МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
им. М.В. ЛОМОНОСОВА

Лазарев Александр Алексеевич
Гафаров Евгений Рашидович

ТЕОРИЯ РАСПИСАНИЙ

ЗАДАЧИ И АЛГОРИТМЫ

МОСКВА – 2011

УДК 519.854.2

ББК

Л

Рецензенты: д.т.н., профессор В.Н. Бурков;
д.ф.-м.н., профессор В.В. Шкурба

Под редакцией академика РАН С.Н. Васильева

В данном учебном пособии приводятся базовые сведения о специальном разделе дискретной математики - Теории расписаний. Описаны этапы становления теории, свойства и классификации задач теории расписаний, методы их решения. На примерах классических задач представлены приемы доказательства их трудоемкости и алгоритмы решения.

Учебное пособие основано на курсе лекций, читаемых в МФТИ, МГУ и ВШЭ, и предназначено для студентов и преподавателей вузов математических специальностей, специалистов в области управления и практиков, сталкивающихся с задачами объемно-календарного планирования.

Оглавление

Предисловие редактора

Уважаемый читатель!

Книга, которую Вы держите в руках, – одно из первых учебных пособий, написанных на кафедре физико-математических методов управления МГУ, созданной в 2009 году на физическом факультете университета. На кафедре преподаются методы решения разнообразных актуальных задач управления системами физико-технической, организационно-экономической и другой природы.

Решение подобных задач планирования и управления стало особенно актуальным в 20-м веке. Именно в это время формируется новая область математики – исследование операций, а также смежные с ней дисциплины – теория массового обслуживания, теория расписаний, теория автоматического управления, теория оптимального управления, теория многокритериального принятия решений.

Этот учебник посвящен теории расписаний – одной из самых популярных с теоретической и практической точек зрения области исследования операций. Задачи теории расписаний, разумеется, связаны с построением расписаний, т.е. с упорядочиванием некоторых работ (операций) по времени и/или по исполнителям (приборам). При этом необходимо учитывать ограничения на последовательность выполнения работ, ограничения, связанные с исполнителями, и т.п. Цель решения таких задач – построение допустимых расписаний, при котором все ограничения соблюдены, или, что является более сложным, – нахождение оптимального допустимого расписания по тому или иному критерию оптимальности. Например, построение оптимального расписания по быстродействию (т.е. с минимизацией общего времени выполнения всех работ), расписания с минимальными финансовыми затратами и т.п.

Решение задач теории расписаний усложняется тем фактом, что большинство из них являются NP-трудными, т.е. алгоритмы их решения, реализованные на ЭВМ, могут требовать неприемлемо большое время работы для решения практических задач "большой размерности".

В этом учебнике излагаются предмет исследования теории расписаний и история ее становления как отдельной дисциплины. Приводятся основная терминология и классификация задач, применяемые в рамках теории расписаний.

Для облегчения понимания связи теории расписаний с другими разделами математики во второй главе с более общих позиций рассказывается о том, как ставятся и решаются задачи дискретной математики, к которым относятся и задачи теории расписаний.

В последующих главах рассматриваются классические задачи построения расписаний – для одного и нескольких приборов. Кроме того, в развитие этих постановок излагается проблематика управления проектами в части построения расписания для проекта.

С учетом сведений, приведенных во второй главе, данная книга может использоваться автономно, т.е. без привлечения других источников.

Книга рекомендуется как основной учебник к читаемому на кафедре курсу "Теория расписаний".

Заведующий кафедрой
физико-математических методов управления МГУ,
академик С.Н. Васильев,
февраль 2011

Введение

На протяжении всей жизни мы сталкиваемся с различными расписаниями: расписанием учебных занятий, транспорта, телевизионных передач. Каждый из нас составляет разумное на его взгляд расписание выполнения повседневных работ. Для решения бытовых вопросов применение интуитивного подхода оказывается достаточным.Часто мы планируем наши действия в порядке возрастания крайних сроков исполнения работ. Например, студенты во время экзаменационной сессии учат предмет с наименьшим директивным сроком (ближайший по дате сдачи экзамен), тем самым они минимизируют максимальное временно́е смещение. Так как лучше равномерно подготовиться к 3-м экзаменам и получить три четверки, чем неравномерно и получить две пятерки и тройку – можно не получить стипендию.

Но перед обществом стоят и более трудные задачи. Развивающаяся стремительными темпами автоматизация производства и неуклонно увеличивающиеся его масштабы требуют разработки алгоритмов составления расписаний, в которых учтены разнообразные ограничения.

Теория расписаний является одним из разделов исследования операций. Данное направление в науке, берёт свое начало с известной работы Генри Гантта 1903 г. (Gantt H.L. [63]), предложившего то, что сегодня называют диаграммами Гантта, которые встречаются во многих работах по теории расписаний, в том числе и в данной книге. Термин “теория расписаний” предложил Р. Беллман в 1956 году [39]. Методы и алгоритмы решения задач теории расписаний применяются для решения задач комбинаторной оптимизации.

С 50-х годов 20-го века началось активное теоретическое исследование задач теории расписаний, следует отметить работы Джонсона (Johnson [69]), Джексона (Jackson [67]) и Смита (Smith [98]), а также монографии [23,

48].

Одним из главных вопросов нового направления была классификация задач и установление их сложности. Наиболее устоявшаяся на нынешний день классификация задач теории расписаний была предложена Грэхэмом и др. (Graham at al. [65]). Достаточно полные обзоры по задачам теории расписаний и их сложности представлены в работах Гэри и Джонсона (Garey, Johnson [6]), Ленстры и др. (Lenstra et al. [82]), Лоулера и др. (Lawler et al. [80], Танаева и др. [23, 24, 25, 26], Брукера и др. [43, 41].

Подавляющее большинство исследованных задач теории расписаний являются NP-трудными. Несмотря на это, практика требует решения таких задач. Для этого существует несколько подходов.

Первым подходом является разработка полиномиальных эвристических алгоритмов. Для некоторых эвристических алгоритмов известны оценки погрешности получаемого решения. Такие алгоритмы называются *приближёнными* [10, 11, 12]. Существуют приближённые алгоритмы, гарантирующие как относительную погрешность [9, 96], так и абсолютную погрешность [21]. Некоторые NP-трудные задачи допускают существование так называемой *аппроксимационной схемы*. В рамках данной схемы можно найти приближённое решение с относительной погрешностью не более любого заданного значения $\varepsilon > 0$ за время, полиномиально зависящее от $1/\varepsilon$ и от размера входной информации задачи,— вполне полиномиальная аппроксимационная схема ($FPTAS$). Для задач теории расписаний такие схемы разработали, например, Ковалёв [13], Алон и др. (Alon et al. [30]), Мастролилли (Mastrolilli [83]), Севастьянов и Вёгингер (Sevastianov, Woeginger [97]. Для задач, не имеющих аппроксимационной схемы, большое значение имеет установление предельного значения ε, для которого возможно нахождения ε-приближённого решения за полиномиальное время,— полиномиальная аппроксимационная схема ($PTAS$).

В настоящий момент широкое распространение имеют метаэвристические алгоритмы, которые находят “хорошее” решение, близкое к оптимальному, за приемлемое время. Недостатком таких алгоритмов является отсутствие оценок качества полученного решения. Неизвестно, насколько решение отличается от оптимального в наихудшем случае.

Точным методам решения NP-трудных задач также уделено немалое

внимание в работах по теории расписаний. Наибольшее распространение получили методы сокращённого перебора, называемые методами ветвей и границ [22, 40, 45, 75]. Для сокращения перебора вычисляются нижние оценки целевой функции (в случае её минимизации) и используются комбинаторные свойства задач. Также для решения задач теории расписаний широко применяется метод динамического программирования [1, 6, 19, 43, 71].

Часто задачи теории расписаний могут быть сформулированы как задачи целочисленного линейного программирования. Решению таких задач посвящены, например, работы [33, 34, 93, 99].

В последнее время широкое распространение получил метод программирования в ограничениях (ПвО, в англоязычной литературе – Constraint Programming). Одной из областей его успешного применения является теория расписаний [35].

Некоторые сложные задачи теории расписаний могут быть оптимально решены с помощью алгоритмов, использующих элементы сразу нескольких методов. Одно из их названий — "гибридные алгоритмы" [68]. Данное направление, на наш взгляд, является одним из перспективных.

Материал этого учебника разделен на главы следующим образом. В первой главе даются общие сведения о теории расписаний, истории ее возникновения. Приводится используемые терминология, обозначения, классификация задач. Во второй главе кратко описаны классические задачи дискретной оптимизации, методы их решения и классификация задач по сложности. Эти сведения необходимы для понимания последующего материала.

В третьей главе рассматриваются одноприборные задачи теории расписаний, а в четвертой – многоприборные. В пятой главе представлены некоторые результаты по задачам с отношениями предшествования и ресурсными ограничениями.

Основное внимание в данном учебнике уделено алгоритмам решения задач и их принадлежности классам сложности.

Алгоритмы решения задач можно найти на страницах 54, 55, 74, 75, 90, 93, 101, 108, 127, 150.

Как доказать NP-трудность задачи – страницы 92, 113, 117, 117, 119, 124, 140, 158, 183.

Полная полиномиальная аппроксимационная схема ($FPTAS$) представлена в параграфе 3.3.2. Метаэвристические алгоритмы можно найти в параграфах 3.3.3 и 5.7. Как проводить эксперименты можно узнать в параграфе 3.3.5.

Благодарности

Считаем своим долгом выразить благодарность: В.С. Танаеву, С.Н. Васильеву, Ю.И. Журавлёву, В.В. Шкурбе, В.К. Леонтьеву, И.Х. Сигалу, Ю.А. Флёрову, Л.С. Иртышёвой, К.В. Рудакову, В.Н. Буркову, Ф.Т. Алескерову, Д.А. Новикову, С.В. Севастьянову, М.Я. Ковалёву, Я.М. Шафранскому, Ю.Н. Сотскову, Н.Н. Кузюрину, В.В. Шкурбе, А.А. Сапоженко, М.Н. Вялому, С.П. Тарасову, Р.Р. Садыкову, А.Г. Кварацхелия, Р.Р. Сираеву, С.А. Скиндереву, А.Н. Черных, Я.И. Заботину, О.Н. Шульгиной, Р.Ф. Хабибуллину, А.А. Корбуту, П. Брукеру, Ф. Вернеру (Германия), Ф. Баптисте (Франция), В. Шварцу (США) за помощь в работе и обсуждение полученных результатов.

Авторы выражают благодарность Е.Г. Лазаревой и Т.С. Ефимовой за большую помощь в подготовке и оформлении материалов книги и постоянную поддержку.

Работа выполнена при частичной финансовой поддержке научных программ 15 и 29 РАН, а также фондов: РФФИ (Россия), DAAD (Германия), CNRS (Франция), NSC (США).

Глава 1

Общие сведения о теории расписаний

1.1 Предмет теории расписаний

Люди на протяжении всей своей жизни сталкиваются с необходимостью составления расписаний. Каждый из нас занимается планированием личного времени. Мы знаем **что** и к **какому сроку** нужно выполнить, а также имеем представление, сколько времени потребуется на каждое дело. Исходя из этих данных мы составляем расписание, согласуя наши дела по времени. Чаще всего составление личного расписания не вызывает сложности. Практически все люди при этом руководствуются "похожими алгоритмами", стремясь сделать **все** дела и **вовремя**.

Часто мы планируем наши действия в порядке возрастания крайних сроков исполнения работ (дел). Например, студенты во время экзаменационной сессии учат предмет с наименьшим директивным сроком (ближайший по дате сдачи экзамен). Если первый экзамен необходимо сдать 12-го января, второй 15-го, а третий 19-го, при этом на каждый экзамен необходимо 3 дня, то большинство студентов составят такое расписание: "с 9-го по 11-ое я готовлюсь к первому экзамену, с 12-го по 14-ое ко второму и с 16-го по 18-ое к третьему".

Сложности при составлении расписаний появляются тогда, когда работ становится много, нужно учесть множество дополнительных условий и/или составить расписание не для одного человека, а для целого коллектива. Представьте себе сотни работ и десятки исполнителей, для которых необходимо составить расписание.

В процессе решения таких задач, были выработаны общие рекомендации, принципы и методики составления расписаний. В последствии подобные задачи стали исследоваться в рамках специального раздела науки — **Теории Расписаний** (далее **ТР**).

Эта наука появилась не на пустом месте, а возникла на стыке других областей научного знания. Прежде чем дать определение, что такое **Теория Расписаний**, опишем одну важную область математики.

Исследование операций *(ИО) – научный метод выработки количественно обоснованных рекомендаций по принятию решений. Важность количественного фактора в ИО и целенаправленность сформулированных рекомендаций позволяют определить ИО как теорию принятия оптимальных решений. ИО способствует превращению искусства принятия решений в математическую дисциплину. Термин "ИО" возник в результате буквального перевода выражения "operation research", введенного в конце 30-х годов 20-го века как условное наименование одного из подразделений британских ВВС, занимающегося вопросами использования радиолокационных установок в общей системе обороны. Первоначально ИО было связано с решением задач военного содержания, но уже с конца 40-х годов прошлого века оно используется для решения технических, технико-экономических задач, а также задач управления на различных уровнях.*

Теперь мы можем дать определение **Теории Расписаний**.

Теория расписаний – это раздел исследования операций, в котором строятся и анализируются математические модели[1] календарного планирования (т.е. упорядочивания во времени) различных целенаправленных действий с учетом целевой функции и различных ограничений.

Задачи составления расписаний возникают в частности:

- на производстве, когда нужно упорядочить отдельные операции по исполнителям (цеха, станки) и по времени;

[1]Построить математическую модель – значит описать с использованием математического аппарата изучаемые процесс или явление.

- на транспорте при составлении расписания движения поездов, самолетов, общественного городского транспорта;
- при планировании занятий в учебных заведениях;
- при планировании занятости персонала, например, дежурства врачей;
- при выполнении сложных продолжительных проектов строительства зданий, кораблей и т.п.;
- при планировании проведения спортивных мероприятий;
- в компьютерных сетях при планировании очередности передачи пакетов информации и т.д.

Содержательно многие задачи **ТР** являются оптимизационными, т.е. состоят в выборе (нахождении) среди множества допустимых расписаний (расписаний, допускаемых условиями задачи) тех решений, на которых достигается "оптимальное" значение целевой функции. Обычно под "оптимальностью" понимается минимальное или максимальное значение некоторой целевой функции. *Допустимость* расписания понимается в смысле его осуществимости, а *оптимальность* — в смысле его целесообразности.

Пример. *Необходимо построить дом как можно быстрее, при этом последовательность работ должна быть соблюдена. Для данной задачи допустимое расписание то, при котором будет соблюдена последовательность работ, а оптимальное расписание – это допустимое расписание, при котором дом будет построен в минимальные сроки.*

Другой тип задач заключается в поиске допустимого расписания, удовлетворяющего всем условиям. Приведем примеры обоих типов задач.

Задача нахождения допустимого расписания

На одном процессоре требуется выполнить множество $N = \{1, 2, \ldots, n\}$ заданий. Для каждого задания $j \in N$ определены длительность выполнения $p_j > 0$, время поступления задания на процессор $r_j \geq 0$ и крайний директивный срок $D_j > 0$, к которому задание должно быть выполнено. Процессор готов к выполнению заданий с момента времени 0 и

может выполнять одновременно только одно задание. Прерывания при выполнении любого задания запрещены. Необходимо построить допустимое расписание выполнения заданий, при котором все условия задачи соблюдены. То есть необходимо для каждого требования $j \in N$ определить момент начала выполнения S_j такой, что $S_j \geq r_j$ и момент окончания выполнения $C_j = S_j + p_j \leq D_j$. Причем если $S_j < S_i$, то $S_j + p_j \leq S_i$, где S_i – момент начала выполнения другого задания $i \in N, i \neq j$.

В задачах **TP** время задается в условных единицах. Параметры p_j, r_j, D_j, C_j, S_j могут измеряться в минутах, часах, днях и т.п. С точки зрения вычислительного процесса два примера – $p_1 = 2$ минуты, $p_2 = 3$ минуты, $r_1 = 0$ – нулевая минута, $r_2 = 1$ – первая минута и, пример, где $p_1 = 2$ часа, $p_2 = 3$ часа, $r_1 = 0$ – нулевой час, $r_2 = 1$ – первый час, – являются идентичными, поэтому единицу измерения времени при определении задач опускают.

Задача нахождения оптимального расписания

Модифицируем предыдущую задачу следующим образом. Пусть крайние сроки D_j не заданы (т.е. $D_j = +\infty,\ j = 1, 2, \ldots, n$). Все остальные условия остаются теми же. Обозначим C_j – момент окончания выполнения задания j, т.е. $C_j = S_j + p_j$, где S_j – момент начала выполнения задания j. Необходимо построить допустимое расписание, при котором значение функции $\sum_{j=1}^{n} C_j$ будет минимальным.

Теперь с оглядкой на две приведенные задачи можно дать следующие два определения.

Определение 1 *Задача, в которой все входные данные полностью определены, называется* **индивидуальной задачей**.

Определение 2 Массовая задача – *бесконечное множество индивидуальных задач.*

В дальнейшем мы будем называть массовую задачу просто *задачей* (например, “Задача коммивояжера”), а индивидуальную задачу – *примером.*

Таблица 1.1: Пример. Исходные данные, допустимое и оптимальное расписания

j	1	2	3	4
p_j	6	2	2	3
r_j	0	1	2	11
D_j	7	10	9	15
Допустимое расписание. Время начала выполнения S_j	0	8	6	11
Допустимое расписание. Время окончания выполнения C_j	6	10	8	14
Оптимальное расписание. Время начала выполнения S_j	5	1	3	11
Оптимальное расписание. Время окончания выполнения C_j	11	3	5	14

В таблице 1.1 представлены допустимое и оптимальное расписания для обеих задач на одном и том же примере. Схематично полученные расписания изображены на рисунке 1.1. На этом рисунке расписания представлены в виде набора прямоугольников, расположенных вдоль виртуальной временной оси t. Каждый прямоугольник соответствует некоторому заданию, а длина прямоугольников соответствует продолжительности выполнения заданий, а их расположение на оси указывает время начала и окончания выполнения каждого задания.

Для первой задачи существует единственное допустимое расписание, при котором порядок выполнения заданий определен следующим образом: $(1, 3, 2, 4)$. Сначала выполняется первое задание, потом третье, второе и четвертое. Очевидно, что для данного примера любой другой порядок выполнения заданий недопустим. Например, при порядке обслуживания $(2, 1, 3, 4)$ будет нарушено условие $C_1 \leq D_1$, где $D_1 = 7$ и $C_1 = 9$.

Для второй задачи схематично представлено оптимальное расписание. На этом же рисунке для второй задачи изображено допустимое (в терминах второй задачи), но неоптимальное расписание, заданное порядком $(1, 2, 3, 4)$. Легко убедиться, что при оптимальном расписании, заданном порядком обслуживания $(2, 3, 1, 4)$ имеем $\sum_{j=1}^{4} C_j = 11 + 3 + 5 + 14 = 33$, а при расписании, заданном как $(1, 2, 3, 4)$, имеем $\sum_{j=1}^{4} C_j = 6 + 8 + 10 + 14 = 38$.

Задачи **TP** (как задачи раздела Исследования Операций) обладают рядом черт, обуславливающих методику их составления и решения. Во-первых, даже для простых параметрических задач не удается предста-

Рис. 1.1: Пример. Графическое представление расписаний.

вить решения в виде аналитического выражения от соответствующих параметров (в виде формулы). Поэтому задачи **ТР**, в подавляющем большинстве, не поддаются аналитическому решению и должны решаться численно. Во-вторых, большинство задач **ТР** содержит в своих формулировках большое количество числового материала, не сводящегося к аналитическим выражениям. Поэтому численное решение этих задач, за немногими исключениями, возможно лишь с помощью компьютера.

Для решения задач **ТР** необходимо разработать *алгоритм решения*. То есть последовательность действий, выполняемых компьютером, с помощью которых можно построить искомое расписание (допустимое или оптимальное).

Алгоритм решения задачи – это последовательность действий, с помощью которых можно построить искомое решение для любого примера задачи.

Встречающиеся на практике задачи составления расписаний содержат тысячи, а порой и миллионы заданий. Поэтому основная цель при исследовании моделей (задач) **ТР** – это **построение эффективных, т.е. быстрых, алгоритмов решения**. Решение примера должно быть получено за "разумное" время.

1.1.1 Возникновение и этапы развития теории расписаний

До 20-го века решение **задач ТР** не требовало много времени или большого числа вычислений. В начале прошлого века развитие науки и техники ускорилось. Увеличился темп и обыденной жизни. Задачи составления расписаний оперировали все большим количество взаимосвязанных работ, и упорядочить их во времени становилось все труднее.

К началу 20-го века относятся два показательных примера. В период с 1903 по 1919 гг. американский ученый Генри Гантт публикует ряд научных работ и предлагает новый способ представления расписаний, получивший название "Диаграмма Гантта". Гантт занимался исследованием и улучшением руководства (менеджмента) на промышленных предприятиях (например, в компании по производству хлопчатобумажных тканей и на предприятиях по постройке кораблей). Диаграмма Гантта – это схематичное изображение календарного плана. В ней работы представлены в виде прямоугольников, размещенных вдоль оси времени. Длина прямоугольника соответствует времени, необходимому на выполнение соответствующей работы. На диаграмме указаны как продолжительность работы так и время ее начала и окончания. На рис. 1.2 представлен пример такой диаграммы[2].

Другой показательный пример – изобретение конвейера Генри Фордом. В 1908-м году он организует поточное производство на своем автомобильном заводе по "конвейерному типу". В данной интерпретации конвейер – это способ организации производства каких-либо изделий, при которой:

- процесс производства разделяется на отдельные операции (стадии);
- одновременно в производстве находится несколько изделий, находящиеся на различных стадиях.

Операции при таком способе производства упорядочены. То есть существует очередность операций и нарушать ее нельзя. Например, не имеет смысла прикручивать колеса или вставлять стекла, пока не собрана рама. Еще одна важная особенность такого производства – выполнение

[2]На диаграмме представлена хронология подготовки студентом дипломной работы.

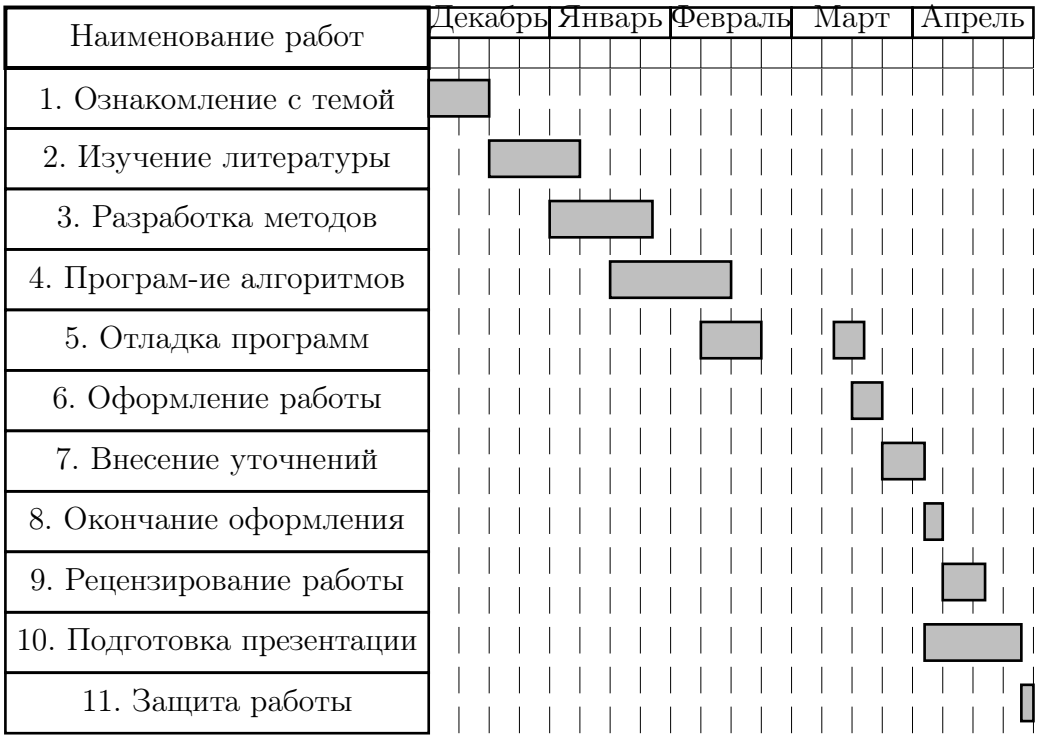

Рис. 1.2: Диаграмма Гантта

независимых операций параллельно. То есть пока собирается кузов, параллельно на другой линии можно собирать двигатель.

Конвейерное производство позволило Форду в 1,5 раза сократить время выпуска автомобилей. Теперь конвейер Форда можно встретить на многих массовых производствах.

Еще один пример уже из советской истории. В 60–70-е годы 20-го века на Новочеркасском электровозостроительном заводе, а также при создании автоматизированной системы управления на Львовском телевизорном заводе, была использована следующая методика управления. В основе ее лежит простая идея, известная в отечественной литературе как система “управления работой на склад”: восстановление запаса деталей, созданного на складе, по мере расходования деталей на сборке – разновидность системы регулирования запасов. “Обнуление” запаса интерпретируется как “точка заказа”, а “партия заказа”, т.е. решение, сколько деталей следует произвести для восстановления запаса, фиксировалось в специальной карточке, которую брал как сменное задание рабочий из ячейки, соответствующей дате рабочего дня. Так создавался необходи-

мый запас деталей к необходимому сроку (т.е. “Вóвремя”) и упрощалось оперативное управление в массовом производстве дискретного типа на предприятиях, взявших за образец новочеркасскую систему. Картотека заданий расставлялась по ячейкам в расчете на начальный запас, соответствующий потребности сборки в определенный стандартный интервал времени, например месяц. Впоследствии идеи этих двух систем были полностью повтороены на японских автомобильных заводах. Теперь всему миру известны “японские” методы “Канбан” (яп. “карточка”) и “Just-in-Time” (“Вóвремя”).

К середине 20-го века выделились еще две области знаний, некоторые задачи которых впоследствии стали частью **ТР** . Это *сетевое планирование* и *теория массового обслуживания*. Чтобы описать эти две теории, дадим несколько определений.

Сетевая модель – информационно-аналитическая модель реализации некоторого комплекса взаимосвязанных работ, рассматриваемая как ориентированный граф без контуров, отображающий естественный порядок выполнения этих работ во времени.

Сетевая модель может содержать некоторые другие характеристики (например, время, стоимость, ресурсы), относящиеся к отдельным работам или комплексу в целом.

Сетевое планирование – совокупность методов анализа, планирования и управления, использующих сетевую модель как основную форму представления информации об управляемом комплексе работ.

Использование сетевого планирования позволяет повысить качество планирования и управления при реализации комплекса работ. Например, дает возможность четко координировать деятельность всех сторон (исполнителей), участвующих в реализации, выделять наиболее важные задачи, определять сроки реализации и т.д.

Методы сетевого планирования являются основой специальных компьютерных программ – систем сетевого планирования и управления, таких как Microsoft Project, Primavera и т.д.

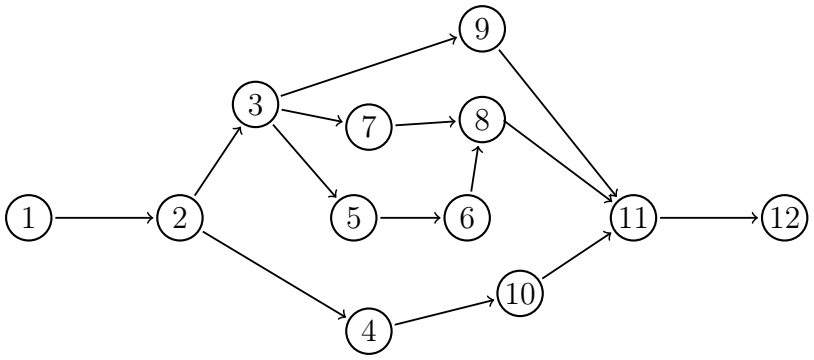

Рис. 1.3: Сетевой график

В 1958-м году по заказу Министерства Обороны США для проекта создания ракетной системы “Поларис” был разработана специальная техника анализа и оценки комплексов взаимосвязанных работ. Проект “Поларис” был ответом на кризис, наступивший после запуска Советским Союзом первого космического спутника. Созданная техника получила название PERT (Project Evaluation and Review Technique). Двумя ее основными элементами было – наглядное представление комплекса работ на бумаге в виде т.н. сетевого графика (см. сетевая модель) и метод расчета критического пути по этому графику. Позднее мы подробнее раскажем о сетевом графике как о способе представления комплекса работ на бумаге (плоскости). Пример сетевого графика представлен на рис. 1.3. Этот сетевой график соответствует проекту подготовки дипломной работы, изображенному на Диаграмме Гантта на рис. 1.2.

В настоящее время сетевое планирование – это раздел **ТР**, в котором рассматриваются задачи составления расписаний для комплексов взаимосвязанных работ, например, при строительстве здания, сборке корабля или самолета, при выполнении крупных проектов (строительство БАМ) и т.п.

Теперь несколько слов об одной смежной области **ТР**. Создание систем телефонной связи и необходимость расчета их пропускной способности послужили в 20-х годах 20-го века стимулом развития *Теории массового обслуживания* (ТМО). В ТМО изучаются потоки требований на обслуживание (заданий), поступающих в системы обслуживания и выходящих из них, длительность ожидания, длины очередей и т.д. Основная цель исследований в ТМО – рационализация системы обслуживания. Яркий

пример задачи ТМО – планирование расписания работы кассиров в продуктовом супермаркете. Если нам известно, как меняется количество покупателей в зависимости от времени суток (вечером покупателей обычно больше), и мы хотим добиться того, чтобы покупатели не стояли в очереди слишком долго, но при этом и кассиры не "простаивали", то мы должны спланировать количество кассиров в каждое время оптимальным образом. На стыке двух областей знаний – ТМО и **ТР** – появились новые задачи, получившие названия "стохастические задачи" **ТР**.

Итак, в первой половине 2-го века был сформулирован ряд практических и теоретических задач составления расписаний. В 1956-м году Ричард Беллман [39] предложил термин "Теория расписаний" для обозначения совокупности данных задач и относящихся к ним научных знаний. В 1967-м году публикуется монография Конвея, Максвелла и Миллера "Теория расписаний". В 1975-м году перевод этой книги на русский язык синхронно выходит с книгой советских авторов В.С. Танаева и В.В. Шкурбы "Введение в теорию расписаний". С этих пор можно считать **ТР** сформировавшейся теорией.

В 70-х годах 20-го века (после выхода работ о теории сложности решения оптимизацонных задач) акцент в исследовании задач **ТР** сместился. Теперь при изучении задач **ТР** исследователь не только строит эффективные алгоритмы решения, но и ищет ответ на вопрос – насколько сложна задача в терминах NP-трудности и полиномиальной разрешимости, и какой быстроты алгоритм "в лучшем случае" можно для этой задачи построить.

1.1.2 Способы представления расписаний

Ранее мы уже познакомились с некоторыми способами представления расписаний. Обозначим главные из них.

- **Табличное представление.** В таблице представлены промежутки времени, в которые выполняются задания, а также их исполнители (номер станка, процессор и т.п.). Пример расписания в виде таблицы представлен в первой главе;

- **Графическое представление.** Например, с помощью Диаграммы Гантта.

- Для некоторых задач **TP** возможно **векторное (перестановочное) представление** расписания. При этом указывается лишь порядок выполнения заданий, например $(2, 3, 4, 1)$. Пример такого порядка приводится в первой главе.

1.2 Классификация задач ТР

Приведем некоторые способы классификации задач **ТР** , а затем подробнее расскажем о некоторых из этих задач.

Способы классификации задач ТР :

- **По типу искомого решения:**
  - *Задачи упорядочивания.* В этих задачах уже задано распределение работ по исполнителям, а также определены все параметры работ (продолжительность выполнения, время поступления и т.д.). Необходимо составить расписание (или порядок) выполнения работ каждым исполнителем;
  - *Задачи согласования.* Основное внимание в этих задачах уделяется выбору продолжительности выполнения работ, времени поступления и другим параметрам;
  - *Задачи распределения* подразумевают поиск оптимального распределения работ по исполнителям.
- **По типу целевой функции:**
  - *Задачи с суммарными критериями оптимизации.* В предыдущей главе мы привели пример такой задачи, в которой необходимо было минимизировать *суммарное* значение моментов окончания обслуживания работ $\sum\limits_{j=1}^{n} C_j$;
  - *Задачи с tiptax (минимаксными) критериями оптимизации.* Отличие этих задач от задач с суммарными критериями заключается в том, что нужно минимизировать не сумму некоторых значений, а лишь максимальное из них. Например, если в упомянутой задаче необходимо *минимизировать максимальое* значение $C_{\max}$, где $C_{\max} = \max\limits_{j \in N} C_j$, то мы получим одну из тривиальных задач этого класса;
  - *Многокритериальные задачи оптимизации.* Если в исследуемых задачах необходимо построить оптимальное решение с точки зрения нескольких целевых установок (функций), то такие

задачи называются многокритериальными. Например, если в упомянутой задаче необходимо не только минимизировать значение $\sum_{j=1}^{n} C_j$, но минимизировать и время простоя процессора (прибора), то это многокритериальная задача[3];

– *Задачи на построение допустимого расписания.* В предыдущем разделе был дан пример такой задачи. Необходимо отметить, что данный класс задач можно свести к оптимизационным задачам, введя специальную функцию штрафа, который нужно минимизировать. Тем не менее, принято выделять такие задачи в отдельный класс.

- **По способу задания входной информации:**
  - *Детерминированные задачи* (**off-line**). Дла таких задач характерно, что все входные данные задачи точно известны, т.е. даны значения всех параметров до начала ее решения;
  - *Динамические задачи* (**on-line**). Для данных задач расписания строятся в режиме реального времени, т.е. перед началом решения задачи мы не знаем значения всех параметров. Расписание строится по частям по мере поступления новой информации. При этом в любой момент может быть понадобиться ответ о качестве построенного "частичного" расписания.
- **По разделу ТР .** В рамках **ТР** принято выделять следующие разделы:
  - *Сетевое планирование или построение расписания для проекта, Project scheduling (PS)*;
  - *Календарное планирование или построение расписания для приборов, Machine scheduling (MS)*;
  - *Составление временны́х таблиц (Time Tabling)*;
  - *Доставка товаров в магазины (Shop-Floor Scheduling)*;
  - *Составление расписаний движения транспортных средств (Transport Scheduling), Циклические расписания для транспортных средств (Vehicle Routing)*;

[3]Некоторые многокритериальные задачи могут быть преобразованы (сведены) к однокритериальным. Другие же могут подразумевать нахождение "Парето-оптимального решения" или нахождение оптимума иерархической целевой функции.

– *Составление расписаний спортивных мероприятий (Sports scheduling).*

Приведенные классификации задач **TP** условны и лишь указывают на некоторые характерные особенности решаемых задач.

1.2.1 Дополнительные условия в задачах TP

В первой главе мы использовали слово "задание" для обозначение тех действий, которые нужно упорядочить во времени. В дальнейшем мы будем использовать другие, общепринятые в TP , слова обозначающие задания: *Требование* или *Работа*.

В задачах **TP** могут быть заданы дополнительные ограничения на параметры требований, порядок обслуживания требований, на приборы. Перечислим некоторые обозначения, которые потребуются нам в дальнейшем. Параметры требований обозначаются следующим образом:

r_j – *момент поступления требования на обслуживание.* Данный параметр определяет момент времени, начиная с которого требование может быть поставлено на обслуживание, но не обязательно его обслуживание начнется в этот момент;

p_j – *продолжительность обслуживания требования.* Параметр определяет время, которое необходимо для обслуживания требования;

d_j – *директивный срок завершения обслуживания.* Данный параметр определяет момент времени, к которому желательно завершить обслуживание требования. Необходимо различать желательный и предельный моменты завершения обслуживания (англ. – due date d_j и deadline D_j, соответственно). Желательный момент завершения обслуживания можно нарушать, хотя при этом накладывается штраф, который влияет на значение целевой функции задачи;

D_j – *предельный срок завершения обслуживания.* Предельный срок завершения нарушать нельзя, и любое расписание, в котором есть завершающееся после своего предельного момента требование, является недопустимым. Примером директивных сроков d_j (due date) и D_j (deadline) могут служить: момент окончания ужина (который

можно и нарушить) и день проведения экзамена (который нарушать крайне нежелательно);

w_j – *вес требования*. Данный параметр характеризует "важность", "значимость" требования и учитывается в целевой функции задачи при подсчете штрафа, который накладывается на обслуживание требования.

Существует специальная система кодирования задач **TP** , с которой мы познакомимся позднее. В этой системе можно встретить следующие обозначения:

pmtn – данная запись означает, что допустимы прерывания в обслуживании требований. То есть можно прервать обслуживание требования, обслужить другое требование и после продолжить обслуживание прерванного;

prec – означает, что между требованиями заданы отношения предшествования. Эта же запись может выглядеть как $tree$, $out-tree$, $in-tree$, $chain$, которые означают, что граф отношений предшествования имеет вид дерева или цепочки;

batch – свидетельствует о том, что рассматривается задача batching, когда требования объединены в группы.

1.2.2 Целевые функции в задачах TP

Обычно задача теории расписаний характеризуется целевой функцией (критерием оптимальности), которую необходимо минимизировать (реже, максимизировать) на множестве допустимых расписаний. Целевая функция в задачах **TP** вычисляется на основе некоторого набора штрафов (штрафных функций), которые возникают при фиксации порядка обслуживания требований в расписании.

В теории расписаний различают следующие основные типы штрафных функций:

- C_j – *момент завершения*, равный моменту окончания обслуживания требования j;

- L_j – *временно́е смещение*, равное величине $C_j - d_j$;
- T_j – *запаздывание*, равное величине $\max\{0, C_j - d_j\}$;
- E_j – *опережение*, равное величине $\max\{0, d_j - C_j\}$;
- U_j – *требование запаздывает*, равно 0, если $C_j \leq d_j$, и 1, – в противном случае.

В задачах, когда задан *вес* требования w_j, указанные выше критерии называются *взвешенными*, а их значение вычисляется путем умножения исходного значения на коэффициент w_j. Например, *взвешенное временно́е запаздывание* $w_j T_j$ вычисляется как $w_j \max\{0, C_j - d_j\}$.

Ранее мы приводили классификацию задач **TP** в зависимости от типа целевой функции. Теперь мы можем привести конкретные примеры. Можно выделить следующие критерии оптимальности:

1. *минимаксные критерии* – в задачах с такими критериями целевая функция представляет собой функцию максимума от значений штрафов требований. Примеры, минимаксных критериев:
   - $C_{\max} \to \min$ – критерий минимизации максимального момента завершения требований, $C_{\max} = \max\limits_{j \in N} C_j$. Задачи с такой целевой функцией называют задачами на *быстродействие*, (makespan – в англоязычной литературе);
   - $L_{\max} \to \min$ – критерий минимизации максимального временно́го смещения $L_{\max} = \max\limits_{j \in N} L_j$.
2. *суммарные критерии* – в задачах с такими критериями целевая функция представляет собой сумму значений штрафов требований. Примеры суммарных критериев:
   - $\sum\limits_{j \in N} C_j \to \min$ – критерий минимизации суммарного времени окончания обслуживания требований;
   - $\sum\limits_{j \in N} T_j \to \min$ – критерий минимизации суммарного запаздывания требований;

- $\sum_{j \in N} U_j \to \min$ – критерий минимизации количества запаздывающих требований.

В **TP** также исследуются задачи на максимизацию аналогичных целевых функций, например, $\sum_{j \in N} T_j \to \max$.

1.2.3 Построение расписания для проекта. Project scheduling (*PS*)

В данном разделе принято выделять одну базовую задачу. Остальные задачи раздела, как правило, либо являются ее частными случаями либо же модификациями. Это задача *построения расписания выполнения работ проекта с учетом отношений предшествования и ограничения на ресурсы* (Resource-Constrained Project Scheduling Problem. *RCPSP*).

Проект – совокупность взаимосвязанных действий, направленных на достижение конкретных целей. К примеру, проект "строительство дома" может состоять из работ "выемка грунта", "укладка фундамента", "возведение стен первого этажа" и т.д. Очевидно, что по технологии "укладка фундамента" производится после "выемки грунта", и т.д. В этом случае говорят, что между работами существует отношение предшествования.

В задаче *RCPSP* необходимо построить оптимальное расписание проекта (выполнения работ проекта) с учетом сетевого графика (отношений предшествования между работами) и с учетом необходимых/доступных ресурсов, при котором будет оптимизирована некоторая целевая функция. Самая популярная целевая функция – общее время выполнения проекта (makespan или $C_{\max}$).

Данные задачи часто возникают на практике. Например, при строительстве того или иного объекта на разных стадиях строительства необходимо разное количество трудовых ресурсов, строительной техники, материалов и т.п. Между отдельными стадиями строительства существуют отношения предшествования обусловленные технологией. Требуется составить расписание выполнения работ, при котором не нарушаются отношения предшествования, ресурсные ограничения, и при этом срок окончания строительства был бы минимален. Алгоритмы решения этой зада-

чи используются в известных программных продуктах Microsoft Project, Spyder Project, Primavera, 1С: Управление Строительной Организацией и т.д.

Постановка задачи *RCPSP* звучит следующим образом.

Дано множество требований $N = \{1, \ldots, n\}$ и K возобновляемых ресурсов $k = 1, \ldots, K$. В каждый момент времени t доступно Q_k единиц ресурса k. Заданы продолжительности обслуживания $p_i \geq 0$ для каждого требования $i = 1, \ldots, n$. Во время обслуживания требования i требуется $q_{ik} \leq Q_k$ единиц ресурса $k = 1, \ldots, K$. После завершения обслуживания требования, освобожденные ресурсы в полном объеме могут быть мгновенно назначены на обслуживание других требований.

Между некоторыми парами требований заданы ограничения предшествования: $i \to j$ означает, что обслуживание требования j начинается не раньше окончания обслуживания требования i.

Обслуживание требований начинается в момент времени $t = 0$. Прерывание при обслуживании требований запрещены.

Необходимо определить моменты времени начала обслуживания требований $S_i,\ i = 1, \ldots, n$, так, чтобы минимизировать время выполнения всего проекта, т.е. минимизировать значение

$$C_{\max} = \max_{i=1,\ldots,n} \{C_i\},$$

где $C_i = S_i + p_i$. При этом должны быть соблюдены следующие ограничения:

1) в каждый момент времени $t \in [0, C_{\max})$ должно выполняться $\sum\limits_{i:=1}^{n} q_{ik}\varphi_i(t) \leq Q_k,\ k = 1, \ldots, K$, где $\varphi_i(t) = 1$, если требование i обслуживается в момент времени t и $\varphi_i(t) = 0$, в противном случае. То есть требования в процессе своего обслуживания должны быть полностью обеспечены ресурсами;

2) не нарушаются отношения предшествования между требованиями, т.е. $S_i + p_i \leq S_j$, если $i \to j$ для $i, j \in N$.

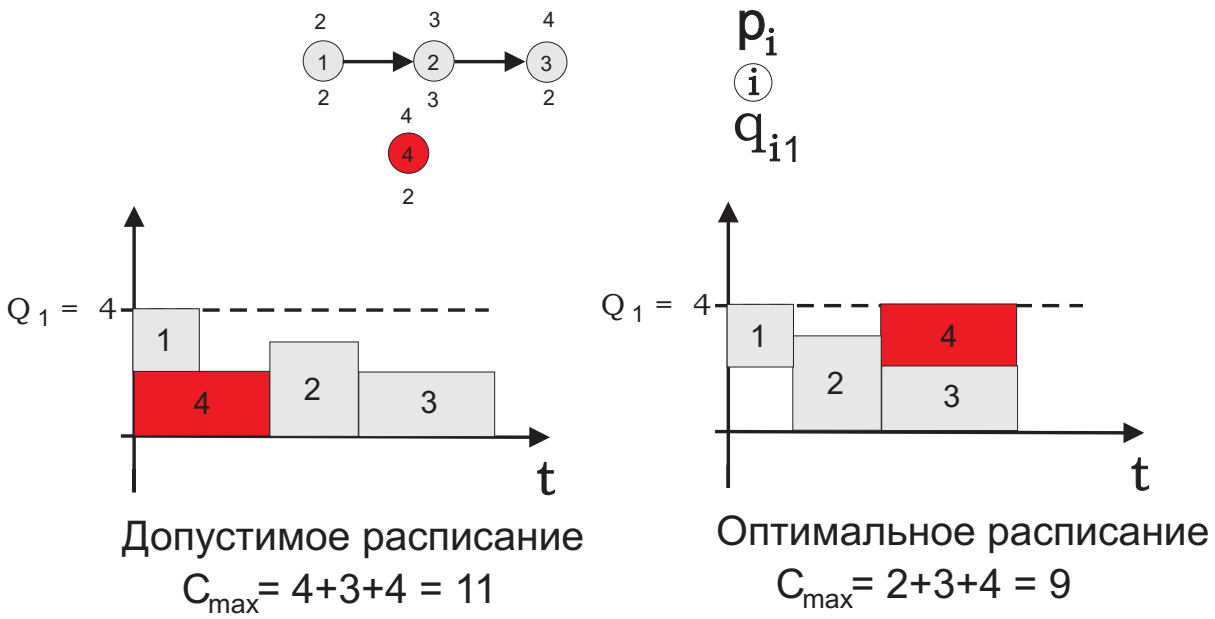

Рис. 1.4: Пример задачи $RCPSP$.

Значение $C_{\max}$ в англоязычной литературе называется *makespan*. К сожалению, нам неизвестен адекватный перевод этого термина на русский язык. На рисунке 1.4 представлен пример задачи с сетевым графиком и двумя расписаниями – оптимальным и неоптимальным.

Требования (работы) в такой задаче могут быть, например, такими: “выемка грунта”, “укладка фундамента” и т.п. В качестве необходимых ресурсов могут выступать: экскаваторы, разнорабочие, каменщики и т.д. Поэтому для выполнения работы “выемка грунта” может потребоваться одновременное участие 3-х экскаваторов, прораба и 7-ми разнорабочих.

Необходимо заметить, что существует класс задач $RCPSP$ с невозобновимыми ресурсами, например, деньги, горюче-смазочные материалы и т.п.

Необходимо заметить, что существует класс задач $RCPSP$ с невозобновимыми ресурсами, например, деньги, горюче-смазочные материалы и т.п.

1.2.4 Построение расписания для приборов. Machine scheduling (MS)

В отличие от Project Scheduling, где для выполнения одной работы требуется одновременное участие нескольких исполнителей, в задачах для приборов каждое требование обычно выполняется (обслуживается) единовременно только на одном приборе (машине).

Для задач MS исполнителями являются *Приборы*, *Машины* или *Процессоры*. Если не приводятся уточнения, эти три термина считаются эквивалентными.

Задачи для одного прибора

С примерами задач для одного прибора мы познакомились в предыдущих разделах. Их характерной особенностью является то, что одновременно прибор может обслуживать (выполнять) только одно требование (задание). Несмотря на кажущуюся простоту в определении, решение одноприборных задач представляется трудной, но важной вычислительной проблемой. Алгоритмы их решения используются для решения более сложных задач, поэтому одноприборные задачи называются фундаментальными.

Задачи для параллельных приборов

Для этих задач вместо одного прибора доступно m приборов M_1, M_2,..., M_m. Между требованиями могут быть заданы отношения предшествования. Каждое требование может выполняться на любом приборе. Если приборы идентичны, то время обслуживания p_j требования j не зависит от выбора машины, на которой требование будет обслужено. Эта задача соответствует частному случаю задачи $RCPSP$, где $K = 1$, $Q_1 = m$ и необходимое количество ресурса $q_{j1} = 1$, для всех требований $j \in N$.

Пример. Рассмотрим пример на рис. 1.5. Дано $n = 8$ требований и $m = 3$ приборов, времена обслуживания $p_1 = 1$, $p_2 = 3$, $p_3 = 4$, $p_4 = 2$, $p_5 = 2$, $p_6 = 3$, $p_7 = 1$, $p_8 = 5$. Допустимое расписание, для которого $C_{\max} = 9$, представлено на том же рисунке.

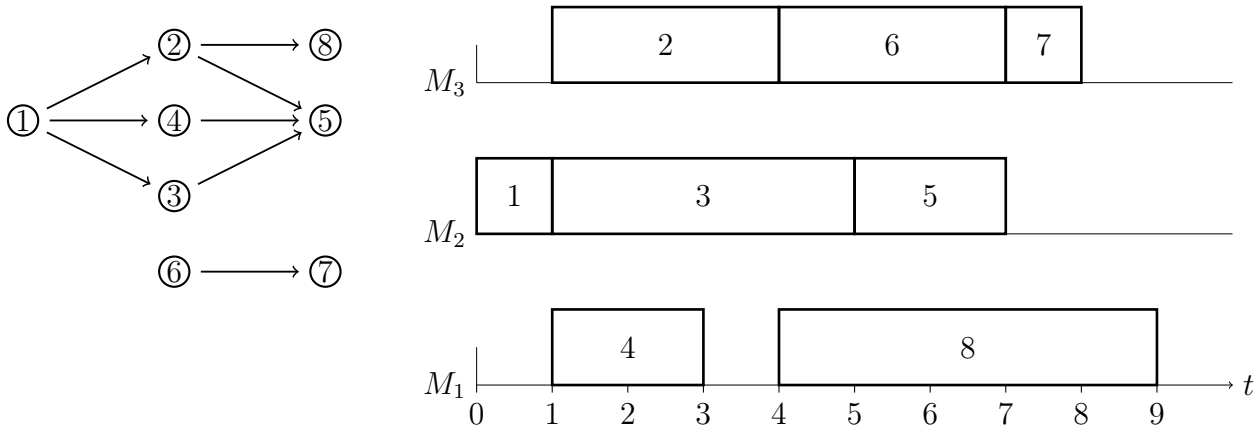

Рис. 1.5: Расписание для идентичных машин

Помимо идентичных приборов, могут рассматриваться приборы с разной производительностью. Для каждой работы j и прибора k может быть задано свое время p_{jk} обслуживания требования j на приборе k.

Задачи Цеха (Shop Scheduling)

В этих задачах каждое требование состоит из операций, выполнение которых может назначаться только на определенные приборы (машины). В общем случае дано m приборов $M_1, M_2, \ldots, M_m$ и каждое требование j содержит операции $O_{1j}, \ldots, O_{n_j j}$. Между операциями могут быть заданы отношения предшествования (маршрут обработки детали). Две операции одного и того же требования не могут выполняться одновременно, и каждый прибор может выполнять единовременно только одну операцию. Время выполнения операции O_{ij} равно p_{ij}, и она может выполняться на машине $\mu_{ij} \in \{M_1, M_2, \ldots, M_m\}$. Данная задача также может быть преобразована (сведена) в задачу $RCPSP$.

Далее мы опишем важные частные случаи задачи цеха.

Job-shop

Для данного случая заданы отношения предшествования между операциями вида $O_{1j} \to O_{2j} \to \cdots \to O_{n_j j}$. При этом нет отношений предшествования между отдельными требованиями. Количество операций у

разных требований может быть различным.

Flow-shop

Для данного частного случая задачи цеха каждая работа состоит из одних и тех же операций, т.е. $n_j = m, \forall j \in N$, а также задана машина, на которой обслуживаются операции, т.е. $\mu_{ij} = M_i$, $i = 1, \ldots, m$, $j = 1, \ldots, n$. Тогда расписание для каждого прибора задается вектором – порядком обслуживания операций, относящихся к разным работам. В русскоязычной литературе данные задачи порой называют задачами конвейерного типа.

Open-shop

Задачи этого типа имеют такую же постановку, как и *Flow-shop* задачи. Единственное отличие – отсутствие отношений предшествования между операциями. То есть количество операций у каждого требования равно m, но порядок их выполнения может отличаться для разных требований.

Прочие задачи Machine scheduling

Особняком стоят задачи, в которых на приборы налагаются специальные ограничения.

Например, в задачах batching (“группирования” или “партий”) один прибор может обслуживать одновременно несколько требований. При этом все требования из одной и той же “партии” имеют одно и тоже время начала обслуживания и одно и тоже время окончания.

В мультипроцессорных задачах обслуживание требования (операции) может производиться одновременно несколькими процессорами (приборами).

1.2.5 Система обозначений для задач Machine Scheduling

В разделе Machine Scheduling для краткого обозначения задач принята специальная система обозначений. Эта система позволяет использовать не громоздкие названия задач (например, "Минимизация взвешенного суммарного запаздывания для одного прибора с разным временем поступления и одинаковой продолжительностью обслуживания требований"), а их краткое обозначение (соответственно, $1|p_j = p, r_j| \sum w_j T_j$). Подробнее расскажем об этой системе обозначений, т.к. она повсеместно используется в публикациях по **TP** .

Грэхем, Лаулер, Ленстра и Ринной Кан (Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G) в 1979-м году [65] для задач Machine Scheduling предложили трехпозиционную систему обозначений вида $\alpha|\beta|\gamma$, общепринятую на данный момент. Поле α описывает характеристики задачи, связанные с приборами, и содержит всего одну запись. Значения в поле β уточняют характеристики обслуживания и ограничения, накладываемые на процесс обслуживания требований. Количество записей в данном поле может быть произвольным (пустое значение, одна запись, несколько записей). Поле γ описывает целевую функцию задачи, значение которой необходимо минимизировать (или максимизировать), и обычно содержит всего одну запись.

В поле α допустимы следующие значения:

- 1 – *задача для одного прибора*;
- Pm – *идентичные параллельные приборы.* Количество идентичных приборов равно m, а P расшифровывается как *parallel*, то есть параллельные или идентичные приборы;
- Qm –*параллельные приборы с различной производительностью*;
- Fm –*системы типа Flow-Shop*;
- Om –*системы типа Open-Shop*;
- Jm –*системы типа Job-Shop.*

В этом поле могут быть указаны также другие записи, например (sa) или (nd), поясняющие характер обслуживания требований, о которых будет рассказано позже.

Ограничения и условия, накладываемые на обслуживание требований, перечисляются в поле β с помощью перечисления одной или нескольких записей:

- r_j – *моменты поступления (release dates)*. Если данное значение указано в поле β, то обслуживание требования j не может быть начато ранее его момента поступления r_j. Если r_j отсутствует в поле β, то предполагается, что все требования поступают на обслуживание одновременно в момент времени $t = 0$;
- D_j – *предельные сроки завершения обслуживания требований*;
- *pmnt* – *допустимы прерывания (preemption)*. Если этот параметр опущен, то прерывания обслуживания требований запрещены;
- *prec* – *отношения предшествования (precedence relations)*. Вместо этой записи в обозначениях задач можно встретить записи $tree$, $in - tree$, $out - tree$ или $chain$ То есть отношения предшествования заданы в виде: дерева; входящего, выходящего дерева; цепочек;
- $batch(b)$ – эта запись означает, что рассматривается задача, где требования обслуживаются партиями. То есть речь идет о задачах типа $batching$.

В поле β могут быть указаны и другие понятные значения. Например, запись $p_j = p$ означает, что для всех требований задана одна и та же продолжительность обслуживания (константа p).

В поле γ указывается целевая функция. Классические целевые функции были перечислены выше.

Согласно этой системе обозначений запись $F2|r_j|C_{\max}$, например, означает задачу “Минимизация общего времени обслуживания требований для системы Flow-shop с двумя приборами при неодновременном поступлении требований”. Думаем, читателю не составит труда разобраться и в записях вида: $1|p_j = p, r_j| \sum w_j T_j$ или $Pm|r_j, pmtn| \sum C_j$.

1.2.6 Составление временны́х таблиц (Time Tabling)

Каждый из нас сталкивался с учебным расписанием в школе или в ВУЗе. Чаще всего учебное расписание для группы студентов представляется в виде таблицы (поэтому этот раздел **TP** называется "*Time Tabling*"), в которой на пересечении строк (дни недели, время лекций) и столбцов (номер группы) указан предмет и номер аудитории, в которой состоится занятие по этому предмету. Общее расписание ВУЗа – совокупность расписаний для каждой группы. Фактически в таком расписании согласованы между собой во времени аудитории, группы учащихся и преподаватели. Составление таких расписаний, порой, нелегкая задача, особенно когда существует дефицит помещений, а количество занятий и групп студентов большое. При составлении расписания нужно учесть разнообразные требования к аудитории, времени и т.д. Приведем некоторые примеры таких условий:

- Условия, связанные с аудиториями. Понятно, что аудитория должна вмещать всех учеников, и в ней должно быть соответствующие оборудование. Единовременно в аудитории может проходить только одно занятие;
- Условия, связанные со студентами. Желательно, чтобы между лекциями не было больших перерывов. Необходимо чтобы студент успел перейти в другой учебный корпус, если занятия проходят в разных зданиях;
- Условия, связанные с преподавателями. У преподавателей также есть свои личные предпочтения, например, в какие дни и время проводить занятия;
- Условия, предъявляемые к учебному процессу. Желательно, чтобы после занятий по физкультуре не было лекционных занятий.

Задачи *Time Tabling* возникают при планировании занятости персонала, при согласовании времени различных встреч и т.д. Зачастую задачи *Time Tabling* можно свести к задачам *Project Scheduling*.

Библиографическая справка

При подготовке главы использованы термины из работы [20], численные примеры и рисунки из работ [41, 43], а также некоторые пояснения из монографии [23].

Существует множество монографий, посвященных **TP** . Среди них особо стоит отметить серию работ под руководством академика В.С. Танаева [23, 24, 25, 26].

Глава 2

Методы решения задач комбинаторной оптимизации

2.1 Классические задачи дискретной оптимизации

Дискретное программирование – *раздел математического программирования, посвященный нахождению экстремумов функций, заданных на конечных множествах.*

Под задачей дискретного программирования (дискретной оптимизации) понимается задача математического программирования в которой обычно необходимо минимизировать (или максимизировать) значение некоторой функции $f(x)$ на множестве допустимых решений G. То есть необходимо найти такое решение $x^0 \in G$, что

$$f(x^0) = \min_{x \in G} f(x). \tag{2.1}$$

При этом множество допустимых решений G конечно, то есть $0 < |G| < \infty$, где $|G|$ – число элементов множества G.

Существует достаточно широкий класс задач дискретной оптимизации, когда необходимо максимизировать значение целевой функции. Интересным является вопрос: можно ли модифицировать алгоритм решения задачи минимизации так, чтобы использовать его для решения задачи с обратным критерием оптимизации, т.е. для задачи максимизации.

Так как множество G конечно, то можно перебрать все допустимые решения, т.е. вычислить $f(x)$ для каждого $x \in G$. Но задачи дискретного

программирования нетривиальны в том смысле, что число допустимых решений в реальных задачах настолько велико, что их полный перебор (т.е. вычисление и сравнение всех значений функций) практически невозможно даже с использованием компьютера. В данном случае "невозможно" означает "невозможно осуществить полный перебор за приемлемое время".

Приведем некоторые примеры задач дискретной оптимизации.

ЗАДАЧА ОБ ОДНОМЕРНОМ РЮКЗАКЕ (ЗАДАЧА О РАНЦЕ, *Knapsack problem*). Имеется n предметов, каждый из которых характеризуется стоимостью $p_j > 0$ и весом $w_j > 0$, $j = 1, 2, \ldots, n$. Имеется ранец, грузоподъемность которого равна $C > 0$. Требуется положить в ранец набор предметов максимальной суммарной стоимости. При этом вес выбранных предметов не должен превышать грузоподъемности ранца. Предполагается, что $\sum_{j=1}^{n} w_j > C$ и $w_j < C$, $j = 1, 2, \ldots, n$. Для описания задачи на языке целочисленного линейного программирования введем следующие булевы переменные $x_j \in \{0, 1\}$, $j = 1, 2, \ldots, n$. Если $x_j = 1$, то предмет "кладем" в рюкзак, в противном случае – $x_j = 0$.

Получаем следующую задачу булева линейного программирования:

$$\begin{aligned} & f(x) = \sum_{j=1}^{n} p_j x_j \to \max, \\ & \sum_{j=1}^{n} w_j x_j \le C, \\ & x_j \in \{0, 1\},\ j = 1, 2, \ldots, n. \end{aligned} \tag{2.2}$$

В дальнейшем под "решением задачи" будем понимать только "допустимое решение" или же "оптимальное решение", если это ясно из контекста. Множество G для этой задачи – множество векторов $x = (x_1, x_2, \ldots, x_n)$, удовлетворяющих условию $\sum_{j=1}^{n} w_j x_j \le C$.

Определение 3 Размерность задачи – *число переменных, неравенств и уравнений, определяющих задачу.*

Для ЗАДАЧИ О РАНЦЕ говорят, что размерность равна n. Очевидно, что число допустимых решений для этой задачи может быть порядка 2^n.

Если мы попробуем решить эту задачу на компьютере путем перебора всех допустимых решений, то это может занять неприемлемо много времени. Попробуем оценить время необходимое компьютеру для решения задачи размерности $n = 100$. При $n = 100$ компьютеру необходимо перебрать порядка 2^{100} вариантов (решений). Причем для проверки каждого решения необходимо выполнить несколько "элементарных" операций. То есть количество операций, которое должен выполнить компьютер, может превышать 2^{100}.

Современные суперкомпьютеры (по состоянию на конец 2010 г.) способны выполнять порядка 2^{30} элементарных операций в секунду. Тогда для перебора всех решений нашего примера даже мощному суперкомпьютеру понадобится порядка 2^{70} секунд (более 2^{50} дней), то есть долгие годы, хотя это довольно таки привычная для практики размерность задачи. Из этого примера мы можем сделать следующий принципиальный вывод:

Для задач дискретной оптимизации необходимо построить алгоритм решения, время работы которого минимально. Искомый алгоритм должен находить оптимальное или приближенное решение за "разумное" время.

То есть необходимо сократить количество выполняемых операций. Сделать это можно сузив круг поиска оптимального решения, то есть рассматривая не все решения из множества G, а лишь небольшую их часть, подмножество, которое гарантированно содержит оптимальное решение.

ЗАДАЧА КОММИВОЯЖЁРА.

Странствующий торговец (коммивояжёр) должен посетить n городов. Для удобства пронумеруем города $1, \ldots, n$. Между каждой парой городов i и j задано расстояние $c_{ij} \geq 0$, причем может быть ситуация, что $c_{ij} \neq c_{ji}$, т.е. в разном направлении расстояния между городами могут быть разными. В начальный момент времени коммивояжёр находится в городе номер 0. Он выезжает из этого города, объезжает все города, посетив каждый из них ровно один раз, и возвращается в город 0. Необходимо найти такую последовательность посещения городов, при которой суммарная длина пройденного пути минимальна. Вместо расстояний могут быть заданы стоимости переездов из города в город.

Как и в задаче из первой главы, мы можем представить решение этой

задачи как вектор, характеризующий порядок посещения городов, т.е. в виде $(0, j_1, j_2, \ldots, j_n, 0)$, где $\{j_1, j_2, \ldots, j_n\} = \{1, 2, \ldots, n\}$. Этот порядок называют "маршрутом" и обозначают буквой π. Для маршрута π суммарную длину пройденного расстояния $l(\pi)$ можно вычислить так:

$$l(\pi) = c_{0j_1} + \sum_{i=1}^{n-1} c_{j_i j_{i+1}} + c_{j_n 0}.$$

Пусть G – множество возможных маршрутов. Тогда количество решений в этом множестве $|G| = n!$ Величина $n!$ "астрономически" быстро растёт с ростом n, поэтому алгоритм полного перебора явно не приемлем ...

ЗАДАЧА РАЗБИЕНИЯ.

Задано множество $N = \{b_1, b_2, \ldots, b_n\}$ чисел $b_1 \geq b_2 \geq \cdots \geq b_n > 0$, где $b_i \in Z_+,\ i = 1, 2, \ldots, n$. Необходимо ответить на вопрос, существует ли такое подмножество $N' \subset N$ чисел такое, что

$$\sum_{i \in N'} b_i = A = \frac{1}{2} \sum_{i=1}^{n} b_i ?$$

Несложно показать, что размерность этой задачи равна n и существует 2^n различных подмножеств множества N, включая пустое подмножество и подмножество равное самому множеству N.

ЗАДАЧУ РАЗБИЕНИЯ можно сформулировать и следующим образов: $|\sum_{i \in N'} b_i - A| \to \min$. Нетрудно доказать, что ЗАДАЧА РАЗБИЕНИЯ в такой постановке является частным случаем ЗАДАЧИ О РАНЦЕ.

Определение 4 *Если в некоторой задаче дискретной оптимизации сократить множество G допустимых решений введением (добавлением) каких-либо новых ограничений, то полученную задачу называют* **частным случаем задачи**. *Исходную задачу при этом называют* **общим случаем**.

Если в ЗАДАЧЕ О РАНЦЕ мы зафиксируем условие $p_j = w_j$, при этом примем $C = A = \frac{1}{2} \sum_{j=1}^{n} w_j$, то мы получим ЗАДАЧУ РАЗБИЕНИЯ. Действительно, если $p_j = w_j = b_j$, $j = 1, 2, \ldots, n$, и оптимальное значение

целевой функции для ЗАДАЧИ О РАНЦЕ $f(x^0) = A$, тогда ответ для ЗАДАЧИ РАЗБИЕНИЯ "ДА, подмножество существует", иначе "НЕТ, такое разбиение не существует".

Задачи, в которых нужно дать ответ "ДА" или "НЕТ" (например, РАЗБИЕНИЕ) на некоторый вопрос, называются **задачами распознавания**.

Три упомянутые задачи являются классическими задачами дискретной оптимизации. В учебнике мы многократно будем использовать их для иллюстрации некоторых свойств рассматриваемых задач, а на примере ЗАДАЧИ О РАНЦЕ мы расскажем о методах, с помощью которых подобные задачи можно решать.

Эти три задачи также являются задачами *комбинаторной оптимизации.*

Задачи комбинаторной оптимизации – это задачи выбора и/или расположения некоторого, обычно конечного, множества в соответствии с заданными правилами с целью минимизировать (максимизировать) некоторую целевую функцию.

Действительно, в ЗАДАЧЕ О РАНЦЕ мы должны выбрать оптимальный набор предметов, а в ЗАДАЧЕ КОММИВОЯЖЁРА – "упорядочить" множество городов. Большинство задач **TP** также являются задачами комбинаторной оптимизации.

Размерность задачи комбинаторной оптимизации обычно равно мощности множества, элементы которого необходимо выбрать или упорядочить.

Поэтому говорят, что размерность ЗАДАЧИ КОММИВОЯЖЁРА равна n, где n – число городов, а размерности ЗАДАЧИ О РАНЦЕ и РАЗБИЕНИЕ также равны n, где n – число предметов или чисел.

2.2 Некоторые сведения о сложности (трудоёмкости) задач комбинаторной оптимизации

В этой части главы даются краткие сведения о трудоёмкости, классах P и NP задач дискретной оптимизации, объясняется, что такое NP-полные и NP-трудные задачи, а также даются сведения о сводимости задач друг к другу. Приведенная информация не исчерпывающая и только призвана дать общее представление читателям о характере решаемых задач, необходимое для дальнейшего изложения. Подробнее об этих сведениях можно прочитать в монографии М.Гэри и Д.Джонсона [**?**].

Когда мы решаем задачу комбинаторной оптимизации, возникает важный вопрос – насколько сложна (или "трудоёмка") эта задача. Если для некоторой новой задачи не удается построить быстрый алгоритм решения, то пытаются определить, не является ли эта задача NP-трудной. Если это подтверждается, то в предположении $P \neq NP$, "быстрый" алгоритм для данной задачи построить не удастся. В комбинаторной оптимизации задачи условно делятся на полиномиально разрешимые (относительно простые) и NP-трудные (соответственно, сложные).

2.2.1 Трудоемкость алгоритмов и полиномиально разрешимые задачи

Эффективность алгоритмов оценивается по времени их работы, то есть по количеству необходимых операций, которые выполняются в алгоритме для конкретного примера. Ранее было сказано, что алгоритмы решения реализуются, как правило, на компьютерах. Для решения какого-либо примера мы должны "сообщить" компьютеру не только алгоритм (последовательность операций, которые компьютер должен выполнить), но и передать ему *входные данные*, то есть значения числовых параметров решаемого примера. Размер входных данных *input size* определяется обычно числом битов, необходимых для бинарного кодирования входных данных. Например, для представления числа a в бинарной кодировке необходимо $\log_2 a$ бита[1], а для кодирования n чисел, самое большее из

[1]Подробнее о том, как данные кодируются в вычислительной машине можно узнать из курса информатики. В современных компьютерах информация на техническом уровне представляется

которых равно a, необходимо $n \log_2 a$ бит. Таким образом, для кодирования примера ЗАДАЧИ КОММИВОЯЖЁРА размерности n необходимо примерно $n^2 \log_2 a$ бит (на вход мы подаем расстояния между городами, максимальное из которых равно a), а для кодирования ЗАДАЧИ РАЗБИЕНИЯ $n \log_2 a$ бит.

При оценке времени работы алгоритма, как правило, берут в расчет не длину входных данных (например, $n \log_2 a$), а размерность примера n. То есть трудоёмкость разумного алгоритма должна зависеть от размерности задачи n. Очевидно, что для двух примеров задачи с одной и той же размерностью, время работы алгоритма (количество операций) может быть разным. Поэтому важно определить максимальное время работы. Для этой цели вводят функцию $T(n)$ – трудоёмкость алгоритма, которая является верхней границей времени работы алгоритма для всех примеров размерности n. Чаще всего тяжело задать функцию $T(n)$ точно, поэтому рассматривают только ее наибольший порядок. Мы говорим, что трудоёмкость алгоритма $T(n)$ имеет порядок $O(g(n))$, если существует такие константы (числа) C и n_0, что выполняется $T(n) \leq CG(n)$ для любого $n \geq n_0$. Причём константа C не зависит от размерности задачи n. Например, если трудоёмкость некоторого алгоритма $T(n) = 37n^3 + 5n^2 + 17$, то говорят, что алгоритм имеет трудоёмкость порядка $O(n^3)$ операций (или для краткости "трудоёмкость алгоритма $O(n^3)$"). Для функции $T(n) = n^2 2^n + n^{500} + 7n^2 + 4$ имеем трудоёмкость порядка $O(n^2 2^n)$ операций. Говорят, что данный алгоритм имеет экспоненциальную трудоёмкость.

Если трудоёмкость алгоритма решение задачи равна $O(n^k)$, где k – некоторая константа, не зависящая от n, то говорят, что задача *полиномиально разрешима.*

Полиномиально разрешимые задачи причисляют к классу задач, который называется P. Обычно задачи из класса P принято считать "несложными" или "простыми". Соответственно, алгоритмы с такой трудоёмкостью называются **полиномиальными**.

в виде набора единичек и нулей (1 и 0, или "правда" и "ложь"). Бит – это наименьшая единица информации. Фактически, это ячейка в памяти компьютера, в которой могут быть записаны либо 1 либо 0. При бинарной кодировке число 617 представляется в виде "1001101001", т.е. $2^0 + 2^3 + 2^5 + 2^6 + 2^9 = 617$. Таким образом, для кодирования числа 617 нам необходимо 10 бит. Такую кодировку также называют двоичной.

Если трудоёмкость алгоритма зависит от значений числовых параметров примера, например $O(n4)$ для ЗАДАЧИ РАЗБИЕНИЯ, то говорят, что алгоритм **псевдо-полиномиальный**. Алгоритм трудоёмкости вида $O(n^x y^n)$, где x и y – константы, называется экспоненциальным. Очевидно, что для некоторых n время работы полиномиального алгоритма может быть больше, чем время работы экспоненциального алгоритма. Например, для двух алгоритмов трудоёмкости $O(n^3)$ и $O(2^n)$ операций соответственно, для $n = 5$ получим $5^3 = 125 > 32 = 2^5$, но уже для $n \geq 10$ мы получим $2^n > n^3$, а так как нам важна асимпотическая оценка при $n \to \infty$, то полиномиальный алгоритм считается более предпочтительным.

2.2.2 Класс NP и труднорешаемые задачи

Не для всех задач комбинаторной оптимизации можно построить полиномиальный алгоритм. Мы уже обозначили класс задач P, которые могут быть решены за полиномиальное время от длины входной информации. Представим, что у нас есть специальный компьютер, который содержит "угадывающий" блок (компоненту) – "оракул". Допустим, мы решаем на таком компьютере задачу распознавания (например, ЗАДАЧУ РАЗБИЕНИЯ). На вход такого вымышленного компьютера передается решаемый пример. Если для данного примера существует ответ "ДА", то оракул выдает некоторое подмножество N', такое, что $\sum_{i \in N'} b_i = A$ и нам остается только проверить, правильность решения, выданного оракулом.

Класс NP содержит все задачи распознавания, в которых для каждого примера с ответом "ДА" оракул выдаст решение (подмножество чисел в ЗАДАЧЕ РАЗБИЕНИЯ), такое что:

- **длина решения полиномиально ограничена длиной входных данных, и**
- **решение может быть проверено за полиномиальное время от размерности задачи.**

Любой оптимизационной задаче можно поставить в соответствие задачу распознавания. Например, рассмотрим задачу *RCPSP*. Если поста-

вить вопрос таким образом: "Существует ли допустимое расписание, что $C_{\max} \leq y$?", то мы получим задачу распознавания. Данная задача распознавания принадлежит классу NP. Нетрудно показать, что большинство задач теории расписаний, сформулированных как задача распознавания, принадлежат классу NP. Более того, все задачи распознавания, принадлежащие классу P, принадлежат и классу NP, т.е. $P \subseteq NP$. Совпадают ли эти два класса задач – это один из важнейших на данный момент вопросов математики, ответ на который пока не найден.

Говорят, что задача A за полиномиальное время может быть сведена к задаче B (обозначается как $A \propto B$), если существует полиномиальный алгоритм модификации за полиномиальное время любого примера I_A задачи A в пример I_B задачи B так, что если ответ на пример I_A "ДА", тогда и только тогда, когда ответ на пример I_B тоже "ДА".

Полиномиальное сведение $A \propto B$ говорит о том, что задача B как минимум также трудна, как и задача A. Если существует полиномиальный алгоритм решения задачи B, то существует и полиномиальный алгоритм решения задачи A. Если же задача A не может быть решена за полиномиальное время, то задача B также не может быть "полиномиальной". Если $A \propto B$ и $B \propto C$, то верно и $A \propto C$.

Задача B называется NP-полной (NP-complete), если:

- **B принадлежит классу NP, и**
- **любая другая задача $A \in NP$ полиномиально сводится к задаче B.**

Если выполняется только второе условие, то задача B называется **NP-трудной** (NP-hard). То есть оптимизационная задача является **NP-трудной**, если соответствующая задача распознавания является **NP-полной**. Если какая-либо **NP-полная** задача полиномиально разрешима, то и все задачи из класса NP могут быть решены за полиномиальное время, то есть окажется, что $P = NP$.

Задача B называется **NP-трудной в сильном смысле**, если для нее не существует псевдо-полиномиальный алгоритм решения (в предположении $P \neq NP$).

В 1971 году Стив Кук доказал, что **NP-полные** задачи существуют на примере задачи о выполнимости. Формулировку задачи О ВЫПОЛНИМОСТИ и доказательство Кука можно найти в монографии М.Гэри и Д.Джонсона.

ЗАДАЧА О ВЫПОЛНИМОСТИ. Даны n булевых переменных $x_1, x_2, \ldots, x_n$ таких, что $x_i \in \{0, 1\}$, $i = \overline{1, n}$. Определим три функции:

x	y	конъюнкция $x\&y$	дизъюнкция $x \vee y$	отрицание $\overline{x}$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Пусть $f(x_1, \ldots, x_n)$ некоторая булева функция. Из курса "Дискретный анализ"[2] известно, что любая булева функция может быть выражена с помощью перечисленных выше трех функций.

Задача О ВЫПОЛНИМОСТИ. Существует ли такой набор значений переменных $x_1, \ldots, x_n$, что булева функция $f(x_1, \ldots, x_n)$ принимает значение 1 ("Истина")?

Согласно этой классификации трудоёмкости задач: ЗАДАЧА РАЗБИЕНИЯ является NP-**полной**; ЗАДАЧА О РАНЦЕ NP-**трудной**; а ЗАДАЧА КОММИВОЯЖЁРА NP-**трудной в сильном смысле**.

Чтобы показать, что некоторая задача X является NP-трудной, необходимо доказать сводимость к ней некоторой NP-полной задачи Y, то есть необходимо показать, что $Y \propto X$.

В главе 5 приводится доказательство NP-**трудности** задачи *RCPSP*.

2.2.3 Классификация алгоритмов решения

Далее приводятся некоторые классификации алгоритмов решения задач дискретной оптимизации:

[2]Материалы курса можно найти на сайте кафедры физико-математических методов управления физического факультета МГУ http://physcontrol.phys.msu.ru/index.php/study/special

- **Классификация по трудоёмкости:**
  - *Полиномиальные* алгоритмы;
  - *Псевдо-полиномиальные* алгоритмы;
  - *Экспоненциальные* алгоритмы.
- **Классификация по точности:**
  - Результатом работы *точного алгоритма* всегда является оптимальное решение. Как правило, в этих алгоритмах применяют различные способы сокращения перебора;
  - *Приближенные алгоритмы* находят решение без каких либо гарантий точности получаемых решений. Обычно это быстрые полиномиальные алгоритмы, основанные на какой либо догадке о свойствах оптимальных решений. Часто такие алгоритмы называют эвристическими;
  - *Приближенные алгоритмы, с гарантированными оценками качества.* Их отличие от алгоритмов из предыдущего пункта заключается в том, что можно оценить абсолютную или относительную погрешность, т.е. отклонение полученного значения целевой функции от оптимального;
  - *Приближенные алгоритмы, с регулируемой точностью.* Время работы таких алгоритмов зависит от выбранной пользователем необходимой точности. Например, нас может устроит отклонение от оптимума не больше чем на 1%. К этим алгоритмам относятся понятия $PTAS$ и $FPTAS$, о которых мы расскажем позднее.

Подробнее расскажем, что такое *абсолютная и относительная погрешности*. Пусть для некоторого примера I некоторой задачи минимизации с помощью приближенного алгоритма было получено значение целевой фукнции $A(I)$. Обозначим через $OPT(I)$ оптимальное значение целевой функции для этого примера. Очевидно, что $OPT(I) \leq A(I)$.

Относительной погрешностью называется величина

$$\frac{A(I) - OPT(I)}{OPT(I)}.$$

Очевидно, что если на некоторых примерах оптимальное значение $OPT(I) = 0$, то вычисление относительной погрешности может оказаться бессмысленным. Для таких задач рассматривают *абсолютную погрешность* $A(I) - OPT(I)$.

Если для некоторой оптимизационной задачи существует приближенный **полиномиальный** алгоритм такой, что для любого примера I задачи, величина относительной погрешности ограничена константой k, т.е. выполняется

$$\frac{A(I) - OPT(I)}{OPT(I)} \leq k,$$

то говорят, что задача принадлежит классу **APX** (от англ. "approximable"). Если NP-**трудная** задача принадлежит этому классу, то это хорошая новость. Например, ЗАДАЧА О РАНЦЕ принадлежит классу APX.

Теорема 2.1 *ЗАДАЧА КОММИВОЯЖЁРА не принадлежит классу APX, если* $P \neq NP$.

Доказательство. Для доказательства этой теоремы нам понадобится определение следующей NP-трудной задачи:

Задача Гамильтонов Цикл. Задан граф $G = (V, E)$. Требуется ответить на вопрос, существует ли в этом графе Гамильтонов цикл, то есть путь (цепь), содержащий каждую вершину графа ровно один раз.

Построим сведение задачи Гамильтонов Цикл к ЗАДАЧЕ КОММИВОЯЖЁРА. Пусть в некотором примере I этой задачи $V = \{1, 2, \ldots, n\}$. В соответствующем примере I' ЗАДАЧИ КОММИВОЯЖЁРА мы задаем множество городов $\{0, 1, 2, \ldots, n\}$. Определим все расстояния между городами: $c_{ij} = 1$ и $c_{ji} = 1$, если существует ребро $(i, j) \in E$, иначе $c_{ij} = c_{ji} = M$, где $i, j = 1, 2, \ldots, n$ и $M \gg n$ – сколь угодно большое число. Дополнительно определим $c_{0j} = c_{j0} = 0$ и $c_{jj} = M$ для всех $j = 1, 2, \ldots, n$.

Пример I имеет ответ "ДА" тогда и только тогда, когда длина оптимального маршрута в примере I' равна n. Иначе длина оптимального маршрута больше M.

Предположим, что существует приближенный полиномиальный алгоритм решения ЗАДАЧИ КОММИВОЯЖЁРА, относительная погреш-

ность которого не превосходит числа k. Пусть $M > kn$. Тогда решив любой пример I' ЗАДАЧИ КОММИВОЯЖЁРА, соответствующий примеру I задачи о Гамильтоновом цикле, мы можем сказать, есть ли в примере I Гамильтонов цикл, то есть исходная задача может быть решена за полиномиальное время, что противоречит NP-трудности исходной задачи. Согласно полученному противоречию, теорема верна. □

2.3 Методы решения задач дискретной оптимизации

Метод решения задач дискретной оптимизации – это общая идея, применимая к широкому классу задач. Алгоритм решения – это реализация метода решения для конкретной задачи.

Различают следующие методы решения, некоторые из которых описаны в этом учебнике:

- **Эвристические алгоритмы (в т.ч. вероятностные алгоритмы и локальный поиск)**;
- **Метод динамического программирования**;
- **Метод Ветвей и Границ**;
- **Метаэвристические методы**;
- **Графический метод** и т.д.

Далее мы подробнее расскажем о некоторых методах на примере ЗАДАЧИ О РАНЦЕ.

2.3.1 Эвристические алгоритмы

Эвристические алгоритмы – алгоритмы, основанные на правдоподобных, но не обоснованных математически предположениях о свойствах оптимального решения задачи.

Фактически в эвристическом алгоритме учитывается одно или несколько свойств оптимального решения, на основе которых производится сокращение перебора возможных решений.

Понятно, что у каждой задачи могут быть особенные свойства оптимальных решений. Поэтому эвристические алгоритмы в разных задачах могут быть разными. Для ЗАДАЧИ КОММИВОЯЖЁРА можно предложить следующий эвристический алгоритм: из города 0 мы едем в ближайший город j (т.е. в город j, соответствующий наименьшему значению $c_{0i},\ i = 1, 2, \ldots, n$). Из города j едем в ближайший из оставшихся городов и т.д. Такие алгоритмы называются *жадными*, т.к. на каждом шаге конструирования решения мы пытаемся решить локальную оптимизационную задачу (например, переезжать в ближайший город).

Пусть мы получили некоторое допустимое решение для ЗАДАЧИ КОММИВОЯЖЁРА. Можно попытаться его улучшить применив алгоритмы локального поиска. Основная идея локального поиска – перебор всех допустимых решений *близких* в каком-то роде к исходному решению. Общую схему алгоритмов локального поиска можно описать следующим образом.

Пусть $x^k \in G,\ k = 0, 1, 2, \ldots$ – первоначальное допустимое решение задачи дискретной оптимизации. Построим окрестность этого решения, то есть выделим подмножество *близких* решений $G(x^k) \subset G$ и найдем $f(x^{k+1}) = \min\limits_{x \in G(x^k)} f(x)$. Теперь будем рассматривать решение x^{k+1} в качестве исходного и повторим процедуру. Остановка такого локального поиска может быть совершена, например, если более "хорошее" решение в очередной окрестности $G(x^k)$ не найдено. Могут применяться и другие правила остановки.

Для ЗАДАЧИ КОММИВОЯЖЁРА можно предложить следующий эвристический алгоритм локального поиска – попарная перестановка, когда в полученном расписании пытаются переставить местами любые два города в маршруте.

Теперь опишем простой эвристический алгоритм для ЗАДАЧИ О РАНЦЕ 2.2. Разумным кажется вкладывать в ранец те предметы, для которых удельная стоимость максимальна, т.е. для которых значение $\lambda_j = \dfrac{p_j}{w_j}$ наибольшее. На основе этой идеи построен алгоритм 1.

В результате работы алгоритма мы получим допустимое решение $(x_1, x_2, \ldots, x_n)$ и соответствующее значение целевой функции F.

Algorithm 1 Эвристический алгоритм для ЗАДАЧИ О РАНЦЕ

1: Для каждого предмета $j = 1, 2, \ldots, n$, вычислим $\lambda_j = \dfrac{p_j}{w_j}$;
2: Перенумеруем предметы согласно правилу $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$;
3: $F := 0$;
4: $C_0 := C$;
5: $x_j := 0,\ j = 1, 2, \ldots, n$;
6: **for** $j := 1$ to n **do**
7: **if** $C_0 \geq w_j$ **then**
8: $x_j := 1$, $C_0 := C_0 - w_j,\ F := F + p_j$;
9: **end if**
10: **end for**

Нетрудно оценить трудоёмкость этого алгоритма. Для перенумерации (сортировки) n предметов (чисел) необходимо порядка $O(n \log n)$ операций. На выполнение основного цикла необходимо порядка $O(n)$ операций. Следовательно, трудоёмкость данного алгоритма $O(n \log n)$. Несмотря на кажущуюся разумность, Алгоритм 1 может дать сколько угодное большое отклонение от оптимального решения. Достаточно рассмотреть пример с двумя предметами, где $p_1 = 2$, $w_1 = 1$; $p_2 = M$, $w_2 = M$; $C = M$, где M – сколь угодно большое число. Для данного примера оптимальное решение $x^0 = (0, 1)$, соответствует значению целевой функции $f(x^0) = M$, хотя алгоритм 1 построит решение $x^1 = (1, 0)$, для которого $f(x^1) = 2 \ll M = f(x^0)$.

В 30–40-е годы 20-го века для решения некоторых задач вычислительной математики (для решения интегральных уравнений, для оценки многократных интегралов и др.) стали применять численные методы, которые в дальнейшем получили название методы статистических испытаний (в частности, *Метод Монте-Карло* и *Метод Лас-Вегас*). В последствии идеи этих методов были применены для поиска приближенных решений и в задачах дискретной оптимизации.

Эти идеи можно описать на примере модификации Алгоритма 1 для ЗАДАЧИ О РАНЦЕ. В этой модификации решение о включении предмета j в ранец принимается исходя из вероятности $\dfrac{\lambda_j}{\sum\limits_{i=1}^{n} \lambda_i}$. То есть модифицированный Алгоритм 2 выглядит следующим образом:

Алгоритм 2 запускается несколько раз (например, m раз). Поэтому могут

Algorithm 2 Алгоритм вероятностного поиска для ЗАДАЧИ О РАНЦЕ

1: Для каждого предмета $j = 1, 2, \ldots, n$, вычислим $\lambda_j = \dfrac{p_j}{w_j}$;
2: $F := 0$; $C_0 := C$; $x_j := 0,\ j = 1, 2, \ldots, n$; $N' := N$;
3: **while** $N' \neq \emptyset$ **do**
4: для каждого предмета $i \in N'$ вычислим вероятность $\rho_i := \dfrac{\lambda_i}{\sum\limits_{k \in N'} \lambda_k}$;
5: случайным образом согласно вероятностям ρ_i выберем предмет $j \in N'$;
6: $x_j := 1$, $C_0 := C_0 - w_j$, $F := F + p_j$;
7: Исключим из множества N' предмет j, а также все предметы $i \in N'$, для которых $p_i > C_0$;
8: **end while**

быть получены разные решения $x^1, x^2, \ldots, x^m$, которым соответстсвуют разные значения целевой функции. Из этих решений необходимо выбрать "наилучшее" с точки зрения целевой функции. Если Алгоритм 2 запускать сколь угодно много раз, то вероятность получения в результате его работы оптимального решения стремится к 1 (к 100%). Трудоемкость Алгоритма 2 равна $O(m \cdot n^2)$ операций.

2.3.2 Метаэвристические методы

В 70–90-е годы прошлого века в результате совершенствования эвристических алгоритмов появились т.н. *метаэвристические методы*, например, *Генетические алгоритмы* (Genetic algorithms), *метод имитации отжига* (Simulated annealing), *метод муравьиных колоний* (Ant Colony Optimization) и др. Идеи этих методов были "заимствованы" из разных областей науки. Например, идея Генетического алгоритма из генетики, идея метода муравьиных колоний из биологии и т.д.

Метод муравьиные колонии основан на следующем наблюдении из жизни муравейника. Муравьи пытаются отыскать пищу поближе к муравейнику. О том, где находится ближайшая пища, они сообщают друг другу следующим образом. При своем передвижении муравьи оставляют на поверхности "феромонные следы", "запах" которых могут чувствовать другие муравьи. Очередной муравей при поиске пищи ориентируется на феромонный след других муравьев и старается с небольшими отклонениями следовать в том направлении, куда ведет большинство следов. Небольшие отклонения от общего направления позво-

ляют строить новые маршруты и находить новую пищу, а накопление феромона позволяет двигаться в разумном (локально оптимальном) направлении. Основываясь на этом наблюдении, Марко Дориго в 1992 году [51, 52] предложил метод решения оптимизационных задач, названный методом муравьиных колоний.

В генетических алгоритмах стараются комбинировать хорошие "решения", чтобы получить еще более "лучшее". В генетике также пытаются скрещивать разные биологические виды (или особей одного вида), чтобы полученное потомство обладало полезными качествами своих родителей.

Метаэвристические методы обычно обладают двумя важными особенностями:

- В результате их работы последовательно строятся несколько решений;
- Построение каждого нового решения основывается на накопленных знаниях о качестве предыдущих полученных решений.

В терминах "муравейника" каждое решение – это некоторый маршрут, а накопленные знания – это феромонные следы. В главе, посвященной одноприборным задачам **ТР**, будет подробно описан алгоритм муравьиные колонии для задачи минимизации суммарного запаздывания $1||\sum T_j$.

2.3.3 Метод динамического программирования

Метод динамического программирования получил большое распространение при решении некоторых задач дискретной оптимизации. Далее приводится формальное описание сути динамического программирования. Разобраться с этим описанием легче после ознакомления с алгоритмом динамического программирования для ЗАДАЧИ О РАНЦЕ, приведенного ниже.

Динамическое программирование – раздел математического программирования, посвященный исследованию многошаговых задач принятия оптимальных решений. При этом многошаговость

задачи отражает реальное протекание процесса принятия решений во времени, либо вводится в задачу искусственно за счет расчленения процесса принятия однократного решения на отдельные этапы, шаги. Цель такого представления состоит в сведении исходной задачи высокой размерности к решению на каждом шаге задачи меньшей размерности.

Общая схема многошагового процесса принятий оптимальных решений (с дискретным временем) состоит в следующем.

Пусть имеется некоторая система S, состояние которой в начальный момент времени 0 характеризуется числом x_0. В каждый момент времени $k,\ k = 1, \ldots, n$, делается некоторый выбор x_k, в результате чего система изменяет свое состояние. При этом каждое решение (выбор) x_k должно удовлетворять как исходным ограничениям системы S так и ограничениям, возникающим за счет ранее сделанных выборов $x_1, x_2, \ldots, x_{k-1}$. Каждое решение (выбор) приносит определенный выигрыш (доход). Для применимости схемы динамического программирования необходимо, чтобы общий доход за n шагов был равен сумме доходов от отдельных шагов. Последовательность допустимых решений (допустимого выбора) на отдельных шагах обычно называют *стратегией*. Требуется найти среди всех политик оптимальную, дающую максимум суммарного дохода.

В основе динамического программирования лежит следующая простая характеристика оптимальной стратегии, сформулированная Р.Беллманом в 50-е годы 20-го века и названная **принципом оптимальности Беллмана** [1]: каково бы ни были начальное состояние и первое решение, последующие решения составляют оптимальную политику по отношению к начальному состоянию, полученному в результате первого решения.

Алгоритм динамического программирования для ЗАДАЧИ О РАНЦЕ, основанный на принципе оптимальности Беллмана, опишем следующим образом. Предположим, что все числовые параметры задачи – целые положительные числа, то есть $p_i, w_i \in Z^+,\ i = 1, \ldots, n$, $C \in Z^+$.

Алгоритм состоит из n шагов. На каждом шаге $j, j = 1, 2, \ldots, n$, принимается решение о том, включать или не включать предмет в ранец (т.е. $x_j = 1$ или $x_j = 0$). Это решение принимается для каждого состояния t, характеризующую суммарную вместимость ранца для предметов

$i = 1, \ldots, j$. То есть, если на последующих шагах в ранец будут включены предметы общим весом равным $\sum_{i=j+1}^{n} x_i w_i$, то для предметов $i = 1, \ldots, j$ у нас остается свободными $t = C - \sum_{i=j+1}^{n} x_i w_i$ единиц веса. Мы строим решение с учетом всех возможных будущих состояний системы.

На каждом шаге $j = 1, \ldots, n$ строится функция выигрыша

$$g_j(t) = \max_{x_j \in \{0,1\}} (p_j x_j + g_{j-1}(t - w_j x_j)), t \geq w_j x_j,$$

определенная в каждой целочисленной точке $0 \leq t \leq C$. Если $t < w_j$, то $x_j := 0$. Для каждой точки t фиксируется соответствующее значение $x_j = \arg\max g_j(t)$. На шаге $j = n$ в точке $t = C$ имеем оптимальное решение.

Продемонстрируем работу алгоритма на примере [22].

$$\begin{gathered} f(x) = 5x_1 + 7x_2 + 6x_3 + 3x_4 \longrightarrow \max \\ 2x_1 + 3x_2 + 5x_3 + 7x_4 \leq 9 \\ x_i \in \{0, 1\},\ i = 1, \ldots, 4. \end{gathered} \tag{2.3}$$

Работу алгоритма можно представить в виде таблицы

t	$g_1(t)$	$x(t)$	$g_2(t)$	$x(t)$	$g_3(t)$	$x(t)$	$g_4(t)$	$x(t)$
0	0	(0,,,)	0	(0,0,,)	0	(0,0,0,)	0	(0,0,0,0)
1	0	(0,,,)	0	(0,0,,)	0	(0,0,0,)	0	(0,0,0,0)
2	5	(1,,,)	5	(1,0,,)	5	(1,0,0,)	5	(1,0,0,0)
3	5	(1,,,)	7	(0,1,,)	7	(0,1,0,)	7	(0,1,0,0)
4	5	(1,,,)	7	(0,1,,)	7	(0,1,0,)	7	(0,1,0,0)
5	5	(1,,,)	12	(1,1,,)	12	(1,1,0,)	12	(1,1,0,0)
6	5	(1,,,)	12	(1,1,,)	12	(1,1,0,)	12	(1,1,0,0)
7	5	(1,,,)	12	(1,1,,)	12	(1,1,0,)	12	(1,1,0,0)
8	5	(1,,,)	12	(1,1,,)	13	(0,1,1,)	13	(0,1,1,0)
9	5	(1,,,)	12	(1,1,,)	13	(0,1,1,)	13	(0,1,1,0)

Таким образом получаем оптимальное решение $(0, 1, 1, 0)$ и соответствующее значение целевой функции $g_4(9) = 13$. Очевидно, что трудоёмкость такого алгоритма – $O(nC)$ операций.

Для ЗАДАЧИ О РАНЦЕ существует и другой алгоритм динамического программирования трудоёмкости $O(nF_{\max})$ операций, где $F_{\max}$ – оптимальное значение целевой функции. Ознакомится с ним можно в монографии Пападимитриу [19].

2.3.4 Графический метод

Графический метод – это модификация метода динамического программирования.

Для краткости будем обозначать алгоритм динамического программирования через *DPA* (Dynamic Programming Algorithm), а Графический алгоритм, соответственно, *GrA* (Graphical Algorithm).

В DPA необходимо вычислить значение $f_j(t)$ некоторой частичной функции для каждого возможного значения аргумента t (для каждого состояния) процесса принятия решения, где $t \in [0, C]$ и $t \in Z$. Если эти вычисления необходимо провести на каждом шаге $j = 1, 2, \ldots, n$, где n размерность задачи, то трудоёмкость DPA обычно равна $O(nC)$. Однако зачастую нет необходимости вычислять (и сохранять в памяти) значения $f_j(t)$ для каждой точки t. Может оказаться, что на некотором интервале $[t_l, t_{l+1})$ вычисляемая функция представима аналитически в виде $f_j(t) = \varphi(t)$ (например, $f_j(t) = k_j \cdot t + b_j$, т.е. функция $f_j(t)$ определена в том числе для нецелых значений аргумента t).

Пусть для некоторой проблемы минимизации целевого функционала заданы следующие рекурсивные уравнения Беллмана:

$$f_j(t) = \min \begin{cases} \Phi^1(t) = \alpha_j(t) + f_{j-1}(t - a_j), & j = 1, 2, \ldots, n; \\ \Phi^2(t) = \beta_j(t) + f_{j-1}(t - b_j), & j = 1, 2, \ldots, n. \end{cases} \tag{2.4}$$

с начальными условиями

$$\begin{array}{lll} f_0(t) = 0, & \text{для} & t \geq 0, \\ f_0(t) = +\infty, & \text{для} & t < 0. \end{array} \tag{2.5}$$

В (2.4) функция $\Phi^1(t)$ соответствует значению переменной $x_j = 1$, а функция $\Phi^2(t)$ соответствует значению $x_j = 0$. Переменная x_j – управляемый параметр на шаге j процесса (включать или не включать предмет в ранец).

Таблица 2.1: Вычисления в *DPA*

t	0	1	2	...	y	...	C
$f_j(t)$	$value_0$	$value_1$	$value_2$	...	$value_y$	...	$value_C$
частичное оптим. решение $X(t)$	$X(0)$	$X(1)$	$X(2)$	...	$X(y)$	...	$X(C)$

Таблица 2.2: Вычисления в *GrA*

t	$[t_0, t_1)$	$[t_1, t_2)$	...	$[t_l, t_{l+1})$	...	$[t_{m_j-1}, t_{m_j}]$
$f_j(t)$	$\varphi_1(t)$	$\varphi_2(t)$	...	$\varphi_{l+1}(t)$	...	$\varphi_{m_j}(t)$
частичное оптим. решение $X(t)$	$X(t_0)$	$X(t_1)$	...	$X(t_l)$	...	$X(t_{m_j-1})$

На шаге j, $j = 1, 2, \ldots, n$, DPA мы вычисляем и сохраняем в памяти компьютера данные, представленные в таб. 2.1.

В таблице 2.1 $X(y)$, $y = 0, 1, \ldots, C$, – вектор, описывающий частичное оптимальное решение, который состоит из j элементов (значений) $x_1, x_2, \ldots, x_j \in \{0, 1\}$ (например, см. *DPA* для ЗАДАЧИ О РАНЦЕ).

Та же самая информация иногда может быть представлена в виде, приведенном в таб. 2.2, где $0 = t_0 < t_1 < t_2 < \cdots < t_{m_j} = C$.

Для вычисления функции $f_{j+1}(t)$ в *GrA* мы сравниваем две промежуточные функции $\Phi^1(t)$ и $\Phi^2(t)$, которые определяются следующим образом.

Функция $\Phi^1(t)$ является комбинацией двух функций $\alpha_{j+1}(t)$ и $f_j(t-a_{j+1})$. Функция $f_j(t-a_{j+1})$ имеет ту же структуру, как представлено в таб. 2.2, но где каждый интервал $[t_l, t_{l+1})$ заменен на интервал $[t_l - a_{j+1}, t_{l+1} - a_{j+1})$, то есть мы сдвигаем график функции $f_j(t)$ вправо на a_{j+1} единиц. Если мы можем представить функцию $\alpha_{j+1}(t)$ в той же форме, как и в таб. 2.2 с μ_1 столбцами, тогда мы можем сохранить функцию $\Phi^1(t)$ в памяти компьютера в той же форме, как и в таб. 2.2 с $m_j + \mu_1$ столбцами. Аналогично мы можем сохранить функцию $\Phi^2(t)$ в форме таб. 2.2 с $m_j + \mu_2$ столбцами. μ_1 и μ_2 – количество "устойчивых" (неизменных) состояний функций $\alpha_j(t)$ и $\beta_j(t)$, соответственно.

Функцию $f_{j+1}(t) = \min\{\Phi^1(t), \Phi^2(t)\}$ мы вычисляем следующим образом. Пусть таблица, которой задана функция $\Phi^1(t)$, содержит интервалы

(столбцы)

$$[t_0^1, t_1^1), [t_1^1, t_2^1), \ldots, [t_{(m_j+\mu_1)-1}^1, t_{(m_j+\mu_1)}^1],$$

а таблица, соответствующая функции $\Phi^2(t)$, интервалы (столбцы)

$$[t_0^2, t_1^2), [t_1^2, t_2^2), \ldots, [t_{(m_j+\mu_2)-1}^2, t_{(m_j+\mu_2)}^2].$$

Для вычисления функции $f_{j+1}(t)$, мы сравниваем две функции $\Phi^1(t)$ и $\Phi^2(t)$ на каждом интервале, сформированном точками

$$\{t_0^1, t_1^1, t_2^1, \ldots, t_{(m_j+\mu_1)-1}^1, t_{(m_j+\mu_1)}^1, t_0^2, t_1^2, t_2^2, \ldots, t_{(m_j+\mu_2)-1}^2, t_{(m_j+\mu_2)}^2\},$$

при этом мы определяем точки пересечения обеих функций на этих интвералах – $t_1^3, t_2^3, \ldots, t_{\mu_3}^3$, где μ_3 – общее количество точек пересечения. Поэтому в таблице, соответствующей функции $f_{j+1}(t)$, может быть $2m_j + \mu_1 + \mu_2 + \mu_3 \leq C$ интервалов.

Фактически на каждом шаге j, $j = 1, 2, \ldots, n$, мы рассматриваем не все точки $t \in [0, C]$, $t \in Z$, но только точки, в которых меняется оптимальное частичное решение или меняется аналитический вид функции $f_j(t)$. Для некоторых задач (для некоторых целевых функций), количество таких точек M небольшое, и поэтому графический алгоритм имеет трудоёмкость $O(n \min\{C, M\})$ операций, вместо $O(nC)$ операций как в алгоритме динамического программирования.

Более того, *GrA* имеет следующие преимущества перед *DPA*:

- с помощью *GrA* можно решать примеры с нецелочисленными параметрами, т.е. примеры, где a_j, b_j, $j = 1, 2, \ldots, n$ и/или C не принадлежат множеству целых чисел Z;
- время работы *GrA* на двух примерах с параметрами $\{a_j, b_j, \}$ и $\{a_j \cdot 10^k \pm 1, b_j \cdot 10^k \pm 1, \cdot 10^k \pm 1\}, k > 1$, совпадает, в то время, как время работы *DPA* будет в 10^k раз больше на втором примере. То есть с помощью *GrA*, можно решать примера "большого масштаба";
- параметры задачи a_j, b_j, $j = 1, \ldots, n$, могут быть и отрицательными. Все версии алгоритмов *DPA* требуют положительность параметров задачи;

- принимаются во внимание внутренние свойства задачи (например, для ЗАДАЧИ О РАНЦЕ, можеть оказаться, что предмет с наименьшим значением $\frac{p_j}{w_j}$ не оказывает влияние на целевую функцию);
- известно, что *GrA* для некоторых задач имеет полиномиальную трудоёмкость, в то время, как исходный алгоритм *DPA* – псевдополиномиальную. Или же *GrA* существенно сокращает трудоёмкость *DPA*.

Графический алгоритм для ЗАДАЧИ О РАНЦЕ

Для ЗАДАЧИ О РАНЦЕ рекурсивные уравнения Беллмана имеют вид:

$$f_j(t) = \max \begin{cases} \Phi^1(t) = p_j + f_{j-1}(t - w_j), & j = 1, 2, \ldots, n; \\ \Phi^2(t) = f_{j-1}(t), & j = 1, 2, \ldots, n. \end{cases} \tag{2.6}$$

где

$$f_0(t) = 0, \quad t \geq 0,$$
$$f_0(t) = +\infty, \ t < 0.$$

Функция $\Phi^1(t)$ в *DPA* соответствует значению $x_j = 1$ (т.е. предмет j помещается в рюкзак), а $\Phi^2(t)$ соответствует значению $x_j = 0$ (т.е. предмет j не берется). На каждом шаге j, $j = 1, 2, \ldots, n$, алгоритма *DPA* значения функции $f_j(t)$ вычисляются в каждой точке (т.е. для каждого состояния системы) $0 \leq t \leq C$. Для каждого значения t, в памяти компьютера сохраняется частичное оптимальное решение $X(t) = (x_1(t), x_2(t), \ldots, x_j(t))$.

Идея *GrA* для ЗАДАЧИ О РАНЦЕ заключается в следующем. На каждом шаге j *GrA*, мы сохраняем $f_j(t)$ в табличном виде, представленном в таб. 2.3, где $0 = t_1 < t_2 < \ldots < t_{m_j}$ и $W_1 < W_2 < \ldots < W_{m_j}$. Эти данные означают следующее. Для каждого значения $t \in [t_l, t_{l+1}),\ 1 \leq l < m_j$, мы имеем частичное оптимальное решение $X_j = (x_1, x_2, \ldots, x_j)$ и соответствующее значение функции $f_j(t) = \sum_{i=1}^{j} p_i \cdot x_i = W_j$. Точки t_l называются *точками излома* (или *точками разрыва*, как в данном случае), т.е. имеем $f_j(t') < f_j(t'')$ для $t' < t_l \leq t''$, $1 \leq l < m_j$.

На следующем шаге $j+1$, мы трансформируем функцию $f_j(t)$ в функции

$$\Phi^1(t) = p_{j+1} + f_j(t - w_{j+1}) \qquad \text{и} \qquad \Phi^2(t) = f_j(t)$$

Таблица 2.3: Функция $f_j(t)$ в *GrA* для ЗАДАЧИ О РАНЦЕ

t	t_1	t_2	...	t_{m_j}
$f_j(t)$	W_1	W_2	...	W_{m_j}
частичное оптим. решение $X(t)$	X_1	X_2	...	X_{m_j}

Таблица 2.4: Сокращение числа интервалов для функции $f_{j+1}(t)$

t	...	t_l	t_{l+1}	...
$f_{j+1}(t)$	...	W_l	$W_{l+1} = W_l$	...
частичное оптим. решение	...	X_l	X_{l+1}	...

за $O(m_j)$ операций, т.е. график функции $\Phi^1(t)$ строится из графика функции $f_j(t)$ "смещением вверх" на p_{j+1} и "смещением вправо" на w_{j+1}. В каждой таблице, соответствующей функциям $\Phi^1(t)$ и $\Phi^2(t)$, мы имеем не более m_j точек. Далее мы вычисляем новую таблицу, соответствующую функции

$$f_{j+1}(t) = \max\{\Phi^1(t), \Phi^2(t)\}$$

за $O(m_j)$ операций. В таблице $f_{j+1}(t)$ при этом будет не более $2m_j$ точек разрыва (обычно, число точек существенно меньше). Фактически мы не рассматриваем все точки t из интервала $[0, C]$, но только точки, в которых изменяется значение целевой функции.

Если мы встретили ситуацию, подобную ситуации из таб. 2.4, то мы удаляем столбец, соответствующую точке t_{l+1}, т.е. мы соединяем два интервала $[t_{l-1}, t_l)$ и $[t_l, t_{l+1})$, для которых мы устанавливаем одно и то же частичное оптимальное решение: X_l.

Продемонстрируем работу *GrA* на том же примере, на котором иллюстрировался *DPA*. Результаты вычислений представлены в таблицах 2.5, 2.6, 2.7, 2.8.

На втором шаге алгоритма *GrA* $w_2 = 3$, поэтому необходимо рассмотреть интервалы, образованные точками $0, 2, 0+3, 2+3$, чтобы построить функцию $f_2(t)$. Результаты приведены в таб. 2.6.

Таблица 2.5: Шаг $j = 1$

t	0	2
$f_1(t)$	0	5
$X(t)$	(0, , ,)	(1, , ,)

Таблица 2.6: Шаг $j = 2$

t	0	2	3	5
$f_2(t)$	0	5	7	12
$X(t)$	(0, 0, ,)	(1, 0, ,)	(0, 1, ,)	(1, 1, ,)

Таблица 2.7: Шаг $j = 3$

t	0	2	3	5	8
$f_3(t)$	0	5	7	12	13
$X(t)$	(0, 0, 0,)	(1, 0, 0,)	(0, 1, 0,)	(1, 1, 0,)	(0, 1, 1,)

Чтобы построить функцию $f_3(t)$, необходимо рассмотреть интервалы, образованные точками 0, 2, 3, 5, $0 + 5$, $2 + 5$, $3 + 5$, т.к. $w_3 = 5$. Точка $5 + 5 > 9$ не рассматривается. Полученные результаты приведены в таб. 2.7.

На последнем шаге, необходимо рассмотреть интервалы, образованные точками 0, 2, 3, 5, 8, $0 + 7$, $2 + 7$, чтобы построить функцию $f_4(t)$. Точки $3 + 7$, $5 + 7$, $8 + 7$ не рассматриваются, т.к. они больше $C = 9$. Поэтому необходимо рассмотреть только 5 точек. Полученные результаты приведены в таб. 2.8.

Графические изображения полученных функций представлены на рис. 2.1.

Очевидно, что трудоёмкость *GrA* составляет $O(\min\{2^n, n \min\{C, F_{opt}\}\})$ операций, где F_{opt} – оптимальное значение целевой функции.

Для данного числового примера, в процессе работы *GrA* было сохранено (рассмотрено) 12 точек разрыва: 1 на первом шаге ($t = 2$), 3 на втором ($t = 2, 3, 5$), 4 на третьем ($t = 2, 3, 5, 8$), и 4 на последнем ($t = 2, 3, 5, 8$), в отличие от *DPA*, где было рассмотрено $4 \times 9 = 36$ точек, т.е. при помощи *GrA* трудоёмкость решения может быть существенно уменьшена.

Позднее будет приведен метод Ветвей и Границ (Branch and Bounds,

Таблица 2.8: Шаг $j = 4$

t	0	2	3	5	8
$f_4(t)$	0	5	7	12	13
$X(t)$	(0, 0, 0, 0)	(1, 0, 0, 0)	(0, 1, 0, 0)	(1, 1, 0, 0)	(0, 1, 1, 0)

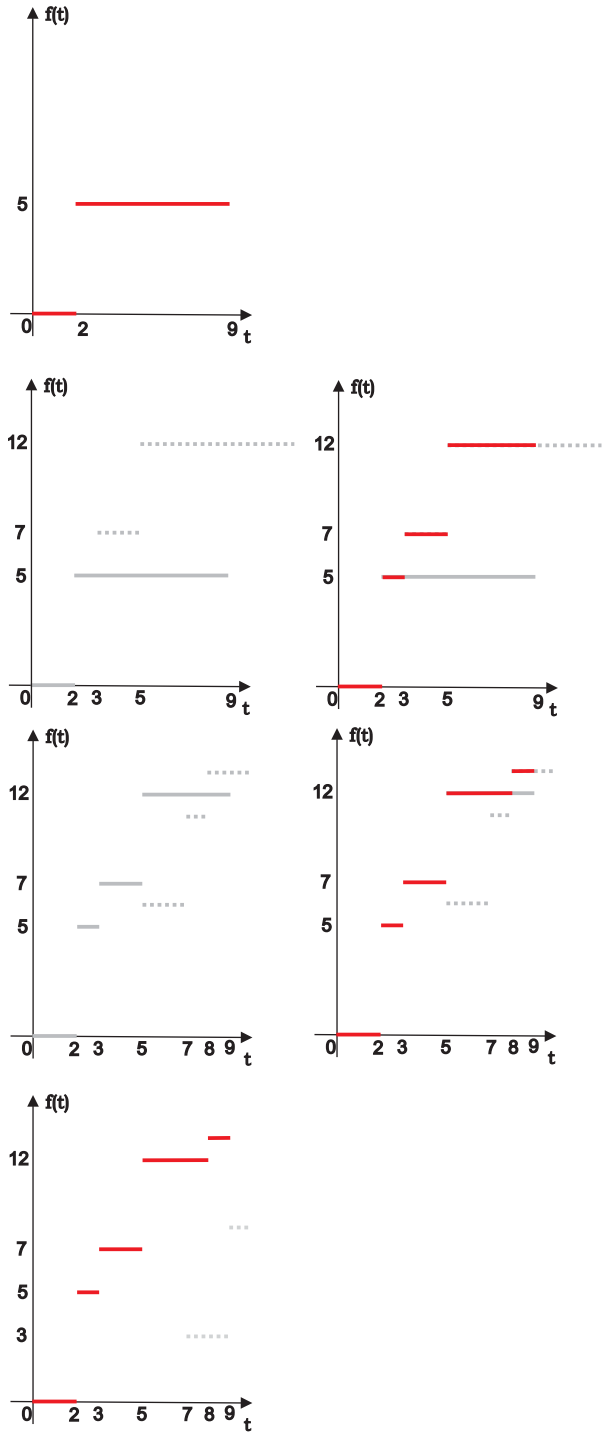

Рис. 2.1: Иллюстрация к графическому алгоритму

B&B) решения оптимизационных задач. Принято считать, что алгоритмы B&B в целом эффективнее (быстрее) алгоритма DPA для ЗАДАЧИ О РАНЦЕ. Тем не менее существуют классы примеров, на которых алгоритмы DPA и GrA быстрее чем B&B.

Известно, что для следующего типа примеров ЗАДАЧИ О РАНЦЕ известные алгоритмы B&B имеют экспоненциальную трудоёмкость:

$$\begin{cases} f(x) = \sum_{j=1}^{n} 2 \cdot x_j \to \max \\ \sum_{j=1}^{n} 2 \cdot x_j \le 2 \left\lfloor \frac{n}{2} \right\rfloor + 1. \end{cases} \tag{2.7}$$

Рассмотрим более общий случай:

$$\begin{cases} f(x) = \sum_{j=1}^{n} a \cdot 2 \cdot x_j \to \max \\ \sum_{j=1}^{n} a \cdot 2 \cdot x_j \le a \cdot \left(2 \left\lfloor \frac{n}{2} \right\rfloor + 1 \right), \end{cases} \tag{2.8}$$

где $a > 1$.

Легко доказать, что для любого a, графический алгоритм GrA имеет трудоёмкость $O(n)$ операций, т.к. на каждом шаге GrA, рассматриваются только следующие значения целевой функции – $\{0, a, 2a, \ldots, n \cdot a\}$. Для первого типа примеров, трудоёмкость DPA составляет не меньше $n \cdot 2 \left\lfloor \frac{n}{2} \right\rfloor$ операций, а для второго примера не меньше $n \cdot a \cdot 2 \left\lfloor \frac{n}{2} \right\rfloor$ операций. Необходимо заметить, что алгоритмы решения задачи (2.7), основанные на методе B&B имеют экспоненциальную трудоёмкость $O\left(\frac{2^n}{\sqrt{n}} \right)$.

Рассмотрим другой нетривиальный частный случай ЗАДАЧИ О РАНЦЕ [88]. Пусть задано множество чисел $B = \{b_1, b_2, \ldots, b_n\}$, $b_i \in R^+$, $i = 1, 2, \ldots, n$. Необходимо:

$$\begin{cases} f(x) = \sum_{j=1}^{m+n} p_j x_j \to \max \\ \sum_{j=1}^{m+n} p_j x_j \le ka + 1, \\ a \ge 2, \\ p_j = a, \ j = 1, 2, \ldots, m, \\ p_j = b_j \cdot a, \ j = m+1, 2, \ldots, m+n, \\ x_j \in \{0, 1\}, \ j = 1, 2, \ldots, m+n. \end{cases} \tag{2.9}$$

Очевидно, что трудоёмкость *GrA* для данного частного случая не зависит от a и равно $O(m+n\min\{k,2^n\})$, где $k \le m+\sum\limits_{i=1}^{n} b_i$, что существенно меньше трудоёмкости алгоритмов типа B&B и трудоёмкости *DPA*.

2.3.5 Алгоритм динамического программирования для задачи о двух конвейерах

Не все алгоритмы динамического программирования имеют псевдополиномиальную трудоёмкость. Алгоритм динамического программирования для ЗАДАЧИ КОММИВОЯЖЁРА имеет экспоненциальную трудоёмкость. В данном параграфе представлен алгоритм *DPA* решения ЗАДАЧИ О ДВУХ КОНВЕЙРАХ, трудоёмкость которого полиномиальна.

ЗАДАЧА О ДВУХ КОНВЕЙЕРАХ

Автомобильный завод оснащен двумя конвейерами. На оба конвейера, на каждом из которых предусмотрено по несколько рабочих мест, поступают для сборки автомобильные шасси, после чего на каждом рабочем месте конвейера добавляются все новые детали. Наконец, собранный автомобиль покидает конвейер. На каждом конвейере имеется n рабочих мест, пронумерованных от 1 до n. Обозначим рабочее место под номером $j,\ j = 1, 2, \ldots, n$, на конвейере $i,\ i = 1, 2$, через $M_{i,j}$. На обоих конвейерах на рабочих местах с одинаковыми номерами выполняются один и те же операции. Однако конвейеры создавались в разное время и по разным технологиям, поэтому время выполнения одних и тех же операций на разных конвейерах различается. Обозначим время сборки на рабочем месте $M_{i,j}$ через $p_{i,j}$.

Для каждого из двух конвейеров задано время a_i – постановка шасси на конвейер (перед первой операцией) и время b_i – снятие автомобиля с конвейера (после последней операции), $i = 1, 2$.

Обычно поступившее на конвейер шасси проходит все этапы сборки на одном и том же конвейере. Временем перехода от одного рабочего места к другому, если этот переход выполняется в пределах одного конвейера, можно пренебречь. Однако иногда заказчик требует, чтобы автомобиль был собран за кратчайшее время, и такой порядок приходится нарушить.

Подобная ситуация возникает, когда на автозавод поступают автомобили с выявленными конструктивными недостатками. Для ускорения сборки шасси по-прежнему проходит все n рабочих мест в обычном порядке, однако менеджер может дать указание перебросить частично собранный автомобиль с одного конвейера на другой, причем такая переброска возможна на любом этапе сборки. Время, которое требуется для перемещения шасси с конвейера i на соседний после прохождения рабочего места $M_{i,j}$, равно $t_{i,j},\ j = 1, 2, \ldots, n-1$, (поскольку после прохождения n-го рабочего места сборка завершается). Задача заключается в том, чтобы определить, какие рабочие места должны быть выбраны на первом конвейере, а какие – на втором, чтобы минимизировать общее время, затраченное на заводе на сборку ОДНОГО автомобиля.

На примере этой задачи будет показано, как строится алгоритм *DPA* и в чем состоит суть принципа оптимальности Беллмана.

Рассмотрим способ, при котором шасси, поступившее на первый конвейер, попадет на рабочее место $M_{1,j}$ за кратчайшее время. Если $j = 1$, существует всего один способ прохождения такого отрезка пути, поэтому легко определить время достижения рабочего места $M_{1,j}$. Однако, если $j = 2, 3, \ldots, n$, то возможен один из двух вариантов: на рабочее место $M_{1,j}$ шасси могло попасть непосредственно с рабочего места $M_{1,j-1}$ или с $M_{2,j-1}$. В первом случае временем перехода от одного рабочего места к другому можно пренебречь, а во втором переход займет время $t_{2,j-1}$. Рассмотрим обе эти возможности отдельно, хотя впоследствии станет ясно, что у них много общего.

Сначала предположим, что самый быстрый путь через рабочее место $M_{1,j}$, проходит также через рабочее место $M_{1,j-1}$ Основной вывод, который можно сделать в этой ситуации, заключается в том, что отрезок пути от начальной точки до $M_{1,j-1}$ тоже должен быть самым быстрым. Почему? Если бы на это рабочее место можно было бы попасть быстрее, то при подстановке данного отрезка в полный путь получился бы более быстрый путь к рабочему месту $M_{1,j}$, а это противоречит начальному предположению.

Теперь предположим, что самый быстрый путь, проходящий через рабочее место $M_{1,j}$ проходит также через рабочее место $M_{2,j-1}$. В этом случае можно прийти к выводу, что на рабочее место $M_{2,j-1}$ шасси должно

попасть за кратчайшее время. Объяснение аналогично предыдущему: если бы существовал более быстрый способ добраться до рабочего места $M_{2,j-1}$, то при подстановке данного отрезка в полный путь получился бы более быстрый путь к рабочему месту $M_{1,j}$, а это снова противоречит сделанному предположению.

Обобщая эти рассуждения, можно сказать, что задача по составлению оптимального расписания (определение самого быстрого пути до рабочего места $M_{i,j}$) содержит в себе оптимальное решение двух подзадач – нахождение самого быстрого пути до рабочего места $M_{1,j-1}$ и $M_{2,j-1}$. С помощью этого свойства можно показать, что определение оптимального решения задачи сводится к определению оптимального решения ее подзадач. Обратите внимание, как это перекликается со свойством оптимальности Беллмана.

В задаче по составлению расписания работы конвейера рассуждения проводятся следующим образом. Наиболее быстрый путь, проходящий через рабочее места $M_{1,j}$, должен также проходить через рабочее место под номером $j-1$, расположенное на первом или втором конвейере. Таким образом, если самый быстрый путь проходит через рабочее место $M_{1,j}$, то справедливо одно из таких утверждений:

- этот путь проходит через рабочее место $M_{1,j-1}$ после чего шасси попадает непосредственно на рабочее место $M_{1,j}$;
- этот путь проходит через рабочее место $M_{2,j-1}$, после чего автомобиль “перебрасывается” со второго конвейера на первый, а затем попадает на рабочее место $M_{1,j}$;

Аналогичные выводы можно сделать и для позиции $M_{2,j}$.

Чтобы решить задачу определения наиболее быстрого пути до рабочего места j, которое находится на каждом из конвейеров, нужно решить вспомогательную задачу определения самого быстрого пути до рабочего места $j-1$ (также на любом из конвейеров). Таким образом, оптимальное решение задачи об оптимальном расписании работы конвейера можно найти путем поиска оптимальных решений подзадач. Обратите внимания, что этот вывод в точности соответствует основной идее метода динамического программирования, в которой исходная задача разбивается

на подзадачи.

Теперь мы можем задать рекурсивные отношения. Необходимо рекурсивно определить оптимальное решение в терминах оптимальных решений подзадач. В задаче по составлению расписания работы конвейера роль подзадач играют задачи по определению самого быстрого пути, проходяшего через j-ое рабочее место на любом из двух конвейеров для $j = 2, 3, \ldots, n$. Обозначим через $f_{i,j}$ минимальное время, в течение которого шасси проходит от стартовой точки до рабочего места $M_{i,j}$.

Конечная цель заключается в том, чтобы определить кратчайшее время f^*, в течение которого шасси проходит по всему конвейеру. Для этого автомобиль, который находится в сборке, должен пройти весь путь до рабочего места n, находящегося на первом или втором конвейере, после чего он покидает цех. Поскольку самый быстрый из этих путей является самым быстрым путем прохождения всего конвейера, справедливо следующее соотношение:

$$f^* = \min\{f_{1,n} + b_1, f_{2,n} + b_2\}.$$

Легко также сделать заключение по поводу величин $f_{1,1}$ и $f_{1,2}$. Чтобы за кратчайшее время пройти первое рабочее место на любом из конвейеров, шасси должно попасть на это рабочее непосредственно. Таким образом, можно записать уравнения

$$f_{1,1} = a_1 + p_{1,1}, \quad f_{2,1} = a_2 + p_{2,1}.$$

Напомним, что a_1 и a_2 – это время движения шасси автомобиля до первого и второго конвейера, соответственно; b_1 и b_2 – время выхода автомобиля, с соответствующего конвейера, из цеха. А теперь рассмотрим, как вычислить величину $f_{i,j}$, $j = 2, 3, \ldots, n$, $i = 1, 2$. Рассуждая по поводу $f_{1,j}$, мы пришли к выводу, что самый быстрый путь до рабочего места $M_{1,j}$ является также либо самым быстрым путем до рабочего места $M_{1,j-1}$ после чего шасси попадает прямо на рабочее место $M_{1,j}$, либо самым быстрым путем до рабочего места $M_{2,j-1}$, с последующим переходом со второго конвейера на первый. В первом случае можно записать соотношение $f_{1,j} = f_{1,j-1} + p_{1,j-1}$, а во втором $f_{1,j} = f_{2,j-1} + t_{2,j-1} + p_{2,j-1}$. Таким образом, при $j = 2, 3, \ldots, n$, выполняется уравнение

$$f_{1,j} = \min\{f_{1,j-1} + p_{1,j-1},\ f_{2,j-1} + t_{2,j-1} + p_{2,j-1}\}.$$

Аналогично можно записать:

$$f_{2,j} = \min\{f_{2,j-1} + p_{2,j-1},\ f_{1,j-1} + t_{1,j-1} + p_{1,j-1}\}.$$

Величины $f_{i,j}$ являются значениями, соответствующим оптимальным решениям подзадач. Чтобы было лучше понять, как конструируется оптимальное решение, обозначим через $x_{i,j},\ i = 1, 2,\ j = 2, 3, \ldots, n$, номер конвейера (1 или 2), содержащего рабочее место под номером $j-1$, через которое проходит самый быстрый путь к рабочему месту $M_{i,j}$. Кроме того, обозначим через x^* номер конвейера, через n-е рабочее место которого проходит самый быстрый полный путь сборки. На каждом шаге алгоритма для нахождения f_{ij} требуется $O(1)$ операций, поэтому трудоёмкость алгоритма $O(n)$ операций.

Рассмотрим пример, изображенный на рисунке 2.2. На рисунке в кружочках, соответствующих рабочим местам указаны значения $p_{i,j}$, значения $t_{i,j},\ a_i,\ b_i$ указаны на диагональных стрелках. Процесс решения примера показан в таб. 2.9. Для этого примера $f^* = 38$.

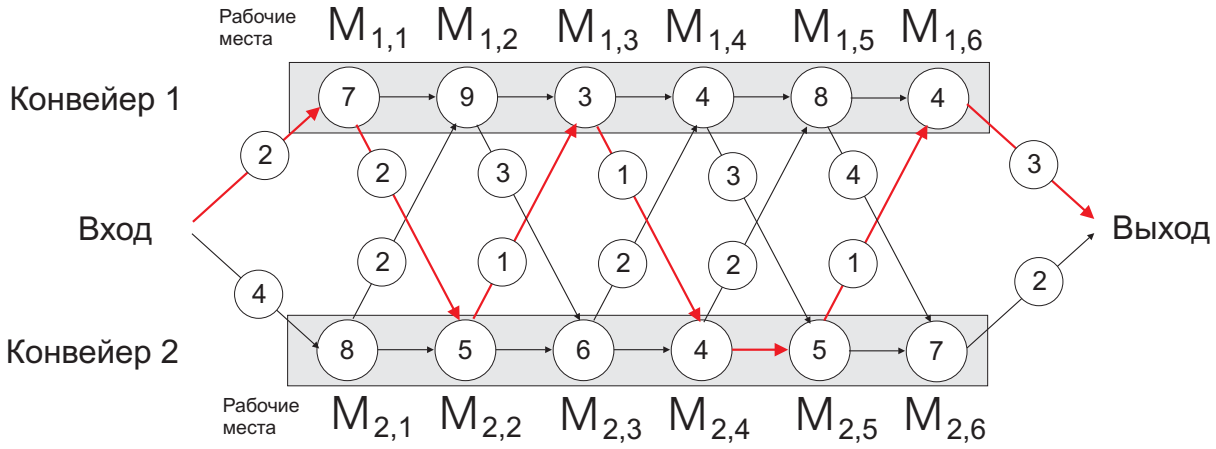

Рис. 2.2: Пример задачи с двумя конвейерами

Таблица 2.9: Процесс вычисления и построение технологического маршрута

j	1	2	3	4	5	6
$f_{1,j}$	9	18	20	24	32	35
$f_{2,j}$	12	16	22	25	30	37

j	2	3	4	5	6
$x_{1,j}$	1	2	1	1	2
$x_{2,j}$	1	2	1	2	1

Начнем с $x^* = 1$, при этом используется рабочее место $M_{1,6}$. Теперь посмотрим на величину $x_{1,6}$, которая равна 2, из чего следует, что используется рабочее место $M_{2,5}$. Продолжая рассуждения, смотрим на величину $x_{2,5} = 2$ – используется рабочее место $M_{2,4}$, величину $x_{2,4} = 1$ (рабочее место $M_{1,3}$), величину $x_{1,3} = 2$ (рабочее место $M_{2,2}$) и величину $x_{2,2} = 1$ – рабочее место $M_{1,1}$.

Получили следующее оптимальное решение: "Конвейер 1 – рабочее место 6; Конвейер 2 – рабочее место 5; Конвейер 2 – рабочее место 4; Конвейер 1 – рабочее место 3; Конвейер 2 – рабочее место 2; Конвейер 1 – рабочее место 1".

Данный алгоритм может быть распространен и на случай m параллельных конвейеров. В этом случае трудоёмкость алгоритма составит $O(m\,n)$ операций, т.к. на каждом шаге j, $j = 1, \ldots, n$, требуется $O(m)$ операций. Также заметим, что объём необходимой для решения задачи памяти составляет $o(m\,n)$.

2.3.6 Метод Ветвей и Границ

Метод Ветвей и Границ (B&B) широко используется для нахождения точного (оптимального) решения задач дискретной оптимизации.

Нам понадобится следующее определение:

Подзадачей задачи (2.1) называется задача $f(x^0) = \min\limits_{x \in G'} f(x)$, где $G' \subseteq G$.

Далее перечислены три основные элемента метода B&B решения задачи (2.1):

- **Ветвление**. Множество G разбивается на подмножества $G_i \subseteq G$, $i = 1, 2, \ldots, r$, причем $\bigcup\limits_{i=1}^{r} G_i = G$. Таким образом, исходная задача (2.1) разбивается на подзадачи, определенные подмножествами допустимых решений G_i, $i = 1, 2, \ldots, r$. Этот процесс называется *ветвлением*. Ветвление – это рекурсивный процесс, т.е. каждая подзадача G_i в свою очередь является базисом для другого ветвления. В процессе ветвления образуется *дерево поиска* оптимального

решения. При этом задача G называется *корнем*, а подзадачи G_i – *ветками* или *потомками*;

- **Верхняя оценка (граница)**. Верхней оценкой для позадачи G_i называется значение $UB_i \geq \min\limits_{x \in G_i} f(x)$. Верхняя оценка используется в алгоритме, чтобы предварительно оценить перспективность той или иной подзадачи, т.е. оценить возможность того, что подзадача содержит оптимальное решение исходной задачи;

- **Нижняя оценка (граница)**. Нижней оценкой для позадачи G_i называется значение $LB_i \leq \min\limits_{x \in G_i} f(x)$. В процессе работы алгоритма Ветвей и Границ последовательно строится несколько допустимых решений. В памяти компьютера мы храним лучшее из построенных допустимых решений (лучшее, с точки зрения целевой функции). Этому лучшему решению соответствует значение целевой функции, которые мы называем *текущим РЕКОРДом*. Пусть в процессе работы алгоритма для некоторой подзадачи G_i мы вычислили нижнюю оцену LB_i. Если LB_i больше текущего РЕКОРДа, значит подмножество G_i не содержит оптимальное решение исходной задачи, а следовательно, не имеет смысла рассматривать подмножество G_i в дальнейшем. Таким образом ветку, соответствующую подмножеству G_i, в дереве поиска можно *отсечь*.

Чтобы построить алгоритм, основанный на методе Ветвей и Границ, необходимо определить способ ветвления и способы вычисления нижних и верхних оценок. Иногда верхнюю оценку не вычисляют. Для задачи максимизации роли нижней и верхней оценок взаимозаменяются. Формально опишем алгоритм 3 Ветвей и Границ.

При реализации алгоритма существенным является вопрос: в каком порядке рассматривать “висячие” ветви в дереве поиска? Обычно для продолжения ветвления выбирается подзадача с наименьшей нижней оценкой (или верхней оценкой UB_i) или подзадача с наименьшим значением $|G_i|$. Как правило, трудоёмкость алгоритма B&B растет экспоненциально с ростом размерности задачи.

Algorithm 3 Алгоритм Ветвей и Границ для задачи минимизации

1: Найдем некоторое допустимое решение $x \in G$. Это может быть сделано, например, при помощи какого-нибудь эвристического алгоритма;
2: РЕКОРД := $f(x)$;
3: Branch&Bound(G);

Procedure Branch&Bound(G)

1: Разбиваем G на подмножества $G_1, \ldots, G_r$, где $\bigcup_{i=1}^{r} G_i = G$;
2: **for** КАЖДОГО подмножества G_i **do**
3: Вычислим нижнюю оценку LB_i для G_i;
4: **if** $LB_i <$РЕКОРД **then**
5: **if** G_i содержит только одно решение x^i **then**
6: РЕКОРД:= min{РЕКОРД, $f(x^i)$};
7: **else**
8: Branch&Bound(G_i);
9: **end if**
10: **end if**
11: **end for**

Алгоритм Ветвей и Границ для ЗАДАЧИ О РАНЦЕ

Способ ветвления зададим следующим образом. Множество G разбивается на два подмножества G_0 и G_1. В подмножестве G_0 находятся все решения, соответствующие $x_1 = 0$, то есть первый предмет в рюкзак не кладется, а в подмножестве G_1 решения, соответствующие $x_1 = 1$. Аналогичным образом разбиваются на подмножества G_0 и G_1, исходя из двух возможных значений $x_2 = 0$ или $x_2 = 1$ и т.д.

Каждой подзадаче G' в этом дереве поиска соответствует ситуация, когда значения некоторых переменных $x_1, x_2, \ldots, x_{j-1}$ уже фиксированы. Перед очередным ветвлением, т.е. перед выбором значения для переменной x_j необходимо проверить, "помещается" ли предмет j в рюкзак. Если $\sum_{k=1}^{j-1} w_k x_k + w_j > C$, тогда для этой ветки (подзадачи) фиксируется $x_j = 0$, и работа продолжается только по ветке $G_0 = G'$.

Теперь опишем алгоритм вычисления верхней оценки для некоторой подзадачи, где значения переменных $x_1, x_2, \ldots, x_{j-1}$ уже фиксированы. В качестве нижней оценки мы будем рассматривать сумму оптимального решения релаксированной (упрощенной) ЗАДАЧИ О РАНЦЕ и значе-

ния $\sum_{k=1}^{j-1} p_k x_k$. В этой упрощенной задаче $0 \le x_i \le 1$, $i = 1, 2, \ldots, n$, т.е. x_i может принимать любое значение из интервала $[0, 1]$. Известно, что следующий алгоритм 4 Данцига позволяет найти оптимальное решение для данной релаксированной задачи:

Algorithm 4 Алгоритм Данцига решения релаксированной ЗАДАЧИ О РАНЦЕ

1: Для каждого предмета $j = 1, 2, \ldots, n$, вычислим $\lambda_j = \dfrac{p_j}{w_j}$;
2: Перенумеруем требования согласно правилу $\lambda_1 \ge \lambda_2 \ge \cdots \ge \lambda_n$;
3: $F := 0$;
4: $C_0 := C$;
5: $x_i := 0,\ i = 1, 2, \ldots, n$;
6: **for** $j := 1$ to n **do**
7: **if** $C_0 \ge w_j$ **then**
8: $x_j := 1$, $C_0 := C_0 - w_j,\ F := F + p_j$;
9: **else**
10: Прерываем цикл;
11: **end if**
12: **end for**
13: $x_j := \dfrac{C - \sum_{i:=1}^{j-1} x_i w_i}{w_j}$;
14: $F := F + p_j x_j$;

Фактически по жадному алгоритму Данцига происходит последовательное заполнение ранца наиболее “ценными” предметами и только один предмет помещается в ранце “частично”.

Если в подзадаче G' значения некоторых переменных $x_1, x_2, \ldots, x_{j-1}$ уже фиксированы, то мы применяем алгоритм Данцига для релаксированной подзадачи (со множеством переменных $x_j, x_{j+1}, \ldots, x_n \in [0, 1]$) и находим соответствующее оптимальное значение целевой функции F. Тогда верхняя оценка для этой подзадачи $UB := F + \sum_{k=1}^{j-1} p_k x_k$. Рассмотрим работу алгоритма на примере 2.3. Дерево поиска для данного примера представлено на рис. 2.3.

Каждый кружок на рисунке соответствует подзадаче. Номера подзадач указаны внутри кружков римскими цифрами, а соответствующие верхние оценки арабскими цифрами.

Для исходной задачи I имеем $UB_I = 5 + 7 + 6\dfrac{9-2-3}{5} = 16,8$.

В подзадаче II, соответствующей ситуации $x_1 = 1$, имеем $UB_{II} = 5+7+6\dfrac{9-2-3}{5} = 16,8$, а в подзадаче III, соответствующей ситуации $x_1 = 0$, получаем $UB_{III} = 7+6+3\dfrac{9-3-5}{7} = 13\dfrac{3}{7}$. Первоначально для ветвления выбираем подзадачу II. В подзадаче IV, соответствующей ситуации $x_1 = 1, x_2 = 1$, имеем $UB_{IV} = 16,8$, а в подзадаче V, соответствующей ситуации $x_1 = 1, x_2 = 0$, имеем $UB_V = 11\dfrac{6}{7}$.

Для ветвления выбираем подзадачу IV, но в этой задаче существует только одно допустимое решение $(1, 1, 0, 0)$. Для которого значение целевой функции равно 12.

Далее для ветвления выбираем задачу III. В подзадаче $VIII$, соответствующей ситуации $x_1 = 0, x_2 = 0$, имеем $UB_{VIII} = 7\dfrac{5}{7}$, а в подзадаче VII, соответствующей ситуации $x_1 = 0, x_2 = 1$, получаем $UB_{VII} = 13\dfrac{3}{7}$. Выбираем для ветвления задачу VII. Здесь имеем две подзадачи IX и X с двумя допустимыми решениями $(0, 1, 0, 1)$ со значением целевой функции равным 8 и $(0, 1, 1, 0)$ со значением целевой функции равным 13.

Так как UB_V и UB_{VIII} меньше 13, то подзадачи V и $VIII$ не рассматриваем. Оптимальное решение задачи $(0, 1, 1, 0)$. Оптимальное значение целевой функции равно 13.

Стоит отметить, что все известные алгоритмы Ветвей и Границ для ЗАДАЧИ О РАНЦЕ имеют экспоненциальную трудоёмкость.

2.4 Задача о назначениях

Частным случаем известной транспортной задачи является ЗАДАЧА О НАЗНАЧЕНИЯХ. Формулируется она следующим образом. Дано n заданий и n исполнителей. На каждое задание $i,\ i = 1, 2, \ldots, n$, необходимо назначить одного исполнителя $j,\ j = 1, 2, \ldots, n$. Каждый исполнитель должен быть назначен только на одно задание. Задано время выпол-

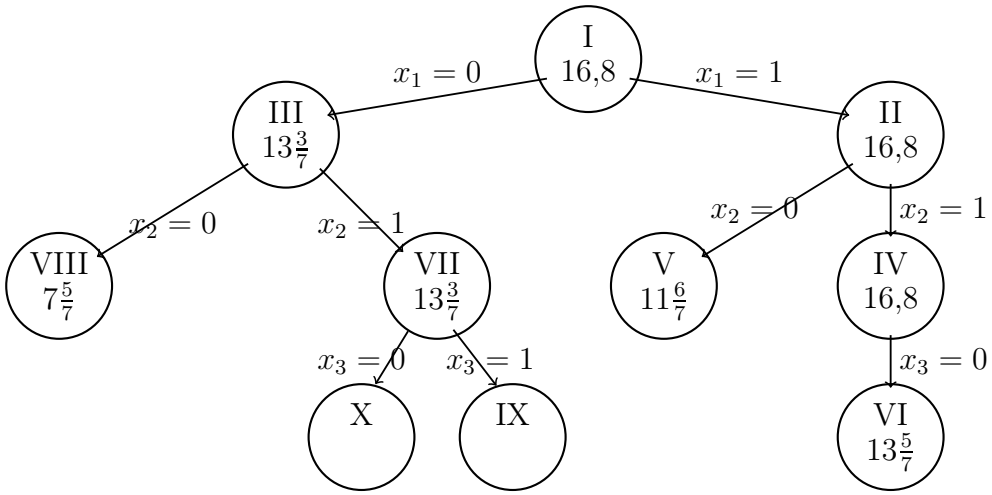

Рис. 2.3: Дерево поиска в алгоритме Ветвей и Границ для ЗАДАЧИ О РАНЦЕ

нения $a_{ij} \geqslant 0$ задания i исполнителем j. Необходимо минимизировать общую сумму времён, потраченного на выполнение всех заданий.

Формально для данной задачи можно определить n^2 булевых переменных $x_{ij} \in \{0, 1\}$, $i, j = 1, 2, \ldots, n$, где $x_{ij} = 1$, если задание i назначается на исполнителя j, иначе $x_{ij} = 0$.

Далее представлен т.н. *венгерский* алгоритм решения задачи и числовой пример. Алгоритм состоит из трех этапов.

Венгерский алгоритм решения ЗАДАЧИ О НАЗНАЧЕНИЯХ

Этап 1.

1. Представим условия задачи в виде квадратной матрицы (таблицы) n x n, содержащей значения a_{ij}. Строки – задания, столбцы – исполнители;
2. В каждой строке i, $i = 1, 2, \ldots, n$, таблицы найдем наименьший элемент a_{ij} и вычтем его из всех элементов данной строки;
3. Повторим ту же самую процедуру для столбцов.

Теперь в каждой строке и в каждом столбце таблицы есть по крайней мере один нулевой элемент. Такая таблица называется “приведённой”. Для полученного примера и для примера с “исходной” таблицей оптимальное

решение будет одним и тем же, но будут отличаться значения целевой функции.

Представленная с помощью описанного выше приема "приведённой" транспортной таблице задача о назначениях эквивалентна исходной задаче, и оптимальное решение для обеих задач будет одним и тем же. Сущность Венгерского алгоритма заключается в продолжении процесса приведения матрицы до тех пор, пока все подлежащие распределению единицы x_{ij} не попадут в клетки с нулевой стоимостью. Это означает, что итоговое значение приведенной целевой функции будет равно нулю. Так как существует ограничение на неотрицательность переменных ($a_{ij} \geqslant 0$), нулевое значение целевой функции будет оптимальным.

Этап 1 алгоритма повторяется до тех пор пока в каждой строке и каждом столбце приведённой таблицы будет находиться хотя бы одно нулевое значение.

Этап 2.

Если некоторое решение является допустимым, то каждой строке и каждому столбцу соответствует только один элемент. Если процесс распределения элементов осуществляется только в клетки с нулевой стоимостью, то он приведет к получению минимального значения целевой функции.

1. Найдем строку i, содержащую только одно нулевое значение стоимости, и в клетку, соответствующую данному значению, поместим единицу. Если такие строки отсутствуют, можно начать с любого нулевого значения стоимости.

2. Зачеркнем оставшиеся нулевые значения данного столбца.

3. Пункты 1 и 2 повторяем до тех пор, пока продолжение описанной процедуры окажется невозможным.

   Если на данном этапе окажется, что есть несколько нулей, которым не соответствуют назначения и которые являются незачеркнутыми, то необходимо выполнить пункты 4 – 5.

4. Найдем столбец, содержащий только одно нулевое значение, и в соответствующую клетку поместим один элемент.

5. Зачеркнем оставшиеся нули в данной строке.

6. Повторяем пункты 4 и 5 до тех пор, пока дальнейшая их реализация окажется невозможной.

Если окажется, что таблица содержит неучтенные нули, повторить операции 1–6. Если решение является допустимым, т.е. все элементы распределены в клетки, которым соответствует нулевая стоимость, то полученное решение одновременно является оптимальным. Если решение является недопустимым, осуществляется переход к этапу 3.

Этап 3.

1. Проведем минимальное число прямых через строки и столбцы матрицы (но не по диагоналям) таким образом, чтобы они проходили через все нули, содержащиеся в таблице.
2. Найдем наименьшее значение среди элементов, через которые не проходит ни одна из проведенных прямых.
3. Вычтем его из всех элементов, через которые не проходят прямые.
4. Прибавим найденный элемент ко всем элементам таблицы, которые лежат на пересечении проведенных ранее прямых.
5. Все элементы матрицы, через которые проходит только одна прямая, оставим без изменения.

В результате применения данной процедуры в таблице появляется по крайней мере один новый ноль. Необходимо возвратиться к этапу 2 и повторять алгоритм до тех пор, пока не будет получено допустиме решение, которое и будет оптимальным.

Пример.

Рассмотрим пример с 4-мя заданиями и 4-мя исполнителями, заданный следующей матрицей:

$$\begin{pmatrix} 68 & 72 & 75 & 83 \\ 56 & 60 & 58 & 63 \\ 38 & 40 & 35 & 45 \\ 47 & 42 & 40 & 45 \end{pmatrix}.$$

В каждой строке найдем наименьший элемент:

$$\begin{matrix} 68 \\ 56 \\ 35 \\ 40 \end{matrix}\begin{pmatrix} 68 & 72 & 75 & 83 \\ 56 & 60 & 58 & 63 \\ 38 & 40 & 35 & 45 \\ 47 & 42 & 40 & 45 \end{pmatrix}.$$

Вычтем наименьший элемент из всех элементов соответствующей строки и найдём наименьший элемент в каждом столбце:

$$\begin{matrix} 0 & 2 & 0 & 5 \end{matrix}\\ \begin{pmatrix} 0 & 4 & 7 & 15 \\ 0 & 4 & 2 & 7 \\ 3 & 5 & 0 & 10 \\ 7 & 2 & 0 & 5 \end{pmatrix}.$$

Вычтем наименьший элемент из всех элементов соответствующего столбца:

$$\begin{pmatrix} 0 & 2 & 7 & 10 \\ 0 & 2 & 2 & 2 \\ 3 & 3 & 0 & 5 \\ 7 & 0 & 0 & 0 \end{pmatrix}.$$

В каждой строке и каждом столбце таблицы у нас имеется не меньше одного нуля.

В соответствии с этапом 2, осуществляем назначения. Наличие назначения обозначается рамкой:

$$\begin{pmatrix} \boxed{0} & 2 & 7 & 10 \\ 0 & 2 & 2 & 2 \\ 3 & 3 & \boxed{0} & 5 \\ 7 & \boxed{0} & 0 & 0 \end{pmatrix}.$$

На данном этапе мы можем осуществить только три нулевых назначения, тогда как требуемое их количество равно четырем. Полученное распределение является недопустимым. Переходим к этапу 3. Проводим наименьшее число прямых, проходящих через все нули таблицы. Полужирным

шрифтом мы обозначили строки и столбцы, через которые проходят эти 3 прямые:

$$\begin{pmatrix} \boxed{\mathbf{0}} & 2 & \mathbf{7} & 10 \\ \mathbf{0} & 2 & \mathbf{2} & 2 \\ \mathbf{3} & 3 & \boxed{\mathbf{0}} & 5 \\ \mathbf{7} & \boxed{\mathbf{0}} & \mathbf{0} & \mathbf{0} \end{pmatrix}.$$

Наименьшим элементом, через который не проходит ни одна из прямых, является число 2. Скорректируем таблицу так, как это описано выше в соответствии с этапом 3, т.е. вычтем 2 из каждого элемента, через который не проходит ни одна прямая, и добавим 2 ко всем элементам, лежащим на пересечении двух прямых, оставив без изменения все прочие элементы, через которые проходит только одна прямая. Теперь перераспределим соответствующие назначения заданий и исполнителей. Возможны три варианта и, как следствие, три оптимальных решения, представленных ниже. Рамкой выделены соответствующие назначения в каждом из решений.

$$\begin{pmatrix} \boxed{0} & 0 & 7 & 8 \\ 0 & \boxed{0} & 2 & 0 \\ 3 & 1 & \boxed{0} & 3 \\ 9 & 0 & 2 & \boxed{0} \end{pmatrix} \begin{pmatrix} \boxed{0} & 0 & 7 & 8 \\ 0 & 0 & 2 & \boxed{0} \\ 3 & 1 & \boxed{0} & 3 \\ 9 & \boxed{0} & 2 & 0 \end{pmatrix} \begin{pmatrix} 0 & \boxed{0} & 7 & 8 \\ \boxed{0} & 0 & 2 & 0 \\ 3 & 1 & \boxed{0} & 3 \\ 9 & 0 & 2 & \boxed{0} \end{pmatrix}.$$

Для первого оптимального решения, в котором на задание 1 назначается исполнитель 1, на задание 2 – исполнитель 2 и т.д., имеем оптимальное значение целевой функции $68 + 60 + 35 + 45 = 208$.

Этим же алгоритмом можно решить задачу, в которой n заданий и m исполнителей, где $n \neq m$. Известно, что трудоёмкость данного алгоритма равна $O(n^3)$ операций.

2.5 Некоторые сведения из теории графов

В задачах комбинаторной оптимизации и, в частности, в задачах **ТР** нередко можно встретить некоторые понятия из Теории Графов.

Теория Графов – раздел дискретной математики, изучающий свойства графов.

Принято считать, что первой задачей этого раздела была задача о кёнигсбергских мостах, решенная великим математиком Л.Эйлером в 18-м веке. Задача эта была поставлена Эйлеру прусским королем, у которого на службе в тот момент состоял Эйлер. Формулируется она следующим образом. В Кёнигсберге (современный Калининград) есть 7 мостов. Каждый из них интересен с эстетической точки зрения. Вопрос – можно ли составить маршрут движения так, чтобы пройти по каждому мосту ровно один раз? Эйлер доказал, что это невозможно.

Новые задачи, возникающие на практике, сложно сразу отнести к какому то разделу науки. Например, не ясно было, к чему относятся первые задачи теории графов, исследования операций и информатики, но, как ни странно, решать их удавалось именно математикам. Так и Теория Графов сформировалась в качестве составной части математики.

В задачах комбинаторной оптимизации (и **ТР**) мы имеем дело со множествами. Удобной и наглядной формой представления множеств с определенными на них отношениями являются графы.

Графом обычно принято называть следующую конструкцию. Заданы n точек, произвольно расположенных на плоскости (в пространстве). От некоторых точек к другим точкам проведены стрелки (или линии), которые часто называют рёбрами.

Графы обозначаются как $G = (V, E)$, где V – множество вершин (точек), а E – множество пар (i, j), где $i, j \in V$, представляемых графически как стрелки или линии. Если нас интересует направление, то пара (i, j) называется *дугой* (изображается как стрелка), а если направление не задано (при этом вершины (точки) соединены не стрелкой, а линией), то пара (i, j) называется *ребром*.

Граф, где все пары (i, j) – дуги называется *ориентированным*, а где все пары являются ребрами – *неориентированным*.

Подграфом графа $G = (V, E)$ называется граф $G' = (V', E')$, где $V' \subseteq V$ и $E' \subseteq E$.

Путем в ориентированном графе $G = (V,)$ называется последовательность дуг $e_1, e_2, \ldots, e_m \in E$, когда конец каждой предыдущей дуги сов-

падает с началом следующей, т.е. когда последовательность дуг имеет вид $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_k)$, где $v_i \in V,\ i = 1, 2, \ldots, k$. Для неориентированного графа существует аналогичное понятие *цепь*.

Если существует путь (или цепь) вида $(v_1, v_2), (v_2, v_3), \ldots, (v_{k-1}, v_1)$, т.е. начало и конец которого совпадают, то говорят, что в графе есть *цикл*.

Если любые две вершины графа можно соединить цепью, тогда граф называется *связным*.

Деревом называется связный граф, имеющий не менее двух вершин и не содержащий циклов, причем любые две вершины связаны единственной цепью.

Цепочкой называют связный граф, не имеющий циклы, все вершины которого принадлежат одному пути (одной цепи).

Библиографическая справка

Подробное описание методов решения задач дискретной оптимизации можно найти в монографиях [10, 11, 12, 22, 27]. Классической книгой, посвященной исследованию классов P и NP считается монография Гэри М. и Джонсона Д. [6].

Альтернативный алгоритм динамического программирования для решения ЗАДАЧИ О РАНЦЕ представлен в известной работе [19]. Лучшим на данный момент алгоритмом динамического программирования для ЗАДАЧИ РАЗБИЕНИЯ считается алгоритм трудоёмкости $O(nb_{\max})$, представленный в работе [71].

Глава 3

Одноприборные задачи ТР

Задачам для одного прибор уделяется в рамках **ТР** особое внимание в силу их тесной связи с классическими задачами из других областей дискретной математики, а также в связи с тем, что одноприборные задачи являются частными случаями и подзадачами более сложных практических задач. В этой главе представлены некоторые алгоритмы решения классических одноприборных задач.

Для одноприборных задач можно выделить следующее важное свойство.

Теорема 3.1 *Если $r_j = 0$ для всех $j = 1, 2, \ldots, n$, и целевая функция $F(C_1, C_2, \ldots, C_n)$ является монотонно неубывающей функцией, зависящей от моментов окончания обслуживания требований C_j, то в задаче минимизации функции F существует оптимальное расписание без прерываний обслуживания требований и простоев прибора.*

Доказательство. Предположим, что во всех оптимальных расписаниях происходят прерывания в обслуживании хотя бы одной работы. Пусть π – оптимальное расписание, при котором обслуживание работы j прерывается. Пусть работа j обслуживается в интервалах времени $[t_1, t_2)$ и $[t_3, t_4)$, где $t_1 < t_2 < t_3 < t_4$, причем требование j не обслуживается в промежутке (t_2, t_3) и до момента времени t_1. Преобразуем расписание π следующим образом. Передвинем часть обслуживания $[t_1, t_2)$ вперед во времени, чтобы требование j обслуживалось без перерывов в промежутке $[t_3-(t_2-t_1), t_4)$. При этом требования, обслуживавшиеся в промежутке времени (t_2, t_3) "сдвинем назад" во времени на величину t_2-t_1. В результате такой операции значение целевой функции не увеличилось, но при

этом мы сократили одно прерывание. Повторив эту операцию необходимое число раз, равное суммарному количеству всех прерываний, мы получим оптимальное расписание без прерываний.

Так как мы минимизируем целевую функцию F и все моменты поступления равны нулю, то очевидно, что всегда будет существовать оптимальное расписание без простоев прибора. □

Многие известные нам целевые функции являются монотонно возрастающими (неубывающими) функциями, зависящими от моментов окончания обслуживания требований C_j. Например, $\sum C_j, \sum U_j, \sum T_j$ и т.д.

Приведенный факт можно использовать следующим образом. Если некоторая задача соответствует условиям теоремы 3.1, тогда оптимальное расписание для этой задачи однозначно задается перестановкой элементов множества N.

Определение 5 *Перестановка из n элементов – это конечная последовательность длины n, все элементы которой различны.*

Для данных задач перестановка вида $\pi = (j_1, j_2, \ldots, j_n)$, задающая расписание π, определяет порядок обслуживания требований (см. векторное представление расписаний). То есть первым обслуживается требование j_1, за ним – требование j_2 и т.д. Тогда моменты завершения обслуживания для каждого из требований $C_{j_k}(\pi) = \sum_{l=1}^{k} p_{j_l}$. Величину $C_{j_k}(\pi)$ называют временем (моментом) завершения обслуживания требований j_k при расписании π.

Для задач, в которых расписание можно задать перестановкой, важными являются следующие определения:

Определение 6 *EDD (Earliest Due Date) порядок обслуживания требований – очередность обслуживания, при которой требования обслуживаются в порядке неубывания директивных сроков d_j.*

Определение 7 *LDD (Latest Due Date) порядок обслуживания требований – очередность обслуживания, при которой требования обслуживаются в порядке невозрастания директивных сроков d_j.*

Определение 8 *SPT (Shortest Processing Time) порядок обслуживания требований – очередность обслуживания, при которой требования обслуживаются в порядке неубывания времен обслуживания* p_j.

Определение 9 *LPT (Longest Processing Time) порядок обслуживания требований – очередность обслуживания, при которой требования обслуживаются в порядке невозрастания времен обслуживания* p_j.

Определение 10 *Частичное расписание* π' *– фрагмент целого расписания* π*, описывающее порядок обслуживания подмножества требований* $N' \subset N$.

В данной главе запись вида $\{\pi\}$ обозначает множество требований, обслуживаемых при расписании π. Запись вида $i \in \pi$ означает $i \in \{\pi\}$. Через $\pi \setminus \{i\}$, где $\pi = (\pi_1, i, \pi_2)$, будем обозначать частичное расписание вида (π_1, π_2). Запись $j \to i$ означает, что обслуживание требования j предшествует обслуживанию требования i. Соответственно, запись $(j \to i)_\pi$ означает, что это выполняется при расписании π.

3.1 Одноприборные задачи $1|r_j, p_j = 1, pmtn| \sum f_j$

В этом параграфе предлагается алгоритм решения одноприборных задач, для которых $p_j = 1$, $r_j \in Z^+$, $j = 1, 2, \ldots, n$, и нет отношений предшествования между требованиями. Обозначим эти задачи как $1|r_j, p_j = 1, pmtn| \sum f_j$. При этом функция f_j требования j зависит от времени окончания обслуживания C_j. Данную задачу можно решить, сведя ее к ЗАДАЧЕ О НАЗНАЧЕНИЯХ.

Количество заданий в соответствующей ЗАДАЧЕ О НАЗНАЧЕНИЯХ будет равно n, а количество исполнителей равно $r_{\max} + n$, где $r_{j\max} = \max\limits_{j=\overline{1,n}} r_j$. То есть для каждого требования $j, j = 1, 2, \ldots, n$, нужно определить интервал обслуживания $[t, t+1) \in [0, r_{\max} + n)$. Матрица стоимостей формируется следующим образом. Для требования j для каждого $t \in [0, r_{\max}+n)$, $t \geq r_j$, вычислим $a_{tj} = f_j(C_j = t+1)$. Для каждой точки $t < r_j$ и для каждой точки $t \geq D_j = r_{\max} + n$ примем $a_{tj} = +\infty$.

Решив ЗАДАЧУ О НАЗНАЧЕНИЯХ, получим интервалы обслуживания каждого из требований. То есть задачу $1|r_j, p_j = 1| \sum f_j$ можно решить за время $O(n^3)$.

Рассмотрим следующую практическую задачу. У фермера есть один комбайн. Ему необходимо убрать урожай с n' полей. На уборку поля j, $j = 1, 2, \ldots, n'$, необходим 1 день. На полях в разное время засеяна разная культура, поэтому для каждого поля определен "идеальный день" d_j, к которому урожай на поле нужно убрать. Нарушение этого срока ведет к тому, что урожай портится. То есть возникают потери качества, пропорционально зависящие от количество дней запаздывания с уборкой и от ценности культуры. Ценность культуры задается числом w_j. Обозначим через C_j время, к которому поле j убрано. Если $C_j - d_j = T_j > 0$, то потери качества для этого поля могут задаваться формулой $w_j T_j$. Необходимо минимизировать суммарные потери качества по всем культурам. Такую задачу можно решить следующим образом. Очевидно, что период сбора урожая, не превышает 365 дней, т.е. $\sum p_j \leq 365$. Мы конструируем пример задачи $1|r_j, p_j = 1| \sum f_j$ с множеством из n требований. Для каждого возможного дня t, для каждого требования j, $j = 1, 2, \ldots, n$, рассчи-

тываем $a_{tj} = w_j \max\{0, t+1-d_j\}$. *Трудоемкость решения этой задачи* $O(n^3)$ *будет приемлемым для любого современного компьютера, т.к.* $n \leq 365$.

3.2 Минимизация числа запаздывающих требований $1||\sum U_j$

Имеется множество требований $N = \{1, 2, \ldots, n\}$. Для каждого требования $j \in N$ заданы продолжительность обслуживания $p_j > 0$ и директивный срок d_j, к которому требование должно быть обслужено. Прерывания при обслуживании и обслуживание более одного требования в любой момент времени запрещены. Расписание однозначно задается перестановкой $\pi = (j_1, j_2, \ldots, j_n)$. То есть время завершения обслуживания требования j_k при расписании π определяется следующим образом: $C_{j_k}(\pi) = \sum_{l=1}^{k} p_{j_l}$. Если $C_j(\pi) > d_j$, тогда требование j запаздывает, и в этом случае полагают $U_j = 1$. Если $C_j(\pi) \leq d_j$, тогда требование j не запаздывает, и $U_j = 0$. Необходимо построить расписание π, при котором значение целевой функции $F(\pi) = \sum\limits_{j=1}^{n} U_j(\pi)$ минимально.

Далее представлен полиномиальный алгоритм Мура (Moore) [87] решения данной задачи, который основан на следующей лемме.

Лемма 3.1 *Для каждого примера задачи* $1||\sum U_j$ *существует оптимальное расписание вида* $\pi = (G, H)$, *при котором все требования* $i \in G$ *не запаздывают, а все требования* $j \in H$ *запаздывают.*

Доказательство. Предположим, что существует оптимальное расписание вида $\pi = (j_1 \ldots, j_{k-1}, j_k, \ldots, j_n)$, при котором требование j_{k-1} запаздывает, а требование j_k не запаздывает. Тогда $\pi' = (j_1, \ldots, j_k, j_{k-1}, \ldots, j_n)$ также является оптимальным расписанием, т.к. количество запаздывающих требований не увеличилось. После конечного числа попарных перестановок требований, каждая из которых не увеличивает значение целевой функции, получим оптимальное расписание вида $\pi = (G, H)$. $\square$

Докажем еще одно свойство, важное для задач, связанных с запаздыванием.

Лемма 3.2 *Пусть для примера задачи* $1||\sum U_j$ *существует оптимальное расписание, при котором все требования не запаздывают. Тогда расписание* π_{EDD} *(требования обслуживаются в порядке EDD) также является оптимальным для данного примера.*

Доказательство. Предположим, что существует оптимальное расписание вида $\pi = (j_1 \ldots, j_{k-1}, j_k, \ldots, j_n)$, при котором требование $d_{j_{k-1}} > d_{j_k}$. Тогда расписание $\pi' = (j_1, \ldots, j_k, j_{k-1}, \ldots, j_n)$ также является оптимальным, т.к. $F(\pi') \leq F(\pi)$. Произведя данную попарную перестановку необходимое число раз, мы получим оптимальное расписание π_{EDD}. $\square$

Очевидно, что данное свойство выполняется и для многих других задач таких, как $1||\sum w_j U_j$, $1||\sum w_j T_j$ и т.д. Теперь мы можем утверждать, что для задачи $1||\sum U_j$ существует оптимальное расписание, при котором все незапаздывающие требования обслуживаются в EDD порядке в начале расписания, а запаздывающие требования – в произвольном порядке в конце расписания.

Идея алгоритма решения данной задачи описывается следующим образом. Мы последовательно формируем множество незапаздывающих требований $\{G\}$. Требования включаются в множество $\{G\}$ в порядке неубывания директивных сроков и добавляются в конец частичного расписания G. Если после добавления очередного требования j окажется, что оно запаздывает, то мы находим требование $i \in G$, с максимальной продолжительностью обслуживания и "перемещаем" его в множество запаздывающих требований $\{H\}$. Формально алгоритм 5 решения задачи $1||\sum U_j$ описан ниже.

Пример. Рассмотрим пример с 5-ю требованиями. Примеры требований: $p_1 = 2$, $p_2 = 5$, $p_3 = 3$, $p_4 = 2$, $p_5 = 4$, $d_1 = 3$, $d_2 = d_3 = 9$, $d_4 = d_5 = 10$. При исполнении алгоритма 5 включаем в множество G требования 1 и 2. При этом $t = 2 + 5 = 7 < 9 = d_2$. Пробуем включить требование 3. Получаем $t = 2 + 5 + 3 = 10 > 9 = d_3$, тогда находим в G требование с максимальной продолжительностью $i = 2$. Получаем $G = \{1, 3\}$, $t = 2 + 3 = 5$. Включаем требование 4, имеем $t = 2 + 3 + 2 = 7 < 10 = d_4$. Добавляем в множество G требование 5. Имеем $t = 2+3+2+4 = 11 > 10 = d_5$, поэтому находим в G требование с максимальной продолжительностью $i = 5$ и исключаем его из G. Имеем

Algorithm 5 Алгоритм решения задачи $1||\sum U_j$.

```
1: Перенумеруем требования согласно правилу d_1 ≤ d_2 ≤ ··· ≤ d_n;
2: G := ∅; H := ∅; t := 0;
3: for j := 1 to n do
4:     G := G ⋃ {j};
5:     t := t + p_j;
6:     if t ≥ d_j then
7:         Найдем требование i ∈ G с максимальной продолжительность обслуживания, т.е. i := argmax_{k∈G} p_k;
8:         G := G \ {i};
9:         H := H ⋃ {i};
10:        t := t − p_i;
11:    end if
12: end for
```

в итоге $G = \{1, 3, 4\}$ и оптимальное расписание $\pi = (1, 3, 4, 2, 5)$.

Напоминаем, что для задачи $1\| \sum U_j$ рассматриваются только расписания вида $\pi = (G, H)$.

Лемма 3.3 *Пусть при расписании $\pi = (j_1, j_2, \ldots, j_l, j_{l+1}, \ldots, j_n)$ выполняется $d_{j_1} \le d_{j_2} \le \cdots \le d_{j_n}$. Требование j_l – последнее незапаздывающее требование. Тогда существует оптимальное расписание, при котором требование $j^* \in \{j_1, j_2, \ldots, j_{l+1}\}$, $p_{j^*} \ge p_i$, для всех $i \in \{j_1, j_2, \ldots, j_{l+1}\}$, запаздывает.*

Доказательство. Предположим, что существует оптимальное расписание $\pi = (\pi_1, j^*, \pi_2, \pi_3, i, \pi_4)$, при котором требования множества $\{\pi_1, j^*, \pi_2, \pi_3\}$ не запаздывают, а требования $\{i, \pi_4\}$ запаздывают. Согласно лемме 3.2 для рассматриваемого примера не существует расписания, при котором все требования $j_1, j_2, \ldots, j_l, j_{l+1}$ не запаздывают. Тогда существует запаздывающее требование $i \in \{j_1, j_2, \ldots, j_l, j_{l+1}\}$, $i \ne j^*$. В соответствии с леммой 3.2 все требования $\{\pi_1, j^*, \pi_2, \pi_3\}$ обслуживаются в порядке EDD.

Если $d_{j^*} \le d_i$, то расписание $\pi' = (\pi_1, i, \pi_2, j^*, \pi_3, \pi_4)$ также оптимальное, т.к. $F(\pi') \le F(\pi)$. При расписании π' требование j^* запаздывает, а требование i не запаздывает.

Рассмотрим случай $d_{j^*} > d_i$. Пусть для всех $j \in \{\pi_3\}$, $d_j \ge d_i$, и для всех $j \in \{\pi_2\}$ выполняется $d_j < d_i$.

Рассмотрим расписание $\pi'' = (\pi_1, \pi_2, i, \pi_3, j^*, \pi_4)$. Для всех $j \in \{\pi_2\} \bigcup \{\pi_3\}$ имеем $C_j(\pi'') \geq C_j(\pi)$, так как $p_{j^*} \geq p_i$. Пусть k – последнее требование из частничного расписания π_2. Тогда $C_i(\pi'') \leq C_k(\pi') \leq d_k < d_i$, т.е. при расписании π'' все требования из множества $\{\pi_2\} \bigcup \{i\} \bigcup \{\pi_3\}$ не запаздывают. Поэтому расписание π'' оптимальное.

Лемма доказана. □

Теорема 3.2 *При помощи алгоритма 5 для задачи* $1||\sum U_j$ *за* $O(n \log n)$ *операций строится оптимальное расписание.*

Доказательство. Докажем теорему методом математической индукции. Алгоритм верен при $n = 1$.

Предположим, что алгоритм верен для всех примеров размерности $n-1$ требований. Рассмотрим пример размерности n требований. Предположим, что при помощи алгоритма 5 было построено расписание $\pi = (G, H)$, но существует оптимальное расписание $\pi' = (G', H')$ такое, что $|G'| \geq |G|$.

При использовании алгоритма 5 для $n-1$ требований $1, 2, \ldots, j-1, j+1, \ldots, n$, где требование $j = j^*$ выбрано согласно лемме 3.3, будет получено оптимальное расписание $(G \setminus \{j^*\}, H)$. Расписание $(G' \setminus \{j^*\}, H')$ является допустимым для соответствующего примера из $n-1$ требований. Поэтому имеем $|G'| \leq |G|$, тогда $|G'| = |G|$. Значит алгоритм строит оптимальное решение.

Трудоёмкость сортировки требований в порядке EDD на первом шаге алгоритма составляет $O(n \log n)$ операций. Трудоёмкость цикла 3–12 равна $O(n)$ операций. То есть трудоемкость алгоритма равна $O(n \log n)$ операций. □

3.3 Минимизация взвешенного числа запаздывающих требований $1||\sum w_j U_j$

Очевидным является тот факт, что минимизация взвешенного числа запаздывающих требований эквивалентно максимизации взвешенного числа **не**запаздывающих требований. То есть критерий оптимизации

$F(\pi) = \sum w_j U_j(\pi) \to \min$ может быть заменен на критерий $F(\pi) = \sum w_j[1 - U_j(\pi)] \to \max$.

Теорема 3.3 *Задача* $1||\sum w_j U_j$ *является* **NP-трудной**.

Доказательство. Доказательство проведем сведением ЗАДАЧИ О РАНЦЕ к частному случаю задачи $1||\sum w_j U_j$ с общим директивным сроком $d_j = d$. Множеству предметов $N = \{1, 2, \ldots, n\}$ ЗАДАЧИ О РАНЦЕ ставим в соответствие множество требований $N' = \{1, 2, \ldots, n\}$ задачи $1||\sum w_j U_j$. Пусть вес требования $w_j = p_j$ (стоимость предмета в ЗАДАЧЕ О РАНЦЕ), продолжительность обслуживания требования $p_j = w_j$ (вес предмета в ЗАДАЧЕ О РАНЦЕ) для каждого j, $j = 1, 2, \ldots, n$. Зададим общий директивный срок $d_j = C$, $j = 1, 2, \ldots, n$. Тогда при оптимальном расписании π^* значение $\sum w_j(1 - U_j(\pi^*))$ равно максимальной суммарной стоимости предметов, помещенных в рюкзак, то есть равно значению целевой функции для ЗАДАЧИ О РАНЦЕ. Таким образом сводимость доказана.

Очевидно, что сведение полиномиально, т.е. выполняется за полиномиальное число шагов. □

Теорема 3.4 *Для задачи* $1||\sum w_j U_j$ *существует оптимальное расписание вида* $\pi = (G, H)$, *где все требования* $j \in H$ *запаздывают, а все требования* $i \in G$ *не запаздывают. Требования из множества* G *обслуживаются в порядке EDD, а требования из множества* H *– в порядке LDD.*

Доказательство этой теоремы аналогично доказательству лемм 3.1 и 3.2. Заметим, что в оптимальном расписании незапаздывающие требования обслуживаются в порядке EDD в то время, как запаздывающие требования могут обслуживаться в любом порядке (в том числе в порядке LDD). Алгоритм решения 6 основан на теореме 3.4.

В данном алгоритме $\pi_j(t)$ означает оптимальное частичное расписание для требований $1, 2, \ldots, j$, при котором обслуживание требований начинается в момент времени t, а $f_j(t) = \sum_{i=1}^{j} w_i[1 - U_i(\pi_j(t))]$ соответствует

Algorithm 6 Алгоритм решения задачи $1||\sum w_j U_j$.

Перенумеруем требования согласно правилу $d_1 \le d_2 \le \cdots \le d_n$;
for $t := 0$ to $\sum_{i=2}^{n} p_i$ **do**
    $\pi_1(t) := (1)$;
    if $t + p_1 - d_1 \le 0$ **then**
        $f_1(t) := w_1$;
    else
        $f_1(t) := 0$;
    end if
end for
for $j := 2$ to n **do**
    for $t := 0$ to $\sum_{i=j+1}^{n} p_i$ **do**
        $\pi^1 := (j, \pi_{j-1}(t + p_j))$, $\pi^2 := (\pi_{j-1}(t), j)$;
        if $t + p_j - d_j \le 0$ **then**
            $\Phi^1(t) := w_j + f_{j-1}(t + p_j)$;
        else
            $\Phi^1(t) := f_{j-1}(t + p_j)$;
        end if
        if $t + \sum_{i=1}^{j} p_i - d_j \le 0$ **then**
            $\Phi^2(t) := f_{j-1}(t) + w_j$;
        else
            $\Phi^2(t) := f_{j-1}(t)$;
        end if
        if $\Phi^1(t) < \Phi^2(t)$ **then**
            $f_j(t) := \Phi^2(t)$;
            $\pi_j(t) := \pi^2$;
        else
            $f_j(t) := \Phi^1(t)$;
            $\pi_j(t) := \pi^1$;
        end if
    end for
end for

$\pi_n(0)$ – оптимальное расписание, а $f_n(0)$ – соответствующее оптимальное значение целевой функции – максимальное взвешенное число незапаздывающих требований.

максимальному взвешенному числу незапаздывающих требований при этом частичном расписании.

В Алгоритме 6 происходит последовательная "окантовка" ранее построенного расписания. В расписании $\pi_l(t)$ для любого l, $l = 1, 2, \ldots, n$, требования множества $\{1, 2, \ldots, l\}$ расположены "компактно", т.е. между ними нет требований с большими номерами.

Теорема 3.5 *Алгоритм 6 строит оптимальное расписание задачи максимизации взвешенного числа незапаздывающих требований за* $O(n \sum p_j)$ *операций.*

Доказательство. Доказательство от противного. Допустим, что существует оптимальное расписание $\pi^* = (G, H)$, где все требования $j \in H$ запаздывают, а все требования $i \in G$ не запаздывают. Требования из множества G обслуживаются в порядке EDD, а требования из множества H – в порядке LDD, для которого $f(\pi^*) < f(\pi_n(0)) = f_n(0)$.

Пусть $\pi' := \pi^*$. Для каждого $l = 1, 2, \ldots, n$ мы последовательно рассматриваем часть $\bar{\pi}_l \in \pi'$, $\{\bar{\pi}_l\} = \{1, \ldots, l\}$, расписания. Пусть $\pi' = (\pi_\alpha, \bar{\pi}_l, \pi_\beta)$.??? Если $\bar{\pi}_l \neq \pi_l(t)$, где $t = \sum\limits_{i \in \{\pi_\alpha\}} p_i$ (для определения см. строки 23–28 алгоритма), тогда примем $\pi' := (\pi_\alpha, \pi_l(t), \pi_\beta)$. Очевидно, что $f((\pi_\alpha, \bar{\pi}_l, \pi_\beta)) \leq f((\pi_\alpha, \pi_l(t), \pi_\beta))$. Аналогичную операцию проделываем для каждого $l = 1, 2, \ldots, n$. В конце получаем $f(\pi^*) \leq f(\pi') \leq f_n(0)$. Поэтому расписание $\pi_n(0)$ так же оптимально.

Очевидно, что трудоёмкость алгоритма составляет $O(n \sum p_j)$ операций, так как в цикле 10–31 выполняется $n - 1$ итераций и просматриваются все целочисленные точки, ограниченного интервалом $\left[0, \sum\limits_{j=1}^{n} p_j\right]$. $\square$

Фактически в представленном алгоритме 6 динамического программирования используются следующие функциональные уравнения Беллмана:

$$f_j(t) = \max \begin{cases} \Phi^1(t) = \alpha(t) + f_{j-1}(t + p_j), & j = 1, 2, \ldots, n; \\ \Phi^2(t) = \beta(t) + f_{j-1}(t), & j = 1, 2, \ldots, n, \end{cases} \tag{3.1}$$

где

$$f_0(t) = 0 \text{ для } t \geq 0.$$

Функция $\Phi^1(t)$ соответствует значению переменной $x_j = 1$ (т.е. требование j добавляется в начало частичного расписания и начинает обслуживаться в момент времени t), а функция $\Phi^2(t)$ соответствует значению $x_j = 0$ (т.е. требование добавляется в конец, после обслуживания уже упорядоченных требований). Если $t + p_j - d_j \leq 0$, тогда $\alpha(t) = w_j$, иначе $\alpha(t) = 0$. Если $t + \sum_{i=1}^{j} p_i - d_j \leq 0$, тогда $\beta(t) = w_j$, иначе $\beta(t) = 0$.

Алгоритм 6 может быть модифицирован таким образом, что для каждого $j = 1, 2, \ldots, n$ в основном цикле (строки 10–31) будут рассматриваться только значения $t \in [0, d_j - p_j]$, вместо значений $t \in [0, \sum_{i=j+1}^{n} p_i]$, т.к. для любого $t > d_j - p_j$, требование j запаздывает в любом частичном расписании $\pi_j(t)$, и частичное расписание $\pi^2 := (\pi_{j-1}(t), j)$ является оптимальным. Поэтому трудоёмкость модифицированного алгоритма 6 равна $O(nd_{\max})$, где $d_{\max} = \max_{j \in N} d_j$. Без ограничения общности считаем $d_{\max} < \sum_{j=1}^{n} p_j$, т.к. в противном случае требование с максимальным директивным сроком при любом расписании незапаздывающее и может быть исключено из рассмотрения.

3.3.1 Графический алгоритм для задачи $1||\sum w_j U_j$

Напомним, что мы рассматриваем задачу максимизации взвешенного числа незапаздывающих требований.

Идея графического алгоритма (GrA) для данной задачи можно описать следующим образом. На каждом шаге j алгоритма GrA мы сохраняем функцию $f_j(t)$ в табличном виде, представленном в таб. 3.1, где $t_1 < t_2 < \ldots < t_{m_j}$ и $W_1 > W_2 > \ldots > W_{m_j}$.

Записи в таблице означают следующее. Для каждого значения $t \in (t_l, t_{l+1}]$, $1 \leq l < m_j$, оптимальным будет расписание $\pi_l = (G_l, H_l)$, где все требования $j \in H_l$ запаздывают, а все требования $i \in G_l$ не запаздывают. Требования из множества G_l обслуживаются в порядке EDD, а требова-

t	t_1	t_2	$\ldots$	t_{m_j}
$f_j(t)$	W_1	W_2	$\ldots$	W_{m_j}
оптим. частичное расписание	π_1	π_2	$\ldots$	π_{m_j}

Таблица 3.1: Функция $f_j(t)$

t	$\ldots$	t_l	t_{l+1}	$\ldots$
$f_{j+1}(t)$	$\ldots$	W_l	$W_{l+1} = W_l$	$\ldots$
оптим. частичное расписание	$\ldots$	π_l	π_{l+1}	$\ldots$

Таблица 3.2: Сокращение числа интервалов для функции $f_{j+1}(t)$

ния из множества H_l – в порядке LDD, которому соответствует значение $f_j(t) = W_l = \sum\limits_{i \in G_l} w_i$ (суммарное взвешенное число незапаздывающих требований). Точки t_l называются *точками излома*, т.е. выполняется $f_j(t') > f_j(t'')$ для $t' \leq t_l < t''$.

На очередном шаге $j+1$, мы трансформируем функцию $f_j(t)$ в две функции $\Phi^1(t)$ и $\Phi^2(t)$ согласно шагам 12–28 алгоритма 6 за $O(m_j)$ операций. Функция $\Phi^1(t)$ получается смещением графика функции $f_j(t)$ влево на p_{j+1} и добавлением новой точки $t = p_{j+1} - d_{j+1}$, а функция $\Phi^2(t)$ – добавлением новой точки $t = \sum\limits_{i=1}^{j+1} p_i - d_{j+1}$. Все эти операции производятся при помощи таблиц вида 3.1. В каждой таблице $\Phi^1(t)$ и $\Phi^2(t)$ будет не более $m_j + 1$ точек излома. Далее мы вычисляем таблицу для функции

$$f_{j+1}(t) = \max\{\Phi^1(t), \Phi^2(t)\}$$

за $O(m_j)$ операций. В новой таблице, соответствующей функции $f_{j+1}(t)$ будет не более $2m_j + 2$ точек излома (обычно это число существенно меньше). Фактически, мы рассматриваем не все точки t из интервала $[0, \min\{d_j - p_j, \sum\limits_{i=j+1}^{n} p_i\}]$, но только точки интервала, в которых меняется суммарное взвешенное число незапаздывающих требований.

Если встречается ситуация, описанная в таб. 3.2, то мы удаляем столбец, соответствующий точке t_{l+1} и принимаем $\pi_l := \pi_{l+1}$.

Очевидно, что на каждом шаге GrA будет рассмотрено не более F_{opt} точек излома, где $F_{opt} \leq \sum\limits_{j=1}^{n} w_j$ – максимальное взвешенное число незапаз-

дывающих требований. На каждом шаге $j = 1, 2, \ldots, n$ алгоритма GrA, необходимо рассмотреть не более $\min\{2^j,\ d_j - p_j,\ \sum_{i=j+1}^{n} p_i, \sum_{i=1}^{j} w_i, F_{opt}\}$ точек излома, т.е. трудоёмкость GrA не превосходит $O(\min\{2^n, n \cdot \min\{d_{\max}, F_{opt}\}\})$ операций.

3.4 Минимизация суммарного запаздывания $1||\sum T_j$

В данном параграфе рассматривается задача минимизации суммарного запаздывания $\sum T_j$, где $T_j = \max\{0, C_j - d_j\}$. То есть в данной задаче необходимо найти расписание π^*, при котором функция $F(\pi) = \sum_{j=1}^{n} T_j(\pi)$ достигает своего минимума. Несмотря на то, что это классическая задача **TP** , доказательство ее **NP-трудности** было получено сравнительно недавно – в 1990-м году, т.е. выяснение её трудоёмкости было нетривиальной проблемой. Далее для этой задачи будут представлены точный и метаэвристический алгоритмы решения, а также приближенный алгоритм с регулируемой точностью.

Доказательство NP-трудности данной задачи приводится в приложении 2.

3.4.1 Точный алгоритм решения задачи $1||\sum T_j$

Алгоритм основан на следующих результатах, опубликованных Е. Лаулерм в 1977-м году. Необходимо отметить, что данный алгоритм строит точное решение и для частного случая задачи $1||\sum w_j T_j$, при котором выполняется правило "если $p_j > p_i$, то $w_j \leq w_i,\ i, j \in N$".

Теорема 3.6 *Пусть π – оптимальное расписание для примера задачи с директивными сроками $d_1, d_2, \ldots, d_n$, и пусть C_j – моменты завершения обслуживания требований $j = 1, 2, \ldots, n$ при этом расписании. Выберем новые директивные сроки d'_j так, что:*

$$\min\{d_j, C_j\} \leq d'_j \leq \max\{d_j, C_j\}.$$

Тогда любое оптимальное расписание π', соответствующее примеру с новыми директивными сроками $d'_1, d'_2, \ldots, d'_n$ является оптимальным и для примера с исходными директивными сроками $d_1, d_2, \ldots, d_n$.

Доказательство. Обозначим через T, суммарное запаздывания для примера с директивными сроками $d_1, d_2, \ldots, d_n$, а через T' – суммарное запаздывания для примера с директивными сроками $d'_1, d'_2, \ldots, d'_n$. Пусть π' – оптимальное расписание для примера с директивными сроками $d'_1, d'_2, \ldots, d'_n$, а C'_j – момент завершения обслуживания требования j, $j = 1, 2, \ldots, n$, при данном расписании. Запишем значения $T(\pi)$ и $T(\pi')$ в следующей форме:

$$T(\pi) = T'(\pi) + \sum_{j=1}^{n} A_j,$$

$$T(\pi') = T'(\pi') + \sum_{j=1}^{n} B_j,$$

где A_j и B_j – значения, зависящие от требования $j \in N$, для которых выполняется следующее::

(a) пусть $C_j \leq d_j$, тогда:

1. если $C'_j \leq d'_j$, тогда $A_j = 0$ и $B_j = 0$;
2. если $d'_j < C'_j \leq d_j$, тогда $A_j = 0$ и $B_j < 0$. $B_j = d_j - C'_j < 0$;
3. если $d_j < C'_j$, тогда $T_j(\pi') \leq T'_j(\pi')$. Поэтому $A_j = 0$ и $B_j < 0$. $B_j = d'_j - d_j < 0$.

(b) пусть $C_j > d_j$, тогда:

1. если $C'_j \leq d_j$, тогда $A_j = d'_j - d_j \geq 0$ и $B_j = 0$;
2. если $d_j < C'_j \leq d'_j$, тогда $A_j = d'_j - d_j \geq 0$ и $B_j = (C'_j - d_j) - 0 < d'_j - d_j = A_j$;
3. если $d'_j < C'_j$, тогда $A_j = B_j = d'_j - d_j$.

Выполняется $A_j \geq B_j$ для всех $j = 1, 2, \ldots, n$. Более того, $T'(\pi) \geq T'(\pi')$, т.к. при расписании π' находится минимальное значение T'. Поэтому получаем

$$T'(\pi) + \sum_{j=1}^{n} A_j \geq T'(\pi') + \sum_{j=1}^{n} B_j,$$

т.е. $T(\pi) \geq T(\pi')$. Теорема доказана. □

Лемма 3.4 *Для задачи* $1||\sum T_j$ *существует оптимальное расписание* π*, при котором:*

- *обслуживание требования* i *предшествует обслуживанию требования* j*, если* $d_i \leq d_j$ *и* $p_i < p_j$*;*
- *все незапаздывающие требования обслуживаются в порядке неубывания директивных сроков.*

Доказательство.

Сначала докажем первое свойство оптимальных расписаний. Пусть для двух требований i и j выполняется $d_i \leq d_j$ и $p_i < p_j$. Предположим, что существует оптимальное расписание $\pi = (\pi_1, j, \pi_2, i, \pi_3)$. Для расписания $\pi' = (\pi_1, i, \pi_2, j, \pi_3)$ выполняется $F(\pi) - F(\pi') \geq (T_j(\pi) - T_j(\pi')) + (T_i(\pi) - T_i(\pi'))$.

Если $C_i(\pi') > d_j$, тогда

$$F(\pi) - F(\pi') \geq -(p_j + \sum_{k \in \pi_2} p_k) + (p_i + \sum_{k \in \pi_2} p_k) > 0.$$

Иначе, если $C_i(\pi') \leq d_j$, выполняется

$$F(\pi) - F(\pi') \geq -\max\{0, C_j(\pi') - d_j\} + \max\{0, C_j(\pi') - d_i\} \geq 0,$$

где $C_j(\pi') = C_i(\pi)$. Таким образом, первое свойство леммы доказано.

Докажем второе свойство. Пусть существует оптимальное расписание $\pi = (\pi_1, j, \pi_2, i, \pi_3)$, при котором $d_j > d_i$ и оба требования i и j не запаздывают. Тогда расписание $\pi' = (\pi_1, \pi_2, i, j, \pi_3)$ также является оптимальным. Повторяя подобную операцию необходимое число раз получим оптимальное расписание, удовлетворяющее второму свойству леммы. □

Теорема 3.7 *Пусть требования упорядочены согласно правилу* $d_1 \leq d_2 \leq \cdots \leq d_n$. *Обозначим через* j^* *– требование с максимальной продолжительностью обслуживания, если таких требований несколько, то из них с максимальным директивным сроком, т.е.* $j^* = \arg\max_{j \in N}\{d_j : p_j = \max_{i \in N} p_i\}$. *Тогда существует требование* $k \geq j^*$ *такое, что при оптимальном расписании все требования* $i, i = 1, 2, \ldots, k,\ i \neq j^*$, *обслуживаются перед требованием* j^*, *а остальные требования после требования* j^*.

Доказательство. Пусть C' – наибольший момент окончания обслуживания требования j^* при всех оптимальных расписаниях, соответствующих примеру с директивными сроками $d_1 \leq d_2 \leq \cdots \leq d_n$. Обозначим через π оптимальное расписание, соответствующее примеру с модифицированными директивными сроками:

$$d_1, d_2, \ldots d_{j^*-1}, d'_{j^*} = \max\{d_{j^*}, C'\}, d_{j^*-1}, \ldots, d_n,$$

и которое соответствует условиям леммы 3.4. Расписание π является оптимальным и для исходного примера согласно теореме 3.6. Пусть C – время завершения обслуживания требования j^* при расписании π. Тогда имеем $C \leq C' \leq d'_{j^*} = \max\{d_{j^*}, C'\}$, т.е. требование j^* не запаздывает в модифицированном примере при расписании π. При расписании π все требования $i,\ d_i > d'_{j^*}$ обслуживаются перед требованием j^*, т.к. иначе требование i не будет запаздывающим, что противоречит второму свойству леммы 3.4. С другой стороны, все требования j такие, что $d_j \leq d'_{j^*}$, должны обслуживаться при этом расписании перед требованием j^*, т.к. $p_j < p_{j^*}$ (см. первое свойство леммы 3.4). Теорема доказана. □

Можно сделать следующий вывод из данной теоремы. Если выбрать такое требование j^* и для него определить требование (позицию) $k \geq j^*$, то исходную задачу можно разбить на две подзадачи решаемые аналогично. Первая подзадача содержит множество требований $i, i = 1, 2, \ldots, k,\ i \neq j^*$, а другая – множество требований $\{k+1, k+2, \ldots, n\}$. Сложность заключается в выборе требования (позиции) k.

На этом факте основан следующий точный алгоритм 7 решения задачи.

Обозначим через $j^*(N')$ требование с наибольшей продолжительностью обслуживания среди требований множества $N' \subseteq N$. Если таких требований несколько, то выбирается требование с наибольшим директивным сроком, т.е. $j^*(N') = \arg\max\limits_{j \in N'}\{d_j : p_j = \max\limits_{i \in N'} p_i\}$. Для сокращения записи вместо $j^*(N')$ будем записывать j^*, если очевидно о каком множестве идет речь.

Рассмотрим пример (подпример) обслуживания требований множества $N' \subseteq N$, $N' = \{1, 2, \ldots, n'\}$, с момента времени $t' \geq 0$. Множество $L(N', t')$ есть множество всех индексов $i \in \{j^*, j^* + 1, \ldots, n'\}$.

Algorithm 7 Алгоритм решения задачи $1||\sum T_j$.

Procedure ProcL (N, t)

1: Дан пример $\{N, t\}$ с множеством требований $N = \{j_1, j_2, \ldots, j_n\}$ и моментом начала обслуживания t, где $d_{j_1} \leq d_{j_2} \leq \cdots \leq d_{j_n}$.
2: **if** $N = \emptyset$ **then**
3: π^*:= пустое расписание;
4: **else**
5: Найдем требование j^* из множества N;
6: Найдем множество $L(N, t)$ для требования j^*;
7: **for** ALL $k \in L(N, t)$ **do**
8: $\pi_k := (\mathbf{ProcL}(N', t'), j^*(N), \mathbf{ProcL}(N'', t''))$, где
$N' := \{j_1, \ldots, j_k\} \setminus \{j^*\}$, $t' := t$,
$N'' := \{j_{k+1}, \ldots, j_n\}$, $t'' := t + \sum_{i:=1}^{k} p_{j_i}$;
9: **end for**
10: π^*:=$\arg\min\limits_{k \in L(N,t)}\{F(\pi_k, t)\}$;
11: **end if**
12: **RETURN** π^*;

Алгоритм решения.

1: π^*:=$\mathbf{ProcL}(N, 0)$;

Запись $F(\pi_k, t)$ в алгоритме означает суммарное запаздывание при расписании π_k выполнение которого начинается с момента времени t. На следующих двух рисунках 3.1 показан процесс выбора позиции k и разбиение исходной задачи на две подзадачи.

Список рассматриваемых позиций $L(N', t')$ можно значительно сократить, т.к. в это множество стоит включать только требования $k \in \{j^*, j^* + 1, \ldots, n'\}$, для которых выполняются условия:

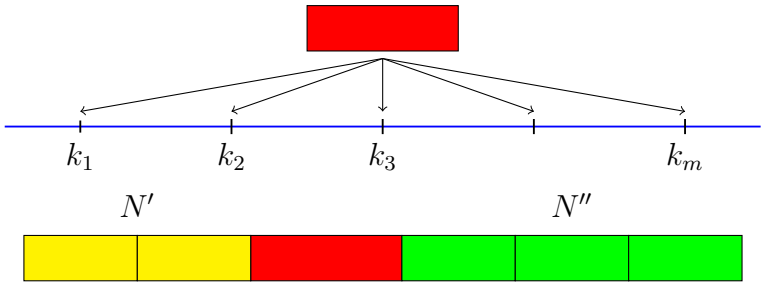

Рис. 3.1: Рисунки к алгоритму

(а) $t' + \sum_{j=1}^{k} p_j < d_{k+1}$ (**правило исключения 1**) и

(б) $d_j + p_j \leq t' + \sum_{j=1}^{k} p_j$, для всех $j = \overline{j^*(N') + 1, k}$ (**правила исключения 2, 3**),

где $d_{n'+1} := +\infty$.

Существуют и другие правила исключения, позволяющие сократить время работы алгоритма.

Теорема 3.8 *[78] Трудоёмкость алгоритма 7 составляет* $O(n^4 \sum p_j)$ *операций.*

Модификации данного алгоритма с различными правилами сокращения перебора периодически появляются в литературе. Но на текущий момент, лучший из предложенных алгоритмов может решать "простые" примеры, построенные по схеме Поттса и ван Вассенхова [89] (описание этой схемы приводится в параграфе 3.4.5.), размерностью не больше 500 требований.

3.4.2 Аппроксимационный алгоритм

В данном параграфе для задачи $1||\sum T_j$ представлена ПОЛНАЯ ПОЛИНОМИАЛЬНАЯ АППРОКСИМАЦИОННАЯ СХЕМА (fully polynomial time approximation schema, FPTAS). Фактически, это приближенный алгоритм решения с настраиваемой погрешностью. То есть для любого числа $\varepsilon > 0$ с помощью алгоритма можно найти расписание π, такое, что

$F(\pi) - F(\pi^*) \leq \varepsilon F(\pi^*)$, где π^* – оптимальное расписание. Трудоёмкость такого алгоритма полиномиально зависит от n и от значения $1/\varepsilon$. Поэтому алгоритм называется *полиномиальной аппроксимационной схемой.* Так как алгоритм работает с любой заданной погрешностью $\varepsilon > 0$, схема называется *полной.* Существуют комбинаторные задачи, где построена *не полная* схема, при которой ε не может быть меньше некоторого числа $a > 0$.

Для описания схемы нам понадобится следующая лемма.

Пусть π_{EDD} – расписание, при котором требования упорядочены по неубыванию директивных сроков (т.е. по правилу EDD). Обозначим через $T_{\max} = \max\limits_{j \in N}\{T_j(\pi_{EDD})\}$ – максимальное запаздывание при данном расписании.

Лемма 3.5 *Выполняется* $T_{\max} \leq F(\pi^*) \leq F(\pi_{EDD}) \leq nT_{\max}$, *где* π^* – *оптимальное расписание задачи.*

Доказательство этой леммы оставляем читателю в качестве упражнения.

Аппроксимационный алгоритм основан на точном алгоритме 7. Покажем, что трудоёмкость точного алгоритма 7 не превосходит $O(n^5 T_{\max})$. Обозначим через $T(N,t)$ – максимальное возможное суммарное запаздывание для множества требований N, обслуживание которых начинается с момента времени t. Несложно вычислить момент времени t^*, при котором $T(N,t) = 0$, $t \leq t^*$ и $T(N,t) > 0$, $t > t^*$, т.е. момент времени, начиная с которого, в некоторых допустимых расписаниях суммарное запаздывание больше 0. Тогда очевидно, что в алгоритме 7 имеет смысл выполнять процедуру $ProcL$(N,t) только для точек t таких, что $t^* \leq t < T_{EDD} < nT_{\max}$. Тогда трудоёмкость $O(n^4 \sum p_j)$ ограничена значением $O(n^5 T_{\max})$.

Построим модифицированный пример, в котором все продолжительности обслуживания подверглись масштабированию, т.е. продолжительности обслуживания $q_j = \lfloor \frac{p_j}{K} \rfloor$ (округление в нижнюю сторону), соответственно, директивные сроки $d'_j = \frac{p_j}{K}$, $j = 1, 2, \ldots, n$ (без округления), где K – некоторое положительное число. Предположим, что с помощью алгоритма 7 для данного модифицированного примера мы построили оптимальное расписание π. Обозначим через T'_A суммарное запаздывание

при этом расписании для примера с продолжительностями обслуживания $p'_j = Kq_j$ и с директивными сроками, совпадающими с исходным примером (d_j). А через T_A – сумарное запаздывание для оригинального примера.

Очевидно, что $Kq_j \leq p_j \leq K(q_j + 1),\ j = 1, 2, \ldots, n$. Используя этот факт, покажем, что $T_A \leq T'_A + K\frac{n(n+1)}{2}$. Обозначим через C'_j – моменты окончания обслуживания требований при расписании π для примера с продолжительностями обслуживания $p'_j = Kq_j$. А через C_j – моменты окончания для исходного примера. Пусть $\pi = (j_1, j_2, j_3, \ldots, j_n)$. Тогда $C_{j_1} - C'_{j_1} \leq K$, $C_{j_2} - C'_{j_2} \leq 2K$, $C_{j_3} - C'_{j_3} \leq 3K$ и т.д., а следовательно, $T_A - T'_A \leq K(1 + 2 + \cdots + n) = K\frac{n(n+1)}{2}$.

Тогда мы получаем, что $T'_A \leq F(\pi^*) \leq T_A \leq T'_A + K\frac{n(n+1)}{2}$. Следовательно, $T_A - F(\pi^*) \leq K\frac{n(n+1)}{2}$. Зададим $K = \frac{2\varepsilon}{n(n+1)}T_{\max}$. Тогда $T_A - F(\pi^*) \leq \varepsilon T_{\max} \leq \varepsilon F(\pi^*)$, т.к. $T_{\max} \leq F(\pi^*)$.

Таким образом, за время $O(n^5 T_{\max}/K)$ эквивалентное $O(\frac{n^7}{\varepsilon})$ было найдено допустимое расписание π с относительной погрешностью не превосходящей ε.

Тогда аппроксимационный алгоритм можно описать следующим образом. Для выбранного значения ε вычисляется K и строится модифицированный пример с продолжительностями обслуживания $q_j = \lfloor \frac{p_j}{K} \rfloor$ и директивными сроками $d'_j = \frac{p_j}{K},\ j = 1, 2, \ldots, n$. Для найденного примера с помощью точного алгоритма 7 строится оптимальное расписание π, которое является приближенным решением для исходного примера.

3.4.3 Алгоритм Муравьиные Колонии

В данном параграфе рассматривается алгоритм решения, основанный на методе Муравьиные Колонии (Ant Colony Optimization, далее – алгоритм ACO) . Этот итерационный алгоритм основан на идее последовательного приближения к оптимальному решению.

Каждая итерация – “запуск искусственного муравья”, – который “пытается” по некоторому правилу выбрать наилучший маршрут к “пище” (к оптимуму функции), используя метки своих предшественников.

Каждый муравей выполняет цепочку шагов. На каждом шаге i, $i = 1, 2, \ldots, n$, выбирается требование j для постановки на место i в расписании, используя "вероятности перехода".

В алгоритме используются параметры:

- η_{ij} – эвристическая информация о том, насколько хорошим "кажется" постановка требования j на место i в расписании. Этот параметр вычисляется эвристически по одному из вариантов:
  1. По правилу EDD: $\eta_{ij} = \frac{1}{d_j},\ i = 1, \ldots, n$;
  2. По правилу MDD (modified due date). В алгоритме MDD последовательно на позиции $i = 1, \ldots, n$ выбирается еще неупорядоченное требование j с наименьшим значением $\max\{S + p_j, d_j\}$, где S – сумма продолжительностей предшествующих упорядоченных требований. Эвристическая информация рассчитывается следующим образом: $\eta_{ij} = \frac{1}{\max\{S+p_j, d_j\}}$;
  3. По правилу L-MDD (Look-ahead MDD): $\eta_{ij} = \frac{1}{Tard_j}$, где $Tard_j$ – суммарное запаздывание при модифицированном расписании MDD, при котором на позиции i обслуживается требование j, а все остальные требования упорядочены в соответствии с правилом MDD;
  4. По правилу SPT: $\eta_{ij} = \frac{1}{p_j},\ i = 1, \ldots, n$;
- τ_{ij} – "след" (в природе: след феромона). После каждой итерации этот параметр корректируется. Параметр показывает насколько "хорошим" для требования j оказалась позиция i. То есть это накопленная статистическая информация о качестве выбора для позиции i требования j, в то время как η_{ij} характеризует предполагаемую выгоду такой постановки при недостатке накопленной информации.

Перед первой итерацией полагают $\tau_{ij} = 1/(mT_{EDD})$, где T_{EDD} – суммарное запаздывание при EDD-расписании, а m – количество итераций (муравьёв). Параметры η_{ij} рассчитываются один раз перед первой итерацией.

На каждом шаге i, $i = 1, 2, \ldots, n$, вычисляется **Матрица вероятностей перехода**:

$$\rho_{ij} = \begin{cases} \dfrac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum\limits_{h \in \Omega} [\tau_{ih}]^\alpha [\eta_{ih}]^\beta}, & j \in \Omega, \\ 0, & j \notin \Omega, \end{cases}$$

где Ω – множество неупорядоченных требований, α и β – "настраиваемые" параметры алгоритма.

Правило, по которому на позицию i выбирается требование j определяется следующим образом:

$$\begin{cases} j = \arg\max\limits_{h \in \Omega} [\tau_{ih}]^\alpha [\eta_{ih}]^\beta, & q < q_0, \\ j \text{ определяется случайным образом,} \\ \text{согласно распределению вероятностей } \rho_{ij}, & q \geq q_0, \end{cases}$$

где $0 \leq q_0 \leq 1$ – параметр алгоритма, а значение q вычисляется случайным образом на каждом шаге.

После того, как требование j было поставлено на позицию i, пересчитывается "локальный след":

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\tau_0,$$

где $\tau_0 = 1/(mT_{EDD})$, а $\rho \in [0, 1]$ – коэффициент распада феромона (параметр алгоритма).

После каждой итерации "глобальный след" τ_{ij} корректируется по правилу:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho/T^*,$$

если в "лучшем" найденном расписании на позиции i обслуживается требование j. Иначе

$$\tau_{ij} = (1 - \rho)\tau_{ij},$$

Значение T^* – суммарное запаздывание для "лучшего" найденного расписания.

Алгоритм ACO дает хорошие результаты при интеграции в него локального поиска, к примеру, попарной перестановки требований. Локальный

поиск запускается после каждой итерации перед пересчетом параметров τ_{ij}.

Нетрудно убедиться, что трудоёмкость алгоритма без локального поиска составляет $O(mn^2)$ операций. Для каждой позиции i (всего n позиций) выбирается требование j за $O(n)$ операций.

Трудоёмкость локального поиска – попарной перестановки – $O(n^3)$. Тогда теоретическая трудоёмкость алгоритма ACO с локальным поиском составляет не меньше $O(mn^3)$ операций, где m – количество муравьёв.

3.4.4 Гибридный алгоритм решения

На основе алгоритма ACO и точного алгоритма 7 с правилами исключения 1–3 можно построить *Гибридный алгоритм*[4]. Идея алгоритма заключается в том, что на каждой итерации запускается модифицированный алгоритм 7, в котором очередное требование j^* "случайным" образом ставится на некоторую позицию $k \in L(N,t)$.

"Случайный" выбор происходит согласно методу "Муравьиные колонии", т.е. подзадача выбора подходящей позиции решается алгоритмом ACO. Модифицированная процедура ProcL, используемая в алгоритме, представлена далее (см. алгоритм 8).

После каждой итерации "глобальный след" τ_{ij} корректируется по правилу:

$$\tau_{ij} = (1-\rho)\tau_{ij} + \rho/T^*,$$

если в "лучшем" найденном расписании на позиции i обслуживается требование j. Иначе

$$\tau_{ij} = (1-\rho)\tau_{ij},$$

где $\rho \in [0,1]$ – параметр алгоритма. Значение T^* – суммарное запаздывание для "лучшего" найденного расписания.

После каждой итерации запускается процедура локального поиска – попарная перестановка требований.

Нетрудно убедиться, что трудоёмкость Гибридного алгоритма без локального поиска составляет $O(mn^2)$ операций. Процедура ProcL запус-

Algorithm 8 Модифицированная процедура ProcL

Procedure ProcL (N, t)

1: Дан пример $\{N, t\}$ с множеством требований $N = \{j_1, j_2, \ldots, j_n\}$ и моментом начала обслуживания t, где $d_{j_1} \leq d_{j_2} \leq \cdots \leq d_{j_n}$.
2: **if** $N = \emptyset$ **then**
3: π^*:= пустое расписание;
4: **else**
5: Найдём требование j^* из множества N;
6: Найдём множество $L(N, t)$ для требования j^*;
7: Рассчитаем матрицу вероятностей перехода для каждой позиции $i \in L(N, t)$.

$$\rho_{ij*} = \frac{\tau_{ij*}/F(\pi^i, t)}{\sum\limits_{h \in L(N,t)} \tau_{hj*}/F(\pi^h, t)},$$

где $\pi^i = (j_1, \ldots, j_{m-1}, j_{m+1}, \ldots, j_i, j^*, j_{i+1}, \ldots, j_n)$, $d_{j_1} \leq \ldots, d_{j_n}$, $j^* = j_m, m < i$, $F(\pi^i, t)$ – суммарное запаздывание при расписании π^i, построенном с момента времени t.
8: Выберем позицию $k \in L(N, t)$ произвольным образом согласно распределению вероятностей ρ_{ij*}.
9: Пересчитаем "локальный след":

$$\tau_{kj*} = (1 - \rho)\tau_{kj*} + \rho\tau_0,$$

где $\tau_0 = 1/(mT_{EDD})$. T_{EDD} – суммарное запаздывание при EDD-расписании.
10: $\pi_k := (\mathbf{ProcL}(N', t'), j^*(N), \mathbf{ProcL}(N'', t''))$, где

$N' := \{j_1, \ldots, j_k\} \setminus \{j^*\}$, $t' := t$,

$N'' := \{j_{k+1}, \ldots, j_n\}$, $t'' := t + \sum\limits_{i:=1}^{k} p_{j_i}$;

11: **end if**
12: **RETURN** π^*;

кается n раз. В каждой процедуре выполняется порядка $O(n)$ действий.

Трудоёмкость локального поиска (попарная перестановка) – $O(n^3)$. Тогда теоретическая трудоёмкость Гибридного алгоритма с локальным поиском составляет не меньше $O(mn^3)$ операций. То есть трудоёмкость идентична трудоёмкости алгоритма ACO.

3.4.5 Эффективность алгоритмов для тестовых примеров Поттса и ван Вассенхова

Эффективность различных алгоритмов решения некоторой задачи комбинаторной оптимизации зачастую сравнивают экспериментально. Например, можно сравнить время работы алгоритмов, количество итераций, производимых алгоритмом, точность полученного решения и т.д. В качестве иллюстрации того, как могут проводится эксперименты, в данном параграфе представлены результаты экспериментального сравнения Гибридного алгоритма и алгоритма ACO. Эксперименты проводились для примеров размерности $n = 4, \ldots, 70, 100$, которые генерировались с помощью схемы Поттса и ван Вассенхова [89].

Примеры генерировались следующим образом. Значения $p_j \in Z$ распределены по равномерному закону на промежутке $[1, 100]$. Значения d_j генерировались случайным образом согласно равномерному закону распределения на промежутке

$$\left[\sum_{j:=1}^{n} p_j(1 - TF - RDD/2), \sum_{j:=1}^{n} p_j(1 - TF + RDD/2)\right].$$

Параметры TF и RDD принимают значения из множества $\{0.2, 0.4, 0.6, 0.8, 1\}$. Для каждой комбинации (TF, RDD) генерировалось по 100 примеров (всего 2500 примеров для каждого n).

Примеры, для которых $F(\pi_{EDD}) = 0$, не рассматривались, т.к. в этом случае Гибридный алгоритм и ACO гарантированно находят точное решение – π_{EDD} – расписание построенное по правилу EDD.

В алгоритмах использовались следующие параметры: $\alpha = 1$; $\beta = 2$; $\rho = 0, 1$. Локальный поиск – попарная перестановка. Применяемая эвристика – MDD.

Для каждого примера запускался точный алгоритм 7. В результате его работы мы получали оптимальное значение целевой функции F_{opt}.

В алгоритме ACO муравьи генерировались до тех пор, пока не находилось расписание π, при котором $F(\pi) = F_{opt}$. Это "необходимое" количество муравьёв фиксировалось. Число муравьёв нами было ограничено $m \leq 100$. Перед первой итерацией параметры τ_{ij} рассчитывались по

формуле $\tau_{ij} = 1/(T_{EDD}),\ i = 1, \ldots, n, j = 1, \ldots, n,\ \tau_0 = 1/(T_{EDD})$.

Алгоритм запускался до 10 раз (пока не найдено оптимальное расписание). Таким образом мы пытались избежать "случайности" работы алгоритма.

Лучшее найденное значение целевой функции F_{ACO} фиксировалось. Вычислялась относительная погрешность $\frac{F_{ACO}-F_{opt}}{F_{opt}}$.

Такая же схема использовалась для Гибридного алгоритма. Таким образом мы могли сравнить относительную погрешность двух алгоритмов (ACO и Гибридного), количество муравьёв, необходимое каждому алгоритму для нахождения оптимального расписания.

Результаты экспериментов представлены в таб. 3.3.

Для размерности задачи $n = 100$ рассмотрено 617 примеров. Для остальных размерностей по 2500 примеров. В таблице выводятся только те строки, соответствующие размерности задачи n, для которых число неточно решенных примеров больше 0.

В первой колонке представлено значение n – размерность задачи. Во второй и третьей колонках, соответственно, число примеров, для которых алгоритм ACO или Гибридный не нашли точные решения. В четвертой и пятой колонках представлена максимальная относительная погрешность алгоритмов (с точностью до сотых процента). В последних двух колонках – среднее число муравьёв, необходимое для поиска оптимального решения.

Как видно из таблицы, примерно для 99% примеров оба алгоритма находят точные решения.

Число примеров, неточно решенных Гибридным алгоритмом, значительно меньше, чем алгоритмом ACO, и не превосходит 0,44% от общего числа примеров. Относительная погрешность не превосходит 0,46%. Число муравьёв, необходимое Гибридному алгоритму также меньше. Заметно, что для $n = 100$ Гибридному алгоритму в среднем необходимо 4–5 муравьёв для нахождения оптимального расписания, в то время как алгоритму ACO требуется в среднем более 27 муравьёв.

Относительная погрешность алгоритма ACO превосходит 1,26% для $n =$

Таблица 3.3: Результаты экспериментов

n	Число неоптимально решенных примеров		Максимальная относительная погрешность		Число муравьёв	
	ACO	Гибридный	*ACO*	Гибридный	*ACO*	Гибридный
19	2	0	0,22	0	1,6164	1,4004
20	1	0	0,58	0	1,6064	1,4204
22	1	0	0,16	0	1,626	1,4844
28	2	0	0,09	0	1,9704	1,6688
34	1	0	0,15	0	2,2212	1,9568
36	0	1	0	0,04	2,2332	2,154
37	1	1	0,38	0,01	2,4796	2,102
40	1	0	0,04	0	2,6036	2,2424
42	1	1	0,05	0,01	2,7888	2,4092
43	1	1	0,07	0,06	2,7316	2,3656
44	3	0	0,04	0	2,8464	2,3784
45	2	0	0,68	0	2,9736	2,4728
46	1	0	0,03	0	3,1624	2,4088
47	2	0	0,01	0	3,248	2,5152
48	9	0	0,56	0	3,4516	2,5196
49	3	1	0,15	0,08	3,4252	2,7
50	9	1	0,35	0,29	3,716	2,6336
51	8	0	0,22	0	3,8412	2,7768
52	4	1	0,04	0,07	3,5816	2,86
53	4	2	0,03	0,42	3,8948	2,9668
54	9	3	0,1	0,29	4,0324	2,9924
55	8	2	0,11	0,06	4,1048	3,0496
56	9	1	0,83	0,01	4,2916	3,0064
57	7	0	0,23	0	4,1568	3,158
58	14	0	0,17	0	4,71	3,3724
59	14	4	0,24	0,1	4,81	3,3372
60	11	1	0,22	0,01	4,7268	3,4224
61	18	2	1,26	0,02	5,3032	3,5216
62	10	2	0,26	0,01	5,0964	3,5032
63	17	7	0,16	0,08	5,3016	3,5728
64	15	6	0,57	0,46	5,2388	3,6504
65	18	7	0,1	0,14	5,548	3,6604
66	17	11	0,15	0,14	5,4288	3,8552
67	17	7	0,83	0,1	6,1068	4,1016
68	25	4	0,2	0,08	6,3864	3,7252
69	18	6	0,12	0,1	6,1912	4,0796
70	33	4	0,23	0,05	6,974	3,8672
100	36	0	0,31	0	27,35	4,66

61. Число "неточно" решенных примеров для $n = 70$ больше 1% от общего числа примеров. Следует ожидать, что с ростом n будет наблюдаться значительное преимущество Гибридного алгоритма.

Ознакомившись с полученными результатами, можно сделать вывод, что построение "смешанных" гибридных алгоритмов дает хорошие результаты.

3.5 Минимизация обобщенной функции запаздывания

В предыдующих параграфах мы познакомились с двумя задачами, целевые функции которых связаны с запаздыванием требований $1||\sum w_jU_j$ и $1||\sum T_j$. В этом параграфе приводится обобщение этих двух задач.

В дополнении к обычным параметрам p_j и d_j, для каждого требования j, $j = 1, 2, \ldots, n$, заданы *квота запаздывания* $b_j \geq 0$, *коэффициент нормального запаздывания* $v_j \geq 0$ и *коэффициент ненормального запаздывания* $w_j \geq 0$.

Обобщенное запаздывание задается следующим образом:

$$GT_j(\pi) = \begin{cases} 0, & \text{если } C_j(\pi) - d_j \leq 0, \\ v_j \cdot (C_j(\pi) - d_j), & \text{если } 0 < C_j(\pi) - d_j \leq b_j, \\ w_j, & \text{если } b_j < C_j(\pi) - d_j, \end{cases}$$

где $w_j \geq v_jb_j$ для каждого $j \in N$. Целевая функция задачи:

$$F(\pi) = \sum_{j=1}^{n} GT_j(\pi) \to \min.$$

Функцию обощённого запаздывания можно интерпретировать следующим образом. Если запаздывание требования j превышает параметр b_j, то значение штрафа уже не зависит от запаздывания, остается постоянным и равным w_j. Необходимо найти расписание, минимизирующее значение $F(\pi)$. Эту задачу будем обозначать как $1||\sum GT_j$.

Очевидно, что эта задача NP-трудна, так как ее частный случай $b_j = 0$ соответствует NP-трудной проблеме $1||\sum w_jU_j$.

В данном параграфе мы рассмотрим частный случай задачи, при котором

$$b_j = p_j, \quad v_j = 1, \quad w_j = p_j, \tag{3.2}$$

т.е., $GT_j(\pi) = \min\{\max\{0, C_j(\pi) - d_j\}, p_j\}$ для всех $j \in N$.

Теорема 3.9 *Частный случай (3.2) задачи* $1\|\sum GT_j$ *является NP-трудным.*

Доказательство. Сведем к данному частному случаю ЗАДАЧУ РАЗБИЕНИЯ. Дан пример ЗАДАЧИ РАЗБИЕНИЯ. Построим пример частного случая (3.2) с $n+1$ требованиями. Продолжительности обслуживания $p_j = b_j$, $j = 1, 2, \ldots, n$, и $p_{n+1} = 1$, т.е. $\sum\limits_{j=1}^{n+1} p_j = 2A + 1$. Директивные сроки $d_j = A$, $j = 1, 2, \ldots, n$, и $d_{n+1} = A + 1$. При любом расписании π выполняется $F(\pi) \geq A = \sum\limits_{j=1}^{n+1} p_j - d_{n+1}$. Более того, для всех расписаний выполняется $F(\pi) \leq A + 1$.

При оптимальном расписании $\pi^* = (\pi_1, n+1, \pi_2)$ (без простоев) равенство $C_{n+1}(\pi^*) = A + 1$ выполняется тогда и только тогда, когда ответ в примере ЗАДАЧИ РАЗБИЕНИЯ – "ДА". Все требования из частичного расписания π_1 и требование $n+1$ не запаздывают, а требования из π_2 запаздывают. Требование из частичного расписания π_1 соответствуют элементам найденного подмножества чисел N' примера ЗАДАЧИ РАЗБИЕНИЯ. Причем, $F(\pi^*) = A$. Таким образом задачи (3.2) $1\|\sum GT_j$ и РАЗБИЕНИЯ являются эквивалентными, что и доказывает NP-трудность частного случая (3.2). □

Лемма 3.6 *Для частного случая (3.2) существует оптимальное расписание вида* $\pi = (G, H)$*, где для всех требований* $i \in G$ *выполняется* $0 \leq GT_i(\pi) < p_i$*, а для требований* $j \in H$ *выполняется* $GT_i(\pi) = p_i$*. Требования из множества* G *обслуживаются в порядке EDD, а требования из множества* H *– в порядке LDD.*

Доказательство. 1) Допустим, что существует оптимальное расписание $\pi^* = (\pi_1, j, \pi_2)$. Если $GT_j(\pi) = p_j$, тогда расписание $\pi' = (\pi_1, \pi_2, j)$ также оптимальное. Т.е существует оптимальное расписание вида $\pi =$

(G, H), где все требования $j \in H$ запаздывают и $GT_j(\pi) = p_j$. Для всех требований $i \in G$ выполняется $0 \leq GT_i(\pi) < p_i$.

2) Рассмотрим оптимальное расписание вида $\pi = (G, H)$, где все требования $j \in H$ запаздывают и $GT_j(\pi) = p_j$, а для всех требований $i \in G$ выполняется $0 \leq GT_i(\pi) < p_i$. Докажем, что требования $i \in G$ обслуживаются в порядке EDD.

Предположим, что существует оптимальное расписание вида $\pi = (\pi_1, \alpha, \beta, \pi_2)$, где требования $\alpha, \beta \in G$, и выполняется $d_\alpha > d_\beta$. А также выполняется $C_\alpha(\pi) - d_\alpha < p_\alpha$ и $C_\beta(\pi) - d_\beta < p_\beta$, тогда $C_\alpha(\pi) < d_\alpha$.

Рассмотрим расписание $\pi' = (\pi_1, \beta, \alpha, \pi_2)$. Определим $C = C_\beta(\pi) = C_\alpha(\pi')$. Тогда

$$F(\pi) - F(\pi') = (GT_\alpha(\pi) - GT_\alpha(\pi')) + (GT_\beta(\pi) - GT_\beta(\pi')) =$$

$$= -\min\{p_\alpha, \max\{0, C - d_\alpha\}\} + \min\{p_\alpha, \max\{0, C - d_\beta\}\} \geq 0$$

и расписание π' также оптимально.

3) Докажем, что требования $j \in H$ обслуживаются в порядке LDD. Для всех требований $j \in H$ имеем $d_j \leq \sum\limits_{l=1}^{n} p_l - \sum\limits_{k \in H} p_k$, иначе, если $d_j > \sum\limits_{l=1}^{n} p_l - \sum\limits_{k \in H} p_k$, тогда при расписании $\pi' = (G, j, H \setminus \{j\})$ мы имеем меньшее значение целевой функции, т.е. получили противоречие, т.к. π – оптимальное расписание. Поэтому требования из H могут быть обслужены в любом порядке. □

Этот результат аналогичен теореме 3.4. Пользуясь этой леммой, можно построить точный алгоритм решения данного частного случая, аналогичный алгоритму 6. Трудоёмкость построенного алгоритма будет равна $O(nd_{\max})$ операций, где $d_{\max}$ – максимальный директивный срок.

3.6 Одноприборные задачи с обратными критериями оптимизации

Представьте себе, что нам необходимо максимизировать суммарное запаздывание или максимизировать число запаздывающих требований, в

отличие от классических задач, где эти значения нужно минимизировать. Несмотря на абсурдность такой максимизации, данные задачи представляют собой теоретический и практический интерес. В данном параграфе рассматриваются одноприборные задачи с "обратными" критериями оптимальности, например, задачи максимизации суммарного запаздывания и максимизации количества запаздывающих требований для одного прибора.

Формулируются эти задачи следующим образом, практически совпадающим с формулировками классических задач минимизации.

Необходимо обслужить n требований на одном приборе. Прерывания при обслуживании и обслуживание более одного требования в любой момент времени запрещены. Для каждого требования $j \in N = \{1, 2, \ldots, n\}$ заданы продолжительность обслуживания $p_j > 0$ и директивный срок его окончания d_j, где N – множество требований, которые необходимо обслужить. Прибор начинает обслуживание требований с момента времени 0. **Простои прибора при обслуживании требований запрещены** (иначе задачи максимизации становятся тривиальными). Расписание обслуживания требований $\pi = (j_1, j_2, \ldots, j_n)$ строится с момента времени 0 и однозначно задаётся перестановкой элементов множества N. Обозначим через $C_{j_k}(\pi) = \sum_{l=1}^{k} p_{j_l}$ время завершения обслуживания требований j_k при расписании π. Если $C_j(\pi) > d_j$, тогда требование j запаздывает, и в этом случае полагают $U_j = 1$. Если $C_j(\pi) \leq d_j$, тогда требование j не запаздывает, и $U_j = 0$

Требуется построить расписание π^* обслуживания требований множества N, при котором достигается максимум функции $F(\pi) = \sum_{j=1}^{n} U_j(\pi)$. Обозначим данную задачу через $1(nd)||\max\sum U_j$. Аналогично, обозначается задача максимизации суммарного запаздывания $1(nd)||\max\sum T_j$ и другие задачи максимизации.

Запись (nd) означает, что рассматриваются расписания, при которых прибор не простаивает (non-delay), т.е. все требования обслуживаются в интервале $[0, \sum p_j]$ без перерывов. В литературе можно встретить и другие задачи максимизации, обозначаемые символами $1(sa)|\ldots|\max\ldots$, в которых рассматриваются т.н. semi-active расписания.

В этом параграфе приводятся доказательства NP-трудности некоторых задач максимизации, а также полиномиальный графический алгоритм решения задачи $1(nd)||\max\sum T_j$.

С одной стороны, исследование данных задач само по себе является важной теоретической задачей. Алгоритмы решения данных задач могут быть использованы для вычисления верхних оценок, исследования свойств оптимальных расписаний, вычисления частичного порядка обслуживания требований для "исходных" минимизационных задач, а также для сокращения перебора в алгоритмах решения "исходных" задач. Например, алгоритм решения задачи максимизации количества запаздываюших требований $1||\max\sum U_j$ используется для вычисления максимального количества запаздываюших требований, что, в свою очередь, позволяет сократить перебор в алгоритме решения задачи максимизации суммарного запаздывания $1||\max\sum T_j$. Значение $\max\sum T_j$ может быть использовано при решении классической задачи минимизации $1||(\alpha\sum E_j+\beta\sum T_j)$. Например, если $\alpha>\beta$, тогда можно вычислить максимальное значение $\beta\sum T_j$ и после искать расписание, оптимальное только с точки зрения критерия $\sum E_j$.

Также теоретический интерес представляет корреляция между полиномиально разрешимыми и NP-трудными случаями для "исходной" и "обратной" задач.

С другой стороны, для данных проблем существуют практические интерпретации и приложения. Например, монтажная команда должна смонтировать ветряные электрогенераторы (турбины) в разных районах страны. В каждом районе j необходимо смонтировать определенное количество турбин. Время монтажа p_j зависит только от количества турбин и не зависит от погодных или климатических условий. Однако погода влияет на дополнительные расходы (например, на расход топлива, на зарплату и стоимость проживания рабочих, которые могут быть выше зимой). Сумма этих дополнительных расходов начинает быстро снижаться после схода снега. Для каждого региона (т.е. для каждого требования) дан прогноз, когда ожидается сход снега, т.е. когда снег растает (этот момент времени можно интерпретировать как директивный срок). Целевая функция – минимизировать эти дополнительные расходы, т.е. целевая функция может быть интерпретирована как $\max\sum\max\{0,S_j-d'_j\}$, где

$S_j = C_j - p_j$ и $d'_j = d_j - p_j$. В результате получили задачу $1||\max\sum T_j$.

Сведения о трудоёмкости задач максимизации сведены в таблицу 3.4

Большинство результатов для задач максимизации, представленных в таблице, могут быть получены без особых усилий. Далее приведены три простых доказательства NP-трудности некоторых из перечисленных задач.

Лемма 3.7 *Задачи* $1(nd)|D_j|\max\sum T_j$ *и* $1(nd)|D_j|\max\sum U_j$ *NP-трудны.*

Доказательство. Доказательство проведем сведением ЗАДАЧИ РАЗБИЕНИЯ к данным задачам.Дан пример ЗАДАЧИ РАЗБИЕНИЯ с n числами. Конструируем пример исследуемых задач следующим образом. В примере $n+1$ требование, где $p_j = b_j$ и $d_j = D_j = \sum\limits_{j=1}^{n} p_j + 1$, $j = 1, 2, \ldots, n$, т.е. при любом допустимом расписании эти требований не запаздывают. Пусть $p_{n+1} = 1$, $d_{n+1} = A$, $D_{n+1} = A + 1$, где $A = \frac{1}{2}\sum\limits_{j=1}^{n} p_j$.

Если ответ в примере ЗАДАЧИ РАЗБИЕНИЯ "ДА", тогда существует оптимальное расписание $\pi = (\pi_1, n+1, \pi_2)$, где $\{\pi_1\} = N'$, $\sum\limits_{j\in N'} p_j = A$, $\{\pi_2\} = N \setminus N'$ и $\sum\limits_{j=1}^{n+1} T_j(\pi) = 1$. Если ответ "НЕТ", тогда для всех допустимых расписаний $\sum\limits_{j=1}^{n+1} T_j = 0$. Соответственно, выполняются $\sum\limits_{j=1}^{n+1} U_j = 1$ и $\sum\limits_{j=1}^{n+1} U_j = 0$. □

Лемма 3.8 *Задачи* $1(nd)|r_j|\max\sum w_jC_j$ *и* $1(nd)|D_j|\max\sum w_jC_j$ *NP-трудны.*

Доказательство. Доказательство проведем сведением ЗАДАЧИ РАЗБИЕНИЯ к данным задачам.Дан пример ЗАДАЧИ РАЗБИЕНИЯ с n числами. Конструируем пример задачи $1(nd)|r_j|\max\sum w_jC_j$ следующим образом. В примере $n+1$ требование, где $p_j = w_j = b_j$ и $r_j = 0$, $j = 1, 2, \ldots, n$. Пусть $p_{n+1} = 1$, $w_{n+1} = 0$, $r_{n+1} = A$, где $A = \frac{1}{2}\sum\limits_{j=1}^{n} p_j$.

Таблица 3.4: Сведения о трудоёмкости задач максимизации

Задача	трудоёмкость и алгоритмы	трудоёмкость соотв. задачи минимизации
$1(nd)\|\|\max\sum U_j$	Полин. разрешима за время $O(n\log n)$	Полин. разрешима за время $O(n\log n)$
$1(nd)\|D_j\|\max\sum T_j$	NP-трудна	???
$1(nd)\|D_j\|\max\sum U_j$	NP-трудна	???
$1(nd)\|D_j\|\max\sum w_jC_j$	NP-трудна	???
$1(nd)\|D_j\|\max\sum C_j$	NP-трудна	???
$1(nd)\|\|\max\sum w_jU_j$	NP-трудна	NP-трудна
$1(nd)\|r_j\|\max\sum U_j$	NP-трудна	NP-трудна
$1(nd)\|r_j\|\max\sum w_jC_j$	NP-трудна	NP-трудна
$1(nd)\|r_j\|\max\sum C_j$	Полин. разрешима за время $O(n\log n)$	NP-трудна
$1(nd)\|prec,r_j\|\max C_{\max}$	$O(n^2)$	NP-трудна
$1(nd)\|prec,r_j\|\max f_{\max}$	$O(n^4)$	NP-трудна
$1(nd)\|r_j\|\max f_{\max}$	$O(n^3)$	NP-трудна
$1(nd)\|prec,r_j\|\max\sum C_j$	NP-трудна	NP-трудна
$1(nd)\|prec,r_j\|\max\sum U_j$	NP-трудна	NP-трудна
$1(nd)\|\|\max\sum w_jT_j$	NP-трудна, Алгоритм решения трудоёмкости $O(n\min\{\sum w_j, d_{\max}\})$	NP-трудна
$1(nd)\|r_j\|\max\sum w_jT_j$	NP-трудна	NP-трудна
$1(nd)\|r_j\|\max\sum T_j$	NP-трудна	NP-трудна
$1(nd)\|\|\max\sum T_j$	Алгоритм решения трудоёмкости $O(n\sum p_j)$. Алгоритм решения трудоёмкости $O(n^2)$	NP-трудна. Алгоритм решения трудоёмкости $O(n^4\sum p_j)$
$1(sa)\|r_j\|\max\sum T_j$	Полин.разрешима за время $O(n^3)$	
$1(sa)\|r_j\|\max\sum w_jT_j$	NP-трудна	

Если ответ в примере ЗАДАЧИ РАЗБИЕНИЯ “ДА”, тогда существует оптимальное расписание $\pi = (\pi_1, n+1, \pi_2)$, где $\{\pi_1\} = N'$, $\sum\limits_{j\in\pi_1} p_j = A$, $\{\pi_2\} = N \setminus N'$, $\sum\limits_{j\in\pi_2} p_j = \sum\limits_{j\in N} b_j - A$ и

$$\sum_{j=1}^{n+1} w_j C_j(\pi) = \sum_{1\le i\le j\le n} b_i b_j + \left(\sum_{j\in N} b_j - A\right),$$

т.к. при расписании $\pi' = (\pi_1, \pi_2, n+1)$ выполняется

$$\sum_{j=1}^{n+1} w_j C_j(\pi) = \sum_{1\le i\le j\le n} b_i b_j.$$

Требования в частичных расписаниях π_1 и π_2 могут быть обслужены в любом порядке. Если ответ “НЕТ”, то

$$\sum_{j=1}^{n+1} w_j C_j(\pi) < \sum_{1\le i\le j\le n} b_i b_j + \left(\sum_{j\in N} b_j - A\right).$$

Следовательно частный случай задачи $1(nd)|r_j|\max\sum w_jC_j$ эквивалентен NP-трудной задаче РАЗБИЕНИЯ, поэтому задача $1(nd)|r_j|\max\sum w_jC_j$ является NP-трудной.

Аналогично для задачи $1(nd)|D_j|\max\sum w_jC_j$, мы конструируем пример с множеством из $n+1$ требований, где $p_j = b_j, w_j = 0$, $D_j = \sum\limits_{i=1}^{n+1} p_i$, $j = 1, 2, \ldots, n$, а также $p_{n+1} = 1$, $w_{n+1} = 1$ и $D_{n+1} = A+1$. Ответ для примера ЗАДАЧИ РАЗБИЕНИЯ “ДА” тогда и только тогда, когда при оптимальном расписании π выполняется $C_{n+1}(\pi) = A+1$. □

Лемма 3.9 *Задача* $1(nd)|D_j|\max\sum C_j$ *NP-трудна.*

Доказательство. Очевидно, что две задачи $1|r_j|\min\sum C_j$ и $1|r_j|\min\sum S_j$ эквивалентны, т.к. $\sum\limits_{j\in N} C_j = \sum\limits_{j\in N} S_j + \sum\limits_{j\in N} p_j$. Известно, что задача $1|r_j|\min\sum C_j$ NP-трудна. Легко показать, что задачи $1(nd)|D_j|\max\sum C_j$ и $1|r_j|\min\sum S_j$ также эквивалентны. Пусть имеется

пример задачи $1|r_j|\min\sum S_j$ с множеством N из n требований. Для примера задачи $1(nd)|D_j|\max\sum C_j$, зададим n требований с параметрами p'_j, D'_j, определенными по правилам: $p'_j = p_j$, $D'_j = \sum\limits_{j\in N} p_j - r_j$. Рассмотрим расписание $\pi = (j_1, j_2, \ldots, j_k, j_{k+1}, \ldots, j_n)$. Определим $H_{j_k} = \sum\limits_{i=k+1}^{n} p_{j_i}$. Тогда задача максимизации $\sum\limits_{j\in N} C_j = n\sum\limits_{j\in N} p_j - \sum\limits_{j\in N} H_j$ сводится к задаче минимизации $\sum\limits_{j\in N} H_j$, т.е. сводится к задаче $1|r_j|\min\sum S_j$.

Пусть $\pi = (1, 2, \ldots, n)$ – оптимальное расписание для примера задачи $1|r_j|\min\sum S_j$. Тогда расписание $\pi' = (n, \ldots, 2, 1)$ будет оптимальным для соответствующего примера задачи $1|D_j|\max\sum C_j$.

Поэтому две проблемы эквивалентны и, следовательно, исследуемая задача $1(nd)|D_j|\max\sum C_j$ является NP-трудной. □

3.6.1 Доказательство NP-трудности задачи $1(nd)||\max\sum w_jT_j$

Далеко не все доказательства NP-трудности для задач **TP** столь тривиальны. Для иллюстрации нетрививиального доказательства рассмотрим задачу $1(nd)||\max\sum w_jT_j$. Доказательство проведем сведением ЗАДАЧИ РАЗБИЕНИЯ к частному случаю задачи $1(nd)||\max\sum w_jT_j$. Без потери общности, пусть в примере ЗАДАЧИ РАЗБИЕНИЯ $n > 3$ и $\sum\limits_{i=1}^{n} b_i > 10$.

Дан произвольный пример ЗАДАЧИ РАЗБИЕНИЯ, мы конструируем следующий пример задачи $1(nd)||\max\sum w_jT_j$:

$$\begin{cases} w_{2i} = M^i,\ i = 1, 2, \ldots, n, & (3.3.1)\\ w_{2i-1} = w_{2i} + b_i,\ i = 1, 2, \ldots, n, & (3.3.2)\\ p_{2i} = \sum\limits_{j=1}^{i-1} w_{2j} + \dfrac{1}{2}\sum\limits_{j\in N\setminus\{i\}} b_j,\ i = 1, 2, \ldots, n, & (3.3.3)\\ p_{2i-1} = p_{2i} + b_i,\ i = 1, 2, \ldots, n, & (3.3.4)\\ d_{2i} = d_{2i-1} = P - \sum\limits_{j=i}^{n} p_{2j},\ i = 1, 2, \ldots, n, & (3.3.5) \end{cases} \tag{3.3}$$

где $M = (n\sum_{i=1}^{n} b_i)^{10}$ и $P = \sum_{j=1}^{2n} p_j$.

Обозначим работы из множества $\bar{N} = \{1, 2, \ldots, 2n\}$ следующим образом:

$$V_1, V_2, V_3, V_4, \ldots, V_{2i-1}, V_{2i}, \ldots, V_{2n-1}, V_{2n}.$$

Каноническим расписанием будем называть расписание вида $(V_{n,1}, V_{n-1,1}, \ldots, V_{i,1}, \ldots, V_{1,1}, V_{1,2}, \ldots, V_{i,2}, \ldots, V_{n-1,2}, V_{n,2})$, где $\{V_{i,1}, V_{i,2}\} = \{V_{2i-1}, V_{2i}\}$, $i = 1, 2, \ldots, n$.

Лемма 3.10 *Для каждого примера задачи* $1(nd)||\max\sum w_jT_j$ *существует оптимальное расписание вида* $\pi = (G, H)$, *где все требования* $j \in H$ *запаздывают, а все требования* $i \in G$ *не запаздывают. Все требования из множества* G *обслуживаются в порядке невозрастания значений* $\frac{w_j}{p_j}$, *а все требования из множества* H *обслуживаются в порядке неубывания значений* $\frac{w_j}{p_j}$.

Доказательство.

1) Предположим, что существует оптимальное расписание вида $\pi = (\pi_1, j, \pi_2, i, \pi_3)$, при котором требование j запаздывает, а требование i не запаздывает. Для расписания $\pi' = (\pi_1, i, j, \pi_2, \pi_3)$ выполняется: $F(\pi') - F(\pi) \geq w_j(T_j(\pi') - T_j(\pi)) + w_i(T_i(\pi') - T_i(\pi)) = w_j(p_i) + (0) > 0$. Получили противоречие, так как для расписания π' значение целевой функции больше, а значит расписание π не оптимальное. Повторив необходимое число раз подобное преобразование расписания, мы получим оптимальное расписание вида $\pi = (G, H)$.

2) Рассмотрим оптимальное расписание вида $\pi = (G, H)$, где все требования $j \in H$ запаздывают, а все требования $i \in G$ не запаздывают. Покажем, что все требования $j \in H$ обслуживаются в порядке неубывания значений $\frac{w_j}{p_j}$. Предположим, что существует оптимальное расписание $\pi = (\pi_1, j_1, j_2, \pi_2)$, при котором требования j_1 и j_2 запаздывают и $\frac{w_{j_1}}{p_{j_1}} > \frac{w_{j_2}}{p_{j_2}}$, т.е. $w_{j_1}p_{j_2} > w_{j_2}p_{j_1}$. Для расписания $\pi' = (\pi_1, j_2, j_1, \pi_2)$, выполняется $F(\pi') - F(\pi) = w_{j_1}(T_{j_1}(\pi') - T_{j_1}(\pi)) + w_{j_2}(T_{j_2}(\pi') - T_{j_2}(\pi)) \geq$

$w_{j_1}p_{j_2} - w_{j_2}\min\{p_{j_1}, T_{j_2}(\pi)\} > 0$. Получили противоречие, а значит, расписание $\pi = (\pi_1, j_1, j_2, \pi_2)$ не оптимальное. После необходимого количества преобразований расписания π мы получим оптимальное расписание, при котором все требования множества H будут упорядочены в порядке неубывания величин $\frac{w_j}{p_j}$.

3) Рассмотрим оптимальное расписание вида $\pi = (G, H)$, где все требования $j \in H$ запаздывают, а все требования $i \in G$ не запаздывают. Покажем что все требования $i \in G$ могут обслуживаться в порядке невозрастания значений $\frac{w_j}{p_j}$ при оптимальном расписании. Для всех требований $i \in G$ выполняется $d_i \geq \sum\limits_{k \in G} p_k$, иначе, если $d_i < \sum\limits_{k \in G} p_k$, то расписание $\pi' = (G\backslash\{i\}, i, H)$ "лучше" (т.е. для данного расписания значение целевой функции больше), а следовательно, получено противоречие. Поэтому все требования $i \in G$ могут быть обслужены в любом порядке (в том числе, и в порядке невозрастания величин $\frac{w_j}{p_j}$), так как при любом порядке обслуживания все требования из G не запаздывают. □

Следствием из этой леммы является следующее утверждение.

Лемма 3.11 *Для каждого примера задачи* $1(nd)||\max\sum T_j$ *существует оптимальное расписание вида* $\pi = (G, H)$*, где все требования* $j \in H$ *запаздывают, а все требования* $i \in G$ *не запаздывают. Все требования из множества* G *обслуживаются в порядке неубывания продолжительности обслуживания (SPT), а все требования из множества* H *обслуживаются в порядке невозрастания продолжительности обслуживания (LPT).*

Лемма 3.12 *Для частного случая (3.3), все оптимальные расписания являются каноническими, или могут быть сведены к каноническим расписаниям, если упорядочить первые* n *требований в расписании по правилу LPT (longest processing time).*

Доказательство.

1) Сначала докажем, что одно из двух требований, V_{2n} или V_{2n-1}, запаздывает при любом оптимальном расписании. Предположим, что оба требования не запаздывают при некотором оптимальном расписании

$\pi = (\pi_1, V_{2n}, \pi_2, V_{2n-1}, \pi_3, \pi_4)$, где запаздывают только требования из частичного расписания π_4. Мы рассматриваем только оптимальные расписания вида $\pi = (G, H)$, где все требования $j \in H$ запаздывают, а все требования $i \in G$ не запаздывают (см. лемму 3.10). Для расписания $\pi' = (\pi_1, \pi_2, V_{2n-1}, \pi_3, \pi_4, V_{2n})$ имеем:

$$\sum_{j=1}^{2n} w_j T_j(\pi') - \sum_{j=1}^{2n} w_j T_j(\pi) \geq w_{2n} T_{2n}(\pi') - p_{2n} \sum_{j \in \{\pi_4\}} w_j \geq$$

$$\geq p_{2n} M^n - p_{2n} \sum_{i=1}^{2n-1} (2M^i + b_i) = p_{2n} M^n - p_{2n} \left(2M \frac{M^{n-1} - 1}{M - 1} + \sum_{i=1}^{2n-1} b_i \right) > 0.$$

Значит расписание π не оптимальное, т.е. одно и только одно (см.директивные сроки) из двух требований, V_{2n} или V_{2n-1}, запаздывает при любом оптимальном расписании.

2) Докажем верность следующего неравенства: существует оптимальное расписание, при котором

$$\frac{w_2}{p_2} < \frac{w_4}{p_4} < \cdots < \frac{w_{2n}}{p_{2n}}.$$

Необходимо доказать, что

$$\frac{M^{i-1}}{\sum\limits_{j=1}^{i-2} w_{2j} + \frac{1}{2} \sum\limits_{j \in N \setminus \{i-1\}} b_j} < \frac{M^i}{\sum\limits_{j=1}^{i-1} w_{2j} + \frac{1}{2} \sum\limits_{j \in N \setminus \{i\}} b_j}.$$

Обозначим $B_i = \frac{1}{2} \sum\limits_{j \in N \setminus \{i\}} b_j,\ i = 1, 2, \ldots, n$. Тогда мы имеем:

$$\frac{M^{i-1}}{M\frac{M^{i-2}-1}{M-1} + B_{i-1}} < \frac{M^i}{M\frac{M^{i-1}-1}{M-1} + B_i}$$

$$\iff \frac{1}{\frac{M(M^{i-2}-1)+B_{i-1}(M-1)}{M-1}} < \frac{M}{\frac{M(M^{i-1}-1)+B_i(M-1)}{M-1}}$$

$$\iff M(M^{i-1}-1) + B_i(M-1) < M[M(M^{i-2}-1) + B_{i-1}(M-1)]$$

$$\iff 0 < M^2(B_{i-1}-1) - M(B_{i-1}+B_i-1) + B_i, \quad i = 1, 2, \ldots, n.$$

Последнее неравенство истинно, т.к. $M^2 > M \cdot 2\sum\limits_{j=1}^{n} b_j$ и $(B_{i-1}-1) > 1$. То есть исходное неравенство верно. Аналогично можно доказать, что

$$\frac{w_{2(i-1)-1}}{p_{2(i-1)-1}} < \frac{w_{2i}}{p_{2i}}, \quad \frac{w_{2(i-1)}}{p_{2(i-1)}} < \frac{w_{2i-1}}{p_{2i-1}} \quad \text{и} \quad \frac{w_{2(i-1)-1}}{p_{2(i-1)-1}} < \frac{w_{2i-1}}{p_{2i-1}}$$

для любого $i = 2, 3, \ldots, n$.

Тогда одно из двух требований, V_{2n} и V_{2n-1} является **последним** запаздывающим требованием при любом оптимальном расписании (см. лемму 3.10). В дальнейшем будем рассматривать только расписания вида $(V_{n,1}, \pi_\alpha, V_{n,2})$, где $\{V_{n,1}, V_{n,2}\} = \{V_{2n-1}, V_{2n}\}$.

3) Аналогично доказательствам из пп. 1) и 2), можно доказать, что для каждого $i = n-1, n-2, \ldots, 1$, одно из двух требований, V_{2i} и V_{2i-1} запаздывает при любом оптимальном расписании. □

Теорема 3.10 *Задача* $1(nd)||\max\sum w_jT_j$ *NP-трудна в обычном смысле.*

Доказательство. Для расписания π вида

$$\pi = (V_{2n-1}, V_{2(n-1)-1}, \ldots, V_3, V_1, V_2, V_4, \ldots, V_{2(n-1)}, V_{2n})$$

имеем следующее значение целевой функции:

$$F(\pi) = \sum_{j=1}^{n} w_{2j}T_{2j}(\pi) = \sum_{j=1}^{n} w_{2j}p_{2j}.$$

Рассмотрим каноническое расписание вида

$$\pi' = (V_{n,1}, V_{n-1,1}, \ldots, V_{i,1}, \ldots, V_{1,1}, V_{1,2}, \ldots, V_{i,2}, \ldots, V_{n-1,2}, V_{n,2}).$$

Определим переменную

$$x_i = \begin{cases} 1, & \text{если } V_{i,2} = V_{2i-1}, \\ 0, & \text{если } V_{i,2} = V_{2i}. \end{cases}$$

Тогда мы имеем:

$$\begin{aligned}
F(\pi') &= F(\pi) + \sum_{i=1}^{n} x_i \left[(w_{2i-1} - w_{2i}) \cdot T_{V_{2i}}(\pi) - (p_{2i-1} - p_{2i}) \left(\sum_{j=1}^{i-1} w_{V_{j,2}} \right) \right] \\
&= F(\pi) + \sum_{i=1}^{n} x_i \left[b_i \cdot p_{2i} - b_i \cdot \left(\sum_{j=1}^{i-1} w_{V_{j,2}} \right) \right] \\
&= F(\pi) + \sum_{i=1}^{n} x_i \left[b_i \cdot p_{2i} - b_i \cdot \left(\sum_{j=1}^{i-1} w_{2j} + \sum_{j=1}^{i-1} x_j b_j \right) \right] \\
&= F(\pi) + \sum_{i=1}^{n} x_i b_i \left[p_{2i} - \frac{1}{2} \sum_{j=i+1}^{n} x_j b_j - \sum_{j=1}^{i-1} w_{2j} - \frac{1}{2} \sum_{j=1}^{i-1} x_j b_j \right] \\
&= F(\pi) + \frac{1}{2} \sum_{i=1}^{n} x_i b_i \left(\sum_{j \in N \setminus \{i\}} b_j - \sum_{j \in N \setminus \{i\}} x_j b_j \right) \\
&= F(\pi) + \frac{1}{2} \sum_{i=1}^{n} \sum_{j \in N \setminus \{i\}} x_i \cdot b_i \cdot b_j \cdot (1 - x_j).
\end{aligned}$$

В случае, когда существует подмножество $N' \subset N$ для примера ЗАДАЧИ РАЗБИЕНИЯ такое, что $\sum\limits_{i \in N'} b_i = \frac{1}{2} \sum\limits_{i \in N} b_i$, тогда при оптимальном каноническом расписании π^*, мы имеем

$$F(\pi^*) = F(\pi) + \frac{1}{2} A^2,$$

где $x_i = 1$, если $i \in N'$, и $x_j = 0$, если $i \in N \setminus N'$, т.к.

$$F(\pi^*) = F(\pi) + \frac{1}{2} \sum_{i \in N'} \sum_{j \in N \setminus N'} b_i \cdot b_j = F(\pi) + \frac{1}{2} A \cdot A.$$

Если ответ в примере ЗАДАЧИ РАЗБИЕНИЯ "НЕТ", тогда

$$\begin{aligned} F(\pi^*) &= F(\pi) + \frac{1}{2} \sum_{i=1}^{n} \sum_{j \in N \setminus \{i\}} x_i \cdot b_i \cdot b_j \cdot (1 - x_j) \\ &= F(\pi) + \frac{1}{2}(A - y)(A + y) \\ &= F(\pi) + \frac{1}{2} A^2 - \frac{1}{2} y^2, \end{aligned}$$

где $y > 0$.

Максимальное значение целевой функции задачи $1(nd)||\max \sum w_j T_j$ достигается тогда и только тогда, когда ответ в примере ЗАДАЧИ РАЗБИЕНИЯ "ДА".

□

Для задачи $1(nd)||\max \sum w_j T_j$ существует псевдополиномиальный алгоритм решения, поэтому задача NP-трудна в обычном смысле (не в сильном смысле).

3.6.2 Псевдополиномиальный алгоритм решения задачи $1(nd)||\max \sum T_j$

Алгоритм 9 основан на лемме 3.11, на том факте, что существует оптимальное расписание вида $\pi = (G, H)$, где все требования $j \in H$ запаздывают, а все требования $i \in G$ не запаздывают. Все требования из множества G обслуживаются в порядке неубывания продолжительности обслуживания (SPT), а все требования из множества H обслуживаются в порядке невозрастания продолжительности обслуживания (LPT).

Теорема 3.11 *Алгоритм 9 строит оптимальное расписание для задачи* $1(nd)||\max \sum T_j$ *за* $O(n \sum p_j)$ *операций.*

Доказательство теоремы аналогично доказательству теоремы 3.5.

Algorithm 9 Алгоритм решения задачи $1(nd)||\max\sum T_j$.

Перенумеруем требования согласно правилу: $p_1 \geq p_2 \geq \cdots \geq p_n$. Если $p_i = p_{i+1}$, тогда $d_i \geq d_{i+1}$;
for $t := 0$ to $\sum_{i=2}^{n} p_i$ **do**
    $\pi_1(t) := (1)$;
    $f_1(t) := \max\{0, p_1 + t - d_1\}$;
end for
for $l := 2$ to n **do**
    for $t := 0$ to $\sum_{i=l+1}^{n} p_i$ **do**
        $\pi^1 := (j, \pi_{j-1}(t + p_j))$, $\pi^2 := (\pi_{j-1}(t), j)$;
        $\Phi^1(t) := \max\{0, p_l + t - d_l\} + f_{l-1}(t + p_l)$;
        $\Phi^2(t) := f_{l-1}(t) + \max\{0, \sum_{j=1}^{l} p_j + t - d_l\}$;
        if $\Phi^1(t) < \Phi^2(t)$ **then**
            $f_l(t) := \Phi^2(t)$;
            $\pi_j(t) := \pi^2$;
        else
            $f_l(t) := \Phi^1(t)$;
            $\pi_j(t) := \pi^1$;
        end if
    end for
end for

$\pi_n(0)$ – оптимальное расписание, а $f_n(0)$ – соответствующее оптмальное значение целевой функции (максимальное суммарное запаздывание).

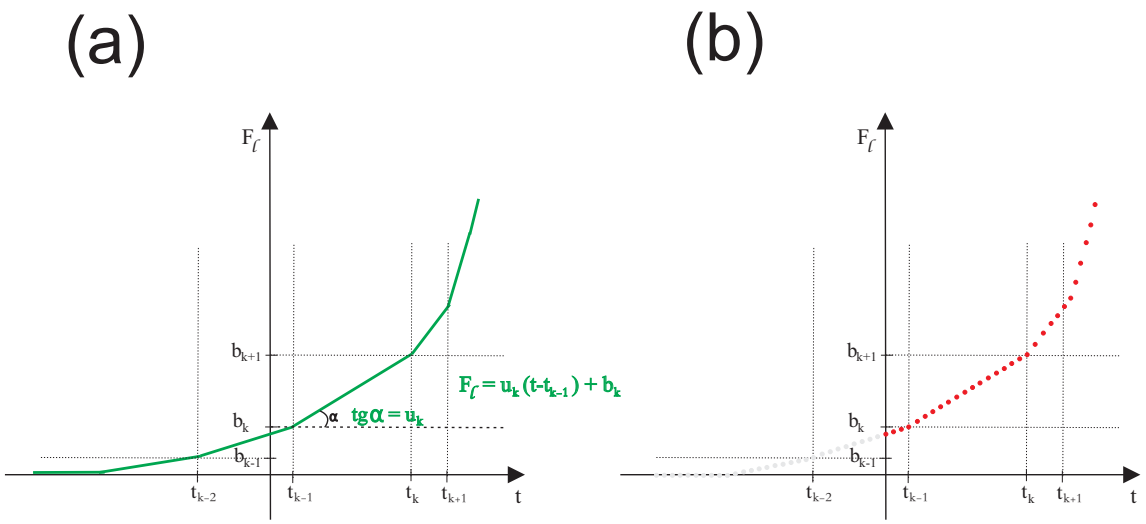

Рис. 3.2: Функция $f_l(t)$ в алгоритме 9 и в алгоритме GrA

3.6.3 Графический алгоритм решения задачи $1(nd)||\max\sum T_j$

В этом параграфе представлен Графический Алгоритм (GrA) решения задачи $1(nd)||\max\sum T_j$, основанный на уже известной нам идее модификации алгоритма динамического программирования. GrA представляет собой модификацию алгоритма 9, в которой функция $f_l(t)$ определена для любого значения $t \in (-\infty, +\infty)$ (не только для целых t), но мы вычисляем ее значения только в *точках излома*. Как будет показано далее, количество таких точек полиномиально. В этом параграфе помимо самого алгоритма GrA проиллюстрирована работа алгоритма на конкретном примере.

На каждом шаге алгоритма GrA, мы сохраняем функцию $f_l(t)$ в табличном виде, как представлено в таб. 3.5.

Таблица 3.5: Функция $f_l(t)$

t	$(-\infty, t_1]$	$(t_1, t_2]$	...	$(t_w, +\infty)$
b	$b_1 = 0$	b_2	...	b_{w+1}
кол-во запазд.треб-ий u	$u_1 = 0$	u_2	...	u_{w+1}
частич.оптим.расписание	π_1^l	π_2^l	...	π_{w+1}^l

Записи в таблице означают следующее. Для каждого значения $t \in (t_{k-1}, t_k]$, оптимальным будет частичное расписание π_k^l, в котором u_k запаздывающих требований, и которому соответствует значение целевой функции

$f_l(t) = u_k \cdot (t - t_{k-1}) + b_k$ (см. рис. 3.2). Функция $f_l(t)$ определена не только для целых t, но и для вещественных t.

Для упрощения описания алгоритма GrA мы рассматриваем всю ось t, т.е. $t \in (-\infty, +\infty)$. В теореме 3.12 показано, что эта таблица соответствует непрерывной, кусочно-линейной выпуклой функции $f_l(t)$. Точки $t_1, t_2, \ldots, t_w$ называются *точками излома*. Только в них происходит изменение (увеличение) количества запаздывающих требований с u_{k-1} на u_k (это означает, что меняется наклон кусочно-линейной функции). Заметим, что точки t_k могут быть нецелочисленными. Для описания каждого линейного сегмента функции, мы храним его наклон u_k и значение функции b_k в точке $t = t_{k-1}$.

В GrA функция $f_l(t)$ соответствует тем же рекурсивным уравнениям Беллмана, что и функция $f_l(t)$ в алгоритме 9, т.е. для каждого $t \in Z \bigcap [0, \sum_{j=2}^{n} p_j]$, $f_l(t)$ имеет тоже значение, что и в алгоритме 9 (см. рис.3.2), но теперь функция определена на всем интервале $t \in (-\infty, +\infty)$. В результате, "состояния" t, соответствующие одному и тому же оптимальному частичному расписанию, сгруппированы в интервалы. На рис. 3.2 (a), показана функция $f_l(t)$ из GrA, а на рис. 3.2 (b) – функция $f_l(t)$ из алгоритма 9.

Графический алгоритм GrA

Шаг 1. Перенумеруем требования согласно правилу: $p_1 \geq p_2 \geq \cdots \geq p_n$. Если $p_i = p_{i+1}$, тогда $d_i \geq d_{i+1}$;

Шаг 2. $l := 1$, $\pi_1^1(t) := (1)$, функция $f_1(t) := \max\{0, p_1 + t - d_1\}$ определена для всех t. Функция $f_1(t)$ задается таблицей 3.6.

Таблица 3.6: Функция $f_1(t)$

t	$(-\infty, d_1 - p_1]$	$(d_1 - p_1, +\infty)$
b	0	0
u	0	1
частич.оптим.расписание	(1)	(1)

Шаг 3. Пусть $l > 1$, а функция $f_{l-1}(t)$ вычислена (см. таб. 3.7).

Из функции $f_{l-1}(t)$ построим функцию $f_l(t)$. Промежуточные функции

Таблица 3.7: Функция $f_{l-1}(t)$

t	$(-\infty, t_1]$	$(t_1, t_2]$	...	$(t_w, +\infty)$
b	$b_1 = 0$	b_2	...	b_{w+1}
u	$u_1 = 0$	u_2	...	u_{w+1}
частич.оптим.расписание	π_1^{l-1}	π_2^{l-1}	...	π_{w+1}^{l-1}

$\Phi^1(t)$ и $\Phi^2(t)$ также хранятся в виде таб. 3.5.

Шаг 3.1. Функция $\Phi^1(t)$ строится из функции $f_{l-1}(t)$ при помощи следующих операций. Мы сдвигаем график функции $f_{l-1}(t)$ влево на p_l единиц и в таблицу, соответствующую "сдвинутой" функции $f_{l-1}(t)$, добавляем новую колонку, соответствующую новой точке излома $t' = d_l - p_l$. Если $t_k - p_l < t' < t_{k+1} - p_l,\ k+1 \leq w$, тогда в таблицу $\Phi^1(t)$ мы добавляем два интервала: $(t_k - p_l, t']$ и $(t', t_{k+1} - p_l]$. Далее мы увеличиваем все значения $u_{k+1}, u_{k+2}, \ldots, u_{w+1}$ на 1, т.е. количество запаздывающих требований (и наклон соотв. линейного сегмента графика) возрастает. Соответствующее частичное расписание π^1 строится добавлением требования l в начало предыдущего частичного расписания. Таким образом получена таб. 3.7.1, соответствующая функции $\Phi^1(t)$.

Шаг 3.2. Функция $\Phi^2(t)$ строится из функции $f_{l-1}(t)$ при помощи следующих операций. В таблицу, соответствующую функции $f_{l-1}(t)$, добавляем новую колонку, соответствующую новой точке излома $t'' = d_l - \sum_{i=1}^{l} p_i$. Если $t_h < t'' < t_{h+1},\ h+1 \leq w$, тогда в таблицу $\Phi^2(t)$ мы добавляем два интервала: $(t_h, t'']$ и $(t'', t_{h+1}]$. Далее мы увеличиваем все значения $u_{h+1}, u_{h+2}, \ldots, u_{w+1}$ на 1, т.е. количество запаздывающих требований (и наклон соотв. линейного сегмента графика) возрастает. Соответствующее частичное расписание π^2 строится добавлением требования l в конец предыдущего частичного расписания. Таким образом получена таблица 3.7.2, соответствующая функции $\Phi^2(t)$.

Шаг 3.3. Теперь мы строим таблицу, соответствующую функции

$$f_l(t) = \max\{\Phi^1(t), \Phi^2(t)\}.$$

В интервалах, образованных точками из двух таблиц 3.7.1 и 3.7.2 мы сравниваем значения функций $\Phi^1(t)$ и $\Phi^2(t)$ и ищем точки пересечения их графиков. Этот шаг требует порядка $O(w)$ операций.

Более подробно – конструируем список $t_1, t_2, \ldots, t_s,\ t_1 < t_2 < \ldots, t_s,$

Таблица 3.7.1. Функция $\Phi^1(t)$

t	$(-\infty, t_1 - p_l]$	$(t_1 - p_l, t_2 - p_l]$	$\dots$	$(t_k - p_l, t']$	$(t', t_{k+1} - p_l]$	$(t_{k+1} - p_l, t_{k+2} - p_l]$	$\dots$	$(t_w - p_l, +\infty)$
b	$b_1 = 0$	b_2	$\dots$	b_{k+1}	b'	b'_{k+2}	$\dots$	b'_{w+1}
u	$u_1 = 0$	u_2	$\dots$	u_{k+1}	$u_{k+1} + 1$	$u_{k+2} + 1$	$\dots$	$u_{w+1} + 1$
частич. оптим. распи-сание π^1	(l, π_1^{l-1})	(l, π_2^{l-1})	$\dots$	(l, π_{k+1}^{l-1})	(l, π_{k+1}^{l-1})	(l, π_{k+2}^{l-1})	$\dots$	(l, π_{w+1})

$b' = b_{k+1} + (t' - (t_k - p_l)) \cdot u_{k+1}$, $b'_{k+2} = b' + ((t_{k+1} - p_l) - t') \cdot (u_{k+1} + 1)$, $\dots$, $b'_{m+1} = b_m{}' + ((t_m - t_{m-1}) \cdot (u_m + 1)$

Таблица 3.7.2. Функция $\Phi^2(t)$

t	$(-\infty, t_1]$	$(t_1, t_2]$	$\dots$	$(t_h, t'']$	$(t'', t_{h+1}]$	$(t_{h+1}, t_{h+2}]$	$\dots$	$(t_w, +\infty)$
b	$b_1 = 0$	b_2	$\dots$	b_{h+1}	b''	b''_{h+2}	$\dots$	b''_{w+1}
u	$u_1 = 0$	u_2	$\dots$	u_{h+1}	$u_{h+1} + 1$	$u_{h+2} + 1$	$\dots$	$u_{w+1} + 1$
частич. оптим. распи-сание π^2	(π_1^{l-1}, l)	(π_2^{l-1}, l)	$\dots$	(π_{h+1}^{l-1}, l)	(π_{h+1}^{l-1}, l)	(π_{h+2}^{l-1}, l)	$\dots$	(π_{w+1}^{l-1}, l)

$b'' = b_{h+1} + (t'' - t_h) \cdot u_{h+1}$, $b''_{h+2} = b'' + (t_{h+1} - t'') \cdot (u_{h+1} + 1)$, $\dots$, $b''_{w+1} = b_w{}'' + (t_w - t_{w-1}) \cdot (u_w + 1)$

Таблица 3.7.3. Функция $f_l(t)$

t	$(-\infty, t_1]$	$(t_1, t_2]$	$\dots$	$(t_{k-1}, t_k]$	$(t_k, t_{k+1}]$	$\dots$	$(t_w, +\infty)$
b	$b_1 = 0$	b_2	$\dots$	b_k	b_{k+1}	$\dots$	b_{w+1}
u	$u_1 = 0$	u_2	$\dots$	u_k	u_{k+1}	$\dots$	u_{w+1}
частич. оптим. расписание	π_1^l	π_2^l	$\dots$	π_k^l	π_{k+1}^l	$\dots$	π_{w+1}^l

всех точек t из обоих таблиц $\Phi^1(t)$ и $\Phi^2(t)$, которые являются границами интервалов из этих таблиц. Далее рассматриваем каждый интервал $(t_i, t_{i+1}]$, $i = 1, 2, \ldots, s-1$, и сравниваем функции $\Phi^1(t)$ и $\Phi^2(t)$ на этом интервале. Пусть интервал $(t_i, t_{i+1}]$ включен в интервал $(t_{x-1}, t_x]$ из таб. $\Phi^1(t)$ и в интервал $(t_{y-1}, t_y]$ из таблицы $\Phi^2(t)$. Тогда $\Phi^1(t) = t \cdot u_x + b_x + (t_i - t_{x-1}) \cdot u_x$ и $\Phi^2(t) = t \cdot u_y + b_y + (t_i - t_{y-1}) \cdot u_y$. Выбираем колонку из двух таблиц, соответствующую максимуму из двух функций и вставляем эту колонку в таблицу $f_l(t)$. Если на данном интервале существует точка пересечения t''' функций $\Phi^1(t)$ и $\Phi^2(t)$, то мы вставляем две колонки, соответствующие интервалам $(t_i, t''']$ и $(t''', t_{i+1}]$.

Если в итоговой таблице, соответствующей функции $f_l(t)$, встречается ситуация как в таб. 3.8, мы удаляем колонку $(t_k, t_{k+1}]$ и объединяем оба интервала, т.е. принимаем $t_k := t_{k+1}$.

Таблица 3.8: Удаление колонки

t	$\ldots$	$(t_{k-1}, t_k]$	$(t_k, t_{k+1}]$	$\ldots$
b	$\ldots$	$\ldots$	$\ldots$	$\ldots$
u	$\ldots$	u_k	$u_{k+1} = u_k$	$\ldots$
частич.оптим.расписание π_l	$\ldots$	$\ldots$	$\ldots$	$\ldots$

В теореме 3.12 доказано, что для каждого l, число колонок в таблице $f_l(t)$ не больше $l + 1 \leq n + 1$.

Шаг 3.3. Если $l = n$, тогда переходим к шагу 4 иначе $l := l + 1$ и переходим к шагу 3.

Шаг 4. В таблице $f_n(t)$, мы находим колонку $(t_k, t_{k+1}]$, где $t_k < 0 \leq t_{k+1}$. Тогда мы имеем оптимальное расписание $\pi^* = \pi_{k+1}$ и оптимальное значение целевой функции $F(\pi^*) = b_{k+1} + (0 - t_k) \cdot u_{k+1}$.

Очевидно, что GrA строит то же решение, что и алгоритм 9. В то время, как в алгоритме 9 рассматриваются все целочисленные точки $t \in [0, \sum_{j=2}^{n} p_j]$, в GrA рассматриваются только точки излома, определенные на всей оси t, такие, что все значения $f_l(t)$, $l \in \{1, 2, \ldots, n\}$, $t \in Z \cap [0, \sum_{j=l+1}^{n} p_j]$ соответствует тем же значениям, определенным в алгоритме 9. Поэтому алгоритм GrA строит точное решение.

Теорема 3.12 *Трудоёмкость алгоритма GrA составляет $O(n^2)$ операций.*

Доказательство. Покажем, что все функции $f_l(t)$, $l = 1, 2, \ldots, n$, являются непрерывными кусочно-линейными **выпуклыми** функциями.

Очевидно, что функция $f_1(t)$ – непрерывная кусочно-линейная выпуклая функция с одной точкой излома. Согласно шагу 3 алгоритма обе функции $\Phi^1(t)$ и $\Phi^2(t)$ также являются непрерывными, кусочно-линейными и выпуклыми. Поэтому функция $f_2(t) = \max\{\Phi^1(t), \Phi^2(t)\}$ также непрерывная, кусочно-линейная и выпуклая.

Продолжая данные рассуждения для других l, получим, что все функции $f_l(t), l = 1, 2, \ldots, n$, обладают этим характеристиками, т.е. являются непрерывными, кусочно-линейными и **выпуклыми**.

Теперь предположим, что на шаге 3 алгоритма мы получили таблицу 3.7.3. для некоторой функции $f_l(t)$. Покажем, что выполняется неравенство $u_1 < u_2 < \cdots < u_k < u_{k+1} < \cdots < u_{w+1}$. Предположим, что $u_k > u_{k+1}$. Тогда для каждого $t \in (t_k, t_{k+1}]$, где t – время начала обслуживания требований из частичного расписания (π_k или π_{k+1}), имеем $F(\pi_k) > F(\pi_{k+1})$, т.к. выполняется $u_k > u_{k+1}$. То есть получили противоречие[1]. Напомним, что функция $f_l(t)$ является непрерывной кусочно-линейной выпуклой функцией, и $b_{k+1} = b_k + (t_k - t_{k-1}) \cdot u_k$.

Согласно алгоритму GrA, если $u_k = u_{k+1}$, тогда мы удаляем колонку, соответствующую точке излома u_{k+1} и объединяем оба интервала.

Мы имеем $u_1 < u_2 < \cdots < u_k < u_{k+1} < \cdots < u_{w+1}$, где u_i, $i = 1, 2, \ldots, w+1$, – количество запаздывающих требований. Тогда $w + 1 \leq l+1 \leq n+1$. Как следствие мы имеем не более l точек излома, что значит, не более $l + 1 \leq n + 1$ колонок для каждой функции $f_l(t)$, $l = 2, \ldots, n$.

Поэтому трудоёмкость Шага 3 алгоритма составляет $O(n)$ операций для каждого $l = 2, \ldots, n$. Значит трудоёмкость алгоритма GrA составляет $O(n^2)$ операций. □

Стоит заметить, что уже для задачи $1(nd)||\max\sum w_jT_j$ аналогичный

[1]Заметим, что данный факт, что $u_1 < \cdots < u_w$, можно доказать иначе. Функции $\Phi^1(t)$ и $\Phi^2(t)$ выпуклы, а следовательно функция $f_l(t) = \max\{\Phi^1(t), \Phi^2(t)\}$ также выпукла. Значения u_k соответствуют $tg\alpha$, где α угол между осью t и линейным сегментом функции $f_l(t)$.

алгоритм имеет трудоёмкость $O(\min\{2^n, n \cdot \min\{d_{\max}, \sum w_j\}\})$ операций.

Иллюстрация работы Графического Алгоритма на примере

Рассмотрим пример задачи $1||\max \sum T_j$, входные данные которого представлены в таблице 3.9.

Таблица 3.9: Входные данные

j	1	2	3	4
p_j	30	22	12	5
d_j	32	35	38	40

Опишем таблицы, сохраняемые на каждой итерации l, алгоритма GrA.

Пусть $l = 1$. Сохраняем таблицу 3.10.

Таблица 3.10: Функция $f_1(t)$

t	$(-\infty, 2]$	$(2, +\infty)$
$f_1(t)$	0	0
u	0	1
π_1	(1)	(1)

График функции $f_1(t)$ представлен на рис. 3.3 (а).

Пусть $l = 2$. Получаем следующие результаты. Вычисляем новую точку излома $t' = d_2 - p_2 = 13$, и таб.3.11 для функции $\Phi^1(t)$.

Таблица 3.11: Функция $\Phi^1(t)$

t	$(-\infty, -20]$	$(-20, 13]$	$(13, +\infty)$
$\Phi^1(t)$	0	0	33
u	0	1	2
π^1	(2,1)	(2,1)	(2,1)

Вычисляем новую точку излома $t' = d_2 - (p_1 + p_2) = -17$, и таб. 3.12 для функции $\Phi^2(t)$.

Теперь проверим каждый из интервалов $(-\infty, -20]$, $(-20, -17]$, $(-17, 2]$, $(2, 13]$, $(13, +\infty)$, и проверим, не пересекаются ли функции $\Phi^1(t)$ и $\Phi^2(t)$ на этих интервалах. Для интервала $t \in (2, 13]$ из неравенства $\Phi^1(t) = (t + 20) \cdot 1 + 0 = \Phi^2(t) = (t - 2) \cdot 2 + 19$ получаем точку пересечения

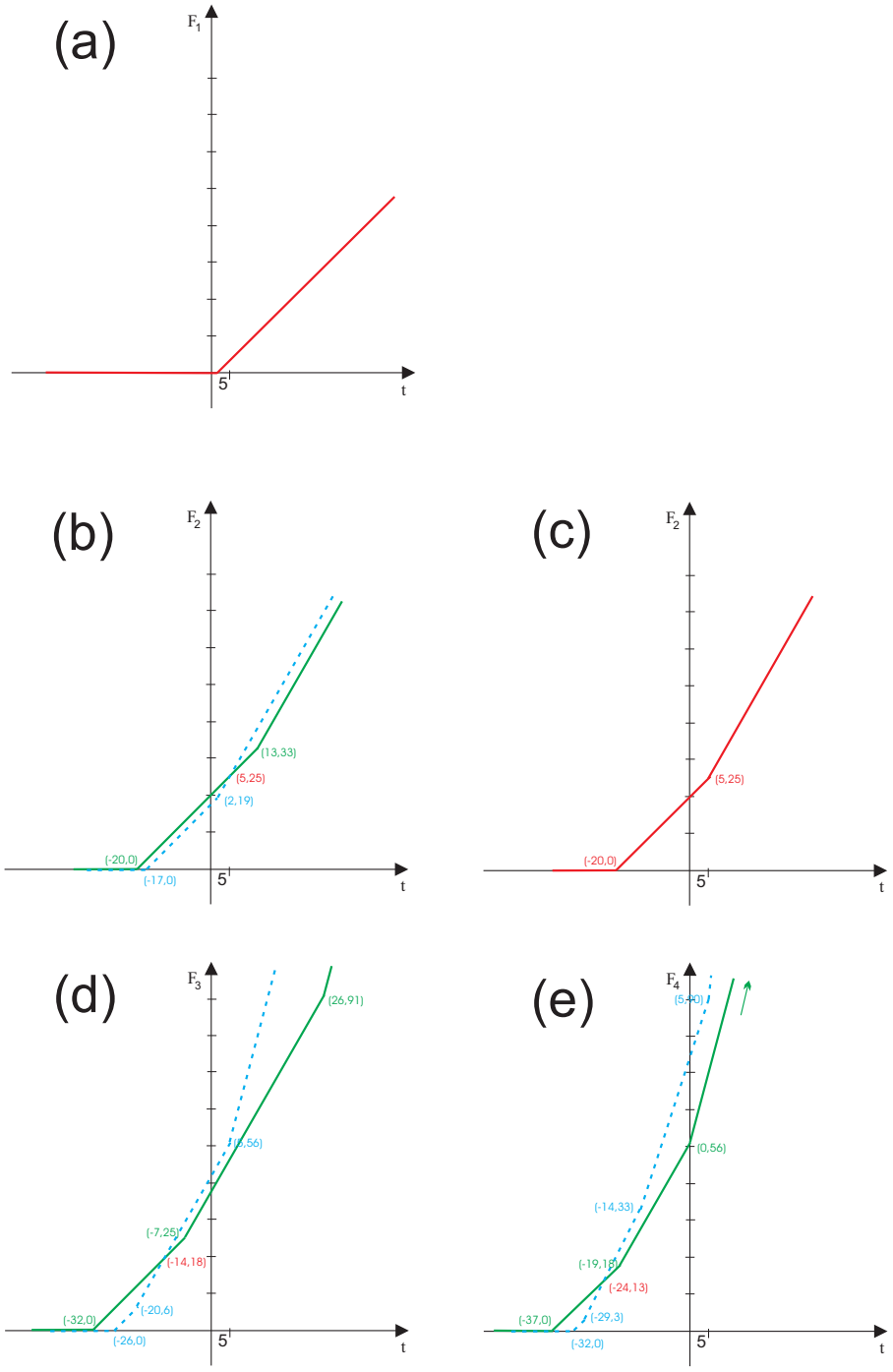

Рис. 3.3: Пример

$(5, 25)$. Комбинируя результаты из таблиц $\Phi^1(t)$ и $\Phi^2(t)$, получаем таблицу 3.13 функции $f_2(t)$. Заметим, что для $t \leq 5$, частичное расписание (2,1), соответствующее функции $\Phi^1(t)$ является оптимальным, в то время, как для $t > 5$ оптимальным является частичное расписание $(1, 2)$,

Таблица 3.12: Функция $\Phi^2(t)$

t	$(-\infty, -17]$	$(-17, 2]$	$(2, +\infty)$
$\Phi^2(t)$	0	0	19
u	0	1	2
π^2	(1,2)	(1,2)	(1,2)

соответствующее функции $\Phi^2(t)$.

Таблица 3.13: Функция $f_2(t)$

t	$(-\infty, -20]$	$(-20, 5]$	$(5, +\infty)$
$f_2(t)$	0	0	25
u	0	1	2
π_2	(2,1)	(2,1)	(1,2)

График функции $f_1(t)$ представлен на рис. 3.3 (c). Он получен из графиков, представленных на рис. 3.3 (b). Аналогично рассмотрим оставшиеся итерации $l = 3, 4$.

Пусть $l = 3$**.** Получаем следующие результаты.

Вычисляем новую точку излома $t' = d_3 - p_3 = 26$, и таб.3.14 для функции $\Phi^1(t)$.

Таблица 3.14: Функция $\Phi^1(t)$ для $l = 3$

t	$(-\infty, -32]$	$(-32, -7]$	$(-7, 26]$	$(26, +\infty)$
$\Phi^1(t)$	0	0	25	91
u	0	1	2	3
π^1	(3,2,1)	(3,2,1)	(3,1,2)	(3,1,2)

Вычисляем новую точку излома $t' = d_3 - (p_1 + p_2 + p_3) = -26$, и таб. 3.15 для функции $\Phi^2(t)$.

Таблица 3.15: Функция $\Phi^2(t)$ для $l = 3$

t	$(-\infty, -26]$	$(-26, -20]$	$(-20, 5]$	$(5, +\infty)$
$\Phi^2(t)$	0	0	6	56
u	0	1	2	3
π^2	(2,1,3)	(2,1,3)	(2,1,3)	(1,2,3)

Точка пересечения двух графиков функций $\Phi^1(t)$ и $\Phi^2(t)$ – $(-14, 18)$. Функция $f_3(t)$ представлена в таб. 3.16. Ее график показан на рис. 3.3 (d).

Пусть $l = 4$**.** Получаем следующие результаты.

Таблица 3.16: Функция $f_3(t)$

t	$(-\infty,-32]$	$(-32,-14]$	$(-14,5]$	$(5,+\infty)$
$f_3(t)$	0	0	18	56
u	0	1	2	3
π_3	(3,2,1)	(3,2,1)	(2,1,3)	(1,2,3)

Вычисляем новую точку излома $t' = d_4 - p_4 = 35$ и таб.3.17 для функции $\Phi^1(t)$.

Таблица 3.17: Функция $\Phi^1(t)$ для $l = 4$

t	$(-\infty,-37]$	$(-37,-19]$	$(-19,0]$	$(0,35]$	$(35,+\infty)$
$\Phi^1(t)$	0	0	18	56	161
u	0	1	2	3	4
π^1	(4,3,2,1)	(4,3,2,1)	(4,2,1,3)	(4,1,2,3)	(4,1,2,3)

Вычисляем новую точку излома $t' = d_4 - (p_1 + p_2 + p_3 + p_4) = -29$ и таб. 3.18 для функции $\Phi^2(t)$.

Таблица 3.18: Функция $\Phi^2(t)$ для $l = 4$

t	$(-\infty,-32]$	$(-32,-29]$	$(-29,-14]$	$(-14,5]$	$(5,+\infty)$
$\Phi^2(t)$	0	0	3	33	90
u	0	1	2	3	4
π^2	(3,2,1,4)	(3,2,1,4)	(3,2,1,4)	(2,1,3,4)	(1,2,3,4)

Точка пересечения двух графиков функций $\Phi^1(t)$ и $\Phi^2(t)$ есть точка на плоскости $(-24, 13)$. Функция $f_4(t)$ представлена в таб. 3.19. Ее график показан на рис. 3.3 (e).

Таблица 3.19: Функция $f_4(t)$

t	$(-\infty,-37]$	$(-37,-24]$	$(-24,-14]$	$(-14,5]$	$(5,+\infty)$
$f_4(t)$	0	0	13	33	90
u	0	1	2	3	4
π_4	(4,3,2,1)	(4,3,2,1)	(3,2,1,4)	(2,1,3,4)	(1,2,3,4)

Мы получили оптимальное значение искомой целевой функции $f_4(0) = 33 + 14 \cdot 3 = 75$, которому соответствует оптимальное расписание $\pi_4(0) = (2, 1, 3, 4)$.

3.7 Задачи с одним невозобновимым ресурсом

В данном параграфе рассматриваются одноприборные задачи с невозобновимым ресурсом. Этот ресурс необходим для обслуживания требований, причем потребности в ресурсе у требований может отличаться. В качестве такого ресурса могут выступать, например, деньги, топливо, прочие расходные невозобновимые материалы. Так как подобные задачи часто возникают в бюджетных организациях, для которых ресурсом являются деньги, эти задачи также называются *финансовыми*.

Постановка этих задач совпадает с классической постановкой одноприборных задач. Дополнительно заданы невозобновимый ресурс G (деньги, топливо и т.п.) и моменты времени поступления ресурса $\{t_0, t_1, \ldots, t_y\}$, $t_0 = 0$, $t_0 < t_1 < \cdots < t_y$. В каждый момент времени t_i, $i = 0, 1, \ldots, y$, поступает $G(t_i) \geq 0$ единиц ресурса. Для каждого требования $j \in N$, задано потребление ресурса $g_j \geq 0$, которое происходит в момент начала обслуживания. Выполняется равенство

$$\sum_{j=1}^{n} g_j = \sum_{i=0}^{y} G(t_i).$$

Обозначим через S_j время начала обслуживания требования j. Расписание $S = (S_{j_1}, S_{j_2}, \ldots, S_{j_n})$ описывает в том числе порядок обслуживания требований: $\pi = (j_1, j_2, \ldots, j_n)$. Расписание $S = (S_{j_1}, S_{j_2}, \ldots, S_{j_n})$ является допустимым, если выполняется, помимо классических условий, следующее неравенство:

$$\sum_{k=1}^{i} g_{j_k} \leq \sum_{\forall l:\ t_l \leq S_{j_i}} G(t_l), \quad i = 1, 2, \ldots, n.$$

Перестановку π также называют расписанием, т.к. по этой перестановке можно вычислить расписание $S = (S_{j_1}, S_{j_2}, \ldots, S_{j_n})$ за время $O(n)$.

Такие одноприборные задачи обозначают $1|NR, ...|...$, где NR обозначает присутствие в условии задачи невозобновимого ресурса (non-renewable).

В таб. 3.20 представлены некоторые сведения о трудоёмкости одноприборных задач с невозобновимым ресурсом.

Таблица 3.20: Трудоёмкость задач

Цел. функц.	**Частный случай**	**Трудоёмкость**
$C_{\max}$		NP-трудна в сильном смысле. Сведение ЗАДАЧИ 3-РАЗБИЕНИЯ
$\sum U_j$		NP-трудна в сильном смысле. Сведение ЗАДАЧИ 3-РАЗБИЕНИЯ
$\sum C_j$		NP-трудна в сильном смысле. Сведение $1\|r_j\|\sum C_j$
$L_{\max}$		NP-трудна в сильном смысле. Сведение ЗАДАЧИ 3-РАЗБИЕНИЯ
$\sum T_j$	$d_j = d$	NP-трудна в сильном смысле. Сведение ЗАДАЧИ 3-РАЗБИЕНИЯ
$\sum T_j$	$g_j = g$	NP-трудна (задача $1\|\|\sum T_j$ является частным случаем)
$\sum T_j$	$p_j = p$	NP-трудна. Сведение ЗАДАЧИ РАЗБИЕНИЯ.
$\sum T_j$	$d_j = d,\ g_j = g$	NP-трудна. Сведение ЗАДАЧИ РАЗБИЕНИЯ.
$\sum T_j$	$p_j = p$, $g_1 \le g_2 \le \cdots \le g_n$, $d_1 \le d_2 \le \cdots \le d_n$	полиномиально разрешима, оптимальное расписание: $\pi^* = (1, 2, \ldots, n)$

В качестве иллюстрации приведем несложное доказательство NP-трудности некоторых из перечисленных задач.

ЗАДАЧА 3-РАЗБИЕНИЯ. Задано множество $N = \{b_1, b_2, \ldots, b_n\}$ чисел, где $n = 3m$, $\sum\limits_{i=1}^{n} b_j = mB$ и $\frac{B}{4} \leq b_j \leq \frac{B}{2}$, $j = 1, 2, \ldots, n$. Необходимо ответить на вопрос, существует ли такое разбиение множества N на m подмножеств $N_1, N_2, \ldots, N_m$, каждое из которых содержит ровно три числа, и суммы чисел которых равны, т.е.

$$\sum_{b_j \in N_1} b_j = \sum_{b_j \in N_2} b_j = \cdots = \sum_{b_j \in N_m} b_j = B?$$

Известно, что ЗАДАЧА 3-РАЗБИЕНИЯ является NP-трудной в сильном смысле.

Теорема 3.13 *Задачи* $1|NR|C_{\max}$, $1|NR, d_j = d|\sum T_j$, $1|NR|\sum U_j$ *и* $1|NR|L_{\max}$ *являются NP-трудными в сильном смысле.*

Доказательство. Доказательство проведем сведением ЗАДАЧИ 3-РАЗБИЕНИЯ к частному случаю задач.

Рассмотрим задачу $1|NR|C_{\max}$. Дан произвольный пример ЗАДАЧИ 3-РАЗБИЕНИЯ, построим пример задачи $1|NR|C_{\max}$ следующим образом. В пример зададим n требований с параметрами $g_j = p_j = b_j$, $j = 1, 2, \ldots, n$. Времена поступления ресурса определены как

$$\{t_0, t_1, \ldots, t_{m-1}\} = \{0, B, 2B, \ldots, (m-1)B\}$$

и $G(t_0) = G(t_1) = \cdots = G(t_{m-1}) = B$. Очевидно, что $C_{\max} = mB$ тогда и только тогда, когда ответ в примере ЗАДАЧИ 3-РАЗБИЕНИЯ "ДА". В этом случае в оптимальном расписании нет простоев прибора.

Для задач $1|NR, d_j = d|\sum T_j$, $1|NR|\sum U_j$ и $1|NR|L_{\max}$ дополнительно определим $d_j = d = mB$ для $j = 1, 2, \ldots, n$. Тогда $\sum T_j = 0$ выполняется тогда и только тогда, когда ответ в примере ЗАДАЧИ 3-РАЗБИЕНИЯ "ДА". Аналогично $\sum U_j = 0$ и $L_{\max} = 0$ выполняются тогда и только тогда, когда ответ в примере ЗАДАЧИ 3-РАЗБИЕНИЯ "ДА". $\square$

Библиографическая справка

Задача $1||\sum U_j$ полиномиально разрешима за время $O(n \log n)$ при помощи алгоритма Мура [87].

Алгоритмы решения задачи $1||\sum w_j U_j$ и частного случая задачи $1||GT_j$ представлены в работе [77].

Задача $1||\sum T_j$ является NP-трудной в обычном смысле [5, 53]. Е.Л. Лаулер [78] предложил псевдополиномиальный алгоритм решения общего случая проблемы трудоёмкости $O(n^4 \sum p_j)$. В. Шварц и др. [101, 102] построили алгоритмы решения проблемы, которые были протестированы для примеров $n < 500$ (тестовые примеры С.Н. Поттса и Л.Н. ван Вассенхова [89]). Исследование приближенных алгоритмов решения проблемы было проведено в работе [49], где построены примеры, на которых известные приближенные алгоритмы находят решение с относительной погрешностью порядка размерности примера n.

Задачи с обратными критериями максимизации рассматриваются в работах [31, 32, 56, 59, 61]. Задачи с невозобновимым ресурсом в работе [57].

Глава 4

Задачи цеха (Shop problems)

4.1 Задачи $F||C_{\max}$

В этом разделе главы рассматривается задача Flow-Shop, в которой необходимо минимизировать значение $C_{\max} = \max\limits_{j \in N} C_j$. Напомним, что для данной задачи каждое требование $j,\ j = 1, 2, \ldots, n$, состоит из одних и тех же операций, т.е. $O_{j1} \to \cdots \to O_{jm}, \forall j \in N$, причем для каждой операции с номером $i,\ i = 1, 2, \ldots, m$, задан прибор, выполняющий i эту операцию. Операции каждого требования j выполняются в заданной последовательности $O_{j1} \to \cdots \to O_{jm}$, причем выполнение операции $k,\ k = 2, 3, \ldots, m$, может начаться не раньше окончания выполнения операции $k-1$ для этого же требования.

Расписание для каждого прибора задается вектором – порядком обслуживания на данном приборе операций, относящихся к разным требованиям. На рис. 4.1 представлено два допустимых расписания для одного и того же примера, где $n = 2$ и $m = 4$. При этих расписаниях $C_{\max} = 11$. Оптимальное расписание представлено на рис. 4.2, где $C_{\max} = 10$.

Теорема 4.1 *Для задачи $Fm||C_{\max}$ существует оптимальное расписание, обладающее свойствами:*

- *порядок ослуживания требований (вектор) для первых двух приборов совпадает;*
- *порядок ослуживания требований (вектор) для последних двух приборов совпадает;*

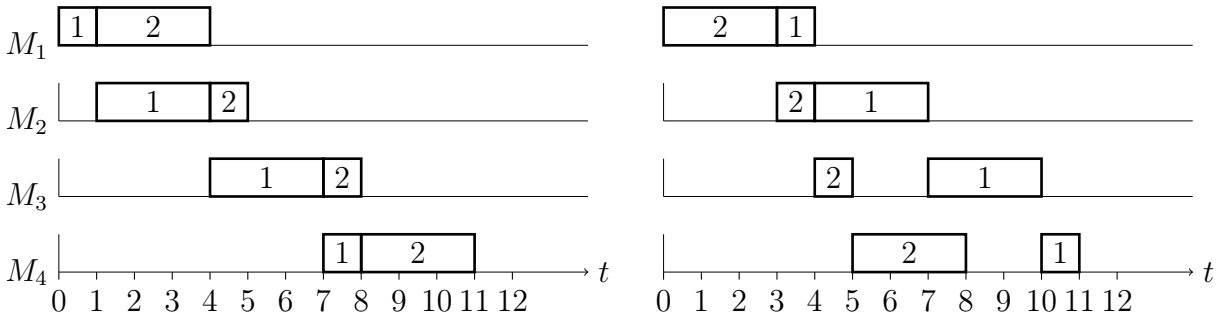

Рис. 4.1: Расписания для двух требований

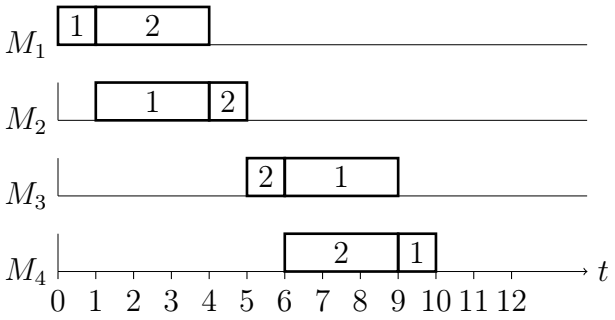

Рис. 4.2: Оптимальное расписание для двух требований

На основании этой теоремы можно сделать вывод, что если $m \leq 3$, то существует порядок обслуживания (вектор), оптимальный (одинаковый) для всех трех приборов.

Далее представлен алгоритм Джонсона решения задачи $F2||C_{\max}$, полученный в 1954-м году. Идея алгоритма заключается в следующем. Последовательно мы конструируем начало π_1 и конец π_2 вектора $\pi = (\pi_1, \pi_2)$, задающего порядок обслуживания. На каждом шаге мы рассматриваем операцию O_{ji} с наименьшим значением p_{ji} среди рассматриваемых операций. Если $i = 1$, то в конец частичного расписания π_1 мы добавляем требование j, т.е. $\pi_1 = (\pi_1, j)$, иначе требование добавляется в начало другого частичного расписания $\pi_2 = (j, \pi_2)$. После этого требование j исключается из рассмотрения и мы повторяем итерацию до тех пор, пока не будут расставлены все требования.

Необходимо отметить, что алгоритм 10 имеет трудоёмкость $O(n \log n)$ операций, если пункт 3 алгоритма – сортировку продолжительности обслуживания требований – выполнить до основного цикла 2–10.

Algorithm 10 Алгоритм Джонсона решения задачи $F2||C_{\max}$.

1: $N' := N;\ \pi_1 := (),\ \pi_2 := ();$
2: **while** $N' \neq \emptyset$ **do**
3: Найдем $O_{j^*i^*}$ с минимальной продолжительностью $p_{j^*i^*} = \min\{p_{ji} | j \in N',\ i = 1, 2\}$;
4: **if** $i = 1$ **then**
5: $\pi_1 = (\pi_1, j)$;
6: **else**
7: $\pi_2 = (j, \pi_2)$;
8: **end if**
9: $N' := N' \setminus j$;
10: **end while**
11: $\pi = (\pi_1, \pi_2)$;

Продемонстрируем работу алгоритма на примере.

Таблица 4.1: Входные данные

j	1	2	3	4	5
p_{j1}	4	3	3	1	8
p_{j2}	8	3	4	4	7
Куда включаем требование j	π_1	π_2	π_1	π_1	π_2

Для данного примера оптимальный порядок (вектор) имеет вид $\pi = (4, 3, 1, 5, 2)$. На рис. 4.3 соответствующее расписание представлено графически.

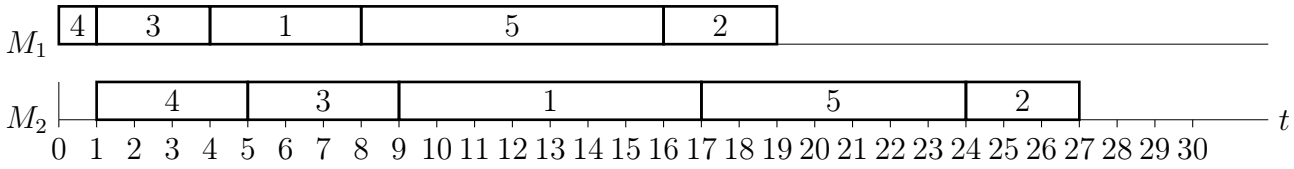

Рис. 4.3: Оптимальное расписание для примера

Библиографическая справка

Подробнее о многоприборных задачах можно прочитать в монографиях [25, 41]. Сводные таблицы с информацией о NP-трудности многоприборных задач и лучших известных алгоритмах их решения представлены на сайте
http://www.mathematik.uni-osnabrueck.de/research/OR/class/ .

В следующей главе приводятся некоторые сведения о задаче с параллельным идентичными приборами.

Глава 5

Построение расписания для проекта. Project Scheduling

В этой главе рассматриваются задачи построения расписания для проекта (Project Scheduling). Базовая задача этого раздела **TP** – "Минимизация времени выполнения проекта с учетом ограничения на ресурсы" (RCPSP). Ее постановка была приведена в первой главе.

Здесь будут представлены некоторые свойства задачи RCPSP, алгоритмы ее решения (в том числе алгоритм Ветвей и Границ, метаэвристический и эвристический алгоритмы), а также рассмотрены некоторые практические ее интерпретации и частные случаи.

Заметим, что в задаче RCPSP одновременно применяют оба термина, *требования* и *работы*, являющиеся эквивалентными.

Решение задачи RCPSP представляется в виде набора моментов начала обслуживания требований $S = (S_1, \ldots, S_n)$. Решение, удовлетворяющее ресурсным ограничениям и ограничениям предшествования, будем называть допустимым.

Структуру проекта, его требования и взаимосвязь между ними, можно представить в виде сетевого графика (см. первую главу), т.е. в виде *требования-в-вершинах* ориентированного графа $G = (V, E)$, где каждой вершине из $V = \{1, \ldots, n\}$ соответствует некоторое требование множества $N = \{1, \ldots, n\}$, а множество дуг $E = \{(i, j) | i, j \in V; i \to j\}$ соответствует ограничениям предшествования. Очевидно, что допустимое решение существует только, когда граф предшествования ацикличен.

Обычно в рассмотрение вводят два фиктивных требования 0 и $n+1$ с продолжительностями обслуживания $p_0 = p_{n+1} = 0$ и отношениями предшествования $0 \to j \to n+1$, $j = 1, \ldots, n$, $q_{0k} = q_{n+1k} = 0$, $k = 1, \ldots, K$.

Будем обозначать через UB верхнюю границу для оптимального значения целевой функции $C^*_{\max}$. К примеру, можем принять $UB = \sum\limits_{i:=1}^{n} p_i \geq C^*_{\max}$.

Для каждого требования $i \in N$ определим временно́е окно $[r_i, d_i]$, в котором требование i должно быть обслужено при любом оптимальном расписании S:

$$r_0 = 0;\ r_i = \max_{j|(j,i)\in E}\{r_j + p_j\},\ i = 1, \ldots, n+1,$$

$$d_{n+1} = UB;\ d_i = \min_{j|(i,j)\in E} d_j - p_j,\ i = n, n-1, \ldots, 0.$$

5.1 Практическая задача составления расписания проекта

На практике задача RCPSP обладает рядом отличий от математической модели, представленной в первой главе. Далее перечислены некоторые возможные отличия на примере практической задачи RPCPS, возникающей в строительстве.

- Работы могут **классифицироваться** по типу:
  - *Продолжительность.* Это классический тип работы. Важная ее характеристика – продолжительность выполнения;
  - *Контрольное событие.* К примеру – составление отчетности, разовый контроль. Такая работа имеет нулевую продолжительность;
  - *Гамак.* Работа выполняется строго между моментами времени окончания работ-предшественников и началом работ-последователей, т.е. ее продолжительность не фиксирована;

- **Продолжительность**. Задается в днях или часах. На практике редко планирование ведется в меньших единицах времени. Очевидно, что продолжительность может зависеть от интенсивности работы (т.е. от количества и качества привлеченных ресурсов);

- **Невозобновимые ресурсы**. Некоторые ограничения на допустимые расписания могут задавать невозобновимые ресурсы, которые необходимы для выполнения работ. Например, материалы, топливо и т.п.;

- **Возобновимые ресурсы**. Техника, персонал. Ресурсы могут объединятся в "бригады", которые выступают неделимым ресурсом. Зачастую у разных ресурсов, задействованных на работе, бывает разный график работы. Это необходимо учитывать, при составлении календарного плана проекта;

- **График работы**. Разные работы могут выполнятся с разным графиком. К примеру, основные монтажные работы в 2 смены 7 дней в неделю, а вспомогательные работы – только в 1 смену и только в первую половину недели;

- **Ограничение на время выполнения**. Параметр принимает значения "Как можно раньше", "Как можно позже", "Фиксированная дата", "Начало не позже", "Начало не раньше".

- **Характеристики взаимосвязей** между работами также могут отличаться. Обыденной представляется взаимосвязь типа "Окончание-Начало", то есть работа-последователь начнется не раньше, чем закончится "работа-предшественник". На практике дополнительно выделяют связи "Начало-Начало" (когда работа-последователь не может начаться раньше, чем начнется работа-предшественник), "Окончание-Окончание", "Начало-Окончание". Более того, по технологии между окончанием работы-предшественника и началом работы-последователя может быть задана задержка по времени.

Некоторые из перечисленных особенностей могут быть включены в математическую модель RCPSP в качестве дополнительных условий. В этом случае говорят о задаче RCPSP *в расширенной постановке*. Перечислим некоторые из этих условий.

- Количество возобновимого ресурса k может зависеть от времени. То есть вместо константы Q_k может быть определена функция от времени $Q_k(t)$. Такая функция обычно задается в табличном виде. Например, с 10-го по 15-ое февраля доступно 3 экскаватора. С 16-го по 18-ое – 2 экскаватора, а с 18-го по 25-ое февраля – 4 экскаватора;

- Продолжительность работы может зависеть от количества занятых на ней ресурсов. Очевидно, что если на выполнение работы назначить больше исполнителей (на кладку стен – больше каменщиков, на рытье котлованов – больше экскаваторов и самосвалов), то работа может быть выполнена быстрее. Для каждой работы могут быть заданы фиксированные варианты выполнения. Например, "Вариант 1: продолжительность = 5 дней, необходимо 2 экскаватора и 5 разнорабочих", "Вариант 2: продолжительность = 4 дня, необходимо 3 экскаватора и 5 разнорабочих" и т.д. Задачу RPCSP в такой постановке называют *мультимодальной* и обозначают MRCPSP. При такой постановке необходимо определить вариант, по которому будет выполняться работа и уже потом построить расписание;

- Отличие взаимосвязей может быть задано условиями. Например, для описания взаимосвязи "НАЧАЛО-НАЧАЛО", можно задать условие $S_j \geq S_i + \triangle_{ij}$, где $\triangle_{ij}$ – продолжительность задержки.

Даже без этих дополнительных ограничений задача RCPSP остается *экстремально* NP-трудной. Лучший из известных точных алгоритмов Брукера[43] за приемлемое время может решать примеры размерности не больше $n = 60$. Более того, даже для частного случая с одним невозобновимым ресурсом $K = 1$ неизвестны полиномиальные алгоритмы решения с относительной погрешностью, гарантированно не превосходящей некоторой константы.

5.2 Алгоритм диспетчеризации для задачи RCPSP

В этом параграфе приведен известный алгоритм построения допустимого расписания выполнения работ проекта.

Algorithm 11 Алгоритм List Scheduling (LS).

1: Пусть EL – список всех требований без предшественников. Полагаем $Q_k(\tau) = Q_k, \forall \tau, k = 1, \ldots, K$.
2: **while** $EL \neq \emptyset$ **do**
3: Выберем требование $j \in EL$;
4: $t := \max\limits_{i|(i,j)\in A} \{S_i + p_i\}$. Если для требования j предшественники не определены, тогда $t = 0$;
5: **while** Существует такой ресурс k, что $q_{jk} > Q_k(\tau)$ для некоторого $\tau \in [t, t+p_j)$ **do**
6: Вычислим такое минимальное значение $t_k > t$, что требование j может быть обслужено на интервале $[t_k, t_k + p_j)$, если рассматривается только ресурс k;
7: $t := t_k$;
8: **end while**
9: Назначим выполнение требования j на интервал $[S_j, C_j) = [t, t + p_j)$;
10: Резервируем ресурсы под выполнение требования j: $Q_k(\tau) = Q_k(\tau) - q_{jk},\ k = 1, \ldots, K,\ \tau \in [t, t + p_j)$;
11: $EL = EL \setminus \{j\}$;
12: Добавляем в EL последователей требования j, для которых все предшественники поставлены в расписание;
13: **end while**

Полученное расписание зависит от выбора требования j на шаге 3 алгоритма. Идея данного алгоритма заключается в том, что требование j ставится в расписание с наиболее раннего момента времени, при котором не нарушаются ресурсные ограничения и ограничения предшествования. Трудоемкость алгоритма $O(n^2K)$ операций.

Активным назовем такое допустимое расписание (решение) $S = (S_1, \ldots, S_n)$, для которого не существует другого допустимого расписания $S' = (S'_1, \ldots, S'_n)$ такого, что $S'_j \leq S_j$, $\forall j \in N$, и хотя бы одно из неравенств строгое.

Легко убедиться, что алгоритмом LS строятся только активные расписания. Очевидно также, что оптимальное расписание следует искать среди множества активных.

В следующей теореме доказано, что активному расписанию $(S_1, \ldots, S_n)$ соответствует некоторая перестановка элементов множества N, которую будем обозначать через $\pi = (j_1, \ldots, j_n)$.

Теорема 5.1 *Активное расписание однозначно представляется в виде перестановки $\pi = (j_1, \ldots, j_n)$ из n требований.*

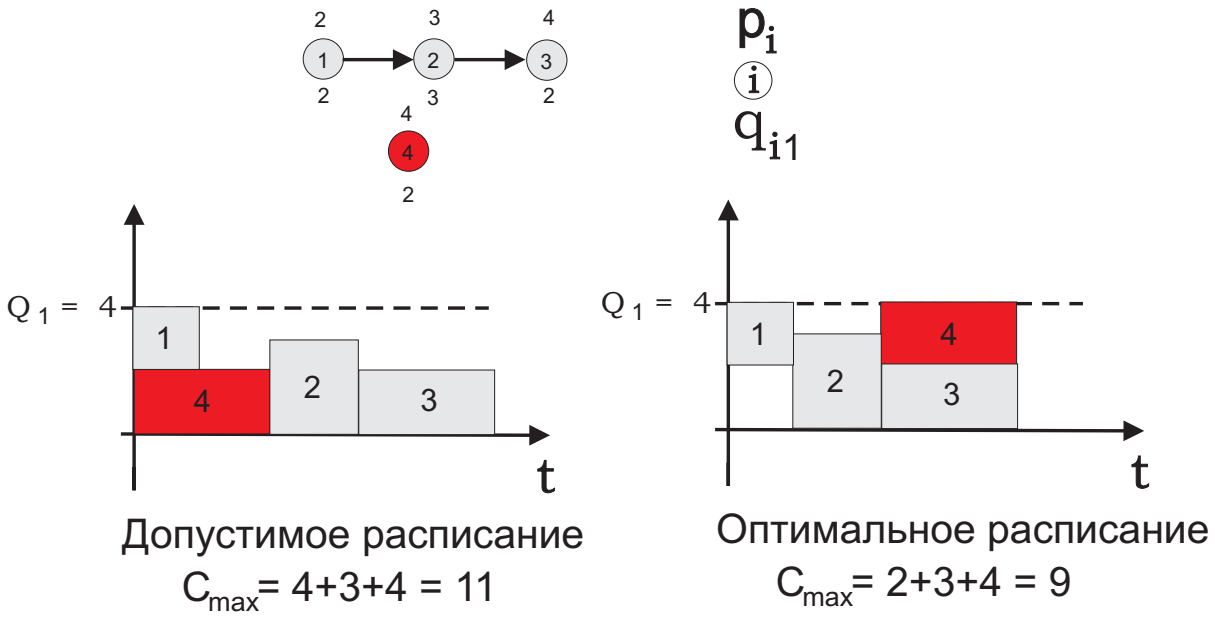

Рис. 5.1: Пример задачи RCPSP.

Перестановка π задает последовательность выбора требований j на шаге 3 алгоритма LS. Подобный алгоритм зачастую называют “конкурсом”, т.к. работа на шаге 3 может выбираться согласно некоторому “приоритету”, конкурсу.

Пример 1. Рассмотрим пример, представленный на рис. 5.1. Проект состоит из 4-х требований. На сетевом графике сверху над узлами указана продолжительность обслуживания требований p_i, снизу – необходимое для обслуживания требования количество q_{i1} возобновляемого ресурса 1. Всего доступно 4 единицы ресурса 1 (к примеру, доступно 4 монтажника).

На рисунке 5.1 представлены 2 допустимых расписания, при которых соблюдены отношения предшествования, ограничения на ресурсы и продолжительность обслуживания требований.

Продемонстрируем работу алгоритма LS на данном примере.

Шаг 1. $EL = \{1, 4\}$
Шаги 2-13. Пусть $j = 1$. Самое раннее время выполнения требования 1: $[0, 2)$. $EL = \{4, 2\}$;
Шаги 2-13. Пусть $j = 4$. Самое раннее время выполнения требования 4: $[0, 4)$. $EL = \{2\}$;

Шаги 2-13. $j = 2$. Самое раннее время выполнения требования 3: $[4, 7)$, так как на интервале $[2, 4)$ для требования 2 уже не хватает ресурса (зарезервированы под требование 4). $EL = \{3\}$;
Шаги 2-13. $j = 3$. Самое раннее время выполнения требования 3: $[7, 11)$, т.к. требование 2 заканчивается в момент времени 7. $EL = \emptyset$

В итоге мы получили расписание, при котором время выполнения проекта $C_{\max} = 11$. Мы можем представить последовательность постановки требований в расписание как перестановку $\pi = (1, 4, 2, 3)$.

Если бы мы ставили требования в порядке $(1, 2, 4, 3)$ или $(1, 2, 3, 4)$, то получили бы расписание, при котором $C_{\max} = 9$, то есть расписание, соответствующее перестановке $\pi = (1, 2, 4, 3)$, лучше с точки зрения целевой функции. Таким образом, полиномиальный алгоритм LS не гарантирует нахождение оптимального решения.

На основании данного алгоритма диспетчеризации можно построить точный алгоритм Ветвей и Границ. Ветвление происходит при выборе требования j. Схема ветвления данного алгоритма для примера рис.5.1 представлена на рис.5.2. Так как алгоритм Ветвей и Границ является одним из наиболее эффективных точных алгоритмов для решения данной задачи, то становится актуальным нахождение "хороших" нижних и верхних оценок.

5.3 Задача RCPSP с прерываниями обслуживания требований

Задачу RCPSP, где разрешены прерывания в обслуживании требований, будем называть PRCPSP (Preemption Resource-Constrained Project Scheduling Problem).

Решение задачи PRCPSP представляют в виде набора множеств, состоящих из пар чисел

$$S' = \{S_1 = \{[s_{11}, c_{11}), \dots, [s_{1m_1}, c_{1m_1})\},$$

$$S_2 = \{[s_{21}, c_{21}), \dots, [s_{2m_2}, c_{2m_2})\}, \dots\}.$$

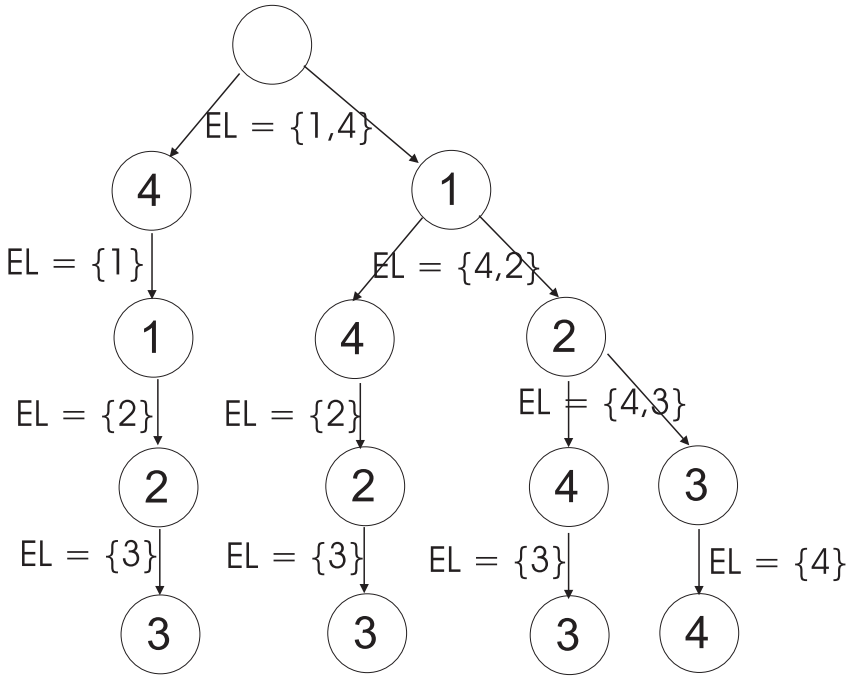

Рис. 5.2: Дерево поиска оптимального расписания.

Требование 1 обслуживается непрерывно на промежутках времени $[s_{11}, c_{11}), \ldots, [s_{1m_1}, c_{1m_1})$, с $m_1 - 1$ прерываниями.

Пусть $C^*_{\max}$ – оптимальное значение целевой функции для примера RCPSP. Обозначим через $C^*_{\max}(pmtn)$ – оптимальное значение целевой функции для того же примера, когда разрешены прерывания.

На рис. 5.3 представлен пример RCPSP и оптимальные расписания в случаях с прерываниями и без прерываний. В примере количество работ равно n, и в каждый момент времени доступно две единицы ресурса.

Сетевой график проекта для данного примера состоит из двух несвязных фрагментов. Первый фрагмент – последовательность "длинных" и "коротких" требований (продолжительностью p и ε, соответственно, где ε – достаточно малая величина). Второй фрагмент состоит из одного требования, продолжительностью $\left(\frac{n}{2}\right) p$. Значение целевой функции:

$$C^*_{\max} = \left(\frac{n}{2} - 1\right)(p + \varepsilon) + \frac{n}{2}p,\ C^*_{\max}(pmtn) = \left(\frac{n}{2} - 1\right)(p + \varepsilon) + p,$$

$$C^*_{\max} - C^*_{\max}(pmtn) = \left(\frac{n}{2} - 1\right)p < C^*_{\max}(pmtn),$$

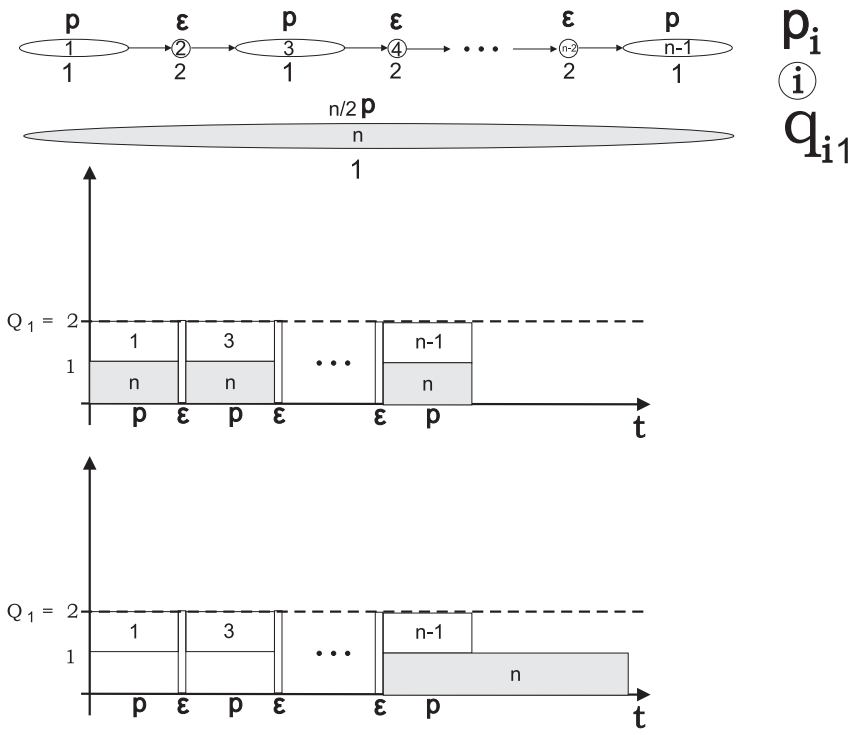

Рис. 5.3: Пример задачи RCPSP с прерываниями.

$$C^*_{\max} < 2C^*_{\max}(pmtn).$$

То есть для данного примера время выполнения проекта без прерываний превышает время выполнения проекта с прерываниями не более чем в 2 раза.

Обратим внимание, что при оптимальном расписании с прерываниями каждый фрагмент $l = [s_{nl}, c_{nl})$ требования с номером n обслуживается параллельно с некоторым другим требованием.

5.4 Нижние оценки для задачи RCPSP

В данном параграфе приводятся некоторые известные нижние оценки для задачи RCPSP, а также сведения об их точности.

Определение 11 *Путь соединяющий вершины* 0 *и* $n+1$ *в сетевом графике и имеющий наибольшую длину называется критическим путем*

(длина складывается из продолжительности обслуживания требований, входящих в этот путь).

Длина критического пути будет равна $C^*_{\max}$, когда нет ресурсных ограничений. То есть длина критического пути является нижней оценкой для $C^*_{\max}$. Эту нижнюю оценку принято обозначать LB_0.

Лемма 5.1 *Существует пример RCPSP, для которого* $\frac{C^*_{\max}}{LB_0} = n$.

Доказательство. Рассмотрим пример RCPSP, где требования i, $i = 1, \ldots, n$, имеют продолжительности обслуживания $p_i = p$. Ограничений предшествования между требованиями нет. Требуется всего один ресурс, причем $q_{i1} = Q_1$, $i = 1, \ldots, n$. Очевидно, что $LB_0 = p$. Но $C^*_{\max} = pn$, т.к. никакие два требования не могут обслуживаться параллельно в силу ограничения на ресурсы. Поэтому $\frac{C^*_{\max}}{LB_0} = n$. □

Так как граф предшествования ацикличный, то оценка LB_0 может быть найдена за $O(n^2)$ операций.

Другая нижняя оценка LB_1 может быть найдена за $O(nK)$ операций при рассмотрении каждого ресурса отдельно.

Определение 12 *Назовем величину* $\sum_{i=1}^{n} q_{ik} p_i$ *суммарной загрузкой ресурса* k.

Очевидно, что $\sum_{i=1}^{n} q_{ik} p_i \le Q_k \cdot C^*_{\max}, k = 1, \ldots, n$. Тогда значение

$$LB_1 = \max_{k=1}^{K} \left\lceil \sum_{i=1}^{n} q_{ik} p_i / Q_k \right\rceil$$

является нижней оценкой для $C^*_{\max}$.

Лемма 5.2 *Существует пример RCPSP, для которого* $C^*_{\max} - LB_1 = \sum_{i=1}^{n} p_i - 1$.

Доказательство. Рассмотрим пример RCPSP, где требования $i = 1, \ldots, n$ имеют продолжительности обслуживания p_i. Заданы отношения предшествования $i \to i+1,\ i = 1, \ldots, n-1$. $q_{i1} = \varepsilon,\ i = 1, \ldots, n$. Пусть $Q_1 = \sum\limits_{i=1}^{n} p_i\varepsilon$. Очевидно, что $LB_1 = \sum\limits_{i=1}^{n} q_{i1}p_i/Q_1 = 1$. Но $C^*_{\max} = \sum\limits_{i=1}^{n} p_i$. Тогда $C_{\max}(S^*) - LB_1 = \sum\limits_{i=1}^{n} p_i - 1$. □

Рассмотрим еще одну тривиальную нижнюю оценку LB_S, которую называют "дополнением критического пути".

Обозначим множество требований, принадлежащих критическому пути через CP. Для каждого требования $i \notin CP$ обозначим через e_i максимальную длину интервала, входящего в $[r_i, d_i]$, в котором требование i может обслуживаться параллельно с требованиями критического пути без нарушения ограничений на ресурсы. Если выполняется $e_i < p_i$, то не существует допустимого расписания, при котором $C^*_{\max} = LB_0$. Тогда значение

$$LB_S = LB_0 + \max_{i \notin CP}\{\max\{p_i - e_i, 0\}\}$$

также является нижней оценкой для $C^*_{\max}$.

LB_S может быть найдена за $O(nK|CP|)$ операций, где $|CP|$ – количество требований в критическом пути.

Лемма 5.3 *Существует пример RCPSP, для которого* $\frac{C^*_{\max}}{LB_S} = \frac{n}{2}$.

Доказательство. Рассмотрим пример RCPSP, где требования $i = 1, \ldots, n$ имеют продолжительности обслуживания $p_i = p$. Ограничений предшествования между требованиями нет. Пусть $q_{i1} = Q_1,\ i = 1, \ldots, n$, тогда $LB_0 = p$. Нетрудно вычислить $LB_S = p + \max\{p - 0, 0\} = 2p$.

Но $C^*_{\max} = pn$, т.к. никакие два требования не могут обслуживаться параллельно в силу ограничения на ресурсы $q_{i1} = Q_1, i = 1, \ldots, n$. Поэтому $\frac{C^*_{\max}}{LB_S} = \frac{n}{2}$. □

5.4.1 Нижняя оценка Mingozzi

Одной из самых эффективных нижних оценок является оценка Mingozzi. В работе [86] представлена модель задачи RCPSP как задачи линейного программирования. Нижняя оценка Mingozzi получена частичной релаксацией данной модели. Стоит отметить, что идея такой оценки не нова. В более ранней отечественной работе [44] уже был рассмотрен подобный метод получения нижней оценки.

Перед описанием алгоритма вычисления оценки Mingozzi дадим некоторые определения. Если существует путь из вершины i в вершину j в графе отношений предшествования, то будем говорить, что между требованиями i и j существуют *косвенные* отношения предшествования.

Если $i \to j$, то будем говорить, что между данными требованиями заданы *прямые* отношения предшествования.

Множество требований $X \subset N$ будем называть *допустимым множеством*, если между каждой парой требований $i, j \in X$ нет отношений предшествования (прямых или косвенных) и требования могут обслуживаться параллельно без нарушения ресурсных ограничений ($\sum\limits_{i \in X} q_{ik} \leq Q_k, k = 1, \ldots, K$).

Допустимое множество X назовем *доминирующим*, если не существует другого допустимого множества Y такого, что выполняется $X \subset Y$.

Рассмотрим список всех доминирующих множеств $X_1, \ldots, X_f$ и соответствующие им векторы $a^j \in \{0,1\}^n,\ j = 1, \ldots, f$: $a_i^j = 1$, если $i \in X_j$, иначе $a_i^j = 0,\ i = 1, \ldots, n$.

Определим через x_j длину интервала, в котором все требования множества X_j обслуживаются параллельно. Тогда решение следующей задачи линейного программирования позволяет вычислить нижнюю оценку LB_M для $C^*_{\max}$ [86]:

$$\begin{cases} \min \sum\limits_{j=1}^{f} x_j, \\ \sum\limits_{j=1}^{f} a_i^j x_j \geq p_i, \ i = 1, \ldots, n, \\ x_j \geq 0, \ j = 1, \ldots, f. \end{cases} \tag{5.1}$$

В данной постановке допускается, что требование i может обслуживаться больше чем за p_i единиц времени, но не меньше. В модели частично нарушаются отношения предшествования и допускаются прерывания обслуживания требований.

В известной монографии [43] отмечается, что f растет экспоненциальным образом с ростом n. При $n = 60$ имеем $f \approx 300000$, для $n = 90$ даже $f \approx 8000000$. В той же монографии приводится эффективная техника вычисления нижней оценки LB_M. Но результаты экспериментов показали, что "качество оценки плохое". Тем не менее, оценка является одной из самых "сильных" на данный момент.

Лемма 5.4 *Вычисление оценки LB_M является NP–трудной задачей.*

Доказательство. Рассмотрим NP-трудную задачу УПАКОВКИ В ПАЛЛЕТЫ. Паллеты – это "коробки", в которые по ширине и высоте помещается только один предмет. Даны стандартные паллеты длиной W и n предметов, каждый длиной $w_i, \ i = 1, \ldots, n$. Необходимо упаковать все предметы в паллеты так, чтобы число использованных паллет было минимальным.

Преобразуем исходную задачу УПАКОВКИ В ПАЛЛЕТЫ в задачу RCPSP. Каждое требование $i = 1, \ldots, n$, соответствует предмету i. Зададим $p_i = 1$, $q_{i1} = w_i$, $i = 1, \ldots, n$, $Q_1 = W$. Отношения предшествования между требованиями не заданы. Тогда $C_{\max}(S^*)$ соответствует минимальному необходимому количеству паллет в исходной задаче.

Очевидно, что $LB_M = C^*_{\max}$ для построенного примера RCPSP. Следовательно, задача нахождения оценки LB_M эквивалентна решению NP-трудной задачи упаковки в паллеты. □

Решение LB_M допускает прерывание обслуживания некоторых требований. Тогда верно следующее утверждение.

Лемма 5.5 *Существует пример RCPSP, для которого* $\frac{C_{\max}(S^*)}{LB_M} \approx 2$.

Доказательство. Рассмотрим пример на рис. 5.3. Для данного примера при вычислении LB_M имеем доминирующие множества: $X_1 = \{1, n\}$, $X_2 = \{2\}$, $X_3 = \{3, n\}$, $X_4 = \{4\}, \ldots, X_{n-1} = \{n-1, n\}$ и оптимальное решение соответствующей задачи линейного программирования (5.1): $x_1 = x_3 = \cdots = x_{n-1} = p$, $x_2 = x_4 = \cdots = x_{n-2} = \varepsilon$.

Можно так подобрать значения n, p, ε, что получим $\frac{C_{\max}(S^*)}{LB_M} \approx 2$. □

5.5 Соотношение оптимальных значений для задачи RCPSP с прерываниями и без прерываний

Пусть $C^*_{\max}(pmtn)$ – оптимальное значение целевой функции, когда прерывания разрешены. Можно предположить (анализируя, например, рис. 5.3), что на одном и том же примере $C^*_{\max} < 2 \cdot C^*_{\max}(pmtn)$. Это предположение верно для некоторых частных случаев задачи. Однако в общем случае предположение ошибочно.

Предположение не выполняется для примера, изображенного на рис. 5.4, где $p \gg \varepsilon$ (например, $\varepsilon = 1/(np)^n,\ p > 1$). Можно построить аналогичный пример, в котором $2h$ требований имеют продолжительности обслуживания равные p, другие $2h+1$ требований имеют *короткие* продолжительности обслуживания равные ε и потребности в ресурсах $q = Q_1$, для одного *длинного* требования продолжительность обслуживания равно $2h \cdot p$, а еще два требования имеют продолжительности обслуживания $h \cdot p$, т.е. $n = 4h + 4$. Если h достаточно большое число, а ε – достаточно маленькое, то выполняется $C^*_{\max} \approx 2.5 \cdot C^*_{\max}(pmtn)$.

Нетрудно убедиться, что для данного примера $LB_M = C^*_{\max}(pmtn)$. То есть мы можем доказать следующую лемму:

Лемма 5.6 *Существует пример RCPSP, для которого*

$$\frac{C^*_{\max}}{LB_M} \approx 2.5.$$

Как показано далее, существует похожий пример, для которого $C^*_{\max} > 2 \cdot C^*_{\max}(pmtn)$ и, как следствие, $\frac{C^*_{\max}}{LB_M} = O(\log n)$.

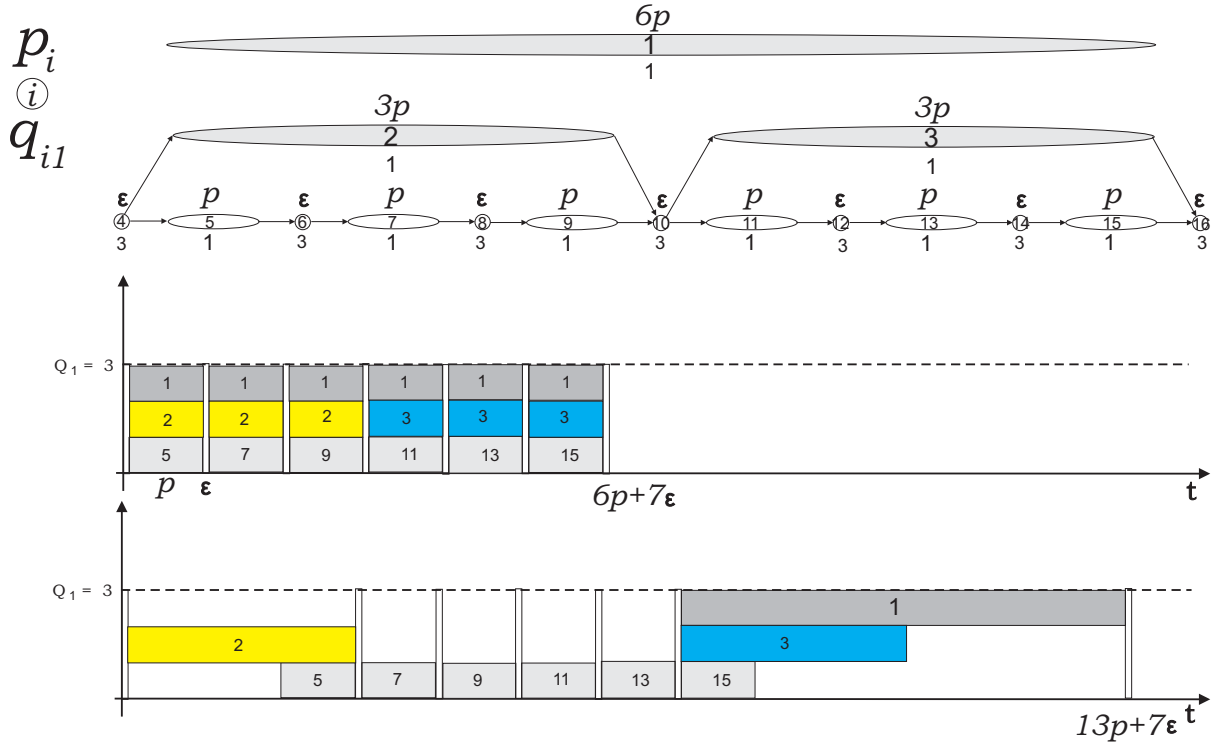

Рис. 5.4: Пример $C^*_{\max} > 2 \cdot C^*_{\max}(pmtn)$

Теорема 5.2 *Существует пример задачи RCPSP, для которого*

$$\frac{C^*_{\max}}{C^*_{\max}(pmtn)} = O(\log n).$$

Доказательство. Рассмотрим пример вида, представленного на рис. 5.5 (а). В этом примере задано $m+1$ уровней требований. На верхнем уровне находится требование j^1_1 с продолжительностью обслуживания Mp, на втором сверху уровне имеем два требования j^2_1 и j^2_2 с продолжительностями обслуживания равными $\frac{M}{2}p$, на уровне h сверху, $h = 2, 3, \ldots, m$, имеем 2^{h-1} требований с продолжительностями обслуживания $p_{j^h_i} = \frac{M}{2^{h-1}}p$, и т.д. На самом нижнем уровне, задана цепочка из коротких и очень коротких требований $e_1 \to j^{m+1}_1 \to e_2 \to j^{m+1}_2 \to \cdots \to e_M \to j^{m+1}_M \to e_{M+1}$, где $p_{e_i} = \varepsilon$, $q_{e_i} = m+1$, $i = 1, 2, \ldots, M+1$, и $p_{j^{m+1}_i} = p$, $q_{j^{m+1}_i} = 1$, $i = 1, 2, \ldots, M$. Для каждого требования j^h_i уровня h, $h = 1, 2, \ldots, m$, имеем $q_{j^h_i} = 1$. На каждом уровне h, $h = 2, 3, \ldots, m$, заданы отношения предшествования в виде цепочки $e_1 \to j^h_1 \to e_{\frac{M}{2^{h-1}}+1} \to j^h_2 \to e_{2\frac{M}{2^{h-1}}+1} \to \cdots \to j^h_{2^{h-1}} \to e_{M+1}$, в которых также участвуют очень

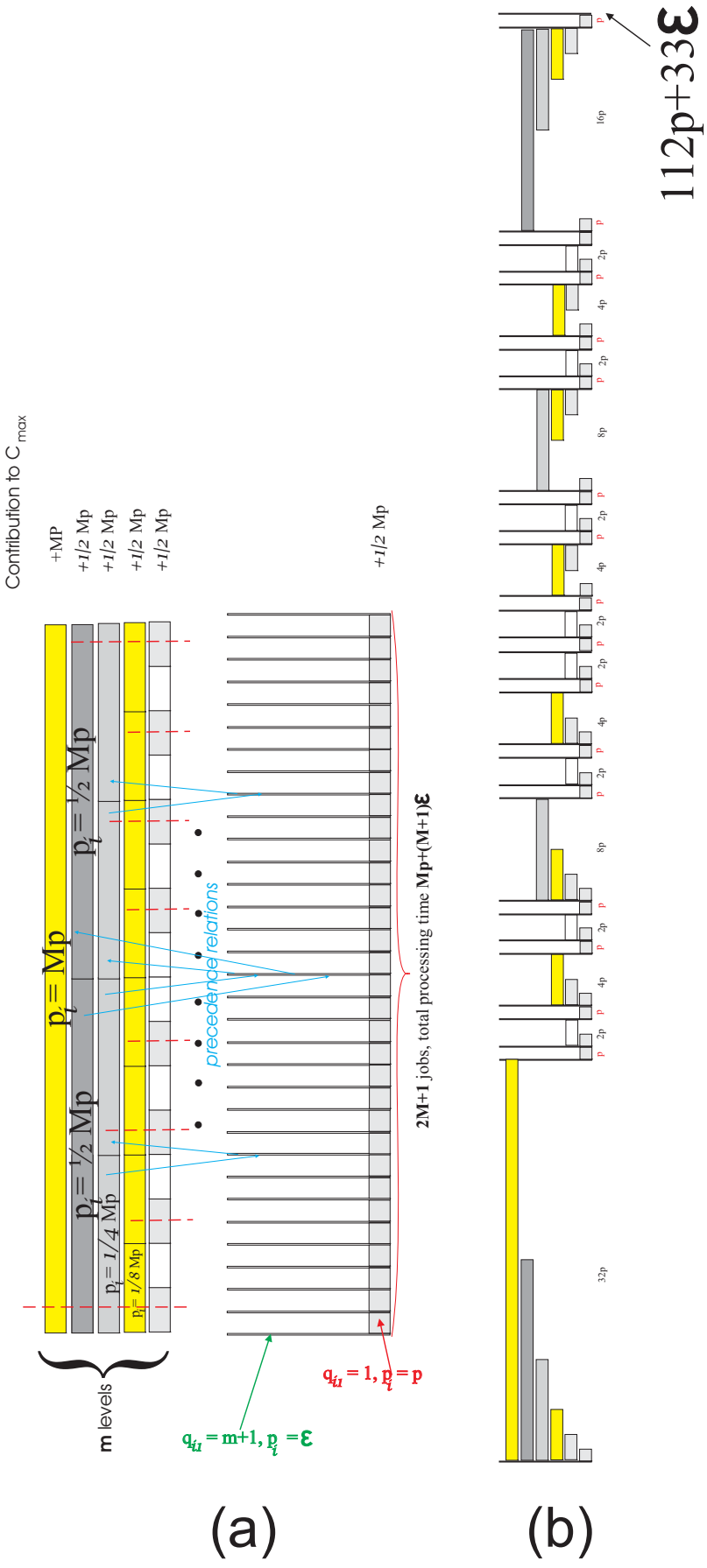

Рис. 5.5: Пример из доказательства теоремы 5.2

короткие требования из нижнего уровня. Некоторые из этих отношений предшествования представлены на рис. 5.5 (a). Пусть $(M+1)\varepsilon \ll p$ (например, $\varepsilon = 1/((M+1)p)^2$, $p \in Z_+$). Пунктирными линиями обозначены требования, обслуживаемые параллельно при оптимальном расписании в задаче без прерываний.

В качестве иллюстрации на рис. 5.5 (b) для подобного примера с количеством уровней $m+1=6$ представлено оптимальное расписание без прерываний. Для этого примера выполняется $C^*_{\max}(pmtn) = 32p + 33\varepsilon$ и $C^*_{\max} = 112p + 33\varepsilon$.

В общем случае, для произвольного $m > 2$, имеем $C^*_{\max}(pmtn) = Mp + (M+1)\varepsilon$ и $C^*_{\max} = Mp + \frac{m}{2}Mp + (M+1)\varepsilon$. Тогда выполняется

$$\frac{C^*_{\max}}{C^*_{\max}(pmtn)} \approx \frac{m+2}{2}.$$

Выразим значение M через m. Имеем $2^m = M$. Тогда

$$n = 2M + 1 + 1 + 2 + \cdots + 2^{m-1} = 2M + 1 + 2^m - 1 = 2 \cdot 2^m + 2^m = 3 \cdot 2^m,$$

из чего следует

$$m = \log \frac{n}{3}.$$

Поэтому верно соотношение

$$\frac{C^*_{\max}}{C^*_{\max}(pmtn)} \approx \frac{m+2}{2} = \frac{\log n - \log 3 + 2}{2}.$$

□

Как следствие, для данного примера выполняется $\frac{C^*_{\max}}{LB_M} = O(\log n)$, и мы можем сформулировать следующую теорему:

Теорема 5.3 *Существует класс примеров задачи RCPSP, для которого выполняется*

$$\frac{C^*_{\max}}{LB_M} = O(\log n),$$

и вычисление оценки LB_M для данного класса является NP-трудной задачей.

Конструкция такого примера несложна. Пример содержит два подмножества требований N_1 и N_2. Требования из первого подмножества соответствуют примеру, изображенному на рис. 5.5 (а), где $Q_1 = m + 1$. В подможество N_2 входят n независимых требований с продолжительностями обслуживания $p_j = 1$ и $\sum\limits_{j \in N_2} q_j = 2m + 2$. Дополнительно задано фиктивное требование o_1 такое, что $j \to o_1 \to l$ для всех $j \in N_1$, $l \in N_2$. Очевидно, что к этому классу примеров (точнее, "к подмножеству" N_2) сводится ЗАДАЧА РАЗБИЕНИЯ. А следовательно, вычисление оценки LB_M для данного класса примеров является NP-трудной задачей.

5.6 Алгоритмы вычисления верхних оценок для задачи RCPSP

Алгоритм диспетчеризации LS строит допустимое расписание, а следовательно с его помощью можно вычислить верхнюю оценку оптимального значения целевой функции.

Ключевой составляющей алгоритма LS является правило предпочтения, с помощью которого выбирается требование $j \in EL$ для постановки в расписание на шаге 3 алгоритма. Можно предложить несколько элементарных эвристических правил предпочтения или, как их по другому называют, правил диспетчеризации. Таким образом можно получить несколько верхних оценок и выбрать из них лучшую.

Некоторые из этих правил представлены в таб. 5.1.

Далее мы докажем, что алгоритм LS с любым из перечисленных правил предпочтения имеет относительную погрешность не меньше $O(\sqrt{n})$ даже для частного случая задачи с одним ресурсом Q_1. В этом параграфе рассматривается частный случай задачи с одним ресурсом, поэтому мы будем использовать обозначения $q_i = q_{i1}$ и $Q = Q_1$.

Обозначим через $C_{\max}(LS_a)$ значение целевой функции, соответствующее расписанию, полученному при помощи алгоритма LS с использованием правила предпочтения a. Докажем, что для перечисленных правил

Таблица 5.1: Правила предпочтения (диспетчеризации)

Обозн-ие	Описание	Отн. погрешн.
Правила, основанные на параметрах требований		
SPT	выбрать работу с наименьшей продолжительностью обслуживания	$O(n)$
LPT	выбрать работу с наибольшей продолжительностью обслуживания	
Правила, основанные на отношениях предшествования		
MIS	выбрать работу с наибольшим количеством непосредственных последователей	$O(\sqrt{n})$
LIS	выбрать работу с наименьшим количеством непосредственных последователей	
MTS	выбрать работу с наибольшим количеством всех последователей	
LTS	выбрать работу с наименьшим количеством всех последователей	
GRPW	выбрать работу с наибольшим весом, т.е. требование с наибольшой суммарной продолжительностью всех последователей	$O(n)$
Правила, основанные на информации о критическом пути		
EST	выбрать работу с наим. возможным моментом начала r_i	$O(n)$
ECT	выбрать работу с наименьшим значением $r_i + p_i$	
LST	выбрать работу с наименьшим значением $d_i - p_i$	
LCT	выбрать работу с наименьшим значением d_i	
MSLK	выбрать работу с наименьшим ресурсом $d_i - r_i - p_i$,	
MW	выбрать работу с наименьшим окном $d_i - r_i$,	
DEST (Динамический EST)	выбрать работу с наим. возможным моментом начала r_i	
DECT (Динамический ECT)	выбрать работу с наименьшим значением $r_i + p_i$, где r_i вычисляется динамически	$O(\sqrt{n})$
Правила, основанные на информации о необходимых ресурсах		
GRR	выбрать работу с наибольшей потребностью в ресурсах	$O(n)$

выполняется

$$\frac{C_{\max}(LS_a)}{C^*_{\max}} = O(n) \qquad (\text{или } \; O(\sqrt{n})).$$

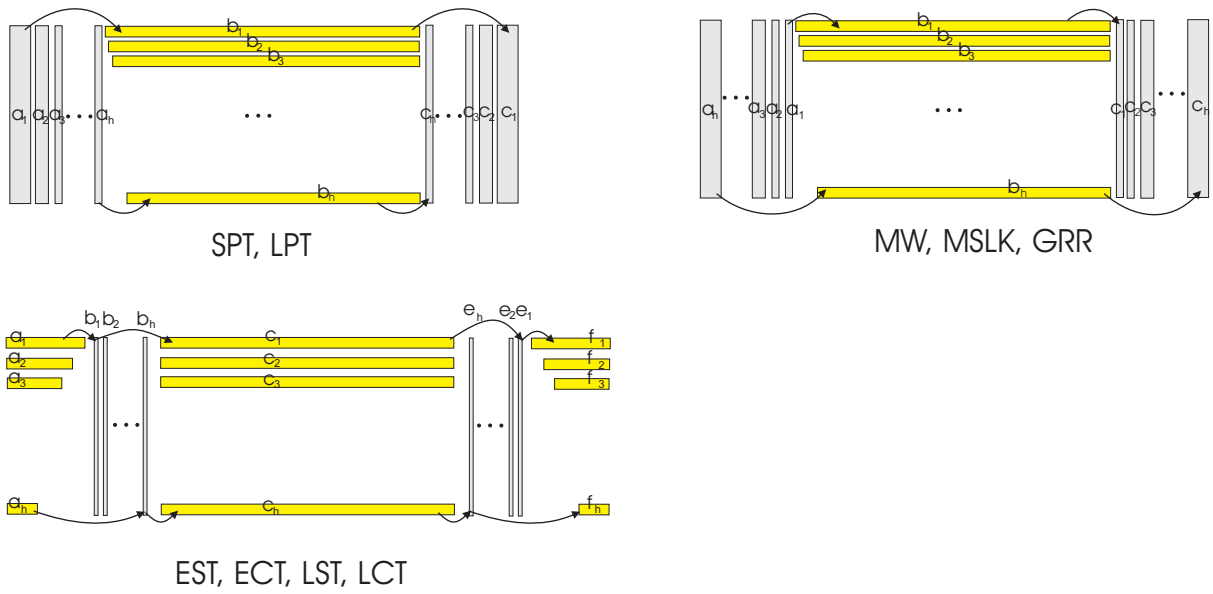

Рис. 5.6: Правила предпочтения. Часть 1.

Лемма 5.7 *Существует пример задачи RCPSP, для которого*

$$\frac{C_{\max}(LS_a)}{C^*_{\max}} = O(n),\ a \in \{SPT, LPT\}.$$

Доказательство. Рассмотрим пример, изображенный на рис. 5.6 (SPT, LPT). Дано h цепочек требований $a_i \to b_i \to c_i$, $i = 1, 2, \ldots, h$. Для каждого требования b_i, $i = 1, 2, \ldots, h$, имеем $p_{b_i} = p - \varepsilon(i-1)$, $q_{b_i} = 1$. Для каждого требования a_i, $i = 1, 2, \ldots, h$, дано $p_{a_i} = \varepsilon(h-i+1)$, $q_{a_i} = h$. Для каждого требования c_i, $i = 1, 2, \ldots, h$, имеем $p_{c_i} = \varepsilon(h-i+1)$, $q_{c_i} = h$, где $h^2\varepsilon \ll p$ (здесь и далее полагаем, что $\varepsilon = 1/(n \cdot p)^n$, $p \in Z_+$) и $Q_1 = h$.

Для этого примера оптимальное расписание соответствует перестановке $\pi = (a_1, a_2, \ldots, a_h, b_1, b_2, \ldots, b_h, c_1, c_2, \ldots, c_h)$ (порядку выбора требований на шаге 3 алгоритма LS). Алгоритмом LS с правилом SPT будет получена перестановка
$\pi_{SPT} = (a_h, a_{h-1}, \ldots, a_1, b_h, c_h, b_{h-1}, c_{h-1}, \ldots, b_1, c_1)$, т.е. все требования $b_1, b_2, \ldots, b_h$ не обслуживаются параллельно ни в один из моментов времени (не перекрываются). Поэтому выполняется $C^*_{\max} = 2(h + (h-1) + \cdots + 1)\varepsilon + p$ и $C_{\max}(LS_{SPT}) = 2(h + (h-1) + \cdots + 1)\varepsilon + p + (p - \varepsilon) + (p - 2\varepsilon) + \cdots + (p - (h-1)\varepsilon)$. Тогда

$$\lim_{\varepsilon \to 0} \frac{C_{\max}(LS_{SPT})}{C^*_{\max}} = h = \frac{n}{3}.$$

Аналогично рассматривается перестановка
$\pi_{LPT} = (a_1, b_1, c_1, a_2, b_2, c_2, \ldots, a_h, b_h, c_h)$ для правила LPT. $\square$

Назовем *перевернутым примером* пример, в котором все ориентации дуг в графе отношений предшествования изменены на противоположные. Нетрудно заметить, что пример из леммы 5.7 равен своему перевернутому примеру. То есть относительная погрешность алгоритма LS с этими же правилами предпочтения будет такой же и для перевернутого примера.

Лемма 5.8 *Существует пример задачи RCPSP, для которого*

$$\frac{C_{\max}(LS_a)}{C^*_{\max}} = O(n),\ a \in \{EST, LST\}.$$

Доказательство. Рассматриваем статичную версию этих правил, т.е. значения r_i, d_i вычисляются только один раз перед запуском алгоритма LS. Рассмотрим пример, изображенный на рис. 5.6 (EST, ECT, LST, LCT). Дано h цепочек требований $a_i \to b_i \to c_i \to e_i \to f_i$, $i = 1, 2, \ldots, h$. Для каждого требования b_i, e_i, $i = 1, 2, \ldots, h$, имеем $p_{b_i} = p_{e_i} = \varepsilon$, $q_{b_i} = q_{e_i} = h$. Для каждого требования a_i, f_i, $i = 1, 2, \ldots, h$, дано $p_{a_i} = p_{f_i} = 2\varepsilon(h - i + 1)$, $q_{a_i} = q_{f_i} = 1$. Для каждого требования c_i, $i = 1, 2, \ldots, h$, имеем $p_{c_i} = p$, $q_{c_i} = 1$, где $h^2\varepsilon \ll p$ и $Q_1 = h$.

Для этого примера оптимальное расписание соответствует перестановке $\pi = (a_1, a_2, \ldots, a_h, b_1, b_2, \ldots, b_h, c_1, c_2, \ldots, c_h, e_1, e_2, \ldots, e_h, f_1, f_2, \ldots, f_h)$. Алгоритмом LS с правилом EST будет получена перестановка
$\pi_{EST} = (a_1, a_2, \ldots, a_h, b_h, c_h, e_h, f_h, b_{h-1}, c_{h-1}, e_{h-1}, f_{h-1}, \ldots, b_1, c_1, e_1, f_1)$, т.е. все требования $c_1, c_2, \ldots, c_h$ не обслуживаются параллельно ни в один из моментов времени (не перекрываются). Поэтому выполняется

$$\lim_{\varepsilon \to 0} \frac{C_{\max}(LS_{EST})}{C^*_{\max}} = h = \frac{n}{5}.$$

Аналогично рассматривается перестановка
$\pi_{LST} = (a_1, b_1, c_1, e_1, f_1, \ldots, a_h, b_h, c_h, e_h, f_h)$ для правила LST. $\square$

Лемма 5.9 *Существует пример задачи RCPSP, для которого*

$$\frac{C_{\max}(LS_a)}{C^*_{\max}} = O(n),\ a \in \{ECT, LCT\}.$$

Доказательство. Рассматриваем статичную версию этих правил, т.е. значения r_i, d_i вычисляются только один раз перед запуском алгоритма LS. Рассмотрим пример, изображенный на рис. 5.6 (EST, ECT, LST, LCT). Дано h цепочек требований $a_i \to b_i \to c_i \to e_i \to f_i$, $i = 1, 2, \ldots, h$. Для каждого требования b_i, e_i, $i = 1, 2, \ldots, h$, имеем $p_{b_i} = p_{e_i} = \varepsilon$, $q_{b_i} = q_{e_i} = h$. Для каждого требования a_i, f_i, $i = 1, 2, \ldots, h$, дано $p_{a_i} = p_{f_i} = 2\varepsilon(h - i + 1)$, $q_{a_i} = q_{f_i} = 1$. Для каждого требования c_i, $i = 1, 2, \ldots, h$, имеем $p_{c_i} = p - 4(i-1)\varepsilon$, $q_{c_i} = 1$, где $h^2\varepsilon \ll p$ и $Q_1 = h$.

Для этого примера оптимальное расписание соответствует перестановке $\pi = (a_1, a_2, \ldots, a_h, b_1, b_2, \ldots, b_h, c_1, c_2, \ldots, c_h, c_k, e_1, e_2, \ldots, e_h, f_1, f_2, \ldots, f_h)$. Алгоритмом LS с правилом ECT будет получена перестановка $\pi_{ECT} = (a_h, b_h, a_{h-1}, b_{h-1}, \ldots, a_1, b_1, c_h, e_h, f_h, c_{h-1}, e_{h-1}, f_{h-1}, \ldots, c_1, e_1, f_1)$, т.е. все требования $c_1, c_2, \ldots, c_h$ не обслуживаются параллельно ни в один из моментов времени (не перекрываются). Поэтому выполняется

$$\lim_{\varepsilon \to 0} \frac{C_{\max}(LS_{ECT})}{C^*_{\max}} = h = \frac{n}{5}.$$

Аналогично рассматривается перестановка $\pi_{LCT} = (a_1, b_1, c_1, e_1, f_1, \ldots, a_h, b_h, c_h, e_h, f_h)$ для правила LCT. □

Лемма 5.10 *Существует пример задачи RCPSP, для которого*

$$\frac{C_{\max}(LS_a)}{C^*_{\max}} = O(n),\ a \in \{MW, MSLK, GRR\}.$$

Доказательство. Рассмотрим пример, изображенный на рис. 5.6 (MW, MSLK, GRR). Дано h цепочек требований $a_i \to b_i \to c_i$, $i = 1, 2, \ldots, h$. Для каждого требования b_i, $i = 1, 2, \ldots, h$, имеем $p_{b_i} = p + (h - i)\varepsilon$, $q_{b_i} = 1 + i\varepsilon$. Для каждого требования a_i, c_i, $i = 1, 2, \ldots, h$, дано $p_{a_i} = p_{c_i} = i\varepsilon$, $q_{a_i} = q_{c_i} = h + 1$, где $\varepsilon < 1/h^2$ и $Q_1 = h + 1$.

Для этого примера оптимальное расписание соответствует перестановке $\pi = (a_1, a_2, \ldots, a_h, b_1, b_2, \ldots, b_h, c_1, c_2, \ldots, c_h)$. Алгоритмом LS с правилом MW будет получена перестановка $\pi_{MW} = (a_h, a_{h-1}, \ldots, a_1, b_h, c_h, b_{h-1}, c_{h-1}, \ldots, b_1, c_1)$, т.к. $d_{a_h} - r_{a_h} = \cdots = d_{a_1} - r_{a_1} = UB - (p + h\varepsilon)$, $d_{b_h} - r_{b_h} = UB - 2h\varepsilon < d_{b_{h-1}} - r_{b_{h-1}} = UB - 2(h-1)\varepsilon < \cdots < d_{b_1} - r_{b_1}$ и $d_{c_h} - r_{c_h} = UB - p - h\varepsilon < d_{b_{h-1}} - r_{b_{h-1}} <$

$\cdots < d_{b_1} - r_{b_1}$, и т.д. Поэтому все требования $b_1, b_2, \ldots, b_h$ не обслуживаются параллельно ни в один из моментов времени (не перекрываются). Поэтому выполняется

$$\lim_{\varepsilon \to 0} \frac{C_{\max}(LS_{MW})}{C^*_{\max}} = h = \frac{n}{3}.$$

Аналогично рассматривается перестановка
$\pi_{MSLK} = (a_h, b_h, c_h, a_{h-1}, b_{h-1}, c_{h-1}, \ldots, a_1, b_1, c_1)$, где $d_{a_i} - r_{a_i} - p_{a_i} = d_{b_i} - r_{b_i} - p_{b_i} = d_{c_i} - r_{c_i} - p_{c_i}$ для $i = 1, 2, \ldots, h$. Таким же образом рассматривается $\pi_{GRR} = (a_1, a_2, \ldots, a_h, b_h, c_h, b_{h-1}, c_{h-1}, \ldots, b_1, c_1)$. $\square$

Лемма 5.11 *Существует пример задачи RCPSP, для которого*

$$\frac{C_{\max}(LS_a)}{C^*_{\max}} = O(\sqrt{n}),\ a \in \{MIS, LIS\}.$$

Доказательство. Рассмотрим пример, изображенный на рис.5.7 (MIS, LIS). Дано h независисых множеств требований, каждое из которых содержит цепочку требований $b_i \to c_i$, и подмножество требований DB_i, DC_i, $i = 1, 2, \ldots, h$, где $b_i \to DB_i$ и $c_i \to DC_i$. Запись $j \to X_i$ означает, что для всех требований $l \in X_i$ выполняется $j \to l$. Дополнительно определим $|DB_i| = (i-1)$, $|DC_i| = i$ для $i = 1, 2, \ldots, h$. Для всех требований l из этих подмножеств пусть $p_l = \varepsilon$, $q_l = \varepsilon$.

Для каждого требования b_i, $i = 1, 2, \ldots, h$, имеем $p_{b_i} = p + (h-i)\varepsilon$, $q_{b_i} = 1$. Для каждого требования c_i, $i = 1, 2, \ldots, h$, имеем $p_{c_i} = \varepsilon$, $q_{c_i} = h$, где $h^2\varepsilon \ll p$ и $Q_1 = h$.

Как и прежде, можно убедиться, что при оптимальном расписании все требования b_i обслуживаются параллельно, в то время как при расписании π_{MIS}, эти требования не обслуживаются параллельно ни в один из моментов времени. Поэтому выполняется

$$\lim_{\varepsilon \to 0} \frac{C_{\max}(LS_{MIS})}{C^*_{\max}} = h.$$

Имеем $n = h + (2+4+6+\ldots+2h) = h + 2(1+2+3+\ldots+h) = h + h(h+1)$. Тогда $h = O(\sqrt{n})$.

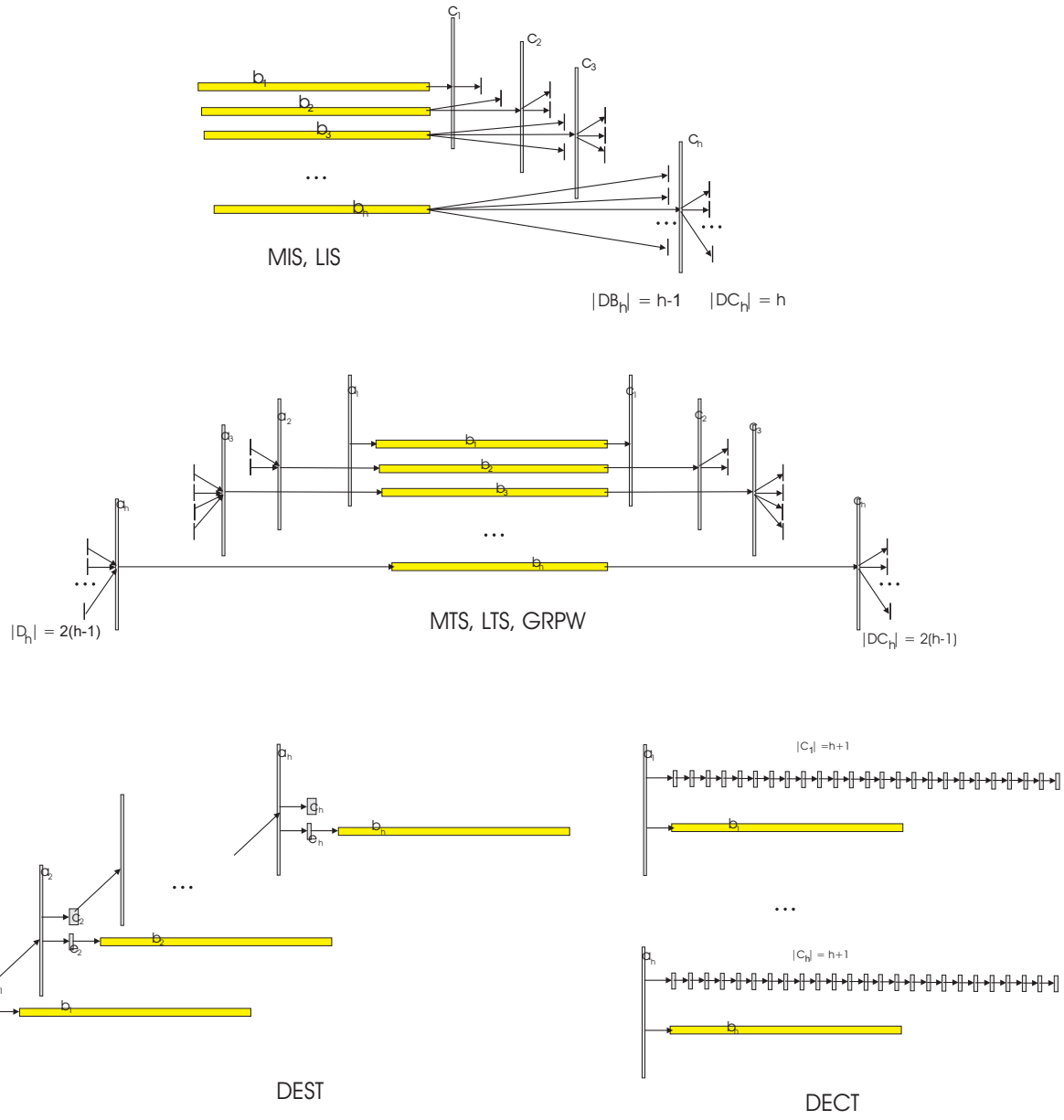

Рис. 5.7: Правила предпочтения. Часть 2

Аналогично рассматривается пример, в котором $p_{b_i} = p - (h-i)\varepsilon,\ i = 1, 2, \ldots, h$, и соответствующая перестановка π_{LIS}.

Очевидно, что для этого примера легко построить симметричный граф отношений предшествования (см. идею теоремы 5.4). $\square$

Лемма 5.12 *Существует пример задачи RCPSP, для которого*

$$\frac{C_{\max}(LS_a)}{C^*_{\max}} = O(\sqrt{n}),\ a \in \{MTS, LTS, GRPW\}.$$

Доказательство. Доказательство этой леммы аналогично доказательству леммы 5.11 . См. пример на рис. 5.7 (MTS, LTS, GRPW), где $|DC_i| = |D_i| = 2(i-1)$, $i = 1, 2, \ldots, h$. $\square$

Лемма 5.13 *Существует пример задачи RCPSP, для которого*

$$\frac{C_{\max}(LS_{DEST})}{C^*_{\max}} = O(n).$$

Доказательство. Здесь мы рассматриваем динамичную версию правила EST, т.е. значения r_i вычисляются перед каждой итерацией алгоритма LS. Здесь r_i – самый ранний возможный момент начала обслуживания требования i если учитывается время обслуживания других требований уже поставленных в расписание.

Рассмотрим пример, изображенный на рис. 5.7 (DEST). Дано дерево требований, которое содержит подмножества работ $\{a_i, b_i, c_i, e_i\}$, $i = 1, 2, \ldots, h$. Для каждого требования b_i, $i = 1, 2, \ldots, h$, имеем $p_{b_i} = p$, $q_{b_i} = 1$. Для каждого требования a_i, $i = 1, 2, \ldots, h$, дано $p_{a_i} = \varepsilon$, $q_{a_i} = h$. Для каждого требования c_i, $i = 1, 2, \ldots, h$, имеем $p_{c_i} = 2\varepsilon$, $q_{c_i} = \varepsilon$. Для каждого требования e_i, $i = 1, 2, \ldots, h$, дано $p_{e_i} = \varepsilon$, $q_{e_i} = \varepsilon$. Отношения предшествования изображены на рис. 5.7. Полагаем $h^2\varepsilon \ll p$ и $Q_1 = h$.

Для этого примера оптимальное расписание соответствует перестановке $\pi = (a_1, c_1, e_1, a_2, c_2, e_2, \ldots, a_h, c_h, e_h, b_1, b_2, \ldots, b_h)$. Алгоритмом LS с правилом DEST будет получена перестановка
$\pi_{DEST} = (a_1, c_1, e_1, b_1, \ldots, a_h, c_h, e_h, b_h)$, т.е. все требования $b_1, b_2, \ldots, b_h$ не

обслуживаются параллельно ни в один из моментов времени. Поэтому выполняется

$$\lim_{\varepsilon \to 0} \frac{C_{\max}(LS_{DEST})}{C^*_{\max}} = h = \frac{n}{4}.$$

Очевидно, что для этого примера легко построить симметричный граф отношений предшествования. □

Лемма 5.14 *Существует пример задачи RCPSP, для которого*

$$\frac{C_{\max}(LS_{DECT})}{C^*_{\max}} = O(\sqrt{n}).$$

Доказательство. Здесь мы рассматриваем динамичную версию правила ECT, т.е. значения $r_i + p_i$ вычисляются перед каждой итерацией алгоритма LS. Здесь $r_i + p_i$ – самый ранний возможный момент окончания обслуживания требования i если учитывается время обслуживания других требований уже поставленных в расписание.

Рассмотрим пример, изображенный на рис. 5.7 (DECT). Дано h независимых множеств требований таких, что множество с номером i содержит подмножества $\{a_i, b_i\} \bigcup C_i$, где $|C_i| = h + 1$, $C_i = \{j^i_1, \ldots, j^i_{h+1}\}$, $i = 1, 2, \ldots, h$. Для каждого требования b_i, $i = 1, 2, \ldots, h$, имеем $p_{b_i} = p$, $q_{b_i} = 1$. Для каждого требования a_i, $i = 1, 2, \ldots, h$, дано $p_{a_i} = 2\frac{p}{h}$, $q_{a_i} = h+1$. Для каждого требования $c \in C_i$, $i = 1, 2, \ldots, h$, имеем $p_c = \frac{p}{h}$, $q_c = \frac{1}{h}$. Пусть $Q_1 = h + 1$.

Для этого примера оптимальное расписание соответствует перестановке $\pi = (a_1, a_2, \ldots, a_h, b_1, b_2, \ldots, b_h, j^1_1, j^2_1, \ldots, j^h_1, \ldots, j^1_{h+1}, j^2_{h+1}, \ldots, j^h_{h+1})$. Алгоритмом LS с правилом DECT будет получена перестановка $\pi_{DECT} = (a_1, j^1_1, j^1_2, \ldots, j^1_{k+1}, b_1, a_2, \ldots, b_2, a_3, \ldots)$, т.е. все требования $b_1, b_2, \ldots, b_h$ не обслуживаются параллельно ни в один из моментов времени. Поэтому выполняется

$$\lim_{n \to \infty} \frac{C_{\max}(LS_{DECT})}{C^*_{\max}} = h.$$

Пусть $n = 2h + (h + 1)h$, т.е. выполняется $h = O(\sqrt{n})$. □

Теперь мы можем представить следующий обобщенный результат

Теорема 5.4 *Существует пример задачи RCPSP, для которого*

$$\frac{C_{\max}(LS_a)}{C^*_{\max}} \geqslant O(\sqrt{n})$$

для всех правил предпочтения a перечисленных выше.

Доказательство. Данный пример содержит подмножества требований $N_1, N_2, \ldots, N_{11}$. Каждое подмножество N_i, $i = 1, 2, \ldots, 11$, соответствует примерам из лемм 5.7–5.14 (некоторые леммы содержат по два примера). В каждом из подмножеств N_i, $i = 1, 2, \ldots, 11$, дано h требований с *длинными* продолжительностями обслуживания примерно равными p. Дополнительно задано 10 фиктивных требований $o_1, o_2, \ldots, o_{10}$ таких, что для всех $j \in N_i$ и для всех $l \in N_{i+1},\ i = 1, 2, \ldots, 10$, имеем $j \to o_i \to l$.

Тогда при оптимальном расписании длинные требования из одного подмножества обслуживаются параллельно. Если же мы используем любое из перечисленных правил предпочтения, найдется подмножество N_i, где все длинные требования не обслуживаются параллельно ни в один из моментов времени. □

5.7 Алгоритм Муравьиные Колонии для задачи RCPSP

Для решения задачи RCPSP в расширенной постановке на практике применяют эвристические или метаэвристические алгоритмы. В этом параграфе описан метаэвристический алгоритм решения, основанный на методе Муравьиные Колонии и алгоритме диспетчеризации LS. Этот итерационный алгоритм основан на идее последовательного приближения к оптимальному решению.

Каждая итерация – “запуск искусственного муравья”, который “пытается” по некоторому правилу выбрать наилучший маршрут к “пище” (к оптимуму функции), используя метки своих предшественников.

Каждый муравей выполняет алгоритм LS, выбирая требование $j \in EL$ исходя из значений параметров:

а) η_{ij} – эвристическая информация о том, насколько хорошим кажется постановка работы j на место i в перестановке $(j_1, j_2, \ldots, j_n)$ (напомним, что каждому активному расписанию соответствует некая перестановка). То есть насколько хорошим кажется выбор работы $j \in EL$ на i-ой итерации цикла. Этот параметр можно вычислить следующим образом:

1. По правилу d: $\eta_{ij} = \frac{1}{d_j},\ i = 1, \ldots, n$;
2. По правилу $CriticalPatch$ (критический путь): $\eta_{ij} = d_j - r_j,\ i = 1, \ldots, n$;

б) τ_{ij} – "след" (в природе: след феромона). После каждой итерации этот параметр корректируется. Параметр показывает насколько "хорошим" для работы j оказалась позиция i в перестановке $(j_1, j_2, \ldots, j_n)$. То есть это накопленная статистическая информация о качестве выбора для позиции i работы j, в то время как η_{ij} характеризует предполагаемую выгоду такой постановки при недостатке накопленной информации.

Параметры η_{ij} рассчитываются один раз перед первой итерацией.

На каждом шаге вычисляется **Матрица вероятностей перехода**:

$$\rho_{ij} = \begin{cases} \dfrac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum\limits_{h \in EL} [\tau_{ih}]^\alpha [\eta_{ih}]^\beta}, & j \in EL, \\ 0, & j \notin EL. \end{cases}$$

Правило, по которому на позицию i выбирается требование j определяется следующим образом:

$$\begin{cases} j = \arg\max\limits_{h \in EL} [\tau_{ih}]^\alpha [\eta_{ih}]^\beta, & q < q_0, \\ j \text{ определяется случайным образом,} \\ \text{согласно распределению вероятностей } \rho_{ij}, & q \geq q_0, \end{cases}$$

где $0 \leq q_0 \leq 1$ – параметр алгоритма, а значение q вычисляется случайным образом на каждом шаге.

После того, как работа j было поставлено на позицию i в перестановке, пересчитывается "локальный след":

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho\tau_0,$$

где $\tau_0, \rho \in [0, 1]$ – параметры алгоритма, которые задаёт пользователь.

После каждой итерации "глобальный след" τ_{ij} корректируется по правилу:

$$\tau_{ij} = (1 - \rho)\tau_{ij} + \rho/C^*_{\max},$$

если в "лучшем" найденном расписании на позиции i перестановки находится работа j. Иначе

$$\tau_{ij} = (1 - \rho)\tau_{ij},$$

Значение $C^*_{\max}$ – время выполнения проекта для "лучшего" найденного расписания.

Алгоритм ACO дает хорошие результаты при интеграции в него локального поиска, к примеру, попарной перестановки требований (если при этом не нарушаются отношения предшествования). Локальный поиск можно запускать после каждой итерации.

Подобный алгоритм реализован в программном продукте 1С: Управление строительной организацией и используется в сотнях отечественных строительных компаниях.

5.8 Частные случаи задачи RCPSP с одним ресурсом

В параграфе рассмотрены частные случаи задачи RCPSP с одним кумулятивным ресурсом $Q_1 = Q$. Для этих частных случаев приведены правила предпочтения, с помощью которых алгоритм LS строит допустимое решение с относительной погрешностью ограниченной константой. Приведены относительные погрешности некоторых нижних оценок.

5.8.1 Частный случай LSPP

В данном частном случае нет отношений предшествования, и он похож на ЗАДАЧУ УПАКОВКИ В ПОЛОСУ, которую можно сформулировать следующим образом.

Как и в популярной некогда компьютерной игре тетрис (созданной бывшим сотрудником Вычислительного Центра Российской Академии Наук), нам необходимо упаковать "кубики" в коробку. При этом пустых пространств между кубиками должно быть как можно меньше (т.е. мы должны минимизировать использованное пространство). Все "кубики" имеют форму прямоугольника, но различаются по длине и высоте Далее приводится более формальная постановка.

ЗАДАЧА УПАКОВКИ В ПОЛОСУ. Дано множество N из n прямоугольников. Для каждого прямоугольника $i \in N$ задана высота q_{i1} и длина p_i. Необходимо упаковать прямоугольники в бесконечную полосу высоты Q_1 так, чтобы прямоугольники не перекрывались, и использованная длина полосы была минимальной. Переворачивать прямоугольники запрещено.

Задача является NP-трудной в сильном смысле.

Такая задача часто возникает на производстве, где из листа металла необходимо вырезать прямоугольники разного размера, при этом отходы (неиспользованный металл между прямоугольниками) должны быть минимальны. Эту задачу в зарубежной литературе называют Strip packing problem, поэтому частный случай задачи RCPSP без отношений предшествования мы называем LSPP (like the Strip packing problem).

Может возникнуть ощущение, что две задачи эквивалентны. Однако это не так. Обозначим через SPP^* – оптимальную (минимальную) длину полосы в задаче SPP, а через $LSPP^*$ значение целевой функции в частном случае LSPP для примера, где требования имеют такие же параметры, как прямоугольники в задаче SPP. Известно [95], что существует примеры, на которых $LSPP^* < SPP^*$ (т.е. не всегда $LSPP^* = SPP^*$). Также известно, что выполняется следующее неравенство:

$$1 \leq \frac{SPP^*}{LSPP^*} \leq 2.7$$

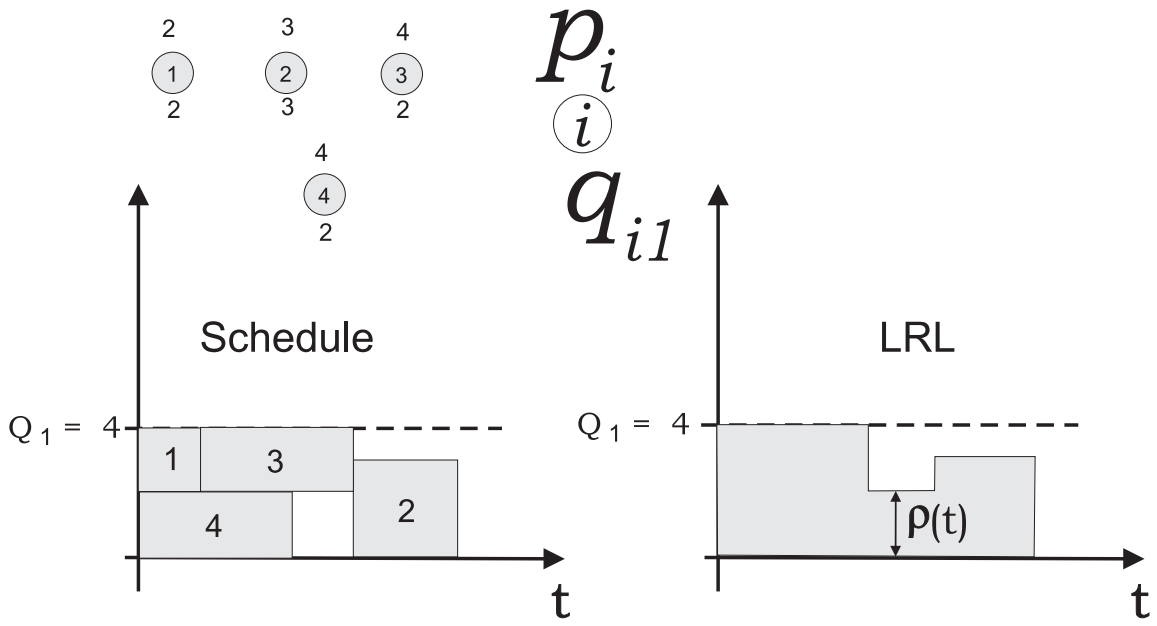

Рис. 5.8: Уровень загрузки ресурса $\rho(t)$

Для частного случая LSPP докажем, что

$$\frac{C_{\max}(LS_{DEST})}{C^*_{\max}} < 3.$$

Рассматриваем алгоритм LS с правилом предпочтения *DEST*, которое заключается в том, то на шаге 3 алгоритма мы выбираем требование i с наиболее ранним возможным моментом начала обслуживания, при котором не нарушаются отношения предшествования или ресурсные ограничения (с учетом уже поставленных в расписание требований).

Теорема 5.5 *Для частного случая LSPP выполняется*

$$\frac{C_{\max}(LS_{DEST})}{C^*_{\max}} < 3.$$

Доказательство. Пусть $\rho(t)$ (см. рис. 5.8) – уровень загрузки ресурса в момент времени t, т.е. $\rho(t) = \sum\limits_{i \in N_t} q_i$, где N_t – множество требований, обслуживаемых в момент времени t.

Обозначим $UB = C_{\max}(LS_{DEST})$. Пусть $[t_1, t_2) \in [0, UB)$ – интервал минимальной длинны, для которого выполняется: $\rho(t) > \frac{Q_1}{2}$ для всех $t \in [0, t_1)$, и $\rho(t) > \frac{Q_1}{2}$ для всех $t \in (t_2, UB)$.

Очевидно, что для расписания, построенного алгоритмом LS с правилом предпочтения DEST, мы имеем $t_2 - t_1 \leq p_{\max}$. Иначе, если $t_2 - t_1 > p_{\max}$, тогда существует требование $j \in N_{t_2}$ с $q_j \leq \frac{Q_1}{2}$ и $S_j > t_1$, т.е. получено противоречие, т.к. в соответствие с правилом предпочтения DEST, мы должны были поставить требование j на обслуживание с момента времени t_1. Значит, выполняется $t_2 - t_1 \leq p_{\max}$. Поэтому $UB - p_{\max} < 2 \cdot \frac{\sum\limits_{i=1}^{n} q_i p_i}{Q_1}$. Более того, выполняется

$$C^*_{\max} \geq \frac{\sum\limits_{i=1}^{n} q_i p_i}{Q_1}$$

и

$$C^*_{\max} \geq p_{\max}.$$

Значит $UB < 3C^*_{\max}$. □

Для данного частного случая выполняется $LB_0 = p_{\max}$ и $LB_1 = \frac{\sum\limits_{i=1}^{n} q_i p_i}{Q_1}$. А значит нижняя оценка $LB = \max\{LB_0, LB_1\}$ имеет относительную погрешность ограниченную числом 3, т.к. $LB \leq C^*_{\max} < UB < 3 \cdot LB$.

5.8.2 Частный случай UPT

Для данного частного случая все продолжительности обслуживания требований равны 1. Поэтому этот частный случай обозначается аббревиатурой UPT (unit processing times). Это NP-трудный в сильном смысле частный случай, к которому сводится ЗАДАЧА УПАКОВКИ В ПАЛЛЕТЫ (см. параграф 3.1. данной главы).

Без потери общности, пусть $Q_1 \in Z$ и $q_i \in Z,\ i = 1, 2, \ldots, n$. Для данной задачи можно представить следующий алгоритм вычисления верхней оценки. Мы разбиваем каждое требование $j = 1, 2, \ldots, n$, для которого $q_j > 1$, на несколько требований $j^1, j^2, \ldots, j^{q_j}$ с продолжительностями обслуживания 1 и $q_l = 1,\ l = j^1, j^2, \ldots, j^{q_j}$. Если существовало отношение предшествования $j \to i$, то полагаем $j^l \to i^e, l = 1, 2, \ldots, q_j,\ e = 1, 2, \ldots, q_i$. Обозначим $m = Q_1$.

Мы построили новый пример, соответствующий частному случаю PMS

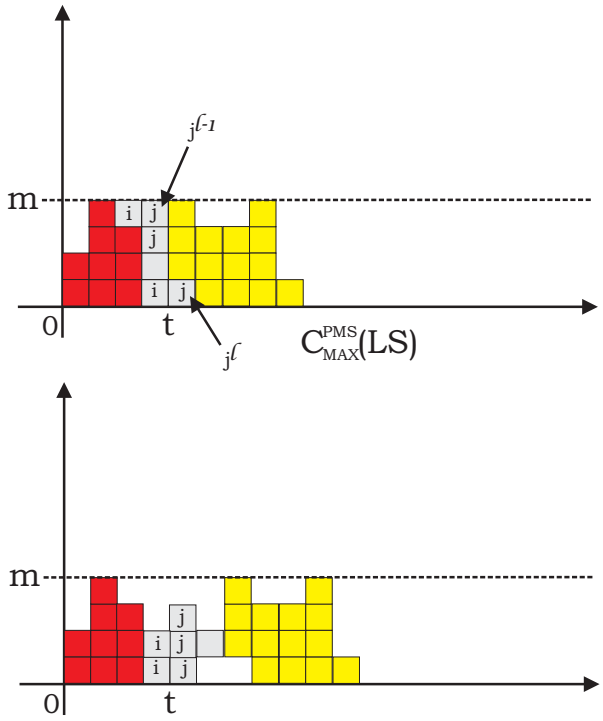

Рис. 5.9: Частный случай UPT, где $C_{\max}(LS_{EST})/C^*_{\max} < 4$

(см. параграф, посвященный частному случаю PMS). Тогда для данного примера выполняется $\frac{C_{\max}(LS_a)}{C^*_{\max}} < 2$ для любого правила предпочтения a.

Для данного примера мы строим допустимое расписание с помощью алгоритма LS, в котором на шаге 3 выбираем требование с наиболее ранним возможным моментом начала обслуживания r_j. Если таких требований несколько, то приоритет имеет требование j^l, если на предыдущей итерации мы выбрали требование j^{l-1}.

В полученном расписании может оказаться ситуация, при которой два требования j^{l-1} и j^l, $2 \le l \le q_j$, обслуживаются в моменты времени t и $t+1$ (см. рис. 5.9). Мы конструируем из этого расписания допустимое расписание для исходного примера UPT согласно трансформации, показанной на рис. 5.9. Более того, выполняется $2 \cdot C^{PMS}_{\max}(LS) > C^{UPT}_{\max}$, где $C^{PMS}_{\max}(LS)$ – значение целевой функции для примера PMS (второго примера), полученное с помощью алгоритма LS, а $C^{UPT}_{\max}$ – значение целевой функции для исходного примера, полученное согласно трансформации, изображенной на рис. 5.9. Последнее значение также может быть получено алгоритмом LS с правилом предпочтения EST, примененным к

исходному примеру. Обозначим через $C_{\max}^{PMS*}$ и $C_{\max}^{UPT*}$ – оптимальные значения целевых функций для модифицированного и исходного примеров. Тогда выполняется $C_{\max}^{PMS*} \leq C_{\max}^{UPT*}$ и

$$2 \cdot (2 \cdot C_{\max}^{PMS*}) > 2C_{\max}^{PMS}(LS) > C_{\max}^{UPT} \geq C_{\max}^{UPT*}.$$

Имеет место следующая теорема.

Теорема 5.6 *Для частного случая UPT выполняется*

$$\frac{C_{\max}(LS_{EST})}{C_{\max}^{*}} < 4.$$

Более того, выполняется неравенство

$$C_{\max}^{PMS}(LS) < \frac{\sum\limits_{i=1}^{n} q_i}{Q_1} + LB_0 \qquad [78],$$

Поэтому имеет место неравенство

$$C_{\max}^{UPT} < 2 \cdot C_{\max}^{PMS}(LS) < 2 \left(\frac{\sum\limits_{i=1}^{n} q_i}{Q_1} + LB_0 \right).$$

То есть для данного частного случая

$$C_{\max}^{*} < 2 \left(\frac{\sum\limits_{i=1}^{n} q_i}{Q_1} + LB_0 \right),$$

где $LB_1 = \frac{\sum\limits_{i=1}^{n} q_i}{Q_1}$ и LB_0 являются нижними оценками.

5.8.3 Частный случай PMS

Для данного частного случая $q_{j1} = 1$ и $Q_1 = m \in Z_+$. Этот частный случай рассматривается как отдельная задача и обозначается $Pm|prec|C_{\max}$.

Эта задача относится к разделу **TP** "параллельные машины", о которых мы говорили в первой главе.

Очевидно, что для решения этой задачи можно использовать алгоритм LS. Причем для некоторых правил предпочтения его относительная погрешность сравнительно небольшая.

Теорема 5.7 *Для частного случая PMS выполняется*

$$\frac{C_{\max}(LS_{DEST})}{C^*_{\max}} \leq 2.$$

Доказательство. Доказательство данной теоремы аналогично доказательству теоремы 5.5. Достаточно показать, что суммарная длина промежутков времени, где не все приборы заняты, меньше или равна длине критического пути (т.е. нижней оценке LB_0). Тогда можно будет доказать, что $C_{\max}(LS_{DEST}) \leq LB_0 + \frac{\sum p_j}{m} \leq 2C^*_{\max}$.

Пусть с помощью алгоритма LS с правилом DEST было получено допустимое расписание. Пусть j_1 – требование с наибольшим моментом окончания обслуживания при этом расписании, т.е. $C_{j_1} = C_{\max}$. Найдем промежуток времени $[t_1, t_2)$, такой, что $t_2 \leq S_{j_1} = C_{j_1} - p_{j_1}$, при котором хотя бы один прибор простаивает, и не существует другого промежутка времени $[t_3, t_4)$ такого, что $t_2 < t_4 \leq S_{j_1}$, т.е. $[t_1, t_2)$ – ближайший к требованию j_1 промежуток времени, при котором некоторый прибор простаивает. Тогда существует требование j_2, которое является прямым или косвенным предшественником требования j_1 (т.е. $j_2 \to i_1 \to i_2 \to \cdots \to i_l \to j_1$). Причем выполняется $C_{j_2} \geq t_2$. Предположим противное, что $C_{j_2} < t_2$, но тогда согласно правилу DEST требование j_1 должно было быть поставлено на обслуживание с момента времени C_{j_2}. Значит выполняется $C_{j_2} \geq t_2$.

Аналогично рассматриваем требование j_2, определяем для него ближайший промежуток простоя и доказываем, что в этом промежутке времени на каком то приборе обслуживается требование j_3, которое является прямым или косвенным предшественником требования j_2 и т.д.

В итоге получим цепочку требований $j_l \to j_{l-1} \to \cdots \to j_1$, причем все простои имеют место только в интервалы обслуживания этих тре-

бований (т.е. все простои параллельны обслуживанию этих требований). Очевидно, что $\sum_{i=1}^{l} p_{j_i} \leq LB_0$.

Длина всех остальных интервалов, где все приборы заняты, ограничена величиной $\frac{\sum_{j=1}^{n} p_j}{m} = LB_1$.

Таким образом, $C_{\max}(LS_{DEST}) \leq LB_0 + LB_1 \leq 2C^*_{\max}$, а значит теорема верна. $\square$

Для данной задачи можно представить несколько модифицированный алгоритм диспетчеризации LS1, смысл которого заключается в том, что при освобождении очередного прибора согласно некоторому правилу предпочтения выбирается "допустимое" требование, и начинается его обслуживание на приборе. Для данного алгоритма известно, что $\frac{C_{\max}(LS1_a)}{C^*_{\max}} < 2$ для любого правила предпочтения a, рассмотренных выше. Доказательство этого факта аналогично доказательству предыдущей теоремы.

Известно, что для данного частного случая не существует полиномиального алгоритма решения, относительная погрешность которого меньше $\frac{4}{3}$ (если $P \neq NP$). Это следует из доказательства NP-трудности в сильном смысле частного подслучая, где все продолжительности обслуживания равны 1, т.е. задачи $Pm|p_j = 1, prec|C_{\max}$. Доказательство это проведено сведением известной ЗАДАЧИ О КЛИКЕ к данному подслучаю. Причем показано, что любому примеру ЗАДАЧИ О КЛИКЕ с ответом "ДА", соответствует оптимальное расписание с $C^*_{\max} = 3$. Для ответа "НЕТ" – $C^*_{\max} = 4$. Поэтому имеет место относительная погрешность не меньше $\frac{4}{3}$.

ЗАДАЧА О КЛИКЕ. Задан неориентированный граф $G = (V, E)$. Вопрос, существует ли в этом графе КЛИКА мощности k, т.е. подграф, в котором между каждой парой вершин есть ребро, и количество вершин которого равно k.

Таблица 5.2: Сведение NP-трудных задач

Задача	**Сведение**
RCPSP с одним ресурсом	УПАКОВКА В ПАЛЛЕТЫ (NP-трудная в сил. смысле) ЗАДАЧА О КЛИКЕ (NP-трудная в сил. смысле) ЗАДАЧА РАЗБИЕНИЯ (NP-трудная в обыч. смысле) ВЕРШИННОЕ ПОКРЫТИЕ (NP-трудная в сил. смысле)
PMS	ЗАДАЧА О КЛИКЕ (Если $p_j = 1$) ЗАДАЧА РАЗБИЕНИЯ (Если $m = 2$)
LSPP	УПАКОВКА В ПАЛЛЕТЫ (Если $p_j = 1$) ЗАДАЧА РАЗБИЕНИЯ (Если $q_j = 1$)
UPT	УПАКОВКА В ПАЛЛЕТЫ ЗАДАЧА О КЛИКЕ (Если $q_j = 1$)

5.9 Сложности приближенного решения задачи RCPSP

Возникает вопрос, почему мы имеем *хорошие* нижние и верхние оценки для частных случаев задачи RCPSP[1] (UPT, LSPP, PMS), но не можем найти такие оценки для общего случая. Можно попробовать доказать, что задача RCPSP с одним куммулятивным ресурсом не принадлежит классу APX, но это представляется нетривиальным. Известно, что в общем случае задача RCPSP с *большим* количеством типов ресурсов [104] не принадлежит классу APX, но такая ситуация не встречается на практике.

Можно предположить, что частный случай RCPSP с одним ресурсом настолько сложен, т.к. он "содержит" в себе четыре разные NP-трудные задачи в то время, как частные подслучаи содержат только по две из них (см. таб. 5.2).

В прошлых параграфах мы показали, что для следующего частного слу-

[1]В этом параграфе рассматривается случай задачи RCPSP с одним куммулятивным ресурсом мощности Q_1

чая задачи RCPSP известные нижние и верхние оценки имеют относительную погрешность, неограниченную константой. В этом частном случае дано два множества требований – множество *высоких* требований N_{high}, для которых $p_j = \varepsilon,\ q_j = n$, и множество *длинных* требований N_{long} с произвольной продолжительностью обслуживания и $q_j = 1$. Причем обычно не заданы прямые отношения предшествования между требованиями из N_{long}, а заданы отношения предшествования вида $i \to k \to j$, где $i, j \in N_{long}$ и $k \in N_{high}$.

Назовем этот частный случай *batch-случай задачи RCPSP*. Далее мы покажем, что этот частный случай соответствует NP-трудной в сильном смысле задаче $1|p - batch, chains, b = \infty|C_{\max}$, которая определена следующим образом

Задача $1|p - batch, prec, b = \infty|C_{\max}$**:**

Дано множество требований $N = \{1, 2, \ldots, n\}$. Для каждого требования $j \in N$ задана продолжительность обслуживания $p_j \geq 0$. Прерывания обслуживания требований запрещены. Заданы отношения предшествования $i \to j$ между требованиями.

Требования обслуживаются партиями, где партия – подмножество требований, объединенных по некоторому признаку. Продолжительность обслуживания партии равно максимальной продолжительности обслуживания среди требований партии. Время завершения обслуживания всех требований из партии равно времени завершения обслуживания партии. Если заданы отношение предшествования между требованиями i и j, тогда эти требования не могут обслуживаться в одной и той же партии. Более того, если $i \to j$, тогда обслуживание партии, содержащей требование j, предшествует обслуживанию партии, содержащей требование i. Прибор может обслуживать только одну партию в каждый момент времени.

Известно, что задача $1|p-batch, chains, b = \infty|C_{\max}$ с отношениями предшествования в виде цепочек является NP-трудной в сильном смысле, что доказано сведением к ней задачи ВЕРШИННОЕ ПОКРЫТИЕ.

Теорема 5.8 *Batch-случай задачи RCPSP является NP-трудным в сильном смысле.*

Доказательство. Дан пример задачи $1|p-batch, prec, b=\infty|C_{\max}$. Построим пример batch-случая задачи RCPSP. Требования $1,2,\ldots,n$ исходного примера соответствуют требованиям $1,2,\ldots,n$ примера задачи RCPSP. Продолжительности обслуживания теже. Принимаем $q_j=1$ для $j=1,2,\ldots,n$. Дополнительно задаем подмножество требований N_{high}. Если в исходном примере $i\to j,\ i,j\in N$, тогда мы создаем *высокое* требование k_{ij} с продолжительностью обслуживания $p_{k_{ij}}=1$ и $q_{k_{ij}}=n$, и отношениями предшествования $i\to k_{ij}\to j,\ i,j\in N_{long}$. Полагаем $Q=n$.

Если $C^{p*}_{\max}$ – оптимальное значение функции для исходного примера, тогда $C^{*}_{\max}=C^{p*}_{\max}+|E|$ – оптимальное значение целевой функции для примера batch-случая, где $|E|$ – количество отношений предшествования в исходном примере. □

5.10 Планарность сетевого графика для задач RCPSP и PMS

Два примера задачи RCPSP будем называть "аналогичными", если первый пример сводится ко второму введением "пустых" требований (с нулевой продолжительностью) и удалением "лишних" связей, так что если между вершинами i и j в сетевом графике был хотя бы один путь, то путь между ними останется. Если пути не было, то он не появится. Очевидно, что значения целевой функции для аналогичных примеров равны.

Теорема 5.9 *Любой пример задачи RCPSP можно преобразовать в аналогичный пример, граф отношений предшествования которого будет планарным.*

Доказательство. Согласно теореме Понтрягина–Куратовского, граф является непланарным тогда и только тогда, когда граф содержит подграфы, гомеоморфные $K_{3,3}$ или K_5. В следующих двух леммах показано, как можно "избавиться" от ситуаций $K_{3,3}$ и K_5 в исходном графе введением порядка $O(n^2)$ "пустых" требований и удалением "лишних" связей.

Лемма 5.15 *Если в графе отношений предшествования встречается*

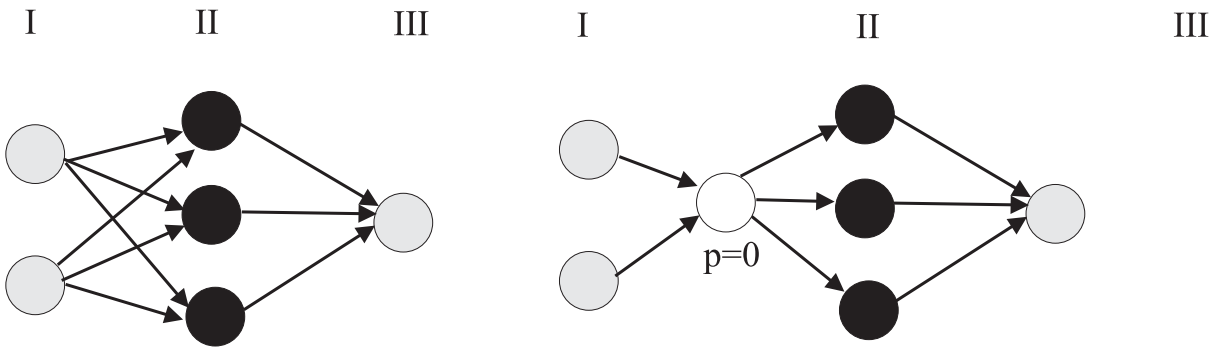

Рис. 5.10: Преобразование графа с фрагментом $K_{3,3}$

подграф $K_{3,3}$, то можно его преобразовать удаляя лишние дуги и добавляя "пустые" вершины.

Доказательство. Пусть имеется подграф вида $K_{3,3}$. То есть существует двудольный граф состоящий из $2k$ вершин. Выделим в графе предшествования "черные" и "серые" вершины. Вершины одного цвета не связаны друг с другом дугами.

Представим двудольный граф в виде сетевого графика. Выделим в сетевом графике уровни $l = I, II, \ldots$ (см. рис. 5.10). Требования принадлежат одному уровню, если соответствующие значения r_j равны. В двух соседних уровнях не могут быть требования одинакового цвета, т.к. "черные" требования не связаны между собой отношениями предшествования (аналогично "серые").

Если уровней больше трех, то некоторые связи излишни (см. рис. 5.12). Добавлением пустых требований и удалением лишних связей (см. рис. 5.10, 5.11, 5.12) "избавимся" от подграфа $K_{3,3}$. □

Лемма 5.16 *Если в графе отношений предшествования встречается подграф K_5, то можно его преобразовать удаляя лишние дуги.*

Доказательство. Пусть имеется подграф вида K_5. Очевидно, что данный граф можно представить в виде сетевого графика единственным способом (см. рис. 5.13). В нем легко выявить и удалить "лишние" связи. □

Основываясь на этих двух леммах можно утверждать, что теорема 5.9 верна. □

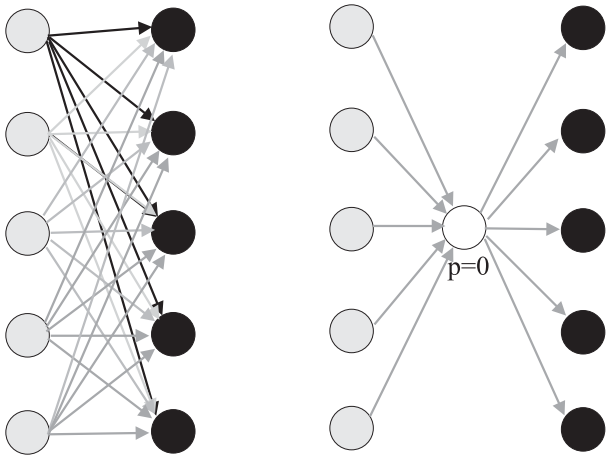

Рис. 5.11: Преобразование графа с фрагментом $K_{3,3}$

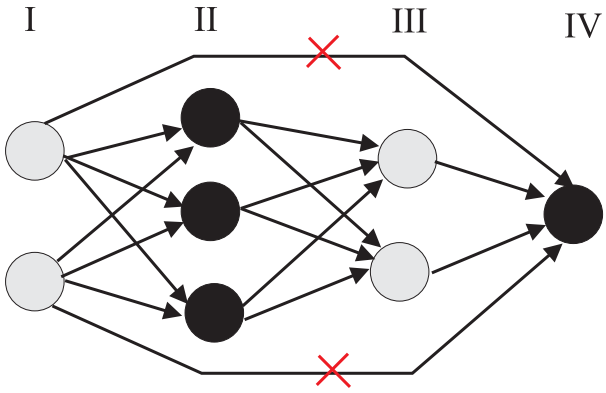

Рис. 5.12: Преобразование графа с фрагментом $K_{3,3}$

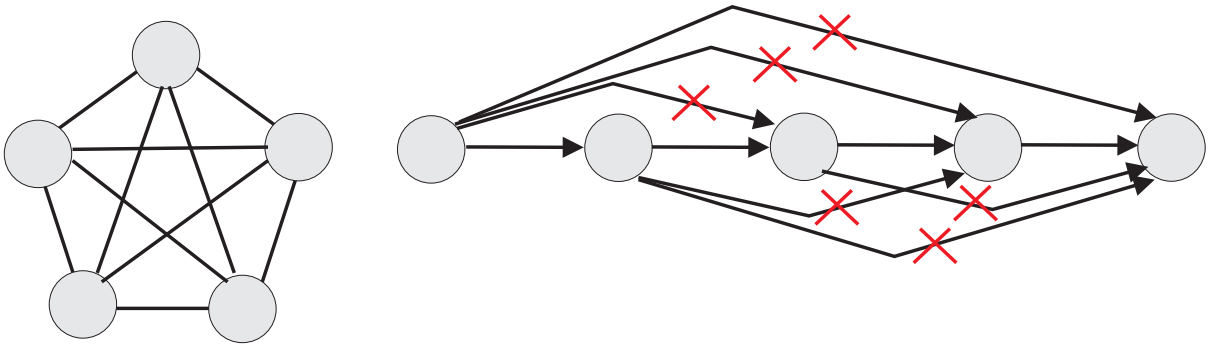

Рис. 5.13: Преобразование графа с фрагментом K_5

Теорема 5.10 *Для любого примера задачи RCPSP с n требованиями и v связями существует аналогичный пример с планарным графом отношений предшествования, n' требованиями и v' связями, причем $n+v \geq n'+v'$.*

Доказательство. В процессе преобразования исходного примера к примеру с планарным графом для каждого подграфа $K_{3,3}$ мы добавляли одно "пустое" требование и уменьшали количество связей минимум на единицу (см. рис. 5.10). Для каждого подграфа K_5 мы только удаляли "лишние" дуги (см. рис. 5.13). Таким образом для преобразованного примера выполняется $n + v \geq n' + v'$. □

Алгоритм укладки сетевого графика на плоскости

На практике существует потребность представить сетевой график по ранее введенному проекту в наиболее "удобном" для пользователя виде. В наиболее популярных программах (MS Project, Spyder) сетевые графики представляются неудобно – дуги пересекаются или сливаются. В данном параграфе представлен алгоритм укладки "планарного" сетевого графика на плоскости без пересечения дуг.

В работе [7] представлен известный алгоритм укладки на плоскости планарного графа. Важное отличие сетевого графика от ориентированного графа заключается в том, что вершины сетевого графика расположены вдоль некоторой виртуальной временной оси (обычно горизонтальной). Причем требование-предшественник изображается всегда "левее" своих последователей.

Кратко опишем алгоритм укладки планарного графа на плоскости и покажем как его можно преобразовать для укладки "планарного" сетевого графика.

Алгоритм γ [7] укладки графа G представляет собой процесс последовательного присоединения к некоторому уложенному подграфу G' графа G новой цепи, оба конца которой принадлежат G'. Тем самым эта цепь разбивает одну из граней графа G' на две. В качестве начального плоского графа G' выбирается любой простой цикл графа G. Процесс продолжается до тех пор, пока не будет построен плоский граф изоморфный

графу G, или присоединение некоторой цепи окажется невозможным. В последнем случае граф G не является планарным.

Введем ряд определений. Пусть построена некоторая укладка подграфа G' графа G. Сегментом S относительно G' (или просто сегментом) будем называть подграф графа G одного из следующих двух видов:

1. Ребро $e = uv \in EG$ такое, что $e \notin EG', u, v \in VG'$, где EG – множество ребер графа G, VG – множество вершин графа G, аналогично EG' – множество ребер подграфа G', VG' – множество вершин подграфа G';
2. Связную компоненту графа $G - G'$, дополненную всеми ребрами графа G, инцидентными вершинам взятой компоненты и концам этих ребер.

Вершина v сегмента S будем называть *контактной*, если $v \in VG'$.

Допустимой гранью для сегмента S называется грань Γ графа G', содержащая все контактные вершины сегмента S.

$\Gamma(S)$ – множество допустимых граней для S. Простую цепь сегмента S, соединяющую две различные контактные вершины и не содержащую других контактных вершин, назовем *α-цепью*.

Два сегмента S_1 и S_2 относительно G' назовем *конфликтующими*, если:

1. $\Gamma(S_1) \cap \Gamma(S_2) \neq \emptyset$;
2. Существуют две α-цепи $L_1 \in S_1$ и $L_2 \in S_2$, которые без пересечений нельзя уложить одновременно ни в какую грань.

В работе [7] показано, что для продолжения алгоритма γ можно выбирать любую α-цепь в любом сегменте и помещать ее в любую допустимую грань.

На рис. 5.14.1 представлен сетевой граф с пересечением дуг. На рис. 5.14.2 изображена начальная цепь (граф G'), его грани и сегменты S_1, S_2. Сегменты конфликтующие.

В алгоритме укладки сетевого графика выбор грани для очередной α-цепи желательно производить согласно правилу – предшественник дол-

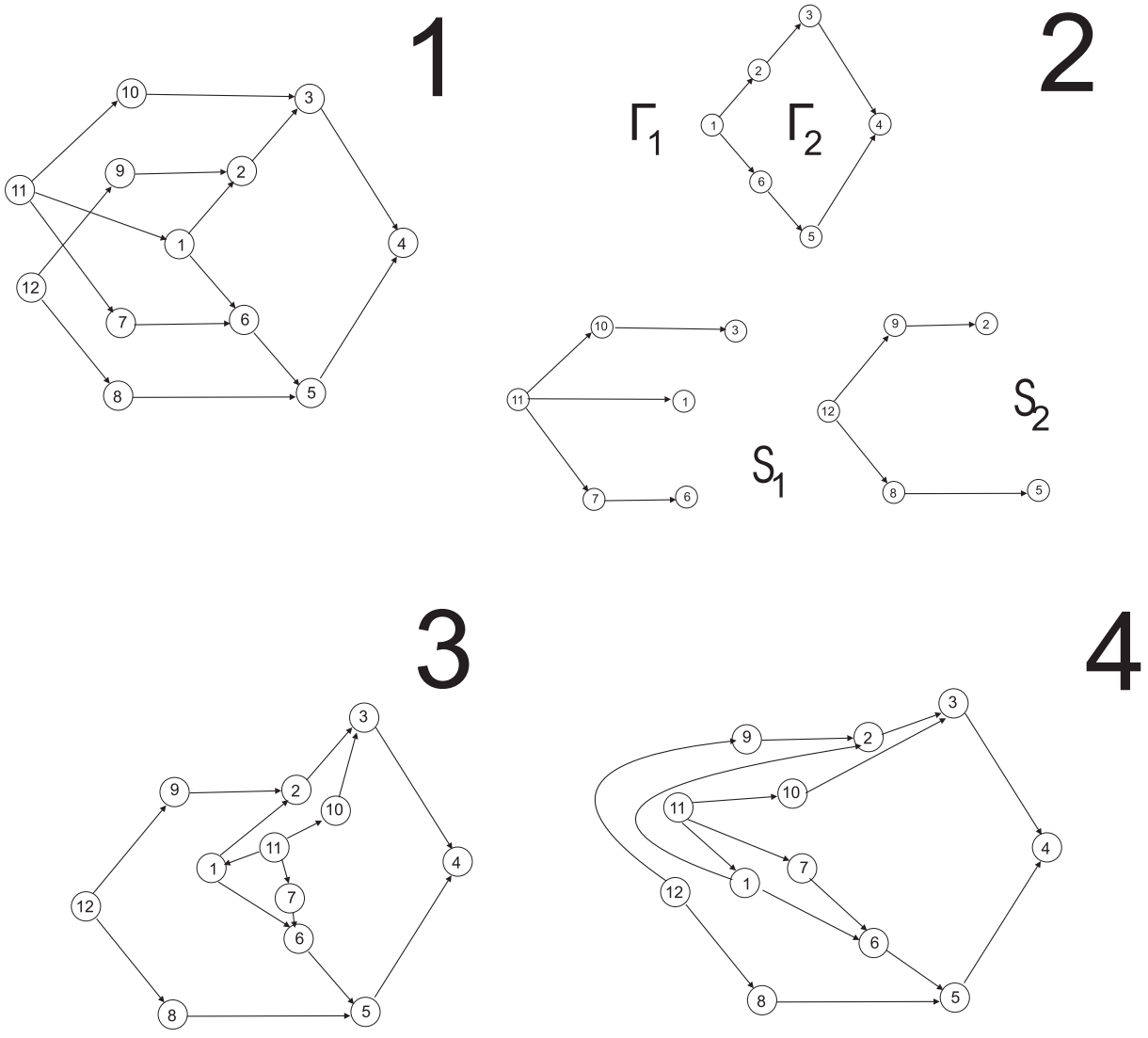

Рис. 5.14: Укладка сетевого графика на плоскости

жен находится левее последователя. Таким образом, при укладке α-цепи, соответствующей сегменту S_2 мы выбрали грань Γ_1, а для укладки S_1 – оставшуюся грань Γ_2.

Укладка соответствующего графа представлена на рис. 5.14.3.

Заметим, что при такой укладке вершина-последователь 1 находится левее вершины-предшественника 11, что неправильно для сетевого графика. Полученную укладку можно "растянуть" как представлено на рис. 5.14.4. Причем ребро $(1, 2)$ станет дугообразным.

Зачастую, полученное представление "без пересечений" предпочтительней преобразовать, допустив пересечения. Такие преобразования лучше производить над полученной "плоской" укладкой, чтобы минимизировать количество пересечений.

Итак, при описании проектов на практике можно дать следующую рекомендацию: **Сетевой график проекта должен быть планарным**. Действительно, в этом случае размерность задачи $n + v$ не увеличивается. Согласно теореме Эйлера количество ребер в планарном графе не превышает $3n - 6$, что можно учитывать при оценке трудоемкости алгоритмов.

Более того, планарный сетевой график можно "удобно" представить на плоскости, что полезно на практике.

Библиографическая справка

Регулярно выпускаются интегрированные обзоры наиболее значимых результатов по задаче RCPSP, к примеру, [72]. Для тестирования и сравнения многочисленных алгоритмов решения задачи RCPSP создана библиотека тестовых примеров PSPLIB [73]. Наиболее "быстрым" на данный момент точным алгоритмом признан алгоритм ветвей и границ Брукера и др. [43]. Среди метаэвристических алгоритмов стоит выделить алгоритм, описанный в работе [85].

Экспериментальный анализ некоторых эвристических алгоритмов приведен в [66] и [73].

Точный алгоритм решения задачи RCPSP с прерываниями предложен в работе [50].

Глава 6

Приложения

Доказательство NP-трудности задачи $1||\sum T_j$

ЗАДАЧА ЧЕТНО-НЕЧЕТНОГО РАЗБИЕНИЯ.

Задано упорядоченное множество из $2n$ положительных целых чисел $B = \{b_1, b_2, \ldots, b_{2n}\}$, $b_i > b_{i+1}$, $1 \leq i \leq 2n-1$. Требуется определить, существует ли разбиение множества B на два подмножества B_1 и B_2, такое, что $\sum_{b_i \in B_1} b_i = \sum_{b_i \in B_2} b_i$ и для каждого $i = 1, 2, \ldots, n$ подмножество B_1 (следовательно, и B_2) содержит в точности один элемент из пары $\{b_{2i-1}, b_{2i}\}$.

Обозначим $\delta_i = b_{2i-1} - b_{2i},\ i = 1, \ldots, n$, $\delta = \sum_{i=1}^{n} \delta_i$.

Построим модифицированный пример ЧНР.

$$\begin{cases} a_{2n} = M + b, \\ a_{2i} = a_{2i+2} + b,\ i = n-1, \ldots, 1, \\ a_{2i-1} = a_{2i} + \delta_i,\ i = n, \ldots, 1, \end{cases}$$

где $b \gg n\delta,\ M \geq n^3 b$.

Очевидно, выполняется $a_i > a_{i+1},\ \forall i = 1, 2, \ldots, 2n-1$, $\delta_i = b_{2i-1} - b_{2i} = a_{2i-1} - a_{2i},\ i = 1, \ldots, n$.

Лемма 6.1 *Исходный пример* **ЧНР** *имеет решение "ДА" тогда и только тогда, когда модифицированный пример* **ЧНР** *имеет решение "ДА".*

Доказательство. Пусть для исходного примера существует два подмножества B_1 и B_2, таких что $\sum_{b_i \in B_1} b_i = \sum_{b_i \in B_2} b_i$. Возьмчм $A_1 = \{a_i | b_i \in B_1\},\ A_2 = \{a_i | b_i \in B_2\}$. Тогда выполняется $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A_2} a_i$.

Допустим, что для модифицированного примера имеется два подмножества A_1 и A_2, таких что $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A_2} a_i$. Возьмчм $B_1 = \{b_i | a_i \in A_1\},\ B_2 = \{b_i | a_i \in A_2\}$. Очевидно, $\sum_{b_i \in B_1} b_i = \sum_{b_i \in B_2} b_i$. □

Рассмотрим частный случай **B-1** задачи $1 \,||\, \sum T_j$:

$$\begin{cases} p_1 \geq p_2 \geq \cdots \geq p_n, \\ d_1 \leq d_2 \leq \cdots \leq d_n, \\ d_n - d_1 \leq p_n. \end{cases} \tag{2.1}$$

Введем необходимые определения.

Расписание $\pi = (j_1, j_2, \ldots, j_n)$ будем называть *SPT-расписанием* (short processing time), если $p_{j_k} \leq p_{j_{i+k}}$, для $p_{j_k} = p_{j_{k+1}}$ выполняется $d_{j_k} \leq d_{j_{k+1}}$, $k = 1, 2, \ldots, n-1$.

Расписание $\pi = (j_1, j_2, \ldots, j_n)$ будем называть *EDD-расписанием* (early due date), если $d_{j_k} \leq d_{j_{k+1}}$, для $d_{j_k} = d_{j_{k+1}}$ выполняется $p_{j_i} \leq p_{j_{k+1}}$, $k = 1, 2, \ldots, n-1$.

Для случая (2.1) расписание $\pi = (1, 2, \ldots, n)$ является EDD- расписанием. Расписание $\pi = (n, n-1, \ldots, 1)$ – SPT- расписанием.

Расписание π' будем называть *частичным расписанием*, если при нчм обслуживается некоторое подмножество требований $N' \subset N$. Обозначим $P(N') = \sum_{i \in N'} p_i$, $\{\pi'\} = N'$, $P(\pi') = \sum_{i \in \{\pi'\}} p_i$.

Лемма 6.2 [17] *Для случая (2.1) существует оптимальное расписание вида $\pi^* = (\pi_{EDD}, l, \pi_{SPT})$, где l – первое запаздывающее требование при расписании π^*, π_{EDD} и π_{SPT} – частичные расписания, построенные по правилам EDD и SPT соответственно.*

Следствие. Для случая (2.1) запаздывающие требования в оптимальном расписании расположены в порядке SPT, кроме, быть может, первого запаздывающего требования.

Приведчм полиномиальную схему сведения модифицированного примера **ЧНР** к частному случаю (2.1) задачи $\mathbf{1}\,||\,\sum \boldsymbol{T_j}$. Количество требований $(2n+1)$. Требования обозначим следующим образом
$V_1, V_2, V_3, V_4, \ldots, V_{2i-1}, V_{2i}, \ldots, V_{2n-1}, V_{2n}, V_{2n+1}$, $N = \{1, 2, \ldots, 2n, 2n+1\}$. Для упрощения записи будем использовать величины $p_{V_i} = p_i,\ d_{V_i} = d_i,\ T_{V_i} = T_i,\ C_{V_i} = C_i,\ i = 1, \ldots, 2n+1$. Пример, удовлетворяющий следующим ограничениям, назовем *каноническим LG-примером.*

$$\begin{cases} p_1 > p_2 > \cdots > p_{2n+1}, \\ d_1 < d_2 < \cdots < d_{2n+1}, \\ d_{2n+1} - d_1 < p_{2n+1}, \\ p_{2n+1} = M = n^3 b, \\ p_{2n} = p_{2n+1} + b = a_{2n}, \\ p_{2i} = p_{2i+2} + b = a_{2i},\ i = n-1, \ldots, 1, \\ p_{2i-1} = p_{2i} + \delta_i = a_{2i-1},\ i = n, \ldots, 1, \\ d_{2n+1} = \sum_{i:=1}^{n} p_{2i} + p_{2n+1} + \frac{1}{2}\delta, \\ d_{2n} = d_{2n+1} - \delta, \\ d_{2i} = d_{2i+2} - (n-i)b + \delta,\ i = n-1, \ldots, 1, \\ d_{2i-1} = d_{2i} - (n-i)\delta_i - \varepsilon\delta_i,\ i = n, \ldots, 1, \end{cases} \tag{2.2}$$

где $b = n^2\delta$, $0 < \varepsilon < \frac{\min_i \delta_i}{\max_i \delta_i}$.

Директивные сроки примера (2.2) представлены на рис. 6.1.

Обозначим $L = \frac{1}{2}\sum_{i:=1}^{2n} p_i$, тогда $d_{2n+1} = L + p_{2n+1}$, так как $\frac{1}{2}\sum_{i:=1}^{2n} p_i = \sum_{i:=1}^{n} p_{2i} + \frac{1}{2}\delta$.

Необходимо отметить, что канонические примеры DL из [53] не удовлетворяют случаю (2.2). Первые три неравенства показывают, что (2.2) является подслучаем (2.1).

Свойства примеров случая (2.2) задачи $1||\sum T_j$**.**

Сформулируем следующую лемму.

Лемма 6.3 *Для (2.2) при любом расписании количество запаздывающих требований равно или n, или n + 1.*

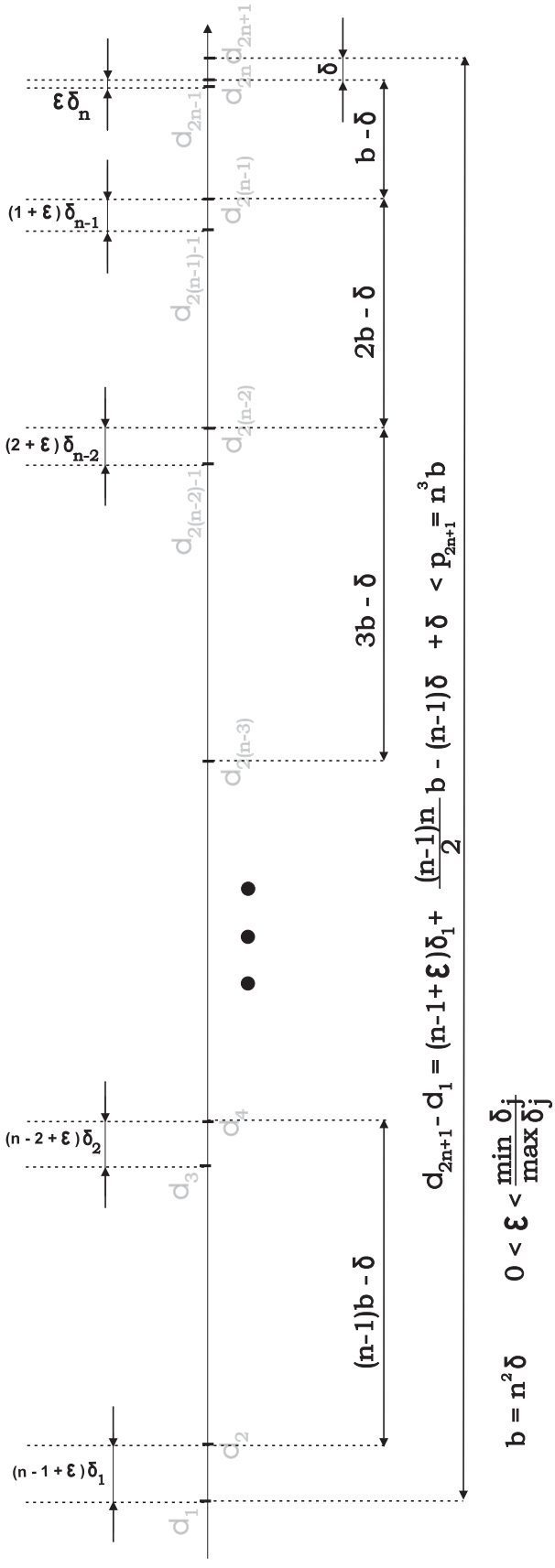

Рис. 6.1: Директивные сроки канонического LG примера.

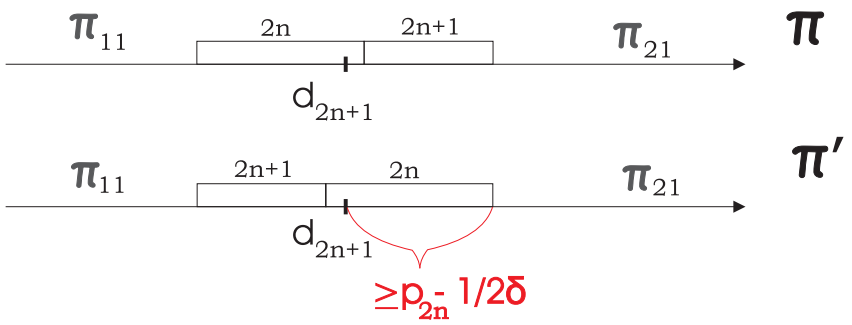

Рис. 6.2: Перестановка требований V_{2n} и V_{2n+1}.

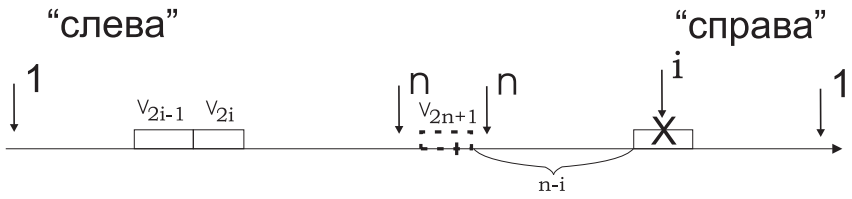

Рис. 6.3: Перестановка в неканоническом расписании.

Доказательство. Доказательство проведем в два этапа.

1. Рассмотрим подмножество требований N' состоящее из $(n+2)$ самых коротких по продолжительности обслуживания требований и упорядочим их вначале расписания. Очевидно, что $\sum_{i\in N'} p_i > (n+2)p_{min} = (n+2)n^3b$, где $p_{min} = \min_{j\in N}\{p_j\} = p_{2n+1}$.

   Из условий 4-8 в (2.2) найдчм

$$d_{max} = \max_{j\in N}\{d_j\} = d_{2n+1} = (n+1)n^3b + (b+2b+\cdots+nb) + \frac{1}{2}\delta,$$

   для которого будет выполняться

$$d_{max} = d_{2n+1} = (n+1)n^3b + \frac{n(n+1)}{2}b + \frac{1}{2}\delta < (n+2)n^3b < \sum_{i\in N'} p_i,$$

   т.е. при любом расписании требование, обслуживаемое $n+2$ по порядку и все последующие будут запаздывать, так как согласно третьему неравенству в (2.2), разность между директивными сроками

любых двух требований меньше продолжительности обслуживания каждого требования. Таким образом, при любом расписании π количество незапаздывающих требований не превышает $n+1$.

2. Рассмотрим подмножество N'', состоящее из n самых длинных по продолжительности обслуживания требований, и упорядочим их вначале расписания. Возможны два случая:

   a) пусть $n = 2k$, тогда $N'' = \{V_1, V_2, \ldots, V_{2k-1}, V_{2k}\}$, будем иметь

$$P(N'') = nn^3b + 2(nb + (n-1)b + \cdots + (n-k+1)b) + \sum_{i:=1}^{k} \delta_i,$$

$$P(N'') = nn^3b + 2(\frac{n(n+1)}{2} - \frac{(n-k)(n-k+1)}{2})b + \sum_{i:=1}^{k} \delta_i.$$

   Из условий 8-11 в (2.2) будем иметь
   $d_{min} = \min_{j \in N}\{d_j\} = d_1 = d_{2n+1} - (\sum_{i:=1}^{n-1}((n-i)b - \delta) + \delta + (n-1)\delta_1 - \varepsilon\delta_1) = (n+1)n^3b + (b + 2b + \cdots + nb) + \frac{1}{2}\delta - (\sum_{i:=1}^{n-1}((n-i)b - \delta) + \delta + (n-1)\delta_1 + \varepsilon\delta_1) > P(N'')$;

   b) пусть $n = 2k+1$, тогда $N'' = \{V_1, V_2, \ldots, V_{2k-1}, V_{2k}, V_{2(k+1)-1}\}$.

$$P(N'') = nn^3b + 2(nb + (n-1)b + \cdots + (n-k+1)b) + (n-k)b + \sum_{i:=1}^{k+1} \delta_i,$$

$$P(N'') = nn^3b + 2(\frac{n(n+1)}{2} - \frac{(n-k)(n-k+1)}{2})b + (n-k)b + \sum_{i:=1}^{k+1} \delta_i,$$

   $d_{min} = d_1 = d_{2n+1} - (\sum_{i:=1}^{n-1}((n-i)b - \delta) + \delta + (n-1)\delta_1 - \varepsilon\delta_1) = (n+1)n^3b + (b + 2b + \cdots + nb) + \frac{1}{2}\delta - (\sum_{i:=1}^{n-1}((n-i)b - \delta) + \delta + (n-1)\delta_1 + \varepsilon\delta_1) > P(N'')$,

т.е. при любом расписании первые n требований не будут запаздывать. Таким образом, при любом расписании π количество незапаздывающих требований больше или равно n.

Следовательно, для случая (2.2) при любом расписании количество запаздывающих требований равно или n, или $n+1$.

□

Лемма 6.4 *Для случая (2.2) для каждого расписания* $\pi = (\pi_1, \pi_2)$ *существует расписание вида* $\pi' = (\pi_{EDD}, \pi_{SPT})$, *где* $\{\pi_1\} = \{\pi_{EDD}\},\ \{\pi_2\} = \{\pi_{SPT}\},\ |\{\pi_1\}| = n+1,\ |\{\pi_2\}| = n$. *Приччм,* $F(\pi) \geq F(\pi')$.

Доказательство.

Рассмотрим частичное расписание π_1. Так как n первых требований при π_1 не запаздывают, а может запаздывать лишь последнее, то EDD порядок обслуживания требований множества $\{\pi_1\}$ будет оптимальным для него. В этом случае на $n+1$ месте будет обслуживаться требование $j = argmax\{d_i : i \in \{\pi_1\}\}$.

Рассмотрим частичное расписание π_2. Так как все n требований при π_2 запаздывают, то порядок SPT обслуживания требований множества $\{\pi_2\}$ будет оптимальным. □

Расписание $(V_{1,1}, V_{2,1}, \ldots, V_{i,1}, \ldots, V_{n,1}, V_{2n+1}, V_{n,2}, \ldots, V_{i,2}, \ldots, V_{2,2}, V_{1,2})$ будем называть *каноническим LG-расписанием*, где $\{V_{i,1}, V_{i,2}\} = \{V_{2i-1}, V_{2i}\},\ i = 1, 2, \ldots, n$.

Лемма 6.5 *Если расписание* $\pi = (\pi_1, \pi_2),\ |\{\pi_1\}| = (n+1),\ |\{\pi_2\}| = n$, *неканоническое LG или не может быть преобразовано к таковому применением правил EDD и SPT к частичным расписаниям* π_1 *и* π_2 *соответственно, то при расписании* π *некоторая пара требований* $\{V_{2i-1},\ V_{2i}\},\ i < n$, *не запаздывает или расписание* π *имеет вид*

$$(V_{1,1}, V_{2,1}, \ldots, V_{i,1}, \ldots, V_{n-1,1}, V_{2n-1}, V_{2n}, V_{2n+1}, V_{n-1,2} \ldots, V_{i,2}, \ldots, V_{2,2}, V_{1,2}), \tag{3.1}$$

т.е. пара требований $\{V_{2n-1}, V_{2n}\}$ *обслуживается до требования* V_{2n+1}, *одно требование из каждой пары* $\{V_{2i-1}, V_{2i}\},\ i = 1, \ldots, n-1$, *обслуживается до требования* V_{2n+1}, *другое требование из пары после* V_{2n+1}, *а требования* V_{2n+1} *обслуживается* $n+2$ *по порядку.*

Доказательство. Рассмотрим некоторое расписание $\pi = (\pi_1, \pi_2)$, где $|\{\pi_1\}| = n+1,\ |\{\pi_2\}| = n$. Проанализируем возможные случаи.

1. Если $\{\pi_2\} = \{V_{1,2}, \ldots, V_{n,2}\}$, т.е. в π_2 упорядочены n требований по одному требованию из всех n пар $\{V_{2i-1}, V_{2i}\},\ i = 1, \ldots, n$. Упорядочим требования из π_2 по правилу SPT, а требования из π_1 по правилу EDD. Тогда получим каноническое расписание π', приччм $F(\pi') \leq F(\pi)$, согласно лемме 6.4.

2. Если $\{\pi_2\} \neq \{V_{1,2}, \ldots, V_{n,2}\}$. То возможны ситуации:

   a) $V_{2n+1} \in \{\pi_2\}$,

   b) существует пара требований $\{V_{2j-1}, V_{2j}\} \subset \{\pi_2\}$.

   Тогда, учитывая, что $|\{\pi_2\}| = n$, для некоторого i будем иметь $\{V_{2i-1}, V_{2i}\} \subset \{\pi_1\}$. □

Далее в теореме 1 будет показано, что для случая (2.2) все оптимальные расписания являются каноническими LG. Доказано, что произвольное неканоническое LG-расписание π может быть преобразовано к каноническому LG π', причем $F(\pi) > F(\pi')$. При доказательстве теоремы используются выводы лемм 6.6- 6.9.

Лемма 6.6 *Пусть расписание π имеет вид (3.1), при котором требование V_{2n+1} обслуживается на $n+2$ по порядку месте. Тогда для канонического LG расписания*
$\pi' = (V_{1,1}, V_{2,1}, \ldots, V_{i,1}, \ldots, V_{n-1,1}, V_{2n-1}, V_{2n+1}, V_{2n}, V_{n-1,2} \ldots, V_{i,2}, \ldots, V_{2,2}, V_{1,2})$
выполняется $F(\pi) > F(\pi')$.

Доказательство. При расписании π требование V_{2n-1} обслуживается на позиции n "слева". Согласно лемме требование V_{2n-1} не запаздывает, так как оно обслуживается n-ым по порядку. Требование V_{2n+1} обслуживается на позиции $n+2$ "слева" поэтому запаздывает.

Для требований множества $\{V_2, V_4, \ldots, V_{2i}, \ldots, V_{2n-2}, V_{2n-1}\}$ имеем

$$P(\{V_2, V_4, \ldots, V_{2i}, \ldots, V_{2n-2}, V_{2n-1}\}) = nn^3b + \sum_{k:=1}^{n} kb + \delta_n = d_{V_{2n+1}} - n^3b - \frac{1}{2}\delta + \delta_n,$$

согласно условию 8 из (2.2). Далее, очевидно,

$$P(\{V_{1,1}, V_{2,1}, \ldots, V_{i,1}, \ldots, V_{n-1,1}, V_{2n-1}\}) + p_{2n} \geq$$
$$P(\{V_2, V_4, \ldots, V_{2i}, \ldots, V_{2n-2}, V_{2n-1}\}) + p_{2n},$$

поэтому

$$C_{2n}(\pi) \geq d_{2n+1} + b - \frac{1}{2}\delta + \delta_n > d_{2n}.$$

Следовательно, требование V_{2n}, которое обслуживается на $n+1$ по порядку месте, при расписании π запаздывает.

Расписание π можно записать как $\pi = (\pi_{11}, V_{2n}, V_{2n+1}, \pi_{21})$. Рассмотрим каноническое LG-расписание $\pi' = (\pi_{11}, V_{2n+1}, V_{2n}, \pi_{21})$. Покажем, что $F(\pi) > F(\pi')$. Выделим два случая.

1. Пусть при расписании π' требование V_{2n+1} не запаздывает. Тогда из условия 8 в (2.2) следует, что $d_{2n+1} - C_{2n+1}(\pi') \leq \frac{1}{2}\delta$, так как расписание π' каноническое.

   Из рис. 6.2 видно, что

   $F(\pi) - F(\pi') = T_{2n}(\pi) + T_{2n+1}(\pi) - (T_{2n}(\pi') + T_{2n+1}(\pi')) = (T_{2n+1}(\pi) - T_{2n+1}(\pi')) - (T_{2n}(\pi') - T_{2n}(\pi)) \geq (p_{2n} - \frac{1}{2}\delta) - p_{2n+1} = p_{2n+1} + b - \frac{1}{2}\delta - p_{2n+1} > 0.$

2. Пусть при расписании π' требование V_{2n+1} запаздывает, тогда

   $F(\pi) - F(\pi') = T_{2n}(\pi) + T_{2n+1}(\pi) - (T_{2n}(\pi') + T_{2n+1}(\pi')) = p_{2n} - p_{2n+1} = b > 0.$ □

Лемма 6.7 *Пусть при некотором расписании* $\pi = (\pi_{11}, V_{2i-1}, V_{2i}, \pi_{12}, \pi_{21}, X, \pi_{22})$ *пара требований* $\{V_{2i-1},\ V_{2i}\}$, $i < n$, *не запаздывает, а на позиции* i *"справа" обслуживается требование* $X \in \{V_{2j-1}, V_{2j}\}$, $j \geq i+1$. *Тогда для расписания* $\pi' = (\pi_{11}, V_{2i-1}, X, \pi_{12}, \pi_{21}, V_{2i}, \pi_{22})$ *выполняется* $F(\pi) > F(\pi')$.

Доказательство.

Пусть при расписании π запаздывают требования только из множества $\{\pi_{21},\ X,\ \pi_{22}\}$, где $|\{\pi_{22}\}| = i-1$. Требование X занимает позицию с номером i "справа" (см. рис. 6.3), в которой при каноническом расписании будет обслуживаться требование $V_{i,2} \in \{V_{2i-1}, V_{2i}\}$.

Построим расписание $\pi' = (\pi_{11}, V_{2i-1}, X, \pi_{12}, \pi_{21}, V_{2i}, \pi_{22})$. При обоих расписаниях запаздывает не меньше n требований. Поэтому при расписании π' перед требованием V_{2i} будет запаздывать не меньше, чем $n-i$ требований (и не больше, чем $n-i+1$), согласно лемме 6. Поэтому

$$F(\pi) - F(\pi') \geq (p_{2i} - p_X)(n-i) - (d_X - d_{2i}).$$

Возможны следующие ситуации.

1. Если $X = V_{2j}$, то $p_{2i} - p_X = (j-i)b$,

   $d_X - d_{2i} = \sum_{k:=i}^{j-1}(n-k)b - (j-i)\delta = n(j-i)b - \sum_{k:=i}^{j-1} kb - (j-i)\delta = n(j-i)b - i(j-i)b - \sum_{k:=0}^{j-1-i} kb - (j-i)\delta$.

   Следовательно,

   $F(\pi) - F(\pi') \geq (j-i)b(n-i) - (n(j-i)b - i(j-i)b - \sum_{k:=0}^{j-1-i} kb - (j-i)\delta) = \sum_{k:=0}^{j-1-i} kb + (j-i)\delta > 0$.

2. Если $X = V_{2j-1}$, то
   $p_{2i} - p_X = ((j-i)b - \delta_j)$,

   $d_X - d_{2i} = \sum_{k:=i}^{j-1}(n-k)b - (j-i)\delta - (n-j)\delta_j - \varepsilon\delta_j = n(j-i)b - \sum_{k:=i}^{j-1} kb - (j-i)\delta - (n-j)\delta_j - \varepsilon\delta_j = n(j-i)b - i(j-i)b - \sum_{k:=0}^{j-1-i} kb - (j-i)\delta - (n-j)\delta_j - \varepsilon\delta_j$.

   Следовательно,

   $F(\pi) - F(\pi') \geq ((j-i)b - \delta_j)(n-i) - (n(j-i)b - i(j-i)b - \sum_{k:=0}^{j-1-i} kb - (j-i)\delta - (n-j)\delta_j - \varepsilon\delta_j) = \sum_{k:=0}^{j-1-i} kb + (j-i)\delta - (j-i)\delta_j + \varepsilon\delta_j > 0$.

□

Лемма 6.8 *Пусть при некотором расписании* $\pi = (\pi_{11}, V_{2i-1}, V_{2i}, \pi_{12}, \pi_{21}, X, \pi_{22})$ *пара требований* $\{V_{2i-1},\ V_{2i}\}$, $i < n$, *не запаздывает, а на позиции* i *"справа" обслуживается требование* $X \in \{V_{2j-1}, V_{2j}\}$, $j < i-1$. *Тогда для расписания* $\pi' = (\pi_{11}, V_{2i-1}, X, \pi_{12}, \pi_{21}, V_{2i}, \pi_{22})$ *выполняется* $F(\pi) > F(\pi')$.

Доказательство.

Допустим, что при расписании π запаздывают требования только из множества $\{\pi_{21},\ X,\ \pi_{22}\}$, где $|\{\pi_{22}\}| = i-1$. Требование X занимает позицию i "справа" (рис. 6.3), в которой при каноническом расписании будет обслуживаться требование $V_{i,2} \in \{V_{2i-1}, V_{2i}\}$.

Построим расписание $\pi' = (\pi_{11}, V_{2i-1}, X, \pi_{12}, \pi_{21}, V_{2i}, \pi_{22})$. При обоих расписаниях запаздывает не меньше n требований. Поэтому при расписании π' перед требованием V_{2i} будет запаздывать не меньше, чем $n-i$ требований (и не больше, чем $n-i+1$), согласно лемме 4. Поэтому

$$F(\pi) - F(\pi') \geq (d_{2i} - d_X) - (p_X - p_{2i})(n-i+1).$$

По аналогии с доказательством леммы 7 выделим два случая.

1. Если $X = V_{2j}$, то $p_X - p_{2i} = (i-j)b$,

   $d_{2i} - d_{2j} = \sum_{k:=j}^{i-1}(n-k)b - (i-j)\delta = n(i-j)b - \sum_{k:=j}^{i-1} kb - (i-j)\delta = n(i-j)b - (i-1)(i-j)b + \sum_{k:=0}^{i-1-j} kb - (i-j)\delta$.

   Следовательно,

   $F(\pi) - F(\pi') \geq n(i-j)b - (i-1)(i-j)b + \sum_{k:=0}^{i-1-j} kb - (i-j)\delta - (i-j)b(n-i+1) = \sum_{k:=0}^{i-1-j} kb - (i-j)\delta > 0$.

2. Если $X = V_{2j-1}$, то $p_X - p_{2i} = (i-j)b + \delta_j$,

   $d_{2i} - d_{2j-1} = \sum_{k:=j}^{i-1}(n-k)b - (i-j)\delta + (n-j)\delta_j + \varepsilon\delta_j = n(i-j)b - \sum_{k:=j}^{i-1} kb - (i-j)\delta + (n-j)\delta_j + \varepsilon\delta_j = n(i-j)b - (i-1)(i-j)b + \sum_{k:=0}^{i-1-j} kb - (i-j)\delta + (n-j)\delta_j + \varepsilon\delta_j$.

   Тогда имеем

$F(\pi) - F(\pi') \geq n(i-j)b - (i-1)(i-j)b + \sum_{k:=0}^{i-1-j} kb - (i-j)\delta + (n-j)\delta_j + \varepsilon\delta_j - ((i-j)b+\delta_j)(n-i+1) = \sum_{k:=0}^{i-1-j} kb - (i-j)\delta - \delta_j + \varepsilon\delta_j > 0.$
□

Лемма 6.9 *Пусть при некотором расписании* $\pi = (\pi_{11}, V_{2i-1}, V_{2i}, \pi_{12}, \pi_{21}, X, \pi_{22})$ *пара требований* $\{V_{2i-1},\ V_{2i}\}$, $i < n$, *не запаздывает, а на позиции* i *"справа" обслуживается требование* $X \in \{V_{2(i-1)-1}, V_{2(i-1)}\}$. *Предположим, что для расписания* $\pi' = (\pi_{11}, V_{2i-1}, X, \pi_{12}, \pi_{21}, V_{2i}, \pi_{22})$ *требование* Y *занимает позицию* $(n+1)$ *и* $T_Y(\pi') < 2\delta$. *Тогда выполняется* $F(\pi) > F(\pi')$.

Доказательство.

Пусть при расписании π запаздывают требования только из множества $\{\pi_{21},\ X,\ \pi_{22}\}$, где $|\{\pi_{22}\}| = i-1$. Требование X занимает позицию i "справа" (рис. 6.3), в которой при каноническом расписании будет обслуживаться требование $V_{i,2} \in \{V_{2i-1}, V_{2i}\}$.

Построим расписание $\pi' = (\pi_{11}, V_{2i-1}, X, \pi_{12}, \pi_{21}, V_{2i}, \pi_{22})$.

При расписании π перед требованием V_{2i} будет запаздывать не меньше $n-i$ требований. Поэтому

$$F(\pi) - F(\pi') > (d_{2i} - d_X) - (p_X - p_{2i})(n-i) - (T_Y(\pi') - T_Y(\pi)) >$$
$$(d_{2i} - d_X) - (p_X - p_{2i})(n-i) - 2\delta.$$

Возможны ситуации:

1. $X = V_{2(i-1)}$, т.е. $p_X - p_{2i} = b$,

   $d_{2i} - d_{2i-2} = (n-i+1)b - \delta$.

   Поэтому
   $F(\pi) - F(\pi') > (n-i+1)b - \delta - (n-i)b - 2\delta = b - 3\delta > 0.$

2. $X = V_{2(i-1)-1}$, т.е. $p_X - p_{2i} = b + \delta_{i-1}$,

$d_{2i} - d_{2i-2} = (n-i+1)b - \delta + (n-i+1)\delta_{i-1} + \varepsilon\delta_{i-1}$.

Следовательно,

$F(\pi) - F(\pi') > (n-i+1)b - \delta + (n-i+1)\delta_{i-1} + \varepsilon\delta_{i-1} - (n-i)(b + \delta_{i-1}) - 2\delta = b - 3\delta + \delta_{i-1} + \varepsilon\delta_{i-1} > 0$, так как $b = n^2\delta$. $\square$

Результаты леммы 6.9 используются для доказательства теоремы 1. При этом имеет место $T_Y(\pi') < 2\delta$. Ситуация $T_Y(\pi') \geq 2\delta$ нами не рассматривается, так как она не встречается.

На основе полученных лемм докажем следующую теорему.

Теорема 6.1 *Для случая (2.2) все оптимальные расписания являются каноническими или могут быть преобразованы к таковым применением правила EDD к $n+1$ требованиям, обслуживаемым в начале расписания.*

Доказательство.

Пусть π - произвольное расписание. Согласно лемме 6.4 можно рассматривать только расписание вида $\pi = (\pi_{EDD}, \pi_{SPT})$, где $|\{\pi_{EDD}\}| = n+1$. При этом расписании требование V_{2n+1} занимает или позицию $(n+1)$, или позицию $n+2$. Допустим, что расписание π не каноническое.

Тогда в силу леммы 6 имеем при расписании π пару незапаздывающих требований $\{V_{2i-1},\ V_{2i}\}$, $i < n$, или расписание π имеет вид (3.1), при котором требование V_{2n+1} обслуживается на $n+2$ месте.

Если расписание имеет вид (3.1), то по лемме 6.6 для канонического расписания
$\pi' = (V_{1,1}, V_{2,1}, \ldots, V_{i,1}, \ldots, V_{n-1,1}, V_{2n-1}, V_{2n+1}, V_{2n}, V_{n-1,2} \ldots, V_{i,2}, \ldots, V_{2,2}, V_{1,2})$.
Выполняется $F(\pi) > F(\pi')$. Переобозначим $\pi = \pi'$.

Далее опишем алгоритм, состоящий из двух циклов преобразования исходного расписания π к каноническому виду.

Цикл 1. Данный цикл продолжается, пока среди незапаздывающих требований при очередном расписании π присутствует пара V_{2i-1}, V_{2i}, причем на позиции i "справа" обслуживается требование

$X \notin \{V_{2(i-1)-1}, V_{2(i-1)}\}, X \neq V_{2n+1}$. Применяем для требований V_{2i} и X перестановку, описанную в леммах 6.7 и 6.8. В результате значение целевой функции уменьшится.
Конец цикла 1. Переобозначим $\pi := \pi'$.

Количество шагов в цикле 1, очевидно, не превышает n.

Первые $n+1$ требований при расписании π упорядочим по правилу **EDD**.

Требование V_{2n+1} занимает позицию $n+1$ или позицию $n+2$ при расписании π. Если V_{2n+1} занимает позицию $n+2$ "слева" , то позиции n и $n+1$ "слева" занимают требования V_{2n-1} и V_{2n} соответственно, согласно **Циклу 1** и сортировке по правилу EDD.

Рассмотрим две возможные ситуации.

Ситуация I. Пусть требование V_{2n+1} занимает позицию $n+2$.

Рассматривается расписание $\pi = (\pi_1, V_{2n-1}, V_{2n}, V_{2n+1}, \pi_2)$, при котором требование V_{2n} обслуживается $n+1$ по порядку.

При частичных расписаниях π_1 и π_2 обслуживается по $(n-1)$ требованию, т.е. $|\{\pi_1\}| = n-1 = |\{\pi_2\}|$.

Учитывая, что после выполнения **Цикла 1** останутся только ситуации описанные в лемме 6.9, поэтому $P(\pi_1) + 2qb + \delta > P(\pi_2) > P(\pi_1) + 2qb - \delta$, где q – количество ситуаций представленные в лемме 10 при расписании π.

Примером подобной ситуации может служить расписание, при котором $\{\pi_1\} = \{V_{2i-1}, V_{2i}\} \cup \{V_{1,1}, V_{2,1}, \ldots, V_{i-2,1}, V_{i+1,1}, \ldots, V_{n-1,1}\}$,

$\{\pi_2\} = \{V_{2(i-1)-1}, V_{2(i-1)}\} \cup \{V_{1,2}, V_{2,2}, \ldots, V_{i-2,2}, V_{i+1,2}, \ldots, V_{n-1,2}\}$.

Тогда $q = 1$ и $P(\pi_1) + 2b + \delta > P(\pi_2) > P(\pi_1) + 2b - \delta$, так как

$-(\delta - \delta_{i-1} - \delta_i - \delta_n) < P(\{V_{1,1}, V_{2,1}, \ldots, V_{i-2,1}, V_{i+1,1}, \ldots, V_{n-1,1}\}) - P(\{V_{1,2}, V_{2,2}, \ldots, V_{i-2,2}, V_{i+1,2}, \ldots, V_{n-1,2}\}) < \delta - \delta_{i-1} - \delta_i - \delta_n$ и

$P(\{V_{2(i-1)-1}, V_{2(i-1)}\}) - P(\{V_{2i-1}, V_{2i}\}) = 2b + \delta_{i-1} - \delta_i$.

Выделим два случая, когда $q = 1$ и $q > 1$. При $q = 0$ расписание π имеет вид (3.1) (см. лемму 6.6). Этот случай мы рассмотрели выше.

Пусть $q = 1$.

Известно, что $\sum_{i:=1}^{2n+1} p_i = 2L + p_{2n+1} = 2L + n^3 b$.

Обозначим $\Delta = P(\pi_2) - (P(\pi_1) + 2b)$, где $-\delta < \Delta < \delta$.

Примем $S = P(\pi_1)$, тогда $2S + 2b + \Delta + p_{2n-1} + p_{2n} + p_{2n+1} = 2S + \Delta + 2b + 3n^3 b + 2b + \delta_n = 2L + n^3 b$.

Поэтому

$$L = S + \frac{1}{2}\Delta + 2b + n^3 b + \frac{1}{2}\delta_n,$$

а значит

$$C_{2n}(\pi) = P(\pi_1) + p_{2n-1} + p_{2n} = S + 2n^3 b + 2b + \delta_n = L + n^3 b + \frac{1}{2}\delta_n - \frac{1}{2}\Delta.$$

Известно, что $L + n^3 b = d_{2n+1}$, тогда $-\delta < C_{2n}(\pi) - d_{2n+1} < \delta$.

Необходимо рассмотреть два подслучая.

1. $C_{2n}(\pi) \geq d_{2n+1}$.

   Рассмотрим расписание $\pi' = (\pi_1, V_{2n-1}, V_{2n+1}, V_{2n}, \pi_2)$.

   $F(\pi) - F(\pi') = T_{2n}(\pi) + T_{2n+1}(\pi) - (T_{2n}(\pi') + T_{2n+1}(\pi')) = (T_{2n+1}(\pi) - T_{2n+1}(\pi')) - (T_{2n}(\pi') - T_{2n}(\pi)) = (p_{2n+1} + (C_{2n}(\pi) - d_{2n+1})) - p_{2n+1} = C_{2n}(\pi) - d_{2n+1} \geq 0$.
2. $C_{2n}(\pi) < d_{2n+1}$.
   При этом $C_{2n}(\pi) > d_{2n}$, так как $d_{2n+1} - d_{2n} = \delta$ и $d_{2n+1} - C_{2n}(\pi) < \delta$.

   Расписание π имеет следующую структуру:

   $$\pi = (\pi_{11}, V_{2i-1}, V_{2i}, \pi_{12}, V_{2n-1}, V_{2n}, V_{2n+1}, \pi_{21}, X, \pi_{22}),$$

   где $|\{\pi_{22}\}| = i - 1$, $X \in \{V_{2(i-1)-1}, V_{2(i-1)}\}$.

   Если $X = V_{2(i-1)-1}$, то транспозиция соседних требований $V_{2(i-1)-1}$ и $V_{2(i-1)}$ согласно правилу SPT не увеличит значение целевой функции.
   Пусть $X = V_{2(i-1)}$. При расписании π запаздывает $n + 1$ требование.
   Построим расписание

   $$\pi' = (\pi_{11}, V_{2i-1}, X, \pi_{12}, V_{2n-1}, V_{2n}, V_{2n+1}, \pi_{21}, V_{2i}, \pi_{22}).$$

Выполняется
$F(\pi) - F(\pi') = (d_{2i} - d_{2(i-1)}) - (n-i+1)(p_{2(i-1)} - p_{2i}) = (n-i+1)b - \delta - (n-i+1)b = -\delta$, т.е. значение целевой функции увеличилось на δ.
Тогда $C_{2n}(\pi') - d_{2n+1} > b - \delta$. Сформируем расписание

$$\pi'' = (\pi_{11}, V_{2i-1}, X, \pi_{12}, V_{2n-1}, V_{2n+1}, V_{2n}, \pi_{21}, V_{2i}, \pi_{22}).$$

Оценим разность $F(\pi') - F(\pi'') > (p_{2n+1} + b - \delta) - p_{2n+1} > b - \delta$.
Тогда $F(\pi) - F(\pi'') = b - \delta - \delta > 0$.

Пусть $q > 1$. Тогда $d_{2n} - C_{2n}(\pi) > b - 2\delta$.

Если $q = 2$ при расписании π', рассмотренном в лемме 6.9, для требования $Y = V_{2n}$ выполняется $T_Y(\pi') < 2\delta$. Другими словами, перестановка, описанная в лемме 6.9, уменьшит значение целевой функции.

Когда $q > 2$ при расписании π' будет запаздывать n требований, поэтому, согласно лемме 6.9, имеет место неравенство $F(\pi) > F(\pi')$

Ситуация II. Предположим, что требование V_{2n+1} занимает позицию $n+1$. Тогда в ситуации, описанной в лемме 10, $T_Y(\pi') = T_{2n+1}(\pi') < \frac{1}{2}\delta$. Преобразование расписания согласно лемме 10 уменьшит значение целевой функции.

Цикл 2. Выполняется, пока среди незапаздывающих требований при очередном расписании π присутствует пара V_{2i-1}, V_{2i}, причем на позиции i "справа" обслуживается требование $X \in \{V_{2(i-1)-1}, V_{2(i-1)}\}$. Применяем для требований X и V_{2i} перестановку, описанную для ситуаций **I** и **II**. При этом значение целевой функции уменьшается.
Конец Цикла 2 и **алгоритма преобразования исходного неканонического расписания.**

Таким образом, произвольное неканоническое расписание π за время $O(n)$ можно преобразовать к каноническому π^* . Причем $F(\pi) > F(\pi^*)$.
□

Теорема 6.2 *Решение примера ЧНР будет "ДА" тогда и только тогда, когда при оптимальном каноническом расписании* $C_{2n+1}(\pi) = d_{2n+1}$.

Доказательство.

Рассмотрим каноническое расписание вида

$$\pi = (V_{1,1}, V_{2,1}, \ldots, V_{i,1}, \ldots, V_{n,1}, V_{2n+1}, V_{n,2}, \ldots, V_{i,2}, \ldots, V_{2,2}, V_{1,2})$$

Известно, что требования $V_{n,2}, \ldots, V_{i,2}, \ldots, V_{2,2}, V_{1,2}$ запаздывают. Требование V_{2n+1} может запаздывать, тогда $F(\pi) = \sum_{i:=1}^{n} T_{V_{i,2}}(\pi) + T_{V_{2n+1}}(\pi)$.

Обозначим $\sum_{i:=1}^{2n+1} p_i = C$.

Тогда

$$\sum_{i:=1}^{n} C_{V_{i,2}}(\pi) = nC - \sum_{i:=1}^{n-1}(n-i)p_{V_{i,2}}.$$

Введем функцию

$$\phi(i) = \begin{cases} 1, \ V_{i,2} = V_{2i-1}, \\ 0, \ V_{i,2} = V_{2i}, \end{cases}$$

имеем

$$d_{V_{i,2}} = d_{2n+1} - (\sum_{k:=i}^{n-1}(n-k)b + (n-i+1)\delta + \phi(i)((n-i)\delta_i + \varepsilon\delta_i)),$$

поэтому

$$\sum_{i:=1}^{n} T_{V_{i,2}}(\pi) = nC - \sum_{i:=1}^{n-1}(n-i)p_{V_{i,2}} -$$
$$\sum_{i:=1}^{n}(d_{2n+1} - (\sum_{k:=i}^{n-1}(n-k)b + (n-i+1)\delta + \phi(i)((n-i)\delta_i + \varepsilon\delta_i))).$$

Задача $\min_\pi F(\pi) = \min(\sum_{i:=1}^{n} T_{V_{i,2}}(\pi) + T_{V_{2n+1}}(\pi))$ сводится к максимизации функции Φ, где $\Phi = \sum_{i:=1}^{n-1}(n-i)p_{V_{i,2}} - \sum_{i:=1}^{n}\phi(i)((n-i)\delta_i + \varepsilon\delta_i) - T_{V_{2n+1}}(\pi)$.

Рассмотрим два возможных случая.

1. Если $V_{i,2} = V_{2i},\ i = 1, \ldots, n$, то $T_{V_{2n+1}}(\pi) = \frac{1}{2}\delta$, $\Phi_1 = \sum_{i:=1}^{n-1}(n-i)p_{2i} - \frac{1}{2}\delta$.

2. Если $V_{i,2} = V_{2i-1},\ i = 1, \ldots, n$, то $T_{V_{2n+1}}(\pi) = \max\{-\frac{1}{2}\delta, 0\} = 0$, $\Phi = \sum_{i:=1}^{n-1}(n-i)p_{2i-1} - \sum_{i:=1}^{n}((n-i)\delta_i + \varepsilon\delta_i) = \sum_{i:=1}^{n-1}(n-i)p_{2i} + \sum_{i:=1}^{n-1}(n-i)\delta_i - \sum_{i:=1}^{n}((n-i)\delta_i + \varepsilon\delta_i) = \Phi_1 + \frac{1}{2}\delta - \sum_{i:=1}^{n}\varepsilon\delta_i$.

Функция Φ достигает максимальное значение $\Phi_1 + \frac{1}{2}\delta - \frac{1}{2}\sum_{i:=1}^{n}\varepsilon\delta_i$, когда $\sum_{i:=1}^{n}\phi(i)(\varepsilon\delta_i) = \frac{1}{2}\sum_{i:=1}^{n}\varepsilon\delta_i$, что равнозначно $\sum_{i:=1}^{n}\phi(i)\delta_i = \frac{1}{2}\sum_{i:=1}^{n}\delta_i$. Следовательно, для модифицированного примера существуют два подмножества A_1 и A_2, таких что $\sum_{a_i \in A_1} a_i = \sum_{a_i \in A_2} a_i$ (ответ "ДА"). При этом $C_{2n+1}(\pi) = d_{2n+1}$.

Если решение модифицированного примера ЧНР "НЕТ то не выполняется равенство $\sum_{i:=1}^{n}\phi(i)\delta_i = \frac{1}{2}\sum_{i:=1}^{n}\delta_i$. Учитывая значение d_{2n+1}, будем иметь $C_{2n+1}(\pi) \neq d_{2n+1}$

Если $C_{2n+1}(\pi) = d_{2n+1}$, то из этого следует, что $\sum_{i:=1}^{n} p_{V_{i,1}} = \sum_{i:=1}^{n} p_{V_{2i}} + \frac{1}{2}\delta = \sum_{i:=1}^{n} p_{V_{i,2}}$, т.е. решение модифицированного примера **ЧНР** будет иметь ответ "ДА". $\square$

Таблица терминов и обозначений

Таблица 6.1: Термины и обозначения

Обозначение	**Название**	**Англ. название**	**Альтернативное обозначение**
p_j	продолжительность обслуживания	processing time	t_j
d_j	директивный срок	due date	D_j
D_j		deadline	
r_j	время поступления	release date	
w_j	вес	weight	
C_j	время окончания обслуживания	completion time	
S_j	время начала обслуживания	start time	
T_j	запаздывание	tardiness	z_j
L_j	временное смещение	lateness	
E_j	опережение	earliness	
$C_{\max}$	быстродействие	makespan	
$prec$	отношения предшествования	precedence relations	
$pmtn$	прерывания	preemptions	
$tree$	дерево	tree	
$chain$	цепочка	chain	
Pm	идентичные машины	parallel machines	
Qm			
Jm		job-shop	
Om		open-shop	
Fm		flow-shop	

Кто есть кто в Теории Расписаний

В алфавитном порядке. Русскоязычные авторы вначале.

Бурков Владимир Николаевич, род. в 1939 г. Академик РАЕН и Нью-Йоркской АН. Работы в области управ. систем, объемно-календарного планирования. Лауреат гос. премии СССР. Возглавляет лабораторию в ИПУ РАН, профессор МФТИ.

Гордон Валерий Сергеевич (1945–2010) Работы в области **TP** , дискретной оптимизации. Лауреат гос. премии Республики Беларусь. Работал г.н.с. в Объед. Институте Проблем Информатики (ОИПИ) НАН Беларуси.

Ковалеёв Михаил Яковлевич, род. в 1959 г. Работы в области **TP** , дискретной оптимизации, теории графов. Лауреат гос. премии Республики Беларусь. Профессор Белорусского государственного университета (г. Минск), зам. дир. Объед. Института Проблем Информатики (ОИПИ) НАН Беларуси.

Севастьянов Сергей Васильевич. Работы в области **TP** , дискретной оптимизации, теории графов, комбинаторной геометрии. В.н.с. Института Математики СО РАН. Профессор Новосибирского ГУ

Сотсков Юрий Назарович, род. в 1948 г. Работы в области **TP** , дискретной оптимизации, теории графов. Лауреат гос. премии Республики Беларусь. Профессор Бел.ГУ, зав.лаб. ОИПИ НАН Беларуси.

Танаев Вячеслав Сергеевич (1940–2002). Лауреат гос. премии Республики Беларусь. Академик НАН Беларуси с 2000 г. С 1987 г. директор Института тех. кибернетики АН БССР. Один из основоположников **ТР** в СССР.

Шкурба Виктор Васильевич, род. в 1935 г. Работы в области АСУП, математической кибернетики, объемно-календарного планирования. Лауреат гос. премий СССР и УССР. Профессор Государственного Университета Управления (Москва). Один из основоположников **ТР** в СССР.

Baptiste Philippe, род. в 1955 г. Работы в области **ТР** , исследования операций. Профессор Ecole Polytechnique.

Blazewicz Jacek, род. в 1951 г. Работы в области **ТР** , исследования операций. Профессор, зам. дир., the Institute of Computing Science, Poznan University of Technology.

Burke Edmund. Работы в области **ТР** , исследования операций. Профессор, декан, University of Nottingham.

Peter Brucker. Работы в области **TP**, исследования операций. Работал университете Оснабрюка.

Kendall Graham. Работы в области **TP**, исследования операций. Профессор, University of Nottingham.

Lawler Eugene Leighton (Gene) (1933–1994). Американский ученый. Работал в унив. Калифорнии и Беркли. Работы в области комбинаторной оптимизации и **TP**.

Leung Joseph Y.-T. Работы в области **TP**. Профессор института технологий Нью-Джерси.

Werner Frank, род. в 1955 г. Работы в области **TP**, дискретной оптимизации, теории графов. Профессор университета г. Магдебурга (Германия)

Литература

[1] *Беллман Р.* Динамическое программирование М.: ИЛ, 1960. 400 с.

[2] *Бурдюк В.Я., Шкурба В.В.* Теория расписаний. Задачи и методы решений // Кибернетика. 1971. № 1. С. 89–102.

[3] *под редакцией Буркова В.Н.* Математические основы управления проектами. М.: Высшая школа, 2005, 432 с.

[4] *Гафаров Е.Р.* Гибридный алгоритм решения задачи минимизации суммарного запаздывания для одного прибора // Информационные технологии, 2007. № 1 С. 30–37.

[5] *Гафаров Е.Р., Лазарев А.А.* Доказательство NP-трудности частного случая задачи минимизация суммарного запаздывания для одного прибора // Известия АН: Теория и системы управления. 2006. № 3. С. 120–128.

[6] *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи. Пер. с англ. М.: Мир, 1982. 416 с.

[7] *Емеличев В.А., Мельников О.И., Сарванов В.И., Тышкевич Р.И.* Лекции по теории графов. М.: Наука, 1990, 384 с.

[8] *Карп Р.М.* Сводимость комбинаторных проблем // Кибернетический сборник. М.: Мир. 1972. Вып. 12. С. 16–38.

[9] *Каширских К.Н., Поттс К.Н., Севастьянов С.В.* Улучшенный алгоритм решения двухмашинной задачи flow shop с неодновременным поступлением работ // Дискретный анализ и исследование операций. 1997.– Т. 4, № 1.– С. 13 – 32.

[10] *Корбут А.А., Сигал И.Х., Финкельштейн Ю.Ю.* Методы ветвей и границ. Обзор теорий, алгоритмов, программ и приложений // Operations Forsch. Statist., Ser. Optimiz. 1977. V. 8. № 2. P. 253–280.

[11] *Корбут А.А., Сигал И.Х., Финкельштейн Ю.Ю.* Гибридные методы в дискретном программировании // Изв. АН СССР. Техн. кибернет. 1988. № 1. С. 65–77.

[12] *Корбут А.А., Финкельштейн Ю.Ю.* Приближенные методы дискретного программирования // Изв. АН СССР. Техн. кибернет. 1983. № 1. С. 165–176.

[13] *Ковалчв М.Я.* Интервальные ϵ – приближчнные алгоритмы решения дискретных экстремальных задач // Дисс. канд. физ.-мат. наук. – Минск, 1986. – 110 с.

[14] *Лазарев А.А.* Алгоритмы в теории расписаний, основанные на необходимых условиях оптимальности // Исследования по прикладной математике. Казань: Изд-во Казан. гос. ун-та, 1984. Вып. 10. С. 102–110.

[15] *Лазарев А.А.* Алгоритмы декомпозиционного типа решения задачи минимизации суммарного запаздывания // Исследования по прикладной математике. Казань: Изд-во Казан. гос. ун-та, 1990. Вып. 17. С. 71–78.

[16] *Лазарев А.А.* Эффективные алгоритмы решения некоторых задач теории расписаний для одного прибора с директивными сроками обслуживания требований: Дис. канд. физ.-мат. наук. Казань, 1989. 108 с.

[17] *Лазарев А.А., Кварацхелия А.Г., Гафаров Е.Р.* Алгоритмы решения NP-трудной проблемы минимизации суммарного запаздывания для одного прибора // Доклады Академии Наук, 2007. Том 412. № 6. С. 739–742. (Англ. вариант статьи: *A. A. Lazarev, A. G. Kvaratskheliya, and E. R. Gafarov.* Algorithms for Solving the NP-Hard Problem of Minimizing Total Tardiness for a Single Machine // Doklady Mathematics. 2007. Vol. 75. No. 1. P. 130–133).

[18] *Лазарев А.А.* Графический подход к решению задач комбинаторной оптимизации // Атоматика и телемеханика. 2007. № 4. С. 13–23.

[19] *Пападимитриу Х., Стайглиц К.* Комбинаторная оптимизация. Алгоритмы и сложность. М.: Мир, 1985. 512 с.

[20] Большой энциклопедический словарь. Математика. *Под. ред. Ю.В. Прохорова* М.: Большая Российская энциклопедия, 1998, 848 с.

[21] *Севастьянов С.В.* Геометрические методы и эффективные алгоритмы в теории расписаний // Дис. док. физ.-мат. наук.– Новосибирск: 2000.– 280 с.

[22] *Сигал И.Х., Иванова А.П.* Введение в прикладное дискретное программирование: теория и вычислительные алгоритмы. М.: Физматлит, 2002. 240 с.

[23] *Танаев В.С., Шкурба В.В.* Введение в теорию расписаний. М.: Наука, 1975.

[24] *Танаев В.С., Гордон В.С., Шафранский Я.М.* Теория расписаний. Одностадийные системы. М.: Наука. Гл. ред. физ.-мат. лит., 1984. 384 с.

[25] *Танаев В.С., Сотсков Ю.Н., Струсевич В.А.* Теория расписаний. Многостадийные системы. – М.: Наука. Гл. ред. физ.-мат. лит., 1989. – 328 с.

[26] *Танаев В.С., Ковалев М.Я., Шафранский Я.М.* Теория расписаний. Групповые технологии. Минск: Институт технической кибернетики НАН Беларуси, 1998. 290 с.

[27] *Финкельштейн Ю.Ю.* Приближенные методы и прикладные задачи дискретного программирования. М.: Наука, 1976.

[28] *Хачатуров В.Р., Веселовский В.Е., Злотов А.В. и др.* Комбинаторные методы и алгоритмы решения задач дискретной оптимизации большой размерности. М.: Наука. 2000.

[29] *Шкурба В.В., Подчасова Т.П., Пшичук А.Н., Тур Л.П.* Задачи календарного планирования и методы их решения. Киев: Наукова думка, 1966. 154 с.

[30] *Alon N., Woeginger G.J., Yadid T.* Approximation schemes for scheduling on parallel machines // J. of Scheduling.– 1998.– V. 1.– P. 55 – 66.

[31] *Aloulou M.A., Kovalyov M.Y., Portmann M.-C.* Maximization Problems in Single Machine Scheduling //Annals of Operations Research, 2004, 129, 21 – 32.

[32] *Aloulou M.A., Kovalyov M.Y., Portmann M.-C.* Evaluation Flexible Solutions in Single Machine Scheduling via Objective Function Maximization: the Study of Computational Complexity //RAIRO Oper. Res., 2007, 41, 1 – 18.

[33] *van de Akker J.M., Hoogeveen J.A., van de Velde S.L.* Parallel machine scheduling by column generation // Oper. Res.– 1999.– V. 47, N 6.– P. 862 – 872.

[34] *van de Akker J.M., Hurkens C.A.J., Savelsbergh M.W.P.* Time-indexed formulations for single-machine scheduling problems: column generation // INFORMS J. on Computing.– 2000.– V. 12, N 2.– P. 111 – 124.

[35] *Baptiste Ph., Le Pape C., Nuijten W.* Constraint-based scheduling: applying constraint programming to scheduling problems // Kluwer Academic Publishers, 2001.– 198 p.

[36] *A. Bauer, B. Bullnheimer, R.F.Hartl, C. Strauss.* Minimizing Total Tardiness on a Single Machine Using Ant Colony Optimization // Proceedings of the 1999 Congress on Evolutionary Computation (CEC99). Washington D.C, 1999. P. 1445–1450.

[37] *Bellman R.* Mathematical aspects of scheduling theory // Journal of the Society of Industrial and Applaid Mathematics. 1956. Vol. 4. P. 168–205.

[38] *Blazewicz J., Ecker K., Pesch E., Schmidt G., Weglarz J.* Scheduling Computer and Manufactoring Processes. Springer Berlin. 1996.

[39] *Bellman R.* Mathematical aspects of scheduling theory // Journal of the Society of Industrial and Applaid Mathematics. 1956. Vol. 4. P. 168–205.

[40] *Brooks G.N., White C.R.* An algorithm for finding optimal or near – optimal solutions to the production scheduling problem // J. Ind. Eng.– 1965.– V. 16, N 1.– P. 34 – 40.

[41] *Brucker P.* Scheduling Algorithms. Springer-Verlag, 2001. 365 p.

[42] *Brucker P., Lenstra J.K., Rinnoy Kan A.N.G.* Complexity of machine scheduling problems // Math. Cent. Afd. Math Beslisk. Amsterdam, 1975. BW 43. 29 pp.

[43] *Brucker P., Knust S.* Complex scheduling Springer-Verlag Berlin, Heidelberg, Germany, 2006.

[44] *Burkov V.N.* Problems of optimum distribution of resources // Control and cibernetics. 1972. Vol. 1. № 1/2. P. 27–41.

[45] *Carlier J.* The one-machine sequencing problem // European J. of Oper. Res.– 1982.– V. 11, N 1.– P. 42 – 47.

[46] *Cheng T.C.-E., Lazarev A.A., Gafarov E.R.* A Hybrid Algorithm for the Single Machine Total Tardiness Problem // Computers & Operations Research. Vol. 36. Issue 2. P. 308–315.

[47] *T.C.E. Cheng, C.T. Ng, J.J. Yuan, Z.H. Liu* Single Machine Parallel Batch Scheduling Subject to Precedence Constraints Naval Research Logistics. Vol. 57. Issue 7. 2004. P. 314–324.

[48] *Conway R.W., Maxwell W.L., Miller L.W.* Theory of Scheduling. Addison-Wesley, Reading, MA. 1967.

[49] *Della Croce F., Grosso A., Paschos V.* Lower bounds on the approximation ratios of leading heuristics for the single-machine total tardiness problem // Journal of Scheduling. 2004. V. 7. P. 85–91.

[50] *Demeulemeester E.L., Herroelen W.S.* Experimental evaluation of state-of-the-art heurisitcs for the resource-constrained project scheduling problem // EJOR. 1996. V. 90. P. 334–348.

[51] *A. Colorni, M. Dorigo et V. Maniezzo.* Distributed Optimization by Ant Colonies, actes de la premiere conference europeenne sur la vie artificielle, Paris, France, Elsevier Publishing. 1991. P. 134–142.

[52] *M. Dorigo.* Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italie, 1992.

[53] *Du J., Leung J. Y.-T.* Minimizing total tardiness on one processor is NP-hard // Math. Operation Research. 1990. V. 15. P. 483–495.

[54] *Emmons H.* One machine sequencing to minimizing certain function of job tardiness // Operations Research. 1969. V. 17. P. 701–715.

[55] *E.R. Gafarov, A.A. Lazarev and F. Werner.* Algorithms for Some Maximization Scheduling Problems on a Single Machine // Automation and Remote Control. 2010. Vol. 10. P. 2070–2084.

[56] *E.R. Gafarov, A.A. Lazarev and F. Werner.* Algorithms for Some Maximization Scheduling Problems on a Single Machine // Preprint 38/09, FMA, Otto-von-Guericke-Universitat Magdeburg. 2009. 29 pp.

[57] *E.R. Gafarov, A.A. Lazarev and F. Werner.* Single Machine Scheduling with a Non-Renewable Financial Resource // Preprint 07/10, FMA, Otto-von-Guericke-Universitat Magdeburg. 2010. 19 pp.

[58] *E.R. Gafarov, A.A. Lazarev and F. Werner.* Single Machine Scheduling with Generalized Total Tardiness Objective Function // Preprint 10/10, FMA, Otto-von-Guericke-Universitat Magdeburg. 2010. 8 pp.

[59] *E.R. Gafarov, A.A. Lazarev and F. Werner.* A Polynomial Time Graphical Algorithm for Maximizing Total Tardiness on a Single Machine // Preprint 12/10, FMA, Otto-von-Guericke-Universitat Magdeburg. 2010. 15 pp.

[60] *E.R. Gafarov, A.A. Lazarev and F. Werner.* On Lower and Upper Bounds for the Resource-Constrained Project Scheduling Problem // Preprint 8/10, FMA, Otto-von-Guericke-Universitat Magdeburg. 2010. 27 pp.

[61] *E.R. Gafarov, A.A. Lazarev and F. Werner.* Classical Combinatorial and Single Machine Scheduling Problems with Opposite Optimality Criteria // Preprint 11/10, FMA, Otto-von-Guericke-Universitat Magdeburg. 2010. 15 pp.

[62] *E.R. Gafarov, A.A. Lazarev and F. Werner.* A Modification of Dynamic Programming Algorithms to Reduce the Time Complexity

// Preprint 20/10, FMA, Otto-von-Guericke-Universitat Magdeburg. 2010. 24 pp.

[63] *Gantt H.L.* ASME Transactions. 1903. 24. P. 1322–1336.

[64] *Graham R.L.* Bounds for certain multiprocessing anomalies // SIAM J. Appl. Math., 1966. V. 17. P. 263–269.

[65] *Graham R.L., Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G* Optimization and approximation in determenistic sequencing and scheduling: a servey // Ann. Descrete Optimization. 1979. V. 2. P. 287–325.

[66] *Hartmann S., Kolisch R.* Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem //European Journal of Operational Research, 2000, Vol. 127, 394 – 407.

[67] *Jackson J.R.* Scheduling a production line to minimize maximum tardiness // Los Angeles, CA: University of California, 1955. Manag. Sci. Res. Project. Research Report N 43.

[68] *Jain V., Grossmann I.E.* Algorithms for hybrid MILP/CLP models for a class of optimization problems // INFORMS J. Computing.– 2001.– V. 13.– P. 258 – 276.

[69] *Johnson S.M.* Optimal two- and three-stage production schedules with setup times included. // Naval Research Logistics Quarterly. 1954. V. 1. P. 61–68.

[70] *Kao E.P.C., Queyranne M.* On dynamic programming methods for assembly line balancing // Operations Research. 1982. V. 30. P. 375–390.

[71] *Keller H., Pferschy U., Pisinger D.* Knapsack problems // Springer. 2004. 546 p.

[72] *Kolish R., Padman R.* An Integrated Survey of Project Scheduling. 1997.

[73] *Kolisch R., Hartmann S.* Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis //Manuskripte aus den Instituten für Betriebswirtschaftslehre, 1998, No. 469, Kiel, Germany.

[74] *Koulamas C.P.* The total tardiness problem: Review and extensions // Operations Research. 1994. V. 42. P. 1025–1041.

[75] *Lageweg B.J., Lenstra J.K., Rinnooy Kan A.H.G.* Minimizing maximum lateness on one machine: computational experience and applications // Statistica Neerlandica.– 1976.– V. 30.– P. 25 – 41.

[76] *Lawler E.L.* On scheduling problems with deferral costs // Management Science. 1964. V. 11. P. 280–288.

[77] *Lawler E.L., Moore J.M.*, A Functional Equation and its Application to Resource Allocation and Sequencing Problems // Management Science, 1969. Vol. 16. No. 1. P. 77–84.

[78] *Lawler E.L.* A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness // Ann. Discrete Math. 1977. V. 1. P. 331–342.

[79] *Lawler E.L.* A fully polynomial approximation scheme for the total tardiness problem // Oper. Res. Lett. 1982. V. 1. P. 207–208.

[80] *Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G., Shmoys D.B.* Sequencing and Schedling: Algorithms and Complexity // Elsevier Science Publishers B.V. 1993. V. 4. P. 445–520.

[81] *Lenstra J.K., A.H.G. Rinnoy Kan* Complexity of scheduling under precedence constraints // Operation Research. 1978. V. 26. P. 22–35.

[82] *Lenstra J.K., Rinnooy Kan A.H.G., Brucker P.* Complexity of machine scheduling problems // European Journal of Operational Research. 1980. V. 4. P. 270–275.

[83] *Mastrolilli M.* Efficient Approximation Schemes for Scheduling Problems with Release Dates and Delivery Times // J. of Scheduling.– 2003.– V. 6, N 6.– P. 521 – 531.

[84] *McNaughton R.* Scheduling with deadlines and loss functions // Management Science. 1959. V. 6. P. 1–12.

[85] *Merkle D., Middendorf M., Schmeck H.* Ant Colony Optimization for Resource-Constrainted Project Scheduling // IEEE Transactions on Evolutionary Computation. 2002. Vol. 6. No. 4. P. 333–346.

[86] *Mingozzi A., Maniezzo V., Ricciardelli S., Bianco L.* An exact alforithm for project scheduling with recource constraints based on new mathematical formulation // Management Science. 1998. V. 44. P. 714–729.

[87] *Moore J.M.* An n job, one machine sequencing algorithm for minimizing the number of late jobs // Manag. Sci. 1968. V. 15. No. 1. P. 102–109.

[88] *Posypkin M.A., Sigal I. Kh. Speedup estimates for some variants of the parallel implementations of the branch-and-bound method*, Computational Mathematics and Mathematical Physics. 2006. Vol. 46, No. 12. P. 2189–2202.

[89] *Potts C.N., Van Wassenhove L.N.* A decomposition algorithm for the single machine total tardiness problem // Oper. Res. Lett. 1982. V. 5. P. 177–182.

[90] *Potts C.N., Van Wassenhove L.N.* A branch-and-bound algorithm for the weighted tardiness problem // Operations Research. 1985. V. 33. P. 363–377.

[91] *Potts C.N., Van Wassenhove L.N.* Dynamic programming and decomposition approaches for the single machine total tardiness problem European Journal of Operational Research. 1987. V. 32. P. 405–414.

[92] *Potts C.N., Van Wassenhove L.N.* Single machine tardiness sequencing heuristics // IIE Transactions. 1991. V. 23. P. 93–108.

[93] *Queyranne M., Schultz A.S.* Polyhedral approaches to machine scheduling // Preprint N 408/1994. Berlin: Technical University of Berlin, Department of Mathematics, 1994.– 61 p.

[94] *Rinnooy Kan A.H.G., Lageweg B.J., Lenstra J.K.* Minimizing total cost in one machine scheduling // Operations Research. 1975. V. 23. P. 908–927.

[95] *Rykov I.* Approximate solving of RCPSP // Abstract Guide of OR 2006, Karlsruhe 6.09-8.09, Germany, P. 226.

[96] *Sevastianov S.V., Tchernykh I.D.* Computer-Aided Way to Prove Theorems in Scheduling // Bilardi G., Italiano G.F., Pietracaptina

A., Pucci G. (eds.) Proceedings of Sixth Annual European Symposium on Algorithms ESA'98, 24 – 26 august.– Venice, Italy: 1998.– Springer-Verlag, LNCS, V. 1461, 1998.– P. 502 – 513.

[97] *Sevastianov S.V., Woeginger G.J.* Makespan minimization in open shops: A polynomial time approximation scheme. // Math. Prog.– 1998.– V. 82.– P. 191 – 198.

[98] *Smith W.E.* Various optimizers for single stage production // Naval Research Logistics Quarterly. 1956. V. 3. P. 59–66.

[99] *Sousa J.P., Wolsey L.A.* A time-indexed formulation of non-preemptive single-machine scheduling problems // Math. Prog.– 1992.– V. 54.– P. 353 – 367.

[100] *Szwarc W.* Single machine total tardiness problem revised // Creative and Innovate Approaches to the Science of Management, Quorum Books. 1993. P. 407–419.

[101] *Szwarc W., Della Croce F., Grosso A.* Solution of the single machine total tardiness problem // Journal of Scheduling. 1999. V. 2. P. 55–71.

[102] *Szwarc W., Grosso A., Della Croce F.* Algorithmic paradoxes of the single machine total tardiness problem // Journal of Scheduling. 2001. V. 4. P. 93–104.

[103] *Szwarc W., Mikhopadhyay S.* Decomposition of the single machine total tardiness problem // Oper. Res. Lett. 1996. V. 19. P. 243–250.

[104] *Uetz M.* Algorithms for Deterministic and Stochastic Scheduling Ph.D. thesis, Cuvillier Verlag, 2002.

[105] *Ullman J.D.* Complexity of sequencing problems // Coffman, 1976. P. 139–164.