Министерство образования Российской Федерации

Нижегородский государственный университет им. Н.И. Лобачевского

ЗАДАЧИ И МЕТОДЫ КОНЕЧНОМЕРНОЙ ОПТИМИЗАЦИИ

Учебное пособие

Часть 3.

Д.И. Коган

Динамическое программирование и дискретная многокритериальная оптимизация

Издательство Нижегородского университета

Нижний Новгород

2004

УДК 519.6

Коган Д.И. Динамическое программирование и дискретная многокритериальная оптимизация: учебное пособие.

Нижний Новгород: Изд-во Нижегородского ун-та, 2004. 150 с.

Рецензенты:

Часть 3 посвящена задачам дискретной оптимизации и методу динамического программирования как одному из наиболее эффективных инструментов их решения. Для записи общих соотношений динамического программирования вводится концепция дискретной управляемой системы, формулируются задачи синтеза оптимальных траекторий. Полученные уравнения позволяют строить оптимальные траектории методами прямого и обратного счета. Показывается, что в терминах дискретных управляемых систем записываются и решаются многие задачи дискретной оптимизации, включая задачи синтеза расписаний обслуживания. В реальных системах качество принимаемых решений следует, как правило, оценивать по ряду показателей, поэтому значительное внимание уделяется многокритериальным задачам. При этом рассматриваются вопросы синтеза представительных совокупностей эффективных оценок и соответствующих Парето-оптимальных решений; излагаемые процедуры основываются на соответствующих модификациях метода динамического программирования. Приводятся постановки и решающие процедуры для многокритериальных модификаций задачи о ранце, задачи коммивояжера, задач синтеза расписаний обслуживания и т.д. Значительное внимание уделяется вопросам вычислительной сложности. Для ряда труднорешаемых задач выделяются полиномиально разрешимые подклассы, строятся эвристические процедуры синтеза решений.

Пособие предназначено для студентов, специализирующихся в области прикладной математики и информатики.

ВВЕДЕНИЕ

Пособие посвящено задачам дискретной оптимизации и методу динамического программирования как одному из наиболее универсальных подходов к их решению. В основе метода динамического программирования лежит чрезвычайно просто формулируемый принцип Р. Беллмана: какова бы ни была предыстория управляемого процесса, приведшего систему в рассматриваемое состояние, последующие решения по отношению к этому состоянию должны быть оптимальными [3]. Динамическое программирование – принцип последовательного анализа протекающего во времени процесса. С целью записи рекуррентных соотношений динамического программирования в наиболее общей форме в пособии вводится концепция дискретной управляемой системы и ставятся вопросы синтеза ее оптимальных траекторий. Конструируются соотношения, позволяющие различными способами строить оптимальные траектории.

Показывается, что в терминах дискретных управляемых систем записываются и решаются многие задачи дискретной оптимизации. В числе рассматриваемых ряд классических оптимизационных задач (задача коммивояжера, задача о ранце, задача оптимального распределения капиталовложений) и их модификаций, а также ряд задач синтеза оптимальных расписаний обслуживания.

В связи с тем, что в реальных системах управления, планирования, проектирования качество принимаемых решений следует, как правило, оценивать по ряду показателей, значительное внимание уделяется многокритериальным задачам. Излагается основанная на многокритериальном аналоге принципа динамического программирования технология синтеза полных совокупностей эффективных оценок с одновременным обеспечением возможности определения по любой выбранной эффективной оценке Парето - оптимального решения, которое эту оценку порождает. Указанная технология использована при решении конкретных типов задач. В частности, излагаются постановки и решающие процедуры для многокритериальных обобщений задачи о ранце, задачи коммивояжера, задач о назначениях, а также для задач синтеза расписаний обслуживания в многокритериальных постановках.

Специально рассматриваются вопросы вычислительной сложности задач и алгоритмов. Проблема быстродействия решающих алгоритмов очень актуальна. В частности, жесткие ограничения на продолжительности циклов решения задач налагаются при синтезе расписаний для систем обслуживания, где между моментом, когда информация по подлежащей решению конкретной задаче определилась, и моментом начала реализация искомого расписания проходит очень небольшое время. В пособии для ряда труднорешаемых задач дискретной оптимизации выделяются полиномиально разрешимые подклассы; излагаются общие принципы построения приближенных и эвристических алгоритмов, строятся процедуры синтеза качественных в том либо ином смысле решений за реально приемлемое время. Для многокритериальных задач рассматриваются вопросы построения не полных, но достаточно представительных совокупностей эффективных оценок; излагаемые процедуры основываются на соответствующих модификациях метода динамического программирования.

Пособие в компактной форме излагает материал, имеющийся в известных, но зачастую труднодоступных монографиях и учебниках по динамическому программированию [3, 4, 5, 24]. Дополнительно рассматриваются вопросы применения метода динамического программирования к решению многокритериальных задач;

сответствующие теоретические результаты получены недавно и изложены только в журнальных публикациях и сборниках статей (см., например, [2, 11, 26].).

Глава 1. Метод динамического программирования в задачах дискретной оптимизации

Глава посвящена задачам дискретной однокритериальной оптимизации и методу динамического программирования как одному из наиболее эффективных инструментов их решения. Для записи общих соотношений динамического программирования при различных видах целевой функции вводится понятие дискретной управляемой системы, формулируются задачи синтеза ее оптимальных траекторий. Полученные рекуррентные соотношения позволяют строить искомые траектории выполнением процедур прямого или обратного счета. Показывается, что в терминах дискретных управляемых систем записываются и решаются многие задачи дискретной оптимизации.

1.1. Принцип динамического программирования. Основные рекуррентные соотношения динамического программирования.

Метод динамического программирования – один из основных математических методов оптимизации. При его реализации одновременный выбор значений большого числа переменных решаемой экстремальной задачи заменяется поочередным определением каждой из них в зависимости от возможных обстоятельств. Процедуру выбора значений переменных будем трактовать как многоэтапный процесс управления некоторой системой. Далее объект Ω именуем дискретной управляемой системой, если множество его возможных состояний конечно и управления в нем осуществляются в дискретном времени, пошагово; каждое управление заключаются в применении одного из конечного числа возможных воздействий; результатом любого воздействия является изменение состояния системы. С каждым изменением состояния системы связываем платеж (доход или расход); полагаем, что величина платежа на любом шаге зависит от текущего состояния системы и применяемого управляющего воздействия. Начальное состояние системы считается заданным. Определено также множество финальных состояний (достигнув финального состояния, система прекращает функционировать). Каждая последовательность управлений, переводящая систему из начального в одно из финальных состояний, фактически определяет некоторую полную траекторию движения системы. Требуется найти полную траекторию, оптимальную по значению суммарного платежа, т.е. обеспечивающую максимальный суммарный доход или минимальный суммарный расход. Возможно и другое требование – найти полную траекторию с минимальным значением максимального из пошаговых расходов (максимальным значением минимального из пошаговых доходов).

Формально *дискретную управляемую систему* определяем как совокупность

$$\Omega = \{D;\ x_0;\ F;\ V(x),\ f(x,v),\ s(x,v)\},$$

где D – множество *состояний системы* (множество D конечно); x_0 – *начальное состояние*; F – *множество финальных состояний*, $x_0 \in D$, $x_0 \notin F$, $F \subset D$; $V(x)$ – конечное множество возможных в состоянии x *управлений*, $x \in D \setminus F$; $f(x, v)$ – *функция переходов* (из состояния x под воздействием управления v система переходит в

состояние $f(x,v)$), $x \in D \backslash F$, $v \in V(x)$, $f(x, v) \in D$; $s(x,v)$ – *функция платежа*, здесь $x \in D \backslash F$, $v \in V(x)$; значения функции платежа считаются неотрицательными.

Состояние x системы Ω называем *промежуточным*, если оно не является ни начальным, ни финальным.

Конечную последовательность $T = \{x^0, x^1, x^2, \ldots, x^n\}$ состояний системы Ω именуем *траекторией системы*, если выполняются условия:

$$x^t = f(x^{t-1}, v^t), \text{ где } v^t \in V(x^{t-1}), \ t = 1, 2, \ldots, n.$$

Состояние x^0 – *начальное состояние траектории* T, а состояние x^n – ее *конечное состояние.* Состояния $x^2, x^3 \ldots, x^{n-1}$ из T являются *промежуточными состояниями* данной траектории.

Траекторию называем *полной*, если начальным ее состоянием является начальное состояние системы Ω, а конечным – одно из финальных состояний этой системы. Таким образом, для полной траектории $T = \{x^0, x^1, x^2, \ldots, x^n\}$ имеет место $x^0 = x_0$ и $x^n \in F$. Траекторию $T = \{x^0, x^1, x^2, \ldots, x^n\}$ называем *заключительной x-траекторией*, если $x^0 = x$ и $x^n \in F$. Заключительная x -траектория, имея своим начальным состоянием произвольное состояние x системы Ω, заканчивается в одном из финальных состояний системы. Траекторию $T = \{x^0, x^1, x^2, \ldots, x^n\}$ называем *начальной x-траекторией*, если $x^0 = x_0$ и $x^n = x$. Начальная x-траектория, имея своим конечным состоянием x, начинается от начального состояния системы.

На множестве состояний системы Ω введем бинарное отношение достижимости $P(x,y)$: считаем, что произвольные состояния x и y удовлетворяют этому отношению, если существует траектория T, имеющая своим начальным состоянием x, а конечным – состояние y. Отметим, что введенное бинарное отношение обладает свойствами рефлексивности (для любого состояния x имеет место $P(x,x)$) и транзитивности (для любых x, y, z из того, что имеют место $P(x,y)$ и $P(y,z)$ следует, что имеет место и $P(x,z)$). Дополнительно предполагаем, что отношение $P(x,y)$ антисимметрично (из $P(x,y)$ и $P(y,x)$ вытекает $x = y$). Антисимметричность бинарного отношения достижимости означает, что в любой своей траектории система не может реализовывать циклы (возвращаться в состояния, в которых она уже была). Обладая свойствами рефлексивности, транзитивности и антисимметричности, бинарное отношение $P(x,y)$ является отношением частичного порядка.

Состояние x системы Ω именуем *достижимым*, если имеет место $P(x_0, x)$. Отметим, что начальное состояние x_0 достижимо по определению. Состояние x системы Ω именуем *продуктивным*, если для некоторого финального состояния f имеет место $P(x, f)$. Финальные состояния системы продуктивны по определению. Недостижимые и непродуктивные состояния системы Ω можно изъять из рассмотрения; далее, как правило, будет считаться, что каждое состояние системы достижимо и продуктивно. При реальном исследовании систем, порождаемых конкретными оптимизационными задачами, вопрос выделения состояний, являющихся одновременно достижимыми и продуктивными, заслуживает отдельного анализа. Избавление от недостижимых и непродуктивных состояний способствует сокращению объемов выполняемых вычислений.

Пусть $T = \{x^0, x^1, x^2, \ldots, x^n\}$ – произвольная траектория системы Ω, причем $x^t = f(x^{t-1}, v^t)$, где $v^t \in V(x^{t-1})$, $t = 1, 2, \ldots, n$. *Стоимость C(T)* траектории T определяем следующим образом:

$$C(T) = \sum_{t=1}^{n} s(x^{t-1}, v^t).$$

Таким образом, стоимость траектории – это сумма пошаговых платежей, имеющих место при реализации траектории.

Центральной является задача построения полной траектории, оптимальной по значению суммарной стоимости, т.е. обеспечивающей максимальный суммарный доход или минимальный суммарный расход. Обе экстремальные задачи по способу решения идентичны. Для определенности проблему формулируем следующим образом.

Задача A**.** Для системы Ω найти полную траекторию T минимальной стоимости.

Дополнительно введем в рассмотрение совокупность частных задач $A(x)$, где x принадлежит $D \backslash F$.

Задача $A(x)$**.** Для системы Ω и ее состояния x найти заключительную x-траекторию минимальной стоимости.

Решения сформулированных экстремальных задач A и $A(x)$ именуем оптимальной полной траекторией и оптимальной заключительной x-траекторией соответственно.

В основе метода динамического программирования лежит принцип оптимальности, сформулированный американским математиком Ричардом Беллманом следующим образом: "Оптимальное поведение обладает тем свойством, что каковы бы ни были первоначальное состояние и решение в начальный момент, последующие решения должны составлять оптимальное поведение относительно состояния, являющегося результатом первого решения".

Следующая теорема выражает ***принцип Беллмана*** в применении к задачам построения оптимальных траекторий.

Т е о р е м а 1.1. Если траектория $T = \{x^0, x^1, x^2, \ldots, x^n\}$ оптимальна, то любая ее заключительная часть $T_k = \{x^k, x^{k+1}, \ldots, x^n\}$ также оптимальна.

Теорема легко доказывается методом от противного. Предположим, что траектория $T = \{x^0, x^1, x^2, \ldots, x^k, x^{k+1}, \ldots, x^n\}$ оптимальна, а ее заключительная часть $T_k = \{x^k, x^{k+1}, \ldots, x^n\}$ оптимальной не является. Стоимость траектории T представим как $C(T) = P + Q$, где $P = \sum_{t=1}^{k} s(x^{t-1}, v^t)$ и $Q = \sum_{t=k+1}^{n} s(x^{t-1}, v^t)$. Оптимальную заключительную x^k-траекторию запишем в виде $T^1_k = \{y^k, y^{k+1}, \ldots, y^m\}$, здесь $y^k = x^k$; $y^{k+i} = f(y^{k+i-1}, \mu^i)$, где $\mu^i \in V(y^{k+i-1}), i = 1,2,\ldots, m-k$; $y^m \in F$. Пусть $R = \sum_{i=1}^{m-k} s(y^{k+i-1}, \mu^i)$. Так как y^k-траектория T^1_k оптимальна, то $R < Q$. Рассмотрим траекторию $T^* = \{x^0, x^1, x^2, \ldots, x^k, y^{k+1}, y^{k+2}, \ldots, y^m\}$. Как очевидно, $C(T^*) = P + R$. Далее имеем $C(T^*) = P + R < P + Q = C(T)$. Таким образом, $C(T^*) < C(T)$. Последнее неравенство означает, что траектория T неоптимальна. Справедливость теоремы вытекает из полученного противоречия.

Стоимость оптимальной в задаче A полной траектории обозначим символом B_A. Стоимость заключительной x-траектории, оптимальной в задаче $A(x)$, обозначим $B(x)$. Функция $B(x)$ называется функцией Беллмана. Отметим, что $B_A = B(x_0)$.

Очевидны следующие равенства:

$$B(x) = 0,\ x \in F. \quad (1.1)$$

Пусть x - произвольное нефинальное состояние системы. В этом состоянии можно реализовать любое принадлежащее множеству $V(x)$ управление. Предположим, что выбрано управление v, $v \in V(x)$. Данное управление влечет платеж $s(x,v)$, а следующим состоянием системы оказывается $f(x, v)$. Если далее реализуется оптимальная заключительная $f(x, v)$- траектория, то суммарная величина последующих платежей оказывается равной $B(f(x,v))$. Из принципа Беллмана получаем:

$$B(x) = \min_{v \in V(x)} \{s(x, v) + B(f(x,v))\},\ x \in D \setminus F. \quad (1.2)$$

Вычисление значений функции Беллмана по соотношениям (1.1) - (1.2) выполняется поэтапно в следующем порядке. На первом этапе фиксируются значения $B(x) = 0$ для всех $x \in F$. Далее на каждом следующем этапе вычисление очередного значения функции Беллмана выполняется для произвольного состояния x такого, что $B(x)$ неизвестно, но значения $B(y)$ для всех непосредственно следующих за x состояний y уже найдены (состояние y системы Ω относим к числу непосредственно следующих за состоянием x, если пара $\{x, y\}$ является траекторией системы Ω). Последним в процессе счета определяется значение $B(0)$.

Для синтеза оптимальной полной траектории системы при выполнении определяемой соотношениями (1.1)-(1.2) процедуры счета следует дополнительно составить *список оптимальных переходов* (СОП). Каждая запись этого списка имеет вид $[x_i, x_j]$, где x_i – произвольное нефинальное состояние, а $x_j = f(x_i, v^*)$ – состояние, в которое система переходит из x_i под воздействием управления v^*, возможного в состоянии x_i и обращающего в минимум правую часть (1.2); если таких управлений несколько, фиксируется любое из них (нашей целью считаем построение любой из оптимальных траекторий, но не всех таких траекторий). Записи СОП взаимно однозначно соответствуют нефинальным состояниям системы, при этом запись $[x_i, f(x_i, v)]$ считается соответствующей состоянию x_i. Составленный СОП полагаем упорядоченным по возрастанию индексов первых компонент входящих в него записей. Чтобы построить оптимальную траекторию, из СОП извлекаем последовательность записей W, первым элементом которой является запись, соответствующая состоянию x_0; каждая следующая запись последовательности W имеет своей первой компонентой вторую компоненту предыдущей записи этой последовательности. В последней записи последовательности W второй компонентой является некоторое финальное состояние системы Ω. Из составленного СОП последовательность W извлекается однозначно. Пусть $W = \{[x_0, y_1], [y_1, y_2], [y_2, y_3], \ldots, [y_{m-1}, y_m]\}$, здесь $y_m \in F$. Тогда $T = \{x_0, y_1, y_2, y_3, \ldots, y_m\}$ – искомая оптимальная траектория.

Уравнения (1.1) – (1.2) – *рекуррентные соотношения динамического программирования* для решения задачи A. Основанный на этих соотношениях метод решения задачи A называется *методом динамического программирования.*

Пусть N – число состояний системы Ω. Тогда верхняя оценка числа вычисляемых значений функции Беллмана в процессе решения задачи A равна N. Число элементарных операций, выполняемых при определении по формуле (1.2)

каждого конкретного значения этой функции не превышает CN, где C - некоторая не зависящая от N константа. Таким образом, верхней оценкой числа элементарных операций, выполняемых при решении задачи $\boldsymbol{A}$ методом динамического программирования является CN^2, где N – число состояний системы $\boldsymbol{\Omega}$, а C – не зависящая от N константа. Для запоминания в процессе вычислений всех найденных значений функции Беллмана необходим объем памяти, пропорциональный N; объем памяти такого же порядка нужен для создания списка оптимальных переходов. В итоге получаем, что объем памяти, нужной для решения задачи **A** методом динамического программирования, линейно зависит от N.

Следует отметить, что в связи с конечностью числа состояний системы $\boldsymbol{\Omega}$ число ее траекторий конечно и задача $\boldsymbol{A}$ в принципе может быть решена путем перебора конечного числа вариантов. Метод динамического программирования позволяет определенным образом упорядочить и существенно сократить перебор.

Систему $\boldsymbol{\Omega}$ можно представлять конечным взвешенным ориентированным графом $G(\boldsymbol{\Omega})$, вершины которого взаимно однозначно соответствуют состояниям системы, дуги – управлениям, веса дуг – стоимостям соответствующих переходов. Вначале положим, что вершины названы именами соответствующих состояний. Вершину $\boldsymbol{x_0}$ именуем начальной; вершины, соответствующие состояниям подмножества $\boldsymbol{F}$, называем финальными. В графе $G(\boldsymbol{\Omega})$ дуга, идущая из произвольной вершины $\boldsymbol{x}$ в вершину $\boldsymbol{y}$, проводится тогда и только тогда, когда в системе $\boldsymbol{\Omega}$ для некоторого возможного в состоянии $\boldsymbol{x}$ управления $\boldsymbol{v^*}$ $(\boldsymbol{v^* \in V(x)})$ имеет место $\boldsymbol{y = f(x, v^*)}$; вес этой дуги полагаем равным $\boldsymbol{s(x, v^*)}$. Вершину $\boldsymbol{x}$ называем *непосредственно предшествующей* вершине $\boldsymbol{y}$, если в графе имеется дуга $(\boldsymbol{x, y})$. В графе $G(\boldsymbol{\Omega})$ путь из произвольной вершины $\boldsymbol{x}$, $\boldsymbol{x \in D}$, в вершину $\boldsymbol{y}$, $\boldsymbol{y \in D}$, имеется тогда и только тогда, когда выполняется бинарное отношение $P(\boldsymbol{x,y})$. Так как $P(\boldsymbol{x,y})$ – отношение частичного порядка, граф $G(\boldsymbol{\Omega})$ – ациклический. Из свойств транзитивности и антисимметричности бинарного отношения $P(\boldsymbol{x, y})$ и сделанного предположения о достижимости любого состояния системы $\boldsymbol{\Omega}$ вытекает, что в графе $G(\boldsymbol{\Omega})$ нет дуг, входящих в вершину $\boldsymbol{x_0}$. По определению системы $\boldsymbol{\Omega}$, достигнув финального состояния, она прекращает функционировать; поэтому в графе отсутствуют дуги, исходящие из финальных вершин.

Вершины графа $G(\boldsymbol{\Omega})$ нумеруем целыми неотрицательными числами. Вершине $\boldsymbol{x_0}$ (начальной вершине) присваиваем номер 0. Далее реализуется многоэтапная процедура, на каждом шаге которой из совокупности пока непронумерованных выбирается какая-либо вершина, для которой все непосредственно предшествующие вершины уже пронумерованы; выбранной вершине присваивается следующий (в порядке возрастания) номер. Введенные номера далее будем считать новыми наименованиями вершин графа и соответствующих им состояний системы $\boldsymbol{\Omega}$. Отсюда имеем следующее: если для некоторых состояний i , j и управления $\boldsymbol{v}$, $\boldsymbol{v \in V(i)}$, имеет место $j = \boldsymbol{f(i, v)}$, то $j > i$.

При выполненной нумерации в вычислительной процедуре, определяемой соотношениями (1.1) - (1.2), значения функции Беллмана для состояний системы $\boldsymbol{\Omega}$ вычисляются последовательно, в порядке убывания их номеров. Поэтому в ситуации, когда для состояния i по соотношению (1.2) определяется значение $\boldsymbol{B(i)}$, все значения $\boldsymbol{B(f(i,v))}$ при $\boldsymbol{v \in V(i)}$ уже известны. Последней в процедуре вычисляется величина $\boldsymbol{B(0)}$ - оптимальное значение критерия в решаемой задаче $\boldsymbol{A}$.

ПРИМЕР 1.1. Имеется дискретно управляемая система система Ω^l, определяющий ее граф $G(\Omega^l)$ представлен на рис. 1.1. Начальным является состояние *0*, множество финальных состояний одноэлементно: ***F={9}.*** Требуется найти полную траекторию минимальной стоимости.

Для состояний системы Ω^l (вершин графа $G(\Omega^l)$) вычисляем значения функции Беллмана. На основании (1.1) фиксируем ***B(9)=0.*** Далее, пользуясь (1.2), последовательно получаем: ***B(8)=1; B(7)= min[3, 1+B(8)] = 2; B(6)=1+B(7)=3; B(5)=min[1+ B(6), 3+B(7), 7+B(8] = min[4, 6, 8] = 4; B(4)=min[2+ B(5), 3+B(8), 5+B(9)] = min[6, 4, 5] = 4; B(3)=min[3+ B(5), 4+B(4)] = min[7, 8] = 7; ; B(2)=min[1+ B(3), 5+B(5)] = min[8, 9] = 8; B(1)=min[9+ B(6), 6+B(2)] = min[12, 14] = 12; B(0)=min[1+ B(1), 5+B(2), 4+B(3] = min[13, 13, 11] = 11.*** Таким образом, минимальная из стоимостей полных траекторий в рассматриваемой задаче равна 11.

При подсчетах значений функции Беллмана в графе $G(\Omega^l)$ для каждой нефинальной вершины специально выделяем (рисуем жирной линией) дугу, соответствующую управлению, в ней принимаемому (т.е. обращающему в минимум правую часть (1.2)). Получаемый результат представлен на рис. 1.2.

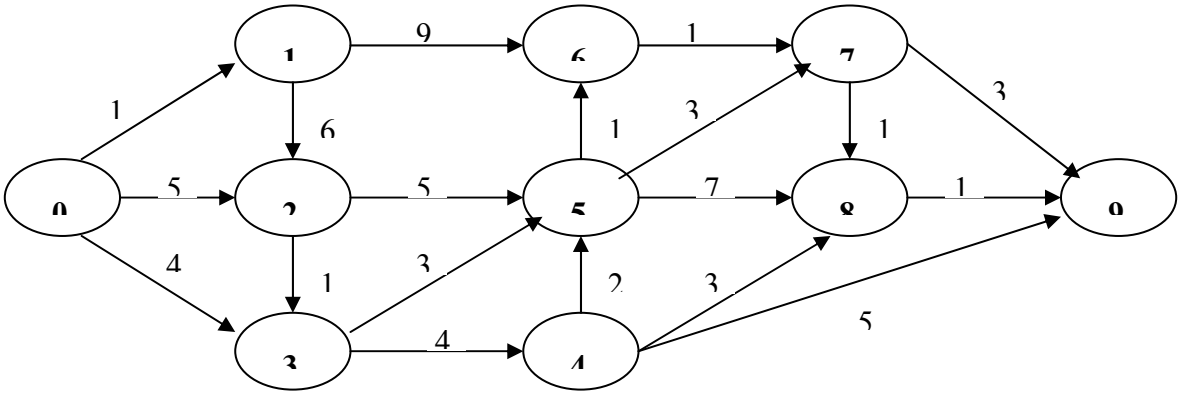


Рисунок 1.1.Граф $G(\Omega^l)$

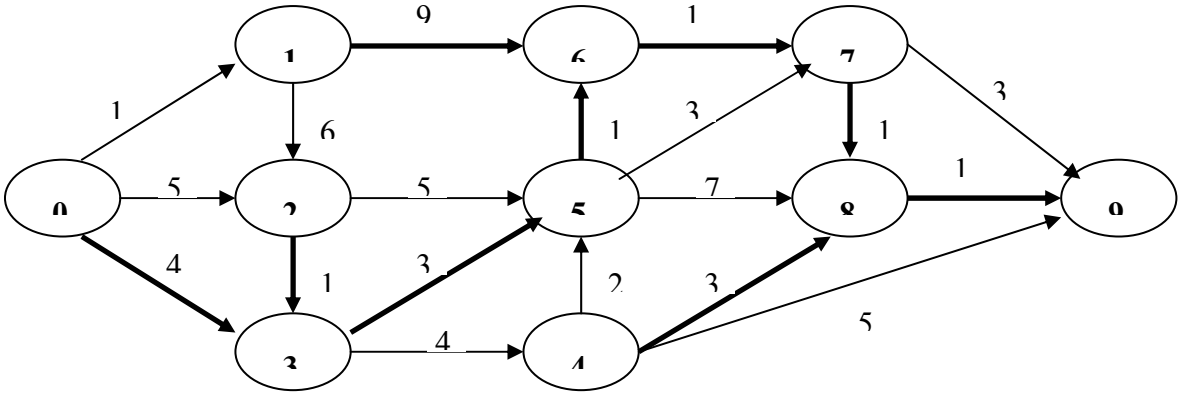


Рисунок 1.2. Граф $G(\Omega^l)$ с выделенными дугами.

Как показывают выделенные дуги, на первом такте процесса управления следует выполнить переход из вершины *0* в вершину *3*, на втором такте - из вершины *3* в вершину *5*, на третьем такте - из вершины *5* в вершину *6*, на четвертом такте - из

вершины *6* в вершину *7*, на пятом такте - из вершины *7* в вершину *8*, на шестом, заключительном такте - из вершины *8* в вершину *9*. Таким образом, оптимальной по критерию суммарной стоимости полной траекторией является $T = \{0,3,5,6,7,8,9\}$.

Далее введем совокупность частных задач следующего вида.

Задача *A*(x).* Для системы $\boldsymbol{\Omega}$ и ее состояния $\boldsymbol{x}$ найти начальную $\boldsymbol{x}$-траекторию минимальной стоимости.

Оптимальное решение задачи ***A*(x)*** именуем оптимальной начальной $\boldsymbol{x}$-траекторией системы $\boldsymbol{\Omega}$.

Имеет место следующий факт.

Т е о р е м а 1.2. Если траектория $T= \{x^0, x^1, x^2, \ldots, x^n\}$ оптимальна, то любая ее начальная часть также оптимальна.

Доказательство данной теоремы идентично доказательству теоремы 1.1.

Стоимость начальной $\boldsymbol{x}$-траектории, оптимальной в задаче ***A*(x)***, обозначим ***B*(x)***. Очевидно, что

$$\boldsymbol{B^*(x_0)} = 0. \qquad (1.3)$$

В системе $\boldsymbol{\Omega}$ произвольное состояние $\boldsymbol{x}$ непосредственно предшествует состоянию $\boldsymbol{y}$, если пара $\{\boldsymbol{x}, \boldsymbol{y}\}$ является траекторией системы $\boldsymbol{\Omega}$. Управление, переводящее систему $\boldsymbol{\Omega}$ из состояния $\boldsymbol{x}$ непосредственно в состояние $\boldsymbol{y}$, обозначим $\boldsymbol{v[x, y]}$. Множество состояний системы, непосредственно предшествующих состоянию $\boldsymbol{y}$, обозначим $\Gamma(\boldsymbol{y})$. Основываясь на теореме 1.2, запишем соотношение, позволяющее организовать процесс вычислений значений функции ***B*(y)*** для всех состояний $\boldsymbol{y}$, отличных от начального:

$$\boldsymbol{B^*(y)} = \min_{x \in \Gamma(y)} \{ \boldsymbol{B^*(x) + s(x, v[x, y])} \}, \quad \boldsymbol{y \in D \setminus x_0}; \qquad (1.4)$$

Отметим, что

$$\boldsymbol{B_A} = \min_{x \in F} \boldsymbol{B^*(x)}. \qquad (1.5)$$

Вычисление значений функции ***B*(y)*** на основе рекуррентных соотношений (1.3) - (1.4) выполняется поэтапно в следующем порядке. На первом этапе фиксируется значение $\boldsymbol{B^*(x_0)} = 0$. Далее на каждом следующем этапе вычисление очередного значения функции ***B**** выполняется для произвольного состояния $\boldsymbol{y}$ такого, что ***B*(y)*** неизвестно, но значения ***B*(x)*** для всех состояний $\boldsymbol{x}$, непосредственно предшествующих состоянию $\boldsymbol{y}$, уже найдены.

Метод решения задачи $\boldsymbol{A}$, основанный на соотношениях (1.3) – (1.5), – *прямой метод Беллмана*. В отличие от него, метод, основанный на соотношениях (1.1) – (1.2), носит название *обратного метода Беллмана*. Функции ***B*(x)*** и ***B(x)*** будем именовать функциями Беллмана для прямого и обратного счета соответственно. Реализация как обратного, так и прямого метода Беллмана для решения задачи $\boldsymbol{A}$ требует квадратично зависящего от числа состояний системы $\boldsymbol{\Omega}$ количества элементарных операций. Иногда прямой и обратный метод Беллмана, подразумевая, что задача решается с использованием соотношений динамического программирования, называют *методами прямого и обратного счета* соответственно.

ПРИМЕР 1.2. Имеется дискретно управляемая система система $\boldsymbol{\Omega^l}$, определяющий ее граф $G(\boldsymbol{\Omega^l})$ представлен на рис. 1.1. Начальным является состояние *0*, множество финальных состояний одноэлементно: ***F={9}.*** Методом прямого счета требуется найти полную траекторию минимальной стоимости.

Для состояний системы $\boldsymbol{\Omega^l}$ (вершин графа $G(\boldsymbol{\Omega^l})$) вычисляем значения прямой функции Беллмана. На основании (1.3) фиксируем ***B*(0)=0.*** Далее, пользуясь (1.4), последовательно получаем: ***B*(1)=1; B*(2)= min[5, B*(1)+6] = 5; B*(3)= min[4, B*(2)+1] = 4; B*(4)= B*(3)+4 = 8; B*(5)=min[B*(2)+ 5, B*(3)+ 3, B*(4)+2] = 7; B*(6)=min[B*(1)+ 9, B*(5)+ 1] =8; B*(7)=min[B*(6)+ 1, B*(5)+3] =9; B*(8)=min[B*(5)+ 7, B*(7)+1] =10; B*(9)=min[B*(7)+ 3, B*(8)+1, B*(4)+5] =11.***

При вычислении значений функции Беллмана в графе $G(\Omega^l)$ для каждой вершины ***y***, отличной от начальной, специально выделяем (рисуем жирной линией) дугу (***x, y***), где ***x*** - вершина для которой значение правой части (1.4) при подсчете ***B*(y)*** оказывается минимальным (этот способ выделения дуг с учетом используемого метода счета называем *прямым*; способ выделения, использованный в решении предыдущего примера, именуем *обратным*). Полученный результат представлен на рис. 1.3. Как показывают выделенные дуги, на последнем такте процесса управления следует выполнить переход из вершины *8* в вершину *9*, при этом переход в вершину *8* реализуется из вершины *7*, в вершину *7* - из вершины *6*, в вершину *6* - из вершины *5*, в вершину *5* - из вершины *3*, в вершину *3* - из вершины *0*. Оптимальной является траектория *{0,3,5,6,7,8,9}*.

Полученные в результате решения примеров 1.1 и 1.2 оптимальные в системе $\boldsymbol{\Omega^l}$ траектории совпали.

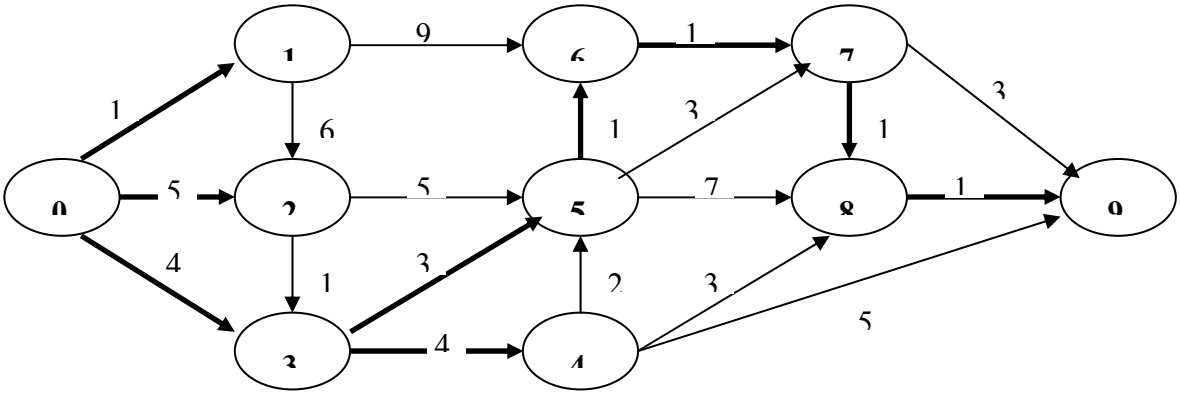


Рисунок 1.3. Граф $G(\boldsymbol{\Omega^l})$ с выделенными дугами (способ выделения - прямой).

Отметим следующее обстоятельство. Пусть в процессе вычислений по соотношениям (1.3) - (1.4) для некоторого финального состояния ***x**** оказалось найденным значение ***B*(x*)***; полную траекторию стоимости ***B*(x*)***, имеющую своим конечным состоянием ***x*,*** обозначим *T**. Если каждая из остальных траекторий имеет в своем составе как минимум одно состояние из некоторого подмножества состояний *Q*, причем все значения ***B*(x)*** при $\boldsymbol{x} \in Q$ уже вычислены и оказались не меньшими ***B*(x*)***, то *T**.– полная оптимальная траектория. Сделанное утверждение нередко дает возможность ограничить объем требуемых вычислений.

Подмножество промежуточных состояний *M* системы $\boldsymbol{\Omega}$ назовем *разрезающим*, если каждая полная траектория ***T*** этой системы имеет в своем составе состояния из *M*. Разрезающее подмножество *M** назовем *минимальным разрезающим*, если среди его

собственных подмножеств нет разрезающих. Минимальная из стоимостей полных траекторий системы Ω равна

$$\min_{x \in M^*} (B^*(x). + B(x)), \qquad (1.6)$$

здесь M^* – любое минимальное разрезающее подмножество состояний системы Ω.

Основанный на данном факте метод решения задачи синтеза полной траектории минимальной стоимости называется *методом встречного счета*.

Для системы Ω^1, определяемой представленным на рис. 1.1 графом $G(\Omega^1)$, минимальным разрезающим является, в частности, подмножество вершин-состояний $M^* = \{4, 5, 6\}$. При решении задачи синтеза оптимальной по стоимости полной траектории методом встречного счета выполняем следующие действия. Прямым счетом последовательно определяем: $B^*(1)=1; B^*(2)= 5; B^*(3) = 4; B^*(4)= 8; B^*(5) = 7; B^*(6) = 8$. Обратным счетом находим: $B(8)=1; B(7) = 2; B(6) =3; B(5)= 4; B(4) = 4$. Для принадлежащих подмножеству M^* состояний находим: $B^*(4)+ B(4)= 12; B^*(5)+ B(5)= 11; B^*(6)+ B(6)= 11$. Получаем, что минимальная из стоимостей полных траекторий системы Ω^1 равна *11*.

Оптимальные полные траектории системы Ω^1 (их, в принципе, может быть несколько) проходят через вершины *5* и *6*. При выполненном подсчете значений функции $B^*(x)$, см. решение примера 1.2, определено, что оптимальной начальной *5*-траекторией системы Ω^1 является $T^*(5) = \{0,3,5\}$, а оптимальной начальной *6*-траекторией этой системы является $T^*(6) = \{0,3,5,6\}$. При подсчете значений функции $B(x)$, см. решение примера 1.1, установлено, что оптимальной заключительной *5*-траекторией системы Ω^1 является $T(5) = \{5,6,7,8,9\}$, а оптимальной заключительной *6*-траекторией этой системы является $T(6) = \{6,7,8,9\}$. Получаем, что единственной оптимальной полной траекторией системы Ω^1 является $T = \{0,3,5,6,7,8,9\}$.

Метод встречного счета оказывается эффективным, в частности, при решении задач в вычислительных системах с несколькими процессорами.

Выше траектории системы Ω оценивались по показателю суммарной стоимости. В некоторых задачах целесообразной оценкой каждой траектории $T = \{x^0, x^1, x^2, \ldots, x^n\}$ является величина максимального из пошаговых платежей в ходе реализации этой траектории. Введем показатель

$$K(T) = \max_{t=1,2,\ldots,n} s(x^{t-1}, v^t)$$

и рассмотрим задачу синтеза полной траектории с минимальным значением этого показателя. Обозначим через $T(\Omega)$ совокупность всех полных траекторий системы Ω. Сформулированная задача записывается в виде

$$\min_{T \in T(\Omega)} \max_{t=1,2,\ldots,n} s(x^{t-1}, v^t). \qquad (1.7)$$

Оптимальную в задаче (1.7) траекторию будем именовать *минимаксной полной траекторией системы* Ω.

Идентичным образом вводится понятие *минимаксной заключительной* x *- траектории системы* Ω.. Имеет место следующий факт.

Т е о р е м а 1.3. Если $T= \{x^0, x^1, x^2, ..., x^k, x^{k+1}, ..., x^n\}$ – минимаксная полная траектория, а $T' = \{y^1, y^2, ..., y^m\}$ – минимаксная заключительная x^k - траектория, здесь $y^1 = x^k$, то траектория $\{x^0, x^1, x^2, ..., x^k, y^2, ..., y^m\}$ – минимаксная полная траектория.

Доказательство данной теоремы легко выполняется методом от противного.

Из теоремы 1.3 вытекает

Следствие 1.1. В системе Ω имеется минимаксная полная траектория, любая заключительная часть которой также минимаксна.

Максимальный из пошаговых платежей при реализации минимаксной полной траектории T системы Ω обозначим B'. Максимальный из пошаговых платежей при реализации минимаксной заключительной **x**-траектории системы Ω обозначим $B'(x)$. Функция $B'(x)$ – функция Беллмана для рассматриваемой минимаксной задачи.

Очевидны следующие равенства:

$$B'(x_k) = 0, \quad x_k \in F . \qquad (1.8)$$

Пользуясь следствием из теоремы 3, запишем общее уравнение

$$B'(x_i) = \min_{v \in V(x_1)} \{\max \{s(x_i, v) , B'(f(x_i,v))\} \}, \quad x_i \in D \backslash F. \qquad (1.9)$$

Формулы (1.8)-(1.9) – рекуррентные соотношения динамического программирования для задачи синтеза минимаксной полной траектории в системе Ω. В вычислительной процедуре, определяемой этими соотношениями, значения функции Беллмана для нефинальных состояний системы Ω вычисляются последовательно, в порядке убывания их номеров. В ситуации, когда для состояния $\mathbf{x}_i$ по формуле (1.9) определяется значение $B'(x_i)$, все значения $B'(f(x_i,v))$ при $v \in V(x_i)$ уже известны. Последней в реализации процедуры вычисляется величина $B'(x_0)$ – оптимальное значение критерия в решаемой задаче (1.7). Синтезируемая с использованием рекуррентных соотношений (1.8)-(1.9) минимаксная полная траектория такова, что любая ее заключительная часть также минимаксна. Количество элементарных операций, необходимых для решения изложенным способом задачи (1.7) квадратично зависит от от числа состояний системы Ω.

Идентичным образом вводится понятие максиминной полной траектории и записываются соотношения динамического программирования для ее синтеза.

В рамках моделей управления системами описанного вида формулируются и решаются многие типы задач дискретной оптимизации. Число состояний системы Ω, конструируемой по исходным данным решаемой оптимизационной задачи, от размерности этой задачи очень часто зависит экспоненциально. Данное обстоятельство является причиной значительных затрат на время вычислений уже при средних размерностях решаемых задач. При изучении каждого конкретного класса оптимизационных задач актуальны проблемы построения моделирующих систем с возможно меньшим числом состояний и синтеза алгоритмов, которые экономят время вычислений и объем используемой памяти за счет отыскания только тех значений функции Беллмана, которые действительно оказываются необходимыми.

1.2. Решение методом динамического программирования дискретных оптимизационных задач.

В каждой из рассматриваемых ниже задач оптимальное решение в принципе может быть получено полным перебором возможных вариантов с подсчетом значения оптимизируемого критерия для каждого из таких вариантов. Множество вариантов конечно, но число их, как правило, уже при относительно малых размерностях решаемых задач оказывается столь большим, что полный перебор практически неосуществим. Использование схемы динамического программирования дает возможность определенным образом упорядочить перебор, существенно сократив при этом число подлежащих рассмотрению вариантов.

Задача оптимального распределения капиталовложений (ЗОРК). Имеются денежная сумма (капитал) C и возможные сферы вложений $S_1, S_2,\ldots,S_n$. Считается, что в каждую сферу S_t может быть вложен целочисленный вклад, не превосходящий константы C_t. Для каждой сферы S_t полагается известной монотонно возрастающая в нестрогом смысле функция $f_t(u)$, определяющая величину дохода, получаемого при вложении в данную сферу вклада u. Требуется определить обеспечивающее максимальный суммарный доход распределение суммы C или ее части по сферам вложений.

Математическая модель задачи следующая:

$$\sum_{i=1}^{n} f_t(u_t) \to \max$$

при условиях:

$$\sum_{i=1}^{n} u_t \le C;$$

$$u_t \in \{0,1, \ldots, C_t\},\ t = \overline{1,n}.$$

Покажем, что сформулированную задачу можно трактовать как задачу поиска полной траектории, обеспечивающей максимальный суммарный доход в соответствующим образом построенной дискретной управляемой системе. Исходные данные *ЗОРК* определяют систему Ω*(ЗОРК),* управления в которой осуществляется в дискретном времени $t=0, 1, 2, \ldots, n-1$ (в момент t определяется вклад в сферу S_{t+1}). Состояния системы - пары (t, U), где t – момент дискретного времени, а U – сумма, которая к данному моменту еще не распределена, здесь $t \in\{0,1,\ldots, n\}$, $U\in\{0,1,\ldots, C\}$. Начальное состояние системы $(0,C)$; финальными являются состояния вида (n, U). В любом нефинальном состоянии (t, U) выбор реализуемого в этом состоянии управления u_{t+1} осуществляется из совокупности $V(t,U) =\{0,1,\ldots, \min (C_{t+1},U)\}$; выбранная величина u_{t+1} - размер вклада в сферу S_{t+1} Под воздействием управления u_{t+1} система Ω*(ЗОРК*) из состояния (t, U) переходит в состояние $(t + 1, U - u_{t+1})$, получаемый при этом доход равен $f_{t+1}(u_{t+1})$. Каждая последовательность управлений, переводящая систему из начального состояния в одно из финальных состояний, определяет некоторое допустимое решение *ЗОРК* (распределение $(u_1, u_2, \ldots, u_n)$ капиталовложений). Исходную *ЗОРК* трактуем как задачу отыскания траектории системы Ω*(ЗОРК),* обеспечивающей максимальный суммарный доход.

Функция Беллмана для системы Ω*(ЗОРК)* определяется соотношениями

$$\boldsymbol{B}(n, U) = 0, \quad U \in \{0,1, , \dots , C\}; \qquad (1.10)$$

$$\boldsymbol{B}(t, U) = \max_{u_{t+1} \in \{0,1,\dots,\min(C_{t+1}, U)\}} \{ f_{t+1}(u_{t+1}) + \boldsymbol{B}(t+1, U - u_{t+1})\}, \qquad (1.11)$$

здесь $t \in \{0,1, , \dots , n-1\}$, $U \in \{0,1, , \dots , C\}$.

При реализации обратного метода Беллмана процесс вычислений реализуется в соответствии с записанными соотношениями. Зная, что $\boldsymbol{B}(n, U) = 0$ при всех U, по формуле (1.11) находим все значения $\boldsymbol{B}(n-1, U)$ при $U \in \{0,1,\dots, S\}$; здесь оказывается справедливым тождество $\boldsymbol{B}(n-1, U) = f_n(\min(C_n, U))$. Далее, имея вычисленными значения $\boldsymbol{B}(n-1, U)$, по той же формуле (1.11) определяем все значения $\boldsymbol{B}(n-2,U)$, где $U \in \{0,1, \dots , S\}$, и т.д. до тех пор, пока не окажется найденной величина $\boldsymbol{B}(0, C)$ - оптимальное значение критерия в решаемой *ЗОРК*. В процессе вычислений, с целью обеспечения возможности определить оптимальное распределение капиталовложений, после отыскания каждого следующего значения $\boldsymbol{B}(t, U)$, где $t \in \{0,1, , \dots , n-1\}$ и $U \in \{0,1,\dots, C\}$, для пары (t, U) отдельно фиксируем значение u_{i+1}, на котором достигается максимум правой части (1.11).

Прямой метод Беллмана для решения *ЗОРК* заключается в последовательном вычислении значений функции $\boldsymbol{B}^*(t, U)$ в порядке возрастания значений первого аргумента. При этом

$$\boldsymbol{B}^*(0, U) = 0, \quad U \in \{0,1, , \dots , C\}; \qquad (1.12)$$

$$\boldsymbol{B}^*(t+1, U) = \max_{u_{t+1} \in \{0,1,\dots,\min(C_{t+1}, U)\}} \{\boldsymbol{B}^*(t, U - u_{t+1}) + f_{t+1}(u_{t+1})\}, \qquad (1.13)$$

здесь $t \in \{0,1,\dots, n-1\}$, $U \in \{0,1, , \dots , C\}$. Основанные на формуле (1.13) вычисления заканчиваются отысканием величин $\boldsymbol{B}^*(n, U)$, где $U \in \{1, 2,\dots, C\}$; максимальная из найденных величин – оптимальное значение критерия в решаемой *ЗОРК*.

ПРИМЕР 1.3. Имеются сферы капиталовложений S_1, S_2, S_3, S_4; в каждую из них может быть внесен целочисленный вклад, не превосходящий 5. Величина подлежащего распределению капитала равна 6. Функции, определяющие доходы, получаемые при вложении в различные сферы вклада u, заданы таблицей 1.1. Требуется найти оптимальное распределение капитала между сферами вложений.

Решение данного примера можно выполнить как прямым, так и обратным методом Беллмана. Будем основываться на рекуррентных соотношениях (1.10) - (1.11), т.е. применим обратный метод. Отыскиваемые в процессе счета значения функции $B(t, U)$ вносим в таблицу 1.2. В каждую заполняемую клетку (t, U) этой таблицы (параметр t определяет столбец, параметр U – строку) одновременно с $B(t, U)$ вносится и записанное в квадратных скобках значение u_{i+1}, на котором достигается максимум правой части (1.11) при подсчете $B(t, U)$; если таких значений несколько, вносится любое из них (нашей целью является отыскание какого-либо из оптимальных решений, а не полной их совокупности).

В связи с тем, что $B(4, U)$ тождественно равно нулю, процесс вычислений начинаем с определения значений $B(3, U)$ при всех возможных значениях U. Из равенства (1.11) следует, что $B(3, U) = f_4(U)$ при $U = 0,1,2,3,4,5$ и $B(3, 6) = f_4(5)$. Так заполняется

Таблица 1.1. Значения функций дохода $f_t(u)$.

	$F_1(u)$	$f_2(u)$	$f_3(u)$	$f_4(u)$
u = 0	0	0	0	0
u = 1	0,2	0	0,1	0
u = 2	0,2	0	0,2	0
u = 3	0,3	0,4	0,3	0,3
u = 4	0,3	0,4	0,4	0,4
u = 5	0,5	0,4	0,5	0,5

Таблица 1.2. Значения функции B(t, U).

	t =0	t =1	t =2	t =3
U=0		0 [0]	0 [0]	0 [0]
U=1		0,1 [0]	0,1 [1]	0 [0]
U=2		0,2 [0]	0,2 [2]	0 [0]
U=3		0,4 [3]	0,3 [3]	0,3 [3
U=4		0,5 [3]	0,4 [4]	0,4 [4
U=5		0,6 [3]	0,5 [5]	0,5 [5
U=6	0,8 [1]	0,7 [3]	0,6 [1]	0,5 [5

последний столбец таблицы 1.2. То же равенство (1.11) дает возможность последовательного (справа налево) заполнения остальных столбцов. Заметим, что для решения задачи в первом столбце достаточно найти только один, нижний элемент. В рассматриваемом примере оказалось, что $B(0,6) = 0,8$. Это оптимальное значение критерия (максимально возможный доход). В клетке (0,6) в квадратных скобках записано число 1. Значит, в оптимальном распределении капиталовложений взнос в первую сферу должен равняться единице. В таком случае суммарный взнос в остальные сферы не превосходит 5. В клетке (1,5) в квадратных скобках записано число 3. Значит, в синтезируемом оптимальном распределении капиталовложений взнос во вторую сферу равняется трем. Получаем, что суммарный взнос в третью и четвертую

сферы не превосходит 2. В клетке (2,2) в квадратных скобках записано число 2. В оптимальном распределении взнос в третью сферу должен равняться двум. В итоге получаем, что оптимальное распределение делит сумму $C = 6$ между первой, второй и третьей сферами капиталовложений; вносимые в них вклады равны соответственно 1, 3, 2. При этом первая сфера приносит доход 0,2; вторая сфера дает доход 0,4 и третья сфера - доход 0,2. Суммарный доход действительно равен 0,8.

Задача о выборе траектории набора высоты и скорости(ЗВТНВС). Летательный аппарат, в начальный момент имеющий скорость V_0 и высоту H_0, должен

быть поднят на высоту H_1, а скорость его должна достигнуть значения V_1; здесь $V_1>V_0$ и $H_1>H_0$. Считаются известными функции $F(V, H)$ и $G(V, H)$, определяющие соответственно дополнительный расход горючего при увеличении скорости полета с V до $V+1$ при неизменной высоте полета H и при увеличении высоты полета от H до $H+1$ при неизменной скорости полета V. Требуется найти режим набора высоты и скорости, при котором общий расход горючего будет минимальным.

Вводим дискретизацию объекта. Считаем, что числа V_0, H_0, V_1, H_1 - целые и что процесс набора высоты и скорости можно разбить на ряд последовательных этапов, результатом каждого из которых является увеличение на единицу одного из параметров полета, высоты или скорости. В таком случае функция $F(V, H)$ определена для значений $V\in\{V_0, V_0+1, V_0+2,\ldots, V_1-1\}$ и $H\in\{H_0, H_0+1, H_0+2,\ldots, H_1\}$; функция $G(V, H)$ определена для значений $V\in\{V_0, V_0+1, V_0+2,\ldots, V_1\}$ и $H\in\{H_0, H_0+1, H_0+2,\ldots, H_1-1\}$; можно считать эти функции заданными таблично. Исходные данные рассматриваемой задачи определяют систему $\boldsymbol{\Omega}$(ЗВТНВС), управление которой осуществляется в дискретном времени; управление определяет, значение какого параметра полета, высоты или скорости, следует увеличить на единицу. Состояния системы – пары целых чисел (V,H), где V – достигнутая скорость, а H – имеющаяся высота полета. Начальное состояние системы (V_0, H_0); финальным является состояние (V_1, H_1). Пусть $\boldsymbol{B}(V,H)$ - минимально возможный расход горючего, при котором значения скорости и высоты полета могут быть увеличены от V и H до V_1 и H_1 соответственно; здесь $V\in\{V_0, V_0+1, V_0+2,\ldots, V_1\}$ и $H\in\{H_0, H_0+1, H_0+2,\ldots, H_1\}$. Очевидно, что

$$\boldsymbol{B}(V_1, H_1)=0; \quad (1.14)$$

$$\boldsymbol{B}(V, H_1)= F(V, H_1) + \boldsymbol{B}(V+1, H_1),\ V\in\{V_0, V_0+1, V_0+2,\ldots, V_1-1\}; \quad (1.15)$$

$$\boldsymbol{B}(V_1, H)= G(V_1, H)+ \boldsymbol{B}(V_1, H+1),\ H\in\{H_0, H_0+1, H_0+2,\ldots, H_1-1\}. \quad (1.16)$$

В общем случае, для $V\in\{V_0, V_0+1, V_0+2,\ldots, V_1-1\}$ и $H\in\{H_0, H_0+1, H_0+2,\ldots, H_1-1\}$, имеем:

$$\boldsymbol{B}(V,H)= \min\{F(V, H) + \boldsymbol{B}(V+1,H), G(V, H) + \boldsymbol{B}(V,H+1)\}. \quad (1.17)$$

Равенства (1.14)-(1.17) являются рекуррентными соотношениями динамического программирования, позволяющими определить оптимальную траекторию набора высоты и скорости летательного аппарата. Если в ситуации, характеризуемой парой (V,H), в правой части последнего записанного соотношения минимум достигается на первой компоненте, то следует на единицу увеличить скорость объекта; если же минимум достигается на второй компоненте, то на единицу следует увеличить высоту полета.

Вычисления по записанным рекуррентным соотношениям можно представить как процесс заполнения таблицы 1.3, строки которой соответствуют имеющимся значениям высоты, а столбцы – имеющимся значениям скорости. В каждую клетку, стоящую в пересечении столбца H и строки V вносятся: 1) соответствующее значение функции $\boldsymbol{B}(V,H)$; 2) заключенный в квадратные скобки символ v или h; внесение символа v (символа h) означает, что в ситуации, определяемой парой (V,H), на единицу следует увеличить значение скорости (высоты) полета. Заполнение таблицы осуществляется в следующем порядке. Сначала на основании равенств (1.14)-(1.16) заполняются клетки нижней (последней) строки и крайне правого (последнего) столбца. Дальнейшее заполнение реализуется на основании формулы (1.17); заполняются клетки предпоследней строки и предпоследнего столбца, и т.д. Если в (1.17) минимум правой части реализуется на первой компоненте, то в заполняемую

клетку в качестве второй компоненты вносится [*v*], в противном случае в заполняемую клетку вносится [*h*].

Таблица 1.3. Значения функции $\boldsymbol{B}(V,H)$.

	V_0	$V_0 + 1$	$V_0 + 2$	……	V_1
H_0	$\boldsymbol{B}(V_0,H_0)$ [v или h]	$\boldsymbol{B}(V_0+1,H_0)$ [v или h]	$\boldsymbol{B}(V_0+2,H_0)$ [v или h]	…… ….	$\boldsymbol{B}(V_1,H_0)$ [h]
$H_0 + 1$	$\boldsymbol{B}(V_0,H_0+1)$ [v или h]	$\boldsymbol{B}(V_0+1,H_0+1)$ [v или h]	$\boldsymbol{B}(V_0+2,H_0+1)$ [v или h]	…… ….	$\boldsymbol{B}(V_1,H_0+1)$ [h]
$H_0 + 2$	$\boldsymbol{B}(V_0,H+2)$ [v или h]	$\boldsymbol{B}(V_0+1,H_0+2)$ [v или h]	$\boldsymbol{B}(V+2,H_0+2)$ [v или h]	…… ….	$\boldsymbol{B}(V_1,H_0+2)$ [h]
…… ….	…… ….	…… ….	…… ….	…… ….	…… ….
H_1	$\boldsymbol{B}(V_0,H_1)$ [v]	$\boldsymbol{B}(V_0+1,H_1)$ [v]	$\boldsymbol{B}(V_0+2,H_1)$ [v]	…… ….	$\boldsymbol{B}(V_1,H_1)$

Дадим оценку вычислительной сложности предложенной процедуры счета. Число вычисляемых значений функции Беллмана $\boldsymbol{B}(V,H)$ оценивается произведением $(V_1-V_0)\times(H_1-H_0)$. Каждое очередное значение этой функции вычисляется по записанным рекуррентным соотношениям посредством выполнения нескольких элементарных операций. Общая оценка числа выполняемых для решения задачи элементарных операций $C(V_1- V_0)(H_1- H_0)$, где C - достаточно малая натуральная константа (оценка 10 для C является верхней). В случае, например, $V_1- V_0 = H_1- H_0 =10$ число выполняемых при синтезе оптимальной траектории элементарных операций не превышает 1 000.

Примем следующие обозначения: $V_1- V_0= p$, $H_1- H_0=q$. Сумма $p+ q$ - общее число этапов, на каждом из которых на единицу возрастает значение V или H. Если *ЗВТНВС* решать методом полного перебора, то общий расход горючего придется подсчитать для C_{p+q}^{p} различных траекторий набора высоты и скорости полета (через C_n^m как обычно, обозначается число сочетаний из n по m). В случае $V_1- V_0 = H_1- H_0 =10$ расход горючего приходится подсчитать для C_{20}^{10} (т.е. более чем для 180 000) различных траекторий.

При решении методом динамического программирования дискретных оптимизационных задач построение соответствующих управляемых систем в явном виде, как правило, не выполняется. Вместо этого по имеющейся оптимизационной задаче формулируется и далее решается определенным образом упорядоченная совокупность частных задач. В применениях обратного метода Беллмана каждая частная задача фактически определяет экстремальную проблему перевода системы, соответствующей исходной задаче, из некоторого промежуточного (в частности – начального) состояния в финальное. В применениях прямого метода Беллмана каждая частная задача определяет экстремальную проблему перевода системы из начального

состояния в некоторое промежуточное (в частности – финальное) состояние. Экстремальной проблеме перевода системы из начального в финальное состояние соответствует исходная задача.

В соответствии со сказанным, метод динамического программирования относят к числу схем инвариантного (не меняющего существа проблемы) погружения. Суть схемы инвариантного погружения состоит в замене исходной задачи семейством однотипных задач, одной из которых является исходная. Последовательное, в порядке возрастания (или убывания) значений одного или нескольких параметров решение задач вводимого семейства приводит в итоге к определению оптимального решения исходной задачи.

Достаточно часто при реализации метода динамического программирования в явном виде не записываются и частные задачи, вводится только соответствующая функция Беллмана.

Классическая задача коммивояжера. Приведем несколько предварительных определений и фактов. Цикл в конечном ориентированном графе называется простым, если он не имеет самопересечений (нет вершины, которую цикл проходит дважды). Простой цикл называется гамильтоновым, если он проходит через все вершины графа. Граф G называется полным, если для любой упорядоченной пары i, j его вершин в G имеется дуга (i, j). Гамильтоновы циклы имеются не во всяком графе. Граф G именуется гамильтоновым, если гамильтонов цикл в нем есть. В полном n- вершинном ориентированном графе имеются $(n-1)!$ гамильтоновых циклов.

Если G – взвешенный ориентированный граф; то весом произвольного цикла этого графа называется сумма весов дуг, входящих в состав этого цикла.

Классическая задача коммивояжера (*ЗК*) формулируется следующим образом: имеется полный взвешенный ориентированный граф без петель G с множеством вершин $N = \{1, 2, \ldots, n\}$; веса всех дуг неотрицательны; в этом графе требуется найти гамильтонов цикл минимального веса.

Исходную информацию по *ЗК* считаем представленной в виде $(n \times n)$- матрицы $S=\{s_{ij}\}$, где s_{ij} – вес дуги (i,j) графа G, $i = \overline{1,n}$, $j = \overline{1,n}$, $i \neq j$; все элементы главной диагонали матрицы S являются нулями.

В типовой интерпретации вершины $1, 2, \ldots, n$ графа G – это города. Дуги отображают возможные элементарные переходы. Коммивояжеру, изначально находящемуся в городе 1, необходимо обойти все остальные города, побывав в каждом из них ровно по одному разу, и затем вернуться в город 1. Веса дуг графа трактуются как длины соответствующих элементарных переходов. Требуется найти имеющий минимальную длину допустимый (т.е. удовлетворяющий наложенным требованиям) маршрут коммивояжера. С учетом других возможных интерпретаций, на матрицу S требование симметричности не налагается, не считается обязательным и выполнение неравенства треугольника.

Один из основных алгоритмов решения *ЗК* основан на принципе динамического программирования. При изложении этого алгоритма будем придерживаться терминологии, соответствующей приведенной типовой интерпретации задачи.

Пусть i – произвольный город ($i \in N$), а V – любое подмножество городов, не содержащее города 1 и города i. Через $M(i, V)$ обозначим совокупность путей, каждый из которых начинается в городе i, завершается в городе 1 и проходит в качестве

промежуточных только через города множества V, заходя в каждый из них ровно по одному разу. Через $\boldsymbol{B}(i, V)$ обозначим длину кратчайшего пути множества $M(i, V)$. Для решаемой задачи $\boldsymbol{B}(i, V)$ – функция Беллмана. Как очевидно, $\boldsymbol{B}(1, \{2, 3,\ldots,n\})$ – искомая минимальная длина простого (без самопересечений) замкнутого пути, проходящего через все города.

Если V – одноэлементное множество, $V=\{j\}$, где $j \neq 1$ и $j \neq i$, то совокупность $M(i, V)$ состоит из единственного пути $\pi = (i, j, 1)$. Поэтому

$$\boldsymbol{B}(i, \{j\}) = s_{ij} + s_{j1},\ i \in N,\ j \in \{2, 3,\ldots, n\},\ j \neq i. \qquad (1.18)$$

Предположим, что значения функции $\boldsymbol{B}(i, V)$ для всех $i \in N \setminus \{1\}$ и всех возможных k-элементных ($k < n-1$) множеств V уже вычислены. Тогда значение $\boldsymbol{B}(i, V')$, где V' – произвольное $(k+1)$-элементное подмножество совокупности $N \setminus \{1, i\}$, вычисляется по формуле

$$\boldsymbol{B}(i, V') = \min_{j \in V'} (s_{ij} + \boldsymbol{B}(j, V' \setminus \{j\})). \qquad (1.19)$$

Уравнения (1.18)-(1.19) – рекуррентные соотношения динамического программирования для решения задачи коммивояжера, они реализуют обратный метод Беллмана.

ПРИМЕР 1.4. Решить задачу коммивояжера, определяемую матрицей

$$S = \begin{pmatrix} 0 & 5 & 1 & 4 \\ 2 & 0 & 3 & 5 \\ 1 & 4 & 0 & 2 \\ 3 & 5 & 4 & 0 \end{pmatrix}$$

Сначала, пользуясь формулой (1.18), определяем значения $\boldsymbol{B}(i, \{j\})$:

$B(2, \{3\}) = 4;\ B(2, \{4\}) = 8;\ B(3, \{2\}) = 6;\ B(3, \{4\}) = 5;\ B(4, \{2\}) = 7;\ B(4, \{3\}) = 5.$

Далее по формуле (1.19) последовательно получаем (в левой части каждого из ниже записанных равенств подчеркнуты те значения параметра j, на которых при подсчете реализуется указанный в правой части (1.19) минимум):

$B(2, \{\underline{3},4\}) = \min (3 + 5, 5 + 5) = 8;$

$B(3, \{2,\underline{4}\}) = \min (4 + 8, 2 + 7) = 9;$

$B(4, \{\underline{2},3\}) = \min (5 + 4, 4 + 6) = 9;$

$B(1, \{2,\underline{3}, 4\}) = \min (5 + 8, 1 + 9, 4 + 9) = 10.$

Итак, оптимальное значение критерия в рассматриваемом примере равно 10.

Выполненные подчеркивания позволяют определить оптимальный маршрут. Он следующий:

$$1 \to 3 \to 4 \to 2 \to 1.$$

Для записи соотношений, по которым реализуется прямой метод Беллмана, введем новые обозначения. Пусть M'(V, i) - совокупность путей, каждый из которых начинается в городе 1, проходит в качестве промежуточных только через города подмножества V, заходя в каждый из них ровно по одному разу, и завершается в городе

i; здесь, как и ранее, i - произвольный город (i∈ N), а V - любое подмножество N, не содержащее городов 1 и i. Длину кратчайшего пути множества M'(V, i) обозначим **B***(V, i). Как очевидно, **B***({2, 3,…,n},1) - искомая минимальная длина простого (без самопересечений) замкнутого пути, проходящего через все города. Если V - одноэлементное множество, V ={j}, где j ≠ 1 и j ≠ i, то совокупность M'(V, i) состоит из единственного пути π =(1, j, i). Поэтому

$$\boldsymbol{B^*}(\{j\},i) = s_{1j} + s_{ji},\ i \in N,\ j \in \{2, 3, \ldots, n\},\ j \neq i. \qquad (1.20)$$

Предположим, что значения функции $\boldsymbol{B^*}(V, i)$ для всех $i \in N$ и всех возможных k- элементных ($k < n-1$) множеств V уже вычислены. Тогда значение $\boldsymbol{B^*}(V',i)$, где V' – произвольное ($k+1$)- элементное подмножество совокупности $N\backslash\{1, i\}$, вычисляется по формуле

$$\mathbf{B^*}(V',i) = \min_{j \in V'} (\mathbf{B^*}(V'\backslash \{j\},j) + s_{ji}) . \qquad (1.21)$$

Уравнения (1.20)-(1.21) – рекуррентные соотношения динамического программирования для решения классической задачи коммивояжера, они реализуют прямой метод Беллмана.

Заметим, что при решении задачи коммивояжера обратным методом состояния моделирующей системы – пары вида *(i, V)*, где *i* – город, в котором сейчас находится коммивояжер, а *V* – множество городов, которые ему еще предстоит обойти перед тем, как вернуться в исходный город 1. При решении задачи коммивояжера прямым методом состояния моделирующей системы – пары вида *(V, i),* где *i* – город, в котором коммивояжер находится сейчас, а *V* – множество городов, которые коммивояжер уже обошел после того, как вышел из города 1. По сути, *(i, V)* и *({2,3,…,n} \ (V∪{i}),i)* – это разные обозначения одной и той же ситуации. Данный факт используется при решении рассматриваемой задачи методом встречного счета.

ПРИМЕР 1.5. Методом встречного счета решить задачу коммивояжера, определяемую матрицей

$$S = \begin{matrix} 0 & 5 & 1 & 4 \\ 2 & 0 & 3 & 5 \\ 1 & 4 & 0 & 2 \\ 3 & 5 & 4 & 0 \end{matrix}$$

(заметим, что матрица *S* в данном примере та же, что и в предыдущем).

В качестве разрезающего выберем множество, включающее следующие состояния:

(2, {3}) /совпадет с ({4},2)/;
(2, {4}) /совпадет с ({3},2)/;
(3, {2}) /совпадет с ({4},3)/;
(3, {4}) /совпадет с ({2},3)/;
(4, {2}) /совпадет с ({3},4)/;
(4, {3}) /совпадет с ({2},4)/.;

Реализуя метод обратного счета, определяем:

B(2, {3}) = 4; B(2, {4}) = 8; B(3, {2}) = 6; B(3, {4}) = 5; B(4, {2}) = 7; B(4, {3}) = 5.

Реализуя метод прямого счета, получаем:

B({4}, 2) = 9; B*({3},2) = 5; B*({4},3) = 8; B*({2},3) = 8; B*({3},4) = 3; B*({2},4) = 10.*

Далее имеем:

B(2, {3}) + B({4}, 2) = 13;*

B(2, {4}) + B({3},2) = 13;*

B(3, {2}) + B({4},3) = 14;*

B(3, {4}) + B({2},3) = 13;*

B(4, {2}) + B({3},4) = 10;*

B(4, {3}) + B({2},4) = 15.*

Минимальное, равное 10, значение правая часть принимает в пятом равенстве. Отсюда следует: а) оптимальное значение критерия в рассматриваемой задаче равно 10; б) оптимальная траектория системы проходит через состояние *({3},4)* или, что то же самое, через состояние*(4, {2})*. Последнее дает возможность установить оптимальный маршрут коммивояжера:

$$1 \to 3 \to 4 \to 2 \to 1.$$

Применение метода встречного счета в сравнении с прямым или обратным счетом тем более выгодно, чем больше размер решаемой задачи.

Задача коммивояжера с минимаксным критерием (ЗК ММК). Считается заданным полный взвешенный ориентированный граф без петель G с множеством вершин $N = \{1, 2, \ldots, n\}$; веса всех дуг неотрицательны. Для каждого цикла C этого графа определена характеристика $\omega(C)$, значение которой равно максимальному из весов дуг, данный цикл образующих. В рассматриваемой задаче требуется найти гамильтонов цикл C графа G, имеющий минимально возможное значение характеристики $\omega(C)$. Исходную информацию по задаче считаем представленной в виде (nxn)- матрицы $S=\{s_{ij}\}$, где s_{ij} – вес дуги (i, j) графа G, $i = \overline{1,n}$, $j = \overline{1,n}$, $i \neq j$; все элементы главной диагонали матрицы S нулевые.

Придерживаемся прежней интерпретации: вершины графа G – это города, которые коммивояжеру необходимо обойти. Пусть i – произвольный город $(i \in N)$, а V – любое подмножество городов, не содержащее города 1 и города i. Как и ранее, через $M(i, V)$ обозначаем совокупность путей, каждый из которых начинается в городе i, завершается в городе 1 и проходит в качестве промежуточных только через города множества V, заходя в каждый из них ровно по одному разу. Каждый путь π характеризуем показателем $\omega(\pi)$ – максимальным из весов дуг, которые образуют данный путь. Минимальное значение показателя $\omega(\pi)$ для путей множества $M(i, V)$ обозначим $\boldsymbol{B'}(i, V)$. Как очевидно, $\boldsymbol{B'}(1, \{2, 3, \ldots, n\})$ – искомое оптимальное значение критерия в решаемой *ЗК ММК* .

Если V – одноэлементное множество, $V = \{j\}$, где $j \neq 1$ и $j \neq i$, то совокупность $M(i, V)$ состоит из единственного пути $\pi = (i, j, 1)$. Поэтому

$$\boldsymbol{B'}(i, \{j\}) = \max(s_{ij}, s_{j1}),\ i \in N,\ j \in \{2, 3, \ldots, n\},\ j \neq i; \qquad (1.22)$$

Предположим, что значения функции $\boldsymbol{B'}(i, V)$ для всех $i \in N$ и всех возможных k-элементных ($k < n-1$) множеств V уже вычислены. Тогда значение $\boldsymbol{B'}(i, V')$, где V' – произвольное $(k+1)$-элементное подмножество совокупности $N \setminus \{1, i\}$, вычисляется по формуле

$$\boldsymbol{B'}(i, V') = \min_{j \in V'} \{ \max (s_{ij}, \boldsymbol{B'}(j, V' \setminus \{j\}) \} . \qquad (1.23)$$

Уравнения (1.22) - (1.23) – рекуррентные соотношения динамического программирования для решения *ЗК ММК*.

Отметим, что изложенным алгоритмом может решаться вопрос определения по произвольному n-вершинному ориентированному графу G, является ли он гамильтоновым. С этой целью по графу G строим полный взвешенный ориентированный граф без петель G^* с множеством вершин $N = \{1, 2, \ldots, n\}$. Веса дуг графа G^* назначаем по следующему правилу: вес s_{ij} дуги (i, j) равен 1, если дуга (i, j) в исходном графе G имеется; вес s_{ij} дуги (i, j) равен 2 в противном случае. Далее решаем определяемую графом G^* задачу коммивояжера с минимаксным критерием. Если оптимальное значение этого критерия оказывается равным 1, то граф G гамильтонов; в противном случае оптимальное значение критерия равно 2, и граф G гамильтоновым не является.

Задача инкассации. Задача определяется полным ориентированным графом без петель G с множеством вершин $N = \{1, 2, \ldots, n\}$.Каждая дуга и каждая вершина этого графа, кроме вершины *1*, имеет определенный вес. В интерпретации вершины $2,3, \ldots, n$ – это объекты, которые инкассатору необходимо обойти по замкнутому маршруту, посетив каждый из них ровно по одному разу. При этом считаем, что маршрут начинается и заканчивается в вершине *1* (банке). Веса дуг графа G трактуются как длины реализуемых по ним переходов, веса вершин $2, 3, \ldots, n$ – это денежные суммы, которые соответствующими объектами передаются инкассатору для транспортировки в банк. Из банка инкассатор выходит без денег, в банк он возвращается со всей собранной по объектам множества $\{2, 3, \ldots, n\}$ денежной суммой.

С целью упрощения обозначений будем считать, что вершина *1* также имеет вес, он равен нулю. Исходную информацию по задаче считаем представленной $(n \times n)$-матрицей $S=\{s(i, j)\}$ и n-мерным вектором $\boldsymbol{d} = (d(1), d(2), \ldots, d(n))$, где $s(i, j)$ – вес дуги (i, j), а $d(i)$ – вес вершины i графа G, $i = \overline{1, n}$, $j = \overline{1, n}$, $i \neq j$. Все элементы главной диагонали матрицы S считаются нулевыми, первая координата вектора $\boldsymbol{d}$ равна нулю. Каждый гамильтонов цикл $C = (1, i_2, i_3, \ldots, i_n, 1)$, здесь $(i_2, i_3, \ldots, i_n)$ – некоторая перестановка чисел $2, 3, \ldots, n$, оцениваем показателем

$$\chi(C) = d(i_2) \times s(i_2, i_3) + (d(i_2) + d(i_3)) \times s(i_3, i_4) + (d(i_2) + d(i_3) + d(i_4)) \times s(i_4, i_5) + \ldots$$
$$+ (d(i_2) + d(i_3) + d(i_4) + \ldots + d(i_n)) \times s(i_n, 1).$$

Задача инкассации состоит в отыскании гамильтонова цикла с наименьшим значением показателя $\chi(C)$.

В интерпретации $\chi(C)$ – общее количество "деньго-километров" при реализации гамильтонова цикла C. Введенный показатель $\chi(C)$ можно считать характеристикой безопасности маршрута. Чем меньше $\chi(C)$, тем безопаснее маршрут.

Пусть i – произвольная вершина графа ($i \in N$), а V – любое подмножество его вершин, не содержащее вершины *1* и вершины i. Как и ранее, через $M(i, V)$ обозначаем совокупность путей, каждый из которых начинается в вершине i, завершается в

вершине *1* и проходит в качестве промежуточных только через вершины множества V, заходя в каждую из них ровно по одному разу. Каждый путь π характеризуем показателем $\eta(\pi)$, общим числом деньго-километров в его реализации; при этом считаем, что в начальной вершине i пути π из множества $M(i, V)$ инкассатор имеет денежную сумму

$$R(N \setminus V) = \sum_{\alpha \in N \setminus V} d(\alpha);$$

именно эта сумма собрана инкассатором в не принадлежащих множеству V пунктах. Минимальное значение показателя $\eta(\pi)$ для путей множества $M(i, V)$ обозначим $\boldsymbol{B}(i, V)$. Как очевидно, $\boldsymbol{B}(1, \{2, 3, \ldots, n\})$ – искомое оптимальное значение критерия в решаемой задаче инкассации.

Если V – одноэлементное множество, $V = \{j\}$, где $j \neq 1$ и $j \neq i$, то совокупность $M(i, V)$ состоит из единственного пути $\pi = (i, j, 1)$. Поэтому

$$\boldsymbol{B}(i, \{j\}) = R(N \setminus \{j\})\, s(i, j) + R(N)\, s(j, 1),\ i \in N,\ j \in \{2, 3, \ldots, n\},\ j \neq i. \quad (1.24)$$

Предположим, что значения функции $\boldsymbol{B}(i, V)$ для всех $i \in N$ и всех возможных k-элементных ($k < n-1$) подмножеств V уже вычислены. Тогда значение $\boldsymbol{B}(i, V')$, где V' – произвольное $(k+1)$-элементное подмножество совокупности $N \setminus \{1, i\}$, вычисляется по формуле

$$\boldsymbol{B}(i, V') = \min_{j \in V'} \{ R(N \setminus V')\, s(i, j) + \boldsymbol{B}(j, V' \setminus \{j\}) \}. \quad (1.25)$$

Уравнения (1.24) - (1.25) – рекуррентные соотношения динамического программирования для решения задачи инкассации. Отметим, что в рассмотренной задаче на оценку гамильтонова цикла $C = (1, i_2, i_3, \ldots, i_n, 1)$ никак не влияет значение $s(1, i_2)$, т.е. длина первого реализуемого перехода; это вызвано тем, что в данном переходе транспортируется нулевая денежная сумма. В главе 4 будет рассмотрена бикритериальная задача инкассации, где один критерий – суммарная длина (стоимость) маршрута, а другой – количество деньго-километров.

Задача двух коммивояжеров. Математическая постановка задачи следующая. Имеется полный взвешенный ориентированный граф без петель G с множеством вершин $N = \{1, 2, \ldots, n\}$; веса всех дуг неотрицательны. Требуется найти пару циклов $\langle C_1, C_2 \rangle$ этого графа, имеющую минимальный суммарный вес и удовлетворяющую условиям:

а) вершина 1 – единственная вершина, через которую проходит как цикл C_1, так и цикл C_2;

б) через каждую вершину множества $N \setminus \{1\}$ проходит либо цикл C_1, либо цикл C_2.

Исходную информация по задаче представлена (nxn)-матрицей $S = \{s_{ij}\}$, где s_{ij} – вес дуги (i, j) графа G, $i = \overline{1, n}$, $j = \overline{1, n}$, $i \neq j$; все элементы главной диагонали матрицы S нулевые.

Придерживаемся интерпретации, соответствующей изложенной выше. Вершины графа G – это города, которые коммивояжерам необходимо обойти по замкнутым маршрутам; веса дуг графа G – длины соответствующих переходов. Пусть $i \in N$, $j \in N$, $i \neq j$ за исключением случая $i = j = 1$, а V – любое подмножество из N, не

содержащее городов *1, i* и *j*. Через *M''(V, i, j)* обозначим совокупность всех пар (π_1, π_2) путей, обладающих следующими свойствами:

а) каждый из путей начинается в городе *1*;

б) путь π_1 заканчивается в городе *i*;

в) путь π_2 заканчивается в городе *j*;

г) в паре (π_1, π_2) пути простые (без самопересечений), проходящие в качестве промежуточных только через города подмножества *V*, при этом через каждый город этого подмножества проходит ровно один путь из данной пары.

Кратчайшую суммарную длину, подсчитанную для пар путей из *M''(V, i, j)* обозначим ***B*(V, i, j)***. Ясно, что ***B*({2, 3,…,n},1,1)*** – искомая минимальная суммарная длина пары циклов $<C_1, C_2>$, удовлетворяющих требованиям решаемой задачи о двух коммивояжерах.

Как очевидно,

$$\boldsymbol{B^*}(\varnothing, i, j) = s_{1i} + s_{1j}. \qquad (1.26)$$

Множество *M''({k}, i, j)* состоит из двух пар путей: $\{\pi_1 = (1, k, i), \pi_2 = (1, j)\}$ и $\{\pi_1 = (1, i), \pi_2 = (1, k, j))\}$. Суммарная длина путей первой пары: $s_{1k} + s_{ki} + s_{1j} = (s_{1k} + s_{1j}) + s_{ki} = \boldsymbol{B^*}(\varnothing, k, j) + s_{ki}$. Аналогичным образом получаем, что суммарная длина путей второй пары равна $\boldsymbol{B^*}(\varnothing, i, k) + s_{kj}$. Поэтому

$$\boldsymbol{B^*}(\{k\}, i, j) = \min \{\boldsymbol{B^*}(\varnothing, k, j) + s_{ki}; \boldsymbol{B^*}(\varnothing, i, k) + s_{kj}\}. \quad (1.27)$$

В общем случае, когда $i \in N, j \in N, i \neq j$ за исключением варианта $i = j = 1$, а *V* - любое подмножество *N*, не содержащее городов *1, i* и *j*, имеем следующее. В путях пары (π_1, π_2) из *M''(V, i, j)* последними являются города *i* и *j* соответственно. Если некоторый принадлежащий подмножеству *V* город *k* является предпоследним в первом пути, то при данном дополнительном условии минимальная суммарная длина пары путей (π_1, π_2) равна $\boldsymbol{B^*}(V \setminus \{k\}, k, j) + s_{ki}$; если же принадлежащий *V* город *k* является предпоследним во втором пути, то при данном дополнительном условии минимальная суммарная длина пары путей (π_1, π_2) равна $\boldsymbol{B^*}(V \setminus \{k\}, i, k) + s_{kj}$. Отсюда получаем:

$$\boldsymbol{B^*}(V, i, j) = \boldsymbol{\min}\{\min_{k \in V'} (\boldsymbol{B^*}(V \setminus \{k\}, k, j) + s_{ki}), \min_{k \in V'} (s_{ik} + \boldsymbol{B^*}(V \setminus \{k\}, i, k) + s_{kj})\}. \quad (1.28)$$

Равенства (1.26)-(1.28) являются рекуррентными соотношениями динамического программирования для решения задачи двух коммивояжеров.

<u>*Классическая задача о ранце (КЗР)*</u>. Имеются ранец и предметы $П_1, П_2, \ldots, П_n$; каждый предмет $П_i$ характеризуется двумя целыми положительными показателями: c_i – стоимость предмета; v_i – вес предмета, $i = \overline{1, n}$. Предметы недробимы, т.е. каждый из них можно поместить в ранец только целиком. Требуется найти совокупность предметов, которые следует поместить в ранец с тем, чтобы суммарная стоимость предметов в ранце оказалась максимальной при условии, что суммарный вес предметов в ранце не может превысить заданной натуральной константы *W* (предполагается, что суммарный вес всех имеющихся предметов больше *W*).

КЗР формулируется как следующая задача булева линейного программирования (БЛП):

$$\sum_{i=1}^{n} c_i x_i \to max \qquad (1.29)$$

при условиях

$$\sum_{i=1}^{n} v_i x_i \le W; \qquad (1.30)$$

$$x_i \in \{0,1\},\ i=\overline{1,n}. \qquad (1.31)$$

Задачу (1.29)- (1.31) далее будем именовать задачей ***Z***.

По начальным данным задачи ***Z*** формулируем совокупность частных задач о ранце ***Z***(*k, p),* где $k \in \{1, 2, \ldots, n\}$; $p \in \{1, 2, \ldots, W\}$. Задача ***Z****(k, p)* записывается следующим образом:

$$\sum_{i=1}^{k} c_i x_i \to max$$

при условиях

$$\sum_{i=1}^{k} v_i x_i \le p;$$

$$x_i \in \{0,1\},\ i=\overline{1,k}.$$

(в частной задаче ***Z****(k, p)* имеющимися в наличии считаются только первые *k* предметов из совокупности $\{П_1, П_2, \ldots, П_n\}$, суммарный вес предметов в ранце не может превышать константы *p*).

Оптимальное значение критерия в частной задаче ***Z****(k, p)* обозначим ***B****(k, p)*. Как очевидно, задача ***Z****(n,W)* совпадает с исходной задачей ***Z***; поэтому ***B****(n,W)* – оптимальное значение критерия в задаче ***Z*** .

Легко видеть, что

$$B^*(1, p) = \begin{cases} 0 \text{ при } p \in \{0,1,2,\ldots, v_1 - 1\}; \\ c_1 \text{ при } p \in \{v_1, v_1+1, \ldots, W\}. \end{cases} \qquad (1.32)$$

Пусть для некоторого *k*, принадлежащего множеству *{1, 2,..., n-1}* и всех $p \in \{1, 2, \ldots, W\}$ значения функции ***B****(k, p)* уже вычислены. При $p \in \{0,1,2,\ldots, v_{k+1} - 1\}$ ситуация, возникающая при нахождении ***B****(k+1, p),* фактически не отличается от ситуации, имевшей место при подсчете ***B****(k, p),* ибо дополнительно появляющийся при переходе от задачи ***Z****(k, p)* к задаче ***Z****(k+1, p)* предмет $П_{k+1}$ в ранец поместить невозможно. В случае, когда $p \in \{v_{k+1}, v_{k+1}+1, \ldots, W\}$, при переходе от задачи ***Z****(k, p)* к задаче ***Z****(k+1, p)* мы получаем дополнительную возможность поместить в ранец предмет $П_{k+1}$. Этой возможностью можно: а) не воспользоваться; б) воспользоваться. Очевидно, что в случае а) мы получим уже вычисленную величину ***B****(k, p)*. В случае б) в ранец кладется предмет $П_{k+1}$ стоимостью c_{k+1} и так как вес этого предмета равен v_{k+1}, то далее мы оказываемся в условиях задачи ***Z****(k, p - v_{k+1})*. В итоге получаем:

$$\boldsymbol{B^*}(k, p), \text{ если } p \in \{0,1,2,\ldots, v_{k+1}- 1\};$$

$$\boldsymbol{B^*}(k+1,p)= \qquad (1.33)$$

$$max\ \{\ \boldsymbol{B^*}(k, p),\ \ c_{k+1} + \boldsymbol{B^*}(k, p - v_{k+1})\ \}\ при\ p \in \{v_{k+1}, v_{k+1}+1,\ldots, W\};$$

здесь $k= 2, 3,\ldots, n-1$.

Равенства (1.32) - (1.33) – рекуррентные соотношения динамического программирования для решения задачи о ранце. Пользуясь ими, мы находим сначала величины $\boldsymbol{B^*}(1, p),\ p \in \{1, 2,\ldots, W\}$, затем величины $\boldsymbol{B^*}(2, p),\ p \in \{1, 2,\ldots, W\}$, и т.д., вплоть до отыскания $\boldsymbol{B^*}(n, W)$ – оптимального значения критерия в исходной задаче $\boldsymbol{Z}$.

Процедуру вычисления значений функции $\boldsymbol{B^*}(k, p)$ целесообразно реализовывать как процесс последовательного заполнения строк таблицы стоимостей. В этой таблице строки соответствуют значениям параметра k (по возрастанию), а столбцы – значениям параметра p (также по возрастанию). В каждую клетку, расположенную в пересечении произвольной строки k и произвольного столбца p, вносится значение $\boldsymbol{B^*}(k, p)$. Первая строка заполняется согласно формуле (1.32). Далее заполнение каждой $(k+1)$-ой строки, $k= 1,2,\ldots, n-1$, осуществляется в соответствии с формулой (1.33). Одновременно целесообразно фиксировать множества предметов $M(k, p)$, которые обеспечивают значения $\boldsymbol{B^*}(k, p)$ критериев рассматриваемых частных задач. Как очевидно, $M(1, p) =\varnothing$, если $p < v_1$; $M(1, p) =\{1\}$ в противном случае. Далее можно считать

$$M(k, p), \text{ если } \boldsymbol{B^*}(k+1, p) = \boldsymbol{B^*}(k, p);$$

$$M(k+1, p) =$$

$$M(k+1, p) = M(k, p - v_{k+1}) \cup \{ k+1\} \text{ в противном случае.}$$

В заполненной таблице стоимостей содержимое клетки (n, W) – оптимальное значение критерия в задаче $\boldsymbol{Z}$. Для обеспечения этого значения в ранец следует поместить предметы совокупности $M(n, W)$.

Изложенный алгоритм решения *КЗР* далее будем называть алгоритмом $A_{кзр}$.

ПРИМЕР 1.6. Решить *КЗР*:

$$5x_1 + 2x_2 + 4x_3 + 7x_4 + 5x_5 \to max$$

при условиях

$$2x_1 + x_2 + 2x_3 + 4x_4+3x_5 \le 8;$$

$$x_i \in \{0,1\},\ i=\overline{1,5}\,.$$

Таблица 1.4. Таблица стоимостей для примера 1.6.

	1	2	3	4	5	6	7	8
1	0	5 (1)	5 (1)	5 (1)	5 (1)	5 (1)	5 (1)	5 (1)
2	2 (2)	5 (1)	7 (1,2)	7 (1,2)	7 (1,2)	7 (1,2)	7 (1,2)	7 (1,2)
3	2 (2)	5 (1)	7 (1,2)	9 (1,3)	11 (1,2,3)	11 (1,2,3)	11 (1,2,3)	11 (1,2,3)
4	2 (2)	5 (1)	7 (1,2)	9 (1,3)	11 (1,2,3)	12 (1,4)	14(1,2,4)	16(1,3,4)

5	2 (2)	5 (1)	7 (1,2)	9 (1,3)	11 (1,2,3)	12 (1,4)	14(1,2,4)	16(1,2,3,5)

Задача решается путем последовательного, сверху вниз, заполнения строк таблицы стоимостей (таблица 1.4). В каждой клетке *(k,p)* данной таблицы вслед за полученным значением ***B*(k, p)*** в скобках малыми цифрами перечисляются номера предметов, которые согласно оптимальному решению частной задачи ***Z(k, p)*** следует положить в ранец. Если задача ***Z(k, p)*** имеет несколько оптимальных решений, то указывается только одно из них. Заполнение первой строки таблицы выполняется в соответствии с формулой (1.32); здесь считается, что в нашем распоряжении имеется только предмет $П_1$, его стоимость равна *5* и вес равен *2*. Предмет $П_1$ можно положить в ранец, если дозволенный вес предметов в ранце не меньше чем *2*. Поэтому ***B*(1, 1)= 0*** *и* ***B*(1, p) = 5, если*** $p \geq 2$. Заполнение второй и каждой из нижеследующих строк выполняется в соответствии с формулой (1.33). Рассмотрим в качестве иллюстрации, причем на содержательном уровне, как заполняется клетка (*4, 8*). Данная клетка соответствует задаче ***Z(4, 8)***. В сравнении с задачей ***Z(3, 8)***, которой соответствует клетка (*3, 8*), в ***Z(4, 8)*** появляется новая возможность – в ранец можно положить предмет $П_4$. Согласно формуле (1.33), выбирается лучший из двух вариантов. Первый вариант предполагает, что возможностью положить в ранец предмет $П_4$ мы не воспользовались. Тогда имеет место абсолютно та же ситуация, что в задаче ***Z(3, 8)***, и мы можем обеспечить суммарную стоимость предметов в ранце равную ***B*(3, 8)***, т.е. числу *11*, записанному в клетке, расположенной непосредственно над заполняемой. Итак, первый вариант обеспечивает суммарную стоимость предметов в ранце, равную *11*; реализация данного варианта означает, что в ранец кладутся предметы $П_1$, $П_2$ и $П_3$. Второй вариант предусматривает, что предмет $П_4$, его стоимость *7*, кладется в ранец. Вес предмета $П_4$ равен *4*. Поэтому от заполняемой клетки в четвертой строке перемещаемся на *4* клетки влево, оказываемся в столбце, соответствующем оставшемуся резерву веса ранца после того, как в него положен предмет $П_4$; непосредственно над полученной клеткой, а именно в клетке (*3, 4*) записано число *9*, это максимально возможная суммарная стоимость предметов, взятых из совокупности $\{П_1, П_2, П_3\}$, при условии, что их суммарный вес не превышает четырех (в этой ситуации как показывает заполнение клетки (*3, 4*), из указанной совокупности следует взять первый и третий предмет). Таким образом, второй вариант обеспечивает суммарную стоимость предметов в ранце *7 + 9=16*; реализация данного варианта означает, что в ранец кладутся предметы $П_1$, $П_3$ и $П_4$. Второй вариант дает лучший результат, заполнение клетки (*4, 8*) осуществляется по этому варианту.

Некоторые действия, выполненные нами при решении примера 1.6, являются излишними. Для решения *КЗР* все клетки таблицы стоимостей заполнять не обязательно. В последней строке нам достаточно заполнить только одну, крайне правую клетку *(n, W)*. Согласно формуле (1.33), для этого достаточно знать содержимое только двух клеток предпоследней строки, а именно клетки *(n-1, W)* и клетки *(n-1, W-*v_n*)*. Для заполнения указанных клеток предпоследней строки достаточно знать заполнение максимум четырех клеток строки *n-2,* и т.д. Процессу счета по соотношениям динамического программирования (1.32)-(1.33) целесообразно предварить процесс разметки, когда двигаясь по строкам таблицы снизу вверх, мы отмечаем, заполнение каких клеток таблицы действительно необходимо для решения заданной *КЗР*.

К рассмотренной *КЗР* легко сводится и, следовательно, может решаться алгоритмом $A_{кзр}$ задача "*Разбиение*", формулируемая следующим образом. Имеется

конечное множество натуральных чисел $\boldsymbol{W}=\{v_1,v_2,\ldots,v_n\}$. Требуется определить, можно ли разбить совокупность $\boldsymbol{W}$ на два непересекающихся подмножества так, чтобы сумма элементов, входящих в одно подмножество, совпала с суммой элементов, входящих в другое подмножество.

Задачу "Разбиение" определяет набор натуральных чисел $\{v_1,v_2,\ldots,v_n\}$. Будем считать, что сумма всех входящих в множество $\boldsymbol{W}$ чисел четна (в противном случае рассматриваемая задача положительного решения заведомо не имеет). Пусть $v_1+v_2+\ldots+v_n = 2V$, где V – натуральное число. По имеющемуся набору $\{v_1,v_2,\ldots,v_n\}$ строим следующую *КЗР*:

$$v_1x_1+v_2x_2+\ldots+v_nx_n \to \max$$

$$v_1x_1+v_2x_2+\ldots+v_nx_n \le V;$$

$$x_i \in \{0,1\},\ i=\overline{1,n}.$$

Оптимальное значение критерия в этой *КЗР* не может превысить V, ибо критерий задачи совпадает с левой частью ее линейного ограничения. Оптимальное значение критерия равно V тогда и только тогда, когда исходная задача "Разбиение" имеет положительное решение (т.е. совокупность $\boldsymbol{W}$ можно разбить на два непересекающихся подмножества так, чтобы сумма элементов, входящих в одно подмножество, совпала с суммой элементов, входящих в другое подмножество.). В отвечающем оптимальному решению *КЗР* разбиении элемент w_i включается в первое подмножество, если переменная x_i принимает значение *1*, и во второе подмножество, если x_i принимает значение *0*, $i=\overline{1,n}$.

Часто задачу "Разбиение" называют *задачей о камнях*. При этом предполагается следующая интерпретация. Имеется куча, состоящая из n камней. Камни недробимы, вес каждого камня – заданное натуральное число. Вес любой кучи – это сумма весов составляющих ее камней. Требуется определить, можно ли имеющуюся кучу камней разделить на две подкучи равного веса..

Отметим, что алгоритм $A_{кзр}$ легко обобщается для решения следующей задачи:

$$\sum_{i=1}^{n} c_i x_i \to \max \qquad (1.34)$$

при условиях

$$\sum_{i=1}^{n} v^j_i x_i \le W^j,\ j=\overline{1,m}; \qquad (1.35)$$

$$x_i \in \{0,1\}. \qquad (1.36)$$

Задачу (1.34) - (1.36) будем называть многомерной задачей о ранце и обозначать символом $\boldsymbol{Z}^m$.

По начальным данным задачи $\boldsymbol{Z}^m$ формулируем совокупность частных задач $\boldsymbol{Z}(k, \boldsymbol{p})$, где $k \in\{1, 2, \ldots, n\}$; $\boldsymbol{p} = (p_1, p_2, \ldots, p_m)$, причем $p_j \in\{0, 1, 2, \ldots, W^j\}$, $j=\overline{1,m}$. В частной задаче $\boldsymbol{Z}(k, \boldsymbol{p})$ считаются имеющимися в наличии только первые k предметов из совокупности $\{П_1, П_2, \ldots, П_n\}$, величина суммарного веса предметов в ранце по j-ому

показателю, $j=\overline{1,m}$, не может превышать константы p_j. Оптимальное значение критерия в частной задаче $\boldsymbol{Z(k, p)}$ обозначим $\boldsymbol{B^*(k, p)}$. Задача $\boldsymbol{Z^m}$ совпадает с задачей $\boldsymbol{Z(n,W)}$, где $\boldsymbol{W} = (W^1, W^2,..., W^m)$. Поэтому $\boldsymbol{B^*(n, W)}$ – оптимальное значение критерия в задаче $\boldsymbol{Z^m}$. Процедура решения задачи $\boldsymbol{Z^m}$ методом динамического программирования представима как процесс заполнения таблицы стоимостей, строки которой соответствуют возможным значениям параметра k (по возрастанию), а столбцы – возможным значениям векторного параметра $\boldsymbol{p} = (p_1, p_{2,...,} p_m)$. Для определенности считаем, что в заголовочной для столбцов строке этой таблицы векторы перечисляются в соответствии с их лексикографическим порядком по возрастанию. В клетку, расположенную в пересечении строки k и столбца $\boldsymbol{p}$, вносится значение $\boldsymbol{B^*(k, p)}$. Заполнение клеток производится по строкам, в порядке возрастания их номеров. Рекуррентные соотношения динамического программирования для решения задачи $\boldsymbol{Z^m}$ очевидным образом обобщают равенства (1.32)-(1.33) на случай многомерной задачи.

За изложенным обобщением алгоритма $A_{кзр}$ сохраняем прежнее название. Оценим вычислительную сложность алгоритма в применении к m-мерной задаче. Пусть максимальная из координат вектора $\boldsymbol{W}$ равна W. В таком случае число подлежащих заполнению столбцов таблицы стоимостей имеет верхнюю оценку $(W+1)^m$. Число строк таблицы равно n. Верхняя оценка числа подлежащих заполнению клеток $n\times(W+1)^m$. Число элементарных операций, необходимых для определения заполнения каждой отдельной клетки, является функцией, линейно зависящей от m. Получаем, что $Cmn(W+1)^m$ – верхняя оценка числа элементарных операций, выполняемых изложенным алгоритмом при решении m-мерной задачи о ранце; здесь C – константа, не зависящая от конкретных характеристик задачи. В частном случае, для одномерной задачи о ранце ($m=1$) получаем оценку CnW; можно считать, что константа C не превосходит 10.

Задача отыскания кратчайших путей (ЗОКП). Считается заданным конечный взвешенный ориентированный граф G с множеством вершин $N = \{1, 2,..., n\}$, веса всех дуг неотрицательны. Веса дуг трактуем как их длины. Последовательность $i_1,i_2,..., i_k$ вершин графа G определяет путь из вершины i_1 в вершину i_k, если для каждого $t = 1, 2,..., k-1$ в данном графе имеется дуга (i_t, i_{t+1}); указанные дуги образуют путь $i_1,i_2,..., i_k$. Сумма длин дуг, образующих путь, называется длиной этого пути. Для каждой вершины x графа требуется найти путь минимальной длины (кратчайший путь) из вершины 1.в вершину x.

Вес дуги (i,j) графа G обозначаем $c(i,j)$. Длину кратчайшего пути из 1 в x будем называть расстоянием от вершины 1 до вершины x. Расстояние от вершины 1 до вершины x будем обозначать $s(x)$.

Ясно, что $s(1) = 0$. Пусть H – множество вершин, длины кратчайших путей из вершины 1 в которые уже известны. В начальной ситуации это множество одноэлементно: $H = \{1\}$. Через $s(1, M)$ обозначим минимальную из длин кратчайших путей от вершины 1 до вершин множества M, $M\subseteq\{2, 3,..., n\}$. Как очевидно, для множества вершин H, содержащего вершину 1, имеет место

$$s(1, N \setminus H) = \min_{i\in H, j\in N\setminus H} (s(i) + c(i,j)). \quad (1.40)$$

Пусть минимум правой части (1.40) реализуется на паре (i^0, j^0). Тогда кратчайший путь из вершины 1 в вершину j^0 получаем добавлением к кратчайшему пути из вершины 1 в вершину i^0 дуги (i^0,j^0), длина $s(j^0)$ этого пути равна $s(i^0) + c(i^0,j^0)$.

Формула (1.40) – рекуррентное соотношение динамического программирования для решения *ЗОКП*. Пользуясь этой формулой, на первом шаге определяем ближайшую к вершине *1* вершину i^1 из совокупности $\{2, 3,..., n\}$. На втором шаге определяем ближайшую к вершине *1* вершину i^2 из совокупности $\{2, 3,..., n\}\backslash\{i^1\}$. На третьем шаге определяем ближайшую вершине 1 вершину i^3 совокупности $\{2, 3,..., n\}\backslash\{i^1, i^2\}$, и т.д. В процессе счета строится дерево *D* кратчайших путей из вершины *1* в остальные вершины. Корнем *D* является вершина *1*; если на произвольном шаге алгоритма минимум правой части (1.40) реализуется на паре (i^0,j^0), к конструируемому дереву добавляется ребро (i^0,j^0).

Ход выполняемых вычислений можно оформить в виде процесса заполнения таблицы, строки которой соответствуют вершинам графа *G* (*i*-ая строка – вершине *i*, $i = \overline{1,n}$). Таблица заполняется по столбцам, слева направо (каждый столбец для определенности заполняется сверху вниз). При этом первый (левый) столбец таблицы имеет имя *"1"*. При заполнении столбца "*1*" в клетку, стоящую в пересечении с *j*-ой строкой, $j \in \{2, 3,..., n\}$, вносится *c(1 ,j)*, если дуга *(1, j)* в графе *G* имеется, и $+\infty$, если такой дуги в графе нет. После заполнения этого столбца, в нем символом * отмечается наименьший элемент (в случае, когда таких элементов несколько, отмечается любой из них). Если отмеченный символом * элемент равен *u* и находится в строке номер *v*, то длина кратчайшего пути от вершины *1* до вершины *v* найдена, *s(v)= u*. Первый шаг работы алгоритма реализован. Каждый следующий шаг алгоритма предусматривает заполнение очередных двух столбцов таблицы. Результатом каждого следующего шага является определение расстояния от вершины *1* до еще одной вершины. После того, как очередное расстояние *s(x)* определилось, соответствующая строка *x* таблицы считается отмеченной, дальнейшее заполнение клеток отмеченных строк не производится. В момент, когда строка становится отмеченной, во всех незаполненных ее клетках ставим специальный символ *z*. Изначально отмеченной считается первая строка таблицы, ибо значение *s(1)* известно с самого начала: *s(1) = 0.*

На втором шаге алгоритма второй столбец таблицы, а на последующих шагах – каждый следующий четный столбец, приобретает название последней вершины *x* графа *G*, для которой реализуемая вычислительная процедура в результате выполнения предшествующего шага определила расстояние от *1* до *x*. В заголовках первого, второго и далее каждого следующего четного столбца рядом с его именем "*x*" указывается значение *s(x)*. При заполнении второго и каждого следующего четного столбца "*x*" в каждую клетку, находящуюся в пересечении с неотмеченной строкой *j* вносится сумма *s(x) + c(x,j)*; если в графе *G* дуга *(x,j)* отсутствует, в клетку вносится символ $+\infty$. Третий и далее каждый следующий нечетный столбец имеет наименование "min". В каждую клетку такого столбца, стоящую в пересечении с *j*-ой неотмеченной строкой, вносится минимальное из расстояний, записанных в двух соседних слева клетках данной строки. После заполнения каждого очередного столбца "min", в нем отмечается символом * наименьший элемент (если таких элементов несколько, отмечается любой из них). Если отмеченный символом * элемент последнего заполненного столбца "min" равен *u* и находится в строке номер *v*, то, согласно (1.40), считаем найденной длину кратчайшего пути от вершины *1* до вершины *v*, *s(v)= u*. С этого момента *v*-ая строка таблицы переходит в число отмеченных строк, дальнейшее заполнение ее клеток производиться не будет. Следующий подлежащий заполнению столбец четный, ему назначается имя *"v"*, в заголовке этого столбца указывается также найденное значение *s(v)*. Далее заполняется очередной столбец "min". В результате заполнения каждой следующей пары столбцов определяется расстояние от вершины 1 до еще одной вершины графа, таким образом завершается реализация следующего шага алгоритма. Изначально

известно, что *s(1) = 0.* Поэтому общее число заполняемых пар столбцов *n-1*. Заполнение каждого очередного столбца требует выполнения линейно зависящего от *n* числа элементарных операций. Таким образом, реализация изложенного алгоритма требует выполнения квадратично зависящего от *n* числа элементарных операций.

ПРИМЕР 1.7. Взвешенный ориентированный граф *G* задан матрицей

$$M=\begin{pmatrix} 0 & 9 & 7 & \infty & 2 & 1 & 9 \\ \infty & 0 & 4 & \infty & 3 & 8 & 5 \\ 6 & 3 & 0 & 1 & \infty & \infty & 6 \\ \infty & 1 & \infty & 0 & 4 & 5 & 6 \\ \infty & 9 & 1 & 3 & 0 & 7 & \infty \\ \infty & 5 & 9 & 9 & \infty & 0 & 2 \\ 9 & 8 & 7 & 6 & \infty & 5 & 0 \end{pmatrix}$$

Числовой элемент m_{ij} этой матрицы равен весу дуги (*i,j*) графа *G*; если дуга в графе отсутствует, то $m_{ij} = \infty$. Веса дуг трактуются как их длины. Требуется найти расстояния от вершины *1* до остальных вершин графа.

Ход выполняемых в процессе решения вычислений отображаем как последовательное заполнение столбцов таблицы 1.5.

Таблица 1.5. Таблица определения расстояний.

	"1" 0	"6" 1	min	"5" 2	min	"3" 3	Min	"7" 3	min	"4" 4	min
1	z	z	z	z	z	z	Z	z	z	z	z
2	9	9	9	11	9	6	6	11	6	5	5*
3	7	8	7	3	3*	z	Z	z	z	z	z
4	∞	7	7	5	5	4	4	9	4*	z	z
5	2	∞	2*	z	z	z	z	z	z	z	z
6	1*	z	z	z	z	z	z	z	z	z	z
7	9	3	3	∞	3	9	3*	z	z	z	z

В итоге получаем, что в рассматриваемом графе *G* расстояния от вершины *1* до вершин *2, 3, 4, 5, 6* и *7* равны соответственно 5*, 3, 4, 2, 1* и *3*.

Отметим, что ранее для определяющего дискретную управляемую систему $\boldsymbol{\Omega}$ графа $G(\boldsymbol{\Omega})$ был изложен другой, основанный на рекуррентных соотношениях (1.3)-(1.5) и использующий ацикличность графа, алгоритм поиска кратчайших путей. Действительно, если в графе $G(\boldsymbol{\Omega})$ вес каждой дуги (*x, y*) трактуется как длина соответствующего одношагового перехода, то значение функции Беллмана $\boldsymbol{B^*(y)}$ – длина кратчайшего пути из начальной вершины в вершину $\boldsymbol{y}$.

Выше отмечалось, что метод динамического программирования относят к числу схем инвариантного погружения. Суть указанных схем в том, что каждая

подлежащая решению задача заменяется упорядоченным семейством однотипных задач, последней из которых является решаемая. Далее в порядке возрастания значений одного или нескольких параметров последовательно решаются задачи введенного семейства; знание результатов решения предыдущих задач существенно облегчает решение следующих; в итоге решенной оказывается исходная задача. Удачная замена исходной задачи семейством порождаемых ею задач – главное условие эффективности такого подхода.

Задача вычисления произведения матриц. Заданы прямоугольные матрицы M_1, $M_2, \ldots, M_n$ таких размеров, что произведение этих матриц

$$M = M_1 x M_2 x \ldots x M_n$$

определено. Известно, что умножение $(p \times q)$-матрицы на $(q \times r)$-матрицу требует выполнения $f(p,q,r)$ элементарных операций, где функция f – монотонно возрастающая по всем своим аргументам. Требуется найти порядок перемножения матриц $M_1, M_2, \ldots, M_n$, при котором являющаяся результатом перемножения матрица M вычисляется выполнением минимального числа арифметических операций.

Рассмотрим пример, в котором считаем, что умножение $(p \times q)$-матрицы на $(q \times r)$-матрицу требует выполнения $2pqr$ элементарных операций; это практически соответствует действительности. Пусть матрицы M_1, M_2, M_3 и M_4 имеют размеры $10x20$, $20x50$, $50x1$ и $1x100$ соответственно. Если вычислять M в порядке, определяемом записью $(M_1 x(M_2 x(M_3 x M_4)))$, то потребуется 250000 операций. Если же вычислять M в порядке, определяемом записью $((M_1 x(M_2 x M_3))x M_4)$, понадобится выполнить только 4400 операций.

Перебор всех порядков, в которых можно вычислять произведение n матриц, с целью отыскания наиболее экономной схемы, требует выполнения экспоненциально зависящего от n числа элементарных операций. При достаточно большом числе перемножаемых матриц это практически неприемлемо.

Задачу определения оптимальной (т.е. минимизирующей число выполняемых элементарных операций) схемы перемножения матриц $M_1, M_2, \ldots M_n$, каждая матрица M_i имеет размер $r_i x r_{i+1}$, $i = \overline{1,n}$, назовем задачей Z. Введем совокупность частных задач $Z(i,j)$, каждая задача $Z(i,j)$ заключается в синтезе схемы оптимального вычисления произведения $M_i x M_{i+1} x \ldots x M_j$, здесь $i = \overline{1,n}$; $j = \overline{1,n}$; $i \le j$. Минимальное число элементарных операций, необходимое для вычисления $M_i x M_{i+1} x \ldots x M_j$, обозначим m_{ij}; как очевидно, $m_{ii} = 0$.

Значения m_{ij} будем вычислять последовательно, в порядке возрастания разности индексов $j - i$. Считаем, что умножение $(p \times q)$-матрицы на $(q \times r)$-матрицу требует выполнения $f(p,q,r)$ элементарных операций. Тогда

$$m_{i\,i+1} = f(r_i, r_{i+1}, r_{i+2}),\ i = 1, 2, \ldots, n-1. \qquad (1.41)$$

Для $j - i > 1$ имеем:

$$m_{ij} = \min_{k \in \{i, i+1, \ldots, j-1\}} \{ m_{i\,k} + m_{k+1\,j} + f(r_i, r_{k+1}, r_{j+1}) \}. \quad (1.42)$$

Рекуррентные соотношения (1.41)-(1.42) позволяют находить значения m_{ij} и определять соответствующие схемы умножения матриц последовательно, в порядке увеличения разности $j - i$ от 1 до $n-1$. В результате окажется найденной величина m_{1n} и соответствующая оптимальная схема перемножения матриц $M_1, M_2, \ldots M_n$.

Процесс вычислений по приведенным соотношениям продемонстрируем на приведенном выше примере. Полагаем, что $f(p,q,r) = 2pqr$. Тогда в соответствии с (1.41):

$m_{12} = 2x10x20x50 = 20000;\quad m_{23} = 2x20x50x1 = 2\,000;\ m_{34} = 2x50x1x100 = 10000.$

Далее, пользуясь (1.42), получаем:

$m_{13} = \min\{ m_{11} + m_{23} + f(r_1, r_2, r_4),\ m_{12} + m_{33} + f(r_1, r_3, r_4)\} = \min\{ 0 + 2\,000 + 2x10x20x1,\ 20\,000 + 0 + 2x10x50x1\} = \min\{2\,400,\ 21\,000\} = 2\,400.$

$m_{24} = \min\{ m_{22} + m_{34} + f(r_2, r_3, r_5),\ m_{23} + m_{44} + f(r_2, r_4, r_5)\} = \min\{ 0 + 10000 + 2x20x50x100,\ 2\,000 + 0 + 2x20x1x100\} = \min\{20\,000,\ 4\,000\} = 4\,000.$

$m_{14} = \min\{ 0 + 4\,000 + 2x10x20x100,\ 20\,000 + 10\,000 + 2x10x50x100,\ 2\,400 + 0 + 2x10x1x100\} = \min\{44\,000,\ 130\,000,\ 4\,400\} = 4\,400.$

Таким образом, в приведенном примере минимальное число операций, необходимое для вычисления матрицы $M = M_1 x\, M_2 x\, M_3 x\, M_4$ равно *4 400*. Схема перемножения $((M_1x(M_2xM_3))xM_4)$ оказывается оптимальной.

Легко подсчитывается, что описанная процедура определения оптимальной схемы перемножения n матриц имеет кубично зависящую от n оценку числа выполняемых элементарных операций.

Задача отыскания максимального произведения. Пусть M_i – конечные множества натуральных чисел, $M_i = \{m^i_1, m^i_2, \ldots, m^i_{k(i)}\}$, $i = \overline{1,n}$; считаем, что входящие в состав каждого множества числа перечислены по возрастанию. В каждом из множеств M_i, $i = \overline{1,n}$, надо выбрать по одному элементу так, чтобы сумма этих чисел не превысила заданного числа S, а произведение оказалось максимально возможным. Предполагается, что $S < \sum_{i=1}^{n} m^i_{k(i)}$.

Обозначим через $B(r, z)$, где $r \in \{1, 2, \ldots, n\}$ и $z \in \{0, 1, 2, \ldots, S\}$, максимально возможную величину произведения r чисел при условиях: а) эти числа берутся из множеств $M_{n-r+1}, M_{n-r+2}, \ldots, M_n$ (по одному числу из каждого множества); б) сумма взятых чисел не превышает z. Как очевидно, $B(n,S)$ – искомое оптимальное значение критерия в решаемой задаче.

Если каждый элемент множества M_n больше z, полагаем $B(1,z) = 0$; в остальных случаях согласно сделанному определению $B(1,z)$ равно максимальному, не превосходящему z, элементу множества M_n. Далее для значений r, последовательно равных $2, 3, \ldots, n$, имеем:

$$B(r,z) = \max_{j \in U(r,z)} \{m^{n-r+1}_j \times B(r-1,\ z - m^{n-r+1}_j)\}; \quad (1.43)$$

здесь $U(r,z) = \{j \mid (j \in \{1, 2, \ldots, k(n-r+1)\ \&\ (m^{n-r+1}_j < z)\}$.

В случае, если каждый элемент множества M_{n-r+1} больше или равен z, значение $B(r,z)$ определяем равным нулю. Рекуррентное соотношение (1.43) дает возможность синтеза оптимального решения рассматриваемой задачи.

Рассмотренная схема решения задачи отыскания максимального произведения интересна тем, что заранее, не реализуя счет по записанным рекуррентным соотношениям, невозможно оценить стоимость перевода соответствующей системы из

произвольного, характеризуемого парой (r,z) состояния системы в состояние, непосредственно следующее за ним.

Классическая задача о назначениях (КЗН). Имеются множество исполнителей $I = \{1, 2,..., n\}$, множество работ $R = \{r_1, r_2 ,..., r_n\}$ и (nxn)-матрица численных оценок (например, производительностей) $A = \{a_{ij}\}$, где a_{ij} – оценка закрепления исполнителя i за работой r_j , $i=\overline{1,n}$, $j=\overline{1,n}$. *Назначения* – взаимно однозначные отображения множества $\{1, 2,..., n\}$ в себя; назначение π исполнителю i предписывает работу $r_{\pi(i)}$; численной оценкой такого закрепления является $a_{i\pi(i)}$ – элемент матрицы A, здесь $i=1,2,...,n$. Каждое назначение π оцениваем по критерию $K(\pi) = \sum_{i=1}^{n} a_{i\pi(i)}$; в указанной интерпретации значение этого критерия – суммарная производительность исполнителей при назначении π. Требуется найти назначение, при котором суммарная производительность исполнителей максимальна.

КЗН формулируется в виде

$$\max_{\pi} \sum_{i=1}^{n} a_{i\pi(i)}$$

Рассмотрим вопрос о решении записанной *КЗН,* обозначим ее символом *Z,* методом динамического программирования. С этой целью введем в рассмотрение совокупность частных задач $\boldsymbol{Z}(i, W_i)$, формулируемых по исходным данным задачи $\boldsymbol{Z}$; здесь $i \in \{1,2,...,n\}$, а W_i – i - элементные подмножества совокупности $\{1,2,...,n\}$. В задаче $\boldsymbol{Z}(i, W_i)$ между исполнителями $\{1,2,..., i\}$ следует распределить совокупность работ, индексы которых перечислены в W_i; критерий задачи прежний – суммарная производительность имеющихся исполнителей. Оптимальное значение критерия в задаче $\boldsymbol{Z}(i, W_i)$ обозначим $\boldsymbol{B}^*(i, W_i)$. В таком случае $\boldsymbol{B}^*(n, \{1,2,...,n\})$ – оптимальное значение критерия в исходной задаче Z.

Как очевидно,

$$\boldsymbol{B}^*(1, \{j\}) = a_{1j} \text{ для всех } j \in \{1,2,...,n\}. \quad . \quad (1.44)$$

Если $i>1$, имеем

$$\boldsymbol{B}^*(i, W_i) = \max_{j \in W_i} (a_{ij} + \boldsymbol{B}^*(i-1, W_i \setminus \{j\}), \; i= 2,3,...,n;. \quad (1.45)$$

здесь W_i – произвольное i-элементное подмножество из совокупности $\{1,2,...,n\}$.

Записанные равенства (1.44)-(1.45) являются соотношениями динамического программирования для решения *КЗН*. Легко видеть, что оценкой числа элементарных операций, выполняемых основанным на соотношениях (1.44) - (1.45) алгоритмом, является функция, которая экспоненциально зависит от n. В отличие от всех остальных рассматриваемых в данном разделе задач, для *КЗН* имеются алгоритмы, обладающие существенно лучшими в сравнении с процедурой счета по соотношениям динамического программирования, характеристиками быстродействия.

Классическая задача о назначениях с кубично зависящей от n верхней оценкой числа выполняемых элементарных операций решается, в частности, алгоритмом Куна. Изложение этого алгоритма приведено в ряде источников (см., например, [6, 10]).

Задача о назначениях с учетом предпочтений исполнителей Начальные данные задачи определяем как набор объектов

$$\Sigma = \langle I, R, A, L \rangle,$$

где $I = \{1, 2,..., n\}$ – совокупность исполнителей; $R = \{r_1, r_2,..., r_n\}$ – совокупность работ; $A = \{a_{ij}\}$ – (nxn)-матрица численных оценок, a_{ij} – оценка закрепления исполнителя i за работой r_j; $L = \{ \preceq_1, \preceq_2,..., \preceq_n\}$ – упорядочения, определяющие предпочтения исполнителей на множестве работ (выполнение отношения $x \preceq_i y$ означает, что для исполнителя i работа y в сравнении с работой x является не менее предпочтительной).

Если соотношения $x \preceq_i y$ и $y \preceq_i x$ имеют место одновременно, то работы x и y для исполнителя i считаются равноценными, эквивалентными (в таком случае используем обозначение $x \sim_i y$). Если $x \preceq_i y$, но работы x и y для исполнителя i неэквивалентны, то работа y для исполнителя i является лучшей, более предпочтительной в сравнении с работой x, а работа x соответственно худшей в сравнении с работой y (в данной ситуации используем обозначение $x <_i y$). Считаем, что для любого исполнителя i любые две работы либо эквивалентны, либо связаны отношением $<_i$, выполняющимся либо в одну, либо в другую сторону.

Определяющий предпочтения исполнителей набор $L = \{ \preceq_1, \preceq_2,..., \preceq_n\}$ можно задать посредством (nxn)-матрицы балльных оценок $B_L = \{b_{ij}\}$, где b_{ij} – балльная оценка, выставляемая исполнителем i работе r_j. При этом $b_{ix} \leq b_{iy}$ тогда и только тогда, когда

$$r_x. \preceq_i r_y.$$

Если работы x и y для исполнителя i эквивалентны, то данным работам этот исполнитель выставляет равные балльные оценки

Каждое назначение π определяет систему пар $\{(1, r_{\pi(1)}), (2, r_{\pi(2)}), \ldots, (n, r_{\pi(n)})\}$. Совокупность всевозможных назначений в задаче Σ обозначим $H(\Sigma)$.

Назначение π называем *устойчивым*, если не существует группы (коалиции) исполнителей S, $S \subseteq \{1, 2,..., n\}$, члены которой могут обменяться между собой полученными в результате реализации этого назначения работами так, что каждый член коалиции i, $i \in S$, получит работу, с точки зрения его предпочтений лучшую, чем работа $r_{\pi(i)}$

Через $\pi(S)$ обозначим совокупность работ, получаемых исполнителями множества S, $S \subseteq \{1, 2,..., n\}$, при реализации назначения π.

Т е о р е м а 1.4. Назначение π устойчиво тогда и только тогда, когда в любом подмножестве исполнителей S, $S \subseteq \{1, 2,... , n\}$, найдется исполнитель i, которому этим назначением предписана лучшая (с точки зрения предпочтений исполнителя i) работа из множества $\pi(S)$.

Условие теоремы достаточно. Действительно, если назначение π обладает сформулированным свойством, то никакая коалиция S не способна перераспределить внутри себя получаемые по назначению π работы так, чтобы каждый член коалиции улучшил свое положение – ведь по меньшей мере один исполнитель из S по назначению π получает лучшую с точки зрения его предпочтений работу из множества $\pi(S)$.

Перейдем к доказательству необходимости. Пусть назначение π устойчиво. Предположим, что в некотором подмножестве исполнителей S не найдется такого, которому этим назначением предписана лучшая в $\pi(S)$ работа. Построим

ориентированный граф $G(S, \pi)$, вершины которого одноименны с элементами множества S, а дуга из вершины x в вершину y проводится в том и только том случае, когда для исполнителя x работа $r_{\pi(y)}$ предпочтительнее работы $r_{\pi(x)}$. Из конечности графа $G(S, \pi)$ и того, что из каждой его вершины исходит по меньшей мере одна дуга (для любого исполнителя из S в $\pi(S)$ имеется работа, которая лучше предписанной) вытекает наличие в графе $G(S, \pi)$ циклов. Пусть C – один из них, будем считать, что этот цикл охватывает вершины $i_1, i_2,..., i_p$ (полагаем, что вершины перечислены в порядке их следования в цикле, за вершиной i_p следует вершина i_1). В таком случае каждому из исполнителей $i_1, i_2,..., i_p$ выгодно обменяться полученными в назначении π работами, реализовав цикл обмена C. Методом от противного установлена необходимость сформулированного в теореме условия устойчивости. Теорема доказана полностью.

Процедура построения произвольного устойчивого решения задачи $\Sigma = \langle \boldsymbol{I}, \boldsymbol{R}, \boldsymbol{A}, \boldsymbol{L} \rangle$ состоит в следующем. Из множества $\boldsymbol{I}$ выбирается любой исполнитель i_1, которому предписывается лучшая (исходя из системы предпочтений данного исполнителя) работа $r_{i(1)}$. Далее из множества $\boldsymbol{I}\backslash\{i_1\}$ выбирается любой исполнитель i_2, которому предписывается лучшая для него работа из множества $\boldsymbol{R}\backslash\{r_{i(1)}\}$, обозначим эту работу $r_{i(2)}$. Затем из множества $\boldsymbol{I}\backslash\{i_1, i_2\}$ выбирается любой исполнитель i_3, которому предписывается лучшая для него работа из множества $\boldsymbol{R}\backslash\{r_{i(1)}, r_{i(2)}\}$, обозначим эту работу $r_{i(3)}$, и т.д. Описанная процедура выполняется вплоть до распределения между всеми исполнителями всех имеющихся работ. Варьируя на каждом ее этапе выбор очередного исполнителя и определение закрепляемой за ним работы (лучшая для исполнителя работа не всегда оказывается единственной), мы можем построить любое из устойчивых в рассматриваемой задаче назначений.

Для численной оценки назначений вводим критерии $K_1(\pi) = \sum_{i=1}^{n} a_{i\pi(i)}$ и $K_2(\pi) = \min_i a_{i\pi(i)}$. Будем считать, что исполнители свободны в принятии окончательного варианта распределения работ и что качество назначения π каждым исполнителем оценивается предпочтительностью получаемой им работы. В такой ситуации возможными следует считать только устойчивые назначения. Множество устойчивых в задаче Σ назначений будем обозначать $M(\Sigma)$.

Сформулируем две экстремальные задачи о назначениях с учетом предпочтений участников:

Задача 1. Найти $\max K_1(\pi)$ при условии, что $\pi \in M(\Sigma)$.

Задача 2. Найти $\max K_2(\pi)$ при условии, что $\pi \in M(\Sigma)$.

Пусть Σ^* – произвольная подмодель модели Σ, получаемая из нее изъятием некоторого множества исполнителей и такого же по мощности множества работ. Имеющиеся в подмодели Σ^* множества исполнителей и работ обозначим I^* и R^* соответственно. Оптимальные значения введенных критериев $K_1(\pi)$ и $K_2(\pi)$ в формулируемых для подмодели Σ^* задачах 1 и 2 обозначим $K_1(I^*, R^*)$ и $K_2(I^*, R^*)$ соответственно.

Если I^* и R^* – одноэлементные множества, $I^*=\{i\}$ и $R^*=\{r_j\}$, то, как очевидно,

$$K_1(\{i\},\{r_j\}) = K_2(\{i\},\{r_j\}) = a_{ij}. \qquad (1.46)$$

Отметим, что в силу принятых обозначений $K_1(I, R)$ и $K_2(I, R)$ – искомые оптимальные значения критериев задач 1 и 2 соответственно в их полных постановках.

Через $\Sigma^*[\, i, r_j\,]$ обозначим модель, получаемую из Σ^* изъятием имеющихся в ней исполнителя i и работы r_j .

Т е о р е м а 1.5. Имеют место следующие соотношения:

$$K_1(I^*, R^*) = \max_{i \in I^*} \{ \max_{j \in R^*(i)} [a_{ij} + K_1(I^* \setminus i, R^* \setminus r_j)]\}; \qquad (1.47)$$

$$K_2(I^*, R^*) = \max_{i \in I^*} \{ \max_{j \in R^*(i)} \mathbf{min}[a_{ij}, K_2(I^* \setminus i, R^* \setminus r_j)]\}; \qquad (1.48)$$

здесь $R^*(i)$ обозначает множество номеров лучших для исполнителя i (с точки зрения его предпочтений) работ из совокупности R^*.

Данная теорема – непосредственное следствие теоремы 1.4.

Равенства (1.46) - (1.48) – рекуррентные соотношения для решения задач 1 и 2 прямым методом Беллмана.

Следует отметить, что метод динамического программирования не может быть применен в ситуациях, когда совокупность возможных в некоторые моменты управлений зависит не только от состояния, в котором оказалась система, но и от того, какие управления применялись ранее. Этот метод, вообще говоря, неприменим и тогда, когда численная оценка начальной или заключительной части траектории не может быть вычислена без использования некоторой информации о других ее фрагментах. В таких случаях нередко оказывается, что применимость метода можно все же обеспечить за счет расширения в определении концепции состояния. Число состояний системы (а, следовательно, и подлежащих решению частных задач) при этом существенно увеличивается, поэтому вычислительная сложность решающей процедуры значительно возрастает.

Для иллюстрации рассмотрим задачу о ранце с несовместимыми парами предметов. Ее постановка отличается от классической только в следующем: указан перечень пар индексов несовместимых предметов $U = \{(i_h, j_h),\ h = 1, 2, \ldots, k\}$. Считается, что предметы $П_x$ и $П_y$ одновременно не могут находиться в ранце, если пара (x,y) входит в множество U. Считается, что в каждой паре (i_h, j_h) первая компонента строго меньше второй компоненты. Далее будут использоваться также определяемые следующим образом подмножества $U(i)$ совокупности U: входящая в U пара индексов (x, y) принадлежит $U(i)$ тогда и только тогда, когда одна из ее компонент не превосходит i, а другая – строго больше чем i.

При составлении соотношений динамического программирования для решения классической задачи о ранце $\boldsymbol{Z}$ нами вводилась совокупность частных задач $\boldsymbol{Z}(i, p)$. Состояниями системы были пары (i, p), где первый параметр определял совокупность предметов, относительно которых уже принято решение (включать или не включать в ранец), а второй параметр принимал значение, равное суммарному весу предметов, уже включенных в ранец. При составлении соотношений динамического программирования для решения указанной обобщенной задачи о ранце состояния системы следует записывать в виде троек (i, p, M), где параметры i и p имеют то же значение, что раньше, а M – множество индексов помещенных в ранец предметов, для которых в совокупности $U(i)$ имеются парные.

Аналогичное расширение концепции состояния системы целесообразно и при рассмотрении в известном смысле противоположной ситуация – пары совокупности U желанны, добавление в ранец каждого недостающего парного предмета связано с фиксированным дополнительным доходом.

ЗАДАЧИ К ГЛАВЕ 1.

1. Имеются сферы капиталовложений S_1, S_2, S_3, S_4; в каждую из них может быть внесен целочисленный вклад, не превосходящий 5. Величина подлежащего распределению капитала равна 12. Функции, определяющие доходы, получаемые при вложении в имеющиеся сферы вклада u, заданы приведенной таблицей. Прямым методом Беллмана требуется найти оптимальное распределение капитала между сферами вложений.

Таблица значений функций дохода $f_t(u)$.

	$f_1(u)$	$f_2(u)$	$f_3(u)$	$f_4(u)$
u = 0	0	0	0	0
u = 1	0,1	0,1	0,2	0
u = 2	0,3	0,1	0,2	0
u = 3	0,3	0,4	0,3	0,3
u = 4	0,4	0,4	0,3	0,4
u = 5	0,5	0,4	0,3	0,4

2. Автомобиль, начиная от базы и заканчивая на базе эамкнутый маршрут, должен доставить с базы в точки $T_1, T_2, \ldots, T_m$ некоторые грузы. Известно, что вес направляемого в точку T_i груза равен c_i, $i = \overline{1,m}$. Выполняя тот же маршрут, автомобиль должен собрать в точках $R_1, R_2, \ldots, R_n$ грузы, адресованные базе. Известно, что вес груза, направляемого из точки R_j на базу, равен w_j, $j=\overline{1,n}$. Через каждую из перечисленных точек маршрут должен проходить однократно. Матрица расстояний S размера $(m+n+1)\times(m+n+1)$ задана. Требуется записать рекуррентные соотношения динамического программирования для определения маршрута, минимального по количеству выполняемых тонно-километров.

3. Записать рекуррентные соотношения динамического программирования для классической задачи о ранце, дополненной условием: из каждой пары предметов $(П_1, П_n),.(П_2, П_{n-1}),\ldots,),\ldots,(П_k, П_{n-k+1}$ в ранец можно .поместить только один предмет, здесь $k \le n/2$.

4. Решить задачу о ранце:

$$7x_1 + 4x_2 + 4x_3 + 5x_4 + 2x_5 + 3x_6 \to max$$

при условиях

$$2x_1 + x_2 + 2x_3 + 4x_4 + 3x_5 + 2x_6 \le 10;$$

$$x_i \in \{0,1\},\ i=\overline{1,5}.$$

5. Решить следующую модификацию классической задачи о ранце:

$$5x_1 + 2x_2 + 4x_3 + 7x_4 + 5x_5 \to max$$

при условиях

$$2x_1 + x_2 + 2x_3 + 4x_4 + 3x_5 \le 12;$$

$$x_i \in \{0,1\},\ i=1,\ 2,5;$$

$$x_i \in \{0,1,2\},\ i=3,4.$$

6. Имеется множество недробимых камней $\{K_1, K_2, \ldots, K_8\}$. Веса камней (в порядке возрастания индексов) следующие: *7, 5, 8, 4, 3, 6, 3, 1*. Требуется разбить совокупность $\{K_1, K_2, \ldots, K_8\}$ на два подмножества так, чтобы суммарный вес камней одного подмножества минимально отличался от суммарного веса камней другого подмножества.

7. Задан взвешенный ориентированный граф с двумя выделенными вершинами, *s* и *t*. Веса дуг – это их длины; вес каждой дуги – положительное число. Требуется найти самый длинный простой (т.е. без самопересечений) путь, начинающийся в вершине *s* и заканчивающийся в вершине *t*. Записать рекуррентные соотношения динамического программирования для решения задачи.

8. Взвешенный ориентированный граф *G* задан матрицей

$$M = \begin{pmatrix} 0 & 7 & 1 & \infty & 5 & 6 & 9 \\ \infty & 0 & 8 & \infty & 3 & 2 & 5 \\ 1 & 3 & 0 & 1 & \infty & \infty & 6 \\ \infty & 1 & \infty & 0 & 4 & 5 & 6 \\ \infty & 9 & 1 & 3 & 0 & 7 & \infty \\ \infty & 5 & 9 & 9 & \infty & 0 & 2 \\ 5 & 3 & 7 & 6 & \infty & 5 & 0 \end{pmatrix}$$

Числовой элемент m_{ij} матрицы *M* равен весу дуги (*i,j*) графа *G*; если дуга (*i,j*) в графе отсутствует, то $m_{ij} = \infty$. Веса дуг трактуются как их длины. Требуется найти расстояния от вершин *1* и *2* до остальных вершин графа.

9. Найти оптимальную схему перемножения матриц M_1, M_2, M_3 , M_4 и M_5 размеров *10x20, 20x15, 15x5, 5x2* и *2x10* соответственно.

10. Найти устойчивое назначение, максимизирующее суммарную производительность участников при условии, что матрица производительностей *A* и матрица балльных оценок *B* определены следующим образом:

$$A = \begin{pmatrix} 5 & 7 & 7 & 4 & 3 \\ 1 & 2 & 8 & 2 & 1 \\ 3 & 4 & 7 & 4 & 9 \\ 3 & 1 & 3 & 1 & 5 \\ 8 & 2 & 4 & 6 & 1 \end{pmatrix}; \qquad B = \begin{pmatrix} 0 & 2 & 3 & 1 & 2 \\ 1 & 0 & 1 & 2 & 3 \\ 1 & 2 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 2 & 3 & 4 \end{pmatrix}$$

ГЛАВА 2. ПРИМЕНЕНИЕ МЕТОДА ДИНАМИЧЕСКОГО ПРОГРАММИРОВАНИЯ К ЗАДАЧАМ СИНТЕЗА РАСПИСАНИЙ ОБСЛУЖИВАНИЯ.

В данной главе формулируются и решаются задачи синтеза расписаний обслуживания; обслуживанию подлежат объекты, условно именуемые заявками. Излагаемые алгоритмы основаны, главным образом, на принципе динамического программирования. Предполагается, что в каждой задаче счет времени идет от момента начала реализации работ по расписанию; таким образом, выполнение каждого расписания начинается от момента $t = 0$. Время считается дискретным (его счет идет по тактам), в каждой задаче все временные характеристики целочисленные. Рассматриваемые задачи делятся на две группы. В задачах первой группы считается, что по состоянию на момент времени $t=0$ все заявки как физические объекты готовы к обслуживанию. В задачах второй группы считается, что по состоянию на момент $t=0$ в систему обслуживания поступила лишь часть заявок, для каждой из остальных заявок предполагается известным момент ее прибытия (начиная с момента прибытия заявка готова к обслуживанию). Задачи первой группы будем называть задачами обслуживания множеств заявок, задачи второй группы – задачами обслуживания потоков заявок.

2.1. Задачи обслуживания множеств заявок.

Задача мастера. Имеется множество заявок $\{1,2,...,n\}$ и процессор P, на котором должна пройти обслуживание каждая заявка; процессор не может обслуживать несколько (т.е. более одной) заявок одновременно. Обслуживание каждой заявки (от начала до конца) реализуется без прерываний; переход от обслуживания одной заявки к обслуживанию следующей за ней заявки затрат времени не требует. Обслуживание заявок начинается от момента времени $t = 0$. Для каждой заявки i известны две характеристики: $\tau(i)$ – продолжительность обслуживания на процессоре; $a(i)$ – штраф за единицу времени пребывания в системе обслуживания. Считаем, что $\tau(i) > 0$, $a(i) \geq 0$, $i = \overline{1,n}$.

Каждое расписание обслуживания определяем как перестановку $\rho = (i_1, i_2, ..., i_n)$ элементов множества $\{1, 2, ..., n\}$; заявка i_k в расписании ρ обслуживается k-ой по очереди, $k = \overline{1,n}$. Обозначим через $t_{нач}(\rho, i_k)$ и $t^*(\rho, i_k)$ моменты начала и завершения обслуживания заявки i_k при реализации расписания ρ. Имеют место соотношения:

$$t_{нач}(\rho, i_1) = 0; \qquad (2.1)$$

$$t^*(\rho, i_k) = t_{нач}(\rho, i_k) + \tau(i_k), \quad k=1, 2, \ldots, n; \qquad (2.2)$$

$$t_{нач}(\rho, i_k) = t^*(\rho, i_{k-1}), \; k=2, 3, \ldots, n. \qquad (2.3)$$

При реализации расписания ρ величина штрафа $W_i(\rho)$ по произвольной заявке i оказывается равной $a(i)t^*(\rho,i)$. Суммарный штраф по всем заявкам, обслуживаемым согласно расписанию ρ, есть

$$W(\rho) = \sum_{i=1}^{n} W_i(\rho) = \sum_{i=1}^{n} a(i)t^*(\rho,i).$$

Проблема заключается в синтезе расписания, обеспечивающего минимальное значение суммарного штрафа, т.е. в нахождении

$$\min_{\rho} W(\rho) \quad . \qquad (2.4)$$

В стандартной для задачи мастера интерпретации считается, что заявки – это сломавшиеся станки, процессор – мастер по их ремонту, *a(i)* – величина потерь, связанных с каждой единицей времени простоя *i*- го станка; *τ(i)* – время, которое мастер должен затратить на ремонт *i*- го станка, $i = \overline{1,n}$.

Предположим, что расписание $\rho^0 = (1, 2, \ldots, n)$ в задаче (2.4) является оптимальным. Через ρ^q обозначим расписание, получаемое из ρ^0 переменой местами двух соседних заявок, *q* и *q+1*. Очевидно, что для каждого значения *i*, отличного от *q* и от *q +1*, имеет место $t^*(\rho^q, i) = t^*(\rho^0, i)$. Поэтому в суммах $\sum_{i=1}^{n} W_i(\rho^0)$ и $\sum_{i=1}^{n} W_i(\rho^q)$ отличаются между собой только *q*- ое и (*q +1*)-ое слагаемые. Легко определяется, что $\sum_{i=1}^{n} W_i(\rho^0) - \sum_{i=1}^{n} W_i(\rho^q) = a(q+1)\tau(q) - a(q)\tau(q+1)$. Так как расписание ρ^0 –оптимально, записанная разность неположительна. Получаем

$$a(q+1)\ \tau(q) - a(q)\ \tau(q+1) < 0,$$

т.е.

$$a(q)\ \tau(q+1) > a(q+1)\ \tau(q).$$

Разделив обе части полученного неравенства на положительное произведение $\tau(q)\ \tau(q+1)$, получаем

$$\{a(q) / \tau(q)\} > \{a(q+1) / \tau(q+1)\}. \qquad (2.5)$$

Неравенство (2.5) – следствие сделанного предположения об оптимальности расписания ρ^0. Из этого неравенства вытекает простейший алгоритм синтеза оптимального расписания в задаче мастера: для каждой заявки *i* нужно определить показатель $\mu(i) = a(i) / \tau(i)$, $i = \overline{1,n}$, и упорядочить заявки по убыванию значений $\mu(i)$. Полученная перестановка является оптимальным расписанием обслуживания имеющегося множества заявок.

Изложенный способ отыскания правила оптимального упорядочения заявок называется перестановочным приемом. Найденный для задачи мастера способ синтеза оптимального расписания именуем μ - правилом. Содержательный смысл его очевиден: чем меньше продолжительность обслуживания заявки и чем больше по этой заявке штраф за единицу времени пребывания в системе, тем раньше она должна обслуживаться.

Двухпроцессорная задача мастера. Отличие рассматриваемой задачи от предыдущей состоит в наличии двух идентичных процессоров, P_1 и P_2. Каждая заявка множества $\{1,2,\ldots,n\}$ должна быть обслужена либо первым, либо вторым процессором. Оба процессора считаются готовыми к обслуживанию заявок начиная с момента времени $t = 0$. Обслуживание одной заявки двумя процессорами считается невозможным. Расписание определяем как пару $\rho=(\rho_1, \rho_2)$, где $\rho_1= (i_1,i_2,i_3,\ldots,i_p)$ - перестановка, определяющая последовательность обслуживания заявок первым процессором, а $\rho_2= (j_1,j_2,j_3,\ldots,j_q)$ - перестановка, определяющая последовательность

обслуживания заявок вторым процессором. Каждая заявка входит либо в первую, либо во вторую перестановку, поэтому $p + q = n$. Требуется найти расписание, минимизирующее суммарный по всем заявкам штраф.

По аналогии с результатами, полученными для однопроцессорной задачи мастера, можно предположить, что оптимальным является расписание, синтезируемое в соответствии с интерпретируемым следующим образом μ - правилом: упорядочиваем совокупность всех заявок по убыванию показателя $\mu(i)$, в этом порядке выполняем их перенумерацию и в этом же порядке направляем на обслуживание освобождающимися процессорами. В синтезируемом расписании процессоры P_1 и P_2 сначала должны обслужить заявки *1* и *2* соответственно (все номера заявок – новые), заявка *3* направляется следующей на процессор, который раньше освободится. Точно так же заявка *4* направляется следующей на процессор, который раньше обслужит поступившие ему заявки из совокупности {*1, 2, 3*}, и т.д.

Покажем, что в двухпроцессорной задаче мастера конструируемое применением μ - правила расписание не всегда оптимально. Рассмотрим пример, в котором множество подлежащих обслуживанию заявок трехэлементно, их характеристики заданы таблицей 2.1.

Таблица 2.1. Характеристики заявок.

	$A(i)$	$\tau(i)$	$\mu(i)$
Заявка1	3	1	3
Заявка2	4	2	2
Заявка3	100	100	1

Применением μ - правила в рассматриваемом примере строится расписание $\rho^0 = (\rho_1, \rho_2)$, где $\rho_1 = (1, 3)$ *и* $\rho_2 = (2)$. В случае реализации этого расписания суммарный штраф по трем заявкам равен *10111*. В то же время при реализации расписания $\rho^1 = ((1,2), (3))$ будем иметь суммарный штраф, равный *10015*. Таким образом, в двухпроцессорной задаче мастера расписание, синтезируемое в соответствии с μ - правилом, оптимальным вообще говоря, не является.

Для синтеза оптимального в двухпроцессорной задаче мастера расписания применим метод динамического программирования. Будем считать, что все заявки пронумерованы в порядке убывания значений показателя μ. Тогда если $\rho=(\rho_1, \rho_2)$ – оптимальное расписание, то заявки в каждой из последовательностей $\rho_1 = (i_1,i_2,i_3,\ldots,i_p)$ и $\rho_2 = (j_1,j_2,j_3,\ldots,j_q)$ записаны в порядке возрастания номеров.

При синтезе оптимального расписания для каждой следующей (в порядке возрастания номеров, т.е. по убыванию показателя μ) заявки $i + 1$ мы должны определить, на какой из двух процессоров ее направить в качестве подлежащей обслуживанию в следующую очередь. При этом все заявки множества {*1, 2,…,i*} по процессорам уже распределены, известны величины промежутков времени, которое будет потрачено на обслуживание этих заявок первым и вторым процессорами; время работы процессора P_1 над предписанными ему заявками из совокупности {*1, 2,…,i*} обозначим Δ. В таком случае время работы процессора P_2 над заявками из той же совокупности равно $\sum_{\alpha=1}^{i} \tau(i) - \Delta$. Через $W^*(i, \Delta)$ обозначим минимально возможную величину суммарного штрафа по заявкам множества {*1, 2,…,i*}, если на обслуживание направленных на процессор P_1 заявок из этого множества затрачивается время Δ. Как очевидно,

$$W^*(1,\ \tau(1)) = W^*(1,\ 0) = a(1)\cdot\tau(1). \quad (2.6)$$

При значениях аргумента Δ, отличных от нуля и $\tau(1)$, определяемая парой $(1, \Delta)$ ситуация возникнуть не может; удобно положить

$$W^*(1, \Delta) = +\infty, \text{ если } \Delta \notin \{0,\ \tau(1)\};. \quad (2.7)$$

далее для любых нереализуемых ситуаций (i, Δ) из рекуррентных соотношений будем получать $W^*(i, \Delta) = +\infty$.

Предположим, что все значения функции $W^*(i, \Delta)$ для некоторого конкретного значения i уже найдены. Рассмотрим ситуацию в процессе обслуживания, определяемую парой $(i+1, \Delta)$. В случае $\Delta \geq \tau(i+1)$ непосредственно предшествующими для этой ситуации являются две: (i, Δ) и $(i, \Delta - \tau(i+1))$; в случае $\Delta < \tau(i+1)$ непосредственно предшествующей является только ситуация (i, Δ); значение $W^*(i, \Delta - \tau(i+1))$, где второй аргумент функции отрицательный, считается равным $+\infty$. Переход от ситуации (i, Δ) к ситуации $(i+1, \Delta)$ предполагает, что обслуживание заявки $i+1$ выполняется процессором P_2; обслуживание этой заявки начинается в момент времени $\sum_{\alpha=1}^{i}\tau(\alpha) - \Delta$ и завершается в момент $\sum_{\alpha=1}^{i+1}\tau(\alpha) - \Delta$. Переход от ситуации $(i, \Delta - \tau(i+1))$ к ситуации $(i+1, \Delta)$ предполагает, что обслуживание заявки $i+1$ выполняется процессором P_1; обслуживание начинается в момент времени $\Delta - \tau(i+1))$ и завершается в момент Δ..

Отсюда получаем:

$$W^*(i+1, \Delta) = \min\{ W^*(i, \Delta - \tau(i+1)) + a(i+1)\cdot\Delta,\ W^*(i, \Delta) + a(i+1)\{\sum_{\alpha=1}^{i+1}\tau(\alpha) - \Delta\}\} \quad (2.8)$$

Процедуру вычисления значений функции $W^*(i, \Delta)$ удобно представлять как процесс заполнения таблицы, строки которой соответствуют значениям параметра i, а столбцы – значениям параметра Δ; в клетку, являющуюся пересечением строки i и столбца Δ, вносится значение $W^*(i, \Delta)$. Общее число строк таблицы значений функции $W^*(i, \Delta)$ равно n. Величины $W^*(i, \Delta)$ требуется определять для значений второго аргумента из множества $\{0, 1, \ldots, \sum_{\alpha=1}^{n}\tau(\alpha)\}$, поэтому общее число столбцов таблицы равно $\sum_{\alpha=1}^{n}\tau(\alpha) + 1$. Строки таблицы заполняются сверху вниз, в порядке роста значений параметра i. Первая строка заполняется по формулам (2.6) – (2.7), остальные – по формуле (2.8). Отметим, что в силу идентичности процессоров в клетки $(i+1, \Delta)$ и $(i, \sum_{\alpha=1}^{i+1}\tau(\alpha) - \Delta)$.вносятся одинаковые числа; если при вычислении значения $W^*(i+1, \Delta)$ минимум правой части (2.8) реализуется на первой (второй) компоненте, то при отыскании величины $W^*(i, \sum_{\alpha=1}^{i+1}\tau(\alpha) - \Delta)$ минимум правой части (2.8) реализуется на второй (соответственно первой) компоненте. Оптимальным значением критерия в решаемой задаче является минимальный элемент, внесенный в нижнюю строку таблицы значений функции $W^*(i, \Delta)$. Если данный элемент находится в столбце Δ', то общее время работы одного процессора в реализации оптимального расписания равно Δ', время работы другого процессора равно $\sum_{\alpha=1}^{n}\tau(\alpha) - \Delta'$.

Для обеспечения возможности синтеза оптимального расписания в каждую клетку *(i+1, Δ)* таблицы кроме значения *W*(i+1, Δ)*, требуется записывать условный символ, указывающий, на какой компоненте правой части соотношения (2.8) при вычислении *W*(i+1, Δ)* реализуется минимум.

Равенства (2.6) – (2.8) – рекуррентные соотношения динамического программирования для решения двухпроцессорной задачи мастера.

ПРИМЕР 2.1. Требуется построить оптимальное расписание обслуживания заявок 1 –5 в системе, состоящей из двух идентичных процессоров. Характеристики заявок следующие: *a(1) = 5, τ(1) =2; a(2) = 4, τ(2) =1; a(3) =7, τ(3) =3;a(4) =3, τ(4) =3;a(5) =2, τ(5) =2.*

Решение начинается с переупорядочивания (введения новой нумерации) заявок с тем, чтобы заявкам с большими номерами соответствовали меньшие значения показателя μ. В рассматриваемом примере заявке 2 присваивается новый номер 1, заявке 1 – новый номер 2; остальные заявки номеров не меняют. Складывая заданные времена обслуживания заявок, получаем, что максимальное значение параметра Δ равно 11. Результаты выполненной в соответствии с соотношениями (2.6) - (2.8) процедуры счета представлены в таблице 2.2. В клетки, где при вычислении внесенного значения функции минимум правой части (2.8) реализуется на первой компоненте, внесен дополнительный символ *.

Таблица 2.2. Значения функции *W*(i, Δ)*.

	0	1	2	3	4	5	6	7	8	9	10	11
1	4	4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
2	19	14	14*	19*	∞	∞	∞	∞	∞	∞	∞	∞
3	61	49	42	40*	42*	49*	61*	∞	∞	∞	∞	∞
4	88	73	63	58	57	57*	58*	63*	73*	88*	∞	∞
5	110	93	81	74	71*	68*	68	71	74*	81*	93*	110*

Минимальный элемент пятой, нижней строки таблицы равен 68; это минимальная величина суммарного штрафа по всем заявкам в рассматриваемом примере. По заполненной таблице синтез оптимального расписания выполняется следующим образом. В нижней строке фиксируется один из минимальных элементов. Для определенности считаем, что в качестве минимального в нижней строке взят элемент, стоящий в пятом столбце, т.е. заключительной является ситуация (5,5), на обслуживание всех заявок первый процессор затрачивает 5 тактов дискретного времени. Так как взятый в нижней строке таблицы минимальный элемент отмечен звездочкой, заявка 5 должна обслуживаться процессором P_1, на это затрачиваются 2 такта. Из отмеченного вытекает, что ситуацией, непосредственно предшествующей ситуации (5,5), является (4,3). Стоящий в пересечении четвертой строки и третьего столбца таблицы элемент звездочкой не отмечен, заявка 4 должна обслуживаться процессором P_2, непосредственно предшествующей ситуации (4,3) является ситуация (3,3). Элемент, стоящий в пересечении третьей строки и третьего столбца отмечен звездочкой; заявка 3 должна обслуживаться процессором P_1. Непосредственно предшествующей ситуации (3,3) является ситуация (2,0). Последнее означает, что заявки 1 и 2 должны обслуживаться процессором P_2. В итоге получаем, что оптимальным в рассматриваемом примере является (номера заявок - новые) расписание

$\rho=(\rho_1, \rho_2)$, где ρ_1 = (3,5) и ρ_2 = (1, 2, 4). Легко проверяется, что при реализации построенного расписания суммарный штраф по всем заявкам действительно равен 68. В заключение переходим к исходным номерам и фиксируем, что оптимальным в рассмотренном примере является расписание ((3,5),(2, 1, 4)).

Если в нижней строке таблицы 2.2 в качестве минимального зафиксировать элемент, стоящий в :6-ом столбце, то в итоге будет построено также оптимальное расписание ((2, 1, 4), ((3,5)).

Общая задача однопроцессорного обслуживания множества заявок с критерием суммарного штрафа. Данная задача – обобщение однопроцессорной задачи мастера. Ее отличие от задачи мастера в том, что для каждой заявки i вместо числовой характеристики $a(i)$ задается монотонно возрастающая (в нестрогом смысле) функция индивидуального штрафа $\varphi_i(t)$. Значение этой функции – величина штрафа по заявке i , если обслуживание этой заявки завершается в момент времени t. При реализации расписания ρ величина штрафа $W_i(\rho)$ по произвольной заявке i оказывается равной $\varphi_i(t^*(\rho,i))$, где $t^*(\rho,i)$ – момент завершения обслуживания заявки i при реализации расписания ρ. Требуется найти расписание, минимизирующее величину суммарного по всем заявкам штрафа.

Сначала выделим частные случаи, в которых правила оптимального упорядочения заявок получаются применением перестановочного приема.

Рассмотрим ситуацию, когда все функции индивидуального штрафа экспоненциальны:

$$\varphi_i(t) = a_i \exp(\alpha t) + b_i, \quad i = \overline{1,n};$$

константа α считается положительной. Реализуя перестановочный прием, легко получаем алгоритм синтеза оптимального расписания: для каждой заявки i, $i = \overline{1,n}$, нужно вычислить показатель

$$\nu(i) = (1 - \exp(\alpha\tau(i))/ a_i;$$

далее заявки следует упорядочить по убыванию значений $\nu(i)$. Полученная перестановка является оптимальным расписанием обслуживания имеющегося множества заявок.

Выделим случай, когда все функции индивидуального штрафа одинаковы: $\varphi_i(t) = \Phi(t), \ i = \overline{1,n}$; здесь $\Phi(t)$ – произвольная монотонно возрастающая (в нестрогом смысле) функция. Перестановочный прием дает следующий алгоритм синтеза оптимального расписания: заявки следует упорядочить по возрастанию показателя $\tau(i)$ – требуемой продолжительности обслуживания; полученная перестановка является оптимальным расписанием.

Перестановочный прием срабатывает, когда знак разности между величинами суммарного штрафа для расписания $\rho^0 = (1, 2,. . ., n)$ и для расисанияρ^q, получаемого из ρ^0 переменой местами заявок q и $q+1$, зависит только от параметров переставляемых заявок. Теорема Кладова- Лившица [23] устанавливает, что перестановочный прием дает способ определения показателя, сортировка по которому приводит к оптимальному расписанию, только для трех случаев:

1) все функции индививидуального штрафа линейны:

$$\varphi_i(t) = a_i t + b_i, \quad i = \overline{1,n};$$

2) все функции индивидуального штрафа экспоненциальны:

$$\varphi_i(t) = a_i \exp(\alpha t) + b_i, \quad i = \overline{1,n} .$$

3) функции индивидуального одинаковы для всех заявок: $\varphi_i(t) = \Phi(t), \ i = \overline{1,n}$; здесь $\Phi(t)$ – произвольная монотонно возрастающая (в нестрогом смысле) функция.

Теорема Кладова – Лившица доказана в предположении, что функции индивидуального штрафа являются достаточно гладкими (существуют третьи производные).

Отметим, что в трех перечисленных случаях вычислительная сложность алгоритма построения оптимального расписания (число выполняемых алгоритмом элементарных операций) имеет порядок *nlgn*, это соответствует сложности процедуры сортировки.

Если исходные данные определяют задачу, не подпадающую под какой-либо из перечисленных в теореме Кладова – Лившица случаев, то целесообразным способом ее решения является метод динамического программирования.

Рассмотрим обозначаемую символом Z, задачу обслуживания множества заявок $N = \{1,2,\ldots,n\}$ в однопроцессорной системе, в которой единственное ограничение, налагаемое на функции индивидуального штрафа – их монотонное возрастание. Считаем, что для каждой заявки i, $i = \overline{1,n}$, известны: $\tau(i)$ – продолжительность обслуживания на процессоре и $\varphi_i(t)$ – монотонно возрастающая в нестрогом смысле функция индивидуального штрафа; требуется построить расписание обслуживания, обеспечивающее минимальное значение суммарного по всем заявкам штрафа. Пусть S – произвольное подмножество заявок, $S \subseteq \{1,2,\ldots,n\}$. Через $Z(S)$ обозначим частную задачу, получаемую из Z в предположении, что обслуживанию подлежит только совокупность заявок S, а через $K(S)$ - минимально возможный суммарный штраф в частной задаче $Z(S)$. Для одноэлементного множества заявок, как очевидно, имеем:

$$K(\{i\}) = \varphi_i(\tau(i)), \ i \in N. \quad (2.9)$$

Сейчас предположим, что все значения функции $K(S)$ для p-элементных множеств S уже найдены, здесь $p \in \{1,2,\ldots,n-1\}$. Пусть S^* - произвольное $(p+1)$-элементное множество заявок. Если считать, что в совокупности S^* последней обслуживается заявка j, то величина индивидуального штрафа по этой заявке равна $\varphi_j(\sum_{\alpha \in S^*} \tau(\alpha))$, минимально возможный суммарный по всем заявкам данной совокупности штраф в имеющейся ситуации равен $K(S^* \setminus \{j\}) + \varphi_j(\sum_{\alpha \in S^*} \tau(\alpha))$.

Получаем:

$$K(S^*) = \min_{j \in S^*} \{ K(S^* \setminus \{j\}) + \varphi_j(\sum_{\alpha \in S^*} \tau(\alpha)) \}, \quad (2.10)$$

здесь S^* - произвольное подмножество заявок, состоящее не менее, чем из двух элементов. Если минимум правой части (2.10) достигается при $j = q$, то последней в подмножестве S^* следует обслужить заявку q. Равенства (2.9) – (2.10) - рекуррентные соотношения динамического программирования, позволяющие решить задачу Z. Подсчитывая на их основании значения функции $K(S^*)$, вычисления выполняются в порядке роста числа элементов в множествах S^*, мы в итоге находим величину $K(N)$,

т.е. оптимальное значение критерия в решаемой задаче (минимальную величину суммарного штрафа).

ПРИМЕР 2.2. Имеется процессор, обслуживанию которым подлежат заявки множества $N = \{1,2,3\}$. Известно, что $\tau(1) = 3;\ \tau(2) = 2;\ \tau(3) = 4;\ \varphi_1(t) = 2t^2;\ \varphi_2(t) = 3t;\ \varphi_3(t) = t^2 + t$. Требуется построить расписание, минимизирующее величину суммарного штрафа.

По формуле (2.9) вычисляем:

$K(\{1\}) = \varphi_1(3) = 18;\ K(\{2\}) = \varphi_2(2) = 6;\ K(\{3\}) = \varphi_3(4) = 20.$

Далее, пользуясь формулой (2.10), находим последовательно:

$K(\{1, \underline{2}\}) = \min\{K(\{2\}) + \varphi_1(5),\ K(\{1\}) + \varphi_2(5)\} = \min\{6 + 50,\ 18 + 15\} = 33;$

$K(\{1, \underline{3}\}) = \min\{K(\{3\}) + \varphi_1(5),\ K(\{1\}) + \varphi_3(7)\} = \min\{20 + 98,\ 18 + 56\} = 74;$

$K(\{\underline{2}, 3\}) = \min\{K(\{3\}) + \varphi_2(6),\ K(\{1\}) + \varphi_2(5)\} = \min\{20 + 18,\ 6 + 42\} = 38;$

$K(\{\underline{1}, 2, 3\}) = \min\{K(\{2,3\}) + \varphi_1(9),\ K(\{1,3\}) + \varphi_2(9),\ K(\{1, 2\}) + \varphi_3(9)\} = \min\{38 + 162,\ 74 + 27,\ 33 + 90\} = 101;$

в левых частях записанных конкретизаций равенства (2.10) подчеркнуты значения параметра j, обеспечивающие минимальные значения их правых частей. Оптимальное значение критерия в рассматриваемом примере равно 101.

Благодаря сделанным в процессе вычислений подчеркиваниям, легко определяем оптимальную последовательность обслуживания заявок. Действительно, в совокупности $\{\underline{1}, 2, 3\}$ последней надо обслужить заявку 2 (именно этот номер был подчеркнут при определении $K(\{\underline{1}, 2, 3\})$). Заявке 2, таким образом, предшествует пара заявок $\{1,3\}$. В совокупности $\{1,3\}$ последней надо обслужить заявку 3 (номер 3 был подчеркнут при определении $K(\{\underline{1}, 3\})$). И, наконец, заявка 1 должна быть обслужена первой. Легко подсчитывается, что при реализации построенного расписания величина суммарного штрафа действительно равна 101.

Задача однопроцессорного обслуживания множества заявок с минимаксным критерием. Имеется множество заявок $N = \{1,2,\ldots,n\}$ и процессор $П$, на котором должна быть обслужена каждая заявка этого множества; процессор не может обслуживать несколько (более одной) заявок одновременно; переход от обслуживания одной заявки к обслуживанию следующей за ней заявки затрат времени не требует. Обслуживание заявок начинается с момента времени $t = 0$. Для каждой заявки i известны: $\tau(i)$ – продолжительность обслуживания на процессоре; монотонно возрастающая (в нестрогом смысле) функция индивидуального штрафа $\varphi_i(t)$. Если обслуживание заявки i завершается в момент времени t, то $\varphi_i(t)$ – величина штрафа по этой заявке. При реализации расписания ρ моменты завершения обслуживания заявок вычисляются по формулам (2.1) – (2.3), величина штрафа $W_i(\rho)$ по произвольной заявке i оказывается равной $\varphi_i(t^*(\rho,i))$, $i = \overline{1,n}$. Вводим критерий

$$K(\rho) = \max_i \varphi_i(t^*(\rho,i)).$$

Проблема заключается в отыскании расписания, минимизирующего значение максимального из индивидуальных штрафов:

$$\min_\rho K(\rho). \qquad (2.11)$$

Очевидно, что обслуживание процессором заявок множества N завершается в момент времени $T = \tau(1) + \tau(2) + \ldots + \tau(n)$. Подсчитаем все значения $\varphi_i(T)$, $i = \overline{1,n}$; пусть $\varphi_k(T)$ – минимальная из найденных величин. Покажем, что в таком случае имеется оптимальное для задачи (2.8) расписание, в котором заявка k обслуживается последней. Предположим, что в некотором расписании ρ заявка k обслуживается j- ой по очереди ($j \in \{1,2,\ldots, n-1\}$), а последней обслуживается заявка p, причем $\varphi_k(T) < \varphi_p(T)$. Очевидно, что $K(\rho) \geq \varphi_p(T) > \varphi_k(T)$. Последовательными перестановками заявки k и каждой непосредственно следующей за ней по обслуживанию заявки от расписания ρ переходим к расписанию ρ', в котором заявка k обслуживается предпоследней. По расписанию ρ' каждая из заявок множества $N\backslash\{k,p\}$ завершается обслуживанием раньше, чем по расписанию ρ; заявка p завершается обслуживанием в тот же момент времени T. Заявка k по расписанию ρ' завершается обслуживанием в момент $T - \tau(p)$, но $\varphi_k(T - \tau(p)) \leq \varphi_k(T) < \varphi_p(T)$. Из изложенного вытекает, что $K(\rho') \leq K(\rho)$. От расписания ρ' перестановкой местами заявок k и p перейдем к расписанию ρ^*, в котором заявка k обслуживается последней. По расписанию ρ^* каждая из заявок множества $N\backslash\{k,p\}$ завершается обслуживанием в тот же момент, что и по расписанию ρ'; заявка p завершается обслуживанием раньше, чем по расписанию ρ'. Заявка k по расписанию ρ^* завершается обслуживанием в момент T, но $\varphi_k(T) < \varphi_p(T)$. Из изложенного вытекает, что $K(\rho^*) \leq K(\rho')$. Мы показали, что в рассматриваемой задаче однопроцессорного обслуживания всегда имеется оптимальное расписание, в котором заявка с минимальным значением индивидуального штрафа в момент времени T обслуживается последней. Отсюда вытекает следующий алгоритм синтеза оптимального расписания $\rho^{**} = (i_1, i_2, \ldots, i_n)$:

1. Полагаем $N = \{1,2,\ldots,n\}$; $T = \tau(1) + \tau(2) + \ldots + \tau(n)$; $r=n$;

2. Для каждой заявки i из N находим значение $\varphi_i(T)$; пусть $\varphi_k(T)$ – минимальное из найденных значений (если $\varphi_i(T)$ минимально при нескольких значениях индекса i, в качестве k берем любое из них);

3. Полагаем $i_r = k$;

4. Изымаем из множества N заявку k; значение T уменьшаем на $\tau(k)$; значение параметра r уменьшаем на единицу;

5. Если $r = 0$, искомое расписание построено; в противном случае переходим к п.2. алгоритма.

Задача однопроцессорного обслуживания множества заявок при наличии директивных сроков. Считается заданным множество заявок $\{1,2,\ldots,n\}$ и процессор $П$, на котором должна быть обслужена каждая заявка; процессор не может обслуживать несколько (т.е. более одной) заявок одновременно; переход от обслуживания одной заявки к обслуживанию следующей за ней заявки затрат времени не требует. Обслуживание заявок начинается от момента времени $t = 0$. Для каждой заявки i известны две характеристики: $\tau(i)$ – продолжительность ее обслуживания на процессоре; $d(i)$ – директивный срок, не позднее которого обслуживание заявки должно завершиться; здесь $\tau(i) > 0$, $d(i) > 0$, $i = \overline{1,n}$. Расписание обслуживания отождествляем с перестановкой $\rho = (i_1, i_2, \ldots, i_n)$ элементов множества $\{1, 2, \ldots, n\}$; заявка i_k в расписании ρ обслуживается k-ой по очереди. Расписание ρ для каждой заявки i однозначно определяет момент завершения ее обслуживания $t^*(\rho, i)$. Считаем, что заявка i в расписании ρ обслуживается с соблюдением директивного срока, если $t^*(\rho, i) \leq d(i)$.

Проблема заключается в построении расписания, в котором для наибольшего числа заявок директивные сроки соблюдаются.

Сначала рассмотрим частный случай, в котором директивный срок одинаков для всех заявок: $d(1) = d(2) = \ldots = d(n) = D$. Решаемая задача в данном частном случае путем дополнительного введения одной и той же для всех заявок функции индивидуального штрафа $\Phi(t)$ сводится к рассмотренной выше задаче однопроцессорного обслуживания множества заявок с критерием суммарного штрафа. Функция $\Phi(t)$ вводится следующим образом:

$$\Phi(t) = \begin{cases} 1, & \text{если } t > D; \\ 0, & \text{в противном случае.} \end{cases}$$

Благодаря способу задания функции $\Phi(t)$, при реализации произвольного расписания ρ суммарный по всем заявкам штраф оказывается равным числу заявок, которые по этому расписанию обслуживаются с нарушением директивного срока D. Решая задачу минимизации суммарного штрафа, мы фактически строим расписание, в котором для наибольшего числа заявок директивный срок соблюдается. Так как функция индивидуального штрафа для всех заявок одна и та же, оптимальное расписание строится просто: заявки следует обслуживать в порядке возрастания характеристики $\tau(i)$.

Сейчас задачу обслуживания множества заявок при наличии директивных сроков рассмотрим в изложенной выше общей постановке. Систему директивных сроков $\{d(1), d(2),\ldots, d(n)\}$ именуем реальной, если существует расписание обслуживания ρ такое, что для каждого $i = \overline{1,n}$ выполняется неравенство $t^*(\rho, i) \le d(i)$. Пусть $(i_1, i_2, \ldots, i_n)$ – перестановка, перечисляющая заявки в порядке возрастания директивных сроков. Очевидно, что указанная система $\{d(1), d(2),\ldots, d(n)\}$ реальна тогда и только тогда, когда расписание $\rho=(i_1, i_2,\ldots, i_n)$ обеспечивает соблюдение каждого из указанных для заявок сроков.

Синтез расписания, в котором для наибольшего числа заявок директивные сроки соблюдаются, осуществляется следующим образом.

Строим перестановку $(i_1, i_2,\ldots, i_n)$, перечисляющую заявки в порядке возрастания директивных сроков. Если при реализации расписания $\rho^*=(i_1, i_2,\ldots, i_n)$ каждая заявка обслуживается с соблюдением директивных сроков, то расписание ρ^* оптимально. В случае, когда при реализации расписания ρ^* оказывается нарушенным только один из директивных сроков, это расписание также оптимально.

В остальных случаях действуем по следующему алгоритму :

1) полагаем $\rho = \rho^*$; множество W считаем пустым (далее в множество W будут вноситься заявки, которые по конструируемому расписанию обслуживаются с нарушением директивных сроков);

2) последовательность ρ трактуем как расписание обслуживания входящих в нее заявок;

3) если все входящие в ρ заявки обслуживаются с соблюдением директивных сроков или если последовательность ρ оказалась пустой, то переходим к п. 8; в противном случае – к п.4.

4) в последовательности ρ находим минимальное q такое, что заявка i_q по расписанию ρ обслуживается с нарушением директивного срока;

5) в совокупности заявок $\{i_1, i_2,..., i_q\}$ находим заявку i_x, продолжительность обслуживания которой максимальна.

6) новую последовательность ρ получаем из предшествующей изъятием из нее заявки i_x (все остальные заявки перечисляются в прежнем порядке); Заявку i_x включаем в множество W.

7) переходим к п. 2;

8) искомое оптимальное расписание ρ^{**} определяем как последовательность, состоящую из двух частей: последовательность ρ (начальная часть расписания ρ^{**}); перечень элементов множества W в любом порядке (заключительная часть расписания ρ^{**}).

Обоснование изложенного алгоритма выполняется рассуждениями от противного.

Легко определяется, что оценка временной вычислительной сложности алгоритма Cn^2, где С - некоторая не зависящая от n константа.

Задача Джонсона для n станков (ЗД-n). Имеется комплект K, состоящий из деталей $D_1, D_2,..., D_m$; все детали должны быть обработаны на станках $S_1, S_2,...,S_n$. Каждая деталь в процессе обработки должна пройти все станки последовательно, в порядке роста их номеров (индексов). Для каждой детали D_i задано требуемое время τ_{ij} ее обработки на станке S_j; считаем, что все τ_{ij} - целые положительные числа, $i = \overline{1,m}$, $j = \overline{1,n}$. В любой момент времени каждый станок может обрабатывать только одну деталь. Работа каждого станка над любой деталью выполняется без прерываний. Работа над комплектом начинается от момента $t=0$, по состоянию на этот момент все станки считаются свободными. Расписание в *ЗД- n* определяем как кортеж $\boldsymbol{\rho} = \{\rho_1, \rho_2,..., \rho_n\}$, где ρ_j - перестановка элементов множества $\{1, 2,..., m\}$ номеров деталей, определяющая последовательность их обработки на станке S_j, $j = \overline{1,n}$. Требуется найти расписание, при реализации которого момент готовности комплекта (т.е. момент завершения работы станка S_n) оказывается минимальным. Джонсон доказал, что в любой задаче описанного типа всегда имеется оптимальное расписание $\boldsymbol{\rho}$, в котором одновременно $\rho_1 = \rho_2$ и $\rho_{n-1} = \rho_n$, т.е. последовательности обработки деталей на первом и втором станках совпадают и последовательности обработки деталей на предпоследнем и последнем станках также совпадают. Отсюда вытекает, что в задачах Джонсона для двух и трех станков всегда имеются оптимальные расписания, в которых последовательности обработки деталей на всех станках одинаковы. Легко строится пример задачи Джонсона для четырех станков, в которой имеется единственное оптимальное расписание и оно предусматривает, что первые два станка проходятся деталями в одной последовательности, а два следующих станка – в последовательности, отличающейся от предыдущей.

Для задач Джонсона с двумя или тремя станками оптимальное расписание конструируется как перестановка ρ элементов множества $\{1, 2, ... , m\}$, определяющая единую последовательность обработки деталей на всех станках. Синтез оптимального расписания может быть выполнен на основе соответствующим образом построенных соотношений динамического программирования. При этом запись и анализ уравнения динамического программирования для задачи Джонсона с двумя станками дает возможность простого и быстрого определения оптимального расписания.

Задача Джонсона для двух станков (ЗД-2). Для каждой детали D_i считаем, что a_i – требуемое время ее обработки на станке S_1, а b_i – требуемое время ее обработки на станке S_2, $i = \overline{1,m}$. Момент принятия решения (МПР) – это любой момент, когда станок S_1 свободен и требуется определить следующую деталь, которая начиная от данного МПР на нем будет обрабатываться. Каждый МПР характеризуется следующими факторами: M – совокупность деталей, обработка которых пока не начиналась; T_2 – продолжительность отрезка времени, на котором позднее рассматриваемого МПР станок S_2 еще будет загружен обработкой деталей, уже прошедших станок S_1. Через $F(M,T_2)$ обозначим минимально возможную продолжительность отрезка времени от рассматриваемого МПР до момента завершения изготовления комплекта деталей K. Как очевидно, $F(\{D_1, D_2,..., D_m\}, 0)$ - минимальное время, за которое комплект может быть изготовлен. Предположим, что в МПР, характеризуемый парой (M, T_2), на обработку станком следует направить деталь D_x, а в следующий МПР - деталь D_y ($D_x \in S$, $D_y \in S$). В таком случае сначала имеем:

$$F(M, T_2) = a_x + F(M \setminus \{ D_x \}, b_x + max [(T_2 - a_x), 0]).$$

Учитывая сделанное предположение о направлении на обработку вслед за деталью D_x детали D_y, получаем:

$$F(M \setminus \{ D_x \}, b_x + max [(T_2 - a_x), 0]) = a_y + F(S \setminus \{ D_x , D_y \}, b_y + max[(b_x + max [(T_2 - a_x), 0] - a_y), 0]).$$

Далее имеем:

$$F(M, T_2) = a_x + a_y + F(S \setminus \{ D_x , D_y \}, K_{xy}),$$

где

$$K_{xy} = b_y + max[(b_x + max [(T_2 - a_x), 0] - a_y), 0]) = b_y + b_x - a_y + max \{max[(T_2 - a_x),0], a_y - b_x \} =$$

$$= b_y + b_x - a_y + max\{ T_2 - a_x , 0, a_y - b_x\} = b_y + b_x - a_y - a_x + max\{ T_2 , a_x , a_x + a_y - b_x\} =$$

$$= b_y + b_x - a_y - a_x + max\{ T_2, max (a_x , a_x + a_y - b_x)\}.$$

Оказывается очевидным, что если

$$max (a_x , a_x + a_y - b_x) > max (a_y , a_x + a_y - b_y),$$

то детали D_x и D_y имеет смысл поменять местами. Полученное правило можно переписать в виде

$$a_x + a_y + max (-a_y , - b_x) > a_x + a_y + max(-a_x , - b_y)$$

или, что еще проще, как

$$max (-a_y , - b_x) > max(-a_x , - b_y).$$

Последнее неравенство эквивалентно следующему:

$$min(b_x, a_y) > min(b_y, a_x).$$

Так получаем следующий алгоритм определения оптимального в *ЗД-2* расписания (последовательности обработки деталей). Считаем, что исходная информация по задаче записана в виде матрицы $\boldsymbol{A}$ размера mx2, в первом столбце которой последовательно, сверху вниз, перечислены времена обработки деталей на первом станке $a_1, a_2 ,..., a_m$; а во втором столбце также последовательно, сверху вниз, перечислены времена обработки деталей на втором станке $b_1, b_2,..., b_m$. Для записи

оптимального расписания выделяем набор W из m изначально пустых, идущих слева направо позиций. Далее руководствуемся следующей инструкцией:

1.Выделить наименьший непомеченный элемент матрицы $\boldsymbol{A}$ (изначально все элементы матрицы $\boldsymbol{A}$ считаются непомеченными);

2. Если выделен некоторый элемент a_i левого столбца, вписать D_i в крайне левую пустую позицию набора W; Если выделен некоторый элемент b_i правого столбца, вписать D_i в крайне правую пустую позицию набора W;

3. Если в наборе W пустых позиций нет, перейти к п.5; в противном случае - к п.4.

4. В матрице пометить оба элемента i- ой строки; перейти к п. 1.

5. В позициях W зафиксировано оптимальное расписание. Запуск деталей в обработку должен осуществляться в порядке их перечисления (слева направо) в позициях набора W.

Отметим, что выполнение приведенной инструкции не всегда является однозначно определенной процедурой, но любая реализация инструкции строит оптимальное расписание (оптимальное для *ЗД- 2* расписание, вообще говоря, не единственно).

ПРИМЕР 2.3. Задача Джонсона для двух станков определена матрицей

$$\begin{pmatrix} 2 & 4 \\ 5 & 6 \\ 6 & 1 \\ 7 & 4 \\ 3 & 7 \\ 8 & 2 \\ 5 & 3 \end{pmatrix}$$

Требуется найти оптимальную последовательность обработки деталей.

В синтезируемом по приведенной инструкции оптимальном расписании последней (минимальным элементом матрицы является единица, стоящая в третьей строке правого столбца) должна обрабатываться деталь D_3, она записывается в крайне правую позицию набора W. Далее в крайне левую и крайне правую свободные позиции (минимальными непомеченными элементами матрицы оказываются две двойки, в левом и правом столбцах) вписываются детали D_1 и D_6 соответственно. Затем в крайнюю левую и крайнюю правую свободные позиции (минимальными непомеченными элементами матрицы оказываются две тройки, в левом и правом столбцах) вписываются детали D_5 и D_7 соответственно. Далее минимальным непомеченным элементом матрицы оказывается четверка в правом столбце, соответствующая деталь D_4 вписывается в крайне правую свободную позицию набора W. Последней на оставшееся свободным место вносится деталь D_2. Таким образом устанавливается, что оптимальной является следующая последовательность обработки деталей: $D_1, D_5, D_2, D_4, D_7, D_6, D_3$.

2.2. Однопроцессорные задачи синтеза расписаний обслуживания конечных детерминированных потоков заявок.

Каноническая задача однопроцессорного обслуживания потока заявок. Однопроцессорная система должна обслужить поток $R = \{1, 2,..., n\}$ поступающих в дискретном времени заявок. Каждая заявка i характеризуется тремя целочисленными параметрами: $t(i)$ – момент поступления в систему обслуживания (считается, что $0 = t(1) \le t(2) \le ... \le t(n)$); $\tau(i)$ – требуемая продолжительность (число тактов дискретного времени) обслуживания процессором; $a(i)$ – штраф за единицу времени (такт) пребывания заявки в системе обслуживания, здесь $i = \overline{1,n}$. Если обслуживание произвольной заявки i завершается в момент времени $t^*(i)$, то величина индивидуального штрафа по этой заявке считается равной $a(i)\cdot[t^*(i) - t(i)]$; таким образом, функция, определяющая величину индивидуального штрафа, линейна относительно времени пребывания заявки в системе обслуживания. Полагаем, что процессор $П$ готов к обслуживанию заявок потока начиная от момента времени $t = 0$. Обслуживание каждой заявки можно выполнять с любого момента, начиная от момента ее поступления в систему. Обслуживание каждой заявки реализуется без прерываний, одновременное обслуживание процессором нескольких (более одной) заявок невозможно.

Расписание обслуживания отождествляем с перестановкой $\rho = (i_1, i_2,..., i_n)$ элементов множества $\{1, 2,..., n\}$; заявка i_k в расписании ρ обслуживается k-ой по очереди, $k = \overline{1,n}$.. Обозначим через $t_{нач}(\rho, i_k)$ и $t^*(\rho, i_k)$ моменты начала и завершения обслуживания заявки i_k при реализации расписания ρ, $k = \overline{1,n}$. Считаем, что реализация расписания компактна, т.е. имеют место соотношения:

$$t_{нач}(\rho, i_1) = t(i_1) \qquad (2.12)$$

$$t_{нач}(\rho, i_k) = max\,[t^*(\rho, i_{k-1}), t(i_k)],\ k=2, 3,..., n; \quad (2.13)$$

$$t^*(\rho, i_k) = t_{нач}(\rho, i_k) + \tau(i_k),\ k=1, 2,..., n. \quad (2.14)$$

Суммарный штраф по всем заявкам потока R, обслуживаемым согласно расписанию ρ, есть

$$W(\rho) = \sum_{i=1}^{n} a(i)\,[t^*(\rho,i) - t(i)]\ .$$

Проблема заключается в минимизации суммарного штрафа, т.е. в нахождении

$$\min_{\rho} W(\rho)\ . \qquad (2.15)$$

Задачу (2.15) именуем *канонической задачей однопроцессорного обслуживания*. Применим метод динамического программирования для ее решения.

Обозначим через $V(t)$ определяемую исходными данными задачи совокупность заявок, поступающих на обслуживание в момент времени t. При этом $V(0)$ – совокупность заявок, которые по состоянию на момент времени $t=0$ присутствуют в системе и ожидают обслуживания.

Очевидно, что при обслуживании входного потока заявок управленческие решения принимаются для тех моментов дискретного времени, когда процессор свободен; каждое такое решение состоит в определении, какая заявка будет обслуживаться следующей. При этом текущая ситуация вполне характеризуется парой

(t, S), где *t* – момент, для которого принимается решение (сокращенно МПР – момент принятия решения, в этот момент процессор свободен), а *S* – множество заявок, которые по состоянию на момент времени *t* прибыли и ожидают обслуживания. Пары *(t, S)*, где *t* – МПР, а *S* – совокупность заявок, которые по состоянию на момент *t* прибыли и находятся в ожидании обслуживания, будем называть *состояниями системы обслуживания.*

Пусть $\Sigma(t, S)$ обозначает минимальную величину суммарного штрафа по заявкам множества *S* и заявкам, поступающим в систему обслуживания позднее момента времени *t* для определяемой состоянием *(t, S)* ситуации. Как очевидно, $\Sigma(0,V(0))$ – оптимальное значение критерия в решаемой задаче (2.15).

Через $D(t, \mu)$ обозначим совокупность заявок, прибывающих в систему обслуживания на временном отрезке $[t + 1, t + \mu]$. Ясно, что

$$D(t, \mu) = \bigcup_{i=1}^{\mu} V(t + i).$$

Для любого состояния системы обслуживания (*t, S*) множество *S* считаем непустым, что обеспечивается обязательным включением в него фиктивной заявки *0* с параметрами $a(0) = 0$, $t(0) = t$ и $\tau(0) = \min \{\theta \mid D(t, t + \theta) \neq \varnothing\}$. Взятие на обслуживание фиктивной заявки фактически означает простой процессора от момента *t* до ближайшего момента, когда в систему прибудет какая- либо новая заявка потока *R*. Если поступившая в момент *t* фиктивная заявка не принимается на немедленное обслуживание, она выпадает из дальнейшего рассмотрения.

Если в момент времени *t* на немедленное обслуживание процессором отправить заявку *i*, то:

1) величина штрафа по этой заявке окажется равной $a(i) \cdot [t+\tau(i) - t(i)]$;

2) следующим за *t* моментом принятия решения будет $t + \tau(i)$;

3) совокупность заявок, ожидающих обслуживания по состоянию на момент времени $t + \tau(i)$ запишется как $(S \setminus i) \cup D(t, \tau(i))$.

В принятых обозначениях рекуррентное соотношение динамического программирования записывается в виде

$$\Sigma(t, S) = \min_{i \in S} \{a(i) \cdot [t+\tau(i) - t(i)] + \Sigma(t + \tau(i), (S \setminus i) \cup D(t, \tau(i)))\}. \qquad (2.16)$$

Если минимум реализуется при $i = i^*$, то в МПР, характеризуемый парой *(t, S),* на процессор следует направить заявку i^* из совокупности ожидающих обслуживания. Направление на обслуживание фиктивной (нулевой) заявки означает простой процессора по меньшей мере до следующего момента пополнения множества *S* вновь прибывающими заявками. Если в совокупности *S* имеется нефиктивная заявка, которая может быть завершена обслуживанием до следующего момента пополнения множества *S* вновь прибывшими заявками, то начало обслуживания в момент времени *t* фиктивной заявки заведомо нецелесообразно.

Реализации рекуррентной процедуры (2.16) должно предшествовать вычисление величин $\Sigma(t(n), S)$ для всех возможных значений аргумента *S*, $S \subseteq \{1, 2,..., n\}$.

Задача построения оптимального от момента *t(n)* расписания – это задача мастера, рассмотренная в данной главе выше. Построение оптимального от момента *t(n)* расписания выполняется упорядочением не обслуженных по состоянию на этот

момент заявок по убыванию значений показателя $\mu(i) = a(i)/\tau(i)$. При известном расписании каждое значение $\Sigma(t(n), S)$ вычисляется арифметически.

Далее отметим справедливость равенства

$$\Sigma(t(n) + \theta, S) = \Sigma(t(n), S) + (\sum_{i \in S} a(i)) \times \theta \qquad (2.17)$$

для любых $\theta > 0$ и $S \subseteq \{1, 2, \ldots, n\}$. Равенство (2.17) при известных значениях $\Sigma(t(n), S)$ дает возможность элементарным образом вычислять нужные для счета по рекуррентному соотношению (2.16) величины $\Sigma(t(n) + \theta, S)$ для различных множеств S при $\theta > 0$.

В реализации процедуры, определяемой формулой (2.16), последовательно, для всех возможных подмножеств заявок S, вычисляются значения $\Sigma(t(n) - 1, S)$, $\Sigma(t(n) - 2, S)$ и т.д. Процесс счета по соотношению (2.16) заканчивается отысканием $\Sigma(0, V(0))$, т.е. оптимального значения критерия в решаемой задаче.

Дадим верхнюю оценку числа элементарных операций, выполняемых при вычислениях по соотношению (2.16) в процессе решения задачи. Эта оценка зависит от количества состояний (t, S), на которых подсчитываются значения функции $\Sigma(t, S)$, и сложности вычисления каждого отдельного значения этой функции. При подсчетах по формуле (2.16) в рассматриваемых парах (t, S) первый аргумент принимает $t(n)$ различных значений, число возможных значений второго аргумента оцениваем сверху как 2^n (полагаем, что S – произвольное подмножество заявок). Получаем, что верхней оценкой для числа подсчитываемых значений функции $\Sigma(t, S)$ является $t(n) \times 2^n$. Число элементарных операций, выполняемых при подсчете каждого очередного значения функции $\Sigma(t, S)$, по верхней оценке зависит от n линейно. Таким образом, верхней оценкой числа реализуемых алгоритмом элементарных операций является

$$C\, n\, t(n)\, 2^n,$$

где C – не зависящая от n константа.

Приведенная оценка временной вычислительной сложности изложенного алгоритма экспоненциальна. Как будет показано в следующем примере, общий объем вычислений можно существенно сократить за счет достаточно просто выполняемой процедуры определения действительно необходимых наборов (t, S), на которых следует отыскивать значения функции Σ. Вместе с тем, построить алгоритм решения задачи (2.15) с полиномиально зависящей от n оценкой числа выполняемых им элементарных операций в принципе невозможно (рассматриваемая задача относится к числу NP-трудных, см. главу 3 данного пособия).

ПРИМЕР 2.4. Каноническая задача однопроцессорного обслуживания потока из пяти заявок характеризуется следующими данными: $t(1) = 0$, $\tau(1) = 2$, $a(1) = 3$; $t(2)=0$, $\tau(2) = 3$, $a(2) = 5$; $t(3)=1$, $\tau(3) = 1$, $a(3) = 7$; $t(4)=3$, $\tau(4) = 1$, $a(4) = 5$; $t(5)=3$, $\tau(5) = 3$, $a(5) = 3$. Требуется найти оптимальное расписание обслуживания.

Оптимальное значение критерия суммарного штрафа в решаемой задаче равно $\Sigma(0,\{1,2\})$. Из (2.16) определяем, что для подсчета $\Sigma(0,\{1,2\})$ необходимо (с учетом возможности принятия на обслуживание в момент времени 0 фиктивной заявки) предварительно найти значения $\Sigma(1,\{1,2,3\})$, $\Sigma(2,\{2,3\})$, $\Sigma(3,\{1,3,4,5\})$. Для отыскания величины $\Sigma(1,\{1,2,3\})$ необходимо предварительно определить только значение $\Sigma(2,\{1,2\})$; действительно, принятие на обслуживание от момента времени 0 фиктивной заявки могло оказаться целесообразным только в случае, когда далее, от момента

времени *1*, будет проходить обслуживание заявки *3*. Для отыскания величины *Σ(2,{2,3})* необходимо предварительно определить *Σ(5,{3,4,5})* и *Σ(3,{2,4,5})*; фиктивную заявку в состоянии системы (*2,{2,3}*) принимать на обслуживание заведомо нецелесообразно, лучше выбрать из очереди заявку *3* с единичной продолжительностью обслуживания. Для отыскания величины *Σ(2,{1,2})* необходимо предварительно определить *Σ(3,{1,2,4,5}), Σ(4,{2,4,5}) и Σ(5,{1,4,5})*.

Вычислительную процедуру далее проводим следующим образом. Сначала, применяя алгоритм решения задачи мастера и пользуясь формулой (2.17), находим, что *Σ(3,{1,3,4,5}) = 73; Σ(5,{3,4,5}) = 76; Σ(3,{2,4,5})= 61; Σ(3,{1,2,4,5}) = 94; Σ(4,{2,4,5}) =74; Σ(3,{2,4,5})= 61; Σ(5,{1,4,5}) = 63*. Затем определяем нужные значения функции *Σ(t , S)* для значений *t* последовательно равных *2*, *1* и *0*. Начинаем с вычисления *Σ(2,{1,2})* и *Σ(2,{2,3})*. При этом, согласно (2.16), *Σ(2,{1,2}) = min (0 + 94, 12+74, 25 + 63) = 86* (минимум берется по трем суммам, причем первая сумма соответствует выбору фиктивной заявки, вторая сумма - выбору из числа ожидающих заявки *1,* третья - выбору из числа ожидающих заявки *2;* здесь и ниже в каждой сумме первое слагаемое - штраф по выбранной и принимаемой к немедленному исполнению заявке, второе слагаемое - значение *Σ* для следующего состояния системы; после определения минимума правой части соответствующую заявку в записи левой части подчеркиваем). Так как минимум реализуется на второй сумме, в состоянии *(2,{1,2}),* на обслуживание следует направить заявку *1*, в записи *Σ(2,{1,2})* подчеркиваем эту заявку. Затем имеем: *Σ(2,{2,3}) = min(25 + 76, 14 + 61) = 75* (минимум берется по двум суммам, первая сумма соответствует выбору из числа ожидающих обслуживания заявки *2*, вторая сумма - выбору заявки *3*, фиктивная заявка здесь не может быть выбрана по выше описанной причине). Далее определяем *Σ(1,{1,2,3}) = 7+ 86 = 93* (для МПР, характеризуемого парой *(1,{1,2,3})* фактически имеется только один вариант действий - на обслуживание следует направить заявку *3*). В итоге получаем: *Σ(0,{1,2}) = min (93, 6 + 75, 15 + 73) = 81*. Нами определено оптимальное значение критерия в решаемом примере, оно равно *81*.

Так как при подсчете значения *Σ(0,{1,2})* минимум реализуется на второй сумме, которая соответствует выбору заявки *1*, в начальный МПР на обслуживание направляется заявка *1*. Следующим состоянием системы будет *(2,{2,3})*. При подсчете *Σ(2,{2,3})* мы отметили, что минимум реализуется на сумме, соответствующей выбору заявки *3*. Поэтому в оптимальном расписании вслед за заявкой *1* на обслуживание процессором направляется заявка *3*. Следующий МПР характеризуется парой *(3,{2,4,5})*. Так как по состоянию на момент времени *3* все подлежащие обслуживанию заявки уже прибыли, заявки совокупности *{2,4,5}* должны обслуживаться в порядке убывания значений показателя *μ(i) = a(i) / τ(i)*. В итоге получаем оптимальное расписание обслуживания заявок: (*1, 3, 4, 2, 5)*. Данное расписание обеспечивает минимальную, равную *81*, величину суммарного штрафа.

Выполненное решение состоит из трех частей. В первой части сделана так называемая *разметка*, т.е. определены достижимые состояния (состояния, в которых система обслуживания может действительно оказаться). Во второй части с использованием алгоритма решения задачи мастера, формулы (2.17) и рекуррентного соотношения динамического программирования реализован процесс вычисления значений функции *Σ(t, S)* для достижимых состояний. Процедура разметки, как правило, существенно сокращает объем выполняемых вычислений. В третьей части определено оптимальное расписание.

Изложенный алгоритм решения канонической задачи однопроцессорного обслуживания потока заявок использует процедуру обратного счета – значения функции $\Sigma(t, S)$ вычисляются в порядке убывания параметра t. Перейдем к изложению основанного на принципе динамического программирования алгоритма, решающего эту задачу методом прямого счета. Преимуществом алгоритма прямого счета является ненужность предварительного выполнения разметки. Запись соотношений динамического программирования для выполнения прямого счета в явном виде ниже делаться не будет, ибо весьма громоздкими являются описания множеств состояний, которые непосредственно предшествуют рассматриваемым.

Пусть $\Sigma^*(t, S)$ – минимально возможный суммарный штраф по всем обслуженным до момента времени t включительно заявкам, здесь t – произвольный МПР (в момент времени t процессор свободен), а S – совокупность заявок, которые по состоянию на момент t прибыли, но пока не прошли обслуживания. Отметим, что в частности $\Sigma^*(0, V(0))=0$. Стандартными называем записи структуры $<t, S, \Sigma^*(t, S)>$; первые две компоненты стандартной записи – аргументы функции Σ^*, характеризующие состояние системы (момент принятия решения и совокупность прибывших и ожидающих обслуживания заявок), а третья компонента – значение функции Σ^* на данном наборе аргументов. Стандартную запись$< t, S, \Sigma^*(t, S)>$ считаем *терминальной*, если $t \geq t(n)$, соответствующее состояние (t, S) называем *T-состоянием.*

Введем *операцию раскрытия стандартной записи* $\xi = <t, S, \Sigma^*(t, S)>$. Выполняя раскрытие записи ξ, мы для каждой заявки i из S строим расширенную запись, состоящую из следующих шести компонент:

1. $t + \tau(i);$
2. $(S \setminus i) \cup D(t, \tau(i)));$
3. $\Sigma^*(t, S) + a(i)\ (t + \tau(i) - t(i));$
4. $t;$
5. $S;$
6. $i.$

Операцию раскрытия далее будем выполнять только для нетерминальных стандартных записей. Первая и вторая компоненты расширенной записи показывают, в какое очередное состояние перейдет система обслуживания из состояния (t, S) в случае, когда в S для первоочередного обслуживания выбрана заявка i. Третья компонента – суммарный штраф по всем обслуженным к моменту времени $t + \tau(i)$ заявкам в такой ситуации (начальная (t, S)-траектория системы оптимальна, следующий шаг заключается в обслуживании заявки i). Назначение последних трех компонент – указать предыдущее состояние системы и выбранное в нем управление. Общее число расширенных записей, получаемых раскрытием стандартной записи ξ равно числу элементов в множестве S (как всегда, фиктивная заявка считается присутствующей). Расширенную запись именуем терминальной, если ее первая координата имеет значение не меньшее, чем $t(n)$.

Ясно, что если в некоторое состояние система может придти двумя способами, причем первый способ более дешевый, то второй способ надо изъять из рассмотрения. Формализуя и уточняя данное высказывание, введем над списком расширенных записей *операцию усечения*. При выполнении этой операции над списком M мы в случаях, когда в M имеются записи с одинаковым значением второй компоненты $\eta_l =$

$<t^*_1, S^*, C_1, t_1, S_1, i_1>$ и $\eta_2 =< t^*_2, S^*, C_2, t_2, S_2, i_2>$ такие, что $t^*_1 \leq t^*_2$ и $C_1 \leq C_2$ (причем по меньшей мере одно из неравенств – строгое), запись η_2 изымаем; если в записях η_1 и η_2 первые три компоненты соответственно совпадают, изымается любая из этих записей.

Пусть $\eta = < t^*, S^*, C, t, S, i>$ – произвольная расширенная запись. Результатом *свертки* этой записи называем запись $< t^*, S^*, C >$.

Алгоритм прямого счета включает три последовательные процедуры, работающие над списками.

Процедура 1 выполняет следующее:

1.Список записей A полагается состоящим из единственной стандартной записи $<0, V(0), 0>$; списки B, C и D считаются пустыми.

2. Все содержащиеся в списке A стандартные записи из этого списка изымаются и раскрываются, результаты раскрытия пополняют список B.

3. Над списком B выполняется операция усечения.

4. Записи списка B упорядочиваются по возрастанию первой компоненты. Минимальное из значений первых компонент записей списка B фиксируется как k (в процессе работы алгоритма, в результате каждой следующей реализации п. 4 значение k возрастает);

5. Если $k < t(n)$, реализуется переход к п.6; в противном случае – к п. 8.

6. Все записи списка B с минимальным значением первой компоненты из этого списка изымаются и переносятся в список C; одновременно с переносом каждой изъятой из списка B записи в список C, эта запись в свернутом виде переносится в список A.

7. Переход к выполнению п. 1.

8. Процедура работу заканчивает.

В результате работы процедуры 1 мы получаем итоговый список B, содержащий ряд расширенных терминальных записей. Каждая такая запись первыми двумя компонентами задает некоторое *первичное T-состояние*, т.е. T-состояние, для которого непосредственно предшествующее состояние системы, оно указано компонентами 4-5 той же записи, T-состоянием не является. В совокупности полученные расширенные терминальные записи итогового списка B своими первыми двумя компонентами перечисляют все первичные T-состояния X системы за исключением заведомо нецелесообразных. Каждое состояние X содержится только в одной записи итогового списка B, назовем эту запись η_x. Существенно, что третья компонента записи η_x равна минимально возможному суммарному штрафу по заявкам, обслуженным в процессе перехода системы из начального состояния в состояние X.

Процедура 2 выполняет для каждой расширенной терминальной записи итогового списка B *операцию дополнения*. В применении к произвольной записи $\eta = <t, S, C, t_1, S_1, i>$ итогового списка эта операция предусматривает следующие действия. Заявки множества S (другие заявки в систему уже не прибудут, ибо $t \geq t(n)$) в соответствии с алгоритмом решения задачи мастера упорядочиваются по убыванию показателя $\mu(i) = a(i)/\tau(i)$. Получаемую последовательность обозначим $\rho(S)$. Считается, что расписание $\rho(S)$ обслуживания процессором заявок множества S реализуется начиная от момента времени t. Для каждой заявки множества S определяется момент

завершения ее обслуживания и соответствующая величина индивидуального штрафа. Сумму индивидуальных штрафов по всем заявкам множества S обозначим $U(t, S)$. Результатом применения операции дополнения к записи $\eta = < t, S, C, t_1, S_1, i>$ является состоящая из семи компонент запись $\eta' = < t, S, \rho(S), C + U(t, S)\}., t_1, S_1, i>$. Далее каждая дополненная указанным образом запись из списка B переносится в список D. В списке D отыскивается запись с минимальным значением четвертой координаты. Найденная запись, пусть это $(\eta')^* = < t^*, S^*, \rho(S^*), \Sigma^*, t^*_1, S^*_1, i_1^*>$, – итог работы процедуры 2. Четвертая координата найденной записи – оптимальное значение критерия в задаче (2.15).

Процедура 3 заключается в построении расписания ρ_{opt}, обеспечивающего полученное оптимальное значение критерия Σ^*. Указанное расписание состоит из начальной части ρ_1 и заключительной части ρ_2. При этом $\rho_2. = \rho(S^*)$, т.е. это третья компонента записи $(\eta')^*$, найденной в итоге работы процедуры 2. В расписании ρ_1 на последнем месте должна стоять заявка i_1^*. Согласно записи $(\eta')^*$, эта заявка начинается обслуживанием, когда система находится в состоянии (t^*_1, S^*_1). Отыскиваем в списке C запись с первой компонентой t^*_1, и второй компонентой S^*_1; указанному требованию удовлетворяет ровно одна запись списка C. Пусть эта запись своими четвертой, пятой и шестой компонентами имеет соответственно. t^*_2, S^*_2, i_2^*. Тогда в расписании ρ_1 на предпоследнем месте должна стоять заявка i_2^*. Далее отыскиваем в списке C запись с первой компонентой t^*_2, и второй компонентой S^*_2; указанному требованию удовлетворяет ровно одна запись списка C. Пусть эта запись своими четвертой, пятой и шестой компонентами имеет соответственно. t^*_3, S^*_3, i_3^*. Тогда в расписании ρ_1 заявке i_2^* должна непосредственно предшествовать заявка i_3^*. Действуя идентичным образом дальше, реализуем полное построение искомого расписания. Отметим, что последняя найденная в списке C запись своими четвертой и пятой компонентами должна иметь 0 и $V(0))$, что соответствует начальному состоянию системы.

Задача однопроцессорного обслуживания потока заявок с учетом времен переналадок. Данная задача отличается от канонической тем, что при переходе процессора, завершившего обслуживание заявки i, к обслуживанию заявки j необходимо выполнение соответствующей переналадки продолжительностью $p(i,j)$. Считается, что в момент времени 0 процессор находится в нейтральном (нулевом) состоянии и продолжительность его наладки для обслуживания заявки i равна $p(0, i)$. Все числа $p(i,j)$, где $i = \overline{0,n}$ и $j = \overline{1,n}$, предполагаются заданными в виде целочисленной матрицы времен переналадок $P = \{p_{ij}\}$ соответствующей размерности. Условно будем говорить, что процессор, завершивший обслуживание заявки i, находится в конфигурации i.

Расписание обслуживания отождествляем с перестановкой $\rho = (i_1, i_2, \ldots, i_n)$ элементов множества $\{1, 2, \ldots, n\}$; заявка i_k в расписании ρ обслуживается k-ой по очереди. Обозначим через $t_{нач}(\rho, i_k)$ и $t^*(\rho, i_k)$ моменты начала и завершения обслуживания заявки i_k при реализации расписания ρ, $k = \overline{1,n}$. Считаем, что реализация расписания компактна, т.е. имеют место соотношения:

$$t_{нач}(\rho, i_1) = \max \{t(i_1), p(0, i_1)\};$$

$$t_{нач}(\rho, i_k) = \max \{t^*(\rho, i_{k-1}) + p(i_{k-1}, i_k), t(i_k)\}, \; k=2, 3, \ldots, n ;$$

$$t^*(\rho, i_k) = t_{нач}(\rho, i_k) + \tau(i_k), \quad k=1, 2, \ldots, n .$$

Суммарный штраф по всем заявкам потока R, обслуживаемым по расписанию ρ, есть

$$W'(\rho) = \sum_{i=1}^{n} a(i) \times [t^*(\rho,i) - t(i)] . \quad (2.18)$$

Проблема заключается в минимизации суммарного штрафа, т.е. в нахождении

$$\min_{\rho} W'(\rho) . \quad (2.19)$$

Состояние системы обслуживания определяем как тройку *(t, i, S),* где *t* – момент принятия очередного решения по загрузке процессора; в момент *t* процессор считается свободным и находящимся в конфигурации *i*; *S* – множество заявок, которые по состоянию на момент времени *t* прибыли и ожидают обслуживания.

Пусть *Σ(t, i, S)* обозначает минимальную величину суммарного штрафа по заявкам множества *S* и заявкам, поступающим в систему обслуживания позднее момента времени *t,* для системы, находящейся в состоянии *(t, i, S)*. Как очевидно, *Σ(0,0, V(0))* – оптимальное значение критерия в решаемой задаче (2.19).

Как и ранее, через *D(t, μ)* обозначаем совокупность заявок, прибывающих в систему обслуживания на временном отрезке $[t + 1, t + \mu]$.

Существенно, что в состоянии *(t, i, S)* в качестве очередной обслуживаемой может быть выбрана заявка не обязательно из числа имеющихся в совокупности *S* (как это было ранее); она может быть взята из совокупности поступающих во время реализации соответствующей переналадки (при этом для времени переналадки берется верхняя оценка $\max_j p(i,j)$). Для учета новой возможности введем в рассмотрение множество *S′,* состоящее из заявок, поступающих во временном промежутке $[t+1, t + \max_j p(i,j)]$. Выбор очередной обслуживаемой заявки в ситуации, описываемой тройкой (*t, i, S*), будем осуществлять в совокупности $S^* = S \cup S'$.

Кроме того, в множество S* включаем нулевую (фиктивную) заявку. При этом считаем, что принятие на обслуживание фиктивной заявки не предусматривает предварительной переналадки процессора и означает его простой в течение одного такта.

Если в момент принятия решения *t* в совокупности *S** выбрана заявка *j*, то ее обслуживание начнется в момент $\max\{t + p(i,j), t(j)\}$ и завершится в момент $T(t,i,j) = \max\{t + p(i,j), t(j)\} + \tau(j)$.

В принятых обозначениях основное рекуррентное соотношение динамического программирования для рассматриваемой задачи записывается в виде

$$\Sigma(t, i, S) = \min_{j \in S^*} \{ a(j) \times [T(t,i,j) - t(j)] + \Sigma(T(t,i,j), j, (S \cup D(t, T(t,i,j) - t)) \setminus j) \} . \quad (2.20)$$

<u>*Задача однопроцессорного обслуживания потока заявок с нелинейными функциями индивидуальных штрафов*</u>. Минимизируемым в данной задаче критерием считаем суммарный штраф по всем заявкам. Вместо показателя *a(i),* как это имело место в канонической модели, здесь каждой заявке *i* приписывается, вообще говоря, нелинейная функция индивидуального штрафа $\varphi_i(t)$. Если обслуживание заявки *i* завершается в момент времени *t*,* то величина выплачиваемого по данной заявке штрафа равна $\varphi_i(t^*)$, $i = \overline{1,n}$. Все функции $\varphi_i(t)$ считаются монотонно возрастающими в нестрогом смысле. Полагаем, что эти функции заданы программно, вычисление

значения каждой из них в произвольной точке t требует выполнения не более чем k элементарных операций, где k – фиксированная константа. Как и ранее, состояния системы обслуживания – пары (t, S), где t – МПР, а S – множество заявок, которые по состоянию на момент времени t прибыли и ожидают обслуживания; фиктивная заявка в совокупности S присутствует. Пусть $\Sigma_{нл}(t, S)$ обозначает минимальную величину суммарного штрафа по заявкам множества S и заявкам, поступающим в систему позднее момента t, для системы, находящейся в состоянии (t, S). Тогда общее рекуррентное соотношение динамического программирования для решения рассматриваемой задачи записывается в виде

$$\Sigma_{нл}(t,S) = \min_{i \in S} \{\varphi_i(t+ \tau(i)) + \Sigma_{нл}(t + \tau(i),(S \setminus i) \cup D(t, \tau(i))\}. \qquad (2.21)$$

Задача однопроцессорного обслуживания потока заявок с нелинейными функциями индивидуальных штрафов и директивными сроками. Здесь в дополнение к исходным данным предшествующей задачи для каждой заявки i считается указанным обозначаемый через $d(i)$ директивный срок завершения обслуживания, $i = \overline{1,n}$. Расписание ρ именуем допустимым, если для всех $i = \overline{1,n}$ имеет место $t^*(\rho, i) \le d(i)$. Проблема заключается в отыскании допустимого расписания, минимизирующего величину суммарного штрафа.

Пусть $L = \sum_{i=1}^{n} \varphi_i(d(i))$; таким образом, L – оценка сверху величины минимизируемого критерия (возможными считаются только допустимые расписания). Рассматриваемую задачу модифицируем следующим образом: исключим необходимость соблюдения директивных сроков, но введем новые функции индивидуального штрафа: величину штрафа $\varphi'_i(t)$ по заявке i определим следующим образом: $\varphi'_i(t) = \varphi_i(t)$, если $t \le d(i)$, и $\varphi'_i(t) = \varphi_i(t) + L$ в противном случае.

Полученная задача является задачей синтеза расписания, минимизирующего суммарный штраф; при этом функции индивидуального штрафа нелинейны, но директивные сроки отсутствуют. Соответствующий алгоритм решения изложен выше.

Если минимальное значение суммарного штрафа в полученной задаче оказывается меньшим L, то обеспечивающее его расписание является одновременно оптимальным решением исходной задачи (с учетом директивных сроков).

В противном случае для исходной задачи допустимых расписаний не существует; отметим, что тогда построенное для модифицированной задачи оптимальное расписание в исходной задаче является расписанием, минимизирующим значение суммарного штрафа при условии, что максимально возможное число заявок обслуживается с соблюдением директивных сроков.

2.3. Задачи синтеза расписаний обслуживания для систем с параллельными и последовательными процессорами.

Сначала рассмотрим задачу, возникающую в системах с параллельными процессорами.

Задача синтеза оптимального расписания для системы с параллельными процессорами. Каждую заявку входного потока $\boldsymbol{R} = \{1, 2,..., n\}$ следует обслужить одним из m идентичных процессоров $\boldsymbol{\Pi}_j$ ($j = \overline{1,m}$). Полагаем, что процессор $\boldsymbol{\Pi}_j$ готов к обслуживанию заявок потока $\boldsymbol{R}$ начиная от целочисленного момента времени T_j ($0 = T_1$

$\le T_2 \le ... \le T_m$). Одновременное обслуживание каждым процессором более одной заявки, а также неоправданные простои процессоров запрещены; обслуживание каждой заявки осуществляется без прерываний, в момент завершения обслуживания очередной заявки каждый процессор считается готовым к обслуживанию следующей.

Каждая заявка i входного потока характеризуется тремя целочисленными параметрами:

$t(i)$ – момент поступления в систему обслуживания ($0 = t(1) \le t(2) \le \ldots \le t(n-1) \le t(n)$);

$\tau(i)$ – требуемая продолжительность обслуживания любым из процессоров ($\tau(i) > 0$);

$a(i)$ – штраф за единицу времени (такт) пребывания в системе обслуживания.

Если обслуживание заявки i завершается в момент времени $t^*(i)$, то $a(i) \times (t^*(i) - t(i))$ – индивидуальный штраф по данной заявке.

Расписание обслуживания ρ представляет собой кортеж $<\rho^1, \rho^2, \ldots, \rho^m>$, где $\rho^j = (i_{j_1}, i_{j_2}, ..., i_{j_{v(j)}})$ – последовательность обслуживания заявок процессором Π_j: первой на данном процессоре обслуживается заявка i_{j_1}, второй – заявка i_{j_2},…, последней – заявка $i_{j_{v(j)}}$ ($j = \overline{1,m}$). Каждая заявка потока $\boldsymbol{R}$ входит только в одну из составляющих кортеж последовательностей. Таким образом, $v(1) + v(2) + \ldots + v(m) = n$. По смыслу задачи параметры расписания $v(j), j = \overline{1,m}$, неотрицательны, а равенство $v(j') = 0$ означает, что по расписанию ρ процессор $\Pi_{j'}$ не принимает участия в обслуживании заявок потока R.

Обозначим через $t_{нач}(\rho, i_{j_r})$ и $t^*(\rho, i_{j_r})$ соответственно моменты начала и завершения обслуживания заявки i_{j_r} при реализации расписания ρ ($r = \overline{1, v(j)}$, $j = \overline{1,m}$).

Считаем, что реализация расписания компактна, т.е. имеют место соотношения

$$t_{нач}(\rho, i_{j_1}) = \max[T_j, t(i_{j_1})]; \qquad (2.22)$$

$$t_{нач}(\rho, i_{j_r}) = \max[t^*(\rho, i_{j_{r-1}}), t(i_{j_r})], \; r = \overline{2, v(j)}; \qquad (2.23)$$

$$t^*(\rho, i_{j_r}) = t_{нач}(\rho, i_{j_r}) + \tau(i_{j_r}), \; r = \overline{1, v(j)}. \qquad (2.24)$$

Общий штраф $U_j(\rho)$ по заявкам потока $\boldsymbol{R}$, которые по расписанию ρ обслуживаются процессором Π_j подсчитывается по формуле

$$U_j(\rho) = \sum_{r=1}^{v(j)} a(i_{j_r}) \times [(t^*(\rho, i_{j_r}) - t(i_{j_r}))], \; j = \overline{1,m}.$$

Проблема заключается в минимизации суммарного штрафа, т.е. в нахождении

$$\min_{\rho} \sum_{ij=1}^{m} U_j(\rho). \qquad (2.25)$$

Рассмотрим вопрос о решении сформулированной задачи методом динамического программирования. В связи с целочисленностью всех определяющих задачу параметров время считается дискретным.

При обслуживании потока $\boldsymbol{R}$ управленческие решения принимаются для тех моментов времени, когда по меньшей мере один процессор свободен. Каждое решение состоит в выборе заявки, которая немедленно поступает на обслуживание свободным процессором. Если таких процессоров несколько, полагаем, что для обслуживания выбранной заявки назначается свободный процессор с минимальным номером; такой процессор будем называть первым свободным процессором. Текущее состояние в момент принятия решения t определяется как набор $F = (t, S, \gamma_1, \gamma_2, \ldots, \gamma_m)$, где S – множество заявок, которые прибыли и ожидают обслуживания, а γ_j – число тактов

дискретного времени, оставшихся до освобождения процессора Π_j, $j = \overline{1,m}$. Так как t – МПР, по меньшей мере одно из чисел γ_j равно нулю. Выбор очередной обслуживаемой заявки осуществляется из совокупности S.

Через $D(t, \mu)$ обозначаем множество заявок, прибывающих в систему обслуживания на временном отрезке $[t + 1, t + \mu]$, здесь μ – целое неотрицательное число; отметим, что $D(t,1)$ – совокупность заявок, прибывающих на обслуживание в момент времени $t+1$; дополнительно полагаем, что $D(t,0) = \varnothing$. Для любого состояния $F = (t, S, \gamma_1, \gamma_2, \ldots, \gamma_m)$, множество S считаем непустым, что обеспечивается обязательным включением в него нулевой (фиктивной) заявки 0 с параметрами $a(0) = 0$, $t(0) = t$ и $\tau(0) = min\{\theta / D(t, t+\theta) \neq \varnothing\}$. Взятие на обслуживание фиктивной заявки означает простой процессора от момента t до ближайшего момента поступления в систему какой-либо новой заявки потока $\boldsymbol{R}$. Если поступившая фиктивная заявка не принимается на немедленное обслуживание, то она выпадает из дальнейшего рассмотрения. Как и в случае однопроцессорной системы, иногда принятие на обслуживание фиктивной заявки целесообразно и при наличии в S заявок, не являющихся фиктивными.

Пусть $W(t, S, \gamma_1, \gamma_2, \ldots, \gamma_m)$ обозначает минимальную величину суммарного штрафа по заявкам множества S и заявкам, которые поступят в систему обслуживания позднее момента времени t при условии, что $(t, S, \gamma_1, \gamma_2, \ldots, \gamma_m)$ – текущее состояние системы.

Предположим, что в состоянии $F = (t, S, \gamma_1, \gamma_2, \ldots, \gamma_m)$ из совокупности S выбрана и направлена на немедленное обслуживание первым свободным процессором Π_j заявка i. Тогда следующим моментом принятия решения является $t + \omega$, где $\omega = min(\gamma_1, \gamma_2, \ldots, \gamma_{j-1}, \tau(i), \gamma_{j+1}, \ldots, \gamma_m)$. Отметим, что если в рассматриваемом состоянии F свободны несколько процессоров, то $\omega = 0$ и следующее решение по загрузке принимается в тот же момент времени t. По состоянию на момент $t + \omega$ совокупность ожидающих обслуживания заявок состоит из двух подмножеств, $S \setminus \{i\}$ и $D(t, \omega)$. Таким образом, состояние системы в следующий момент принятия решения характеризуется как набор $(t + \omega, (S \setminus \{i\}) \cup D(t, \omega), \gamma_1 - \omega, \gamma_2 - \omega, \ldots, \gamma_{j-1} - \omega, \tau(i) - \omega, \gamma_{j+1} - \omega, \ldots, \gamma_m - \omega)$. В принятых обозначениях рекуррентное соотношение динамического программирования записывается в виде:

$$W(t, S, \gamma_1, \gamma_2, \ldots, \gamma_m) = \min_{i \in S} \{a(i) \times [t + \tau(i) - t(i)] + W(t+\omega, (S \setminus i) \cup D(t, \omega), \gamma_1 - \omega, \gamma_2 - \omega, \ldots, \gamma_{j-1} - \omega, \tau(i) - \omega, \gamma_{j+1} - \omega, \ldots, \gamma_m - \omega)\}, \quad (2.26)$$

здесь j – минимальное значение индекса, при котором компонента γ_j в кортеже $(\gamma_1, \gamma_2, \ldots, \gamma_j, \ldots, \gamma_m)$ равна нулю.

Если в соотношении (2.26) минимум реализуется при $i = i`$, то в момент времени $t$ на первый свободный процессор следует направить заявку $i`$ из совокупности S. Направление на обслуживание нулевой заявки означает простой процессора по меньшей мере до следующего момента пополнения множества S прибывающими заявками. Если в момент принятия решения t на первый свободный процессор направляется нулевая заявка, то одновременно такие же нулевые заявки направляются на остальные свободные процессоры.

В силу значительной вычислительной сложности изложенной процедуры для рассматриваемой задачи актуальна разработка реализуемых в приемлемом времени эвристических алгоритмов. Один из таких алгоритмов предложен далее, в главе 3.

Перейдем к рассмотрению задач обслуживания потока поступающих в дискретном времени заявок $R = \{1, 2,..., n\}$ в системе, состоящей из двух последовательных процессоров $\Pi_j, j = 1,2$.

Задача синтеза оптимального расписания для системы с двумя последовательными процессорами. В рассматриваемой задаче считаем, что каждая поступающая заявка сначала обслуживается первым, а затем вторым процессором. Полагаем, что первый процессор готов к обслуживанию заявок потока начиная от момента времени 0, а второй процессор – от момента времени T_0. Одновременное обслуживание каждым процессором более одной заявки, а также неоправданные простои процессоров запрещены; обслуживание каждой заявки осуществляется без прерываний, в момент завершения обслуживания очередной заявки каждый процессор считается готовым к обслуживанию следующей. Каждая заявка i потока R характеризуется целочисленными параметрами:

$t(i)$ – момент поступления в систему обслуживания ($0 = t(1) \le t(2) \le ... \le t(n - 1) \le t(n)$);

$\tau_1(i)$ – требуемая продолжительность обслуживания первым процессором ($\tau_1(i) > 0$);

$\tau_2(i)$ – требуемая продолжительность обслуживания вторым процессором ($\tau_2(i) > 0$);

$a(i)$ – штраф за единицу времени (такт) пребывания в системе обслуживания.

Момент полного завершения обслуживания заявки – это момент завершения ее обслуживания вторым процессором.

Расписание обслуживания ρ представляет собой кортеж $<\rho^1, \rho^2>$, где $\rho^1 = \{i_1, i_2, ..., i_n\}$ и $\rho^2 = \{j_1, j_2, ..., j_n\}$ – последовательности индексов заявок, определяющие порядок их обслуживания на первом и втором процессорах соответственно. Пусть $t_1^{нач}(\rho, i_k)$ и $t_1^*(\rho, i_k)$ обозначают моменты начала и завершения обслуживания заявки i_k на первом процессоре при реализации расписания ρ; аналогично $t_2^{нач}(\rho, j_k)$ и $t_2^*(\rho, j_k)$ – моменты начала и завершения обслуживания вторым процессором заявки j_k при реализации того же расписания, $k = \overline{1, n}$.

Считаем, что реализация расписания компактна, т.е. имеют место соотношения

$$t_1^{нач}(\rho, i_1) = t(i_1)$$

$$t_1^{нач}(\rho, i_k) = \max [t_1^*(\rho, i_{k-1}), t(i_k)], \; k=2, 3, \ldots, n;$$

$$t_1^*(\rho, i_k) = t_1^{нач}(\rho, i_k) + \tau_1(i_k), \quad k=1, 2, \ldots, n.$$

$$t_2^{нач}(\rho, j_1) = \max(t_1^*(\rho, j_1), T_0)$$

$$t_2^{нач}(\rho, j_k) = \max [t_2^*(\rho, j_{k-1}), t_1^*(\rho, j_k)], \; k=2, 3, \ldots, n;$$

$$t_2^*(\rho, j_k) = t_2^{нач}(\rho, j_k) + \tau_2(j_k), \quad k=1, 2, \ldots, n.$$

Величина суммарного штрафа по всем заявкам при реализации расписания ρ вычисляется по формуле

$$U(\rho) = \sum_{i=1}^{n} a(i) \times [t_2^*(\rho, i)] - t(i)].$$

Рассматриваемая оптимизационная задача формулируется следующим образом:

$$\min_{\rho} U(\rho) \qquad (2.27)$$

Отметим, что в некоторых производственно-транспортных системах возникает специфическая и более простая задача, когда между первым и вторым процессором заявки переставлять нельзя, т.е. последовательности обслуживания заявок на первом и втором процессорах должны совпадать. Данное ограничение является существенным; его наложение, вообще говоря, вызывает рост оптимального значения рассматриваемого минимизируемого критерия. Приведем пример. Пусть $R = \{1, 2\}$, в начальный момент времени 0 свободны оба процессора, характеристики заявок следующие: $t(1) = 0, t(2) = 10; \tau_1(1) = 10, \tau_2(1) = 10, \tau_1(2) = 1, \tau_2(2) = 1; a(1) = 1, a(2) = 10$. Легко проверяется, что единственным оптимальным в общей постановке задачи (2.27) является расписание $<\rho^1, \rho^2>$, где $\rho^1 = \{1, 2\}$, а $\rho^2 = \{2, 1\}$.

Рассмотрим вопрос о решении задачи (2.27) в общей постановке методом динамического программирования. В связи с целочисленностью всех определяющих задачу параметров время считается дискретным.

При обслуживании потока R управленческие решения принимаются для тех моментов времени, когда по меньшей мере один процессор свободен. Каждое решение состоит в выборе заявки, которая немедленно поступает на обслуживание свободным процессором. Если свободны оба процессора, то определяется пара заявок, первая из которых поступает на обслуживание первым, а вторая – вторым процессором.

Состояние системы определяем как набор $F = (t, S_1, S_2, d, T, \xi)$. Здесь t – МПР (в момент времени t по меньшей мере один процессор свободен); S_1 – совокупность заявок, которые к рассматриваемому МПР поступили и ожидают обслуживания первым процессором; S_2 – совокупность заявок, которые к рассматриваемому МПР прошли обслуживание первым процессором и ожидают обслуживания вторым процессором; d – номер процессора, который в момент времени t свободен (d равно 1 или 2); T – число тактов дискретного времени, оставшегося до освобождения занятого процессора; если $T = 0$, то по состоянию на момент времени t свободны оба процессора; ξ – номер заявки, которая в данный момент времени обслуживается первым процессором (в случае, когда $d = 1$, т.е. первый процессор свободен, полагаем $\xi = 0$).

Через $W(t, S_1, S_2, d, T, \xi)$ обозначим минимальную величину суммарного штрафа по заявкам множеств S_1, S_2 и заявкам, которые поступят в систему обслуживания позднее момента времени t, для системы, находящейся в состоянии (t, S_1, S_2, d, T, ξ), здесь t – момент принятия решения.

Как и ранее, $V(t)$ обозначает совокупность заявок, поступающих на обслуживание в произвольный момент времени t. Таким образом, шестерка $(0, V(0), \varnothing, 1, T_0, 0)$ – это начальное состояние системы, а $W(0, V(0), \varnothing, 1, T_0, 0)$ – искомое оптимальное значение критерия в рассматриваемой задаче (минимально возможное значение суммарного по всем заявкам штрафа). Запишем рекуррентные соотношения, позволяющие организовать процесс последовательного вычисления значений функции $W(t, S_1, S_2, d, T, \xi)$ вплоть до отыскания величины $W(0, V(0), \varnothing, 1, T_0, 0)$.

Вначале отметим, что при $t>t(n)$ значения $W(t, \varnothing, S_2, 1, 0, 0)$ вычисляются без затруднений, ибо в таком случае шестерка $(t, \varnothing, S_2, 1, 0, 0)$ определяет относящуюся ко второму процессору задачу мастера, оптимальное расписание в которой строится крайне просто (см. п.1 данной главы).

Далее при записи рекуррентных соотношений, необходимых для вычисления значений функции $W(t, S_1, S_2, d, T, \xi)$ на других наборах значений ее аргументов, нам будет удобно использовать функции $f(\alpha,T)$, $\varphi(\alpha,T)$ и $\psi(\alpha,\beta)$, в которых α и β принимают значения на множестве номеров заявок, а T – на множестве целых

неотрицательных чисел, не превышающих максимальной из заданных для заявок продолжительностей обслуживания первым и вторым процессором. Функция $f(\alpha,T)$ принимает значение *1*, если $T-\tau_1(\alpha) \geq 0$, и значение *0* в противном случае. Функция $\varphi(\alpha,T)$ принимает значение *1*, если $T-\tau_2(\alpha) > 0$, и значение *0* в противном случае. Функция $\psi(\alpha,\beta)$ принимает значение *1*, если $\tau_1(\alpha) \leq \tau_2(\beta)$, и значение *0* в противном случае. Считаем, что в совокупность S_1 всегда входит нулевая (фиктивная) заявка с продолжительностью обслуживания на первом процессоре *1*, а на втором – *0*; в совокупность S_2 всегда входит нулевая (фиктивная) заявка с продолжительностью обслуживания на первом процессоре *0*, а на втором – *1*; штраф по фиктивной заявке всегда нулевой. Кроме того, дополнительно положим, что для отрицательных значений аргумента T значение функции $W(t, S_1, S_2, d, T, \xi)$ равно нулю.

Для случая $T \neq 0$ имеем:

$$W(t, S_1, S_2, 1, T, 0) = \min_{\alpha \in S_1} [f(\alpha,T) \times W(t+\tau_1(\alpha), (S_1 \setminus \{\alpha\})) \cup D(t, \tau_1(\alpha)), S_2 \cup \{\alpha\}, 1, T-\tau_1(\alpha), 0) +$$

$$+ (1- f(\alpha, T)) \times W(t+T, (S_1 \setminus \{\alpha\})) \cup D(t, T), S_2, 2, \tau_1(\alpha) - T, \alpha)\ ]; \quad (2.28)$$

$$W(t, S_1, S_2, 2, T, \xi) = \min_{\alpha \in S_2} \{[a(\alpha) \times [t + \tau_2(\alpha) - t(\alpha)] + \varphi(\alpha,T) \times W(t+\tau_2(\alpha), (S_1 \cup D(t, \tau_2(\alpha)), S_2 \setminus \{\alpha\}, 2, T-\tau_2(\alpha), \xi) +$$

$$+ (1- \varphi(\alpha, T)) \times W(t + T, (S_1 \cup D(t, T), S_2 \cup \{\xi\}, 1, \tau_2(\alpha) - T, 0)]\}; \quad (2.29)$$

Для случая $T=0$ имеем:

$$W(t, S_1, S_2, 1, 0, 0) = \min_{\alpha \in S_2, \beta \in S_2} \{[a(\beta) \times [t+\tau_2(\beta) - t(\beta)] + \psi(\alpha,\beta) \times W(t+\tau_1(\alpha), (S_1 \setminus \{\alpha\})) \cup D(t, \tau_1(\alpha)),$$

$$(S_2 \setminus \{\beta\}) \cup \{\alpha\}, 1, \tau_2(\beta)-\tau_1(\alpha),,0)+(1-\psi(\alpha,\beta)) \times W(t+\tau_2(\alpha), (S_1 \setminus \{\alpha\}) \cup D(t, \tau_2(\beta)), (S_2 \setminus \{\beta\}), 2, \tau_1(\alpha)- \tau_2(\beta),\alpha)\}. \quad (2.30)$$

Сейчас рассмотрим частную задачу, в которой все заявки должны проходить обслуживание на двух последовательных процессорах в одном и том же порядке. Построим соответствующие рекуррентные соотношения динамического программирования.

В частной задаче момент времени t называем моментом принятия решения (МПР), если в данный момент свободен первый процессор. Состояние системы – это тройка объектов (t, S, T), где t – МПР; S – совокупность прибывших и по ситуации на момент времени t ожидающих обслуживания на первом процессоре заявок; T – время, оставшееся (в ситуации на момент t) до освобождения второго процессора от работ по обслуживанию тех заявок, которые уже прошли первый процессор.

Пусть $W'(t, S, T)$ – минимальный суммарный штраф по заявкам множества S и заявкам, которые прибудут на обслуживание позднее момента t, для системы, находящейся в состоянии (t, S, T).

Ясно, что начатая в момент времени t обслуживанием на первом процессоре заявка i из совокупности S на втором процессоре будет начата обслуживанием в момент $t+ max(\tau_1(i), T)$; поэтому величина индивидуального штрафа по этой заявке окажется равной $a(i) \times (t+max(\tau_1(i), T)+ \tau_2(i) - t(i))$. Следующим МПР будет $t^* = t + \tau_1(i)$; по

состоянию на МПР t^* до освобождения второго процессора останется $\max(\tau_1(i),\ T)+ \tau_2(i) - \tau_1(i)$ единиц времени. Отсюда получаем следующее рекуррентное соотношение:

$$W'(t, S, T) = \min_{i \in S} \{ a(i) \times (t + max\ (\tau_1(i),\ T) + \tau_2(i) - t(i)) +$$

(2.31)

$$+ W'(t + \tau_1(i),\ (S \setminus \{i\}) \cup D(t,\ \tau_1(i)),\ max(\tau_1(i),\ T) + \tau_2(i) - \tau_1(i))\}$$

Итоговым результатом вычислений по этому соотношению является величина $W'(0,\ V(0),\ T_0)$ – оптимальное значение критерия в рассматриваемой частной задаче.

2.4. Задачи оптимального обслуживания стационарных объектов, расположенных в одномерной зоне.

Рассматриваются две близкие по описанию модели обслуживания одним или двумя мобильными процессорами стационарно расположенных объектов. В рамках этих моделей формулируются и решаются задачи синтеза оптимальных расписаний (последовательностей обслуживания).

Модель *I* следующая. Имеется n-элементное множество $M = \{o_1,\ o_2,\ldots,o_n\}$ стационарных объектов, расположенных в рабочей зоне Ξ мобильного (перемещаемого) процессора P. Зона Ξ представляет собой направленный отрезок L, в фиксированных точках которого расположены объекты $o_1,\ o_2,\ldots,\ o_n$. Начальная точка отрезка L является базовой для процессора; от этой точки он сначала перемещается к конечной точке отрезка (прямой рейс, примем для него обозначение λ_+), а затем, достигнув ее, возвращается к базовой точке (обратный рейс, примем для него обозначение λ_-). При реализации цикла $\lambda_+\lambda_-$ процессор должен выполнить обслуживание каждого объекта множества M; обслуживание части объектов реализуется в прямом рейсе, обслуживание всех остальных объектов – в обратном рейсе. Осуществляемое процессором обслуживание каждого объекта однократное, без прерываний. Остановки процессора при выполнении цикла $\lambda_+\lambda_-$ связаны только с обслуживанием объектов множества M. Затраты времени на перемещения процессора и на обслуживание каждого объекта, а также штрафы за задержки в обслуживании характеризуются известными параметрами и функциональными зависимостями.

Отличие модели II от модели I заключается в том, что обслуживание объектов множества M осуществляется не одним, а двумя идентичными процессорами P_1 и P_2 при реализации движений только в прямом направлении; движение процессора P_2 от базовой точки начинается позже, чем движение процессора P_1. Удобно считать, что обслуживание объектов процессором P_1 реализуется в рейсе, именуемом λ_+, а обслуживание объектов процессором P_2 реализуется в рейсе, именуемом λ_-.

Примем следующие обозначения:

α_1 и α_n – начальная и конечная точки отрезка L соответственно;

$l_1,\ l_2,\ \ldots,\ l_n$ $(\alpha_1 < l_1 < l_2 <,\ \ldots,\ l_{n-1} < l_n = \alpha_n)$ – точки отрезка L, в которых расположены объекты $o_1,\ o_2,\ \ldots,\ o_n$;

$\gamma_{i-1,i}$ и $\gamma_{i,i-1}$ – затраты времени на перемещение процессора между точками расположения соседних объектов $o_{i-1},\ o_i$ в рейсах λ_+ и λ_- соответственно, $i = \overline{1,n}$; при

этом $\gamma_{0,1}$ и $\gamma_{1,0}$ – затраты времени на перемещения процессора между базовой точкой α_1 и точкой l_1 в рейсах λ_+ и λ_- соответственно;

τ_i – продолжительность обслуживания процессором объекта o_i ($i = \overline{1,n}$).

Все величины $\gamma_{i-1,i}$, $\gamma_{i,i-1}$, τ_i – целые положительные числа, $i = \overline{1,n}$.

Для каждого объекта o_j ($j = \overline{1,n}$) считается заданной монотонно возрастающая в нестрогом смысле функция индивидуального штрафа $\varphi_j(t)$; если обслуживание объекта o_j завершается в момент времени t, то $\varphi_j(t)$ – соответствующая величина потерь (штрафа) по данному объекту.

Для модели I считаем, что $t = 0$ – момент времени, в который процессор начинает рейс λ_+. Для модели II считаем, что процессор P_1 начинает начинает рейс λ_+. в момент времени $t = 0$, а процессор P_2 начинает рейс λ_- в момент времени $t = T$.

Стратегиями обслуживания множества объектов $M = \{o_1, o_2,\ldots,o_n\}$ будем называть подмножества элементов V из совокупности $N = \{1, 2, \ldots, n\}$. В рамках модели I объекты o_j, где $j \in V$, в реализации стратегии V обслуживаются процессором P в рейсе λ_+, все остальные объекты обслуживаются этим процессором в рейсе λ_- (для определенности полагаем, что объект o_n обслуживается при завершении процессором рейса λ_+, т.е. $n \in V$). В рамках модели II объекты o_j, $j \in V$, в реализации стратегии V обслуживаются процессором P_1, все остальные объекты – процессором P_2.

Очевидно, что каждая стратегия однозначно определяет расписание обслуживания объектов. Спецификой рассматриваемых моделей является то, что общее число возможных стратегий обслуживания имеет порядок 2^n, в то время как обычно для задач однопроцессорного обслуживания число возможных стратегий имеет порядок n!

Для объекта o_j ($j = \overline{1,n}$) через $t^*_j(V)$ обозначим момент завершения его обслуживания при реализации стратегии V; при этом $\varphi_j(t^*_j(V))$ – соответствующая величина индивидуального штрафа.

В рамках модели I формулируем две задачи.

Задача I-1:

$$\min_{V \subseteq N} \sum_{j=1}^{n} \varphi_j(t^*_j(V)).$$

Задача I-2:

$$\min_{V \subseteq N} \max_{j} \varphi_j(t^*_j(V)) .$$

В рамках модели II формулируются идентично записываемые задачи **II-**1 и **II-2**.

Задача I-1, алгоритмы решения. Обозначим через $V(j)$ совокупность не превосходящих j элементов из V (здесь V – произвольная стратегия в рассматриваемой задаче). Для выражения $\sum_{k=1}^{j} \gamma_{k-1,\,k} + \sum_{k \in V(j)} \tau_k$ введем обозначение δ_j. Через $M(j)$ обозначим сумму следующих трех слагаемых:

$\sum_{k=j+1}^{n} \gamma_{k-1,\,k}$ – затраты времени на перемещение процессора от точки l_j к точке α_n;

$\sum_{k=j+1}^{n} \gamma_{k,\,k-1}$ – затраты времени на перемещение процессора от точки α_n к точке l_j;

$A_j^n = \sum_{k=j+1}^{n} \tau_k$ – суммарная продолжительность обслуживания объектов $o_j, o_{j+1}, \ldots, o_n$.

Как очевидно,

$$t^*_j(V) = \begin{cases} \delta_j, \text{ если } j \in V; \\ \delta_j + M(j), \text{ если } j \notin V. \end{cases}$$

Существенно, что величины $M(j), j = \overline{1,n}$, не зависят от выбираемой стратегии обслуживания, т.е. от того, какие объекты совокупности $o_{j+1}, o_{j+2}, \ldots, o_{n-1}$ обслуживаются в прямом, а какие – в обратном рейсе процессора.

Для решения задачи I-1 применим метод динамического программирования. Пусть $B^*(i, D)$ – минимально возможная величина суммарного штрафа по объектам совокупности $\{o_1, o_2, \ldots, o_i\}$ при условии, что в рейсе λ_+ общее время, затрачиваемое на обслуживание объектов из этой совокупности, равно D (таким образом, в частном случае $D = 0$ все объекты совокупности $\{o_1, o_2, \ldots, o_i\}$ обслуживаются в рейсе λ_-; в частном случае $i = n$ и $D = A_1^n$ все объекты множества M обслуживаются в рейсе λ_+). Отметим, что возможные значения параметра D целочисленны и лежат в диапазоне $[0, A_1^n]$, при $D > A_1^n$ величины $B^*(i, D)$ заведомо не имеют смысла.

Как очевидно, $B^*(1, 0)$ – величина индивидуального штрафа по первому объекту в случае, когда его обслуживание реализуется в рейсе λ_-, т.е. завершается в момент времени $\gamma_{0,1} + M(1)$. В то же время $B^*(1, \tau_1)$ – величина индивидуального штрафа по первому объекту, если его обслуживание выполняется в рейсе λ_+, т.е. завершается в момент $\gamma_{0,1} + \tau_1$. Поэтому

$$B^*(1,0) = \varphi_1(\gamma_{0,1} + M(1)), \qquad (2.32)$$

$$B^*(1, \tau_1) = \varphi_1(\gamma_{0,1} + \tau_1). \qquad (2.33)$$

При $D \notin \{0, \tau_1\}$ величины $B^*(1, D)$ смысла не имеют. Дополнительно удобно положить

$$B^*(1, D) = +\infty \text{ при } D \notin \{0, \tau_1\}. \qquad (2.34)$$

Предположим, что все значения $B^*(i, D)$ для некоторого $i \in \{1, 2, \ldots, n - 2\}$ уже найдены. При определении значений $B^*(i + 1, D)$, надо учитывать две возможности:

1). Объект o_{i+1} обслуживается в рейсе λ_+. Тогда рассматриваемой паре $(i + 1, D)$ непосредственно предшествует ситуация $(i, D - \tau_{i+1})$.

2). Объект o_{i+1} обслуживается в рейсе λ_-. В этом случае рассматриваемой паре непосредственно предшествует ситуация(i, D).

С учетом этих возможностей получаем:

$$B^*(i+1, D) = \min\{B^*(i, D - \tau_{i+1}) + \varphi_{i+1}(\gamma_{0,1} + \gamma_{1,2} + \ldots + \gamma_{i,i+1} + D), \quad (2.35)$$

$$B^*(i, D) + \varphi_{i+1}(\gamma_{0,1} + \gamma_{1,2} + \ldots + \gamma_{i,,i+1} + D + M(i+1))\};$$

здесь $i \in \{1, 2, \ldots, n-2\}$.

Вычисляемое по формуле (2.35) значение $B^*(i+1, D)$ оказывается равным $+\infty$ в том и только том случае, когда определяемая парой $(i+1, D)$ ситуация возникнуть не может.

При определении значений $B^*(n, D)$ следует иметь в виду, что по принятому соглашению объект o_n обслуживается при завершении движения $\lambda_{+,}$ т.е. $n \in V$. Поэтому каждой рассматриваемой паре (n, D) непосредственно предшествует только ситуация $(n-1, D - \tau_n)$. Следовательно,

$$B^*(n, D) = B^*(n-1, D - \tau_n) + \varphi_n(\gamma_{0,1} + \gamma_{1,2} + \ldots + \gamma_{n-1,n} + D). \quad (2.36)$$

Каждая вычисляемая величина $B^*(n, D)$ – это минимально возможный суммарный штраф по всем объектам множества M, если в рейсе λ_+ на обслуживание объектов этого множества затрачивается время D.

Оптимальное значение критерия K_{opt} рассматриваемой задачи I-1 определяется по формуле

$$K_{opt} = \min_D B^*(n, D) \quad (2.37)$$

Равенства (2.32) – (2.37) суть рекуррентные соотношения динамического программирования для решения задачи (I-1).

В соответствующей вычислительной процедуре сначала по формулам (2.32) - (2.33) определяются значения $B^*(1, 0)$ и $B^*(1, \tau_1)$. В соответствии с формулой (2.34) для $D \notin \{0, \tau_1\}$ величины $B^*(1, D)$ полагаются равными $+\infty$. Далее, на основании формулы (2.35) сначала определяются все значения $B^*(2, D)$, затем все значения $B^*(3, D)$ и т.д. Последними по данной формуле для всех возможных значений D определяются величины $B^*(n-1, D)$. Используя формулу (2.36), по известным величинам $B^*(n-1, D)$ определяем значения $B^*(n, D)$. Оптимальное значение критерия решаемой задачи находится по формуле (2.37).

Процесс вычислений по соотношениям (2.32) – (2.37) удобно представлять как последовательное заполнение клеток таблицы значений функции $B^*(i, D)$. Строки этой таблицы соответствуют возможным значениям параметра i, $i \in \{1, 2, \ldots, n\}$, а столбцы – возможным значениям параметра D, $D \in \{0, 1, 2, \ldots, A_1^n\}$. В каждую клетку (i, D) вносится величина $B^*(i, D)$.Таблица заполняется по строкам, в порядке роста их номеров. Заметим, что в процессе счета найденное значение $B^*(i, D)$ оказывается бесконечным в том и только том случае, когда при определяемом строкой значении параметра i фиксируемое столбцом значение параметра D невозможно.

Зафиксировав для каждого найденного в процессе вычислений по формуле (2.35) значения $B^*(i+1, D)$ номер компоненты, на которой реализуется минимум правой части этой формулы, и определив затем значение параметра D, на котором достигается минимум правой части (2.37), легко строим искомую оптимальную стратегию.

Будем считать, что функции индивидуального штрафа либо заданы таблично, либо значение каждой из них в любой точке вычисляется путем выполнения ограниченного, не превышающего некоторой константы C, числа элементарных операций. В таком случае общее число выполняемых изложенным алгоритмом элементарных операций имеет верхней оценкой величину, прямо пропорциональную числу клеток в заполняемой таблице, т.е. произведению $n \times (A_1^n + 1)$.

ПРИМЕР 2.5. Требуется определить оптимальную стратегию обслуживания пяти стационарных объектов при условиях: $\gamma_{01} = 2,\ \gamma_{12} = 1,\ \gamma_{23} = 3,\ \gamma_{34} = 2, \gamma_{45} = 1,\ \gamma_{54} = 2,\ \gamma_{43} = 2,\ \gamma_{32} = 2,\ \gamma_{21} = 1,\ \gamma_{10} = 3$ (значение γ_{10} для синтеза оптимальной стратегии фактической роли не играет); $\tau_1 = 1;\ \tau_2 = 1;\ \tau_3 = 2;\ \tau_4 = 3;\ \tau_5 = 2;$

$$\varphi_1(t) = \begin{cases} 0 \text{ при } t \le 5; \\ t - 5 \text{ при } t > 5; \end{cases}$$

$$\varphi_2(t) = \begin{cases} 0 \text{ при } t \le 15; \\ 2(t - 15) \text{ при } t > 15; \end{cases}$$

$$\varphi_3(t) = \begin{cases} 0 \text{ при } t \le 20; \\ 4(t - 20) \text{ при } t > 20; \end{cases}$$

$\varphi_4(t) = t;$

$\varphi_5(t) = 2t.$

Легко вычисляются необходимые для применения рекуррентных соотношений величины $M(j)$:

$M(1) = 23;\ M(2) = 20;\ M(3) = 14;\ M(4) = 8.$

Далее подсчитываем значения функции $B^*(i, D)$. По условиям решаемой задачи параметр i принимает значения $1, 2, 3, 4, 5;$ целочисленный параметр D принимает значения из отрезка $[0, 9]$. Результаты вычислений удобно фиксировать в таблице 2.3 значений функции $B^*(i, D)$, параметру i соответствуют строки таблицы, параметру D – ее столбцы. Символы бесконечности в таблицу не вносим.

По формулам (2.32) и (2.33) получаем: $B^*(1,0) = \varphi_1(25) = 20;\ B^*(1,1) = \varphi_1(3) = 0.$

Далее по формуле (2.35) имеем:

$B^*(2,0) = B^*(1,0) + \varphi_2(23) = 36;$

$B^*(2,1) = \min\{ B^*(1,0) + \varphi_2(4);\ B^*(1,1) + \varphi_2(24)\} = \min\{ 20 + 0;\ 0 + 18\} = 18$ (минимум правой части рекуррентного соотношения достигается на второй компоненте, что отмечается цифрой в скобках после записи числа 18 в клетку (2,1) таблицы 2.2);

$B^*(2,2) = \varphi_1(3) + \varphi_2(5) = 0;$

$B^*(3,0) = B^*(2,0) + \varphi_3(20) = 36;\ B^*(3,1) = B^*(2,1) + \varphi_3(21) = 22;$

$B^*(3,2) = \min\{ B^*(2,0) + \varphi_3(8);\ B^*(2,2) + \varphi_3(22)\} = \min\{ 36 + 0;\ 0 + 8\} = 8$ (минимум правой части рекуррентного соотношения достигается на второй компоненте);

$B^*(3,3) = B^*(2,1) + \varphi_3(9) = 18;\ B^*(3,4) = B^*(2,2) + \varphi_3(10) = 0.$

$B^*(4,0) = 52;\ B^*(4,1) = 39;\ B^*(4,2) = 26;\ B^*(4,3) = \min\{ 36 + 11;\ 18 + 19\} = 37$ (минимум правой части рекуррентного соотношения достигается на второй компоненте);

$B^*(4,4) = \min\{ 22 + 12;\ 0 + 20\} = 20$

(минимум правой части рекуррентного соотношения достигается на второй компоненте);

$B^*(4,5) = 21;\ B^*(4,6) = 32;\ B^*(4,7) = 33.$

Далее по формуле (2.36) находим:

$B^*(5,2) = B^*(4,0) + \varphi_5(11) = 74;\ B^*(5,3) = B^*(4,1) + \varphi_5(12) = 63;\ B^*(5,4) = B^*(4,2) + \varphi_5(13) = 52;\ B^*(5,5) = B^*(4,3) + \varphi_5(14) = 65;\ B^*(5,6) = B^*(4,4) + \varphi_5(15) = 50;\ B^*(5,7) = B^*(4,5) + \varphi_5(16) = 53;\ B^*(5,8) = B^*(4,6) + \varphi_5(17) = 66;\ B^*(5,9) = B^*(4,7) + \varphi_5(18) = 67.$

Согласно (2.37), оптимальное значение критерия рассматриваемой задачи – это минимальное из вычисленных значений $B^*(5, D)$. Оно равно *50*, соответствующее значение параметра D равно *6*. Таким образом, оптимальная стратегия предусматривает, что на непосредственное обслуживание объектов в прямом рейсе должно затрачиваться *6* единиц времени. При этом на обслуживание объекта o_5 уходит время $\tau_5 = 2$, на непосредственное в прямом рейсе обслуживание первых четырех объектов остается *4* единицы времени. При вычислении $B^*(4,4)$ минимум правой части реализуется на второй компоненте правой части рекуррентного соотношения; значит объект o_4 в прямом рейсе обслуживать не следует, все четыре единицы времени должны быть затрачены на обслуживание в этом рейсе объектов из числа первых трех.

Таблица 2.3. Значения функции $B^*(i, D)$.

	0	1	2	3	4	5	6	7	8	9
1	20	0								
2	36	$18_{(2)}$	0							
3	36	22	$8_{(2)}$	18	0					
4	52	39	26	$37_{(2)}$	$20_{(2)}$	21	32	33		
5			74	63	52	65	50	53	66	67

Ситуация, непосредственно предшествующая той, что описывается парой аргументов *(3,4)* функции B* единственна – это *(2, 2);* таким образом, объект o_3 должен быть обслужен в прямом рейсе. На непосредственное обслуживание в этом рейсе первых двух объектов остается две единицы времени. Ситуация, непосредственно предшествующая той, что описывается парой аргументов *(2, 2)* функции B* единственна – это *(1, 1),* объект o_2 должен быть обслужен в прямом рейсе. Оставшаяся единица времени уходит на обслуживание в прямом рейсе объекта o_1. Таким образом,

оптимальная стратегия в рассмотренном примере $V_{opt} = \{1, 2, 3, 5\}$; объекты o_j, где $j \in V_{opt}$, в реализации данной стратегии должны обслуживаться при выполнении прямого рейса, оставшийся объект o_4 – при выполнении обратного рейса.

Отметим, что в случае, когда все функции индивидуального штрафа линейны ($\varphi_i(t) = a_i t + b_i$, $i = \overline{1,n}$), имеется возможность применения перестановочного приема, и задача I-1 решается очень просто. Пусть $Q_j = \sum_{i=j+1}^{n} a_i$; вся совокупность вводимых характеристик Q_j, $j \in \{1, 2, \ldots, n-1\}$, вычислима путем выполнения линейно зависящего от n числа операций. Для любой стратегии V и любого элемента $j \in V$ переход от обслуживания объекта o_j, в прямом рейсе к его обслуживанию в обратном рейсе влечет увеличение индивидуального штрафа по этому объекту на $a_j M(j)$ и уменьшение суммарного штрафа по всем остальным объектам на $Q_j \tau_j$. Очевидно, что при реализации оптимальной стратегии объект o_j должен обслуживаться в обратном рейсе, если $a_j M(j) < Q_j \tau_j$; объект o_j должен обслуживаться в прямом рейсе, если $a_j M(j) > Q_j \tau_j$; в случае $a_j M(j) = Q_j \tau_j$ объект o_j может быть обслужен в любом из рейсов. Изложенное правило обеспечивает возможность синтеза оптимальной в задаче I-1 с линейными функциями индивидуального штрафа стратегии за время, линейно зависящее от числа подлежащих обслуживанию объектов.

Задача I-2, алгоритм решения. При решении данной задачи используется следующий факт.

Утверждение 4.1. Пусть V – любая стратегия в модели обслуживания I, удовлетворяющая условию $k \notin V$, и пусть $V^* = V \cup \{k\}$, здесь $k \in \{1, 2, \ldots, n-1\}$. Тогда $t^*_k(V) > t^*_k(V^*)$, $t^*_j(V) = t^*_j(V^*)$ для $j < k$ и $t^*_j(V^*) > t^*_j(V)$ для $j > k$, здесь $j \in \{1, 2, \ldots, n-1\}$.

Доказательство данного утверждения очевидно.

Стратегию, при реализации которой значение максимального из индивидуальных штрафов не превышает константы F, будем именовать F-стратегией. Для заданной константы F поставим вопрос, существует ли в рассматриваемой задаче I-2 какая-либо F-стратегия. Ответ дает следующий, основанный на утверждении 4.1 и обозначаемый через ***A(F),*** алгоритм:

1. Полагаем $V = \{n\}$; проверяем, обеспечивает ли стратегия V индивидуальный штраф по объекту o_n не больший, чем F; если это так, переходим к п.2; в противном случае в рассматриваемой задаче F-стратегии отсутствуют, ответ на поставленный вопрос отрицательный, алгоритм работу заканчивает;
2. Полагаем $k = 1$;
3. Вводим стратегии $V^* = V$ и $V^{**} = V \cup \{k\}$;
4. Проверяем, обеспечивает ли стратегия V^* по объектам o_n и o_k индивидуальные штрафы, не большие F; если это так, переходим к п.6; в противном случае – к п.5;
5. Проверяем, обеспечивает ли стратегия V^{**} по объектам o_n и o_k индивидуальные штрафы не большие F; если это не так, в рассматриваемой задаче F-стратегии отсутствуют, ответ на поставленный вопрос отрицательный, алгоритм работу заканчивает; в случае положительного ответа полагаем $V = V^{**}$ и переходим к п.6;

6. Если $k = n - 1$, полученное множество V является F-стратегией; ответ на поставленный вопрос положительный, алгоритм работу заканчивает; в противном случае следует увеличить имеющееся значение k на 1 и перейти к п.2.

Благодаря описанному алгоритму, экстремальная задача I-2 решается методом деления пополам. Процесс решения начинается с назначения левой границы S_1 и правой границы W_1 диапазона, в котором локализовано искомое оптимальное значение критерия.

Можно положить:

$$S_1 = \max_j \varphi_j(t^*_j(V_j)), \text{ где } V_j = \{j\}.$$

$$W_1 = \max_j \varphi_j(t^*_j(V)), \text{ где } V = \{1, 2, \ldots, n\}.$$

Далее выполняется процедура, состоящая не более чем из $log_2(W_1 - S_1)+1$ однотипных этапов. На произвольном k-ом этапе имеющийся отрезок $[S_k, W_k]$, в котором локализовано оптимальное значение критерия, делится пополам; координату середины отрезка обозначаем через F_k. Затем к решаемой задаче применяется алгоритм $\boldsymbol{A(F_k)}$; если получаемый в результате ответ положителен, оптимальное значение критерия локализовано в левом полуотрезке; в противном случае это значение находится в правом полуотрезке. Поэтому в случае положительного ответа полагаем $S_{k+1} = S_k$ и $W_{k+1} = F_k$; в случае отрицательного ответа устанавливаем $S_{k+1} = F_k + 1$ и $W_{k+1} = W_k$. Далее переходим к выполнению следующего этапа. Процесс заканчивается в ситуации, когда в полученном отрезке локализации имеется только одна целочисленная точка.

Будем считать, что все функции индивидуального штрафа либо заданы таблично, либо значение каждой из них в любой точке вычисляется путем выполнения ограниченного, не превышающего некоторой константы числа элементарных действий. Тогда алгоритм $\boldsymbol{A(F)}$ имеет линейно зависящую от n оценку количества выполняемых элементарных операций и вычислительная сложность изложенной процедуры решения задачи $I\text{-}2$ имеет оценку $Cnlog_2(W_1 - S_1)$, где C – не зависящая от параметров решаемой задачи константа.

Задача II-1, алгоритм решения. Для решения задачи II-1 применим метод динамического программирования. Пусть $B'(i, D)$ – минимально возможная величина суммарного штрафа по объектам совокупности $\{o_1, o_2, \ldots, o_i\}$ при условии, что в рейсе λ_+ общее время, затрачиваемое на обслуживание объектов из этой совокупности равно D. Отметим, что так же, как при рассмотрении задачи I-1, возможные значения параметра D целочисленны и лежат в числовом диапазоне $[0, A_1^n]$.

По определению $B'(1, 0)$ – это величина индивидуального штрафа по объекту o_1 в случае, когда его обслуживание реализуется в рейсе λ_-, т.е. завершается в момент времени $\gamma_{0,1} + \tau_1 + T$. В то же время $B'(1, \tau_1)$ – величина индивидуального штрафа по объекту o_1, если его обслуживание выполняется в рейсе λ_+, т.е. завершается в момент $\gamma_{0,1} + \tau_1$. Получаем:

$$B'(1, 0) = \varphi_1(\gamma_{0,1} + \tau_1 + T), \qquad (2.38)$$

$$B'(1, \tau_1) = \varphi_1(\gamma_{0,1} + \tau_1). \qquad (2.39)$$

При $D \notin \{0, \tau_1\}$ величины $B'(1, D)$ смысла не имеют. Удобно положить

$$B'(1, D) = +\infty \text{ при } D \notin \{0, \tau_1\}. \qquad (2.40)$$

Предположим, что все значения $B'(i, D)$ уже найдены. При отыскании значений $B'(i + 1, D)$, где $i \in \{1, 2, \ldots, n-1\}$, надо учитывать две возможности:

1. Объект o_{i+1} обслуживается в рейсе λ_+. Тогда рассматриваемой паре $(i + 1, D)$ непосредственно предшествует ситуация $(i, D - \tau_{i+1})$.

2. Объект o_{i+1} обслуживается в рейсе λ_-. В этом случае рассматриваемой паре непосредственно предшествует ситуация(i, D).

С учетом указанных возможностей получаем:

$$B'(i + 1, D) = \min\{B'(i, D - \tau_{i+1}) + \varphi_{i+1}(\gamma_{0,1} + \gamma_{1,2} + \ldots + \gamma_{i,i+1} + D),\ B'(i, D) + \qquad (2.41)$$

$$+ \varphi_{i+1}(\gamma_{0,1} + \gamma_{1,2} + \ldots + \gamma_{i,,i+1} + \tau_1 + \tau_2 + \ldots + \tau_{i+1} - D + T)\};$$

здесь $i \in \{1, 2, \ldots, n-1\}$.

Вычисляемое значение $B'(i + 1, D)$ оказывается равным $+\infty$ в том и только том случае, когда определяемая парой $(i + 1, D)$ ситуация реально возникнуть не может.

Каждая определяемая по формуле (2.41) величина $B'(n, D)$, $D \in \{0, 1, 2, \ldots, A_1^n\}$, – это минимально возможный суммарный штраф по всем объектам, если в рейсе λ_+ на их обслуживание затрачивается время D. Оптимальное значение критерия K_{opt} рассматриваемой задачи II-1 определяется по формуле

$$K_{opt} = \min_D B'(n, D) \qquad (2.42)$$

Формулы (2.38) – (2.42) суть рекуррентные соотношения динамического программирования для решения задачи (II-1). Процесс вычислений по этим соотношениям удобно представлять как последовательное заполнение таблицы, строки которой соответствуют значениям параметра i, $i \in \{1, 2, \ldots, n\}$, а столбцы – значениям параметра D, $D \in \{0, 1, 2, \ldots, A_1^n\}$. Таблица заполняется по строкам, в порядке роста их номеров. Зафиксировав в процессе вычислений по формуле (2.41) для каждого найденного значения $B'(i + 1, D)$ номер компоненты, на которой реализуется минимум правой части, и определив значение параметра D, на котором достигается минимум правой части (2.42), легко строим искомую оптимальную стратегию.

Приведенный алгоритм решения задачи II-1 имеет такую же оценку вычислительной сложности, как изложенный выше решающий алгоритм для задачи I-1 в ее общей постановке.

Задача II-2, алгоритм решения Легко видеть, что для модели II утверждение 4.1 неверно. Поэтому предложенный для задачи I-2 решающий метод к задаче II-2 неприменим. Задачу II-2 можно решить методом динамического программирования. Пусть $B''(i, D)$ – минимально возможная величина максимального из индивидуальных штрафов по объектам совокупности $o_1, o_2, \ldots, o_i$ при условии, что в рейсе λ_+ общее время, затрачиваемое на обслуживание объектов из этой совокупности, равно D. Так же, как при рассмотрении предыдущих задач, возможные значения параметра D целочисленны и лежат в числовом диапазоне $[0, A_1^n]$.

$B''(1, 0)$ – это величина индивидуального штрафа по объекту o_1 в случае, когда его обслуживание реализуется в рейсе λ_-, т.е. завершается в момент времени $\gamma_{0,1} + \tau_1 +$

T. В то же время $B''(1,\ \tau_1)$ – величина индивидуального штрафа по тому же объекту o_1, если его обслуживание выполняется в рейсе λ_+, т.е. завершается в момент $\gamma_{0,1} + \tau_1$. Поэтому

$$B''(1,\ 0) = \varphi_1(\gamma_{0,1} + \tau_1 + T), \qquad (2.43)$$

$$B''(1,\ \tau_1) = \varphi_1(\gamma_{0,1} + \tau_1). \qquad (2.44)$$

При $D \notin \{0, \tau_1\}$ величины $B''(1, D)$ смысла не имеют. Полагаем

$$B''(1, D) = +\infty \text{ при } D \notin \{0, \tau_1\}. \quad (2.45)$$

Предположим, что все значения $B''(i, D)$ найдены. Для отыскания величин $B''(i+1, D)$, где $i \in \{1, 2,\ldots, n\text{-}1\}$, с учетом имеющихся возможностей обслуживания объекта o_{i+1} записываем:

$$B''(i+1, D) = \min\{\max [B''(i, D - \tau_{i+1}), \varphi_{i+1}(\gamma_{0,1} + \gamma_{1,2} + \ldots + \gamma_{i,i+1} + D)], \qquad (2.46)$$

$$\max [B''(i, D), \varphi_{i+1}(\gamma_{0,1} + \gamma_{1,2} + \ldots + \gamma_{i,,i+1} + \tau_1 + \tau_2 + \ldots + \tau_{i+1} - D + T)]\};$$

здесь $i \in \{1, 2,\ldots, n - 1\}$.

Вычисляемое значение $B''(i + 1, D)$ оказывается равным $+\infty$ в том и только том случае, когда определяемая парой $(i + 1, D)$ ситуация возникнуть не может.

В итоге итерационного процесса каждая вычисленная по формуле (2.46) величина $B''(n, D)$, $D \in \{0,1, 2,\ldots, A_1^n\}$, – это минимально возможная величина наибольшего из индивидуальных штрафов по всем объектам в ситуации, когда в рейсе λ_+ на их обслуживание затрачивается время D. Оптимальное значение критерия K_{opt} рассматриваемой задачи II-2 определяется по формуле

$$K_{opt} = \min_D B''(n, D) \qquad (2.47)$$

Равенства (2.43) – (2.47) суть рекуррентные соотношения динамического программирования для решения задачи (II-2).

Оценка вычислительной сложности изложенного алгоритма решения задачи (II-2). такая же, как для приведенных общих алгоритмов решения задач I-1 и II-1.

ЗАДАЧИ К ГЛАВЕ 2.

1. В условиях двухпроцессорной задачи мастера требуется построить оптимальное расписание обслуживания заявок *1 – 6* со следующими характеристиками: *a(1) = 4, τ(1) =1; a(2) = 7, τ(2) =2; a(3) =8, τ(3) =3; a(4) =4, τ(4) =3;a(5) =10, τ(5) =5; a(6) =17, τ(6) =4.*

2. Записать рекуррентные соотношения динамического программирования для решения трехпроцессорной задачи мастера.

3. Обобщенная на случай нелинейных функций индивидуального штрафа однопроцессорная задача мастера определяется следующими исходными данными: обслуживанию подлежат заявки *1,2,3,4*; продолжительности обслуживания заявок: *τ(1) = 4, τ(2) = 1, τ(3) =3, τ(4) =1;* функции индивидуального штрафа: $\varphi_1(t) = 4t^2$, $\varphi_2(t) = 3t$, $\varphi_3(t) = 2t^2 + t$, $\varphi_4(t) = t^2 + 2t$. Требуется найти оптимальное расписание обслуживания.

4. Задача однопроцессорного обслуживания множества заявок с минимаксным критерием определяется следующими исходными данными: обслуживанию подлежат заявки *1,2,3,4,5*; продолжительности обслуживания заявок: $\tau(1) = 3,\ \tau(2) = 2,\ \tau(3) = 4,\ \tau(4) = 1,\ \tau(5) = 2;$ функции индивидуального штрафа: $\varphi_1(t) = 4t^2,\ \varphi_2(t) = 3t,\ \varphi_3(t) = t^2 + t,\ \varphi_4(t) = t^2 + 2t,\ \varphi_5(t) = t^3 + 1.$ Требуется найти оптимальное расписание обслуживания.

5. Задача однопроцессорного обслуживания множества заявок при наличии директивных сроков определяется следующими исходными данными: $\tau(1) = 4,\ d(1) = 5;\ \tau(2) = 3,\ d(2) = 6;\ \tau(3) = 5;\ d(3) = 9;\ \tau(4) = 2,\ d(4) = 9;\ \tau(5) = 3;\ d(5) = 10;\ \tau(6) = 4;\ d(6) = 12;\ \tau(7) = 4;\ d(7) = 13;\ \tau(8) = 4,\ d(8) = 15.$ Найти расписание, обеспечивающее выполнение в срок максимально возможного числа заявок.

6. Записать соотношения динамического программирования для решения задачи двухпроцессорного обслуживания множества заявок при наличии директивных сроков. В этой задаче для каждой заявки i, $i = \overline{1, n}$, считаются заданными продолжительность обслуживания $\tau(i)$ и директивный срок $d(i)$; требуется построить расписание, минимизирующее число нарушений предписанных заявкам директивных сроков.

7. Каноническая задача однопроцессорного обслуживания потока из шести заявок характеризуется следующими данными: $t(1) = 0,\ \tau(1) = 3,\ a(1) = 4;\ t(2) = 0,\ \tau(2) = 4,\ a(2) = 7;\ t(3) = 1,\ \tau(3) = 1,\ a(3) = 8;\ t(4) = 3,\ \tau(4) = 2,\ a(4) = 7;\ t(5) = 5,\ \tau(5) = 3,\ a(5) = 3;\ t(6) = 5,\ \tau(6) = 2,\ a(6) = 7.$ Методом динамического программирования (обратный счет) требуется определить оптимальное расписание обслуживания.

8. Каноническая задача однопроцессорного обслуживания потока из шести заявок характеризуется следующими данными: $t(1) = 0,\ \tau(1) = 2,\ a(1) = 3;\ t(2) = 1,\ \tau(2) = 4,\ a(2) = 7;\ t(3) = 1,\ \tau(3) = 1,\ a(3) = 8;\ t(4) = 3,\ \tau(4) = 2,\ a(4) = 7;\ t(5) = 5,\ \tau(5) = 3,\ a(5) = 3;\ t(6) = 5,\ \tau(6) = 2,\ a(6) = 7.$ Методом динамического программирования (прямой счет) требуется определить оптимальное расписание обслуживания.

9. В задаче однопроцессорного обслуживания потока заявок с линейными функциями индивидуальных штрафов и директивными сроками в совокупности расписаний, обеспечивающих соблюдение максимального числа сроков, найти последовательность обслуживания с минимальным значением суммарного штрафа по заявкам. Характеристики заявок следующие: $t(1) = 0,\ \tau(1) = 3,\ \varphi_1(t) = t,\ d(1) = 5;\ t(2) = 2,\ \tau(2) = 3,\ \varphi_2(t) = 4t,\ d(2) = 5;\ t(3) = 3,\ \tau(3) = 4,\ \varphi_3(t) = 2t,\ d(3) = 7;\ t(4) = 4,\ \tau(4) = 3,\ \varphi_4(t) = 3t,\ d(4) = 9;\ t(5) = 6,\ \tau(5) = 1,\ \varphi_5(t) = 4t,\ d(5) = 7.$

10. Записать рекуррентные соотношения динамического программирования для задачи распределения заявок потока $\boldsymbol{R} = \{1, 2, \ldots, n\}$ между двумя идентичными параллельными процессорами. Считается, что каждый процессор обслуживает направляемые ему заявки в порядке возрастания их номеров. Каждая заявка i характеризуется тремя целочисленными параметрами: $t(i)$ – момент поступления в систему обслуживания ($0 = t(1) \le t(2) \le \ldots \le t(n-1) \le t(n)$); $\tau(i)$ – требуемая продолжительность обслуживания любым из процессоров ($\tau(i) > 0$); $a(i)$ – штраф за единицу времени пребывания в системе обслуживания. Минимизируемым критерием является суммарный штраф по всем заявкам.

ГЛАВА 3. ТРУДНОРЕШАЕМЫЕ ЗАДАЧИ. ПОЛИНОМИАЛЬНО РАЗРЕШИМЫЕ КОНКРЕТИЗАЦИИ, ПРИБЛИЖЕННЫЕ И ЭВРИСТИЧЕСКИЕ АЛГОРИТМЫ.

В данной главе изучаются вопросы вычислительной сложности задач и алгоритмов. Выделяются класс задач, разрешимых в полиномиальном времени, и класс труднорешаемых задач. Алгоритм решает задачу в полиномиальном времени, если число выполняемых им элементарных операций не превышает $P(L)$, где P – некоторый полином от L – объема входной информации по решаемой задаче. Число элементарных операций, гарантирующих получение искомого результата в труднорешаемой задаче, в лучшем случае имеет оценку, экспоненциально зависящую от объема определяющей задачу входной информации. Для ряда труднорешаемых задач дискретной оптимизации в данной главе выделяются полиномиально разрешимые конкретизации; излагаются общие принципы построения приближенных и эвристических алгоритмов, строятся процедуры, позволяющие синтезировать качественные в том либо ином смысле решения за реально приемлемое время.

3.1. Полиномиально разрешимые и NP-трудные задачи.

Различные математические задачи имеют различную трудность. Трудность любой задачи естественно характеризовать сложностью простейшего из алгоритмов ее решения. Для оценки сложности алгоритмов имеются два принципиально разных подхода. При первом подходе оценки отображают сложность создания алгоритма, здесь учитывается количество операторов в записи алгоритма, количество циклов, глубины циклов и т.д. При втором подходе оценки характеризуют сложность реализации алгоритма; такими оценками являются используемый объем памяти и количество элементарных операций, выполняемых при реализации алгоритма (что соответствует времени машинного счета). Именно второй подход является целесообразным при классификации задач по их вычислительной сложности.

Каждый алгоритм создается для решения *массовой задачи*, т.е. совокупности однотипных индивидуальных задач, отличающихся только исходными данными. Необходимый объем памяти и количество элементарных операций, выполняемых при реализации алгоритма, являются функциями от объема входной информации по решаемой индивидуальной задаче. Объем входной информации можно задавать, например, в байтах.

Каждому алгоритму A соотносим функции $T_A(L)$ и $V_A(L)$, где $T_A(L)$ – верхняя оценка количества элементарных операций, а $V_A(L)$ – верхняя оценка размера необходимой памяти при решении задачи с объемом входной информации L.

Будем говорить, что задача Z имеет *полиномиальную временную сложность* (решается в полиномиальном времени, *полиномиально разрешима*), если для нее существует решающий алгоритм A^* такой, что $T_{A^*}(L) \le P_1(L)$; здесь $P_1(L)$ – некоторый полином от одной переменной L. Задача Z имеет *полиномиальную емкостную сложность*, если для нее существует решающий алгоритм A' такой, что $V_{A'}(L) \le P_2(L)$; здесь $P_2(L)$ – некоторый полином от одной переменной L. Отметим следующий

очевидный факт: если задача Z имеет полиномиальную временную сложность, то ее емкостная сложность также полиномиальная.

Возможны дальнейшие разбиения классов задач, имеющих полиномиальную временную и емкостную сложность. Так задача Z разрешима в линейном времени, если для нее существует решающий алгоритм A такой, что $T_A(L) \leq CL$; задача Z разрешима в квадратичном времени, если для нее существует решающий алгоритм A такой, что $T_A(L) \leq CL^2$; здесь и далее в приводимых оценочных неравенствах C обозначает некоторую фиксированную константу.

Приведем несколько примеров полиномиально разрешимых задач из числа рассмотренных в предыдущих главах: классическая задача о назначениях; задача отыскания кратчайших путей; задача мастера; задача Джонсона для двух станков.

Наряду с классом полиномиально разрешимых задач, имеется обширная совокупность важных в теоретическом и прикладном аспектах задач дискретной математики, для которых не удалось построить полиномиальных решающих алгоритмов, но которые разрешимы в экспоненциально зависящем от объема входной информации времени.

Будем говорить, что задача Z имеет *экспоненциальную временную сложность* (разрешима в экспоненциальном времени, *экспоненциально разрешима*), если для нее существует решающий алгоритм A такой, что $T_A(L) \leq Ca^L$; здесь a – некоторая превышающая единицу константа.

В качестве примеров экспоненциально разрешимых задач приведем задачу о ранце, задачу коммивояжера, каноническую задачу однопроцессорного обслуживания потока заявок, задачи Джонсона для трех и более станков.

Объем входной информации по подлежащей решению индивидуальной задаче нередко оценивается числом определяющих эту задачу параметров. Если все параметры – лежащие в фиксированном диапазоне целые числа, то между верхней оценкой показателя L и числом параметров n имеется линейная взаимосвязь. В таком случае можно считать, что характеристики временной и емкостной сложности решающего алгоритма, являются функциями от n.

	$n = 10$	$n = 20$	$n = 30$	$n = 40$	$n = 50$	$n = 60$
$T_A(n) = n$	0,00001сек	0,00002 сек	0,00003 сек	0,00004 сек	0,00005 сек	0,00006 се
$T_A(n) = n^2$	0,0001сек	0, 0004сек	0,0009сек	0,0016сек	0.025сек	0,036 сек
$T_A(n) = .n^3$	0,001сек	0,008сек	0,027сек	0,064сек	0,125сек	0,216 сек
$T_A(n) = n^5$	0,1сек	3,2 сек	24,3 сек	1,7 мин	5,2 мин	13,0 мин
$T_A(n) = 2^n$	0,001сек	1,0 сек	17,9 мин	12,7 суток	35,7 лет	$3{,}7 \times 10^4$ лет
$T_A(n) = 3^n$	0,059сек	58 минут	6,5 лет	385500 лет	2×10^{10} лет	$1{,}3 \times 10^{15}$ ле

Таблица 3.1. Затраты времени при полиномиальных и экспоненциальных функциях сложности.

Различие в возможностях использования полиномиальных и экспоненциальных по вычислительной сложности алгоритмов значительно. Взятая из

[7] таблица 3.1 позволяет сравнивать скорости роста нескольких типичных полиномиальных и экспоненциальных функций. В клетках этой таблицы представлены продолжительности решения задач при определяемых столбцами значениях параметра n и определяемых строками функциях временной сложности; быстродействие ЭВМ считается равным 10^6 операций в секунду.

Принципиальная разница между алгоритмами полиномиальной и экспоненциальной временной сложности проявляется также при анализе влияния быстродействия ЭВМ на размеры реально решаемых задач. Взятая из той же монографии [7] таблица 3.2 показывает, как увеличатся размеры решаемых за один час задач при увеличении быстродействия ЭВМ в сто и тысячу раз в сравнении с современными машинами. Заметим, что для функции $T_A(n) = 2^n$ увеличение скорости вычислений в *1000* раз приводит к тому, что размер наибольшей решаемой за *1* час задачи возрастает только на *10*, в то время как при квадратичной функции временной сложности этот размер увеличивается более чем в *30* раз.

Задачу будем называть *труднорешаемой*, если для нее не существует полиномиального по верхней оценке временной вычислительной сложности решающего алгоритма.

	На современных ЭВМ	На ЭВМ, в 100 раз более быстрых	На ЭВМ, 1000 раз более быстр
$T_A(n) = n$	N_1	$100 \times N_1$	$1\,000 \times N_1$
$T_A(n) = n^2$	N_2	$10 \times N_2$	$31{,}6 \times N_2$
$T_A(n) = n^3$	N_3	$4{,}64 \times N_3$	$10 \times N_3$
$T_A(n) = n^5$	N_4	$2{,}5 \times N_4$	$3{,}98 \times N_4$
$T_A(n) = 2^n$	N_5	$N_5+6{,}64$	$N_5+9{,}97$
$T_A(n) = 3^n$	N_6	$N_6+4{,}19$	$N_6+6{,}29$

Таблица 3.2. Наибольшие размеры задач, решаемых за один час.

Следует отметить, что между понятиями "труднорешаемая задача" и "задача, неразрешимая в реально приемлемом времени" нельзя ставить знак тождества. Различие между эффективными и неэффективными алгоритмами может иметь иной характер, если размеры решаемых задач невелики (функция $T_1(n) = 2^n$, например, принимает при $n \leq 20$ меньшие значения, чем функция $T_2(n) = n^5$). Кроме того, широко известны некоторые имеющие экспоненциальную сложность алгоритмы, хорошо зарекомендовавшие себя в практике решения задач средних и даже больших размеров. Это объясняется тем, что временная сложность $T(L)$ определена как верхняя граница числа необходимых операций при любом входе объема L, включая и наихудший возможный случай; именно на наихудший случай ориентирована оценка $T(L)$. Оценка же "в среднем" может иметь существенно меньший порядок роста. Такова, в частности, ситуация с симплекс-методом [26] для решения задач линейного программирования; только в среднем этот метод дает полиномиальную оценку времени счета.

Остановимся на вопросе установления труднорешаемости, т.е. определения невозможности построения для рассматриваемой задачи полиномиального по верхней оценке временной вычислительной сложности алгоритма решения. Будем

рассматривать два класса дискретных задач – задачи оптимизации и задачи распознавания.

В каждой *задаче распознавания* считается определенным некоторый универсум U, в котором выделено подмножество V; по произвольному элементу x из U требуется определить, принадлежит ли x подмножеству V. Приведем два примера.

Пусть U – множество всех натуральных чисел, а подмножество V является совокупностью всех простых чисел. По натуральному числу x требуется определить, является ли оно простым.

Пусть U – множество всех конечных графов, а подмножество V является совокупностью всех гамильтоновых графов [14]. По конечному графу G требуется определить, является ли он гамильтоновым.

Решить задачу распознавания означает ответить "да" или "нет" на поставленный вопрос. В случае, если в задаче распознавания правильным является ответ "да", будем говорить, что эта задача имеет положительное решение.

Для ряда задач распознавания полиномиальные по верхней оценке временной вычислительной сложности алгоритмы известны. В качестве примеров приведем задачу определения по сети с ограниченными пропускными способностями дуг, возможен ли в ней поток заданной мощности [6], и задачу определения по графу, является ли он эйлеровым [14]. Для многих других задач, в том числе важных с точки зрения приложений, таких алгоритмов построить не удалось. К последним относятся, в частности, следующие задачи.

1. **ВЫПОЛНИМОСТЬ КНФ**. Задана конъюнктивная нормальная форма (КНФ) F от n переменных. Требуется определить, является ли F выполнимой (КНФ выполнима, если существует набор логических значений переменных, обращающий ее в истину).

2. **ТРЕХМЕРНОЕ СОЧЕТАНИЕ**. Дано множество $M \subseteq XxYxZ$, где X, Y, Z – непересекающиеся множества, каждое из которых состоит из равного числа элементов q. Требуется определить, содержится ли в M состоящее из q элементов подмножество M' такое, что никакие два элемента из M' не имеют ни одной равной координаты.

3. **ВЕРШИННОЕ ПОКРЫТИЕ**. Заданы n-вершинный граф G и натуральная константа k, $k<n$. Требуется определить, можно ли в заданном графе выделить k-элементное подмножество вершин M такое, что каждое ребро графа имеет по меньшей мере одним своим концом вершину из M.

4. **КЛИКА**. Заданы n-вершинный граф G и натуральная константа k, $k<n$. Требуется определить, имеется ли k-элементное подмножество вершин данного графа M такое, что любые две вершины из M соединены ребром.

5. **ГАМИЛЬТОНОВ ЦИКЛ**. Задан конечный граф G; требуется определить, имеется ли простой цикл, проходящий через все вершины этого графа.

6. **РАЗБИЕНИЕ**. Заданы конечное множество A и "веса" $v(a)$ всех элементов этого множества; указанные веса являются натуральными числами. Весом любого подмножества M, $M \subseteq A$, называем сумму весов входящих в M элементов. Требуется определить, можно ли разбить совокупность A на два подмножества равного веса.

Для всех перечисленных задач распознавания известны имеющие экспоненциальную временную сложность решающие алгоритмы. В то же время ни для одной из этих задач не удалось построить алгоритма с полиномиально зависящей от размера входной информации верхней оценкой числа выполняемых им элементарных операций.

Задачу распознавания *"$\alpha \in P$?"* назовем *полиномиально сводимой* к задаче распознавания *"$\beta \in Q$?"*, если существует вычислимая в полиномиальном времени функция φ такая, что $\alpha \in P$ тогда и только тогда, когда $\varphi(\alpha) \in Q$. Задачи распознавания *"$\alpha \in P$?"* и *"$\beta \in Q$?"* считаются *полиномиально эквивалентными*, если задача *"$\alpha \in P$?"* полиномиально сводима к *"$\beta \in Q$?"*, а задача *"$\beta \in Q$?"* полиномиально сводима к *"$\alpha \in P$?"*.

Если задача *"$\alpha \in P$?"* полиномиально сводима к задаче*"$\beta \in Q$?"* и задача*"$\beta \in Q$?"* имеет полиномиальную временную сложность, то временная сложность задачи *"$\alpha \in P$?"* также полиномиальная.

Как показано в [7], перечисленные выше задачи распознавания 1 – 6 полиномиально эквивалентны. Поэтому наличие полиномиального решающего алгоритма для любой из них автоматически влечет разрешимость в полиномиальном времени каждой из остальных задач.

Известно, что любой алгоритм может быть реализован соответствующим образом построенной машиной Тьюринга [16]. Если некоторый алгоритм предусматривает выполнение полиномиально зависящего от размера входной информации числа элементарных операций, то можно построить машину Тьюринга (*МТ*), которая за полиномиально зависящее от размера входной информации время данный алгоритм реализует. Действия *МТ* на каждом такте ее работы однозначно определяются входной информацией по решаемой задаче.

В [7] изложена концепция недетерминированной машины Тьюринга (*НДМТ*). В процессе работы *НДМТ* возможны ситуации, когда возникает возможность реализации одного из нескольких элементарных действий. Недетерминированные машины Тьюринга предназначаются исключительно для решения задач распознавания. Считается, что недетерминированная машина дает положительный ответ, если для заданного входного слова (оно является записью входной информации по решаемой задаче) существует способ работы данной *НДМТ* такой, что в итоге машина оказывается в специально выделенном положительном заключительном состоянии.

Существенным является следующий факт. Для каждой из перечисленных задач 1 – 6 можно построить решающую ее *НДМТ*, функционирующую в полиномиально зависящем от объема входной информации времени. При решении любой из задач 1 – 6 *НДМТ* работает в два этапа. Первый этап – генерация подтверждения (подтверждение генерируется недетерминированным образом). Второй этап – проверка верности построенного подтверждения (на данном этапе машина функционирует детерминированно); если подтверждение верно, машина переходит в положительное заключительное состояние. Для иллюстрации рассмотрим задачу "Выполнимость КНФ", заключающуюся в определении по произвольной КНФ $\boldsymbol{F}(x_1, x_2,\ldots, x_n)$, является ли данная формула выполнимой. Решающая эту задачу *НДМТ* (входным словом является формула $\boldsymbol{F}(x_1, x_2,\ldots, x_n)$) на первом этапе работы недетерминированным образом генерирует подтверждение, которое в данном случае представляет собой набор из нулей и единиц (в принципе, может быть напечатан любой набор символов *0* и *1*). Далее первый напечатанный элемент подтверждения трактуется как логическое значение переменной x_1, второй элемент – как логическое значение переменной x_2, и

т.д. Если число символов в подтверждении оказалось большим, чем n, то лишние символы роли не играют; если же символов недостает, то всем переменным, для которых в подтверждении соответствующих элементов не оказалось, приписывается значение *0*. На втором, детерминированном этапе работы машина осуществляет проверку верности сгенерированного подтверждения, т.е. обращается ли в истину КНФ F при определенных описанным образом значениях ее переменных. Только в случае истинности КНФ машина переходит в положительное заключительное состояние. Оба этапа работы машины реализуются в полиномиально зависящем от объема входной информации времени. Машина дает положительный ответ (способ функционирования недетерминированной части машины, обеспечивающий в итоге ее переход в положительное заключительное состояние существует) тогда и только тогда, когда рассматриваемая КНФ выполнима.

При решении каждой из перечисленных задач 1 – 6 функционирование машины Тьюринга является недетерминированным только на первом этапе, где реализуется "угадывание" подтверждения. Фактически, недетерминированная машина Тьюринга решает проблему распознавания в полиномиальном времени тогда и только тогда, когда в полиномиальном времени, детерминированным вычислением можно проверить правильность кем-то указанного подтверждения.

Задача распознавания считается задачей класса *NP*, если она решается соответствующим образом построенной недетерминированной машиной Тьюринга за время, ограниченное значением полинома от объема определяющей задачу входной информации.

Задачи распознавания, разрешимые на детерминированных машинах Тьюринга за время, ограниченное значением полинома от объема входной информации, называются задачами класса *P*. Так как концепция недетерминированной машины Тьюринга является обобщением понятия детерминированной машины, класс задач *P* является подклассом класса задач *NP*, т.е. $P \subseteq NP$.

Задачу распознавания именуем *NP*- полной, если:

1) она принадлежит классу *NP*;

2) к ней полиномиально сводима любая принадлежащая классу NP задача.

Установлено (доказательство см. в [7]), что любая задача распознавания класса *NP* полиномиально сводима к задаче "Выполнимость КНФ". Таким образом, задача "Выполнимость КНФ" – пример *NP*-полной задачи.

Из того, что перечисленные выше, принадлежащие классу *NP,* задачи распознавания 1 – 6 полиномиально эквивалентны, делаем вывод: каждая из этих задач *NP*-полна. К настоящему времени число известных, имеющих существенный теоретический интерес и прикладное значение *NP*-полных задач исчисляется тысячами. Все *NP*-полные задачи полиномиально эквивалентны. Наличие полиномиального по верхней оценке временной вычислительной сложности решающего алгоритма для какой-либо одной *NP*-полной задачи автоматически повлекло бы за собой полиномиальную разрешимость всех принадлежащих классу *NP* задач, включая *NP*-полные. Накопленный опыт решения дискретных задач, с учетом факта полиномиальной эквивалентности *NP*-полных задач, позволяет принять следующую гипотезу:

ДЛЯ NP-ПОЛНЫХ ЗАДАЧ ПОЛИНОМИАЛЬНЫХ ПО ВЕРХНЕЙ ОЦЕНКЕ ВРЕМЕННОЙ ВЫЧИСЛИТЕЛЬНОЙ СЛОЖНОСТИ АЛГОРИТМОВ РЕШЕНИЯ НЕ СУЩЕСТВУЕТ.

Отметим, что сформулированная гипотеза часто записывается в виде *"P ≠ NP"*. Формула *"P ≠ NP"*, означающая несовпадение классов задач *P* и *NP*, как очевидно, эквивалентна утверждению, что для любой *NP*-полной задачи не существует решающего алгоритма, полиномиального по верхней оценке временной вычислительной сложности.

Выше понятие полиномиальной сводимости было введено только для задач распознавания. Расширим это понятие следующим образом. Будем говорить, что задача распознавания *"$\alpha \in P$?"* полиномиально сводится к задаче оптимизации *Z,* если по каждой индивидуальной задаче *"$\alpha^0 \in P$?"* за время, полиномиально зависящее от размера ее исходных данных, строится индивидуальная задача оптимизации Z^0 и определяется константа *C* такие, что задача Z^0 обладает решением, обеспечивающим значение критерия не хуже *C* (т.е. большее или равное *C* для задач максимизации, меньшее или равное *C* для задач минимизации) тогда и только тогда, когда исходная задача распознавания имеет положительное решение.

Задачу назовем *NP-трудной*, если к ней полиномиально сводится любая принадлежащая классу *NP* задача. Легко устанавливается, что бинарное отношение полиномиальной сводимости обладает свойством транзитивности. Поэтому для доказательства *NP*-трудности той либо иной задачи достаточно показать, что к этой задаче полиномиально сводится некоторая известная *NP*-полная задача.

Из гипотезы $P \neq NP$ вытекает, что для *NP*-трудных задач полиномиальных по верхней оценке временной вычислительной сложности алгоритмов решения не существует.

Т е о р е м а 3.1. Классическая задача коммивояжера *NP*- трудна.

Доказательство заключается в полиномиальном сведении к классической задаче коммивояжера (*ЗК*) следующей *NP*-полной задачи "ГАМИЛЬТОНОВ ЦИКЛ": задан произвольный ориентированный граф ***G*** с множеством вершин {*1, 2,…,n*}; требуется определить, имеется ли в ***G*** простой, проходящий через все вершины графа цикл. По заданному графу ***G*** определяющую *ЗК* матрицу $S(\boldsymbol{G})=\{s_{ij}\}$ размера *nxn* строим следующим образом. Все стоящие на главной диагонали элементы s_{ii} полагаем равными нулю; в случае $i \neq j$ считаем, что $s_{ij} = 1$, если дуга (i,j) в графе *G* имеется, и $s_{ij} = 2$, если дуги (i,j) в графе *G* нет; здесь $i = \overline{1,n}$, $j = \overline{1,n}$. В построенной задаче коммивояжера длина проходящего через все города кратчайшего замкнутого маршрута не может быть меньше *n*. Такой маршрут действительно имеет длину *n* тогда и только тогда, когда каждый реализуемый в нем элементарный переход имеет длину *1*, т.е. когда каждый переход выполняется по дуге исходного графа ***G*** . В свою очередь, это возможно тогда и только тогда, когда граф ***G*** гамильтонов. Таким образом, решив определяемую матрицей $S(\boldsymbol{G})$ задачу коммивояжера, мы можем ответить на вопрос, является ли граф гамильтоновым. Граф ***G*** гамильтонов тогда и только тогда, когда оптимальное значение критерия в определяемой матрицей $S(\boldsymbol{G})$ задаче коммивояжера равно *n*. Теорема доказана.

В предположении $P \neq NP$ задача коммивояжера не может иметь полиномиального решающего алгоритма, ибо в противном случае за полиномиальное время решалась бы задача "ГАМИЛЬТОНОВ ЦИКЛ".

В выполненном доказательстве получен более сильный результат, чем сформулированный в теореме. Показано, что *NP*-трудным является частный подкласс задач коммивояжера, в которых длины элементарных переходов принимают значения только из множества {1, 2}.

Из выше рассмотренных к числу *NP*- трудных относятся также задача о ранце (данный факт доказывается путем описанного в главе 1 сведения к задаче о ранце *NP*-полной задачи "РАЗБИЕНИЕ"), задачи Джонсона для трех и более станков[22], каноническая задача однопроцессорного обслуживания потока заявок [7, 12].

Отметим, что согласно имеющимся определениям каждая *NP*-полная задача *NP*- трудна. В совокупности *NP*-трудных задач *NP*-полные задачи образуют наиболее простой частный подкласс.

Любая задача оптимизации индуцирует соответствующую ей задачу распознавания. В частности, задача максимизации критерия $K(x)$ при $x \in D$ порождает задачу распознавания, в которой по начальным данным исходной задачи максимизации и константе C требуется определить, имеется ли в D элемент x^* такой, что $K(x^*) \geq C$. Задача минимизации критерия $K(x)$ при $x \in D$ порождает задачу распознавания, в которой требуется определить, имеется ли в D элемент x^* такой, что $K(x^*) \leq C$. Очевидно, что из *NP*-полноты индуцированной задачи распознавания следует *NP*-трудность исходной задачи оптимизации.

Алгоритм A решения задачи Z с целочисленными начальными данными назовем *псевдополиномиальным*, если верхняя оценка количества выполняемых им элементарных операций T_A не превышает значения некоторого полинома от двух переменных $P_A(L, M)$, где L – объем входной информации по задаче, а M – максимальное из чисел в ее составе. Задачу Z назовем *NP*-полной (*NP*-трудной) в сильном смысле, если а) она *NP*-полна (соответственно *NP*-трудна); б) для нее нет псевдополиномиального решающего алгоритма.

Отметим, что рассмотренный в главе 1, основанный на заполнении таблицы стоимостей алгоритм решения m-мерной задачи о ранце с n предметами (1.34) - (1.36) является псевдополиномиальным. В главе 1 было установлено, что верхней оценкой числа выполняемых этим алгоритмом элементарных операций является $Cmn(W+1)^m$, где W –максимальное из чисел, записанных в правых частях линейных ограничений (1.35), а C – константа, не зависящая от конкретных характеристик задачи. Из приведенной оценки следует, что число выполняемых алгоритмом элементарных операций не превышает произведения $CL(M+1)^m$, где L – объем входной информации по задаче, M – максимальное из чисел в ее составе, а C – константа, не зависящая от определяющих задачу параметров. Приведенная оценка является полиномом от двух переменных L и M степени $m+1$.Таким образом, m-мерная задача о ранце *NP*-трудна, но не в сильном смысле.

Достаточно просто показывается, что в предположении $P \neq NP$ задача "Гамильтонов цикл" *NP*-полна в сильном смысле. Действительно, исходная информация по данной задаче определяется матрицей смежности графа G, элементы этой матрицы принадлежат множеству $\{0,1\}$; таким образом, максимальное число в составе входной информации равно единице. Если бы для задачи "Гамильтонов цикл" существовал псевдополиномиальный алгоритм решения, то его вычислительная сложность – полином от двух переменных $P(L, M)$, где $M = 1$. Но $P(L,1)$ – это полином от одной переменной L. Предположение о наличии псевдополиномиального алгоритма для задачи "Гамильтонов цикл" означало бы ее полиномиальную разрешимость.

Последнее же, с учетом $P \neq NP$, невозможно. Аналогично показывается, что введенные выше задачи "Выполнимость КНФ", "Трехмерное сочетание", "Вершинное покрытие" и "Клика" также NP-полны в сильном смысле. Задача "Разбиение" легко сводится к одномерной задаче о ранце и имеет поэтому псевдополиномиальный решающий алгоритм.

Известно [7], что *NP*-полной в сильном смысле является следующая задача **"3-РАЗБИЕНИЕ"**: множество *M* состоит из *3n* элементов, каждый элемент *x* из *M* имеет вес *v(x),* суммарный вес всех элементов множества *M* равен *nV*, веса всех элементов и *V* – натуральные числа; спрашивается, можно ли разбить множество *M* на *n* попарно непересекающихся трехэлементных подмножеств так, чтобы суммарный вес элементов каждого подмножества оказался равным *V*.

Задача сохраняет *NP*-полноту в сильном смысле и при следующем дополнительном условии: для каждого *x* из *M* имеет место $(V/4) < v(x) < (V/2)$.

Отметим ряд дополнительных фактов. Задача линейного программирования полиномиально разрешима (в то время как обычно используемый для ее решения симплекс-метод по верхней оценке временной вычислительной сложности полиномиальным не является), задача целочисленного линейного программирования *NP*-трудна в сильном смысле. *NP*-трудными в сильном смысле являются все рассмотренные выше варианты задачи коммивояжера и каноническая задача однопроцессорного обслуживания потока заявок.

Следует заметить, что наряду с анализом гарантированного функционирования алгоритмов значительный интерес представляют оценки их поведения "в среднем". Отличие здесь может быть принципиальным. В частности, доказано, что в среднем симплекс-метод решает задачи линейного программирования в полиномиальном времени и быстрее чем известные алгоритмы решения этой задачи с гарантированно полиномиальной верхней оценкой трудоемкости вычислений.

3.2. Полиномиально разрешимые подклассы труднорешаемых задач.

После того, как было формализовано понятие "алгоритм", появилась возможность разделения массовых задач на разрешимые и неразрешимые. Задача неразрешима, если не существует алгоритма, ее решающего. Если та либо иная важная массовая задача неразрешима, то интерес представляет вопрос выделения ее частных случаев (подклассов), для которых можно построить решающие алгоритмы. В качестве примера приведем неразрешимую теорию – арифметику натуральных чисел (см., например, [9]); важно, что в этой теории имеются разрешимые подтеории (в частности, теория сложения натуральных чисел).

Дискретные задачи являются задачами переборного типа, каждая из них разрешима перебором конечного числа вариантов. Для таких задач в определенном смысле аналогом концепции неразрешимости является труднорешаемость (*NP*-полнота или *NP*- трудность). Естественно возникают вопросы выделения для труднорешаемых массовых задач полиномиально разрешимых подклассов; эти вопросы важны как в теоретическом плане, так и в чисто прикладном аспекте.

Рассмотрим изучавшуюся в главе 2 каноническую задачу однопроцессорного обслуживания потока заявок. Известно, что эта задача *NP*-трудна в сильном смысле [12]. Результат о труднорешаемости сохраняется, если рассматривать только задачи,

удовлетворяющие легко интерпретируемому и реально выполняемому в приложениях условию

$$t(n) \; \mathbf{O} \; c^*n, \qquad (3.1)$$

где c^* – фиксированная натуральная константа. Класс задач однопроцессорного обслуживания, для которых условие (3.1) выполняется, обозначим $K[c^*]$. Ниже будут рассматриваться только задачи класса $K[c^*]$.

Через $W(\rho)$ будет обозначаться величина суммарного штрафа по всем заявкам при условии, что их обслуживание выполняется по расписанию ρ.

Каждое из вводимых далее дополнительных ограничений на класс рассматриваемых моделей или на класс допустимых расписаний обеспечивает возможность построения для возникающих при этом оптимизационных задач полиномиальных по верхней оценке временной вычислительной сложности решающих алгоритмов. Важно, что налагаемые ограничения естественны для многих практических задач

Частная модель 1. Заявки α и β будем называть однотипными, если $\tau(\alpha) = \tau(\beta)$ и $a(\alpha) = a(\beta)$. Вводимое здесь ограничение касается числа типов заявок в потоке R.

Задачу класса $K[c^*]$ отнесем к подклассу $K[c^*, m]$, если во входном потоке присутствуют заявки не более чем m различных типов.

В подклассе $K[c^*, m]$ любое множество заявок S может быть представлено m-мерным вектором $N(S) = (n_1, n_2, \ldots, n_m)$, где n_i – число заявок i-го типа в множестве S, $i = \overline{1, m}$.

Вместо функции $\Sigma(t, S)$ введем соответствующую функцию $\Sigma'(t, N(S))$, полагая для всех S

$$\Sigma(t, S) = \Sigma'(t, N(S)).$$

Основное рекуррентное соотношение (2.16) для вычисления значений функции Σ' в задачах подкласса $K[c^*, m]$ модифицируется очевидным образом, а общее количество рассматриваемых векторов $(n_1, n_2,,\ldots,n_m)$, являющихся аргументами функции Σ', оценивается сверху величиной n^{m-1}. С учетом этого обстоятельства верхняя оценка числа выполняемых алгоритмом элементарных операций принимает вид $C^{**}n^m$, где C^{**} – константа, не зависящая от значений параметров m и n. Таким образом, верхняя оценка временной вычислительной сложности задач подкласса $K[c^*, m]$ оказывается полиномиальной.

Частная модель 2. В рамках данной модели будем считать, что количество заявок, находящихся в системе и ожидающих обслуживания, не может превысить натуральную константу k. Множество расписаний, удовлетворяющих этому ограничению, обозначим $\Phi(k)$. Рассматриваемая задача записывается в виде

$$\min_{\rho \in \Phi(k)} W(\rho) \qquad (3.2)$$

Пусть $\Sigma^k(t, S)$ обозначает минимальную величину суммарного штрафа по заявкам множества S и заявкам, поступающим в систему обслуживания позднее момента времени t для определяемой состоянием (t, S) ситуации. При этом считается, что t – момент принятия решения, множество S заявок, ожидающих на данный момент обслуживания, не более чем $(k + 1)$-элементное (нулевая заявка учитывается), а допустимыми являются только расписания совокупности $\Phi(k)$.

Если в момент t начинается обслуживание заявки i, то через временной промежуток $\tau(i)$ количество $\boldsymbol{r(t, i)}$ ожидающих обслуживания заявок будет равно

$$|S| - sign(i) + |D(t, \tau(i))|.$$

Обозначим через $S(k, t)$ множество заявок i из S, для которых выполняется неравенство $r(t, i) \le k + 1$. С учетом введенных обозначений рекуррентное соотношение динамического программирования для решения задачи (3.2) записывается следующим образом:

$$\Sigma^k(t, S) = \min_{i \in S(k,t)} \{a(i) \cdot [t + \tau(i) - t(i)] + \Sigma^k(t + \tau(i), (S / i) \cup D(t, \tau(i))\}. \qquad (3.3)$$

Заметим, что для некоторых состояний системы обслуживания (t, S) множество заявок $S(k, t)$ может оказаться пустым. Такие пары считаем запрещенными, а значения соответствующих величин $\Sigma^k(t, S)$ равными ∞. Если при подсчете оказалось, что $\Sigma^k(0, V(0)) = \infty$, то в рассматриваемой задаче множество расписаний $\Phi(k)$ пусто. С учетом принадлежности решаемой задачи классу $K[c^*]$, устанавливаем, что верхней оценкой временной вычислительной сложности основанного на (3.3)алгоритма является полином от n степени $k + 1$.

Частная модель 3. Будем говорить, что в расписании ρ заявка β *обгоняет в обслуживании* заявку α, если $\alpha < \beta$ (т.е. $= t(\alpha) \le t(\beta)$), но заявка β обслуживается раньше, чем заявка α. Разность $\beta - \alpha$ будем называть *длиной обгона*.

Расписание ρ назовем *d-расписанием*, если при его реализации длины обгонов не превышают натуральной константы d. Очевидно, что при реализации d-расписания произвольную заявку i могут обогнать в обслуживании только заявки из совокупности $\{i + 1, i + 2, \dots, i + d\}$. Множество всех d-расписаний будем обозначать $\Phi[d]$.

Рассмотрим экстремальную задачу

$$\min_{\rho \in \Phi[d]} W(\rho). \qquad (3.4)$$

При реализации d-расписания состояния системы – это тройки вида $(t, \gamma, \boldsymbol{x})$, где t – очередной момент принятия решения по загрузке процессора (в момент t процессор считается свободным); γ – порядковый номер-наименование первой (по возрастанию номеров) необслуженной заявки; $\boldsymbol{x} = (x_1, x_2, \dots, x_d)$ – d-мерный вектор с булевозначными координатами, причем $x_i = 1$, если по состоянию на момент времени t заявка $\gamma + i$ уже обслужена, и $x_i = 0$ в противном случае.

Пусть $\Sigma^d(t, \gamma, \boldsymbol{x})$ обозначает минимальную величину суммарного штрафа по всем необслуженным до момента времени t заявкам для системы, находящейся в состоянии $(t, \gamma, \boldsymbol{x})$. При этом считается, что возможными являются только расписания из множества $\Phi[d]$.

Для состояния $(t, \gamma, \boldsymbol{x})$ через $Q(t, \gamma, \boldsymbol{x})$ обозначим совокупность всех необслуженных на данный момент t заявок. В это множество входят: заявка γ, все принадлежащие потоку R заявки $\gamma + i$ такие, что $x_i = 0$ (здесь $i = 1, 2, \dots, d$); все принадлежащие потоку R заявки y такие, что $y > \gamma + d$. Через $Q^d(t, \gamma, \boldsymbol{x})$ обозначим подмножество заявок из $Q(t, \gamma, \boldsymbol{x})$, номера-наименования которых не превосходят $\gamma + d$. Для произвольного множества заявок M через $\nu(M)$ обозначим минимальный из номеров-наименований заявок этого множества, а через $\boldsymbol{w}(M)$ обозначим вектор $(w_1, w_2,$

... , w_d), в котором координата $w_i = 0$ тогда и только тогда, когда заявка $\nu+i$ находится в множестве M; во всех остальных случаях $w_i = 1$; здесь $i=\overline{1,n}$.

Рассмотрим общую ситуацию, когда множество $Q(t, \gamma, \boldsymbol{x})$ состоит более чем из одной заявки. С учетом введенных обозначений, основное соотношение динамического программирования для решения задачи (3.4) записывается в виде

$$\Sigma^d(t,\gamma,\boldsymbol{x}) = \min_{i\in Q^d(t,\gamma,x)} \{a(i)[\max(t,\ t(i)) + \tau(i) - t(i)] + \Sigma^d(t+\tau(i),\ \nu(Q(t,\gamma,\boldsymbol{x})\backslash\{i\}),\ \boldsymbol{w}(Q(t,\gamma,\boldsymbol{x})\backslash\{i\})\}. \quad (3.5)$$

Состояние системы в последний момент принятия решения (все, кроме одной, заявки уже обслужены) записывается как тройка $(t,\gamma^*,(1,1,\ldots,1))$, где $\gamma^* \in \{n-d-1,\ n-d,\ n-d+1,\ldots,\ n\}$. В такой ситуации

$$\Sigma^d(t,\gamma^*,(1,1,\ldots,1)) = a(\gamma^*)[\max(t,\ t(\gamma^*)) + \tau(\gamma^*) - t(\gamma^*)] \quad (3.5')$$

Примем дополнительное предположение, что в рассматриваемом классе задач требуемая продолжительность обслуживания процессором любой заявки ограничена некоторой константой. В таком случае в процессе вычислений по рекуррентным соотношениям (3.5) - (3.5') аргумент t функции $\Sigma^d(t,\gamma,\boldsymbol{x})$ принимает линейно зависящее от n число различных значений; аргумент γ принимает значения из множества $\{1, 2, \ldots, n\}$; для возможных значений векторного аргумента $\boldsymbol{x}$ имеется не более 2^d вариантов. При вычислении каждого следующего значения функции число выполняемых элементарных операций линейно зависит от параметра d. Получаем, что верхней оценкой вычислительной сложности алгоритма решения задачи (3.4), основанного на соотношениях (3.5) - (3.5'), является

$$C\, d\, n^2\, 2^d,$$

где C – константа, не зависящая от значений параметров n и d. При любом фиксированном d верхняя оценка числа необходимых для решения задачи элементарных операций – полином второй степени от n.

Заметим, что общее число расписаний обслуживания n-элементного потока заявок в однопроцессорной системе равно $n!$; метод динамического программирования решает задачу синтеза оптимального расписания за экспоненциальное время. Общее число d-расписаний при любом фиксированном d зависит от n экспоненциально, но синтез оптимального d-расписания реализуется в квадратично зависящем от n времени.

Частная модель 4. Предусматривается следующее ограничение: каждую заявку могут обогнать в обслуживании не более q других заявок (q – заданная константа).

Расписание ρ назовем $\{q\}$-расписанием, если при его реализации любую заявку обгоняют в обслуживании не более q других заявок. Подкласс $\{q\}$-расписаний будем обозначать $\Phi\{q\}$.

Рассмотрим задачу синтеза оптимального $\{q\}$-расписания:

$$\min_{\rho\in\Phi\{q\}} W(\rho). \quad (3.6)$$

Состояния системы обслуживания – это тройки вида (t, γ, U), где t – очередной момент принятия решения по загрузке процессора (в момент t процессор считается свободным); γ – порядковый номер-наименование первой (по возрастанию номеров) необслуженной заявки; U – множество заявок, которые по состоянию на данный

момент уже обогнали в обслуживании заявку γ, количество заявок в этом множестве не больше q ($|U| \le q$).

Пусть $\Sigma_q(t, \gamma, U)$ обозначает минимальную величину суммарного штрафа по всем необслуженным до момента времени t заявкам для системы, находящейся в состоянии (t, γ, U). При этом считается, что возможными являются только расписания из множества $\Phi\{q\}$.

Для состояния (t, γ, U) через $Q(t, \gamma, U)$ обозначим совокупность всех необслуженных заявок; в это множество входят: заявка γ и все заявки $\gamma + i$, не входящие в совокупность U, здесь $i \in \{1, 2, \ldots, n - \gamma\}$. Для произвольного множества заявок M через $\nu(M)$ обозначим минимальный из номеров-наименований заявок этого множества. Через $Y(M)$ обозначим совокупность, заявок i, $i > \nu(M)$, не входящих в множество M.

Рассмотрим общую ситуацию, когда множество $Q(t, \gamma, U)$ состоит более чем из одной заявки. Если при этом $|U| < q$, то на немедленное обслуживание можно взять любую заявку из совокупности $Q(t, \gamma, U)$; если же $|U| = q$, то на немедленное обслуживание в рассматриваемом состоянии следует взять заявку γ. Получаем

$$\Sigma_q(t, \gamma, U) = \min_{i \in Q(t,\gamma,U)} \{a(i)[\max(t, t(i)) + \tau(i) - t(i)] + \Sigma_q(t + \tau(i), \nu(Q(t,\gamma,U)\backslash\{i\}), Y(Q(t,\gamma,\mathbf{x})\backslash\{i\})\} \quad (3.7)$$

для случая $|U| < q$;

$$\Sigma_q(t, \gamma, U) = a(\gamma)[\max(t, t(\gamma)) + \tau(\gamma) - t(\gamma)] + \Sigma_q(t + \tau(\gamma), \nu(Q(t,\gamma,U)\backslash\{\gamma\}), \mathbf{Y}(Q(t,\gamma,\mathbf{x})\backslash\{\gamma\})\} \quad (3.7')$$

для случая $|U| = q$.

Если множество $Q(t, \gamma, U)$ состоит из единственной заявки, в таком случае это заявка γ, то

$$\Sigma_q(t, \gamma, U) = a(\gamma)[\max(t, t(\gamma)) + \tau(\gamma) - t(\gamma)] \quad (3.7'')$$

Примем дополнительное предположение, что в рассматриваемом классе задач продолжительность обслуживания процессором любой заявки ограничена некоторой константой. В таком случае в процессе вычислений по рекуррентным соотношениям (3.7) - (3.7") аргумент t функции $\Sigma_q(t, \gamma, U)$ принимает линейно зависящее от n число различных значений; аргумент γ принимает значения из множества $\{1, 2, \ldots, n\}$; для возможных значений аргумента U имеется не более n^q вариантов. При вычислении каждого следующего значения функции число выполняемых элементарных операций по верхней оценке линейно зависит от параметра n. Получаем, что верхней оценкой вычислительной сложности алгоритма решения задачи (3.6), основанного на соотношениях (3.7) - (3.7"), является

$$C\, n^{q+3},$$

где C – константа, не зависящая от значений параметров n и q.

Частная модель 5. Расписание ρ назовем *(d,q)-расписанием*, если при его реализации произвольная заявка i может быть обогнана в обслуживании только

некоторыми заявками из совокупности $\{i + 1, i + 2, ..., i + d\}$ и не более чем q другими заявками. В рамках рассматриваемой модели допустимыми считаются только *(d, q)*-расписания. Класс *(d, q)*-расписаний будем обозначать $\Omega(d,q)$.

Как очевидно, класс *(d, 0)*-расписаний совпадает с классом *d*-расписаний, а класс *(0, q)*-расписаний совпадает с классом $\{q\}$-расписаний.

Пусть t – произвольный момент принятия решения, S – совокупность заявок, которые по состоянию на момент времени t поступили и ожидают обслуживания. Как и выше, для произвольного множества заявок M через $\nu(M)$ мы обозначаем минимальный из номеров-наименований заявок этого множества. Через $\eta(t)$ обозначим максимальное значение i, для которого $t(i) \le t$, т.е. $\eta(t)$ – последняя из поступивших в систему по состоянию на момент времени t заявок. Пусть

$$M(t, S, d) = \{j \mid \nu(S) + d < j \le \eta(t)\}.$$

Расписание ρ является *(d,q) -расписанием*, если при его реализации в любой момент времени t в множестве $M(t, S, d)$ оказывается не более чем q завершенных обслуживанием заявок.

Рассмотрим задачу

$$\min_{\rho \in \Omega(d,q)} W(\rho) . \qquad (3.9)$$

При реализации (d, q)-расписания в любой момент принятия решения t совокупность всех оставшихся пока необслуженными заявок потока Q можно полностью описать $(d + q + 1)$-мерным вектором $\boldsymbol{X(Q)} = (x_1, x_2,...,x_{d+1}, x^1, x^2, ..., x^q)$, в котором: $x_1 = \nu(Q)$; координаты $x_2, x_3, ... ,x_{d+1}$ – булевы, при этом x_j равно единице тогда и только тогда, когда заявка $\nu(Q)+ j - 1$ уже обслужена или заявки с данным текущим номером не существует в силу того, что $\nu(Q) + j - 1 > n$; координаты $x^1, x^2,..., x^q$ перечисляют номера завершенных обслуживанием заявок из совокупности $M(t, S, d)$. Если в совокупности $M(t, S, d)$ обслужено только s заявок, где $s < q$, то $x^{s+1} = x^{s+2} = ... = x^q = 0$. Вектор $\boldsymbol{X(Q)}$ будем называть вектором-описанием. Доопределим $\boldsymbol{X}(\varnothing)$ как нуль-вектор. Если $\boldsymbol{X}$ – вектор-описание, то пара $(\boldsymbol{X}, t)$ – это состояние системы в момент времени t. Отметим, что по паре $(\boldsymbol{X}, t)$ совокупность не обслуженных по состоянию на момент времени t заявок Q определяется однозначно: $Q = Q(\boldsymbol{X}, t)$.

Пусть $\Psi_{dq}(\boldsymbol{X}, t)$ обозначает минимальную величину суммарного штрафа по всем необслуженным до момента времени t заявкам для системы, находящейся в состоянии $(\boldsymbol{X}, t)$. При этом считается, что возможными являются только расписания из множества $\Omega(d,q)$.

Начальному состоянию системы соответствует вектор-описание $\mathbf{X}(\mathrm{V}(0))$. Поэтому

$$\min_{\rho \in \Omega(d,q)} W(\rho) = \Psi_{dq}(\boldsymbol{X}(V(0)), 0).$$

Пусть $Q_{dq}(\boldsymbol{X}, t)$ – множество заявок, совпадающее с $Q(\boldsymbol{X}, t)$, если последняя координата вектора $\boldsymbol{X}$ нулевая, и состоящее только из заявок множества $Q(\boldsymbol{X}, t)$ с номерами не больше $\nu(Q(\boldsymbol{X}, t))+ d$ в противном случае. При синтезе (d, q)-расписания в ситуации, определяемой парой $(\boldsymbol{X}, t)$, выбор очередной направляемой на процессор заявки осуществляется только из совокупности $Q_{dq}(\boldsymbol{X}, t)$.

Соотношение динамического программирования для решаемой задачи (3.9) записывается следующим образом:

$$\Psi_{dq}(\boldsymbol{X}, t) = \min_{i \in Q_{dq}(X,t)} \{ a(i)[\max(t, t(i)) + \tau(i) - t(i)] + \Psi_{dq}(\boldsymbol{X}(Q(\boldsymbol{X}, t)) \setminus \{i\}), t + \tau(i))\} \qquad (3.10)$$

При этом

$$\Psi_{dq}(\boldsymbol{X}(\varnothing), t(n) + \theta) = 0 \qquad (3.11)$$

для любого $\theta > 0$.

Реализация определяемой рекуррентными соотношениями (3.10) - (3.11) вычислительной процедуры осуществляется в порядке убывания значений аргумента t, процесс счета заканчивается отысканием величины $\Psi_{dq}(\boldsymbol{X}(V(0)), 0)$ – оптимального значения критерия в рассматриваемой задаче (3.9). Отметим, что подсчет по формуле (3.10) каждого отдельного значения функции $\Psi_{dq}(\boldsymbol{X}, t)$ выполняется в линейно зависящем от n времени. Число рассматриваемых $(d + q + 1)$-мерных векторов-состояний $\boldsymbol{X}$ ограничено сверху величиной $n^{q+1}2^d$, поскольку первая и q последних координат этого вектора принимают значения из множества $\{1, 2, \ldots, n\}$, а остальные координаты булевы. В итоге получаем, что общее число рассматриваемых наборов значений аргументов функции Ψ_{dq} имеет порядок $n^{q+2}2^d$, а верхней оценкой временной вычислительной сложности изложенного алгоритма решения задачи (3.9) оказывается $O(n^{q+3}2^d)$.

Качество оптимального (d, q)-расписания обслуживания входного потока заявок зависит от выбора значений параметров: чем больше d и q, тем шире область возможных дисциплин обслуживания, что позволяет синтезировать расписания меньшей суммарной стоимости. При этом увеличение значения q на единицу предпочтительнее такого же увеличения значения d, ибо в большей степени расширяет класс допустимых расписаний. Вместе с тем, повышение значения q на единицу сопровождается ростом вычислительной сложности предложенного алгоритма в n раз, в то время как повышение на единицу значения d увеличивает число выполняемых элементарных операций лишь в два раза.

Сейчас обратимся к еще одной массовой задаче – классической задаче о ранце (*КЗР*); рассмотрим вопрос выделения в рамках этой общей модели полиномиально разрешимых частных классов задач.

Для решения *КЗР* в главе 1 был описан алгоритм $\boldsymbol{A}_{кзр}$ с верхней оценкой числа выполняемых элементарных операций CnW, где n – число имеющихся предметов, W – максимально допустимая величина суммарного веса помещаемых в ранец предметов, а C – константа, не зависящая от конкретных значений n и W. Внешняя простота приведенной оценки не свидетельствует о разрешимости задачи в полиномиально зависящем от объема входной информации времени. Ясно, что увеличение максимально допустимого суммарного веса помещенных в ранец предметов в 10^k раз влечет за собой увеличение длины записи параметра W в составе входной информации по задаче только на k десятичных разрядов. В то же время приведенная оценка числа выполняемых алгоритмом $\boldsymbol{A}_{кзр}$ элементарных операций возрастает в 10^k раз. Классическая задача о ранце относится к числу *NP*-трудных, ибо к ней за полиномиально зависящее от объема входной информации время сводится *NP*-полная задача "Разбиение" (способ построения по исходным данным задачи "Разбиение" соответствующей *КЗР* дан в главе 1).

Через $K_{вес}[Q]$ обозначим подкласс задач о ранце с n предметами, в которых вес каждого предмета не превосходит $Q(n)$, здесь $Q(n)$ – многочлен от одной переменной с неотрицательными коэффициентами. Так как суммарный вес всех имеющихся предметов больше W (иначе задача решается тривиальным образом – в ранец следует поместить все предметы), получаем: $W < nQ(n)$. Для задач рассматриваемого подкласса верхней оценкой числа выполняемых алгоритмом $\boldsymbol{A}_{кзр}$ элементарных операций является $Cn^2Q(n)$, эта оценка от числа предметов n зависит полиномиально. Получаем, что задачи любого подкласса $K_{вес}[Q]$, где $Q(n)$ – произвольная полиномиальная монотонно возрастающая функция от одной переменной, разрешимы в полиномиальном времени.

Изложим еще один способ решения классической задачи о ранце. Предлагаемый алгоритм $\boldsymbol{B}_{кзр}$ основан на последовательном построении списков. Каждая запись любого списка имеет вид $<M, C, V>$, где M – подмножество предметов, C и V – соответственно суммарная стоимость и суммарный вес предметов подмножества M. Список S^0 считается состоящим из одной записи: $<\varnothing, 0, 0>$. Далее поэтапно строятся списки S^i для значений i последовательно равных $1, 2, \ldots, n$. Составление каждого очередного списка S^i реализуется в два шага. На первом шаге для каждой записи $z = <M, C, V>$ списка S^{i-1} определяется новая запись $z^* = <M \cup \{i+1\}, C+c_{i+1}, V+v_{i+1}>$, записи z и z^* включаются в формируемый рабочий список R^i; результатом первого шага является список R^i, содержащий в сравнении со списком S^{i-1} вдвое больше записей. На втором шаге из списка R^i изымаются: а) все записи, в которых значение третьей компоненты превышает W (нарушено весовое ограничение); б) каждая запись $<M', C', V'>$, для которой в этом же списке R^i имеется запись $<M'', C'', V''>$ такая, что одновременно либо $C'' \geq C'$ и $V'' < V'$, либо $C'' > C'$ и $V'' = V'$. Кроме того, если в списке R^i имеется несколько записей, в которых значения вторых и третьих компонент соответственно совпадают, то все такие записи, кроме одной (любой из них) изымаются. Список S^i определяется как совпадающий с получаемым в результате выполненных изъятий сокращенным списком R^i. Результатом n-го этапа является список S^n. В этом списке отыскиваем запись с наибольшим значением второй компоненты (суммарной стоимости). Пусть таковой является запись $z^{(!)} = <M^{(!)}, C^{(!)}, V^{(!)}>$. Получаем оптимальное решение задачи: в ранец следует поместить предметы подмножества $M^{(!)}$, максимальное значение критерия равно $C^{(!)}$.

ПРИМЕР 3.1. Применением алгоритма $\boldsymbol{B}_{кзр}$ решить задачу:

$$x_1 + 3x_2 + 3x_3 + 6x_4 \rightarrow \max$$

$$x_1 + x_2 + 2x_3 + 4x_4 \leq 6;$$

$$x_i \in \{0,1\}, \quad i = 1,2,3,4.$$

По исходным данным сформулированной задачи строим следующие списки. Список S^0, как всегда, состоит из единственной записи $<\varnothing, 0, 0>$. В списке R^1 две записи: $<\varnothing, 0, 0>$ и $<\{1\}, 1, 1>$. Список S^1 совпадает со списком R^1.

Список R^2 включает четыре записи: $<\varnothing, 0, 0>$, $<\{1\}, 1, 1>$, $<\{2\}, 3, 1>$,$<\{1, 2\}, 4, 2>$. Список S^2 получается из R^2 изъятием второй записи; таким образом, $S^2 = \{<\varnothing, 0, 0>, <\{2\}, 3, 1>,<\{1, 2\}, 4, 2>\}$.

Список R^3 включает шесть записей: $<\varnothing, 0, 0>$, $<\{2\}, 3, 1>$, $<\{1, 2\}, 4, 2>$, $<\{3\}, 3, 2>$, $<\{2, 3\}, 6, 3>$, $<\{1, 2, 3\}, 7, 4>$. Список S^3 получается из R^3 изъятием четвертой записи: $S^3 = \{ <\varnothing, 0, 0>, <\{2\}, 3, 1>, <\{1, 2\}, 4, 2>, <\{2, 3\}, 6, 3>, <\{1, 2, 3\}, 7, 4> \}$.

Список R^4 включает десять записей: $<\varnothing, 0, 0>$, $<\{2\}, 3, 1>$, $<\{1, 2\}, 4, 2>$, $<\{2,3\}, 6, 3>$, $<\{1, 2, 3\}, 7, 4>$, $<\{4\}, 6, 4>$, $<\{2,4\}, 9, 5>$, $<\{1, 2, 4\}, 10, 6>$, $<\{2,3,4\}, 12, 7>$, $<\{1, 2, 3, 4\}, 13, 8>$. При составлении списка S^4 из R^4 по причине нарушения весового ограничения изымаются девятая и десятая записи; далее, в соответствии с условием изъятия б), шестая запись. В итоге получаем: $S^4 = \{<\varnothing, 0, 0>, <\{2\}, 3, 1>, <\{1, 2\}, 4, 2>, <\{2,3\}, 6, 3>, <\{1, 2, 3\}, 7, 4>, <\{2,4\}, 9, 5>, <\{1, 2, 4\}, 10, 6>\}$. Вторая компонента принимает наибольшее значение 10 в шестой записи последнего списка. Оптимальное решение задачи следующее: $x_1 = x_2 = x_4 = 1, x_3 = 0;$ оптимальное значение критерия равно 10.

При реализации алгоритма $\boldsymbol{B}_{кзр}$ число записей в каждом из составляемых списков S^i, $i = \overline{1,n}$, не превышает $C^{(!)}+1$ (действительно, в любом из этих списков нет двух записей с одинаковым значением второй компоненты, которое всегда целочисленно и принадлежит отрезку $[0, C^{(!)}+1]$). Отсюда получаем, что в каждом несокращенном списке R^i имеется не более $2(C^{(!)}+1)$ записей. Каждый список R^i составляется по списку S^{i-1} выполнением линейно зависящего от количества записей в списке S^{i-1} числа элементарных операций. При переходе от несокращенного списка R^i к списку S^i число выполняемых элементарных операций квадратично зависит от количества записей в R^i. Получаем, что число элементарных операций, выполняемых при получении каждого следующего списка имеет верхней оценкой функцию, квадратично зависящую от $C^{(!)}$. С учетом того, что последним составляемым является список S^n, получаем в качестве оценки числа выполняемых алгоритмом $\boldsymbol{B}_{кзр}$ элементарных операций $kn(C^{(!)})^2$, здесь k – константа, не зависящая от значений n и $C^{(!)}$.Если максимальная из стоимостей предметов равна c^*, то $C^{(!)}$ не превосходит nc^* и в качестве верхней оценки временной вычислительной сложности изложенного алгоритма мы получаем kc^*n^3.

Через $K^{ст}[Q]$ обозначим подкласс задач о ранце с n предметами, в которых стоимость каждого предмета не превосходит $Q(n)$, здесь $Q(n)$ – некоторая полиномиальная монотонно возрастающая функция от одной переменной. На задачах этого подкласса алгоритм $\boldsymbol{B}_{кзр}$ дает верхнюю оценку числа выполняемых элементарных операций $kQ(n)n^3$. Получаем, что задачи любого подкласса $K^{ст}[Q]$, где $Q(n)$ – произвольная полиномиальная монотонно возрастающая функция от одной переменной, разрешимы в полиномиальном времени.

3.3. Принципы построения приближенных и эвристических алгоритмов.

Приближенный алгоритм – это алгоритм, порождающий решение, которое в рассматриваемой экстремальной задаче может быть неоптимальным, но обеспечивающим относительное отклонение значения критерия от оптимального не большее, чем предписанная константа. Приближенный алгоритм может считаться приемлемым, если трудоемкость его реализации существенно меньше, чем у любого точного алгоритма, а обеспечиваемое им относительное отклонение от оптимума невелико.

Эвристический алгоритм – это алгоритм, базирующийся на эвристиках, т.е. на соображениях, являющихся результатами накопленного опыта или интуитивных рассуждений. Строгие обоснования у эвристик отсутствуют. Некоторые эвристические алгоритмы одновременно являются приближенными.

Каждый приближенный или эвристический алгоритм строит допустимое, т.е. удовлетворяющее имеющимся ограничениям, решение рассматриваемой задачи оптимизации.

Очевидно, что найти любое из допустимых решений, вообще говоря, проще, чем найти оптимальное решение. Вопрос заключается в том, насколько проще. Ответ зависит от типа решаемой задачи. К числу труднорешаемых оптимизационных задач, в которых произвольное допустимое решение строится очень просто, относятся, например, задача о ранце и задача коммивояжера в их стандартных постановках.

Приведем несколько более сложных примеров. Если в классической задаче о назначениях запрещены некоторые закрепления работ за исполнителями, то вопрос о самом существовании допустимых назначений сводится к решению простейшей задачи о назначениях [6], определяемой системой наложенных запретов. Если в задаче коммивояжера для некоторых пар городов (i,j) запретить непосредственные переходы из i в j, то вопрос существования допустимых решений по своей сложности оказывается эквивалентным *NP*-полной задаче "Гамильтонов цикл".

В некоторых случаях задача синтеза допустимого решения по вычислительной сложности оказывается соизмеримой с задачей построения оптимального решения. Допустим, мы имеем алгоритм $\boldsymbol{A}_{Ц}$ синтеза допустимого решения для задачи целочисленного линейного программирования (ЦЛП); в случае отсутствия в рассматриваемой задаче ЦЛП допустимых решений, алгоритм дает ответ "$\varnothing$". Пусть $\boldsymbol{ZC}$ – произвольная индивидуальная задача ЦЛП, оптимальное значение критерия которой принадлежит отрезку $[A, B]$, числа A и B – целые; для определенности считаем, что $\boldsymbol{ZC}$ – задача максимизации, ее критерий обозначаем $L(X)$. Далее через $\boldsymbol{ZC}[P, Q]$ будем обозначать задачу, получаемую из исходной задачи добавлением линейных ограничений $P \le L(X) \le Q$. Пусть C^0 – ближайшая к середине отрезка $[A, B]$ его целочисленная точка. Применением алгоритма $\boldsymbol{A}_{Ц}$ строим допустимое решение задачи $\boldsymbol{ZC}[C^0, B]$. Если такого решения не существует, то оптимальное значение критерия задачи $\boldsymbol{ZC}$ лежит на отрезке $[A, C^0 - 1]$; если решение имеется, обозначим его X^0 , то оптимальное значение критерия задачи $\boldsymbol{ZC}$ лежит на отрезке $[L(X^0), B]$. И в том и другом случае нам удается уменьшить длину отрезка, в котором локализовано искомое оптимальное значение критерия не менее, чем в два раза. Далее находим целочисленную точку C^1, ближайшую к середине полученного отрезка локализации и применением алгоритма $\boldsymbol{A}_{Ц}$ будем строить допустимое решение полученной задачи $\boldsymbol{ZC}[C^1, C^0 - 1]$ или $\boldsymbol{ZC}[C^1, B]$. Так определяется итерационный процесс, на каждом шаге которого длина отрезка, содержащего искомое оптимальное значение критерия, уменьшается не менее чем в два раза. Выполнив не более $log_2(B - A)$ итераций, мы определим, что искомое значение критерия принадлежит некоторому отрезку единичной длины $[h, h+1]$. Далее, применяя алгоритм $\boldsymbol{A}_{Ц}$ к задачам $\boldsymbol{ZC}[h+1, h+1]$ и $\boldsymbol{ZC}[h , h]$, а, возможно, только к первой из них, мы найдем оптимальное решение исходной задачи $\boldsymbol{ZC}$. Задача ЦЛП относится к числу *NP*-трудных, ибо таковым является ее частный случай – классическая задача о ранце. Из гипотезы $P \neq NP$ в предположении, что для любой задачи ЦЛП достаточно просто найти отрезок локализации оптимального значения критерия длины не более чем экспоненциально зависящей от размера входной информации, вытекает, что алгоритм синтеза допустимого решения задачи ЦЛП не может иметь полиномиальной верхней оценки числа выполняемых элементарных операций.

Изложенное показывает целесообразность разделения задач дискретной оптимизации на два класса**:** $\mathbf{K_1}$ – задачи, для которых допустимые решения строить

относительно легко (есть полиномиальные алгоритмы синтеза); $\mathbf{К_2}$ – задачи, для которых допустимые решения строить трудно (нет полиномиальных алгоритмов синтеза). Этот простой вывод для практики оказывается важным. Перспективным следует считать создание приближенных и эвристических алгоритмов для задач класса $\mathbf{К_1}$. Задачи класса $\mathbf{К_2}$ значительно сложнее, в них поиск приближенных решений по трудности сравним с поиском оптимальных планов.

Рассмотрим оптимизационные задачи, в которых процесс синтеза решений состоит из ряда последовательных шагов, причем на каждом шаге осуществляется выбор, вносящий свой вклад в значение оптимизируемого критерия. Один из возможных способов действий заключается в принятии на каждом шаге решения, наилучшим образом изменяющего имеющееся к данному моменту значение критерия. Такого рода алгоритмы синтеза решений именуют *жадными*.

Применяя к определяемой (nxn)-матрицей $A=\{a_{ij}\}$ классической задаче о назначениях (*КЗН*)

$$\max \sum_{i=1}^{n} a_{i\,\pi(i)}$$

жадный алгоритм, мы на первом шаге находим в матрице A наибольший элемент a_{ij} и реализуем закрепление исполнителя i за работой R_j; далее в матрице A', получаемой из A вычеркиванием элементов i-ой строки и j-го столбца вновь находим максимальный элемент и реализуем закрепление определяемого этим элементом исполнителя за соответствующей работой, и т.д. Решая жадным алгоритмом задачу коммивояжера, мы на каждом шаге осуществляем переход в ближайший из возможных городов. При синтезе жадным алгоритмом решения классической задачи о ранце (*КЗР*) упорядочиваем предметы по убыванию их стоимостей (при одинаковой стоимости – по возрастанию весов) и в этом порядке помещаем, если это позволяет сделать весовое ограничение, в ранец.

В построении жадных алгоритмов возможны варианты, зависящие от того, на какие шаги разбивается процесс построения решений. Отличающимся от изложенного выше вариантом жадного алгоритма решения *КЗН* является следующий: на первом шаге выбирается максимальный элемент первой строки матрицы A, и первый исполнитель закрепляется за определяемой этим элементом работой; на втором шаге реализуется наиболее выгодное (из возможных) закрепление за работой второго исполнителя, и т.д.

Рассмотрим класс $\boldsymbol{K^*}$ дискретных оптимизационных задач, определяемых следующим образом: задано конечное множество M, некоторое семейство его подмножеств I и принимающая только неотрицательные значения функция $f(x)$, которая каждому элементу x из M ставит в соответствие его "вес" $f(x)$. Функцию $f(x)$ именуем весовой функцией. Весом произвольного подмножества X множества M называем сумму весов входящих в X элементов. В семействе I требуется найти подмножество наибольшего веса.

В рамках класса $\boldsymbol{K^*}$ можно сформулировать широкий класс дискретных оптимизационных задач (в том числе классическую задачу о назначениях, задачу коммивояжера, задачу о ранце и т.д.). К классу $\boldsymbol{K^*}$ относится и задача отыскания в произвольном связном взвешенном n-вершинном неориентированном графе остова минимального веса (остов n-вершинного графа – это также n-вершинный ациклический связный подграф данного графа; вес остова – сумма весов составляющих его дуг).

Жадный алгоритм для описанной общей модели определяем следующим образом:

1. Полагаем, что Z – пустое множество. Элементы множества M упорядочиваем (нумеруем) по убыванию их весов, m_i – i-ый по полученному порядку элемент множества M, $i=\overline{1,n}$.
2. Полагаем $i = 1$.
3. Определяем, является ли $Z \cup \{m_i\}$ подмножеством некоторого множества Q из I; в случае положительного ответа переходим к п.4, в случае отрицательного – к п.5.
4. Включаем элемент m_i в множество Z.
5. Увеличиваем значение i на единицу.
6. Если $i \neq n+1$, переходим к п.3, в противном случае к п.7.
7. Построенное множество Z считаем решением задачи.

Изложенный алгоритм назовем ***G*-алгоритмом** (от английского greedy – жадный)

Относительно общего класса задач ***K**** поставим вопрос: при каких налагаемых на семейство I условиях ***G*-алгоритм** правильно решает сформулированную экстремальную задачу?

Введем следующее определение.

Матроид – это пара $<M, I>$, где M – произвольное конечное множество, а I – семейство его подмножеств, удовлетворяющее аксиомам:

А1). $\varnothing \in I$ и если $A \in I$ и $B \subseteq A$, то $B \in I$;

А2) Для произвольных $P \in I$, $Q \in I$, таких что $|Q| = |P| + 1$, существует элемент $e \in Q \setminus P$ такой, что $P \cup e \in I$.

Отметим, что условие $\varnothing \in I$ исключает вырожденный случай $I = \varnothing$. Множества семейства I далее будем называть *независимыми*, остальные подмножества совокупности M – *зависимыми множествами матроида* $<M, I>$.

Имеется определенная связь концепции матроида с теорией линейной зависимости. Аксиомы матроида отражают наиболее характерные свойства независимых подмножеств, содержащихся в конечном множестве элементов линейного пространства. Очевидно, что произвольное подмножество элементов из независимого множества линейного пространства независимо (совокупность элементов $e_1, e_2, \ldots, e_n$ линейно независима, если не существует набора скаляров $\lambda_1, \lambda_2, \ldots, \lambda_n$ не все из которых равны нулю такого, что $\lambda_1 e_1 + \lambda_2 e_2 + \ldots + \lambda_n e_n = 0$). Если для линейно независимых подмножеств P, Q линейного пространства имеет место $|Q| = |P| + 1$, то P порождает подпространство размерности $|P|$, которое может содержать не более чем $|P|$ элементов множества Q. Следовательно, существует элемент $e \in Q \setminus P$, не принадлежащий указанному подпространству. Таким образом, множество $P \cup \{e\}$ линейно независимо.

Для произвольного подмножества C из M введем понятие его максимального независимого подмножества. Независимое подмножество A, $A \subseteq C$ именуем *максимальным* в C, если в C не существует независимого подмножества B такого, что $A \subset B$.

Приведем без доказательства следующий результат.

Т е о р е м а 3.2. Пусть M – конечное множество, а I – семейство его подмножеств, удовлетворяющее аксиоме А1. В этой ситуации пара $<M, I>$ является матроидом тогда и только тогда, когда удовлетворяется условие:

А3) Для произвольного подмножества C из M два любых его максимальных подмножества содержат равное число элементов.

Из данной теоремы вытекает, что пара аксиом А1 и А3 образуют эквивалентную паре А1 и А2 систему аксиом для матроидов.

Рангом произвольного подмножества C из M называем число элементов в максимальном независимом подмножестве множества C. Для ранга произвольного подмножества C из M принимаем обозначение r(C). Очевидно, что подмножество C из M независимо тогда и только тогда, когда $r(C)= |C|$. Каждое максимальное независимое подмножество множества M именуется *базой матроида*. Число $r(M)$ – *ранг матроида*.

Очевидно, что две любые базы матроида $< M, I >$ имеют одинаковое число элементов. Каждое независимое множество C из M можно расширить до базы путем поочередного добавления к нему новых элементов, присоединение которых не нарушает независимости, вплоть до момента, когда таких элементов не существует.

Приведем еще один, эквивалентный предыдущим, способ задания матроида. Матроид – это пара $< M, B>$, где где M – произвольное конечное множество, а $\boldsymbol{B}$ – семейство его подмножеств, удовлетворяющее аксиомам:

В1) Никакое множество из семейства $\boldsymbol{B}$ не содержится в другом множестве этого семейства.

В2) Если B_1 и B_2 входят в семейство $\boldsymbol{B}$, то для любого элемента x из B_1 существует элемент y из B_2 такой, что совокупность $(B_1 \setminus x) \cup y$ тоже входит в семейство $\boldsymbol{B}$.

Элементы семейства $\boldsymbol{B}$ – базы матроида.

Рассмотрим несколько примеров матроидов.

1. Пара $<M, \{M\}>$, где M – произвольное конечное непустое множество, является матроидом, единственной базой которого служит само множество M.

2. Пусть L – линейное пространство, $M \subseteq L$ – произвольное конечное непустое подмножество из L, I – множество, элементами которого служат все линейно независимые системы векторов из M и пустое множество. Тогда пара $<M, I>$ является матроидом с набором независимых множеств I. Такой матроид именуется *векторным матроидом*, порожденным множеством векторов M. Базами этого матроида являются все базы множества M. В частности, взяв в качестве M множество векторов, являющихся столбцами (строками) некоторой матрицы B, получаем матричный матроид. Ранг матроида равен рангу матрицы B.

3. Пусть G – произвольный конечный неориентированный граф, M – множество всех его ребер. Подмножество ребер X, $X \subseteq M$, является базой (элементом совокупности B), если это подмножество образует некоторый остов данного графа. Для пары $< M, B >$ аксиомы В1 и В2 выполняются, данная пара образует матроид.

Т е о р е м а 3.3. (Теорема Радо-Эдмондса). Если пара $<M, I >$ – матроид, то для любой весовой функции применением к данной паре G-алгоритма конструируется множество X, являющееся независимым множеством с наибольшим весом. Если же

пара $<M, I>$ не является матроидом, то существует такая весовая функция $f(x)$, что конструируемое применением G-алгоритма множество X не является независимым множеством с наибольшим весом.

Теорема Радо-Эдмондса дает ответ на поставленный для задач класса $\boldsymbol{K^*}$ вопрос об условиях, при которых $\boldsymbol{G}$-алгоритм строит решения, являющиеся оптимальными.

Достаточно интересен следующий вопрос. Пусть в массовой задаче $\boldsymbol{Z}$ жадный алгоритм $\boldsymbol{A}$ строит, вообще говоря, неоптимальные решения (оптимальными они оказываются лишь для некоторых индивидуальных задач - конкретизаций $\boldsymbol{Z}$). По синтезированному алгоритмом решению произвольной индивидуальной задачи-конкретизации $\boldsymbol{Z}_1$ требуется определить, оптимально ли это решение.

В случае, если массовая задача $\boldsymbol{Z}$ – это задача коммивояжера, сформулированный вопрос об оптимальности построенного жадным алгоритмом решения произвольной индивидуальной задачи $\boldsymbol{Z}_1$ не имеет алгоритма получения ответа с полиномиальной верхней оценкой числа выполняемых элементарных операций (считается, что $P \neq NP$). Данный факт прямо следует из доказательств теорем 19.5 и 19.6 монографии [17].

Введем понятие ε-оптимального решения, рассмотрим вопросы синтеза таких решений.

Для произвольной задачи

$$\underset{x \in D}{opt}\, K(X)$$

с обозначаемым через K^* ненулевым оптимальным значением критерия решение X назовем ε - *оптимальным*, если

$$\{|K^* - K(X)|/ K^*\} \leq \varepsilon .$$

Дробь $\{|K^* - K(X)|/ K^*\}$ называем *относительным отклонением критерия* $K(X)$ при решении X от своего оптимального значения.

Если для какой-либо труднорешаемой задачи оптимизации построен эвристический алгоритм, то естественно возникает вопрос, существует ли ε, при котором построенный алгоритм является способом синтеза ε-оптимальных решений. В случае положительного ответа возникает новый вопрос – каково минимальное значение ε, при котором алгоритм сохраняет ε-оптимальность.

Рассмотрим формулируемую следующим образом задачу об упаковке в контейнеры. Задано конечное множество предметов $A = \{a_1, a_2, \ldots, a_n\}$; каждый предмет a_i имеет "размер" $v(a_i)$, где $v(a_i)$ – рациональное число, принадлежащее отрезку [0,1]; требуется найти разбиение множества A на попарно непересекающиеся подмножества $A_1, A_2, \ldots, A_g$ такое, чтобы сумма "размеров" элементов каждого подмножества не превосходила 1 и чтобы значение g (число подмножеств в разбиении) было минимально возможным. Здесь мы считаем, что предметы одного подмножества должны упаковываться в один контейнер "размера" 1 и наша цель – использовать как можно меньшее число контейнеров.

Введенная задача NP-трудна в сильном смысле, ибо в частном случае она дает задачу "3-разбиение". С учетом гипотезы $P \neq NP$, можно утверждать отсутствие для задачи об упаковке в контейнеры даже псевдополиномиального алгоритма синтеза

оптимального решения. Однако имеется несколько весьма простых алгоритмов синтеза для этой задачи ε -оптимальных решений.

Один из таких алгоритмов известен под названием "Первый подходящий (ПП)". Предположим, что имеется бесконечная последовательность контейнеров размера *1*, каждый из них в начальный момент пуст. Предметы множества *A* распределяем по контейнерам согласно следующему простейшему правилу: очередной (по возрастанию индекса) предмет a_i помещается в контейнер с наименьшим номером, у которого сумма размеров уже помещенных в него предметов не превосходит $1 - v(a_i)$. Иными словами, предмет a_i помещается в первый контейнер, куда его поместить можно. Отметим, что алгоритм ПП не начинает заполнять новый контейнер до тех пор, пока можно использовать контейнеры, уже начатые заполнением.

Обозначим через $X_{ПП}$ решение задачи об упаковке в контейнеры, конструируемое изложенным алгоритмом. Число нужных при этом решении контейнеров $K(X_{ПП})$ удовлетворяет неравенству

$$K(X_{ПП}) \leq 2 \sum_{i=1}^{n} v(a_i). \quad (3.14)$$

Действительно, в противном случае в построенном решении обязательно найдутся два непустых, но загруженных менее чем на половину контейнера, что невозможно. То, что указанная граница – наилучшая из возможных, следует из рассмотрения задачи с множеством предметов $A = \{a_1,a_2,...,a_n\}$, где $v(a_i) = (1/2) + \varepsilon$, $i=\overline{1,n}$. Так как никакие два предмета не могут быть помещены в один контейнер, общее число нужных контейнеров равно *n*. В то же время суммарный размер предметов равен $(n/2) + n\varepsilon$; если выбрать ε достаточно малым, суммарный размер окажется сколь угодно близким к $n/2$.

Оценим, как $K(X_{ПП})$ может отличаться от минимально необходимого числа контейнеров K*. Очевидно неравенство

$$K^* \geq \sum_{i=1}^{n} v(a_i). \quad (3.15)$$

Из соотношений (3.14) - (3.15) получаем, что $K(X_{ПП}) \leq 2K^*$. Таким образом, алгоритм ПП для решения задачи упаковки в контейнеры является ε-оптимальным при $\varepsilon = 1$. Более тонкие рассуждения (см. [7]) дают возможность установить ε-оптимальность алгоритма ПП для ε = 7/10.

Очевидно, что алгоритм ПП можно модифицировать, введя, например, такое правило размещения: каждый следующий предмет a_i помещается в тот контейнер, содержимое которого ближе всего к $1 - v(a_i)$, но не превосходит эту величину; если имеется несколько таких контейнеров, то из них выбирается имеющий минимальный номер. Такой алгоритм можно назвать "Наилучший из подходящих (НП)". К сожалению, оказывается, что алгоритм НП не является существенно лучшим, чем алгоритм ПП; характеристики этих алгоритмов практически совпадают.

Сейчас предположим, что предметы совокупности *A* упорядочены не произвольно (как это считалось ранее), а по уменьшению размеров. Процедура применения алгоритма ПП к упорядоченному таким образом списку предметов называется "Первый подходящий в порядке убывания (ППППУ)". Решение, получаемое

этой процедурой, обозначим $X_{ППППУ}$. Тогда $K(X_{ППППУ})$ - число нужных при этом решении контейнеров. Минимально необходимое число контейнеров в рассматриваемой задаче по-прежнему обозначаем K^*. Имеет место следующий результат.

Т е о р е м а 3.4 (Теорема Джонсона). Для любой задачи об упаковке в контейнеры выполняется неравенство

$$K(X_{ППППУ}) \le (11/9)K^*+4.$$

Более того, существуют задачи об упаковке в контейнеры, для которых

$$K(X_{ППППУ}) \ge (11/9)K^*.$$

Сейчас рассмотрим классическую задачу о ранце

$$\sum_{i=1}^{n} c_i x_i \to \max \qquad (3.16)$$

при условиях

$$\sum_{i=1}^{n} v_i x_i \le W; \qquad (3.17)$$

$$x_i \in \{0,1\}. \qquad (3.18)$$

и задачу о ранце с дробимыми предметами, отличающуюся от классической заменой условия (3.18) на

$$x_i \in [0,1]. \quad (3.18')$$

В задаче (3.16)-(3.17)-(3.18') считается, что предметы в ранец можно помещать не только целиком, но и частями. Для этой задачи известен очень простой способ синтеза оптимального решения – алгоритм Данцига [8]. Этот алгоритм заключается в следующем. Для каждого i-го предмета вычисляется показатель $\gamma_i = c_i / v_i$; далее предметы помещаются в ранец последовательно, в порядке убывания показателя γ_i, до тех пор, пока остается выполненным ограничение (3.17); первый предмет, который нельзя поместить в ранец целиком, делится на две части так, что первая (помещаемая в ранец) часть имеет вес, обеспечивающий выполнение условия (3.17) как точного равенства. Вторая часть разделенного на части предмета в ранец не помещается, так же как и все следующие далее предметы.

Для задачи (3.16)-(3.18) алгоритм Данцига может трактоваться как следующая эвристическая процедура: для каждого i- го предмета вычисляем показатель $\gamma_i = c_i / v_i$; помещаем в ранец предмет с наибольшим значением этого показателя (если таких предметов несколько - первый из них по номеру), каждый следующий (по убыванию показателя γ_i) предмет помещается в ранец, если это позволяет сделать ограничение (3.17). Изложенную процедуру именуем эвристическим алгоритмом Данцига (ЭАД).

Рассмотрим следующий пример классической задачи о ранце:

$$2x_1 + Mx_2 \to max$$

$$x_1 + Mx_2 \le M$$

$$x_i \in \{0,1\}, \quad i = 1,2;$$

здесь M – натуральная константа, удовлетворяющая ограничению $M \geq 3$.

Для приведенной задачи ЭАД строит решение $X = (1,0)$; обеспечиваемое этим решением значение критерия равно *2*. В то же время оптимальным является решение $X^* = (0,1)$, обеспечивающее значение критерия M. Так как константа M может быть сколь угодно большой, ЭАД не является ε- оптимальным ни при каком значении ε.

Приближенные решения задачи о ранце можно получать используя некоторые округления исходных данных, жертвуя в точности получаем выигрыш в быстродействии. В качестве иллюстративного примера рассмотрим следующую задачу:

$$299x_1 + 73x_2 + 159x_3 + 221x_4 + 137x_5 + 89x_6 + 157x_7 \rightarrow max$$

при ограничениях

$$4x_1 + x_2 + 2x_3 + 3x_4 + 2x_5 + x_6 + 2x_7 \leq 10;$$

$$x_i \in \{0,1\},\ i = 1, 2, \ldots, 7.$$

Если к данной задаче применить алгоритм $\boldsymbol{B}_{кзр}$, то получим, что оптимальным решением является совокупность предметов с множеством номеров $M^{(!)} = \{1, 2, 3, 6, 7\}$, а оптимальное значение критерия $C^{(!)}$ равно *777*. В процессе реализации алгоритма приходится построить списки, суммарное число записей в которых оказывается равным 91. Естественная идея упрощения заключается в замене нулем младшего разряда в каждом из параметров c_i. В рассматриваемой задаче получаем следующий критерий:

$$290x_1 + 70x_2 + 150x_3 + 220x_4 + 130x_5 + 80x_6 + 150x_7 \rightarrow max.$$

При новой критериальной функции получаем $M^{(!)} = \{1, 3, 4, 6\}$, а оптимальное значение критерия $C^{(!)}$ равно *740*. Здесь при реализации алгоритма $\boldsymbol{B}_{кзр}$ приходится построить списки, суммарное число записей в которых оказывается равным 36, т.е. практически втрое меньше, чем раньше. В то же время отличие в значении критерия около *5*%; если для полученного множества $\{1, 3, 4, 6\}$ вычислить значение первоначальной критериальной функции, то получим *768* (отличие от оптимума порядка *1*%).

В общем случае, если если при округлении отбросить в числах c_i последние t разрядов, то отклонение от оптимального значения критерия не будет превышать $10^t n$. Время, необходимое для работы алгоритма при решении исходной задачи не превосходит kc^*n^3. После отбрасывания t разрядов оценка временных затрат принимает вид $kc^*n^3 10^{-t}$.

Если решение X задачи о ранце с n предметами построено алгоритмом $\boldsymbol{B}_{кзр}$ после того, как в числах c_i отброшены последние t разрядов, разность $K^* - K(X)$ не превышает $10^t n$. При этом K^* не меньше, чем c^*. Получаем, что найденное решение X является ε-оптимальным при некотором $\varepsilon \leq (10^t n) / c^*$. Отсюда

$$c^* \leq (10^t n) / \varepsilon.$$

С учетом последнего неравенства, верхняя оценка числа выполняемых алгоритмом $\boldsymbol{B}_{кзр}$ элементарных операций $kc^*n^3 10^{-t}$ путем замены параметра c^* преобразуется к виду kn^4 / ε.

Будем говорить, что алгоритм является *полиномиальной приближенной схемой (ППС)* для массовой задачи оптимизации Z, если по начальным данным индивидуальной задачи Z_0 и $\varepsilon > 0$ он выдает ε-оптимальное решение этой задачи за

время, ограниченное полиномом (зависящим от ε) от одной переменной – объема входной информации по решаемой задаче.

Таким образом, нами определена *ППС* для решения классической задачи о ранце; полином, зависящий от ε, имеет здесь вид (kn^4) / ε. Полученная оценка полиномиальна относительно n и 1/ε. Это достаточно удачная ситуация, ибо сделанное определение *ППС* допускает полином, в котором наибольшей степенью переменной n является, например, 1/ε.

Алгоритм решения задачи оптимизации *A* именуем *полностью полиномиальной приближенной схемой (ППП С)*, если время его работы ограничено сверху полиномом от двух переменных, объема входной информации по решаемой задаче и 1/ε.

В силу полученной оценки вычислительной сложности, определенная нами *ППС* для решения классической задачи о ранце является *ППП С*.

Т е о р е м а 3.5. В предположении $P \neq NP$ при любом $\varepsilon > 0$ для задачи коммивояжера нет решающей *ППС*.

Положим противное; допустим, что при некотором $\varepsilon_0 > 0$ имеется *ППС*, решающая задачу коммивояжера. Покажем, что в таком случае существует способ решения задачи "Гамильтонов цикл" за полиномиальное время.

По произвольному конечному ориентированному графу *G* с множеством вершин $\{1, 2, \ldots, n\}$ строим (nxn)- матрицу $S=\{s_{ij}\}$, определяющую задачу коммивояжера. Полагаем, что все элементы главной диагонали матрицы *S* нулевые; в случае $i \neq j$ элемент s_{ij} этой матрицы равен *1* тогда и только тогда, когда в графе *G* имеется дуга (i, j); если $i \neq j$ и дуги (i, j) в графе нет, элемент s_{ij} полагаем равным $2+n\varepsilon_0$.

Применим к построенной задаче коммивояжера *ППС*, конструирующую ε_0-оптимальное решение этой задачи. Утверждаем, что этой схемой в построенной задаче коммивояжера будет найдено решение с суммарной длиной обхода *n* тогда и только тогда, когда в исходном графе *G* гамильтонов цикл имеется. Если применением *ППС* в задаче коммивояжера найдено решение с суммарной длиной обхода *n* (решений меньшей длины в построенной задаче существовать не может), то граф G, очевидно, гамильтонов. Докажем справедливость обратного утверждения. Предположим противное, т.е. что граф *G* гамильтонов, а применением *ППС* строится решение задачи коммивояжера с суммарной длиной обхода большей, чем *n*. В таком случае суммарная длина обхода не меньше $n + 1 + n\varepsilon_0$, ибо реализуется по меньшей мере один элементарный переход длины $2+n\varepsilon_0$. Вместе с тем, так как граф *G* гамильтонов, оптимальное значение критерия в построенной задаче коммивояжера равно *n*. Разность между значением критерия для решения, построенного применением *ППС*, и оптимальным его значением оказывается не меньше $n\varepsilon_0 + 1$, а относительное отклонение критерия задачи при найденном *ППС* решении от оптимального его значения оказывается не меньшим $\varepsilon_0 + 1/n$. Но это противоречит утверждению о том, что применяемая *ППС* строит ε_0-оптимальное решение. Получаем, что применением *ППС* в построенной задаче коммивояжера решение с суммарной длиной обхода *n* строится тогда и только тогда, когда в исходном графе *G* гамильтонов цикл имеется. Таким образом, рассматриваемая *ППС* в полиномиальном времени решает проблему определения по произвольному графу, является ли он гамильтоновым. В предположении $P \neq NP$ это невозможно. Методом от противного теорема доказана.

Многие алгоритмы решения оптимизационных задач, в том числе задач комбинаторной оптимизации, основаны на *принципе локального поиска*. Его суть достаточно проста.

Пусть

$$\max_{X \in D} K(X)$$

– произвольная задача дискретной оптимизации. Предполагаем, что для каждого X из D определена окрестность $G(X)$, при этом $X \in G(X)$. Решения, принадлежащие окрестности $G(X)$, это в том либо ином смысле близкие X решения. Алгоритмы локальной оптимизации имеют следующую общую структуру:

1. Находим исходное допустимое решение X_0, $X_0 \in D$; полагаем $k = 0$;
2. По имеющемуся допустимому решению X_k определяем окрестность этого решения $G(X_k)$; перебором принадлежащих окрестности решений находим решение X^*, оптимизирующее значение критерия при условии, что $X \in G(X_k)$. Если $K(X_k) = K(X^*)$, то X_k - локальный оптимум. В противном случае переходим к п.3.
3. Полагаем $k = k + 1$;
4. Полагаем $X_k = X^*$ и переходим к п.2.

Разработка конкретных алгоритмов локального поиска предусматривает несколько принципиальных этапов. Во-первых, нужно определить процедуру определения исходного допустимого решения. Иногда оказывается целесообразным осуществлять поиск, отправляясь из нескольких начальных точек (далее выбирается лучшее из полученных решений). Во-вторых, для рассматриваемой задачи нужно выбрать подходящий способ определения окрестностей; надо иметь в виду, что число точек в каждой совокупности $G(X)$ должно быть относительно небольшим (поиск оптимума в окрестности осуществляется перебором). Вместе с тем, если окрестности слишком малы, то процесс поиска экстремума оказывается малоэффективным.

Отметим, что симплекс-метод решения задач линейного программирования может трактоваться как локальный поиск; совокупность D – вершины многогранника ограничений, окрестность каждой точки X из D включает вершины, смежные с вершиной X.

В задаче коммивояжера понятие k-окрестности (k – натуральное число, фиксирующее размер окрестности, $k \geq 2$) определим следующим образом: гамильтонов цикл C^* принадлежит k-окрестности гамильтонова цикла C, если C^* можно получить из C путем замены не более чем k дуг. Выполненные эксперименты показывают, что использование систем окрестностей при $k = 2$ малоэффективно, при $k = 3$ получаются достаточно хорошие (в среднем) результаты; переход к значению $k = 4$ существенно увеличивает время вычислений и мало улучшает качество получаемых результатов.

3.4. Эвристические алгоритмы для задач синтеза расписаний обслуживания.

В связи с большой вычислительной сложностью многих задач синтеза оптимальных расписаний обслуживания и диктуемыми условиями приложений жесткими ограничениями на продолжительности циклов решения таких задач, целесообразными оказываются эвристические алгоритмы построения расписаний.

Рассмотрим общие схемы некоторых эвристических алгоритмов, прежде всего для введенной в главе 2 канонической задачи однопроцессорного обслуживания потока заявок; наложение каких-либо дополнительных ограничений на модель обслуживания здесь не предполагается.

Следует указать две сферы приложений эвристических алгоритмов: 1) синтез расписаний приемлемого для практического использования качества; 2) получение нижних оценок, которые оказываются необходимыми для точного решения задач синтеза оптимальных расписаний методом ветвей и границ [13]. Вторая сфера предполагает многократное использование эвристического алгоритма в процессе решения одной задачи; он должен обладать достаточным быстродействием, хотя получаемые оценки могут быть относительно грубыми.

В канонической модели однопроцессорного обслуживания заявки потока $R = \{1, 2,..., n\}$ считаются пронумерованными в порядке их поступления ($0 = t(1) \le t(2) \le \ldots \le t(n)$). Дополнительно полагаем, что заявки, поступающие на обслуживание одновременно, получают очередные номера в порядке убывания показателя $\mu(i) = a(i)/\tau(i)$.

Первый эвристический алгоритм тривиален, он основан на принципе «первый пришел – первым обслуживается (ПП-ПО)» Результатом применения алгоритма ПП-ПО, с учетом принятого способа нумерации заявок, всегда является расписание $\rho = (1, 2,..., n)$. Как правило, более качественными, оказываются расписания, конструируемые μ-алгоритмом, предусматривающим, что в каждый момент принятия решения по загрузке освободившегося процессора из совокупности ожидающих обслуживания выбирается заявка с наибольшим значением показателя $\mu = a(i)/\tau(i)$.

Изложим идею следующего эвристического алгоритма, назовем его (p,q)-алгоритмом, здесь p и q – небольшие натуральные числа ($n > p \ge q$). На первом этапе работы алгоритма рассматривается начальный отрезок входного потока (подпоток R_1), содержащий заявки $1, 2,..., p$. Любым точным методом, эффективно работающим благодаря выбранному небольшому значению параметра p, строится минимизирующая суммарный штраф последовательность π_1 обслуживания заявок этого подпотока; начальный, длины q, отрезок последовательности π_1 считаем начальным отрезком синтезируемого расписания обслуживания ρ^* заявок исходного потока $R = \{1, 2, \ldots, n\}$; заявки, нашедшие свое место в расписании ρ^* из потока R изымаются, получаемый в результате поток обозначаем R^1; момент завершения обслуживания заявок, входящих в построенную начальную часть расписания ρ^* обозначим $T(\rho^*,1)$. На втором этапе заявки потока R^1, поступившие не позднее момента $T(\rho^*,1)$, переупорядочиваются в порядке убывания значений показателя $\mu(i)$, получаемый поток обозначаем $R^{(1)}$; далее рассматривается начальный отрезок потока заявок $R^{(1)}$ (подпоток R_2), содержащий p заявок; выбранным точным методом строится минимизирующая суммарный штраф последовательность обслуживания заявок этого подпотока π_2; начальный, длины q, отрезок последовательности π_2 считаем следующим от начала отрезком синтезируемого расписания ρ^* обслуживания заявок потока R; заявки, нашедшие свое место в расписании ρ^* из потока R изымаются, получаемый в результате поток обозначаем R^2; момент завершения обслуживания заявок, входящих в построенную к данному моменту начальную часть расписания ρ^* обозначим $T(\rho^*,2)$. На третьем этапе заявки потока R^2, поступившие не позднее момента $T(\rho^*,2)$, переупорядочиваются в порядке убывания значений показателя $\mu(i)$, получаемый поток обозначаем $R^{(2)}$; далее рассматривается начальный отрезок потока $R^{(2)}$, содержащий p заявок, и т.д., вплоть до момента либо

естественного завершения процесса, либо выполнения после некоторого k-го этапа условия $T(R, k) \geq t(n))$. При реализации последнего варианта заявки, не вошедшие в сформированную часть расписания, приписываются к ней справа в порядке убывания показателя $\mu(i)$.

Легко видеть, что верхней оценкой временной вычислительной сложности изложенного алгоритма (с учетом сортировок) является $Cn\log_2 n$, где C – не зависящая от n константа.

Существенна зависимость множителя C от выбора определяющих качество алгоритма значений параметров p и q. Пусть оптимальные решения получаемых частных задач с p заявками синтезируются методом динамического программирования. Тогда число элементарных операций, выполняемых при решении каждой частной задачи прямо пропорционально $p^2 2^p$. Общее число подлежащих решению задач с p заявками обратно пропорционально q. Получаем, что константа C прямо пропорциональна $p^2 2^p$ и обратно пропорциональна q. Чем больше значение p и меньше значение q, тем выше ожидаемое качество решения и тем больше трудоемкость алгоритма его синтеза.

Реально вместо изложенной канонической (p, q)-процедуры целесообразно применение ее естественных модификаций. Так в случае, когда на $k+1$-ом этапе в совокупность рассматриваемых p заявок подпотока входят только уже поступившие заявки, а первая поступающая позднее момента времени $T(\rho^*, k)$ заявка имеет большее значение показателя $\mu(i)$, то эту заявку следует включить в число перспективных к включению на данном этапе.

Концепция (p,q)-алгоритма играет существенную роль и для периодов с относительно большим горизонтом планирования, когда точной может быть информация о моментах поступления только идущих вначале заявок, для последующих заявок моменты прибытия известны как ожидаемые средние значения. Тогда реализация этапов синтеза (p,q)-алгоритмом расписания обслуживания синхронизируется с поступлением уточненных данных о моментах прибытия последующих заявок. С учетом периодичности поступления таких данных выбираются значения параметров p и q.

Идея следующего эвристического алгоритма, именуемого *алгоритмом вставки*, тоже достаточно проста. На первом этапе определяем оптимальную последовательность π_1 обслуживания заявок *1* и *2* в предположении, что других заявок нет; при этом подсчитываются суммарные штрафы по двум указанным заявкам для обеих возможных последовательностей их обслуживания. На втором этапе: если заявка *3* поступает ранее момента завершения обслуживания первой в π_1 заявки, подсчитываются суммарные штрафы для трех последовательностей, получаемых из π_1 приписыванием заявки *3* к π_1 слева, справа и ее вставкой в промежутке между первым и вторым элементом этой последовательности; если заявка 3 поступает между моментами завершения обслуживания первой и второй заявки последовательности π_1, суммарные штрафы подсчитываются для двух последовательностей, получаемых из π_1 приписыванием заявки *3* справа и ее вставкой между двумя имеющимися элементами последовательности; если заявка *3* поступает не ранее момента завершения обслуживания второй заявки последовательности π_1, приписываем заявку *3* к π_1 справа. Через π_2 обозначаем ту из последовательностей, для которой суммарный штраф минимален. При определении расположения произвольной заявки k составленную на предыдущем этапе последовательность π_{k-2} разбиваем на две последовательные части

π'_{k-2} и π^*_{k-2}, где π'_{k-2} – максимально возможная по числу элементов начальная часть такая, что реализация расписания π'_{k-2} завершается не позднее момента прибытия заявки k; далее из π_{k-2} путем вставки заявки k на место между частями π'_{k-2}, π^*_{k-2} и на все места правее указанного получаем ряд последовательностей обслуживания; для каждой из полученных последовательностей вычисляем суммарный штраф, последовательность с минимальным значением суммарного штрафа обозначаем π_{k-1}. Итогом работы алгоритма является последовательность обслуживания π_{n-1}.

Оценка временной вычислительной сложности алгоритма вставки Cn^2, где C – не зависящая от n константа.

На практике данным алгоритмом удобно пользоваться не только при составлении, но и при корректировке имеющихся расписаний в случаях получения информации о новых, подлежащих обслуживанию, но ранее не включенных в поток заявках.

Отметим возможность построения для канонической задачи однопроцессорного обслуживания алгоритмов локальной оптимизации. При этом k-окрестность произвольного расписания $\boldsymbol{\rho}$ образуют расписания, каждое из которых получается из $\boldsymbol{\rho}$ путем некоторого перераспределения мест между k заявками, в последовательности $\boldsymbol{\rho}$ идущими подряд. Начальный для процедуры локальной оптимизации вариант расписания может быть построен одним из ранее изложенных эвристических алгоритмов (например, (p,q)-алгоритмом или алгоритмом вставки).

Следует также отметить, что идеи, положенные в основу (p,q)-алгоритма и алгоритма вставки легко адаптируются для синтеза расписаний обслуживания потоков заявок в системах с параллельными и последовательными процессорами. Соответствующие вычислительные процедуры могут входить и как отдельные модули в алгоритмы более сложной структуры.

Сейчас рассмотрим сформулированную в главе 2 задачу синтеза расписания обслуживания потока поступающих в дискретном времени заявок $\boldsymbol{R} = \{1, 2,..., n\}$ в системе, состоящей из m идентичных параллельных процессоров $\boldsymbol{\Pi}_j$ ($j = \overline{1,m}$). Для этой задачи в главе 2 записаны рекуррентные соотношения динамического программирования, однако определяемый ими вычислительный процесс очень трудоемок и реально осуществим лишь при малом числе процессоров и небольшом количестве заявок в потоке. Основанная на декомпозиции эвристическая процедура состоит в последовательном выполнении двух этапов.

На первом этапе строится расписание обслуживания $\boldsymbol{\rho}_0$ – "нулевой" вариант искомого решения. При этом используется многошаговая (p, q)-процедура, где p и q – относительно небольшие натуральные числа, $q \le p < n$. При реализации каждого шага этой процедуры с учетом того, что число p подлежащих обслуживанию заявок невелико, можно использовать метод динамического программирования.

В результате выполнении первого этапа мы получаем расписание $\boldsymbol{\rho}_0$, которое разделяет поток заявок $\boldsymbol{R} = \{1, 2,..., n\}$ на m подпотоков, заявки j-го подпотока должны обслуживаться процессором $\boldsymbol{\Pi}_j$ ($j = \overline{1,m}$).

Второй этап – построение расписания обслуживания $\boldsymbol{\rho}_1$ путем решения m канонических задач однопроцессорного обслуживания. В подлежащей решению j-ой задаче процессор $\boldsymbol{\Pi}_j$, должен обслужить заявки j-го подпотока, $j = \overline{1,m}$. Для решения каждой из канонических задач в зависимости от ее размерности, а также требований к быстродействию и точности, может быть применен либо синтезирующий оптимальное

решение метод динамического программирования, либо одна из перечисленных выше эвристических процедур.

ГЛАВА 4. ДИСКРЕТНЫЕ МНОГОКРИТЕРИАЛЬНЫЕ ЗАДАЧИ. МНОГОКРИТЕРИАЛЬНОЕ ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ.

Важной особенностью проблем, возникающих в процессах принятия решений в системах планирования, управления и проектирования, является наличие многих показателей, по которым решения оцениваются. Рассмотренные в предыдущих разделах модели и методы для реально возникающих проблем оказываются недостаточными. В последние десятилетия значительное внимание уделяется изучению дискретных оптимизационных задач в многокритериальных постановках. При этом возникают вопросы, какие решения следует считать целесообразными (оптимально-компромиссными) и как эти решения строить. В данной главе излагается ряд подходов к решению многокритериальных задач, строятся соответствующие алгоритмы. Отметим, что, исходя из исходной информации, единственное целесообразное решение многокритериальной задачи определить, как правило, невозможно. Поэтому в процессах решения многокритериальных задач существенную роль играет *лицо, принимающее решения* (ЛПР). Именно ЛПР определяет тип решающей процедуры и при необходимости назначает ее параметры. В случае, если найдено многоэлементное множество оптимально-компромиссных решений, ЛПР осуществляет выбор одного из них.

4.1 Дискретные многокритериальные задачи, концепция Парето-оптимального решения и схемы компромисса между критериями.

Пусть

$$\max_{\vec{x}\in D}(Q_1(\vec{x}),Q_2(\vec{x}),...,Q_l(\vec{x})) \qquad (4.1)$$

произвольная задача многокритериальной оптимизации. Вектор-функция $\vec{Q}(\vec{x}) = (Q_1(\vec{x}),Q_2(\vec{x}),...,Q_l(\vec{x}))$ отображает область допустимых решений D в множество D_Q из l-мерного пространства критериев: $D_Q = \{\vec{Q} \mid \vec{Q} = (Q_1(\vec{x}),Q_2(\vec{x}),...,Q_l(\vec{x})), где\ \vec{x} \in D\}$; элементы множества D_Q именуются *оценками*. Два допустимых решения задачи (4.1) *эквивалентны*, если их оценки одинаковы. Обозначим через M произвольное множество оценок; оценка $\vec{w} = (w_1, w_2, ..., w_l)$ из M называется *недоминируемой в* M, если в этом множестве не найдется оценки $\vec{w}' = (w_1', w_2', ..., w_l')$ такой, что $w_i' \geq w_i \quad i = \overline{1,l}$, причем по меньшей мере одно из записанных неравенств выполняется как строгое неравенство. Недоминируемые в D_Q оценки именуются *эффективными (по Парето)*. Множество всех эффективных оценок образует обозначаемую через E *область компромиссов* рассматриваемой задачи, $E \subseteq D_Q$. Обратное отображение Q^{-1} совокупности E в

область допустимых решений D определяет полную совокупность *Парето-оптимальных решений*. Очевидно, что любое целесообразное решение многокритериальной задачи должно быть Парето-оптимальным.

Первый подход к решению задачи (4.1) заключается в построении для нее полной совокупности эффективных оценок с одновременным обеспечением возможности восстановления по любой эффективной оценке Парето-оптимального решения, порождающего эту оценку. Подход универсален в следующем смысле: ЛПР получает возможность выбрать любое целесообразное решение, имея полную информацию о его характеристиках. Недостатком данного подхода является высокая вычислительная сложность проблемы синтеза полной совокупности эффективных оценок для задач большой размерности. Одной из причин сложности может быть большое количество эффективных оценок. Приведем два примера, показывающих, что количество эффективных оценок может иметь достаточно большой порядок роста в зависимости от размерности задачи. С этой целью рассмотрим две бикритериальные задачи – о ранце и о назначениях.

Т е о р е м а 4.1. Для любого четного n существует бикритериальная одномерная задача о ранце с n предметами, в которой имеется не менее $2^{n/2}$ различных эффективных оценок.

В конструируемой задаче о ранце

$$K_1(X) = \sum_{j=1}^{n} c_j^1 x_j \to \max; \qquad (4.2)$$

$$K_2(X) = \sum_{j=1}^{n} c_j^2 x_j \to \max; \qquad (4.3)$$

$$\sum_{j=1}^{n} a_j x_j \le b; \qquad (4.4)$$

$$x_j \in \{0;1\}, \quad j = \overline{1,n}, \qquad (4.5)$$

совокупность предметов считаем состоящей из пар (Π_1, Π_2), (Π_3, Π_4), …, (Π_{n-1}, Π_n). Для предметов пары (Π_{2i-1}, Π_{2i}), $i = \overline{1, n/2}$, полагаем $a_{2i} = a_{2i-1} = 10^{i-1}$; $c_{2i-1}^1 = 10^{i-1}$; $c_{2i-1}^2 = 0$; $c_{2i}^1 = 0$; $c_{2i}^2 = 10^{i-1}$. Определяем $b = \sum_{i=1}^{n/2} 10^{(i-1)}$.

Рассмотрим решения $(x_1, ..., x_n)$ сформулированной задачи (предмету Π_j соответствует переменная x_j, $j = \overline{1,n}$), удовлетворяющие условию $x_{2i-1} + x_{2i} = 1$, $i = \overline{1, n/2}$. Множество таких решений обозначим W. Как очевидно, совокупность W содержит $2^{n/2}$ элементов. Каждое решение $(x_1, ..., x_n)$ из W порождает в плоскости критериев оценку $(\beta_{n/2}\beta_{n/2-1}...\beta_1, \overline{\beta}_{n/2}\overline{\beta}_{n/2-1}...\overline{\beta}_1)$, где β_i и $\overline{\beta}_i$ - цифры (система счисления десятичная), определяемые следующим образом:

$$\beta_i = \begin{cases} 0, если\ \ x_{2i} = 1; \\ 1, если\ \ x_{2i} = 0; \end{cases} \qquad (4.6)$$

$$\overline{\beta}_i = \begin{cases} 1, если\ \ x_{2i} = 1; \\ 0, если\ \ x_{2i} = 0; \end{cases} \qquad (4.7)$$

Отметим, что по структуре исходных данных задачи (4.2) – (4.5) для любого решения $X = (x_1,...,x_n)$ выполняется условие:

$$K_1(X) + K_2(X) = \sum_{j=1}^{n} a_j x_j \le b. \qquad (4.8)$$

Оценки, порождаемые решениями из W, удовлетворяют равенству:

$$K_1(X) + K_2(X) = b. \qquad (4.9)$$

Из (4.8) - (4.9) следует, что каждое принадлежащее множеству W решение Парето-оптимально. Если X и Y - различные Парето-оптимальные решения из W, то порождаемые этими решениями оценки различны (см.(4.6)- (4.7)). Таким образом, решениями совокупности W порождается $2^{n/2}$ различных эффективных оценок. Теорема доказана.

Перейдем к рассмотрению бикритериальных задач о назначениях. Каждая такую задачу определяем парой *(nxn)*-матриц $A = \{a_{ij}\}$ *и* $B = \{b_{ij}\}$, элементы которых - целые неотрицательные числа; $a_{ij}(b_{ij})$ трактуется как оценка закрепления исполнителя i за работой r_j по первому (соответственно второму) показателю.

Каждое назначение π (взаимно однозначное отображение множества $\{1, 2,...,n\}$ в себя) закрепляет за исполнителем i работу $r_{\pi(i)}$, $i = 1, 2,...,n$. Назначение π оценивается критериями $Q_1(\pi) = \sum_{i=1}^{n} a_{i\pi(i)}$ и $Q_2(\pi) = \sum_{i=1}^{n} b_{i\pi(i)}$. Критерии считаем максимизируемыми. Возникающая *бикритериальная задача о назначениях (БКЗН)* записывается в виде:

$$\max_{\pi} \{ Q_1(\pi), Q_2(\pi) \} \qquad (4.10)$$

Т е о р е м а 4.2. Для любого натурального n существует *БКЗН*, множество эффективных оценок которой имеет в своем составе $n!$ различных элементов.

Покажем способ построения соответствующей *БКЗН*. Вначале предположим, что $n = 10$. Матрицу $A = \{a_{ij}\}$ определяем следующим образом:

$$a_{1j} = j - 1$$

(в первой строке последовательно стоят одноразрядные числа от 0 до 9);

остальные строки определяются рекуррентно:

$$a_{kj} = 10a_{k-1\,j},\ k = 2, 3, ..., n;\ j = 1, 2, ..., n.$$

Из специфики матрицы A видим, что любое назначение характеризуется числом взятых из первой строки матрицы A единиц, числом взятых из второй строки этой матрицы десятков, числом взятых из третьей строки сотен и т.д. При этом, так как в каждом столбце реализуется только один элемент, сумма выбираемых в матрице A

произвольным назначением десяти элементов представляет собой 10-разрядное число, составленное из цифр 0, 1,... ,9 (каждая цифра в записи числа присутствует ровно один раз, цифра 0 может стоять и в старшем разряде). Очевидно и обратное: любое 10-разрядное число, запись которого является перестановкой цифр 0, 1,...,9 является возможным значением критерия $Q_1(\pi)$; общее число перестановок, а, следовательно, и значений данного критерия,10!.

Максимальным в составе введенной матрицы А является элемент $a_{10\,10}$, равный 9×10^9. Далее, определяя матрицу B, положим для всех возможных значений индексов i и j:

$$b_{ij} = (9\times10^9) - a_{ij}\,.$$

Легко видеть, что при таком определении матриц A и B для любого назначения π имеем:

$$Q_1(\pi) + Q_2(\pi) = 10(9\times10^9) = 9\times10^{10}.$$

Так как для любых двух различных назначений критерий $Q_1(\pi)$ принимает различные значения и так как чем больше величина $Q_1(\pi)$, тем меньше сответствующее значение $Q_2(\pi)$,в построенной *ДКЗН* каждое назначение Парето-оптимально. Оценки, порождаемые различными назначениями не совпадают; общее число эффективных оценок совпадает с числом Парето-оптимальных назначений и равняется 10!. Принцип выполненного доказательства сохраняется и при значениях n, отличных от 10; в качестве основания системы счисления, в которой задаются численные оценки, следует взять n . Теорема доказана.

Второй подход к задаче (4.1) состоит в реализации той либо иной схемы компромисса, сводящей процесс решения многокритериальной задачи к решению одной или нескольких последовательных однокритериальных задач.

В качестве первой схемы укажем *линейную свертку критериев (ЛСК)*. Реализуя *ЛСК*, задачу (4.1) заменяем однокритериальной задачей

$$K(\vec{x}) = \lambda_1 Q_1(\vec{x}) + \lambda_2 Q_2(\vec{x}) + ... + \lambda_l Q_l(\vec{x}) \to \max$$

при условии (4.11)

$$x \in D.$$

Считается, что назначаемые ЛПР и фиксирующие компромисс между критериями коэффициенты свертки удовлетворяют ограничениям:

$$\lambda_i \in (0,1), \quad i = \overline{1,l}\ ;$$

$$\lambda_1 + \lambda_2 + ... + \lambda_l = 1\,.$$

Методом от противного легко доказывается, что оптимальное решение задачи (4.11) в задаче (4.1) является Парето-оптимальным. Обратное, вообще говоря, неверно. Не всякое Парето-оптимальное решение и не всякую эффективную оценку можно получить реализацией линейной свертки критериев. Для примера рассмотрим бикритериальную задачу о ранце

$$\max(10x_1 + 4x_3, 10x_2 + 4x_3) \qquad (4.12)$$

$$x_1 + x_2 + x_3 \le 1\,; \qquad (4.13)$$

$$x_i \in \{0;1\}, \quad i = 1,2,3. \qquad (4.14)$$

В задаче (4.12) – (4.14) имеются три ненулевых допустимых решения: $\vec{x}_1 = (1,0,0)$, $\vec{x}_2 = (0,1,0)$, $\vec{x}_3 = (0,0,1)$; они порождают оценки $(10;0)$, $(0;10)$, $(4;4)$ соответственно. Все эти оценки эффективны, все перечисленные решения Парето-оптимальны. Применением линейной свертки критериев из (4.12) – (4.14) для любого фиксированного $\lambda \in (0,1)$ получаем однокритериальную задачу

$$\max[\lambda(10x_1 + 4x_3) + (1-\lambda)(10x_2 + 4x_3)]$$

$$x_1 + x_2 + x_3 \le 1;$$

$$x_i \in \{0;1\}, \quad i = 1,2,3.$$

Легко проверить, что в этой задаче при $\lambda \in (0;0,5)$ имеется единственное оптимальное решение $\vec{x}_1$, при $\lambda \in (0,5;1)$ - единственное оптимальное решение $\vec{x}_2$, при $\lambda = 0,5$ – два оптимальных решения, $\vec{x}_1$ и $\vec{x}_2$. Таким образом, Парето - оптимальное решение $\vec{x}_3 = (0,0,1)$ и порождаемая этим решением эффективная оценка (4,4) линейной сверткой критериев получены быть не могут.

Пусть M – множество эффективных оценок в некоторой l-критериальной задаче Z. Линейное упорядочение множества M именуется лексикографическим, если для некоторой перестановки $\{i_1, i_2, \ldots, i_l\}$ чисел $1, 2, \ldots, l$ выполняется условие: в случаях, когда произвольная оценка **а** следует в упорядочении раньше оценки **в**, то либо i_1-ая координата оценки **а** больше i_1-ой координаты оценки ***в***, либо i_1-ая, i_2-ая,…, i_k-ая координаты этих оценок соответственно совпадают, а i_{k+1}-ая координата оценки **а** больше i_{k+1}-ой координаты оценки **в** (здесь $k \in \{1, 2, \ldots, l-1\}$). Оценка **m** из M называется *крайней*, если в упорядочении, соответствующем некоторой перестановке $\{i_1, i_2, \ldots, i_l\}$ чисел $1, 2, \ldots, l$, эта оценка стоит первой. *Крайними решениями* многокритериальной задачи Z будем называть решения, порождающие крайние оценками.

Отметим, что для любой задачи дискретной оптимизации любая крайняя оценка может быть получена путем решения однокритериальной задачи, получаемой из исходной путем линейной свертки критериев с соответствующим образом подобранными коэффициентами.

Другой типовой схемой компромисса является *лексикографическое упорядочение критериев (ЛУК)*. Данная схема предусматривает, что последовательность, в которой критерии перечислены, определяет их значимость в следующем смысле: каждый предшествующий критерий несравненно важнее любого из перечисляемых за ним. Синтез решения, оптимального при лексикографическом упорядочении критериев, реализуется следующим образом.

Сначала решаем однокритериальную задачу

$$\max_{x \in D} Q_1(x) \qquad (4.15)$$

Обозначим D_1 множество ее оптимальных решений. Если D_1 – одноэлементное множество, то единственное оптимальное решение задачи (4.15) является одновременно оптимальным по принципу ЛУК решением исходной многокритериальной задачи (4.1). В противном случае далее решаем задачу

$$\max_{x \in D_1} Q_2(x) \qquad (4.15_1)$$

Обозначим D_2 множество оптимальных решений задачи (4.15_1). Если D_2 – одноэлементное множество, то единственное оптимальное решение задачи (4.15_1) является одновременно оптимальным по принципу ЛУК решением исходной многокритериальной задачи (4.1). В противном случае поступаем аналогично предыдущему – решаем задачу максимизации значения критерия Q_3 (x) при условии, что $x \in D_2$, и т.д. Максимальное число последовательно решаемых однокритериальных задач (число этапов в процессе поиска решения, оптимального по принципу *ЛУК*) равно l, т.е. числу критериев исходной многокритериальной задачи. Если решение задачи (4.1) определилось в результате выполнения меньшего числа этапов, то оно единственно. В противном случае может оказаться, что оптимальных (при имеющемся лексикографическом упорядочении критериев) решений этой задачи более чем одно; все они эквивалентны.

Отметим, что в задаче с l критериями имеется $l!$ различных схем лексикографического упорядочения. Решения, получаемые при реализации этих схем, являются крайними. Наибольшее число крайних попарно неэквивалентных решений задачи (4.1) равно ($l!$). Каждое крайнее решение, согласно его определению, Парето-оптимально.

Суть *метода последовательных уступок по значению ведущего критерия* поясним сначала на примере двухкритериальной задачи

$$\max_{x \in D} (Q_1(x), Q_2(x)), \qquad (4.16)$$

в которой критерий $Q_1(x)$ считаем ведущим. Реализуя этот метод, сначала находим решение x^*, оптимальное при лексикографическом упорядочении критериев $(Q_1(x), Q_2(x))$. Пусть $(Q_1(x^*), Q_2(x^*)) = (a,b)$. Точка (a,b) – оценка найденного решения. Если данная оценка (первая ее координата – максимально возможное значение критерия $Q_1(x)$) удовлетворяет ЛПР, то x^* принимается за искомое оптимально-компромиссное решение; в противном случае ЛПР назначает уступку Δ_1, $\Delta_1 > 0$, по допустимому значению первого критерия. Далее решается задача

$$\max_{x \in D} Q_2(x).$$

при дополнительном условии (4.17)

$$Q_1(x) \geq a - \Delta_1.$$

Пусть оптимальное решение x^{**} задачи (4.17) в исходной задаче(4.16) имеет оценку (a_1,b_1). Как очевидно, $b_1 \geq b$. Если данная оценка удовлетворяет ЛПР, то x^{**} принимается за искомое оптимально-компромиссное решение задачи (4.16); в противном случае ЛПР назначает новую уступку Δ_2, где $\Delta_2 > \Delta_1$, по допустимому значению первого критерия. Далее решается задача

$$\max_{x \in D} Q_2(x).$$

при дополнительном условии (4.18)

$$Q_1(x) \geq a - \Delta_2.$$

Пусть оптимальное решение x^{***} задачи (4.18) в исходной задаче(4.16) имеет оценку (a_2,b_2). Как очевидно, $b_2 \geq b_1$. Если данная оценка удовлетворяет ЛПР, то x^{***} принимается за искомое оптимально-компромиссное решение задачи (4.16); в противном случае ЛПР назначает новую уступку $\hat{\hat{m}}_2$, $\hat{\hat{m}}_2 > \hat{\hat{m}}_1$, по допустимому значению первого критерия, и т.д.

Описанный процесс продолжается вплоть до отыскания решения с оценкой, устраивающей ЛПР, или вплоть до ситуации, когда дальнейшие уступки по допустимому значению первого критерия становятся невозможными.

Отметим, что полученное в результате реализации описанной схемы решение, условно обозначим его x^0, не обязательно Парето-оптимально. Парето-оптимальное решение задачи (4.1) мы получим, если решим задачу

$$\max_{x \in D} Q_1(x)$$

при дополнительном условии (4.19)

$$Q_2(x) = Q_2(x^0).$$

В п.4.2 изложенная технология метода последовательных уступок будет использована для синтеза полных совокупностей эффективных оценок в задачах о назначениях с двумя неоднотипными критериями.

Применение метода последовательных уступок к общей задаче (4.1), где критерий $Q_1(\vec{x})$ – ведущий, реализуется в виде следующей многошаговой процедуры. Сначала строится совокупность эффективных оценок, имеющих первой координатой $Q_1^{\max}$, т.е. наибольшее значение критерия $Q_1(\vec{x})$ в задаче (4.1). Если некоторую оценку $\vec{Q}'$ из построенной совокупности ЛПР считает приемлемой, то порождающее ее решение $\vec{x}'$ объявляется оптимально-компромиссным решением задачи (4.1). В противном случае ЛПР, делая последовательно возрастающие уступки δ_1, δ_2, δ_3 и т.д., анализирует при реализации каждого i-го шага на предмет приемлемости оценки, содержащиеся в множестве $E(\delta_i)$, определяемом следующим образом: оценка $(Q_1,Q_2,...Q_l)$ входит в $E(\delta_i)$, если она эффективна в (4.1) и удовлетворяет дополнительному условию:

$$Q_1^{\max} - Q_1 \leq \delta_i .$$

Назначение монотонно возрастающих уступок продолжается вплоть до ситуации, когда в очередном множестве $E(\delta_i)$ окажется оценка $\vec{Q}$, которая удовлетворит ЛПР; порождающее ее решение $\vec{x}$ объявляется оптимально-компромиссным. Назначаемые в процессе реализации метода уступки считаются ограниченными сверху обозначаемой через $\delta_{\max}$ константой.

Метод главного критерия заключается в оптимизации значения наиболее важного (с точки зрения ЛПР) критерия при условии, что остальные критерии принимают значения, не меньшие предписанных пороговых величин. Вводим нумерацию, при которой $Q_1(\vec{x})$ - главный критерий. Тогда задача (4.1) сводится к однокритериальной задаче

$$\max_{x \in D} Q_l(x)$$

при дополнительных условиях (4.20)

$$Q_k(\vec{x}) \geq h_k,\ k=2,3,...,l;$$

здесь $h_2,h_3,...,h_l$ - заданные соответственно для второго, третьего,…, l- го критериев пороги.

Метод идеальной точки реализуется в ситуации, когда в пространстве критериев введена некоторая метрика, а лицом, принимающим решения, в этом же пространстве определена идеальная точка $I=(I_1,I_2,...,I_l)$. Оптимальным считается решение $\vec{x}^*$, порождающее оценку $\vec{Q}(\vec{x}^*)=(Q_1(\vec{x}^*),Q_2(\vec{x}^*),...,Q_l(\vec{x}^*))$, наименее удаленную от идеальной точки.

В реализациях метода часто считается, что расстояние $r(\vec{x},\vec{y})$ между произвольными точками $\vec{x}=(x_1,x_2,...,x_l)$ и $\vec{y}=(y_1,y_2,...,y_l)$ пространства критериев определяется по формуле:

$$r(\vec{x},\vec{y})=\max_{\alpha=\overline{1,l}}|x_\alpha - y_\alpha|$$

Обычно предполагается, что идеальная точка назначена таким образом, что все разности $I_\alpha - Q_\alpha(\vec{x}^*)$ неотрицательны.

4.2 Синтез Парето - оптимальных решений методом последовательных уступок.

В многокритериальных задачах дискретной оптимизации полные совокупности эффективных оценок вместе с порождающими их Парето - оптимальными решениями можно строить руководствуясь методом последовательных уступок. При этом реализация уступок нередко оказывается достаточно сложной. Ниже этот метод применяется к бикритериальным задачам о назначениях двух видов.

Предполагается, что читатель владеет изложенными, например, в [6, 10] алгоритмами решения классической однокритериальной задачи о назначениях (КЗН) и задачи о назначениях с максиминным критерием. Отметим, что верхними оценками числа элементарных операций, выполняемых этими алгоритмами при решении задач размерности $n \times n$, являются Cn^3 (для алгоритма решения КЗН) и $Cn^2 lgn$ (для алгоритма решения максиминной задачи о назначениях), здесь C – некоторая не зависящая от n натуральная константа.

Приведенный в [6, 10] алгоритм одновременно с КЗН решает и задачу, ей двойственную. Решив КЗН, определяемую какой-либо *(nxn)*-матрицей A, мы находим: оптимальное назначение, оптимальное значение критерия, а также являющиеся оптимальными значениями двойственных переменных величины ξ_i, $i=\overline{1,n}$, и η_j, $j=\overline{1,n}$. Указанные величины ξ_i, $i=\overline{1,n}$, именуются потенциалами соответствующих строк матрицы A; величины η_j, $j=\overline{1,n}$, именуются потенциалами соответствующих столбцов матрицы A. Числовой элемент a_{ij} матрицы A потенциален, если он равен сумме потенциалов строки и столбца, в пересечении которых этот элемент расположен, т.е.

если $a_{ij} = \xi_i + \eta_j$. Известно[6, 10], что назначение π оптимально в КЗН тогда и только тогда, когда все элементы $a_{i\pi(i)}$ матрицы A, $i=\overline{1,n}$, потенциальны.

Сначала рассмотрим *задачу о назначениях с двумя неоднотипными критериями.* Эта задача определяется парой *(nxn)*-матриц $A = \{a_{ij}\}$ *и* $B = \{b_{ij}\}$ с целыми неотрицательными элементами. Каждое допустимое назначение π характеризуется критериями $K_1(A,\pi) = \sum_{i=1}^{n} a_{i\pi(i)}$ и $K_2(B,\pi) = \min_i b_{i\pi(i)}$. Оба критерия считаем максимизируемыми. Таким образом, рассматриваемая задача записывается в виде:

$$\max_{\pi} \{K_1(A,\pi), K_2(B,\pi)\}. \qquad (4.21)$$

При лексикографическом упорядочении критериев (4.21), где критерий $K_1(A,\pi)$ – ведущий, возникает следующая задача.

Задача 1. Максимизировать значение критерия $K_2(B,\pi)$ на множестве назначений, оптимальных по критерию $K_1(A,\pi)$

Далее множество назначений, обращающих $K_1(A,\pi)$ в максимум, будем обозначать $S_1(A)$.

В случае, когда критерии лексикографически упорядочены противоположным образом, т.е. ведущим является критерий $K_2(B,\pi)$, возникает следующая задача.

Задача 2. Максимизировать значение критерия $K_1(A,\pi)$ на множестве назначений, оптимальных по критерию $K_2(B,\pi)$.

Далее множество назначений, обращающих $K_2(B,\pi)$. в максимум, будем обозначать $S_2(B)$.

Рассмотрим сначала задачу 1. Решив КЗН, определяемую матрицей A, находим: оптимальное назначение π_1, оптимальное значение критерия $a_1 = K_1(A,\pi_1)$ и оптимальные значения двойственных переменных ξ^1_i, $i=\overline{1,n}$, и η^1_j, $j=\overline{1,n}$, именуемые потенциалами соответствующих строк и столбцов матрицы A. Числовой элемент a_{ij} этой матрицы потенциален, если $a_{ij} = \xi^1_i + \eta^1_j$. Известно, что назначение π оптимально в КЗН тогда и только тогда, когда все элементы $a_{i\pi(i)}$, $i=\overline{1,n}$, потенциальны. Определим (*nxn*)-матрицу $B^1 = \{b^1_{ij}\}$ следующим образом:

$$b^1_{ij} = \begin{cases} b_{ij} + Q, & \text{если } a_{ij} = \xi^1_i + \eta^1_j; \\ 0, & \text{если } \xi^1_i + \eta^1_j > a_{ij}, \end{cases} \qquad (4.22)$$

здесь Q - достаточно большая константа, можно положить $Q = \max_{i,j} b_{ij} + 1$.

Лемма 4.1. Назначение π^* оптимально для задачи 1 тогда и только тогда, когда оно является оптимальным решением задачи максимизации критерия $K_2(B^1,\pi)$.

Необходимость. Пусть назначение π^* оптимально для задачи 1. Так как $\pi^* \in S_1(A)$, то имеет место неравенство $b^1_{i\,\pi^*(i)} \geq Q$, $i=\overline{1,n}$. Если существует решение π^{**} такое, что $\min_i b^1_{i\;\pi^{**}(i)} > \min_i b^1_{i\;\pi^*(i)}$, то π^{**} – принадлежащее $S_1(A)$ назначение, обеспечивающее критерию $K_2(B,\pi)$ значение большее, чем $K_2(B,\pi^*)$. Но это противоречит оптимальности решения π^* для задачи 1. Необходимость доказана.

Достаточность. Так как ξ^1_i, $i=\overline{1,n}$, и η^1_j, $j=\overline{1,n}$, - потенциалы строк и столбцов матрицы A, соответствующие оптимальному для задачи максимизации $K_1(A,\pi)$ назначению, решение π^* задачи максимизации $K_2(B^1,\pi)$ обладает свойством $(\forall i)[b^1_{i\pi^*(i)} \geq Q]$, что обеспечивает его оптимальность по ведущему критерию. Элементы матрицы B^1 определены таким образом, что решая задачу максимизации $K_2(B^1,\pi)$, мы фактически отыскиваем такое оптимальное по критерию $K_1(A,\pi)$ назначение, которое максимизирует на множестве $S_1(A)$ значение критерия $K_2(B,\pi)$. Достаточность доказана.

Перейдем к рассмотрению задачи 2. Обозначим через $S_2(B/A)$ совокупность назначений, являющихся ее оптимальными решениями.

Пусть назначение π^I максимизирует критерий $K_2(B,\pi)$. Введем (nxn)-матрицу $A^1 = \{ a^1_{ij} \}$, где

$$a^1_{ij} = \begin{cases} a_{ij} + W, \text{если } b_{ij} \geq K_2(B,\pi^1) ; \\ 0 \text{ в противном случае;} \end{cases} \qquad (4.23)$$

здесь W - достаточно большая натуральная константа, можно положить эту константу равной сумме всех максимальных по строкам элементов матрицы A, увеличенной на единицу.

Лемма 4.2. Назначение π' оптимально для задачи 2 тогда и только тогда, когда оно является оптимальным решением задачи максимизации критерия $K_1(A^1, \pi)$.

Доказательство этого утверждения (в обе стороны) легко осуществляется методом от противного.

Леммы 4.1 и 4.2 являются обоснованием алгоритмов решения задач 1 и 2 соответственно. Для решения задачи 1 следует:

1) решить КЗН, определяемую матрицей A, определив при этом потенциалы ее строк и столбцов;

2) пользуясь формулой (4.22), определить (nxn)-матрицу $B^1 = \{b^1_{ij}\}$;

3) решить задачу максимизации критерия $K_2(B^1,\pi)$, т.е. определяемую матрицей B^1 максиминную задачу о назначениях; полученное решение является оптимальным для задачи 1.

Для решения задачи 2 следует:

1) решить задачу максимизации критерия $K_2(B, \pi)$, т.е. максиминную задачу о назначениях, определяемую матрицей B; полученное оптимальное назначение обозначить π^1;

2) пользуясь формулой (4.23), определить (nxn)-матрицу $A^1 = \{ a^1_{ij} \}$;

3) решить КЗН, определяемую матрицей A^1; полученное назначение является оптимальным для задачи 2.

Легко проверяется, что изложенными способами задача 1 и задача 2 решаются (с учетом верхней оценки числа выполняемых элементарных операций) в кубично зависящем от n времени.

Перейдем к непосредственному рассмотрению вопроса синтеза для задачи (4.21) полной совокупности эффективных оценок вместе с обеспечивающими эти оценки Парето - оптимальными назначениями.

Пусть π – некоторое Парето-оптимальное в задаче (4.21) назначение, причем $K_1(A, \pi) = a$ и $K_2(B, \pi) = b$. Для определения отличного от π Парето-оптимального назначения π', обеспечивающего ближайшее к b, причем большее b, значение критерия K_2, применяем следующую четырехэтапную процедуру ***Ф***:

1) строится (nxn)-матрица $A' = \{ a'_{ij} \}$, где

$$a'_{ij} = \begin{cases} a_{ij} + W, \text{ если } b_{ij} > b; \\ 0 \text{ в противном случае,} \end{cases} \qquad (4.24)$$

здесь W определяется так же, как в (4.23);

2) решается КЗН, определяемая матрицей A', при этом находятся потенциалы ее строк ξ_i, $i=\overline{1,n}$, и столбцов η_j, $j=\overline{1,n}$; в случае, если найденное оптимальное значение критерия КЗН меньше, чем nW, искомого Парето-оптимального назначения не существует; если же оптимальное значение критерия КЗН больше или равно nW, переходим к п.3;

3) строится (nxn)-матрица $B' = \{ b'_{ij} \}$, где

$$b'_{ij} = \begin{cases} b_{ij} + Q, \text{ если } a'_{ij} = \xi_i + \eta_j; \\ 0, \text{ если } \xi_i + \eta_j > a'_{ij}, \end{cases} \qquad (4.25)$$

здесь Q - достаточно большая константа, можно считать $Q = (\max\limits_{i,j} b_{ij}) + 1$.

4) решается задача максимизации критерия $K_2(B', \pi)$, т.е. максиминная задача о назначениях, определяемая матрицей B′; полученное назначение π′ является искомым.

Обладая описанной процедурой ***Ф***, для построения в задаче (4.21) полной совокупности эффективных оценок поступаем следующим образом. Прежде всего, решаем задачи 1 и 2 с лексикографически упорядоченными критериями; пусть их оптимальными решениями являются назначения μ и ν, порождающие в плоскости критериев (K_1,K_2) точки (a_μ, b_μ) и (a_ν, b_ν) соответственно; здесь должны выполняться неравенства $a_\mu \geq a_\nu$ и $b_\mu \leq b_\nu$, данные соотношения имеют место либо одновременно как равенства, либо одновременно как строгие неравенства. Если имеют место равенства $a_\mu = a_\nu$ и $b_\mu = b_\nu$, то построенная точка (a_μ, b_μ) является единственной эффективной в задаче (4.21) оценкой. Если же $a_\mu > a_\nu$ и $b_\mu < b_\nu$, то нами уже найдены две крайние эффективные оценки, порождаемые назначениями μ и ν соответственно. Реализуя процедуру ***Ф*** в отношении Парето - оптимального назначения μ и числа $b = b_\mu$, и далее последовательно применяя ее к получаемым Парето-оптимальным назначениям и обеспечиваемым ими значениям критерия K_2, строим последовательность $\pi_1, \pi_2, \ldots, \pi_m$ Парето - оптимальных назначений вместе с обеспечиваемыми ими эффективными оценками $(a_1, b_1), (a_2, b_2), \ldots, (a_m, b_m)$ соответственно. Здесь m должно быть таким, что $(a_m, b_m) = (a_\nu, b_\nu)$, это условие окончания процесса; общее количество получаемых эффективных оценок равно $m + 1$, для каждой эффективной оценки оказывается построенным обеспечивающее ее Парето-оптимальное назначение.

Несложно убедиться, что по верхней оценке временной вычислительной сложности одноразовое применение процедуры ***Ф*** реализуемо в кубично зависящем от n времени. При синтезе полной совокупности эффективных оценок в задаче (4.21) одноразово решаются задачи 1 и 2, затем (в случае несовпадения оценок (a_μ, b_μ) и (a_ν,

b_ν) не более $n^2 - n$ раз применяется процедура $\boldsymbol{\Phi}$. Таким образом, верхняя оценка временной вычислительной сложности изложенной технологии синтеза полной совокупности эффективных оценок в задаче (4.21) – полином от n пятой степени.

Изложенная технология реализует метод "последовательного увеличения" значений критерия $K_2(B,\pi)$; по сути это то же самое, что метод последовательных уступок по значениям критерия $K_1(A,\pi)$. Применение данного метода особенно целесообразно в ситуациях, когда именно критерий $K_1(A,\pi)$ принят в качестве ведущего и по его значениям можно сделать лишь малое число уступок.

Сейчас рассмотрим ситуацию, когда ведущим является критерий $K_2(B,\pi)$. Пусть π – некоторое Парето-оптимальное в задаче (4.21) назначение, причем $K_1(A,\pi) = a$ и $K_2(B,\pi) = b$. Для определения отличного от π Парето-оптимального назначения π^*, обеспечивающего ближайшее к b, причем меньшее b, значение критерия K_2 (при переходе от π к π^* значение первого критерия должно увеличиться), применяем процедуру $\boldsymbol{\Psi}$, реализуемую следующим образом:

1) в совокупности меньших чем b элементов матрицы B находим максимальный (здесь мы предполагаем, что $b_\mu < b$, иначе искомого Парето-оптимального назначения не существует); величину найденного элемента обозначим b^*;

2) по матрице A и найденному значению b^* строим (nxn)-матрицу $A^* = \{a^*_{ij}\}$, где

$$a^*_{ij} = \begin{cases} a_{ij} + W, \text{если } b_{ij} \geq b^*; \\ 0 \text{ в противном случае;} \end{cases} \qquad (4.26)$$

здесь W определяется так же, как в (4.23);

3) решаем КЗН, определяемую матрицей A^*; в случае, если оптимальное значение критерия в этой КЗН не превосходит $nW + a$, выполненная уступка для появления требуемого Парето-оптимального назначения недостаточна, следует положить $b = b^*$ и вернуться к выполнению этапа 1; если же оптимальное значение критерия КЗН больше $nW + a$, полученное назначение π^* является искомым.

При реализации метода последовательных уступок в ситуации, когда ведущим является критерий $K_2(B,\pi)$, решив задачу 2, сначала по отношению к найденным назначению ν и обеспечиваемой этим назначением оценке (a_ν, b_ν), а далее к получаемым Парето-оптимальным назначениям и эффективным оценкам последовательно применяем процедуру $\boldsymbol{\Psi}$.

Верхняя оценка временной вычислительной сложности основанной на использовании процедуры $\boldsymbol{\Psi}$ технологии синтеза полной совокупности эффективных в задаче (4.21) оценок – полином от n пятой степени.

ПРИМЕР 4.1. Для задачи (4.21), определяемой парой матриц

$$A = \begin{pmatrix} 5 & 8 & 4 & 2 \\ 6 & 9 & 4 & 6 \\ 5 & 3 & 8 & 3 \\ 3 & 5 & 7 & 9 \end{pmatrix} \quad \text{и} \quad B = \begin{pmatrix} 9 & 4 & 7 & 8 \\ 2 & 1 & 9 & 7 \\ 7 & 8 & 8 & 4 \\ 9 & 7 & 4 & 9 \end{pmatrix},$$

найти полную совокупность эффективных оценок. Для каждой такой оценки записать порождающее ее Парето-оптимальное решение.

Сначала найдем крайние, оптимальные при лексикографических упорядочениях критериев, эффективные оценки; построим соответствующие этим оценкам Парето-оптимальные решения. Решая КЗН, определяемую матрицей A, находим потенциалы ее строк $\xi_1 = 5;\ \xi_2 = 6;\ \xi_3 = 8;\ \xi_4 = 9;$ и потенциалы ее столбцов $\eta_1 = 0;\ \eta_2 = 3;\ \eta_3 = 0;\ \eta_4 = 0.$ В первой строке матрицы A потенциальными являются первый и второй элементы; во второй строке – также первый и второй элементы; в третьей строке потенциален третий элемент, в четвертой – четвертый элемент; оптимальное значение критерия 31. По формуле (4.22), здесь $Q = 10$, определяем матрицу

$$B^1 = \begin{pmatrix} 19 & 14 & 7 & 8 \\ 12 & 11 & 9 & 7 \\ 7 & 8 & 18 & 3 \\ 9 & 2 & 4 & 19 \end{pmatrix}.$$

и решаем задачу максимизации критерия $K_2(B^1, \pi)$. Оптимальным оказывается назначение π^*, закрепляющее за исполнителем 1 вторую работу, за исполнителем 2 – первую работу, за исполнителем 3 – третью работу, за исполнителем 4 – четвертую работу. В решаемой бикритериальной задаче полученное назначение π^* Парето - оптимально и порождает крайнюю эффективную оценку $M^* = (31,2)$.

Для отыскания другой крайней оценки действуем следующим образом. Сначала решаем определяемую матрицей B максиминную задачу о назначениях; оптимальное значение критерия оказывается равным 8. Далее, основываясь на формуле (4.23), здесь $W = 35$, строим матрицу

$$A^1 = \begin{pmatrix} 40 & 8 & 4 & 37 \\ 6 & 9 & 39 & 6 \\ 5 & 38 & 43 & 3 \\ 38 & 5 & 7 & 44 \end{pmatrix}$$

Оптимальным в КЗН, определяемой матрицей A^1, оказывается назначение π^{**}, закрепляющее за исполнителем 1 первую работу, за исполнителем 2 – третью работу, за исполнителем 3 – вторую работу, за исполнителем 4 – четвертую работу. В решаемой бикритериальной задаче полученное назначение π^{**} Парето-оптимально и порождает крайнюю эффективную оценку $M^{**} = (21,8)$. Две крайние эффективные оценки и порождающие их Парето-оптимальные решения получены.

Промежуточные эффективные оценки будем строить, основываясь на процедуре Ψ. В качестве исходной эффективной оценки берем (21, 8). В совокупности меньших, чем 8, элементов матрицы B находим максимальный. Это число 7. Пользуясь формулой (4.26), константа W определяется так же, как в (4.23), по матрице A и найденному значению $b^* = 7$ строим (nxn)-матрицу

$$A^* = \begin{pmatrix} 40 & 0 & 39 & 37 \\ 0 & 0 & 39 & 41 \\ 40 & 38 & 43 & 0 \\ 38 & 40 & 0 & 44 \end{pmatrix}.$$

Оптимальное решение КЗН, определяемой этой матрицей, обозначим π_1. Это назначение за исполнителем 1 закрепляет первую работу, за исполнителем 2 – четвертую работу, за исполнителем 3 – третью работу, за исполнителем 4 – вторую работу. В решаемой бикритериальной задаче полученное назначение π_1 Парето-оптимально и порождает эффективную оценку $M_1 = (24,7)$.

Далее в совокупности меньших чем 7 элементов матрицы B находим максимальный; это число 4. Пользуясь формулой (4.26), где $W = 35$, по матрице A и найденному значению $b^* = 4$ строим новую (nxn)-матрицу

$$A^* = \begin{pmatrix} 40 & 43 & 39 & 37 \\ 0 & 0 & 39 & 41 \\ 40 & 38 & 43 & 38 \\ 38 & 40 & 42 & 44 \end{pmatrix}.$$

Оптимальное решение КЗН, определяемой этой матрицей, обозначим π_2. Это назначение за исполнителем 1 закрепляет вторую работу, за исполнителем 2 – четвертую работу, за исполнителем 3 – первую работу, за исполнителем 4 – третью работу. В решаемой бикритериальной задаче полученное назначение π_2 Парето-оптимально и порождает эффективную оценку $M_2 = (26,4)$.

После отыскания оценки M_2 в совокупности меньших, чем 4, элементов матрицы В находим максимальный; это число 2. Эффективная оценка, вторая координата которой равна 2, у нас уже имеется, это $M^* = (31,2)$. Полная совокупность эффективных в рассматриваемой бикритериальной задаче оценок получена, это $M^* = (31, 2)$, $M_2 = (26, 4)$, $M_1 = (24, 7)$, $M^{**} = (21, 8)$. Для всех эффективных оценок выше получены соответствующие Парето-оптимальные решения.

Сейчас рассмотрим задачу о назначениях с двумя аддитивными критериями. Здесь будем считать, что задача определяется парой (nxn)-матриц $A = \{ a_{ij} \}$ *и* $B = \{ b_{ij} \}$, элементы которых принадлежат множеству $N \cup \{\omega\}$, где N – совокупность всех целых неотрицательных чисел, а ω – специальный символ запрета. Если в позиции (i,j) одной матрицы стоит знак запрета, то этот знак стоит в позиции (i,j) и в другой матрице. Числовой элемент a_{ij} (b_{ij}) есть оценка закрепления исполнителя i за работой r_j по первому (соответственно второму) показателю. Назначения – взаимно однозначные отображения множества $\{1, 2, \ldots, n\}$ в себя. Назначение π закрепляет за исполнителем i работу $r_{\pi(i)}$, $i = 1, 2, \ldots, n$. Назначение π допустимо, если $a_{i\pi(i)}$ отлично от ω при всех $i = \overline{1,n}$. Множество всех допустимых назначений обозначим символом H. Каждое допустимое назначение π определяет значения критериев $Q_1(\pi) = \sum_{i=1}^{n} a_{i\,\pi(i)}$ и $Q_2(\pi) = \sum_{i=1}^{n} b_{i\,\pi(i)}$. Критерии считаем максимизируемыми. Рассматриваемая *бикритериальная задача о назначениях (БКЗН)* записывается в виде:

$$\max_{\pi \in H} \{ Q_1(\pi), Q_2(\pi) \}. \qquad (4.27)$$

Рассмотрим вопрос о построении полного множества эффективных в *БКЗН* оценок методом последовательных уступок.

Отметим, что задача (4.27) с двумя лексикографически упорядоченными критериями легко сводится к *КЗН*. Если в лексикографическом упорядочении ведущим является определяемый матрицей A критерий $Q_1(\pi)$, то оптимальное назначение получается путем решения *КЗН*, определяемой матрицей $(p+1)A + B$, здесь p – достаточно большая константа (можно положить p равным сумме максимальных элементов строк матрицы B). Если в лексикографическом упорядочении ведущим является определяемый матрицей B критерий $Q_2(\pi)$, то оптимальное назначение получается путем решения *КЗН*, определяемой матрицей $A+ (p+1)B$, где p – достаточно большая костанта (можно положить p равным сумме максимальных элементов строк матрицы A).

Построение полного множества эффективных в (4.27) оценок начинаем с решения задач с двумя взаимно противоположными лексикографическими упорядочениями критериев $Q_1(\pi)$ и $Q_2(\pi)$. В результате решения этих задач находим две крайние эффективные оценки: $P^* = (Q_1(\pi^*), Q_2(\pi^*))$ и $P^{**} = (Q_1(\pi^{**}), Q_2(\pi^{**}))$; оценка P^* получается в ситуации, когда ведущим является критерий $Q_1(\pi)$, π^* – соответствующее оптимальное назначение; оценка P^{**} получается при противоположном лексикографическом упорядочении критериев, π^{**} – соответствующее оптимальное назначение. Если $P^* = P^{**}$, то совокупность эффективных оценок $\boldsymbol{E}$ в рассматриваемой задаче уже найдена, она состоит из единственной точки $P^* = P^{**}$. Если же точки P^* и P^{**}. не совпадают, то $\Delta_1 = Q_1(\pi^*) - Q_1(\pi^{**})$ – максимальная допустимая уступка по значению первого критерия, а $\Delta_2 = Q_2(\pi^{**}) - Q_2(\pi^*)$ – максимальная допустимая уступка по значению второго критерия. Для определенности считаем, что $\Delta_1 \le \Delta_2$, тогда уступки целесообразно выполнять по значению первого критерия.

Возникают вопросы, какова минимальная величина первой и каждой из последующих уступок, действительно расширяющих рассматриваемое множество назначений, и как построить назначение, оптимальное по второму критерию при заданной уступке по значению первого критерия.

Пусть $\Gamma(A,k) = \{q_1, q_2,...,q_t\}$ - упорядоченная по убыванию последовательность всех значений функции $Q_1(\pi)$, $\pi \in H$, больших или равных k, где k – натуральная константа, превышающая $Q_1(\pi^{**})$; напомним, что H – множество всех допустимых назначений.

Решив КЗН

$$\max_{\pi \in H} Q_1(\pi), \qquad (4.28)$$

получаем оптимальное значение критерия $q_1 = Q_1(\pi^*)$ и оптимальные значения двойственных переменных ξ^*_i, $i=\overline{1,n}$, и η^*_j, $j=\overline{1,n}$, – потенциалы строк и столбцов матрицы A. Числовой элемент a_{ij} этой матрицы потенциален, если $a_{ij} = \xi^*_i + \eta^*_j$ Назначение π оптимально в (4.28) тогда и только тогда, когда все элементы $a_{i\pi(i)}$, $i=\overline{1,n}$, потенциальны.

По матрице A строим совокупность матриц $S_1 = \{A_i,\ i=\overline{1,n}\}$, где матрица A_i получается из A заменой всех потенциальных элементов i- ой строки символом ω. Для каждой матрицы A_i решаем определяемую ею *КЗН*; оптимальное значение ее критерия обозначаем r_i . Пусть наибольшее из найденных чисел r_i равно r; множество матриц из S_1, для которых $r_i = r$, обозначим M_1. Через H_1 обозначим совокупность назначений, каждое из которых является оптимальным для какой-либо КЗН, определяемой некоторой матрицей из M_1.

Рассуждением от противного легко устанавливается, что r - это второй элемент последовательности $\Gamma(A)$, т. е. $q_2 = r$, и что $Q_1(\pi) = q_2$ тогда и только тогда, когда $\pi \in H_1$.

Отсюда получаем возможность выполнить полный синтез последовательности $\Gamma(A, k)$ и описать множества назначений, реализующих каждый из ее элементов.

Введем следующую процедуру k-разложения матрицы A. На первом этапе строим описанную выше совокупность матриц $S_1 = \{A_i,\ i=\overline{1,n}\ \}$. Для каждой матрицы X из S_1 решаем определяемую этой матрицей КЗН; если для некоторой матрицы X допустимых назначений не существует или же оптимальное значение критерия оказывается меньшим константы k, то матрица X отбрасывается. В оставшихся матрицах, их совокупность обозначим S_1*,. отмечаем потенциальные элементы.

При реализации второго этапа каждая матрица A_i из S_1* порождает матрицы $A_{i1,}$ $A_{i2,\ldots,}$ A_{in}; матрица A_{ij} получается из A_i заменой символом ω всех потенциальных элементов ее j -ой строки. Получаемое множество матриц обозначаем S_2. Для каждой матрицы из S_2 решаем определяемую ею КЗН; если для некоторой матрицы X допустимых назначений не существует или же оптимальное значение критерия оказывается меньшим константы k, то матрица X отбрасывается. В оставшихся матрицах отмечаем потенциальные элементы; множество оставшихся матриц обозначаем S_2*.

Идентичным образом реализуются последующие этапы, вплоть до завершения процесса по причине пустоты оставшегося множества матриц. Как легко видеть, общее число этапов не превышает $min\ [n^2 - n + 1\ ,\ q_1 - k\ ]$.

Совокупность всех рассмотренных и не отброшенных на этапах процесса матриц обозначим **S**; матрица A также относится к числу рассмотренных, $A \in \mathbf{S}$. В качестве результата выполненного процесса формируем именуемый k-разложением матрицы A кортеж

$$\Omega_k(A) = \{p_1, p_2, \ldots, p_L, W_1, W_2, \ldots, W_L\},$$

где $p_1, p_2, \ldots, p_L$ – упорядоченное по убыванию множество максимальных значений критерия в КЗН, определяемых матрицами из **S**, а W_i – подмножество принадлежащих **S** матриц, для которых максимальное значение критерия равно p_i, $i = 1, 2, \ldots, L$. Для последовательности $p_1, p_2, \ldots, p_L$, т.е. начальной части кортежа $\Omega_k(A)$, примем обозначение $\Omega_k^{н}(A)$. Множество назначений, каждое из которых оптимально в некоторой КЗН, определяемой матрицей из W_i, обозначим H_i, $i = 1, 2, \ldots, L$. Как очевидно, $\Gamma(A,k) = \Omega_k^{н}(A)$, а H_i есть множество всех допустимых в рассматриваемой БКЗН назначений, на которых критерий $Q_1(\pi)$ принимает значение p_i , $i = 1, 2, \ldots, L$.

Обозначим $E_k(Z)$ множество всех эффективных оценок БКЗН Z, которые по первой координате не ниже константы k.

Для построения $E_k(Z)$, где $Q_1(\pi^{**}) < k < Q_1(\pi^{*})$, и получения соответствующих Парето-оптимальных решений следует построить k-разложение матрицы A. Далее при фиксированных $i = 2, 3,..., L$ для каждой матрицы M из W_i надо решить задачу о назначениях с двумя лексикографически упорядоченными максимизируемыми критериями, определяемыми матрицами M (ведущий критерий) и B. На каждом из оптимальных назначений первый критерий принимает значение p_i; второй критерий принимает, вообще говоря, различные значения, наибольшее из них обозначим m_i. Для каждой оценки (p_i, m_i) имеется реализующее ее назначение.

Рассмотрим совокупность оценок

$$\{(p_1, Q_2(\pi^{*})), (p_2, m_2), (p_3, m_3),..., (p_L, m_L)\}$$

и исключим из нее доминируемые. В результате получим искомую совокупность $E_k(Z)$, причем для каждой оценки из $E_k(Z)$ имеем реализующее ее Парето-оптимальное назначение.

Как очевидно, $E(Z) = E_K(Z) \cup \{(Q_1(\pi^{**}), Q_2(\pi^{**}))\}$, здесь $K = Q_1(\pi^{**}) + 1$.

ПРИМЕР 4.2. Методом последовательных уступок для БКЗН Z, определяемой матрицами $A = \begin{pmatrix} 10^* & 5 & 0 \\ 5 & 10^* & 15^* \\ 0 & 3 & 10^* \end{pmatrix}$ и $B = \begin{pmatrix} 5 & 10 & 15 \\ 5 & 5 & 5 \\ 15 & 10 & 5 \end{pmatrix}$

требуется построить полное множество эффективных оценок E(Z); необходимо также указать Парето - оптимальные решения, обеспечивающие эти оценки.

Сначала отыскиваем оптимальные при лексикографических упорядочениях критериев назначения π^{*} и π^{**}, при этом находим потенциальные элементы матриц. Получаем $\pi^{*}(1) = 1, \pi^{*}(2) = 2, \pi^{*}(3) = 3; \pi^{**}(1) = 3, \pi^{**}(2) = 2, \pi^{**}(3) = 1$. Крайние эффективные оценки имеют координаты (30, 15) и (10, 35); потенциальные элементы матрицы A отмечены звездочками.

Переходим к построению k-разложения матрицы A, где $k = K = Q_1(\pi^{**}) + 1 = 11$. Совокупность S_1 состоит из матриц

$$A_1 = \begin{pmatrix} \omega & 5^* & 0 \\ 5^* & 10^* & 15^* \\ 0^* & 3 & 10^* \end{pmatrix}; \quad A_2 = \begin{pmatrix} 10^* & 5^* & 0 \\ 5^* & \omega & \omega \\ 0 & 3 & 10^* \end{pmatrix}; \quad A_3 = \begin{pmatrix} 10^* & 5 & 0 \\ 5 & 10 & 15^* \\ 0 & 3^* & \omega \end{pmatrix}.$$

Находим потенциальные элементы этих матриц (они отмечены звездочками). Максимальные значения критериев КЗН, определяемых матрицами A_1, A_2, A_3 равны соответственно $20, 20, 28$; множество матриц S_1^* совпадает с множеством S_1 . Совокупность матриц S_2 следующая (указываем только матрицы, не содержащие строк, в которых стоят лишь знаки ω):

$$A_{11} = \begin{pmatrix} \omega & \omega & 0 \\ 5 & 10 & 15 \\ 0 & 3 & 10 \end{pmatrix}; \quad A_{13} = \begin{pmatrix} \omega & 5 & 0 \\ 5 & 10 & 15 \\ \omega & 3 & \omega \end{pmatrix}; \quad A_{21} = \begin{pmatrix} \omega & \omega & 0 \\ 5 & \omega & \omega \\ 0 & 3 & 10 \end{pmatrix};$$

$$A_{23} = \begin{pmatrix} 10 & 5 & 0 \\ 5 & \omega & \omega \\ 0 & 3 & \omega \end{pmatrix}; \quad A_{31} = \begin{pmatrix} \omega & 5 & 0 \\ 5 & 10 & 15 \\ 0 & 3 & \omega \end{pmatrix};$$

$$A_{32} = \begin{pmatrix} 10 & 5 & 0 \\ 5 & 10 & \omega \\ 0 & 3 & \omega \end{pmatrix}; \quad A_{33} = \begin{pmatrix} 10 & 5 & 0 \\ 5 & 10 & 15 \\ 0 & \omega & \omega \end{pmatrix}.$$

Максимальные значения аддитивного критерия для матриц A_{11}, A_{13}, A_{21}, A_{23}, A_{31}, A_{32}, A_{33} равны соответственно *10, 8, 8, 8, 20, 10, 20.* Множество S_2^* включает только матрицы A_{31} и A_{33}; легко проверяется, что оптимальные значения критерия в КЗН, определяемых матрицами из S_3, не превышают *10*, множество S_3^* оказывается пустым.

Таким образом, (*11*)-разложение матрицы *A* есть кортеж

$$\{30, 28, 20, \{A\}, \{A_3\}, \{A_2, A_3, A_{31}, A_{33}\}\}.$$

Решение определяемой матрицами A_3 и *B* задачи с лексикографически упорядоченными критериями дает оптимальное назначение $\pi_1(1) = 1$, $\pi_1(2) = 3$, $\pi_1(3) = 2$; в плоскости критериев получаем оценку (*28, 20*). Далее решаем четыре аналогичные задачи, в каждой из которых ведущий критерий определяется матрицей A_2, A_3, A_{31} или A_{33}, а дополнительный - матрицей *B*; при этом оказывается, что максимальное значение дополнительного критерия в рассматриваемых случаях равно *30*. Так в области критериев получаем оценку (*20, 30*); эта оценка обеспечивается назначением $\pi_2(1) = 2$, $\pi_2(2) = 3$, $\pi_3(3) = 1$.

В найденной совокупности оценок нет доминируемых. Получаем, что в рассматриваемой задаче *Z* множество эффективных оценок содержит следующие точки: *(30,15), (28,20), (20,30), (10,35).* Эти оценки обеспечиваются назначениями $\pi^*, \pi_1, \pi_2, \pi^{**}$ соответственно.

Отметим, что для получения первых *d* членов *k*-разложения (т.е. чисел $p_1, p_2, \ldots, p_d$) требуется выполнить $d - 1$ этап процесса *k*-разложения, при этом полностью определяются соответствующие множества матриц $W_1, W_2, \ldots, W_d$ (здесь $W_1 = \{A\}$). Число элементарных операций, выполняемых на первом этапе процесса *k*-разложения, имеет порядок n^4 (решается *n* различных КЗН, каждая вычислительной сложности порядка n^3); вычислительная сложность каждого следующего этапа по верхней оценке в *n* раз больше вычислительной сложности предыдущего этапа. Для БКЗН вида (4.27) величину

$$\min\{Q_1(\pi^*) - Q_1(\pi^{**}); \; Q_2(\pi^{**}) - Q_2(\pi^*)\}$$

назовем рассогласованием критериев. Через $\boldsymbol{K}(d)$ обозначим класс БКЗН, в которых рассогласование критериев не превышает натуральную константу *d*.

Очевиден следующий факт.

Т е о р е м а 4.3. Для задач класса $\boldsymbol{K}(d)$, где значение параметра *d* считается фиксированным, синтез полного множества эффективных оценок и и построение обеспечивающих эти оценки Парето-оптимальных решений реализуемы за

полиномиально зависящее от n время (соответствующая оценка числа элементарных операций Cn^{3+d}).

4.3 Синтез полных совокупностей эффективных оценок на основе рекуррентных соотношений многокритериального динамического программирования.

Для простоты изложения ниже рассматриваются только бикритериальные задачи. При выводе общих рекуррентных соотношений бикритериального динамического программирования критерии задач для определенности считаем минимизируемыми. Все построения идентичным образом выполняются для задач с любым конечным числом критериев, максимизируемых или минимизируемых.

Пусть

$$\boldsymbol{\Omega}^{+} = \{\boldsymbol{D};\ \boldsymbol{x}_0;\ \boldsymbol{F};\ \boldsymbol{V(x)},\ \boldsymbol{f(x,v)},\ \boldsymbol{s}_1\boldsymbol{(x,v)},\ \boldsymbol{s}_2\ \boldsymbol{(x,v)}\}$$

– дискретная управляемая система, отличие которой от введенной в главе 1 системы $\boldsymbol{\Omega}$ заключается в наличии не одной, а двух функций платежа; здесь $\boldsymbol{s}_1\boldsymbol{(x,v)}$ и $\boldsymbol{s}_2\boldsymbol{(x,v)}$ – функции платежа по первому и второму показателям соответственно. Стоимость произвольной траектории $T = \{\boldsymbol{x}^0,\ \boldsymbol{x}^1,\ \boldsymbol{x}^2, \ldots,\ \boldsymbol{x}^n\}$ системы $\boldsymbol{\Omega}^{+}$ по i-му показателю обозначаем $C_i(T)$ и определяем следующим образом:

$$C_i(T) = \sum_{t=1}^{n} \boldsymbol{s}_i\boldsymbol{(x^{t-1}, v^t)},\ \ i = 1,2.$$

Обозначим через M множество всех полных траекторий. Рассмотрим вопрос синтеза полной совокупности эффективных оценок для бикритериальной задачи

$$\min_{T \in M} (C_1(T),\ C_2(T)) \qquad (4.29)$$

Для произвольного состояния x через $M(x)$ обозначим множество всех заключительных x-траекторий. Введем в рассмотрение совокупность частных бикритериальных задач

$$\min_{T \in M(x)} (C_1(T),\ C_2(T)). \qquad (4.30)$$

Полные совокупности эффективных в задачах (4.29) и (4.30) оценок будем обозначать $\boldsymbol{E}$ и $\boldsymbol{E(x)}$ соответственно. Для синтеза указанных совокупностей определим две необходимые операции. Пусть $\vec{x}$ – вектор, а Y – множество векторов той же размерности, что и вектор $\vec{x}$. Тогда через $\vec{x} \oplus Y$ будем обозначать совокупность всех векторов $\vec{v}$, представимых в виде $\vec{v} = \vec{x} + \vec{y}$, где $\vec{y} \in Y$. Пусть M - произвольное множество двумерных векторов-оценок. Через $eff(M)$ будем обозначать максимальное по включению подмножество недоминируемых в M векторов. Очевидны следующие равенства:

$$\boldsymbol{E(x)} = \{(0,0)\},\ \boldsymbol{x} \in \boldsymbol{F}. \qquad (4.31)$$

Пусть $\boldsymbol{x}$ - произвольное нефинальное состояние системы $\boldsymbol{\Omega}^{+}$. В этом состоянии можно реализовать любое принадлежащее множеству $\boldsymbol{V(x)}$ управление. Предположим, что в множестве $\boldsymbol{V(x)}$ выбрано управление $\boldsymbol{v}$. Данное управление влечет платеж $(\boldsymbol{s}_1\boldsymbol{(x,\ v)},\ \boldsymbol{s}_2\boldsymbol{(x,\ v)})$, а следующим состоянием системы оказывается $\boldsymbol{f(x,\ v)}$. Выделив из

совокупности оценок, характеризующих заключительные ***f(x, v)***-траектории, эффективные оценки, мы получаем множество ***E(f(x, v)***). Если в состоянии ***x*** системы Ω^+ выбрать управление ***v***, то в итоге начинающегося в ***x*** процесса управления можно обеспечить любую оценку из совокупности $[(s_1(x_i,v),\ s_2(x_i,v)) \oplus E(f(x,\ v))]$. Отсюда получаем:

$$E(x) = eff\{ \bigcup_{v \in V(x)} [(s_1(x,v),\ s_2(x,v)) \oplus E(f(x,\ v))]\},\ x \in D \backslash F. \quad (4.32)$$

Как очевидно, $E(x_0) = E$.

Равенства (4.31) – (4.32) – рекуррентные соотношения бикритериального динамического программирования, позволяющие реализовать обратный метод Беллмана для синтеза полной совокупностей эффективных в задаче (4.29) оценок.

Синтез множеств ***E(x)*** на основании записанных рекуррентных соотношений выполняется поэтапно, в следующем порядке. На первом этапе фиксируем, что ***E(x)*** = $\{(0,0)\}$ для всех финальных состояний ***x***. На каждом следующем этапе построение очередного множества эффективных оценок выполняется для произвольного состояния ***x*** такого, что ***E(x)*** неизвестно, но значения ***E(y)*** для всех непосредственно следующих за ***x*** состояний ***y*** уже найдены (состояние *y* относим к числу непосредственно следующих за состоянием ***x***, если из состояния ***x*** под воздействием некоторого возможного в этом состоянии управления можно совершить непосредственный переход в состояние ***y***). Если нумерация состояний системы выполнена в соответствии с изложенным в п.1 главы 1 алгоритмом, то множества ***E(x)*** находим в порядке убывания номеров состояний. Последним в процессе счета определяется множество $E(x_0)$ – полная совокупность эффективных оценок в задаче (4.29).

Для обеспечения возможности определения по произвольной эффективной в задаче (4.29) оценке соответствующей Парето-оптимальной полной траектории системы Ω^+ в процессе счета по общему рекуррентному соотношению (4.32) следует дополнительно строить список переходов (СП), в который для каждой включаемой в состав любого из множеств ***E(x***) оценки ***m*** вносится запись ***(m, x, v)***, где ***v*** – управление, при котором совокупность $(s_1(x,v),\ s_2(x,v)) \oplus E(f(x,\ v))$содержит оценку ***m.***

По произвольной эффективной в (4.29) оценке, пусть это ***m***(*0*), для построения соответствующей полной Парето-оптимальной траектории выполняются следующие действия. На первом этапе в СП отыскивается запись с первой позицией ***m***(*0*) и второй позицией x_0; третья позиция этой записи, обозначим ее ***v(0)***, представляет собой управление, которое при реализации искомой траектории применяется в начальном состоянии системы x_0. Следующим за x_0 состоянием в синтезируемой траектории является ***x(1)*** = $f(x_0,\ v(0))$. Начинающийся в ***x(1)*** и завершающийся в финальном состоянии фрагмент траектории оценивается вектором стоимости ***m***(*1*) = ***m***(*0*) – $(s_1(x_0,v(0)),\ s_2(x_0,v(0)))$. На втором этапе в СП отыскивается запись с первой позицией ***m***(*1*) и второй позицией ***x(1)***; третья позиция этой записи, обозначим ее ***v(1)***, представляет собой управление, которое при реализации траектории применяется в состоянии *x(1)*. Следующим за ***x(1)*** состоянием в синтезируемой траектории является ***x(2)*** = $f(x(1),\ v(1))$. Начинающийся в ***x(2)*** и завершающийся в финальном состоянии фрагмент искомой траектории оценивается вектором стоимости ***m***(*2*) = ***m***(*1*) – $(s_1(x(1),\ v(1)),\ s_2(x(1),\ v(1)))$. На третьем этапе отыскивается запись с первой позицией ***m***(*2*) и второй позицией ***x(2);*** третья позиция этой записи, обозначим ее ***v(2)***, представляет собой управление, которое при реализации траектории применяется в состоянии ***x(2)***. Следующим за ***x(2)*** состоянием в синтезируемой траектории является *x(3)* = ***f(x(2),***

v(2)). Действуя описанным образом далее, мы последовательно получаем все состояния синтезируемой траектории. Процесс синтеза заканчивается отысканием принадлежащего траектории финального состояния.

Для реализации прямого метода Беллмана обозначим через *M*(x)* множество всех начальных *x*-траекторий системы $\boldsymbol{\Omega}^+$ и введем в рассмотрение бикритериальные задачи

$$\min_{T \in M^*(x)} (C_1(T),\ C_2(T)),\ \ \boldsymbol{x} \in \boldsymbol{D}. \qquad (4.33)$$

Для совокупностей эффективных в (4.33) оценок примем обозначение ***E*(x)***.

Как очевидно,

$$\boldsymbol{E^*(x_0)} = \{(0,0)\}. \qquad (4.34)$$

Произвольное состояние ***x*** считается непосредственно предшествующим состоянию ***y***, если состояние ***y*** относится к числу непосредственно следующих за состоянием ***x***. Управление, переводящее систему $\boldsymbol{\Omega}^+$ из состояния ***x*** в состояние ***y***, обозначаем ***v[x, y]***. Множество состояний системы, непосредственно предшествующих состоянию ***y***, обозначаем ***Г(y)***.

Тогда

$$\boldsymbol{E^*(y)} = eff\{ \bigcup_{x \in \Gamma(y)} [\boldsymbol{E^*(x)} \oplus (\boldsymbol{s_1(x, v[x, y])}, \boldsymbol{s_2(x, v[x, y])})]\}. \qquad (4.35)$$

Синтез множеств ***E*(y)*** на основании записанных рекуррентных соотношений выполняется поэтапно, в следующем порядке. На первом этапе фиксируем, что $\boldsymbol{E^*(x_0)} = \{(0,0)\}$. На каждом следующем этапе построение очередного множества эффективных оценок выполняется для произвольного состояния ***y*** такого, что ***E*(y)*** неизвестно, но множества ***E(x)*** для всех непосредственно предшествующих состоянию ***y*** состояний ***x*** уже построены.

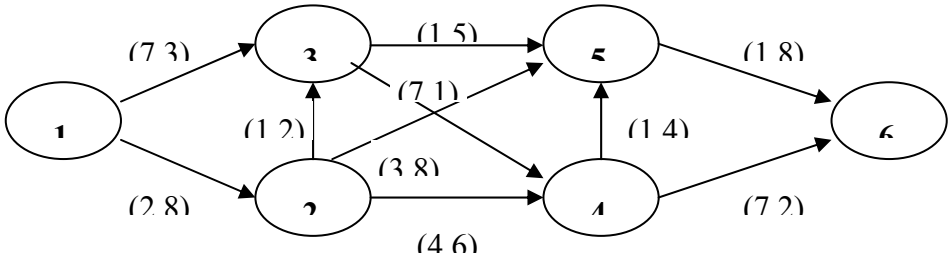
Совокупность эффективных в задаче (4.29) оценок определяется по формуле

$$E = eff\{ \bigcup_{y \in F} E^*(y) \}. \qquad (4.36)$$

Равенства (4.34)-(4.36) – соотношения бикритериального динамического программирования, позволяющие реализовать прямой метод Беллмана для синтеза полной совокупности эффективных оценок в задаче (4.29).

ПРИМЕР 4.3. Дискретно управляемая система $\boldsymbol{\Omega}^+$ определяется представленным на рис. 4.1 графом G^+. Начальным считается состояние *1*, множество финальных состояний одноэлементно: ***F={6}.*** Требуется найти полную совокупность эффективных оценок в соответствующей задаче (4.29).

Рис. 4.1. Граф G^+.



Решение выполним двумя способами.

С п о с о б 1. Применяя рекуррентные соотношения (4.31)-(4.32), последовательно получаем:

E(6) = {(*0,0*)}, ***E(5)*** = {(*1,8*) ⊕ ***E(6)***}= {(*1,8*)}; ***E(4)*** = *eff*{(*1,4*) ⊕ ***E(5)***, (*7,2*)}= {(*2,12*), (*7,2*)}; ***E(3)*** = *eff*{(*1,5*) ⊕ ***E(5),*** (*7,1*) ⊕ ***E(4)***} = *eff*{(*2,13*), (*14,3*),(*9,13*)}= {(*2,13*), (*14,3*)}; ***E(2)*** = *eff*{(*1,2*) ⊕***E(3),*** (*3,8*) ⊕***E(5),(4,6)***⊕***E(4)***} = *eff* {(*3,15*), (*15,5*),(*4,16*), (*6,18*), (*11,8*)}= {(*3,15*), (*15,5*), (*11,8*)}; ***E(1)***}= *eff*{(*2,8*) ⊕ ***E(2)***, (*7,3*) ⊕ ***E(3)***} = *eff*{(*5,23*), (*17,13*),(*13,16*),(*9,16*), (*21,6*)}= {(*5,23*), (*17,13*), (*9,16*), (*21,6*)}. Совокупность оценок {(*5,23*), (*17,13*), (*9,16*), (*21,6*)}является искомой.

Для обеспечения возможности определения по произвольной эффективной в рассматриваемой задаче (4.29) оценке соответствующей Парето-оптимальной полной траектории, в процессе счета по общему рекуррентному соотношению (4.32) последовательно составляем записи списка переходов (СП). При этом управления, возможные в каждой нефинальной вершине *i*, отождествляем с исходящими из этой вершины дугами (*i,j*). Так как в каждой записи списка переходов второй позицией является *i,* в третьей позиции вместо пары (*i,j*) будем указывать только значение *j*. Согласно изложенному, при определении ***E(5)*** в СП вносим запись ((*1,8*),*5*,*6*). При определении ***E(4)*** в СП вносим записи ((*2,12*), *4*,*5*) и ((*7,2*), *4*,*6*). При определении ***E(3)*** в СП вносим записи ((*2,13*), *3*,*5*) и ((*14,3*), *3*,*4*). При определении ***E(2)*** в СП вносим записи ((*3,15*), *2, 3*), ((*15,5*), *2, 3*) и ((*11,8*), *2*, *4*). При определении ***E(1)*** в СП вносим записи ((*5,23*), *1*, *2*), ((*17,13*), *1, 2*), ((*9,16*), *1,3*) и ((*21,6),1*, 3).

Пусть в построенной совокупности эффективных оценок выбрана оценка (*21,6).* Требуется найти соответствующую полную траекторию. Действуем следующим образом. В списке переходов находим запись с первой компонентой (*21,6),* вторая ее компонента должна быть *1*. Условиям удовлетворяет запись ((*21,6),1*, 3). Следовательно, следующим за *1* состоянием траектории является *3*. Стоимость этого перехода (*7, 3*). Получаем, что фрагмент искомой траектории, начинающийся в состоянии *3* и заканчивающийся в финальном состоянии 6, имеет стоимость (*14,3*). В списке переходов находим запись с первой компонентой (*14,3)* и второй компонентой *3*. Условиям удовлетворяет запись ((*14,3*), *3*,*4*). Поэтому следующим за *3* состоянием траектории является *4*. Стоимость перехода из *3* в *4* равна (*7, 1*). Отсюда заключаем, что фрагмент искомой траектории, начинающийся в состоянии *4* и заканчивающийся в финальном состоянии, имеет стоимость (*7,2*). В списке переходов находим запись с первой компонентой (*7,2)* и второй компонентой *4*. Условиям удовлетворяет запись ((*7,2*), *4*,*6*). Поэтому следующим за *4* состоянием траектории является *6.*Состояние *6* - финальное, траектория определена полностью.

С п о с о б 2. Применяя рекуррентные соотношения (4.34)-(4.35), последовательно получаем:

E*(1) = {(*0,0*)}, ***E*(2)*** = {(*2,8*)}; ***E*(3)*** = *eff*{ ***E*(2)*** ⊕ ***(1,2)***, (*7,3*)}= {(*3,10*), (*7,3*)}; ***E*(4)*** = *eff*{***E*(2)*** ⊕ ***(4,6)***, ***E*(3)*** ⊕ ***(7,1)***} = *eff*{(*6,14*), (*14,4*), (*10,11*)}= {(*6,14*), (*4,15*), (*10,11*)}; ***E*(5)*** = *eff*{***E*(2)*** ⊕ ***(3,8)***, ***E*(3)*** ⊕ ***(1,5)***, ***E*(4)*** ⊕ ***(1,4)***} = *eff*{(*8,8*), (*4,15*), (*5,16*), (*7,18*), (*15,8*), (*11,15*)}= {(*8,8*), (*4,15*)}; ***E*(6)*** = *eff*{***E*(4)*** ⊕ ***(7,2)***, ***E*(5)*** ⊕ ***(1,8)***} = *eff*{(*13,16*), (*21,6*), (*17,13*), (*5,23*), (*9,16*)}= {(*21,6*), (*17,13*), (*5,23*), (*9,16*)}.

Полученное в итоге множество эффективных оценок совпадает с множеством оценок, построенных в результате применения первого способа.

Дадим общее описание метода встречного счета при синтезе полной совокупности эффективных в задаче (4.29) оценок. Идентично тому как это было сделано в главе 1, для рассматриваемой системы $\boldsymbol{\Omega}^+$ введем понятия разрезающего и минимального разрезающего подмножеств состояний. Подмножество промежуточных состояний M системы $\boldsymbol{\Omega}^+$ называем *разрезающим*, если каждая траектория $\boldsymbol{T}$ этой системы имеет в своем составе состояния из M. Разрезающее подмножество M^* называем *минимальным разрезающим*, если среди его собственных подмножеств нет разрезающих. Введем также операцию сложения множеств векторов одинаковой размерности. Пусть X и Y – два произвольных множества векторов одинаковой размерности. Через $X[+]Y$ будем обозначать совокупность всех векторов $\vec{v}$, представимых в виде $\vec{v} = \vec{x} + \vec{y}$, где $\vec{x} \in X$ и $\vec{y} \in Y$.

Пусть M^* – фиксированное минимальное разрезающее подмножество состояний. Реализуя метод встречного счета, прямым счетом определяем все множества $\boldsymbol{E^*(x)}$, $\boldsymbol{x} \in M^*$, и обратным счетом – все множества $\boldsymbol{E(x)}$, $\boldsymbol{x} \in M^*$. Далее полная совокупность E эффективных в задаче (4.29) оценок определяется по формуле

$$E = eff\{ \bigcup_{x \in M^*} (\boldsymbol{E(x)} [+] \boldsymbol{E^*(x,)} \}.$$

В многокритериальных задачах метод встречного счета часто оказывается более предпочтительным в сравнении с "однонаправленными" методами, т.к. число эффективных оценок для начальных и заключительных траекторий может иметь экспоненциально зависящий от длины траекторий порядок роста.

Перейдем к рассмотрению ситуации, в которой каждая траектория $T = \{\boldsymbol{x^0}, \boldsymbol{x^1}, \boldsymbol{x^2}, \ldots, \boldsymbol{x^n}\}$ системы $\boldsymbol{\Omega}^+$ характеризуется двумя величинами – суммарной стоимостью по первому показателю $C_1(T)$, как это было выше, и максимальным из пошаговых платежей по второму показателю $K_2(T)$, где

$$K_2(T) = \max_{t=1,2,\ldots,n} \mathbf{s}_2 (\mathbf{x}^{\mathbf{t-1}}, \mathbf{v}^{\mathbf{t}}).$$

Возникает двухкритериальная задача

$$\min_{T \in M} (C_1(T), K_2(T)), . \qquad (4.37)$$

здесь M – множество всех полных траекторий системы $\boldsymbol{\Omega}^+$.

Полную совокупность эффективных оценок в задаче (4.37) обозначим $\boldsymbol{E'}$. Введем в рассмотрение совокупность частных задач

$$\min_{T \in M(x)} (C_1(T(\boldsymbol{x})), K_2(T(\boldsymbol{x})), \qquad (4.38)$$

где $M(x)$ – множество всех заключительных $\boldsymbol{x}$ траекторий системы $\boldsymbol{\Omega}^+$. Совокупность эффективных в (4.38) оценок обозначим $\boldsymbol{E'(x)}$.

Очевидно, что

$$\boldsymbol{E'(x)} = \{(0,0)\}, \boldsymbol{x} \in \boldsymbol{F}. \qquad (4.39)$$

Для записи основного рекуррентного соотношения введем следующую операцию. Пусть $u = (u_1,u_2)$ – произвольный двумерный вектор, а Y – любое множество двумерных векторов. Через $u \otimes Y$ обозначим множество векторов, определяемое

следующим образом. Вектор $w = (w_1,w_2)$ принадлежит множеству $u \otimes Y$ тогда и только тогда, когда в Y имеется вектор (y_1,y_2) такой, что $w_1 = u_1 + y_1$, $w_2 = max\ (u_2, y_2)$.

Далее имеем:

$$\boldsymbol{E'(x) = eff\{ \bigcup_{v \in V(x)} [(s_1(x,v),\ s_2(x,v)) \otimes E'(f(x,v))]\},\ x \in D \backslash F.} \quad (4.40)$$

Равенства (4.39)–(4.40) – рекуррентные соотношения динамического программирования, позволяющие реализовать обратный метод Беллмана для синтеза совокупности эффективных оценок $\boldsymbol{E'}$ в задаче (4.37). Как очевидно, $\boldsymbol{E'(x_0) = E'}$.

В рассмотренных бикритериальных задачах критерии считались минимизируемыми. Синтез рекуррентных соотношений, позволяющих строить полные совокупности эффективных оценок для задач, в которых критерии являются максимизируемыми, проводится идентичным образом. Следует только отметить, что операция выделения из произвольного множества оценок M подмножества эффективных оценок $eff(M)$ в зависимости от того, являются критерии минимизируемыми или максимизируемыми, выполняется по разному. Если, например, $M = \{(1,7),(2,4),(5,5),(6,4)\}$ и решается бикритериальная задача с минимизируемыми критериями, то $eff(M) = \{(1,7),(2,4)\}$; если же критерии задачи являются максимизируемыми, то $eff(M) = \{(1,7),(5,5),(6,4)\}$. Задача, в которой один критерий является максимизируемым, а другой минимизируемым, умножением максимизируемого критерия на (– 1) сводится к задаче с двумя минимизируемыми критериями.

Рассмотрим несколько многокритериальных аналогов введенных ранее задач.

Задача оптимального распределения капиталовложений (ЗОРК) в бикритериальной постановке. Имеется денежная сумма C и возможные сферы вложений $S_1,\ S_2,\ldots,S_n$. Для каждой сферы S_t считаются известными монотонно возрастающие в нестрогом смысле функции $f_t(u)$ и $\varphi_t(u)$, определяющие соответственно ожидаемую и минимальную гарантированную величины доходов, получаемых при вложении в данную сферу суммы u. Считается, что величина суммы, вкладываемой в каждую сферу, целочисленна; поэтому функции $f_t(u)$ и $\varphi_t(u)$ можно положить заданными таблично. Требуется найти совокупность эффективных в рассматриваемой задаче оценок.

Сформулированную задачу назовем задачей $\boldsymbol{Z}$. Введем в рассмотрение совокупность частных задач $Z(k,V)$, *где* $k \in \{1,2,\ldots,n\}$, $V \in \{0,1, , \ldots , C\}$. В задаче $Z(k,V)$ между сферами капиталовложений $S_k,\ S_2,\ldots,S_n$. должна быть распределена сумма V. Множество эффективных в задаче $Z(k,V)$ оценок обозначим $E(k,V)$. Как очевидно, для всех $V \in \{0,1, , \ldots , C\}$:

$$E(n,V) = (f_n(V),\ \varphi_n(V)). \quad (4.41)$$

Для $k \in \{1,2,\ldots,n-1\}$, $V \in \{0,1, , \ldots , C\}$ имеет место следующее равенство:

$$E(k,V) = eff\{ \bigcup_{j=0,1,\ldots,V} [(f_k(j)\ ,\ \varphi_k(j))\ \hat{\oplus}\ E(k+1,V-j)]\ \}. \quad (4.42)$$

Последовательно конструируя множества $E(k,V)$ для все меньших значений параметра k, в итоге находим множество $E(1,C)$, совпадающее с искомой совокупностью эффективных в задаче $\boldsymbol{Z}$ оценок.

Многомерная многокритериальная задача о ранце. Математическая формулировка d-мерной задачи о ранце с n предметами и m аддитивными критериями имеет вид:

$$\max\left(\sum_{j=1}^{n} c_{1j}x_j, \sum_{j=1}^{n} c_{2j}x_j, ..., \sum_{j=1}^{n} c_{mj}x_j\right); \qquad (4.43)$$

$$\sum_{j=1}^{n} a_{ij}x_j \le b_i, \quad i = \overline{1,d}; \qquad (4.44)$$

$$x_j \in \{0;1\}, \quad j = \overline{1,n}. \qquad (4.45)$$

Полагаем, что все определяющие данную задачу параметры – целые неотрицательные числа, при этом $b_i > 0$, $i = \overline{1,d}$; для каждого $k = \overline{1,m}$, $j = \overline{1,n}$ среди коэффициентов k-го критерия c_{kj} найдется по меньшей мере один ненулевой; для каждого $i = \overline{1,d}$, $j = \overline{1,n}$ среди коэффициентов i-го ограничения a_{ij} найдется по меньшей мере один ненулевой; кроме того, для всех $i = \overline{1,d}$, $j = \overline{1,n}$ имеет место $0 \le a_{ij} \le b_i$. Задачу (4.43) – (4.45) обозначим символом Z.

Для синтеза полной совокупности эффективных в рассматриваемой задаче оценок $E(Z)$ введем в рассмотрение совокупность частных задач $Z(k, \vec{p})$:

$$\max\left(\sum_{j=1}^{k} c_{1j}x_j, \sum_{j=1}^{k} c_{2j}x_j, ..., \sum_{j=1}^{k} c_{mj}x_j\right);$$

$$\sum_{j=1}^{k} a_{ij}x_j \le p_i, \quad i = \overline{1,d};$$

$$x_j \in \{0;1\}, \quad j = \overline{1,k},$$

где $k \in \{1,2,...,n\}$, $\vec{p} \in P$; здесь и далее P - множество всех целочисленных векторов $(p_1, p_2, ..., p_d)$ - с координатами, удовлетворяющими условиям $0 \le p_i \le b_i$, $i = \overline{1,d}$.

Полную совокупность эффективных оценок в задаче $Z(k, \vec{p})$ обозначим $E(k, \vec{p})$. При этом $E(Z) = E(n, \vec{b})$, здесь $\vec{b} = (b_1, b_2, ..., b_d)$.

Очевидно, что

$$E(1, \vec{p}) = \begin{cases} (0,0,...,0) & \text{если } \exists i : 0 \le p_i < a_{i1}; \\ (c_{11}, c_{21}, ..., c_{m1}) & \text{при } a_{i1} \le p_i \text{ для всех } i = \overline{1,d}, \end{cases} \qquad (4.46)$$

здесь $\vec{p} \in P$.

Условимся далее обозначать через $\vec{C}_j$ и $\vec{a}_j$ векторы $(c_{1j}, c_{2j}, ..., c_{mj})$ и $(a_{1j}, a_{2j}, ..., a_{dj})$ соответственно, $j = \overline{1,n}$.

Пусть уже найдены значения $E(k, \vec{p})$ для некоторого k и всех возможных $\vec{p}$ из P. Тогда значения $E(k+1, \vec{p})$ вычисляются по формуле

$$E(k+1,\vec{p}) = \begin{cases} E(k,\vec{p}),\ \textit{если}\ \exists i: 0 \le p_i < a_{ik+1}; \\ \textit{eff}\,(E(k,\vec{p}) \cup \{\vec{C}_{k+1} \oplus E(k,\vec{p}-\vec{a}_{k+1})\}) \\ \qquad \textit{при}\ a_{ik+1} \le p_i \ \text{для всех}\ i=\overline{1,d}, \end{cases} \qquad (4.47)$$

здесь $k = 1,2,...,n-1;\ \vec{p} \in P$.

Действительно, задача *Z(k+1, p)* отличается от $Z(k,\vec{p})$ дополнительным наличием булевозначной переменной x_{k+1}. В случае ∃i: $0 \le p_i < a_{ik+1}$ имеем $E(k+1,\ \vec{p}) = E(k,\ \vec{p})$, так как допустимым оказывается только нулевое значение данной переменной. Если же для всех $i=\overline{1,d}$ имеет место $a_{i\,k+1} \le p_i$, то для переменной x_{k+1} следует рассмотреть две возможности: $x_{k+1}= 0$ и $x_{k+1}= 1$. При реализации первой возможности допустимым набором значений остальных переменных может быть обеспечен любой вектор совокупности $E(k,\ \vec{p})$. При реализации второй возможности выбранным значением переменной x_{k+1} в критериальной вектор-функции обеспечивается слагаемое $\vec{C}_{k+1}$, допустимым набором значений остальных переменных в качестве второго слагаемого обеспечивается любой вектор совокупности $E(k,\ \vec{p}-\vec{a}_{k+1})$. Объединив совокупности векторов, получаемых при реализациях обеих возможностей и выделив из полученного множества эффективные оценки получаем совокупность $E(k+1,\ \vec{p})$.

Решение задачи Z сводится к последовательному (сверху вниз) заполнению строк таблицы 4.1. Заголовками строк являются допустимые значения первого аргумента функции $E(k,\vec{p})$, т.е. натуральные числа от 1 до n. Заголовками столбцов являются допустимые значения второго аргумента, т.е. векторы $\vec{p} = (p_1, p_2, ..., p_d)$, удовлетворяющие условию $0 \le p_i \le b_i,\quad i=\overline{1,d}$. Указанные векторы, общее их число $B = \prod_{i=1}^{d}(b_i+1)$, перечисляются как заголовки столбцов в лексикографическом порядке. В каждую клетку $(k,\vec{p})$ таблицы вносится конструируемое в соответствии с (4.46) - (4.47) множество векторов $E(k,\vec{p})$. Процесс решения задачи Z заканчивается нахождением множества $E(Z) = E(n,\vec{b})$.

Таблица 4.1. Таблица эффективных оценок

k \ $\vec{p}$	$\vec{p}_1$	$\vec{p}_2$	$\vec{p}_3$	…	$\vec{p}_{B-1}$	$\vec{p}_B = \vec{b}$
1	$E(1,\vec{p}_1)$	$E(1,\vec{p}_2)$	$E(1,\vec{p}_3)$	…	$E(1,\vec{p}_{B-1})$	$E(1,\vec{b})$
2	$E(2,\vec{p}_1)$	$E(2,\vec{p}_2)$	$E(2,\vec{p}_3)$		$E(2,\vec{p}_{B-1})$	$E(2,\vec{b})$
…	…	…	…	…	…	…
n	$E(n,\vec{p}_1)$	$E(n,\vec{p}_2)$	$E\ (\ n\ ,$		$E(n,\vec{p}_{B-1})$	$E(n,\vec{b})$

Изложенный метод решения задачи Z назовем алгоритмом $A^{мк}_{ранец}$. Отметим, что в частном случае, при $m=1$, соотношения (4.46) - (4.47) очевидным образом упрощаются, алгоритм $A^{мк}_{ранец}$ оказывается совпадающим с изложенным в главе 1 алгоритмом $A_{кзр}$.

Для обеспечения возможности восстановления по любой эффективной оценке порождающего ее Парето-оптимального решения, в процессе вычислений по

соотношениям (4.46) - (4.47) будем последовательно составлять список T с записями $\sigma^t =< \sigma_1^t, \sigma_2^t, \sigma_3^t >$, где

σ_1^t - эффективная в очередной частной задаче $Z(k, \vec{p})$ оценка;

$\sigma_2^t = k$;

$\sigma_3^t = \vec{p}$;

t – порядковый номер записи в списке.

Для каждой оценки $\vec{Q}$ соответствующая ей запись σ^t вносится в список T один раз, когда вектор $\vec{Q}$ впервые появляется в одной из клеток таблицы 4.1.

При определении для произвольной эффективной оценки $\vec{Q}^* \in E(n, \vec{b})$ обеспечивающего ее Парето-оптимального решения $\vec{X}^* = (x_1^*, x_2^*, ..., x_n^*)$ реализуем следующий алгоритм α:

1.Полагаем $\xi = n$, $\vec{\eta} = \vec{b}$.

2. В списке T находим запись $\sigma =< \vec{Q}^*, \xi', \vec{\eta}' >$, где $\xi' \le \xi$, $\vec{\eta}' \le \vec{\eta}$.

3. Полагаем $x_{\xi'}^* = 1$, $\vec{Q}^* = \vec{Q}^* - \vec{C}_{\xi'}$, $\xi = \xi' - 1$, $\vec{\eta} = \vec{\eta}' - \vec{a}_{\xi'}$; если $\vec{Q}^* = \vec{0}$, переходим к п.4, в противном случае – к п.2.

4. Все не определенные до настоящего момента компоненты вектора $\vec{X}^* = (x_1^*, x_2^*, ..., x_n^*)$ считаем нулевыми. Построенное решение $\vec{X}^*$ является искомым.

ПРИМЕР 4.4. Задана 2-х критериальная 2-мерная задача о ранце:

$$x_1 + x_2 + 2x_3 + 3x_4 \to \max \qquad (4.48)$$

$$4x_1 + 7x_2 + 2x_3 + x_4 \to \max \qquad (4.49)$$

$$x_1 + 2x_2 + x_3 + x_4 \le 3 \qquad (4.50)$$

$$2x_2 + x_3 + 2x_4 \le 2 \qquad (4.51)$$

$$x_j \in \{0;1\}, \quad j = \overline{1,4}. \qquad (4.52)$$

Для данной задачи требуется найти полную совокупность эффективных оценок; для каждой эффективной оценки нужно определить Парето-оптимальное решение, эту оценку порождающее.

Последовательно выполняя вычисления по формулам (4.46) - (4.47), заполним таблицу 4.2, являющуюся соответствующей конкретизацией таблицы 4.1 для решаемой задачи (4.48) - (4.52). Полученное заполнение крайне правой клетки последней строки таблицы показывает, что эффективными в рассматриваемой задаче являются оценки (2,11), (3,6) и (4,5).

Таблица 4.2. Таблица эффективных оценок для задачи (4.48) - (4.52).

	(0,0)	**(0,1)**	**(0,2)**	**(1,0)**	**(1,1)**	**(1,2)**	**(2,0)**	**(2,1)**	**(2,2)**	**(3,0)**	**(3,1)**	**(3,2)**
1	(0,0)	(0,0)	(0,0)	(1,4)	(1,4)	(1,4)	(1,4)	(1,4)	(1,4)	(1,4)	(1,4)	(1,4)
2	(0,0)	(0,0)	(0,0)	(1,4)	(1,4)	(1,4)	(1,4)	(1,4)	(1,7)	(1,4)	(1,4)	(1,7) (2,11)
3	(0,0)	(0,0)	(0,0)	(1,4)	(1,4) (2,2)	(1,4) (2,2)	(1,4)	(1,4) (2,2)	(1,7) (2,2) (3,6)	(1,4)	(1,4) (2,2)	(1,7) (2,11) (3,9)
4	(0,0)	(0,0)	(0,0)	(1,4)	(1,4) (2,2)	(1,4) (2,2) (3,1)	(1,4)	(1,4) (2,2) (3,6)	(1,7) (2,2) (3,6) (4,5)	(1,4)	(1,4) (2,2)	**(2,11)** **(3,6)** **(4,5)**

В ходе заполнения таблицы 4.2 в список T оказываются внесенными следующие записи:

$$\sigma^1 =< (1,4),1,(1,0) >; \qquad \sigma^2 =< (1,7),2,(2,2) >; \sigma^3 =< (2,11),2,(3,2) >;$$

$$\sigma^4 =< (2,2),3,(1,1) >; \qquad \sigma^5 =< (3,6),3,(2,2) >; \sigma^6 =< (3,9),3,(3,2) >;$$

$$\sigma^7 =< (3,1),4,(1,2) >; \qquad \sigma^8 =< (4,5),4,(2,2) >.$$

Применяя алгоритм α, восстанавливаем для каждой эффективной оценки порождающее ее Парето-оптимальное решение. Для оценки (2,11) — это $\vec{x}^1 = (1,1,0,0)$, для оценки (3,6) — $\vec{x}^2 = (1,0,1,0)$ и для оценки (4,5) — $\vec{x}^3 = (1,0,0,1)$.

Приведем еще один, также основанный на принципе динамического программирования, способ решения рассматриваемой задачи (4.43) - (4.45). Излагаемый алгоритм, назовем его методом последовательной генерации списков, является обобщением на многокритериальный случай изложенного в главе 3 алгоритма $\boldsymbol{B}_{кзр}$ решения классической задачи о ранце.

Введем необходимые обозначения и определения. Структуру вида $\left(\left\{ x_{r_1},...,x_{r_q} \right\}, \sum\limits_{i\in\{r_1,...r_q\}} \vec{C}_i, \sum\limits_{i\in\{r_1,...,r_q\}} \vec{a}_i \right)$ будем обозначать σ, далее $\vec{C}_\sigma = \sum\limits_{i\in\{r_1,...,r_q\}} \vec{C}_i$, $\vec{a}_\sigma = \sum\limits_{i\in\{r_1,...,r_q\}} \vec{a}_i$. Пусть $M = \{\sigma_1,...,\sigma_u\}$ — произвольное множество структур указанного вида. Будем говорить, что структура $\sigma' = \left(\left\{ x_{r_1'},..,x_{r_q'} \right\}, \vec{C}_{\sigma'}, \vec{a}_{\sigma'} \right)$ является *недоминируемой* в M если в этом множестве не найдется структуры $\sigma'' = \left(\left\{ x_{r_1''},...,x_{r_q''} \right\}, \vec{C}_{\sigma''}, \vec{a}_{\sigma''} \right)$, такой что выполняется следующая система неравенств:

$$\vec{C}_{\sigma''} \geq \vec{C}_{\sigma'}, \qquad (4.53)$$

$$\vec{a}_{\sigma'} \geq \vec{a}_{\sigma''}; \qquad (4.54)$$

причем по меньшей мере одно из $m+d$ неравенств этой системы выполняется как строгое неравенство (неравенства (4.53) и (4.54) трактуются как покоординатные). Через $(M)^*$ будем обозначать операцию выбора из M подмножества всех недоминируемых структур. Через $x_j \langle + \rangle \sigma$ будем обозначать операцию *допустимого дополнения структуры* $\sigma = \left(\left\{x_{r_1},...,x_{r_q}\right\}, \vec{C}_\sigma, \vec{a}_\sigma\right)$ элементом x_j:

$$x_j \langle + \rangle \sigma = \begin{cases} \left(\left\{x_{r_1},...,x_{r_q},x_j\right\}, (\vec{C}_j + \vec{C}_\sigma), \vec{a}_j + \vec{a}_\sigma\right), \text{если } \vec{b} - \vec{a}_\sigma \geq \vec{a}_j \\ \sigma, \textit{впротивномслучае} \end{cases}$$

При реализации для задачи Z метода последовательной генерации списков выполняется n-шаговая процедура, результатом каждого k-ого шага которой является очередной список L^k. При этом список L^1 состоит из структур $\left(\{\varnothing\}, (\vec{0},0), \vec{0}\right)$ и $\left(\{x_1\}, (\vec{C}_1,1), \vec{a}_1\right)$. Пусть сформирован список $L^k = \{\sigma_1,...,\sigma_u\}$. Тогда список L^{k+1} формируется согласно следующей формуле:

$$L^{k+1} = (L^k \cup \{x_{k+1} \langle + \rangle \sigma_i; i = \overline{1,u}\})^*$$

В результате выполнения n шагов получаем список L^n. Вторые компоненты каждой из входящих в этот список структур являются оценками в задаче Z; содержащиеся в этой совокупности оценок эффективные оценки образуют множество $E(Z)$.

ПРИМЕР 4.5. Для задачи (4.48) – (4.52) методом последовательной генерации списков построить полную совокупность эффективных оценок..

Последовательно строим следующие списки.

Список L^1: $\left(\{\varnothing\},(0,0),(0,0)\right)$, $\left(\{x_1\},(1,4),(1,0)\right)$.

Список L^2: $\left(\{\varnothing\},(0,0),(0,0)\right), \left(\{x_1\},(1,4),(1,0)\right), \left(\{x_2\},(1,7),(2,2)\right)$, $\left(\{x_1,x_2\},(2,11),(3,2)\right)$.

Список L^3: $\left(\{\varnothing\},(0,0),(0,0)\right), \left(\{x_1\},(1,4),(1,0)\right), \left(\{x_2\},(1,7),(2,2)\right)$, $\left(\{x_1,x_2\},(2,11),(3,2)\right)$, $\left(\{x_3\},(2,2),(1,1)\right)$, $\left(\{x_1,x_3\},(3,6),(2,1)\right)$.

Список L^4: $\left(\{\varnothing\},(0,0),(0,0)\right)$, $\left(\{x_1\},(1,4),(1,0)\right)$, $\left(\{x_2\},(1,7),(2,2)\right)$, $\left(\{x_1,x_2\},(2,11),(3,2)\right)$, $\left(\{x_3\},(2,2),(1,1)\right)$, $\left(\{x_1,x_3\},(3,6),(2,1)\right)$, $\left(\{x_4\},(3,1),(1,2)\right)$, $\left(\{x_1,x_4\},(4,5),(2,2)\right)$.

Среди оценок (0,0), (1,4), (1,7), (2,11), (2,2), (3,6), (3,1), (4,5), составляющих совокупность вторых компонент последнего списка, эффективными являются (2,11), (3,6), (4,5). По записям заключительного списка L^4, в которых эти оценки указаны, определяем, что оценка (2,11) порождается решением $\vec{x}^1 = (1,1,0,0)$, оценка (3,6) — решением $\vec{x}^2 = (1,0,1,0)$, оценка (4,5) — решением $\vec{x}^3 = (1,0,0,1)$.

Бикритериальные обобщения канонической задачи однопроцессорного обслуживания потока заявок. Рассматриваемые обобщения изученной в главе 2 канонической задачи обслуживания конечного детерминированного потока *R = {1, 2,...,*

$n\}$ заявок предполагают соотнесение с каждой заявкой i функций индивидуального штрафа $\varphi_i(t)$ и $\psi_i(t)$. Указанные функции определяют штрафы по первому и второму показателям соответственно, $i = \overline{1,n}$. Все функции индивидуального штрафа считаем, вообще говоря, нелинейными. В первом рассматриваемом варианте полагаем, что минимизируемыми критериями являются суммарный штраф по первому показателю и суммарный штраф по второму показателю. Во втором варианте минимизируемые критерии — суммарный штраф по первому показателю и максимальный из штрафов по второму показателю.

Задачу обслуживания конечного детерминированного потока заявок R с минимизируемыми критериями суммарных штрафов по первому и второму показателям обозначим через $\boldsymbol{P}$. Введем в рассмотрение ряд частных задач. Задача $\boldsymbol{P}(t, S)$ получается из исходной задачи $\boldsymbol{P}$ в предположении, что расписание надо строить начиная от дискретного момента времени t, при этом следует обслужить заявки совокупности S и все заявки, которые прибудут в систему обслуживания позднее этого момента; в момент t процессор считается свободным, S – произвольное подмножество заявок из числа поступивших (по исходным данным задачи $\boldsymbol{P}$) на обслуживание во временном отрезке $[0, t]$.

Пусть $\boldsymbol{E}(t, S)$ обозначает множество всех эффективных оценок в произвольной задаче $\boldsymbol{P}(t, S)$. Таким образом, $\boldsymbol{E}(0, V(0))$ – полная совокупность эффективных оценок в исходной задаче $\boldsymbol{P}$ (все принятые в главе 2 обозначения сохраняются; в частности, $V(0)$ – совокупность поступивших и ожидающих обслуживания заявок по состоянию на момент времени 0).

Очевидно, что для любого $\theta \geq 0$ и $S = \{j\}$, где j – любая заявка, имеет место

$$\boldsymbol{E}(t(n) + \theta, \{j\}) = (\varphi_j(t(n) + \theta + \tau(j))), \psi_j(t(n) + \theta + \tau(j))). \qquad (4.53)$$

Как и ранее, для любой пары аргументов (t, S) в случае $t < t(n)$ считаем, что в множестве S имеется (в явном виде она может быть не указана) фиктивная заявка 0; полагаем, что $\tau(0) = min\{\theta / D(t, t+\theta) \neq \varnothing\}$, а функции индивидуального штрафа $\varphi_0(t)$ и $\psi_0(t)$ равны нулю тождественно. Взятие на обслуживание фиктивной заявки фактически означает простой процессора от момента t до ближайшего момента, когда в систему обслуживания прибудет какая- либо новая заявка потока R.

Для общей, определяемой парой (t, S), ситуации в совокупности S требуется выбрать заявку, обслуживание которой начнется в момент t. Если из этой совокупности в качестве очередной обслуживаемой выбрана заявка i, то связанные с заявкой i индивидуальные штрафы по первому и второму показателям равны соответственно $\varphi_i(t+\tau(i))$ и $\psi_i(t+\tau(i))$, следующее решение по загрузке процессора будет приниматься для момента времени $t+\tau(i)$, выбор заявки будет осуществляться в множестве $(S\backslash\{i\}) \cup D(t, \tau(i))$. Поэтому

$$\boldsymbol{E}(t,S) = eff\Big[\bigcup_{i \in S} ((\varphi_i(t+\tau(i)), \psi_i(t+\tau(i))) \oplus \boldsymbol{E}(t+\tau(i), (S \setminus \{i\}) \cup D(t, \tau(i)))\Big]. \qquad (4.54)$$

Реализуя счет по рекуррентным соотношениям (4.53) - (4.54), мы в итоге находим множество $\boldsymbol{E}(0, V(0))$ – полную совокупность эффективных оценок для решаемой задачи $\boldsymbol{P}$.

Задача с минимизируемыми критериями суммарного штрафа по первому показателю и максимального из штрафов по второму показателю записывается в виде

$$\min_{\rho} \{\sum_{i=1}^{n} \varphi_i(t^*(\rho, i)), \max_{i} \psi_i(t^*(\rho, i))\}; \qquad (4.55)$$

будем именовать ее задачей ***Q***. Аналогично выполненному выше для задачи ***P***, введем в рассмотрение порождаемые ***Q*** частные задачи. Задача ***Q****(t, S)* получается из исходной в предположении, что расписание строится от момента времени t, где $t \geq 0$; следует обслужить заявки совокупности S и заявки, которые поступят на обслуживание позднее этого момента. В момент t процессор считается свободным, а S – произвольное подмножество заявок из числа поступивших (по исходным данным задачи ***Q***) на обслуживание во временном отрезке $[0,t]$. Пусть $E'(t, S)$ обозначает множество всех эффективных оценок в задаче ***Q****(t,S)*. Таким образом, $E'(0,V(0))$ – полная совокупность эффективных оценок в исходной задаче ***Q***. Для произвольного вектора (a,b) и множества оценок M через $(a,b)\otimes M$ мы обозначаем совокупность оценок, определяемую следующим образом: $(x,y) \in \{(a,b)\otimes M\}$ тогда и только тогда, когда в M имеется оценка (p,q) такая, что $x = a + p$ и $y = max\ (b,q)$.

Очевидно, что для любого $\theta \geq 0$ и любой заявки j имеет место

$$E'(t(n) + \theta, \{j\}) = (\varphi_j(t(n) + \theta + \tau(j)),\ \psi_j(t(n) + \theta + \tau(j))). \qquad (4.56)$$

Далее имеем

$$E'(t,S) = eff[\bigcup_{i \in S} ((\varphi_i(t+\tau(i)), \psi_i(t+\tau(i)) \otimes E'(t+\tau(i), (S \setminus \{i\}) \cup D(t,\tau(i)))]. \qquad (4.57)$$

Реализуя счет по рекуррентным соотношениям (4.56) - (4.57), находим в итоге множество $E'(0,V(0))$ – полную совокупность эффективных оценок для решаемой задачи (4.4.6).

Бикритериальные задачи о назначениях. Полные совокупности эффективных оценок в бикритериальных задачах о назначениях ранее строились нами методом последовательных уступок. Сейчас эти совокупности будет строиться методом динамического программирования.

Начнем с рассмотрения задачи (4.21):

$$\max_{\pi} \{K_1(A,\pi), K_2(B,\pi)\}.$$

Обозначим эту задачу символом ***Z***. Введем в рассмотрение совокупность частных задач ***Z****(i, W_i)*, формулируемых по исходным данным задачи ***Z***; здесь $i \in \{1,2,\ldots,n\}$, а W_i – произвольное i-элементное подмножество совокупности $\{1,2,\ldots,n\}$. В задаче ***Z****(i, W_i)* между исполнителями $1, 2,\ldots, i$ следует распределить совокупность работ, индексы которых перечислены в W_i; первый критерий задачи – суммарная производительность исполнителей совокупности $\{1, 2,\ldots, i\}$, второй критерий – минимальная из производительностей исполнителей данной совокупности; оба критерия являются максимизируемыми. Полную совокупность эффективных оценок в задаче ***Z****(i, W_i)* будем обозначать ***E****(i, W_i)*. Как очевидно

$$\boldsymbol{E}(1, \{j\}) = (a_{1j}, b_{1j}) \text{ для всех } j \in \{1,2,\ldots,n\}. \qquad (4.58)$$

В рамках рассматриваемой задачи для произвольного вектора (a,b) и множества оценок M через $(a,b)\otimes M$ обозначаем совокупность оценок, определяемую следующим образом: $(x,y) \in \{(a,b)\otimes M\}$ тогда и только тогда, когда в M имеется оценка (p,q) такая, что $x = a + p$ и $y = min\ (b,q)$.

Если $i>1$, имеем

$$\boldsymbol{E}(i, W_i) = eff\{\bigcup_{j\in W_i} [(a_{ij}, b_{ij}) \otimes E(i-1, W_i \setminus \{j\})\}, \; i= 2,3,\ldots,n. \quad (4.59)$$

Определив по формуле (4.58) все одноэлементные множества оценок для случая $i=1$, далее, пользуясь формулой (4.59), последовательно находим все множества $\boldsymbol{E}(2, W_2)$, все множества $\boldsymbol{E}(3, W_3)$ и т.д. до тех пор, пока не будет построено множество $\boldsymbol{E}(n, \{1, 2,\ldots,n\})$, т.е. совокупность всех эффективных в задаче $\boldsymbol{Z}$ оценок. Для обеспечения возможности определения по любой эффективной в $\boldsymbol{Z}$ оценке порождающего ее Парето-оптимального назначения, надо в процессе вычислений по формуле (4.59) для каждой пары (i, W_i) и каждого входящего в совокупность $\boldsymbol{E}(i, W_i)$ вектора запоминать, при каком значении j из W_i этот вектор был получен.

Сейчас рассмотрим бикритериальную задачу о назначениях с максимизируемыми критериями $Q_1(\pi) = \sum_{i=1}^{n} a_{i\,\pi(i)}$ и $Q_2(\pi) = \sum_{i=1}^{n} b_{i\,\pi(i)}$. Как и ранее, будем считать, что эта задача определяется парой (nxn)-матриц $A = \{a_{ij}\}$ и $B = \{b_{ij}\}$, элементы которых принадлежат множеству $N \cup \{\omega\}$, где N – совокупность всех целых неотрицательных чисел, а ω – специальный символ запрета. Назначение π допустимо, если $a_{i\pi(i)}$ отлично от ω при всех $i = \overline{1,n}$. Множество всех допустимых назначений обозначим символом H. Рассматриваемая задача записывается в виде

$$\max_{\pi\in H} \{Q_1(\pi), Q_2(\pi)\}, \quad (4.60)$$

Задачу (4.60) будем обозначать символом $\boldsymbol{Z}^0$. Введем в рассмотрение совокупность частных задач $\boldsymbol{Z}^0(i, W_i)$, формулируемых по исходным данным задачи $\boldsymbol{Z}^0$; здесь $i\in\{1,2,\ldots,n\}$, а W_i – произвольное i-элементное подмножество совокупности $\{1,2,\ldots,n\}$. В задаче $\boldsymbol{Z}^0(i, W_i)$ между исполнителями $\{1,2,\ldots, i\}$ следует распределить совокупность работ, индексы которых перечислены в W_i; критерии задачи – суммарные производительности исполнителей по первому и второму показателям.

Множество эффективных в задаче $\boldsymbol{Z}^0(i, W_i)$ оценок обозначим $\boldsymbol{E}^0(i, W_i)$. Как очевидно

$$\boldsymbol{E}^0(1, \{j\}) = (a_{1j}, b_{1j}) \text{ для всех } j\in\{1,2,\ldots,n\}. \quad (4.61)$$

Если $i>1$, имеем

$$\boldsymbol{E}^0(i, W_i) = eff\{\bigcup_{j\in W_i} [(a_{ij}, b_{ij}) \oplus E^0(i-1, W_i \setminus \{j\}]\}, \; i= 2,3,\ldots,n. \quad (4.62)$$

Определив по формуле (4.61) все одноэлементные множества оценок для случая $i=1$, далее, пользуясь формулой (4.62), последовательно находим все множества $\boldsymbol{E}^0(2, W_2)$, все множества $\boldsymbol{E}^0(3, W_3)$ и т.д. до тех пор, пока не будет построено множество $\boldsymbol{E}^0(n, \{1, 2,\ldots,n\})$, т.е. совокупность всех эффективных в задаче $\boldsymbol{Z}^0$ оценок. Для обеспечения возможности определения по любой эффективной в $\boldsymbol{Z}^0$ оценке порождающего ее Парето-оптимального назначения, надо в процессе вычислений по формуле (4.62) для каждой пары (i, W_i) и каждого входящего в совокупность $\boldsymbol{E}^0(i, W_i)$ вектора запоминать, при каком значении j из W_i этот вектор был получен.

Бикритериальная задача коммивояжера (БКЗК).

Бикритериальная задача коммивояжера (*БКЗК*) определяется полным взвешенным ориентированным графом без петель G с множеством вершин $N = \{1, 2, \ldots, n\}$; вес каждой дуги – двумерный вектор (первая координата – вес по первому показателю, вторая координата – вес по второму показателю). Допустимые решения задачи – гамильтоновы циклы графа G. Каждый гамильтонов цикл оценивается двумя критериями – весами по имеющимся показателям (вес цикла по показателю i определяется как сумма весов составляющих данный цикл дуг по этому показателю). Оба критерия считаются минимизируемыми.

Исходную информацию по *БКЗК* считаем представленной парой (nxn)-матриц $S=\{s_{ij}\}$,и $T=\{t_{ij}\}$ где s_{ij} и t_{ij} – веса дуги (i, j) графа G по первому и второму показателям соответственно, $i = \overline{1,n}$, $j = \overline{1,n}$, $i \neq j$; все элементы главных диагоналей матриц S и T считаются нулевыми. В типовой интерпретации вершины графа G – это города, которые коммивояжеру необходимо обойти по замкнутому маршруту, побывав в каждом городе ровно по одному разу; для определенности считаем, что маршрут коммивояжера начинается и заканчивается в первом городе. Веса дуг s_{ij} (по первому показателю) и t_{ij} (по второму показателю) трактуются как стоимости и продолжительности соответствующих переходов; требование симметричности на матрицы не налагается, не считается обязательным и выполнение неравенства треугольника.

Рассматриваемую *БКЗК* обозначим символом $\boldsymbol{Z}$. Полную совокупнсть эффективных в $\boldsymbol{Z}$ оценок $E(\boldsymbol{Z})$ будем строить методом динамического программирования. В изложении придерживаемся терминологии, соответствующей приведенной типовой интерпретации задачи.

Пусть i – произвольный город $(i \in N)$, а V – любое подмножество городов, не содержащее городов 1 и i. Через $M(i, V)$ обозначим совокупность путей, каждый из которых начинается в городе i, завершается в городе 1 и проходит в качестве промежуточных только через города множества V, заходя в каждый из них ровно по одному разу. Каждый путь из $M(i, V)$ оцениваем весами по первому и второму показателям, т.е. стоимостью и временем реализации; оба показателя считаем минимизируемыми критериями. Получаемую бикритериальную задачу будем обозначать $\boldsymbol{Z}(i, V)$. Множество эффективных в $\boldsymbol{Z}(i, V)$ оценок обозначим $\boldsymbol{E}(i, V)$. Как очевидно, задача $\boldsymbol{Z}(1, \{2, 3, \ldots, n\})$ совпадает с задачей $\boldsymbol{Z}$; поэтому $\boldsymbol{E}(1, \{2, 3, \ldots, n\}) = E(\boldsymbol{Z})$.

Если V – одноэлементное множество, $V = \{j\}$, где $j \neq 1$ и $j \neq i$, то совокупность $M(i, V)$ состоит из единственного пути $\pi = (i, j, 1)$. Поэтому

$$\boldsymbol{E}(i, \{j\}) = (s_{ij} + s_{j1}, t_{ij} + t_{j1}),\ i \in N,\ j \in \{2, 3, \ldots, n\},\ j \neq i. \qquad (4.63)$$

Предположим, что множества оценок $\boldsymbol{E}(i, V)$ для всех $i \in N$ и всех возможных k-элементных ($k < n - 1$) множеств V уже вычислены. Тогда значение $\boldsymbol{B}(i, V')$, где V' - произвольное $(k+1)$-элементное подмножество совокупности $N\setminus\{1, i\}$, вычисляется по формуле

$$\boldsymbol{E}(i, V') = eff\{\bigcup_{j \in V'} [(s_{ij}, t_{ij}) \oplus E(j, V' \setminus \{j\})\},\ i = 1,2,\ldots,n;\ V' \subseteq (N \setminus \{1, i\}). \quad (4.64)$$

Уравнения (4.63)-(4.64) – рекуррентные соотношения динамического программирования для решения бикритериальной задачи коммивояжера.

Бикритериальная задача инкассации. Задача определяется полным ориентированным графом без петель G с множеством вершин $N = \{1, 2, \ldots, n\}$.Каждая дуга и каждая вершина этого графа, кроме вершины *1*, имеет определенный вес. В интерпретации вершины $2,3,\ldots, n$ – это объекты, которые инкассатору необходимо обойти по замкнутому маршруту, посетив каждый из них ровно по одному разу. При этом считаем, что маршрут начинается и заканчивается в вершине *1* (банке). Веса дуг графа G трактуются как длины реализуемых по ним переходов, веса вершин $2, 3,\ldots, n$ – это денежные суммы, которые соответствующими объектами передаются инкассатору для транспортировки в банк. Из банка инкассатор выходит без денег, в банк он возвращается со всей собранной по объектам множества $\{2, 3,\ldots, n\}$ денежной суммой.

С целью упрощения обозначений будем считать, что вершина *1* также имеет вес, он равен нулю. Исходную информацию по задаче считаем представленной $(n\times n)$-матрицей $S=\{s(i, j)\}$и n-мерным вектором $\boldsymbol{d} = (d(1), d(2),\ldots,d(n))$, где $s(i, j)$ – вес дуги (i,j), а $d(i)$ – вес вершины i графа G, $i = \overline{1,n}$, $j = \overline{1,n}$, $i \neq j$. Все элементы главной диагонали матрицы S считаются нулевыми, первая координата вектора $\boldsymbol{d}$ равна нулю. Каждый гамильтонов цикл $C =(1, i_2, i_3,\ldots,i_n, 1)$, здесь $(i_2, i_3,\ldots,i_n)$ – некоторая перестановка чисел $2, 3,\ldots,n$, оцениваем критериями

$$K_1(C) = s(1,i_2)+ s(i_2,i_3)+\ldots+ s(i_{n-1},i_n)+ s(i_n,1)$$

и

$$K_2(C) = d(i_2)\, s(i_2,i_3)+ (d(i_2)+ d(i_3))\, s(i_3,i_4)+ (d(i_2)+ d(i_3)+ d(i_4))\, s(i_4,i_5)+\ldots$$
$$\ldots+(d(i_2)+ d(i_3)+ d(i_4)+\ldots+ d(i_n))\, s(i_n,1).$$

Первый критерий – это суммарная длина (стоимость) маршрута; второй критерий – суммарное количество деньго-километров. Оба критерия считаем минимизируемыми.

Сформулированную бикритериальную задачу инкассации обозначим символом Z. Пусть i – произвольная вершина графа ($i \in N$), а V – любое подмножество его вершин, не содержащее вершин *1* и i. Через $M(i, V)$ обозначаем совокупность путей, каждый из которых начинается в вершине i, завершается в вершине *1* и проходит в качестве промежуточных только через вершины множества V, заходя в каждую из них ровно по одному разу. Находящийся в вершине i инкассатор имеет собранной денежную сумму $R(N \setminus V) = \sum_{\alpha\in N\setminus V} d(\alpha)$; в каждой вершине j множества V он дополнительно получает сумму $d(j)$. Каждый путь из совокупности $M(i, V)$ оцениваем двумя показателями – общей длиной (стоимостью) и суммарным числом реализуемых деньго-километров. Возникающую частную бикритериальную задачу обозначаем $Z(i,V)$. Для совокупности эффективных в $Z(i,V)$. оценок принимаем обозначение $E(i,V)$. Как очевидно, $E(1,\{2,3,\ldots,n\})$ – полная совокупность эффективных оценок в задаче Z.

Если V – одноэлементное множество, $V = \{j\}$, где $j \neq 1$ и $j \neq i$, то совокупность $M(i, V)$ состоит из единственного пути $\pi = (i, j, 1)$. Поэтому

$$\boldsymbol{E}(i, \{j\}) = \{(s(i, j)+ s(j,1), R(N \setminus \{j\})s(i, j)+ R(N)\times s(j,1)\},\ i\in N,\ j\in\{2, 3,\ldots, n\},\ j \neq i. \quad (4.65)$$

Предположим, что множества оценок $\boldsymbol{E}(i, V)$ для всех $i\in N$ и всех возможных k-элементных ($k < n-1$) множеств V уже найдены. Тогда совокупность $E(i, V')$, где V' – произвольное $(k+1)$-элементное подмножество из $N\setminus\{1, i\}$, определяется по формуле

$$E(i, V') = eff\,[\bigcup_{j\in V'} \{(s(i, j), R(N \setminus V')\times s(i, j)) \oplus E(j, V' \setminus \{j\})\}]\ . \qquad (4.66)$$

Уравнения (4.65)-(4.66) – рекуррентные соотношения динамического программирования для решения бикритеральной задачи инкассации.

4.4. Вопросы построения представительных подмножеств эффективных оценок; концепция консервативного оператора.

В связи с высокой вычислительной сложностью проблемы синтеза для решаемых многокритериальных задач полных совокупностей эффективных оценок, возникают вопросы построения в рамках схемы многокритериального динамического программирования частных подмножеств, состоящих из эффективных оценок, удовлетворяющих некоторым дополнительным условиям и являющихся в том либо ином смысле достаточно представительными.

Оператором выбора назовем оператор, ставящий в соответствие каждому непустому множеству оценок M некоторое также непустое подмножество оценок $U(M)$ такое, что $U(M) \subseteq eff(M)$. Приведем несколько примеров операторов выбора: оператор, выбирающий в M подмножество крайних оценок; оператор, который лексикографически упорядочивает и в полученном порядке нумерует все эффективные в M оценки, а затем выбирает из них оценки с номерами *1*, *1+p, 1+2p, 1+3p* и т.д. (здесь p – натуральная константа, являющаяся параметром оператора); оператор, выбирающий из M оценки, порождаемые решениями, оптимальными в экстремальных задачах, получаемых посредством линейных сверток критериев (при этом считается, что весовые коэффициенты критериев варьируются в определенных диапазонах). Для произвольной задачи многокритериальной оптимизации Z поставим целью синтез подмножества $U(E(Z))$ без предварительного построения множества $E(Z)$, здесь $E(Z)$ – полная совокупность эффективных в задаче Z оценок.

Начнем рассмотрение с наиболее общей модели, в рамках которой формулируются все рассматриваемые многокритериальные задачи (как и ранее, исключительно для простоты и единообразия изложения ограничиваемся только задачами с двумя критериями).

Пусть

$$\boldsymbol{\Omega^+ = \{D;\ x_0;\ F;\ V(x),\ f(x,v),\ s_1(x,v),\ s_2\ (x,v)\}}$$

– дискретная управляемая система с двумя функциями платежа Для системы $\boldsymbol{\Omega^+}$ рассматриваем бикритериальные задачи (4.29) и (4.30). Полные совокупности эффективных в этих задачах оценок, как и ранее, обозначаем $\boldsymbol{E}$ и $\boldsymbol{E(x)}$ соответственно. При этом $\boldsymbol{E(x_0) = E.}$

Введем в рассмотрение множества $\boldsymbol{E_U(x)}$, определяемые следующими рекуррентными соотношениями

$$\boldsymbol{E_U(x)} = \{(0,0)\},\ \boldsymbol{x \in F}. \qquad (4.67)$$

$$\boldsymbol{E_U(x)} = U(\{\bigcup_{v\in V(x)} [(\boldsymbol{s_1(x,v),\ s_2\ (x,v))} \oplus \boldsymbol{E_U(f(x,\ v))}]\}),\ \boldsymbol{x \in D\setminus F}. \qquad (4.68)$$

Оператор выбора U назовем консервативным, если выполняются два следующих условия:

$$U(M_1 \cup M_2 \cup \ldots \cup M_n) = U(U(M_1) \cup U(M_2) \cup \ldots \cup U(M_n)); \qquad (4.65)$$

$$U(\boldsymbol{d} \oplus M_1) = \boldsymbol{d} \oplus U(M_1); \qquad (4.66)$$

здесь $M_1, M_2, \ldots, M_n$ – произвольные множества векторов (оценок) одинаковой размерности, $\boldsymbol{d}$ – произвольный вектор той же размерности.

Условия (4.65) и (4.66) будем называть первым и вторым условиями консервативности соответственно.

Т е о р е м а 4.4. Если оператор выбора U консервативен, то определяемая соотношениями (4.67)-(4.68) функция $\boldsymbol{E}_U(\boldsymbol{x})$ для любого $\boldsymbol{x} \in \boldsymbol{D}$ удовлетворяет условию

$$\boldsymbol{E}_U(\boldsymbol{x}) = U(\boldsymbol{E}(\boldsymbol{x})).$$

Доказательство данной теоремы легко выполняется методом математической индукции.

Из теоремы 4.4 следует, что для любого консервативного оператора U совокупность оценок $U(E) = \boldsymbol{E}_U(x_0)$ может быть построена применением рекуррентных соотношений динамического программирования (4.67)-(4.68).

Алгоритм, который по соотношениям (4.67)-(4.68) последовательно определяет множества оценок $\boldsymbol{E}_U(\boldsymbol{x})$, $x \in \boldsymbol{D}$, назовем алгоритмом $A(U)$. Отличие $A(U)$ от основанного на соотношениях (4.31)-(4.32) алгоритма синтеза полной совокупности эффективных оценок E заключается в следующем: на каждом этапе применения получаемое множество оценок, условно обозначим его M, заменяется подмножеством $U(M)$; подмножество $U(M)$ на последующих этапах работы алгоритма используется как заменитель множества M. Итогом работы алгоритма $A(U)$ является множество $\boldsymbol{E}_U(x_0)$. В случае, если оператор U консервативен, это множество совпадает с искомой совокупностью $U(E)$. Если процедура выделения из M подмножества $U(M)$ требует выполнения относительно небольшого числа элементарных операций, то при переходе от алгоритма синтеза полной совокупности эффективных оценок к алгоритму $A(U)$ число выполняемых арифметических операций уменьшается пропорционально отношению среднего числа оценок в множествах $\boldsymbol{E}(x)$ к среднему числу оценок в множествах $\boldsymbol{E}_U(x)$.

Приведем несколько примеров консервативных и неконсервативных операторов. Пусть M – непустое множество оценок. Оператор $\widetilde{U}$ осуществляет выбор из M подмножества крайних в M оценок. Легко проверяется, что для оператора $\widetilde{U}$ условия (4.65) - (4.66) выполняются. Следовательно, этот оператор консервативен и алгоритм $A(\widetilde{U})$ является способом построения полной совокупности крайних в решаемой задаче оценок.

Оператор U' осуществляет выбор из M непустого подмножества оценок, которые крайними не являются; в особом случае, когда все оценки множества M крайние, полагаем $U'(M) = M$. Оператор U' неконсервативен. Покажем это на следующем примере.

Введем в рассмотрение систему $\boldsymbol{\Omega}_a$, описываемую графом $\boldsymbol{G}_a$ с состояниями S, A^*, B^*, A_1, A_2, A_3, B_1, B_2, B_3, F. Состояние S – начальное, состояние F – финальное. В графе имеются следующие дуги: (S, A^*), (S, B^*), (A^*, A_1), (A^*, A_2), (A^*, A_3), (B^*, B_1), $(B^*,$

B_2), (B^*, B_3),), (A_1, F), (A_2, F), (A_3, F), (B_1, F),(B_2, F), (B_3, F). Векторные веса дуг *(A_1, F), (A_2, F), (A_3, F), (B_1, F),(B_2, F), (B_3, F)* равны соответственно *(1, 100), (10,10), (100, 1), (90, 11), (7, 13), (2, 90);* каждая из остальных дуг имеет вес *(0,0).* Применяя алгоритм *A(U'),* получаем последовательно: $\boldsymbol{E_{U'}(F)} = \{(0,0)\}$; $\boldsymbol{E_{U'}(A_1)} = \{(1, 100)\}$; $\boldsymbol{E_{U'}(A_2)} = \{(10,10)\}$; $\boldsymbol{E_U(A_3)} = \{(100, 1)\}$; $\boldsymbol{E_U(B_1)} = \{(90, 11)\}$; $\boldsymbol{E_{U'}(B_2)} = \{(7, 13)\}$; $\boldsymbol{E_{U'}(B_3)} = \{(2, 90)\}$; $\boldsymbol{E_{U'}(A^*)} = U'(\{(1, 100), (10,10), (100, 1)\}) = \{(10, 10)\}$; $\boldsymbol{E_{U'}(B^*)} = U'(\{(90, 11), (7, 13), (2, 90)\}) = \{(7, 13)\}$. Для начального состояния *S* имеем: $\boldsymbol{E_{U'}(S)} = \{(10, 10), (7, 13)\}$. В то же время, легко определяется, что $\boldsymbol{E= E(S)} = eff\{(1, 100), (10,10), (100, 1), (90, 11), (7, 13), (2,90)\} = \{(1, 100), (2,90), (7, 13), (90, 11), (100, 1)\}$. Отсюда $U'(E) = \{(2,90), (7, 13), (90, 11)\}$. Таким образом, в рассмотренном примере $\boldsymbol{E_{U'}(S)}$ не совпадает с $U'(\boldsymbol{E(S)})$, оператор U' неконсервативен. Отметим также, что в построенном применением алгоритма *A(U')* множестве $\boldsymbol{E_{U'}(S)}$ содержится оценка *(10,10),* которая в задаче $\boldsymbol{Z}$ эффективной не является.

Оценку $R = (r_1, r_2, ..., r_l)$ из M назовем s-оценкой (от *supported solution*, см. [27]), если она является оптимальным решением задачи

$$\max_{(y_1, y_2, ..., y_l) \in M} \sum_{i=1}^{l} \lambda_i y_i$$

при некотором наборе положительных коэффициентов $\lambda_1, \lambda_2, ..., \lambda_l$. Отметим, что для произвольной задачи многокритериальной оптимизации Z множество s-оценок из $E(Z)$ – это совокупность оценок всех решений, оптимальных в однокритериальных задачах, получаемых из Z линейной сверткой критериев. Весовые коэффициенты свертки $\lambda_1, \lambda_2, ..., \lambda_l$ – произвольные положительные числа. Можно дополнительно предположить, что сумма коэффициентов свертки равна единице. В таком случае при рассмотрении бикритериальных задач считается, что первый коэффициент свертки равен λ, а второй (1 - λ), где λ принадлежит интервалу (*0,1*).

Через U_S обозначим оператор выбора подмножества s-оценок. В [7] доказано, что

$$U_S(M) = eff(conv\, eff(M)) \cap eff(M), \qquad (4.69)$$

где $conv\, eff(M)$ - выпуклая оболочка множества $eff(M)$.

Лемма 4.3. Для любых множеств оценок $M_1, M_2, ..., M_n$ имеет место равенство

$$U_S(M_1 \cup M_2 \cup ... \cup M_n) = U_S(U_S(M_1) \cup U_S(M_2) \cup ... \cup U_S(M_n)).$$

Покажем сначала справедливость включения $U_S(M_1 \cup M_2 \cup ... \cup M_n) \subseteq U_S(U_S(M_1) \cup U_S(M_2) \cup ... \cup U_S(M_n))$. Действительно, пусть принадлежащая некоторому множеству M_j, $j \in \{1,2,...,n\}$, оценка m входит в $U_S(M_1 \cup M_2 \cup ... \cup M_n)$, выделяющий ее в совокупности $M_1 \cup M_2 \cup ... \cup M_n$ вектор весовых коэффициентов обозначим λ. В таком случае тот же вектор λ выделяет оценку m сначала в совокупности M_j; затем в совокупности $U_S(U_S(M_1) \cup U_S(M_2) \cup ... \cup U_S(M_n))$.

Выполнение обратного включения докажем методом от противного. Пусть входящая в множество $U_S(U_S(M_1) \cup U_S(M_2) \cup ... \cup U_S(M_n))$ оценка m совокупности $U_S(M_1 \cup M_2 \cup ... \cup M_n)$ не принадлежит. Так как

$$m \in U_S(U_S(M_1) \cup U_S(M_2) \cup ... \cup U_S(M_n)),$$

то имеется вектор λ^* такой, что для любой оценки t из $U_S(M_1) \cup U_S(M_2) \cup ... \cup U_S(M_n)$ выполняется неравенство:

$$(\lambda^*, m) \geq (\lambda^*, t) \qquad (4.70)$$

Вместе с тем, так как оценка m совокупности $U_S(M_1 \cup M_2 \cup ... \cup M_n)$ не принадлежит, для вектора λ^* в некотором множестве M_j, $j \in \{1,2,...,n\}$, найдется оценка r такая, что:

$$(\lambda^*, r) > (\lambda^*, m). \qquad (4.71)$$

В совокупности $U_S(M_j)$, согласно ее определению, обязательно имеется оценка t' такая, что:

$$(\lambda^*, t') \geq (\lambda^*, r). \qquad (4.72)$$

Из (4.71) и (4.72) имеем:

$$(\lambda^*, t') > (\lambda^*, m). \qquad (4.73)$$

Но так как t' принадлежит совокупности $U_S(M_1) \cup U_S(M_2) \cup ... \cup U_S(M_n)$, на основании (4.70) получаем:

$$(\lambda^*, m) \geq (\lambda^*, t'),$$

что противоречит (4.73).

Лемма доказана.

Лемма 4.4. Для любого множества *l*-мерных векторов-оценок M и произвольного вектора $\boldsymbol{d}$ той же размерности имеет место равенство

$$U_S(\boldsymbol{d} \oplus M) = \boldsymbol{d} \oplus U_S(M).$$

Действительно, оценка $d + m$ принадлежит совокупности $U_S(d + m)$ тогда и только тогда, когда существует вектор λ^* такой, что для любой оценки t из $d \oplus M$ выполняется неравенство $(\lambda^*, d + m) \geq (\lambda^*, t)$. Но это эквивалентно выполнению для любой оценки x из M неравенства $(\lambda^*, m) \geq (\lambda^*, x)$. Последнее же означает, что оценка m принадлежит множеству $U_S(M)$, т.е. вектор $d + m$ входит в совокупность $d \oplus U_S(M)$.

Лемма доказана.

Т е о р е м а 4.5. Оператор U_S консервативен.

Сформулированная теорема является прямым следствием лемм 4.3 и 4.4.

Сейчас введем оператор *U*,* который из совокупности оценок M выделяет непустое подмножество оценок, не являющихся s-оценками; в случае, когда $U_S(M) = M$ считаем, что $U^*(M)$ также совпадает с M. Покажем, что оператор U^* консервативным не является.

Рассмотрим введенную выше систему $\boldsymbol{\Omega_a}$,. Применяя алгоритм *A(U*),* получаем последовательно: $\boldsymbol{E_{U^*}(F)} = \{(0,0)\}$; $\boldsymbol{E_{U^*}(A_1)} = \{(1, 100)\}$; $\boldsymbol{E_{U^*}(A_2)} = \{(10,10)\}$; $\boldsymbol{E_{U^*}(A_3)} = \{(100, 1)\}$; $\boldsymbol{E_{U^*}(B_1)} = \{(90, 11)\}$; $\boldsymbol{E_{U^*}(B_2)} = \{(7, 13)\}$; $\boldsymbol{E_{U^*}(B_3)} = \{(2, 90)\}$; $\boldsymbol{E_{U^*}(A^*)} = U'(\{(1, 100), (10,10), (100, 1)\}) = \{(10, 10)\}$; $\boldsymbol{E_{U^*}(B^*)} = U'(\{(90, 11), (7, 13), (2, 90)\}) = \{(7, 13)\}$. Для начального состояния *S* имеем: $\boldsymbol{E_{U^*}(S)} = \{(10, 10), (7, 13)\}$.

В то же время $\boldsymbol{E = E(S)} = \{(1, 100), (2,90), (7, 13), (90, 11), (100, 1)\}$.Далее получаем $U^*(E) = \{(2,90), (7, 13)\}$. Таким образом, в рассмотренном примере $\boldsymbol{E_{U^*}(S)}$ не совпадает с $U^*(\boldsymbol{E})$, оператор U^* неконсервативен.

Оценку $R = (r_1, r_2, ..., r_l)$ из M назовем $\boldsymbol{S^2}$-оценкой, если она является оптимальным решением задачи

$$\max_{(y_1, y_2, ..., y_l) \in M} \sum_{i=1}^{l} \lambda_i y_i^2 \qquad (4.74)$$

при некотором наборе положительных коэффициентов $\lambda_1, \lambda_2, ..., \lambda_m$.

Через U^2_S обозначим оператор выбора подмножества $\boldsymbol{S^2}$-оценок.

Т е о р е м а 4.6. Оператор U^2_S неконсервативен.

Покажем, что для оператора U^2_S не выполняется второе условие консервативности.

Пусть $M = \{(9,1),(6,6), (1,9)\}$ и $d = (11,11)$. В таком случае $U^2_S(M) = \{(9,1), (1,9)\}$. Действительно, оценка *(6,6)* в множество $U^2_S(M)$ не входит, ибо неравенства

$$36\lambda + 36(1-\lambda) \geq 81\lambda + (1-\lambda)$$

и

$$36\lambda + 36(1-\lambda) \geq \lambda + 81(1-\lambda)$$

ни при каком значении параметра λ одновременно не выполняются (как легко подсчитывается, первое изнеравенство имеет место при $\lambda \leq 7/16$, а второе неравенство - при $\lambda \geq 9/16$). В то же время неравенство

$$81\lambda + (1-\lambda) \geq \lambda + 81(1-\lambda)$$

имеет место при $\lambda \geq 0{,}5$; неравенство

$$\lambda + 81(1-\lambda) \geq 81\lambda + (1-\lambda)$$

верно при $\lambda \leq 0{,}5$.

Получаем, что $\boldsymbol{d} \oplus U^2_S(M) = \{(20,12), (12,20)\}$.

Далее имеем: $\boldsymbol{d} \oplus M = \{(20,12), (17, 17) (12,20)\}$ и $U^2_S(\boldsymbol{d} \oplus M) = \{(20,12), (17, 17) (12,20)\}$. Действительно, оценка *(20,12)* в множество $U^2_S(\boldsymbol{d} \oplus M)$ входит, ибо неравенства

$$400\lambda + 144(1-\lambda) \geq 289\lambda + 289(1-\lambda)$$

и

$$400\lambda + 144(1-\lambda) \geq 144\lambda + 400(1-\lambda)$$

имеют место, например, при = $\lambda = 0{,}9$. Оценка *(17,17)* в множество $U^2_S(\boldsymbol{d} \oplus M)$ входит, ибо неравенства

$$289\lambda + 289(1-\lambda) \geq 400\lambda + 144(1-\lambda)$$

и

$$289\lambda + 289(1-\lambda) \geq 144\lambda + 400(1-\lambda)$$

имеют место, например, при = $\lambda = 0{,}5$. Оценка *(12,20)* в множество $U^2_S(\boldsymbol{d} \oplus M)$ входит, ибо неравенства

$$144\lambda + 400(1-\lambda) \geq 400\lambda + 144\,(1-\lambda)$$

и

$$144\lambda + 400(1-\lambda) \geq 289\,\lambda + 289(1-\lambda)$$

имеют место, например, при = $\lambda = 0{,}1$.

Таким образом, множества $U^2_S(\boldsymbol{d} \oplus M)$ и $\boldsymbol{d} \oplus U^2_S(M)$ не совпадают, оператор U^2_S консервативным не является. Теорема доказана.

Если используемый оператор выбора U консервативен, то для каждой конкретной, моделируемой системой $\boldsymbol{\Omega}^+$ бикритериальной задачи Z синтез совокупности оценок, выделяемых оператором U из полной совокупности $E(Z)$ можно осуществлять использованием рекуррентных соотношений (4.67) - (4.68), адаптированных для задачи Z.

В качестве иллюстрации рассмотрим вопрос построения полной совокупности s-оценок для бикритериальной задачи о ранце.

$$5x_1 + 3x_2 + 4x_3 + 6x_4 + 2x_5 + x_6 \to \max \qquad (4.75)$$

$$x_1 + 2x_2 + 3x_3 + x_4 + 5x_5 + 2x_6 \to \max \qquad (4.76)$$

$$3x_1 + 4x_2 + 3x_3 + 2x_4 + 3x_5 + x_6 \leq 10 \qquad (4.77)$$

$$x_j \in \{0;1\},\ j = \overline{1,6} \qquad (4.78)$$

Определяя для этой задачи полную совокупность эффективных оценок основанным на на ранее записанных соотношениях

$$E(1,\vec{p}) = \begin{cases} (0,0,...,0) & \textit{если } \exists i : 0 \leq p_i < a_{i1}\ ; \\ (c_{11}, c_{21}, ..., c_{m1}) & \textit{при } a_{i1} \leq p_i \text{ для всех } i = \overline{1,d}, \end{cases} \qquad (4.46)$$

$$E(k+1,\vec{p}) = \begin{cases} E(k,\vec{p}), & \textit{если } \exists i : 0 \leq p_i < a_{ik+1}\ ; \\ \textit{eff}\,(E(k,\vec{p}) \cup \{\vec{C}_{k+1} \oplus E(k,\vec{p} - \vec{a}_{k+1})\}) & \\ \qquad \textit{при } a_{ik+1} \leq p_i \text{ для всех } i = \overline{1,d}, \end{cases} \qquad (4.47)$$

алгоритмом $A^{мк}_{ранец}$, заполняем таблицу 4.3.

При синтезе для задачи (4.75)-(4.78) полной совокупности s-оценок используем соотношения

$$E_{U_s}(1,\vec{p}) = \begin{cases} (0,0,...,0) & \textit{если } \exists i : 0 \leq p_i < a_{i1}\ ; \\ (c_{11}, c_{21}, ..., c_{m1}) & \textit{при } a_{i1} \leq p_i \text{ для всех } i = \overline{1,d}, \end{cases} \qquad (4.79)$$

$$E_{U_s}(k+1,\vec{p}) = \begin{cases} E_{U_s}(k,\vec{p}), & \textit{если } \exists i : 0 \leq p_i < a_{ik+1}\ ; \\ U_s(E_{U_s}(k,\vec{p}) \cup \{\vec{C}_{k+1} \oplus E_{U_s}(k,\vec{p} - \vec{a}_{k+1})\}) & \\ \qquad \textit{при } a_{ik+1} \leq p_i \text{ для всех } i = \overline{1,d}, \end{cases} \qquad (4.80)$$

Отличие процедуры счета по соотношениям (4.79)-(4.80) от процесса счета по соотношениям (4.46)-(4.47) в том, что при заполнении каждой очередной клетки таблицы получаемое множество оценок, условно обозначим его M, заменяется подмножеством $U_S(M)$; подмножество $U_S(M)$ на последующих этапах работы алгоритма используется как заменитель множества M. Итогом работы примененной к

исходным данным задачи (4.75)-(4.78) и основанной на соотношениях (4.79)-(4.80) процедуры является множество $E_{U_s}(6{,}10)$.

Так как оператор U_s консервативен, найденное множество $E_{U_s}(6{,}10)$={*(16, 7), (13, 11)*} – полная совокупность s-оценок в рассматриваемой задаче.

Выполним соответствующую проверку. Как показывает заполнение таблицы 4.3, полная совокупность эффективных в (4.75)-(4.78) оценок трехэлементна: *(16, 7), (14, 9)* и *(13, 11)*. Оценки *(16, 7)* и *(13, 11)* являются s-оценками, ибо в задаче (4.75)-(4.78) они – крайние оценки. Оценка *(14, 9)* s-оценкой не является, так как неравенства

$$14\cdot\lambda+9\cdot(1-\lambda)\geq 16\cdot\lambda+7\cdot(1-\lambda)$$

$$14\cdot\lambda+9\cdot(1-\lambda)\geq 13\cdot\lambda+11\cdot(1-\lambda)$$

не выполняются одновременно ни при каком значении λ. Множество U_S({*(16, 7), (14, 9), (13, 11)*}) действительно совпадает с $E_{U_s}(6{,}10)$.

Таблица 4.3. Таблица эффективных оценок для задачи (4.48) - (4.52).

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	(0,0)	(0,0)	(0,0)	(5,1)	(5,1)	(5,1)	(5,1)	(5,1)	(5,1)	(5,1)	(5,1)
2	(0,0)	(0,0)	(0,0)	(5,1)	(5,1) (3,2)	(5,1) (3,2)	(5,1) (3,2)	(8,3)	(8,3)	(8,3)	(8,3)
3	(0,0)	(0,0)	(0,0)	(5,1) (4,3)	(5,1) (4,3)	(5,1) (4,3)	(9,4)	(9,4) (7,5)	(9,4) (7,5)	(9,4) (7,5)	(12,6)
4	(0,0)	(0,0)	(6,1)	(6,1) (4,3)	(6,1) (4,3)	(11,2) (10,4)	(11,2) (10,4)	(11,2) (10,4) (7,5)	(15,5)	(15,5) (13,6)	(15,5) (13,6)
5	(0,0)	(0,0)	(6,1)	(6,1) (2,5) (4,3)	(6,1) (2,5) (4,3)	(11,2) (10,4) (8,6)	(11,2) (10,4) (8,6) (6,8)	(11,2) (10,4) (8,6) (6,8)	(15,5) (13,7) (12,9)	(15,5) (13,7) (12,9)	(15,5) (13,7) (12,9) (9,10)
6	(0,0)	(1,2)	(6,1) (1,2)	(6,1) (2,5) (4,3)	(7,3) (3,7) (5,5)	(11,2) (10,4) (8,6) (3,7)	(12,4) (11,6) (9,8)	(12,4) (11,6) (9,8) (7,10)	(7,10) (15,5) (13,7) (12,9)	(16,7) (14,9) (13,11)	(16,7) (14,9) (13,11)

Таблица 4.4. Таблица *s*-оценок для задачи (4.48) - (4.52).

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	(0,0)	(0,0)	(0,0)	(5,1)	(5,1)	(5,1)	(5,1)	(5,1)	(5,1)	(5,1)	(5,1)
2	(0,0)	(0,0)	(0,0)	(5,1)	(5,1) (3,2)	(5,1) (3,2)	(5,1) (3,2)	(8,3)	(8,3)	(8,3)	(8,3)
3	(0,0)	(0,0)	(0,0)	(5,1) (4,3)	(5,1) (4,3)	(5,1) (4,3)	(9,4)	(9,4) (7,5)	(9,4) (7,5)	(9,4) (7,5)	(12,6)
4	(0,0)	(0,0)	(6,1)	(6,1) (4,3)	(6,1) (4,3)	(11,2) (10,4)	(11,2) (10,4)	(11,2) (10,4) (7,5)	(15,5)	(15,5) (13,6)	(15,5) (13,6)
5	(0,0)	(0,0)	(6,1)	(6,1) (2,5) (4,3)	(6,1) (2,5) (4,3)	(11,2) (10,4) (8,6)	(11,2) (10,4) (6,8)	(11,2) (10,4) (6,8)	(15,5) (12,9)	(15,5) (12,9)	(15,5) (12,9) (9,10)
6	(0,0)	(1,2)	(6,1) (1,2)	(6,1) (2,5) (4,3)	(7,3) (3,7) (5,5)	(11,2) (10,4) (8,6) (3,7)	(12,4) (11,6) (9,8)	(12,4) (11,6) (7,10)	(7,10) (15,5) (12,9)	(16,7) (13,11)	(16,7) (13,11)

ЗАДАЧИ К ГЛАВЕ 4.

1. Методом последовательных уступок построить полную совокупность эффективных оценок для задачи о назначениях с двумя неоднотипными максимизируемыми критериями $K_1(A,\pi) = \sum_{i=1}^{n} a_{i\pi(i)}$ и $K_2(B,\pi) = \min_i b_{i\pi(i)}$. Исходная информация по задаче считается определенной парой матриц

$$A = \begin{matrix} 9 & 7 & 7 & 4 & 3 \\ 1 & 9 & 8 & 2 & 1 \\ 3 & 4 & 7 & 4 & 9 \\ 3 & 1 & 3 & 8 & 5 \\ 8 & 2 & 2 & 6 & 8 \end{matrix} \quad \text{и} \quad B = \begin{matrix} 0 & 6 & 5 & 1 & 9 \\ 1 & 0 & 7 & 9 & 8 \\ 9 & 2 & 1 & 1 & 7 \\ 0 & 9 & 1 & 9 & 5 \\ 3 & 6 & 8 & 7 & 4 \end{matrix}$$

Для каждой эффективной оценки указать порождающее ее Парето-оптимальное решение.

2. Предыдущую задачу решить с использованием рекуррентных соотношений бикритериального динамического программирования.

3. Используя метод последовательных уступок, найти две крайние оценки и по одной эффективной оценке, соседней с каждой крайней оценкой, для задачи о

назначениях с критериями $Q_1(\pi) = \sum_{i=1}^{n} a_{i\ \pi(i)}$ и $Q_2(\pi) = \sum_{i=1}^{n} b_{i\ \pi(i)}$. Исходная информация по задаче считается определенной парой матриц

$$A = \begin{matrix} 9 & 8 & 7 & 4 & 3 \\ 1 & 9 & 8 & 2 & 1 \\ 3 & 4 & 7 & 4 & 9 \\ 3 & 1 & 3 & 8 & 5 \\ 3 & 2 & 2 & 6 & 8 \end{matrix} \quad \text{и} \quad B = \begin{matrix} 0 & 5 & 5 & 1 & 9 \\ 1 & 0 & 7 & 9 & 8 \\ 9 & 2 & 8 & 1 & 7 \\ 0 & 9 & 1 & 1 & 5 \\ 6 & 6 & 8 & 7 & 4 \end{matrix}$$

4. Предыдущую задачу решить с использованием рекуррентных соотношений бикритериального динамического программирования.

5. Алгоритмом $A^{мк}_{ранец}$ построить полную совокупность эффективных оценок для трехкритериальной задачи о ранце

$$7x_1 + 3x_2 + 6x_3 + 9x_4 + x_5 + 4x_6 \to max$$

$$4x_1 + 4x_2 + 4x_3 + 2x_4 + 4x_5 + 6x_6 \to max$$

$$x_1 + 5x_2 + x_3 + 5x_4 + 5x_5 + 3x_6 \to max$$

при условиях

$$2x_1 + x_2 + 2x_3 + 4x_4 + 3x_5 + 2x_6 \le 10;$$

$x_i \in \{0,1\}$, $i = \overline{1,5}$.

Для каждой эффективной оценки указать порождающее ее Парето-оптимальное решение.

6. Предыдущую задачу решить методом последовательной генерации списков.

7. Задача однопроцессорного обслуживания множества заявок определяется следующими исходными данными: обслуживанию подлежат заявки *1,2,3,4,5*; продолжительности обслуживания заявок: $\tau(1) = 3,\ \tau(2) = 2,\ \tau(3) = 4,\ \tau(4) = 1,\ \tau(5) = 2;$ функции индивидуального штрафа по заявкам: $\varphi_1(t) = 4t^2,\ \varphi_2(t) = 3t,\ \varphi_3(t) = t^2 + t,\ \varphi_4(t) = t^2 + 2t,\ \varphi_5(t) = t^3 + 1$. Рассматриваются два минимизируемых критерия – суммарный штраф по всем заявкам и максимальный из индивидуальных штрафов. Требуется построить полную совокупность эффективных оценок. Для двух крайних оценок указать порождающие их Парето-оптимальные решения.

8. Пусть M – множество оценок в произвольной бикритериальной задаче. Является ли консервативным оператор $U_{\lambda,q}(M)$, который для каждой оценки (x,y) из M вычисляет характеристику $\lambda x + (1-\lambda)y$, упорядочивает множество оценок M по убыванию указанной характеристики и далее выбирает из M оценки, стоящие на q первых местах (здесь λ – произвольная константа из интервала $(0,1)$, q – произвольная натуральная константа).

9. Для задачи о ранце

$7x_1 + 2x_2 + 6x_3 + 9x_4 + x_5 + 4x_6 \to max$

$x_1 + 5x_2 + x_3 + 5x_4 + 5x_5 + 7x_6 \to max$

при условиях

$2x_1 + x_2 + 2x_3 + 4x_4 + 3x_5 + 2x_6 \leq 10;$

$x_i \in \{0,1\}$, $i = \overline{1,5}$;

построить полную совокупность s-оценок.

10. Для бикритериальной задачи о назначениях с критериями $Q_1(\pi) = \sum_{i=1}^{n} a_{i\,\pi(i)}$ и $Q_2(\pi) = \sum_{i=1}^{n} b_{i\,\pi(i)}$ построить полную совокупность s-оценок. Исходная информация по задаче считается определенной парой матриц

$$A = \begin{matrix} 9 & 8 & 7 & 4 & 3 \\ 1 & 9 & 8 & 2 & 1 \\ 3 & 4 & 7 & 4 & 9 \\ 3 & 1 & 3 & 8 & 5 \\ 3 & 2 & 2 & 6 & 8 \end{matrix} \quad \text{и} \quad B = \begin{matrix} 0 & 5 & 5 & 1 & 9 \\ 1 & 0 & 7 & 9 & 8 \\ 9 & 2 & 8 & 1 & 7 \\ 0 & 9 & 1 & 1 & 5 \\ 6 & 6 & 8 & 7 & 4 \end{matrix}$$

ЛИТЕРАТУРА

1. Алексеев О.Г. Комплексное применение методов дискретной оптимизации. - М.: Наука, 1987. – 247 с.
2. Батищев Д.И., Коган Д.И., Лейкин М.В. Вопросы синтеза совокупностей эффективных оценок в многокритериальной задаче о ранце // Математическое моделирование и оптимальное управление: Вестник Нижегородского университета. – Н.Новгород, 2002, вып.1 (25), с. 211-223.
3. Беллман Р. Динамическое программирование. - М.: ИЛ, 1960.
4. Беллман Р., Дрейфус С. Прикладные задачи динамического программирования. - М.: Наука, 1965. - 457 с.
5. Габасов Р., Кириллова Ф.М. Основы динамического программирования.- Минск: Издательство Белорусского университета, 1975.-262 с.
6. Гейл Д. Теория линейных экономических моделей.-М.: ИЛ, 1963. – 418 с.
7. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. - М.: Мир, 1982. – 416 с.
8. Коган Д.И. Дискретные многокритериальные задачи распределительного типа.- Нижний Новгород: Издательство Нижегородского университета, 1991.- 82 с.
9. Коган Д.И., Федосенко Ю.С. Задача диспетчеризации: анализ вычислительной сложности и полиномиально разрешимые подклассы. Дискретная математика, 1996 г.,т.8, N 3, с.135-147.
10. Корбут А.А., Финкельштейн Ю.Ю. Дискретное программирование.- М.: Наука, 1969.-368с.
11. Кристофидес Н. Теория графов. Алгоритмический подход. - М.: Мир, 1988. – 432 с.
12. Липский В. Комбинаторика для программистов. - М.: Мир, 1978. – 213 с.
13. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. - М.: Мир, 1985. - 510 с.
14. Подиновский В.В., Ногин В.Д. Парето - оптимальные решения многокритериальных задач. – М.: Наука, 1982.
15. Рубинштейн М.И. Об алгоритмах решения задачи о назначении // Автоматика и телемеханика.1981. N 7 с.145 – 154.
16. Сигал И.Х., Иванова А.П. Введение в прикладное дискретное программирование.- М.: Наука, 2002. – 237 с.
17. Танаев В.С., Гордон В.С., Шафранский Я.М. Теория расписаний. Одностадийные системы. - М.: Наука, 1984. - 382 с.
18. Танаев В.С., Сотсков Ю.Н., Струсевич В.А. Теория расписаний. Многостадийные системы. - М.: Наука, 1989. - 328 с.
19. Танаев В.С., Шкурба В.В. Введение в теорию расписаний. - М.: Наука, 1975. - 256 с.
20. Хедли Дж. Нелинейное и динамическое программирование.-М.: Мир, 1967.- 506 с.
21. Юдин Д.Б., Гольштейн Е.Г. Линейное программирование. - М.: Наука, 1969.
22. Klamroth K., Wiecek M. Dynamic Programming Approaches to the Multiple Criteria Knapsack Problem. – Technical Report #666, Dept. of Math. Sc., Clemson University, Clemson, SC, 1998. (www.math.clemson.edu/affordability/publications/kla-wie3.ps)

Литература:

1.
2.

3. Коган Д.И., Федосенко Ю.С. Задача диспетчеризации: анализ вычислительной сложности и полиномиально разрешимые подклассы. Дискретная математика, 1996 г.,т.8, N 3, с.135-147.

4. Беллман Р., Дрейфус С. Прикладные задачи динамического программирования. - М.: Наука, 1965. - 457 с.

5. Алексеев О.Г. Комплексное применение методов дискретной оптимизации. - М.: Наука, 1987. – 247 с.

1. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. - М.: Мир, 1982. - 416 с.
2. Танаев В.С., Гордон В.С., Шафранский Я.М. Теория расписаний. Одностадийные системы. - М.: Наука, 1984. - 382 с.

Кладов-Лившиц. Шкуба,Танаев

1. Подиновский В.В., Ногин В.Д. Парето-оптимальные решения многокритериальных задач. – М.: Наука, 1982.
2. Асанов М.О., Баранский В.А., Расин В.В. Дискретная математика: графы, матроиды, алгоритмы. - Москва -Ижевск : Издательство РХД, 2001.
3. Беллман Р. Динамическое программирование. - М.: ИЛ, 1960.
4. БеллманДрейфус. 5. Процессы регул с адаптацией6. Болтянский, 7. ГабасовКириллова
5. 8. Хедли
6. Гейл* Рубинштейн-статья
7. Коган Д.И. Двухкритериальные задачи о назначениях: оценки сложно сти и алгоритмы решения. Известия РАН. Теория и системы управления, 1996 г., N3, с. 80-85.
8. Батищев Д.И., Коган Д.И., Лейкин М.В. Вопросы синтеза совокупностей эффективных оценок в многокритериальной задаче о ранце // Математическое моделирование и оптимальное управление: Вестник Нижегородского университета, вып.1 (25), 2002, с.211-223.
9. Klamroth K., Wiecek M. Dynamic Programming Approaches to the Multiple Criteria Knapsack Problem. – Technical Report #666, Dept. of Math. Sc., Clemson University, Clemson, SC, 1998.

(Доступна в Интернет: www.math.clemson.edu/affordability/publications/kla-wie3.ps)

7. Меламед И.И., Сигал И.Х., Владимирова Н.Ю. Исследование линейной свертки критериев в бикритериальной задаче о ранце // Журнал

вычислительной математики и математической физики, Т.39, №5, 1999, с.753-758.

1. 1. Танаев В.С., Гордон В.С., Шафранский Я.М. Теория расписаний. Одностадийные системы . - М.: Наука, 1984. - 382 с.
2. Беллман Р., Дрейфус С. Прикладные задачи динамического программирования. - М.: Наука, 1965. - 457 с.
3. Алексеев О.Г. Комплексное применение методов дискретной оптимизации. - М.: Наука, 1987. – 247 с.
4. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. - М.: Мир, 1982. - 416 с.

1. Рубинштейн М.И. Об алгоритмах решения задачи о назначении // Автоматика и телемеханика.1981. N 7 с.145 – 154.
2. Гейл Д. Теория линейных экономических моделей. - М.: ИЛ, 1963. - 418 с.

СОДЕРЖАНИЕ