* in Branch-and-Bound-Verfahren-Generator.— Computing, 1971, № 8, p. 99—106.
118. Pierce J. F., Crowston W. B. Three-Search Algorithms for Quadratic Assignment Problems.— Naval Research and Logistics Quarterly, 1971, **18**, № 1, p. 1—36.
119. Rothkopf M. H. Scheduling Independent Tasks on Parallel Processors.— Management Science, 1966, **12**, № 5, p. 437—447.
120. Rinnooy Kan A. H. G. On Mitten's Axioms for Branch-and-Bound.— In: Working Paper W/74/45/03, Graduate School of Management. Interfaculteit Bedr'jfskunde, Delft, The Netherlands, April 1974.
121. Rinnooy Kan A. H. G., Lageweg B. J., Lenstra J. K. Minimizing Total Costs in One-Machine Scheduling. Technical Report № BW33/74, Mathematisch Centrum, Amsterdam, 1974.— Operations Research, 1975, 23, № 5, p. 908—927.
122. Roth R. H. An Approach to Solving Linear Discrete Optimization Problems.— Journal of the ACM, 1970, **17**, № 2, p. 303—313.

123. Roy B. Procedure d'Exploration par Separation et Evaluation.— Revue Francaise d'Informatique et de Recherche Operationelle, 1969, № 6, p. 61—90.
124. Reiter S., Sherman G. Discrete Optimizing.— SIAM Journal on Applied Mathematics, 1965, 13, № 3, p. 864—889.
125. Salkin H. M. On the Merit of the Generalized Origin and Restarts in Implicit Enumeration.— Operations Research, 1970, № 18, p. 549—554.
126. Savage S. L. The Solution of Discrete Linear Optimization Problems by Neighborhood Search Techniques, Ph. D. thesis.— Yale University, Computer Science Department, March 1973.
127. Sahni S. Algorithms for Scheduling Independent Tasks. Department of Computer Science, University of Minnesota, 1974.— Journal of the ACM, 1976, 23, № 1, p. 116—127.
128. Schrage L. Solving Resource-Constrained Network Problems by Implicit Enumeration-Nonpreemptive Case.— Operations Research, 1970, 18, № 2, p. 263—278.
129. Schrage L. Solving Resource-Constrained Network Problems by Implicit Enumeration-Preemptive Case.— Operations Research, 1972, 20, № 3, p. 668—677.
130. Sethi R. Scheduling Graphs on Two Processors.— SIAM Journal on Computing, 1976, 5, № 1, p. 73—82.
131. Sethi R. Complexity of Flow-Shop Scheduling. Technical Report № 161.— Pennsylvania State University, Computer Science Department, December 1974.
132. Sevcik K. C. Scheduling for Minimum Total Cost Using Service Time Distributions.— Journal of the ACM, 1974, 21, № 1, p. 66—75.
133. Sahni S., Gonzalez T. *P*-Complete Problems and Approximate Solutions. Technical Report 74-5, Computer.— University of Minnesota; Information, and Control Sciences Department, March 1974.
134. Sidney J. B. One Machine Sequencing with Precedence Relations and Deferral Costs. Part I. Working Paper № 124; Part II. Working Paper № 125.— University of British Columbia, Faculty of Commerce and Business Administration, 1972.
135. Sidney J. B. An Extension of Moore's Due-Date Algorithm. Symposium on the Theory of Scheduling and Its Applications/ S. M. Elmagrabhy (Ed.) — Springer-Verlag, 1973, p. 393—398. (Lect. Notes in Economics and Math. Syst., 1973, № 86, p. 393—398.)
136. Smith W. E. Various Optimizers for Single-Stage Production.— Naval Research and Logistics Quarterly, 1956, 3, № 1, p. 59—66.
137. Schindler S., Simonsmeier W. The Class of All Optimal Schedules for Two Processor Systems.— Proceedings, 7th Annual Princeton Conference on Information Sciences and Systems, 1973.
138. Steiglitz K., Weiner P. Some Improved Algorithms for Computer Solution to the Traveling Salesman Problem.— Proceedings, 6th Allerton Conference on Circuit and System Theory, October 1968, p. 814—821.
139. Steiglitz K., Weiner P., Kleitman D. J. The Design of Minimum Cost Survivable Networks.— IEEE Transactions on Circuit Theory, 1969, CT-16, № 4, p. 455—460.
140. Savage S. L., Weiner P., Krone M. J. Convergent Local Search. Research Report № 14.— Yale University, Computer Science Department, March 1973.

141. Szwarc W. Optimal Elimination Methods in the *mn* Flow-Shop Scheduling Problem.— Operations Research, 1973, **21**, № 6, p. 1250—1259.
142. Tarjan R. E. Efficiency of a Good but not Linear Set Union Algorithm.— Memorandum ERL-M434.— University of California, Department of Electrical Engineering and Computer Science, March 1974.
143. Ullman J. D. Polynomial Complete Scheduling Problems.— Operating Systems Review, 1973, 7, № 4, p. 96—101.
144. Ullman J. D. The Performance of a Memory Allocation Algorithm. Technical Report № 100.— Princeton University, Electrical Engineering Department, 1971.
145. Weiner P. S., Savage S. L., Bagchi A. Neighborhood Search Algorithms for Finding Optimal Traveling Salesman Tours Must Be Inefficient. Research Report № 13.— Yale University, Computer Science Department, March 1973.
146. Yao A. C. Scheduling Unit-Time Tasks with Limited Resources. Proceedings Sagamore Computer Conference, 1974.— Lect. Notes Comput. Sci., 1975, № 24, p. 17—36.
147. Yao A. C. On Scheduling with Limited Resources. Computer Science Dept.— University of Illinois.

Литература, добавленная при переводе*)

148. Авен О. И., Гурин Н. Н., Коган Я. А. Оценка качества и оптимизация вычислительных систем.— М.: Наука, 1982.
149. Андон Ф. И., Кукса А. И., Поляченко Б. Е Об оптимальном планировании процесса обработки на ЭВМ взаимосвязанных задач.— Кибернетика, 1980, № 3, с. 51—53.
150. Бакенрот В. Ю., Макаревич О. Б., Чефранов А. Г. Об эффективности алгоритмов оперативной диспетчеризации сложных задач в пакетном режиме работы однородных вычислительных систем.— Управляющие системы и машины, 1981, № 5, с. 72—75.
151. Баранов С. И., Гребнев А. Д., Щерс А. Л. Методы управления потоком работ в вычислительных системах коллективного пользования (обзор).— М.: ВНИИ ПОУ ГКНТ СССР, 1980.— (Деп. в ВИНИТИ, № 4507-81, 17 сентября 1981 г.).
152. Барский А. Б. Планирование параллельных вычислительных процессов.— М.: Машиностроение, 1980.
153. Бондаренко В. Ф., Кирпичников В. М., Шкляр В. Б. Методы оптимального планирования и диспетчирования вычислительных систем. Обзорная информация.— Минск: БелНИИНТИ, 1976.
154. Бронштейн И. И., Трахтенгерц Э. А., Шурайц Ю. М. Минимизация времени выполнения набора программ с различной длительностью обработки на многопроцессорной ЦВМ.— Автоматика и телемеханика, 1976, № 1, с. 179—186.
155. Буланже Д. Ю., Сушков Б. Г. Алгоритмы управления вычислительными системами жесткого реального времени.— Изв. АН СССР, Техническая кибернетика, 1982, № 6, с. 160—169.
156. Вайрадян А. С., Коровин А. В., Удалов В. Н. Эффективное функционирование мультипроцессорных систем.— М.: Радио и связь, 1983 (в печати).

*) Содержит работы, опубликованные с 1976 г., т. е. после написания данной книги.

157. Визинг В. Г. Минимизация максимального запаздывания в системах обслуживания с прерываниями.— Журнал вычислит. матем. и матем. физики, 1982, 22, № 3, с. 717—722.
158. Гаврилов А. В., Жиратков В. И. Алгоритмы оперативного планирования работы РВС в режиме интерактивных вычислений.— Программирование, 1978, № 5, с. 62—68.
159. Генс Г. В., Левнер Е. В. Приближенные алгоритмы для некоторых универсальных задач теории расписаний.— Изв. АН СССР, Техническая кибернетика, 1978, № 6, с. 38—43.
160. Головкин Б. А. Методы и средства параллельной обработки информации.— В кн.: Итоги науки и техники. Сер. Теория вероятностей. Математическая статистика. Теоретическая кибернетика. Том 17.— М.: ВИНИТИ, 1979, с. 85—193.
161. Головкин Б. А. Расчет характеристик и планирование параллельных вычислительных процессов—.М.: Радио и связь, 1983.
162. Гольдгабер Е. М. Задача минимизации времени исполнения проекта работ, заданного деревом.— Кибернетика, 1977, № 2, с. 102—107.
163. Гордон В. С., Шафранский Я. М. К вопросу минимизации функций на множестве перестановок частично упорядоченных элементов.— Изв. АН БССР. Сер. физ.-матем. наук, 1979, № 2, с. 122—124.
164. Гордон В. С., Танаев В. С. О минимаксных задачах теории расписаний с одним прибором.— Изв. АН БССР. Сер. физ.-матем. наук, 1982, № 3.
165. Гэри М., Джонсон Д. Вычислительные машины и трудноразрешаемые задачи.— М.: Мир, 1982.
166. Евреинов Э. В., Хорошевский В. Г. Однородные вычислительные системы.— Новосибирск: Наука, 1978.
167. Емеличев В. А., Супруненко Д. А., Танаев В. С. О работах белорусских математиков в области дискретной оптимизации.— Изв. АН СССР, Техническая кибернетика, 1982, № 6, с. 25—45.
168. Каляев А. В., Бакенрот В. Ю., Макаревич О. Б. Об алгоритмах функционирования ОВС в режиме пакетной обработки сложных задач.— Кибернетика, 1982, № 3, с. 68—71.
169. Канцедал С. А. Вычислительные алгоритмы решения задач теории расписаний.— Изв. АН СССР, Техническая кибернетика, 1982, № 3, с. 42—51.
170. Клейнрок Л. Вычислительные системы с очередями.— М.: Мир, 1979.
171. Корбут А. А., Финкельштейн Ю. Ю. Приближенные методы дискретного программирования.— Изв. АН СССР, Техническая кибернетика, 1983, № 1, с. 165—176.
172. Кукса А. И. К использованию теории двойственности для решения сетевых задач теории расписаний.— Журнал вычислит. матем. и матем. физики, 1982, 22, № 5, с. 1074—1079.
173. Левин Г. М., Танаев В. С. Декомпозиционные методы оптимизации проектных решений.— Минск: Наука и техника, 1978.
174. Левин М. Ш. Детерминированные задачи планирования при идентичных процессорах и одновременном поступлении заявок.— Изв. АН СССР. Техническая кибернетика, 1982, № 4, с. 51—57.
175. Липаев В. В. Распределение ресурсов в вычислительных системах.— М.: Статистика, 1979.

176. Липаев В. В., Собкин С. С. Эффективность детерминированного планирования вычислительного процесса.— Изв. АН СССР, Техническая кибернетика, 1960, № 1, с. 128—133.
177. Максименков А. В. Формирование расписания с учетом ограничения на интенсивность потребления ресурса.— Кибернетика, 1979, № 6, с. 91—95.
178. Михалевич В. С., Сергиенко И. В. и др. Пакет прикладных программ ДИСПРО, предназначенный для решения задач дискретного программирования.— Кибернетика, 1981, № 3, с. 117—137.
179. Михалевич В. С., Кукса А. И. Методы последовательной оптимизации в дискретных сетевых задачах оптимального распределения ресурсов.— М.: Наука, 1983.
180. Оидаш Й. Алгоритмы распределеиия ресурсов неоднородной многопроцессорной системы.— В кн.: Алгоритмы, математическое обеспечение и архитектура многопроцессорных вычислительных систем. М.: Наука, 1982, с. 320—333.
181. Ондаш Й. Алгоритмы распределения ресурсов однородной многопроцессорной системы.— Там же, с. 292—319.
182. Панфилов И. В., Половко А. М. Вычислительные системы/Под ред. А. М. Половко.— М.: Советское радио, 1980.
183. Пашкеев С. Д., Минязов Р. И. Машинный алгоритм загрузки мультипроцессорной вычислительной системы.— В кн.: Вычислительные средства в технике и системах связи. Вып. 2. М.: Связь, 1977, с. 87—95.
184. Подчасова Т. П., Португал В. М. и др. Эвристические методы календарного планирования.— Киев: Техника, 1980.
185. Сергиенко И. В., Каспшицкая М. Ф. Модели и методы решения иа ЭВМ комбинаторных задач оптимизации.— Киев: Наукова думка, 1981.
186. Серик А. Е. Исследование интервалов очередности для решения задач очередности с ограничениями.— Кибернетика, 1982, № 4, с. 61—65, 73.
187. Танаев В. С., Гордон В. С. О построении расписаний с наименьшим взвешенным числом запаздывающих требований.— Изв. АН БССР. Сер. физ.-матем. наук, 1982, № 5.
188. Танаев В. С., Гордон В. С., Шафранский Я. М. Теория расписаний. Одностадийные системы.— М.: Наука, 1983 (в печати).
189. Трахтеигерц Э. А., Шурайц Ю. М. Минимизация использования вычислительных ресурсов многопроцессорной системы при групповом обслуживаии программ.— Программирование, 1976, № 5, с. 48—51.
190. Шафранский Я. М. Об алгоритме отыскання минимума приоритетопорождающих функций на специальных множествах перестановок. I, II.— Изв. АН БССР. Сер. физ.-мат. наук, 1982, № 3, с. 38—42; 1983, № 1, с. 15—20.
191. Шахбазян К. В., Тушкина Т. А., Сохранская В. С. Статистические испытания различных методов диспетчеризации для многопроцессорных систем.— Программирование, 1976, № 4, с. 91—100.
192. Шахбазян К. В., Лебединская Н. Б. Эффективные методы оптимизации составления расписаний для одной машины (обзор).— Записки научных семинаров ЛОМИ АН СССР, 1981, 111, с. 195—217.
193. Шкурба В. В., Белецкий С. А. Численные методы в

решении задачи балансирования сборочной линии (обзор).— Кибернетика, 1977, № 1, с. 96—108.
194. Шкурба В. В., Селивончик В. М. Расписания, имитационное моделирование и оптимизация.— Кибернетика, 1981, № 1, с. 91—96.
195. Шурайц Ю. М. Оптимальное распределение ресурсов при многократном выполнении комплекса работ.— В кн.: Актуальные вопросы теории и практики управления. М.: Наука, 1977, с. 122—125.
196. Abdel-Waxab H. M., Kameda T. Scheduling to minimize maximum cumulative cost subject to series-parallel constraints.— Oper. Res., 1978, **26**, № 1, p. 141—158.
197. Adorni G., Boccalatte A., Di Manzo M. Evaluation of scheduling algorithms in the multiprocessor environment.— Computer Performance, 1981, 2, № 2, p. 70—76.
198. Adrabinski A., Wodecki M. An algorithm for solving the machine sequencing problem with parallel machines.— Zast. mat., 1979, **16**, № 3, p. 513—541.
199. Arora R. K., Rana S. P., Sharma N. K. On the design of process assigner for distributed computing systems.— Austral. Comput. J., 1981, **13**, № 3, p. 77—82.
200. Baker K. R., Schrage L. E. Finding an optimal sequence by dynamic programming: An extension to precedence-related tasks.— Oper. Res., 1978, **26**, № 1, p. 111—120.
201. Baker K. R., Nuttle H. L. W. Sequencing independent jobs with a single resource.— Nav. Res. Log. Quart., 1980, **27**, № 3, p. 499—510.
202. Blazewicz J. Deadline scheduling of tasks — a survey.— Found. Contr. Eng. (PRL), 1976 (1977), **1**, № 4, p. 203—216.
203. Blazewicz J., Weglarz J. Scheduling under resource constraints-achievements and prospects.— In: 4th Int. Symp. Modell. and Performance Eval. Comput. Syst., Vienna, 1979, **2**, p. 4—22.
204. Brucker P. A linear time algorithm to minimize maximum lateness for the two-machine, unit-time, job-shop, scheduling problem.— Lect. Notes and Inf. Sci., 1982, **38**, p. 566—671.
205. Bruno J., Downey P. Complexity of task sequencing with deadlines, set-up times and changeover costs.— SIAM J. Comput., 1978, 7, № 4, p. 494—504.
206. Bruno J., Jones J. W., So K. Deterministic scheduling with pipelined processors.— IEEE Trans. Comput., 1980, **C-29**, № 4, p. 308—316.
207. Casey L. M. Decentralized scheduling.— Austr. Comput. J., 1981, **13**, № 2, p. 58—63.
208. Cho Y., Sahni S. Preemptive scheduling of independent jobs with release and due times on open flow and job shops.— Oper. Res., 1981, **29**, № 3.
209. Coffman E. G., Garey M. R., Johnson D. S. An application of bin-packing to multiprocessor scheduling.— SIAM J. Comput., 1978, 7, № 1, p. 1—17 (Auerbach Annu. 1979, Best Comput. Pap.— New York, Oxford, 1979, p. 283—301).
210. Coffman E. G., Garey M. R., Johnson D. S., Tarjan R. E. Performance bounds for level-oriented two-dimensional packing algorithms.— SIAM J. Comput., 1980, **9**, № 4, p. 808—826.
211. Davis E., Jaffe J. M. Algorithms for scheduling tasks on unrelated processors.— J. ACM, 1981, **28**, № 4, p. 721—736 (In: MIT

Lab. Comput. Sci. Techn. Memo.— Cambridge, 1979, № 137.— 28 p.).
212. De P., Morton T. E. Scheduling to minimize maximum lateness on unequal parallel processors.— Int. J. Comput. and Oper. Res., 1982, **9**, № 3, p. 221—232.
213. Dessouky M. I., Deogun J. S. Sequencing jobs with unequal ready times to minimize mean flow time.— SIAM J. Comput., 1981, **10**, № 1, p. 192—202.
214. Deterministic and Stochastic Scheduling.— In: Proc. NATO Adv. Study and Res. Inst. Theor. Approaches Scheduling Probl., Durham, 1981/M. A. H. Dempster et al. Dordrecht, 1982.
215. Dutton R. D., Brigham R. C. The complexity of a multiprocessor task assignment problem without deadlines — Theor. Comput. Sci., 1982, **17**, № 2, p. 213—216.
216. Elmaghraby S. A., Eliman A. A. Knapsack-based approaches to the makespan problem on multiple processors.— AIIE Trans. 1980, **12**, № 1, p. 87—96.
217. Erschler J., Fontan G., Merce C., Roubellat F. Applying new dominance concepts to job scheduling optimization.— Eur. J. Oper. Res., 1982, **11**, № 1, p. 60—66.
218. Fernandez E. B., Lang T. Scheduling as graph transformation.— IBM J. Res. and Develop., 1976, **20**, № 6, p. 551—559.
219. French S. Sequencing and scheduling. An introduction to the mathematics of the job-shop.— Chichester: Horwood, 1982.
220. Garey M. R., Graham R. L., Johnson D. S. Performance guarantees for scheduling algorithms.— Oper. Res., 1978, **26**, № 1, p. 3—21.
221. Garey M. R., Johnson D. S., Simons B. B., Tarjan R. E. Scheduling unit-time tasks with arbitrary release times and dead-lines.— SIAM J. Comput., 1981, **10**, № 2, p. 256—269.
222. Gonzalez M. J. Deterministic processor scheduling.— Computing Surveys, 1977, **9**, № 3, p. 173—204.
223. Gonzales T., Johnson D. B. A new algorithm for preemptive scheduling of trees.— J. ACM, 1980, **27**, № 2, p. 287—312-
224. Graham R. L., Lawler E. L., Lenstra J. K., Rinnooy Kan A. H. G. Optimization and approximation in detere ministic sequencing and scheduling: a survey.— Annals of Discrete Math., 1979, **5**, p. 287—326.
225. Jaffe J. M. Efficient scheduling of tasks without full use of processor resources.— Theor. Compt. Sci., 1980, **12**, № 1, p. 1—17 (In: MIT Lab. Comput. Sci. Techn. Memo., 1979, № 122.—40 p.).
226. Kafura D. G., Shen V. Y. Task scheduling on a multiprocessor system with independent memories.— SIAM J. Comput., 1977, **6**, № 1, p. 167—187.
227. King J. R., Spachis A. S. Scheduling: bibliography and review.— Int. J. Phys. Distrib. and Mater. Manag., 1980, **10**, № 3, p. 100—132.
228. Kise H., Ibaraki T., Mine H. Performance analysis of six approximation algorithms for the one-machine maximum lateness scheduling problems with ready times.— J. Oper. Res. Soc. Japan, 1979, **22**, № 3, p. 205—224.
229. Kleinrock L., Nilsson A. On optimal scheduling algorithms for time-shared systems.— J. ACM, 1981, **28**, № 3, p. 477—486.
230. Kunde M. Nonpreemptive *LP*-scheduling on homogeneous multiprocessor systems.— SIAM J. Comput., **10**, № 1, p. 151—173.

231. Kurisu T. Three-machine scheduling problem with precedence constraints.— J. Oper. Res. Soc. Japan, 1977, **20**, № 3, p. 231—242.
232. Lagewegb. J., Lenstra J. K., Rinnooy Kan A. H. G. Minimizing maximum lateness on one machineà computational experience and some applications.— Statistica Neerlandica, 1976, **30**, № 1, p. 25—41.
233. Lawler E. L. Preemptive scheduling of precedence-constrained jobs on parallel machines.— In: Deterministic and Stochastic Scheduling. Proc. NATO Adv. Study and Res. Inst. Theor. Approaches Scheduling Probl., Durham, 1981. Dordrecht, 1982, p. 101—123.
234. Lawler E. L., Lenstra J. K., Rinnooy Kan A. H. G. Recent developments in deterministic sequencing and scheduling: a survey.— In: Deterministic and Stochastic Scheduling. Proc. NATO Adv. Study and Res. Inst. Theor. Approaches Scheduling Probl., Durham, 1981.— Dordrecht, 1982, p. 35—73.
235. Lenstra J. K. Sequencing by enumerative methods.— Amsterdam: Mathematisch Centrum, 1977.
236. Lenstra J. K. Rinnooy Kan A. H. G. Complexity of vehicle routing and scheduling problems.— Networks, **1981**, **11**, p. 221—227.
237. Liu C. L., Liu J. W. S., Liestman A. L. Scheduling with slack time.— Acta Inform., 1982, **17**, № 1, p. 31—41.
238. Liu J. W. S., Liu C. L. Performance analysis of multiprocessor systems containing functionally dedicated processors.— Acta Inform., 1978, **10**, № 1, p. 95—104.
239. Lloyd E. L. Critical path scheduling with resource and processor constraints.— J. ACM, 1982, **29**, № 3, p. 781—811.
240. Martel C. Preemptive scheduling with release times, deadlines, and due times.— J. ACM, 1982, **29**, № 3, p. 812—829.
241. Monma C. L. Sequencing with general precedence constraints.— Discrete Appl. Math., 1981, № 3, p. 137—150.
242. Nakajima K., Hakimi S. L., Lenstra J. K. Complexity results for scheduling tasks in fixed intervals on two types of machines.— SIAM J. Comput., 1982, **11**, № 3, p. 512—520.
243. Panwalkar S. S., Iskander W. A survey of scheduling rules.— Oper. Res., 1977, **25**, № 1, p. 45—61.
244. Pinedo M. On the computational complexity of stochastic scheduling problems.— In: Deterministic and Stochastic Scheduling. Proc. NATO Adv. Study and Res. Inst. Theor. Approaches Scheduling Probl., Durham, 1981. Dordrecht, 1982, p. 355—365.
245. Ramamoorthy C. V., Fox T. F., Li H. F. Scheduling parallel processable tasks for a uniprocessor.— IEEE Trans. Comput., 1976, **C-25**, № 5, p. 485—495.
246. Rao G. S., Stone H. S., Hu T. C. Assignment of tasks in a distributed processor system with limited memory.— IEEE Trans. Comput., 1979, **C-28**, p. 291—299.
247. Rinnooy Kan A. H. G. Machine scheduling problems: Classification, complexity, and computations.— The Hague: Martinus Nijhoff, 1976.
248. Rowicki A. On the optimal preemptive scheduling for multiprocessor system.— Bull. Acad. Pol. Sci. Ser. sci. math., astron. et phys., 1978, **26**, № 7, p. 651—660.
249. Sahni S., Cho Y. Scheduling independent tasks with due times on a uniform processor system.— J. ACM, 1980, **27**, № 3, p. 550—563.

250. Satyanarayanan M. Multiprocessing: an annotated bibliography.— Computer, 1980, 13, № 5, p. 101—116.
251. Schrage L., Baker K. R. Dynamic programming solution of sequencing problems with precedence constraints.— Oper. Res., 1978, 26, № 3, p. 444—449.
252 Sethi R. On the complexity of mean flow time scheduling.— Math. Oper. Res., 1977, 2, № 4, p. 320—330.
253. Sidney J. B. Optimal-single-machine scheduling with earlines and tardines penalties.— Oper. Res., 1977, 25, № 1, p. 62—69.
254. Simons B. On scheduling with release times and deadlines.— In: Deterministic and Stochastic Scheduling. Proc. NATO Adv. Study and Res. Inst. Theor. Approaches Scheduling Probl., Durham, 1981.— Dordrecht, 1982, p. 75—88.
255. Steinacker M., Hennings D., Schindler S. Scheduling two-processor systems.— In: Proc. Int. Conf. Parallel Processing, 1977. New York: IEEE, 1977, p. 31—37.
256. Stone H. S. Critical load factors in two-processor distributed systems.— IEEE Trans. Software Eng., 1978, SE-4, № 3, p. 254—258.
257. Szwarc W. Dominance conditions for the three machine flowshop problem.— Oper. Res., 1978, 26, p. 203—206.
258. Van Wassenhove L. N., Baker K. R. A bicriterion approach to time/cost trade-offs in sequencing.— Eur. J. Oper. Res., 1982, 11, № 1, p. 48—52.
259. Weglarz J. Multiprocessor scheduling with memory allocation — a deterministic approach.— IEEE Trans. Comput., 1980, C-29, № 8, p. 703—709.
260. Yang C.-C. Fast algorithms for bounding the performance of multiprocessor systems.— In: Proc. Int. Conf. Parallel Processing, 1976 New York: IEEE, 1976, p. 73—82.

Бруно Дж. Л., Грэхем Р. Л., Коглер В. Г., Коффман Э. Г., мл., Сети Р., Ульман Дж. Д., Штиглиц К.

ТЕОРИЯ РАСПИСАНИЙ И ВЫЧИСЛИТЕЛЬНЫЕ МАШИНЫ

(Серия: «Экономико-математическая библиотека»)

Редактор *А. Д. Вайнштейн*
Техн. редактор *И. Ш. Аксельрод*
Корректоры *Т. С. Плетнева, Н. Д. Дорохова*

ИБ № 12242

Сдано в набор 23.10.83. Подписано к печати 01.02.84. Формат 84×108¹/₃₂. Бумага тип. № 1. Литературная гарнитура. Высокая печать. Условн. печ. л. 17,64. Условн. кр.-отт. 17,64. Уч.-изд. л. 19,25. Тираж 7900 экз. Заказ № 2334. Цена 1 р. 70 к.

Издательство «Наука»

Главная редакция физико-математической литературы

117071, Москва, В-71, Ленинский проспект, 15

Ордена Октябрьской Революции и ордена Трудового Красного Знамени Первая Образцовая типография имени А. А. Жданова Союзполиграфпрома при Государственном комитете СССР по делам издательств, полиграфии и книжной торговли. Москва, М-54, Валовая, 28

Отпечатано во 2-й типографии издательства «Наука».
121099, Москва, Г-99, Шубинский пер., 10. Зак. 3746

ИЗДАТЕЛЬСТВО «НАУКА»
ГЛАВНАЯ РЕДАКЦИЯ
ФИЗИКО-МАТЕМАТИЧЕСКОЙ
ЛИТЕРАТУРЫ
117071, Москва, В-71, Ленинский проспект, 15

СЕРИЯ: «ЭКОНОМИКО-МАТЕМАТИЧЕСКАЯ БИБЛИОТЕКА»

Готовятся к печати:

Поспелов Г. С., Ириков В. А., Курилов А. Е. Процедуры и алгоритмы формирования комплексных программ.

Танаев В. С., Гордон В. С., Шафранский Я. М. Теория расписаний. Одностадийные системы.

Маленво Э. Лекции по микроэкономическому анализу: Пер. с франц./ Под ред. К. А. Багриновского.

Эльстер К. и др. Введение в нелинейную оптимизацию: Пер. с нем./ Под ред. И. И. Еремина.