УДК 519.8

Ковалев М.Я., Котов В.М, Лепин В.В. **Теория алгоритмов. Часть 2. Приближенные алгоритмы.** – Минск: БГУ, 2002. – с. – ISBN - - -.

Книга рассматриваются вопросы разработки и анализа приближенных алгоритмов для задач дискретной оптимизации. Рассмотрены методы построения алгоритмов локального поиска и метаэвристик. Проведен анализ вычислительной сложности задач, рассмотрены наиболее распространенные и эффективные методы решения.

Курс лекций предназначен для студентов, аспирантов и специалистов в области исследования операций, автоматизации управления и календарного планирования производства.

Табл. . Ил. . Библ. назв.

Рецензенты:

доктор физико-математических наук, профессор Ковалев М.М.

кандидат физико-математических наук, профессор Павловский А.И.

ISBN

Оглавление

ВВЕДЕНИЕ

Данный курс лекций является второй частью курса лекций по теории алгоритмов.

В курсе лекций рассматриваются вопросы построения эвристик и приближенных алгоритмов для решения задач дискретной оптимизации. Основное внимание уделено математическим методам решения задач и методам анализа их вычислительной сложности.

Курс лекций разбит на разделы. В первом разделе приводятся наиболее общие постановки задач, вводится схема обозначений и описываются основы наиболее распространенных методов решения и подходов к классификации вычислительной сложности задач.

Курс лекций предназначен для для студентов, аспирантов и специалистов в области теории алгоритмов, исследования операций и автоматизации управления.

Оптимизационные проблемы

Большинство оптимизационных задач являются формализованными математическими моделями задач, возникающих в различных прикладных областях. Конкретную, т.е когда заданы все входные параметры, оптимизационную задачу будем называть *индивидуальной*. Формально под *оптимизационной проблемой* будем понимать четвёрку $\mathcal{P} = (I_{\mathcal{P}}, \mathrm{SOL}_{\mathcal{P}}, m_{\mathcal{P}}, \mathrm{goal}_{\mathcal{P}})$, где:

1) $I_{\mathcal{P}}$ — множество индивидуальных задач для $\mathcal{P}$;

2) $\mathrm{SOL}_{\mathcal{P}}$ — отображение, определенное на множестве $I_{\mathcal{P}}$ и такое, что для каждой индивидуальной задачи $x \in I_{\mathcal{P}}$ множество образов $\mathrm{SOL}_{\mathcal{P}}(x)$ является множеством всех решений задачи x;

3) $m_{\mathcal{P}} : \{(x, s) : x \in I_{\mathcal{P}} \wedge s \in \mathrm{SOL}_{\mathcal{P}}(x)\} \to \mathbb{N}$ — функция меры, называемая также целевой функцией. Для каждой пары

(x, s) = (индивидуальная задача, ее решение) значение $m_{\mathcal{P}}(x, s)$ является натуральным числом, которое называется стоимостью решения s;

4) $\text{goal} \in \{\max, \min\}$ специфицирует, что $\mathcal{P}$ является максимизационной или минимизационной проблемой.

Таким образом, любую оптимизационную задачу можно сформулировать так: для заданной индивидуальной задачи x, найти такое решение s^* из множества $\text{SOL}_{\mathcal{P}}(x)$, что значение целевой функции $m_{\mathcal{P}}(x, s^*)$ является *оптимальным* (максимальным или минимальным в зависимости от того, что $\text{goal}_{\mathcal{P}} = \max$, либо $\text{goal}_{\mathcal{P}} = \min,$) среди всех решений из $\text{SOL}_{\mathcal{P}}(x)$.

Для заданной индивидуальной задачи x, через $\text{SOL}^*_{\mathcal{P}}(x)$ будем обозначать *множество оптимальных решений* задачи x. Более формально, для каждого решения $s^*(x)$, такого что $s^*(x) \in \text{SOL}^*_{\mathcal{P}}(x)$ выполняется

$$m_{\mathcal{P}}(x, s^*(x)) = \text{goal}_{\mathcal{P}}\{\, k \mid k = m_{\mathcal{P}}(x, z) \wedge z \in \text{SOL}_{\mathcal{P}}(x)\}.$$

Стоимость любого оптимального решения $s^*(x)$ будет обозначаться через $m^*_{\mathcal{P}}(x)$.

NP-Оптимизационные проблемы

Нам понадобятся некоторые понятия теории вычислительной сложности. Оптимизационную проблему $\mathcal{P}$ будем называть *NP-оптимизационной проблемой* если выполняются условия:

1) Множество решений $\text{SOL}_{\mathcal{P}}(x)$ состоит только из *допустимых* решений для индивидуальной задачи $x \in I_{\mathcal{P}}$. Термин *допустимый* означает, что размер элементов $s \in \text{SOL}_{\mathcal{P}}(x)$ полиномиально ограничен размером задачи x и множество $\{(x, s) : s \in \text{SOL}_{\mathcal{P}}(x)\}$ распознаваемо за полиномиальное время;

2) Целевая функция $m_{\mathcal{P}} : \{(x, s) : x \in I_{\mathcal{P}} \wedge s \in \text{SOL}_{\mathcal{P}}(x)\} \to \mathbb{N}$ — полиномиально-вычислима.

Напомним, что каждой NP-оптимизационной проблеме P соответствует проблема распознавания из класса NP, которая называется *распознавательным аналогом* NP-оптимизационной проблемы P.

Глава 1

Основы теории полиномиальной сводимости дискретных задач

Теория полиномиальной сводимости устанавливает, что ряд задач распознавания и экстремальных задач принадлежит классам так называемых NP-полных и NP-трудных задач соответственно. Существование либо несуществование полиномиального алгоритма решения хотя бы одной из NP-полных задач влечет соответственно существование (либо несуществование) такого алгоритма для всех остальных NP-полных задач. Существование эффективного алгоритма решения хотя бы одной из NP-трудных задач влечет существование такого рода алгоритма для любой из NP-полных задач. Из этого следует, что для NP-полных и NP-трудных задач существование полиномиальных алгоритмов решения маловероятно.

Задачи распознавания. Под *задачей распознавания (свойств объектов)* будем понимать некоторый общий вопрос, предполагающий ответ "да", если рассматриваемый в этом вопросе объект обладает свойствами, описанными в формулировке вопроса. Объект и его свойства описываются с помощью некоторых параметров, конкретные значения которых в общей формулировке вопроса отсутствуют. Параметрами могут быть множества, графы, числа, функции и т.п.

Пример задачи получается при замене всех имеющихся в общей формулировке задачи параметров их конкретными значениями. Пример называется *да-примером*, если при соответствующих значениях параметров объект обладает указанными в формулировке задачи свойствами.

Каждой задаче распознавания Π можно сопоставить пару мно-

жеств. Первое из них – это множество D_Π всех примеров задачи П, а второе – это подмножество $Y_\Pi \subseteq D_\Pi$, состоящее из всех да-примеров.

Таким образом, задача распознавания П рассматривается как пара множеств D_Π и Y_Π: $\Pi = (D_\Pi, Y_\Pi)$. При таком представлении задача распознавания распадается на две части. Первая часть содержит *общую формулировку* задачи в терминах некоторых параметров, которыми могут быть множества, графы, числа, функции и другие математические объекты. Во второй части в терминах тех же параметров формулируется да-нет *вопрос*. Пример принадлежит множеству D_Π тогда и только тогда, когда он может быть получен из общей формулировки задачи подстановкой вместо параметров их конкретных значений соответствующих типов. Пример принадлежит множеству Y_Π тогда и только тогда, когда ответом на вопрос задачи при данных конкретных значениях параметров является "да".

Задачи распознавания свойств объектов тесно связаны с задачами распознавания языков. Можно закодировать все возможные примеры задачи распознавания, используя какую-нибудь схему кодирования. В результате получим множество слов, каждое из которых представляет некоторый пример. Затем можно рассматривать исходную задачу как задачу распознавания языка, состоящего из всех тех слов, которым соответствуют да-примеры. Отметим, что при переходе от задачи распознавания к языку кодируется не все описание примера, а лишь набор значений параметров.

Задача распознавания принадлежит классу $\mathcal{P}$ (либо $\mathcal{NP}$) при данной схеме кодирования, если соответствующий этой задаче язык принадлежит $\mathcal{P}$ (либо $\mathcal{NP}$ соответственно).

Напомним, что язык принадлежит классу $\mathcal{P}$, если для него существует распознающая *детерминированная машина Тьюринга (ДМТ)*, имеющая полиномиальную временную сложность (иными словами, существует полиномиальный алгоритм решения). Множество всех языков, для каждого из которых существует распознающая *недетерминированная машина Тьюринга (НМТ)* с полиномиальной временной сложностью, называется *классом* $\mathcal{NP}$.

Поскольку детерминированная машина представляет собой специальный случай недетерминированной, очевидно, что $\mathcal{P} \subseteq \mathcal{NP}$. Однако вопрос, является ли $\mathcal{P}$ собственным подмножеством мно-

жества $\mathcal{NP}$ или же $\mathcal{P} = \mathcal{NP}$, остается открытым. Общепринятой является гипотеза $\mathcal{P} \neq \mathcal{NP}$.

Разумные схемы кодирования. Важным аспектом является то, что задачу распознавания с соответствующим языком связывает некоторая схема кодирования. Схема кодирования сопоставляет цепочку символов (слово) каждому примеру задачи. В зависимости от выбранной схемы кодирования одной и той же задаче сопоставляется тот или иной язык, причем эти языки, вообще говоря, не эквивалентны между собой. Возможно даже, что один из них является NP-полным, а другой принадлежит классу $\mathcal{P}$.

При выборе схемы кодирования необходимо удостовериться, что используемые схемы порождают эквивалентные языки для эквивалентных с нашей точки зрения задач. Мы заинтересованы в получении оценок сложности алгоритмов решения задач распознавания, а оценка – это функция длины записи входной информации задачи, т.е. оценка непосредственно зависит от выбранной схемы кодирования. Можно подобрать схему кодирования, которая дает очень длинные коды примеров задачи, так что почти любой алгоритм будет иметь полиномиальную временную сложность относительно таких входов. Поэтому необходимо иметь такие условия, что различные схемы кодирования, им удовлетворяющие, порождают слова (коды), чьи длины при разных схемах кодирования "полиномиально связаны"друг с другом и не превосходят некоторого полинома от слова минимальной длины. Схемы, удовлетворяющие таким условиям, называются *разумными схемами кодирования.*

Прежде чем переходить к описанию условий "разумности"схемы кодирования, введем понятие количества элементарных параметров примера I задачи П и определение функции $Max(I)$.

Задачи описываются через некоторые параметры такие, как множества, графы, функции, числа и т.п. Соответственно *количество элементарных параметров* – это мощность множества, число вершин и число дуг (и/или ребер) графа, число переменных функции (учитывая количество вхождений переменных в описание функции), количество чисел в описании задачи, число требований и число приборов в задачах теории расписаний, число переменных и число уравнений и неравенств в задачах математического программирования и т.п. Не все перечисленные объекты являются элементарными параметрами. Например, требования в задачах теории

расписаний сами описываются через некоторые элементарные параметры такие, как длительность обслуживания, директивный срок, момент готовности требования к обслуживанию. Тем не менее, говоря о количестве элементарных параметров, можно ограничиться числом требований, поскольку число реальных элементарных параметров отличается от него лишь на мультипликативную константу.

Как правило, в каждой задаче имеется несколько элементарных параметров. Будем полагать, что порядок перечисления элементарных параметров в каждой конкретной задаче фиксирован. Тогда каждому примеру задачи сопоставляется вектор с целыми положительными компонентами. Для примера I обозначим через $\nu(I) = (\nu_1(I), \nu_2(I), \ldots, \nu_r(I))$ вектор, k-й компонентой которого является количество k-го элементарного параметра. Для примеров I_1 и I_2 одной и той же задачи будем писать $\nu(I_1) \geq \nu(I_2)$, если $\nu_1(I_1) \geq \nu_1(I_2), \nu_2(I_1) \geq \nu_2(I_2), \ldots, \nu_r(I_1) \geq \nu_r(I_2)$. Если φ – действительная функция, определенная на множестве действительных чисел, то положим $\varphi(\nu(I)) = (\varphi(\nu_1(I)), \varphi(\nu_2(I)), \ldots, \varphi(\nu_r(I)))$.

В общей формулировке задачи будем различать числовые параметры, с одной стороны, и множества, графы, последовательности, функции, поименованные элементы и т.п., с другой стороны. Все числа предполагаются целыми, поскольку мы будем рассматривать задачи, в формулировках которых присутствуют только рациональные числа, а рациональные числа могут быть представлены как комбинации целых чисел. Числовыми параметрами в формулировках задач являются веса вершин или ребер (дуг) графа, длительности обслуживания требований, директивные сроки, моменты готовности требований к обслуживанию, величины элементов в задаче о рюкзаке и задаче о 3-РАЗБИЕНИИ и т.п. Отметим, что в формулировках многих задач числовые параметры отсутствуют, поскольку, например, количество вершин либо ребер графа либо количество требований не рассматривается как числовой параметр. Далее, если это не приводит к двусмысленности, наряду с термином "числовой параметр"будем использовать в том же смысле термин "число".

Введем для задачи распознавания П функцию $Max\colon D_{\Pi} \to Z^{+}$, сопоставляющую каждому примеру $I \in D_{\Pi}$ целое положительное число $Max[I]$, которое соответствует величине наибольшего (по абсолютной величине) числового параметра в I. Если П

не содержит чисел, то полагаем $Max[I] = 0$ для любого примера $I \in D_{\Pi}$.

Будем говорить, что схема кодирования является разумной, если она удовлетворяет следующим условиям.

(а) *Код примера I задачи должен быть компактным и не должен содержать необязательную информацию или символы.* Ниже дается более полное и формализованное определение понятия компактности.

(б) *Числа из примера I должны быть представлены в двоичном либо каком-нибудь ином базисе, отличном от унарного.*

(в) *Кодирующая схема должна обладать свойством декодируемости.* Смысл декодируемости заключается в следующем. Для каждого параметра задачи должен существовать полиномиальный алгоритм, способный выделить код этого параметра из кода любого примера задачи.

(г) *Схема кодирования должна обеспечивать однородность при кодировании различных примеров одной и той же задачи* Π. Однородность означает, что схема кодирования должна обеспечивать для любых двух различных примеров задачи получение кодов, "близких"по длине, если эти примеры содержат близкие по величине числа и количество элементарных параметров в обеих задачах приблизительно одно и то же. Формальное определение однородности схемы кодирования состоит в следующем.

Схема кодирования e задачи распознавания Π обеспечивает однородность кодов, если существуют три таких полинома p_M, p_ν и p_l, что для любых двух примеров I_1 и I_2 из D_{Π} из справедливости неравенств

$$p_M(\max\{Max(I_1), 1\}) \geq Max(I_2) \quad \text{и} \quad p_\nu(\nu(I_1)) \geq \nu(I_2)$$

следует

$$p_l(|e(I_1)|) \geq |e(I_2)|,$$

где $|e(I)|$ – длина кода $e(I)$ примера I при схеме кодирования e.

Условие однородности имеет особое значение. Рассмотрим, например, следующую достаточно простую задачу отыскания расписания обслуживания n требований двумя идентичными параллельными приборами, доставляющего минимум сумме моментов завершения обслуживания требований $(P2//\sum C_j)$. Эта задача разрешима за время $O(n \log n)$. Алгоритм работает следующим об-

разом. Требования последовательно назначаются на обслуживание приборами. Требование с наименьшей длительностью обслуживания среди еще не назначенных на приборы требований назначается на обслуживание прибором с наиболее ранним моментом готовности к обслуживанию.

Рассмотрим некоторую схему кодирования e, которая сопровождает код значения каждого параметра задачи количеством одинаковых значений этого параметра. Такая схема удобна для представления задач, содержащих большое число идентичных параметров, для задачи $P2/p_j = p/\sum C_j$, например, где длительности обслуживания всех требований одинаковы и равны p. Для этой задачи схема e сопроводит значение p параметра "длительность обслуживания требования"числом n, показывая тем самым, что все длительности принимают одно и то же значение.

Предположим, что схема кодирования e способна распознавать, выполнено ли условие $p_j = p_k$ и, что эта способность используется в процессе кодирования. Встретив несколько требований с одинаковыми длительностями обслуживания, схема e построит код одного из них и укажет количество таких идентичных требований. Существуют примеры задачи $P2//\sum C_j$, в которых $p_1 = p_2 = \ldots = p_n$. При схеме кодирования e длина записи входа такого примера равна $O(\log n + \log p_1)$, и величина $O(n \log n)$ не может рассматриваться как полином относительно такой длины, т.е. приведенный выше алгоритм нельзя рассматривать как полиномиальный. Чтобы избежать подобных ситуаций, множество разумных схем кодирования ограничивается теми схемами, которые удовлетворяют условию (г). Нетрудно заметить, что рассмотренная схема e этому условию не удовлетворяет.

Выполнение условия однородности может быть обеспечено соблюдением следующего простого правила. Схема кодирования не должна заниматься распознаванием специфических свойств кодируемого примера задачи. При построении кода конкретного примера задачи схема может использовать лишь те свойства, которые непосредственно указаны в формулировке задачи и являются общими для всех примеров данной задачи. Таким образом, *код примера задачи должен содержать лишь ту информацию, которую содержит общая формулировка задачи, дополненная конкретными значениями параметров задачи.*

Условие (г) предотвращает появление слишком коротких кодов для отдельных примеров задачи. С другой стороны, условие (а) не дает возможности появляться слишком длинным кодам.

Удобно считать, что каждой задаче распознавания Π сопоставлена не зависящая от схемы кодирования функция $Length: D_\Pi \to Z^+$, которая "полиномиально связана"с длинами входов примеров задачи, которые получались бы при использовании любой разумной схемы кодирования. Под полиномиальной связанностью подразумевается, что для любой разумной схемы кодирования e задачи Π существует пара полиномов p и p', удовлетворяющих следующим условиям. Если $I \in D_\Pi$ и x – слово, кодирующее пример I при схеме e, то

$$Length[I] \leq p(|x|) \quad \text{и} \quad |x| \leq p'(Length[I]),$$

где $|x|$ обозначает длину слова x. Для задачи Π будем использовать обозначение $Length_\Pi[I]$ для $I \in D_\Pi$.

Предполагается, что существует хотя бы одна схема кодирования, полиномиально связанная с функцией $Length$.

Из определения разумной схемы кодирования и определения функции $Length$ следует, что любые две разумные схемы кодирования должны быть полиномиально связаны друг с другом. Предположим, что имеются две схемы кодирования e и e' одной и той же задачи Π. Пусть $|e(I)| < |e'(I)|$ для любого примера I задачи Π. Здесь $|e(I)|$ – длина кодирующего слова $e(I)$ примера I. Если не существует такого полинома p, что $p(|e(I)|) \geq |e'(I)|$ для любых $I \in D_\Pi$, то схему e' нельзя рассматривать как разумную. Это дает возможность формализовать условие (а) – условие компактности – из определения разумной схемы кодирования. Если схема кодирования e' удовлетворяет условиям (б), (в) и (г), и не существует другой схемы e, удовлетворяющей (б), (в) и (г), которая лучше, чем e' в указанном выше смысле, то e' является разумной схемой кодирования. К сожалению, это условие, более формальное по сравнению с условием (а), вряд ли является легче проверяемым, чем (а). По крайней мере, оно дает возможность сказать, что схема e' не является разумной, если удалось отыскать упомянутую лучшую схему e независимо от того, выглядит ли e' "компактной"или нет.

Далее предполагается, что *для любой рассматриваемой задачи*

известна некоторая разумная схема кодирования и при кодировании задачи используется именно такая схема. Кроме того, будем считать, что *все рассматриваемые полиномы имеют лишь положительные коэффициенты.* В противном случае коэффициенты можно заменить на положительные, не уменьшая значения полиномов.

Полиномиальное сведение задач. NP-полные задачи. Будем говорить, что словарная функция Φ переводит задачу распознавания $\Pi = (D_\Pi, Y_\Pi)$ в задачу распознавания $\Pi' = (D_{\Pi'}, Y_{\Pi'})$, если Φ отображает коды примеров из множества D_Π в коды примеров из $D_{\Pi'}$. Если $I \in D_\Pi$, то для простоты записей будем обозначать через $\Phi(I)$ соответствующий ему пример задачи Π'.

Задача распознавания Π называется *полиномиально сводимой* к задаче распознавания Π', если существует словарная функция Φ, переводящая задачу Π в задачу Π' и удовлетворяющая следующим условиям:

(а) для любого $I \in D_\Pi$ соотношение $I \in Y_\Pi$ выполняется тогда и только тогда, когда выполняется $\Phi(I) \in Y_{\Pi'}$;

(б) временная сложность вычисления функции Φ не превосходит некоторого полинома от $Length_\Pi[I]$.

Задача распознавания Π называется NP-*трудной*, если любая задача из $\mathcal{NP}$ полиномиально сводится к ней. Задача распознавания Π называется NP-*полной*, если она NP-трудна и $\Pi \in \mathcal{NP}$.

Понятие полиномиальной сводимости транзитивно. Поэтому для доказательства NP-трудности некоторой задачи достаточно полиномиально свести к ней некоторую NP-трудную задачу.

Понятие NP-трудности, определенное для задач распознавания, может быть распространено и на экстремальные комбинаторные задачи.

NP-трудные экстремальные задачи и принадлежность классу $\mathcal{NP}$. *Экстремальная комбинаторная задача* состоит в следующем. На конечном множестве X' определен функционал $F(x), \quad x \in X'$. Необходимо в заданном подмножестве X множества X' отыскать такой элемент x^0, что $F(x^0) = \min\{F(x) | x \in X\}$ (задача минимизации), либо такой $\bar{x}^0$, что $F(\bar{x}^0) = \max\{F(x) | x \in X\}$ (задача максимизации).

Прежде чем вводить такие понятия, как NP-трудность экстремальной задачи, временная сложность алгоритма и т.п., определим

понятие *элементарной операции*. Использование машины Тьюринга в качестве формализации интуитивного понятия алгоритма удобно при введении ряда определений и доказательстве утверждений. Тем не менее во многих ситуациях использование этого формализма приводит к чрезмерной и явно излишней громоздкости рассуждений. Поэтому далее мы будем нередко говорить просто об алгоритмах (детерминированных или недетерминированных), имея в виду любую возможную формализацию этого понятия (машины Тьюринга, нормальные алгорифмы Маркова или программы, написанные на каком-либо алгоритмическом языке).

Говоря об элементарной операции, будем иметь в виду понятие, определяемое той или иной формализацией понятия алгоритма. Для машины Тьюринга – это шаг работы машины, для алгоритмов, предназначенных для реализации на ЭВМ, под элементарными понимаются такие машинные операции, как сложение, умножение, сравнение двух чисел, запись либо считывание числа, расположенного по известному адресу, и т.п.

Временная сложность алгоритма – это функция от длины записи входной информации задачи, определяемая аналогично временной сложности машины Тьюринга. Отличия заключаются в следующем. Во-первых, вместо шага работы машины должна фигурировать элементарная операция. Во-вторых, под заключительным состоянием алгоритма понимается ситуация, когда получен ответ "да"(в случае задачи распознавания), либо когда найден элемент $x^0 \in X$, доставляющий экстремум целевой функции, если рассматривается экстремальная комбинаторная задача.

Алгоритм решения задачи Π называется *полиномиальным*, если его временная сложность не превосходит некоторого полинома от функции $Length_{\Pi}[I]$ для любого примера I задачи Π. Соответствующая задача называется *полиномиально разрешимой*.

Сопоставим экстремальной задаче B следующую задачу распознавания B'. Определить: существует ли такой элемент x' в заданном множестве X, что $F(x') \leq y$ (либо $F(x') \geq y$ в случае задачи максимизации) для данного действительного числа y. Очевидно, если x^0 – решение задачи B, то элемент $x' \in X$ такой, что $F(x') \leq y$, существует тогда и только тогда, когда $F(x^0) \leq y$. Таким образом, экстремальная задача оказывается не проще соответствующей задачи распознавания.

Экстремальную задачу B будем называть NP-трудной, если соответствующая ей задача распознавания является NP-трудной.

Очевидно, что *из существования полиномиального алгоритма решения какой-нибудь NP-трудной задачи следует существование полиномиальных алгоритмов решения для всех задач из класса $\mathcal{NP}$, включая все NP-полные задачи.*

Принадлежность конкретной задачи классу NP-трудных задач является одной из важных ее характеристик. Если гипотеза о несовпадении классов $\mathcal{P}$ и $\mathcal{NP}$ верна, то ни для одной из NP-трудных задач не существует полиномиальный алгоритм ее решения. Поэтому из NP-трудности задачи следует необходимость построения приближенных или эвристических алгоритмов ее решения, применения схем направленного перебора вариантов (таких, как метод последовательного конструирования, анализа и отсеивания вариантов).

Из приведенных выше определений следует, что для проверки принадлежности задачи распознавания П классу $\mathcal{NP}$ достаточно построить подходящую полиномиальную НМТ. К сожалению, процесс построения такой машины является чрезвычайно трудоемким. Поэтому для осуществления проверки пользуются другим описанием класса $\mathcal{NP}$.

Пусть Λ^* – множество всех слов в некотором алфавите Λ. Обозначим через $P^{(2)}$ класс всех подмножеств множества $\Lambda^*\times\Lambda^*$, каждое из которых распознаваемо за полиномиальное время подходящей ДМТ. Для конкретного $L^{(2)} \in P^{(2)}$ и полинома p определим язык L следующим образом:

$$L = \{\bar{a} \in \Lambda^* | \quad \bar{b} \in \Lambda^* \ ,$$

$$\langle \bar{a}, \bar{b} \rangle \in L^{(2)} \quad |\bar{b}| \leq p(|\bar{a}|)\}.$$

Обозначим множество всех таких языков для всех возможных полиномов p через $\{L/P^{(2)}\}$.

Теорема 1.1. *Язык принадлежит множеству $\{L/P^{(2)}\}$ тогда и только тогда, когда существует полиномиальная НМТ, распознающая этот язык.*

Непосредственно из теоремы следует, что $\mathcal{NP} = \{L/P^{(2)}\}$. Отсюда получаем сравнительно простой механизм проверки принадлежности конкретной задачи распознавания П классу $\mathcal{NP}$. Напомним, что каждой задаче П сопоставляется язык, состоящий из

кодов всех да-примеров задачи Π. Из теоремы 1.1 следует, что для проверки принадлежности задачи Π классу $\mathcal{NP}$ достаточно показать, что для некоторого полинома p и каждого да-примера I задачи Π существует слово $\bar{b}_I$ в алфавите схемы кодирования задачи Π такое, что $|\bar{b}_I| \leq p(|e(I)|)$ и множество всех пар $\langle e(I), \bar{b}_I \rangle$ полиномиально распознаваемо некоторой ДМТ. Здесь $e(I)$ – это код примера I. Для задачи распознавания Π слово $\bar{b}_I$ можно рассматривать как код решения, сопоставленного примеру I. В каждом конкретном случае слова $\bar{b}_I$ определяются обычно достаточно просто. Например, для задачи распознавания, является ли граф Γ гамильтоновым, $\bar{b}_I$ – это описание некоторой последовательности вершин конкретного графа из примера I задачи. Для задачи теории расписаний $\bar{b}_I$ есть подходящее описание некоторого расписания и т.п. Если в первой задаче для каждого конкретного графа Γ и такой последовательности его вершин, длина описания которой не превосходит некоторого полинома от $Length[I]$ для нашей задачи, мы за полиномиальное время с помощью детерминированного алгоритма можем распознать, определяет эта последовательность гамильтонов цикл в Γ или нет, то задача Π принадлежит классу $\mathcal{NP}$. Аналогично, если в задаче теории расписаний для каждого конкретного примера задачи мы можем для подходящего расписания за полиномиальное время детерминированным алгоритмом определить, действительно ли это расписание удовлетворяет всем условиям из описания задачи, то эта задача принадлежит $\mathcal{NP}$. Здесь "подходящее расписание"означает, что существует такой полином p, что для любого примера I задачи длина описания этого расписания не превосходит $p(Length[I])$.

Отметим еще раз два наиболее существенных аспекта проверки принадлежности задачи распознавания классу $\mathcal{NP}$. Во-первых, для каждого да-примера I должно *существовать* подходящее слово $\bar{b}_I$ и, во-вторых, длина этого слова не должна превосходить некоторого (одного и того же для всех примеров) *полинома* от $Length[I]$. В случае задачи теории расписаний, например, это означает, что указанную выше проверку достаточно уметь осуществлять не для любого расписания, а лишь для некоторого подмножества расписаний, заведомо содержащего хотя бы одно такое расписание σ, что значение целевого функционала $F(\sigma) \leq y$ для заданной величины y, если такие расписания вообще существуют.

При этом длина представления σ не должна превосходить некоторого общего для всех примеров задачи полинома от $Length[I]$. Таким образом, исследование свойств, которыми обладают некоторые оптимальные расписания задачи, важно и с точки зрения определения принадлежности сопоставленной ей задачи распознавания классу $\mathcal{NP}$. Если установлено, что при поиске оптимального расписания можно ограничиться рассмотрением лишь тех расписаний, которые обладают некоторыми специфическими свойствами, то и при проверке принадлежности соответствующей задачи распознавания классу $\mathcal{NP}$ достаточно ограничиться анализом лишь этих расписаний. Кроме того, должна быть найдена такая форма представления расписания, при которой длина его кода будет ограничена полиномом от длины кода примера задачи.

Псевдополиномиальное сведение задач и NP-трудные в сильном смысле задачи. Наряду с разделением задач на NP-трудные и полиномиально разрешимые, NP-трудные задачи, в свою очередь, подразделяются на NP-трудные в сильном смысле задачи и задачи, имеющие псевдополиномиальные алгоритмы решения.

Алгоритм решения задачи Π называется *псевдополиномиальным*, если его временная сложность не превосходит некоторого полинома от двух переменных $Length_{\Pi}[I]$ и $Max_{\Pi}[I]$ для любого примера I задачи Π. Соответствующая задача называется *псевдополиномиально разрешимой*.

Нетрудно заметить, что любой *полиномиальный алгоритм является одновременно и псевдополиномиальным*. Кроме того, ни одна из NP-трудных задач, не имеющая числовых параметров, не может иметь псевдополиномиального алгоритма решения, если $\mathcal{P} \neq \mathcal{NP}$. В противном случае такая задача имела бы полиномиальный алгоритм решения, поскольку $Max[I] = 0$ для любого ее примера. Аналогично, если для задачи Π существует такой полином p, что $p(Length_{\Pi}[I]) \geq Max_{\Pi}[I]$ для любого $I \in D_{\Pi}$, то Π, будучи NP-трудной, не может быть псевдополиномиально разрешимой.

Таким образом, существуют NP-трудные задачи, которые не могут иметь даже псевдополиномиальных алгоритмов решения. Такие задачи образуют множество так называемых NP-трудных в сильном смысле задач. Для формализации понятия NP-трудной в сильном смысле задачи введем определение псевдополиномиальной сво-

димости задач распознавания.

Пусть пары функций $\langle Length_{\Pi}, Max_{\Pi}\rangle$ и $\langle Length_{\Pi'}, Max_{\Pi'}\rangle$ сопоставлены задачам Π и Π' соответственно.

Говорят, что задача распознавания Π *псевдополиномиально сводится* к задаче распознавания Π', если существует словарная функция Φ, переводящая задачу Π в задачу Π' и удовлетворяющая следующим условиям:

(а) для любого примера $I \in D_{\Pi}$ соотношение $I \in Y_{\Pi}$ выполняется тогда и только тогда, когда выполняется $\Phi(I) \in Y_{\Pi'}$;

(б) временная сложность вычисления функции Φ не превосходит некоторого полинома q от двух переменных $Length_{\Pi}[I]$ и $Max_{\Pi}[I]$;

(в) существуют такие полиномы q_1 и q_2, что для любого $I \in D_{\Pi}$

$$q_1(Length_{\Pi'}[\Phi(I)]) \geq Length_{\Pi}[I] \tag{1.1}$$

и

$$q_2(Max_{\Pi}[I], Length_{\Pi}[I]) \geq Max_{\Pi'}[\Phi(I)]. \tag{1.2}$$

Задача распознавания Π называется *NP-трудной в сильном смысле*, если существует NP-трудная задача распознавания $\widehat{\Pi}$, которая псевдополиномиально сводится к Π и

$$Max_{\widehat{\Pi}}[I] \leq \widehat{p}(Length_{\widehat{\Pi}}[I]) \tag{1.3}$$

для любого $I \in D_{\widehat{\Pi}}$ и некоторого полинома $\widehat{p}$.

Поскольку любая задача псевдополиномиально сводится к себе, *то любая NP-трудная задача, удовлетворяющая условию* (1.3), *является NP-трудной в сильном смысле.* Кроме того, любая NP-трудная в сильном смысле задача является NP-трудной, и ни одна из NP-трудных в сильном смысле задач не может иметь псевдополиномиального алгоритма решения, если $\mathcal{P} \neq \mathcal{NP}$.

Понятие псевдполиномиальной сводимости транзитивно. Поэтому для доказательства NP-трудности в сильном смысле некоторой задачи достаточно псевдополиномиально свести к ней некоторую NP-трудную задачу.

Задача распознавания Π называется *NP-полной в сильном смысле*, если $\Pi \in \mathcal{NP}$ и Π является NP-трудной в сильном смысле.

Экстремальная комбинаторная задача называется NP-трудной в сильном смысле, если сопоставленная ей задача распознавания NP-трудна в сильном смысле.

Примеры NP-трудных задач

Ниже приведены формулировки NP-трудных задач, которые в дальнейшем используются при доказательстве NP-трудности задач оптимального планирования.

РАЗБИЕНИЕ:

Заданы целые положительные числа $a_1, a_2, \ldots, a_r$ и A такие, что $\sum_{j=1}^{r} a_j = 2A$. Требуется определить, существует ли множество $X \subset N = \{1, 2, \ldots, r\}$ такое, что $\sum_{j \in X} a_j = A$.

РАЗБИЕНИЕ ОДИНАКОВОЙ МОЩНОСТИ:

Заданы целые положительные числа $a_1, a_2, \ldots, a_r$ и A такие, что r – четное, и $\sum_{j=1}^{r} a_j = 2A$. Требуется определить, существует ли множество $X \subset N = \{1, 2, \ldots, r\}$ такое, что его мощность $|X| = r/2$ и $\sum_{j \in X} a_j = A$.

3-РАЗБИЕНИЕ:

Заданы целые положительные числа $a_1, a_2, \ldots, a_{3r}$ и A такие, что $A/4 < a_j < A/2$, $j = 1, \ldots, 3r$, и $\sum_{j=1}^{3r} a_j = rA$. Требуется определить, существует ли разбиение множества $\{1, 2, \ldots, 3r\}$ на r непересекающихся подмножеств $X_1, X_2, \ldots, X_r$ такое, что $\sum_{j \in X_l} a_j = A$, $l = 1, \ldots, r$.

Нетрудно заметить, если решение задачи 3-РАЗБИЕНИЕ существует, то каждое из множеств $X_1, \ldots, X_r$ содержит в точности три элемента.

Задачи РАЗБИЕНИЕ и РАЗБИЕНИЕ ОДИНАКОВОЙ МОЩНОСТИ являются NP-трудными, а задача 3-РАЗБИЕНИЕ является NP-трудной в сильном смысле.

Глава 2

Локальный поиск

Для любой задачи алгоритм локального поиска начинает свою работу от некоторого исходного решения (найденного каким то другим алгоритмом или выбранного случайно) и представляет собой итеративный процесс. На каждом шаге локального спуска происходит переход от текущего решения к соседнему решению с меньшем значением целевой функции до тех пор, пока не будет достигнут локальный оптимум. Соседнее решение не обязательно должно быть наилучшим в окрестности, а вот критерий оценки решения не должен меняться по ходу этого итеративного процесса.

Таким образом, для любого решения s должно быть задано некоторое множество $\mathcal{N}(s)$ *соседних* решений, которое называется окрестностью s. Грубо говоря, окрестность $\mathcal{N}(s)$ состоит из решений незначительно отличающихся от s. Семейство всех окрестностей называется структурой окрестностей. Дадим формальные определения.

Структурой окрестностей для индивидуальной задачи x оптимизационной проблемы $\mathcal{P}$ называется отображение $\mathcal{N}$, которое каждому допустимому решению s задачи ставит в соответствие множество решений $\mathcal{N}(s)$. Структура окрестностей может быть достаточно сложной и отношение соседства не всегда симметрично, то есть s может быть соседом s', но s' может не быть соседом s.

Определение 2.1. *Решение $s \in SOL_{\mathcal{P}}(x)$ называется локально-минимальным (локально-максимальным) по отношению к структуре окрестностей $\mathcal{N}$, если*

$$m_{\mathcal{P}}(s) \leq m_{\mathcal{P}}(s') \ \ (m_{\mathcal{P}}(s) \geq m_{\mathcal{P}}(s')) \text{ для всех } s' \in \mathcal{N}(s).$$

Таким образом, говорят, что решение s является *локально-оптимальным*, если не существует лучшего решения в $\mathcal{N}(s)$.

Множество локально-оптимальных решений обозначим через $\mathrm{SOL}^{\mathcal{N}}(x)$. Это множество, очевидно, зависит от выбора структуры окрестностей $\mathcal{N}$.

Определение 2.2. *Структура окрестностей $\mathcal{N}$ называется точной, если $SOL^{\mathcal{N}}(x) \subseteq SOL^{*}(x)$.*

Для данного решения s поиск более лучшего решения в окрестности $\mathcal{N}(s)$ может быть осуществлен двумя способами: (i) *выбор наилучшего* решения во всей окрестности; (ii) *выбор первого лучшего*, когда окрестность каким либо образом перебирается и этот перебор сразу же прекращается, как только будет найдено более лучшее решение.

Локальным поиском для оптимизационной проблемы $\mathcal{P}$ называется алгоритм, который для любой индивидуальной задачи $x \in I_{\mathcal{P}}$ просматривает подмножество $SOL_{\mathcal{P}(x)}$, до момента когда будет найдено локально-оптимальное решение относительно структуры окрестностей $\mathcal{N}$.

Пусть $\mathcal{P}$ дискретная оптимизационная проблема, тогда структуру окрестностей можно задать с помощью ориентированного графа.

Определение 2.3. *Графом соседства (окрестностей) для индивидуальной задачи $x \in I_{\mathcal{P}}$ называется взвешенный ориентированный граф $G_{\mathcal{N}}(x) = (V(x), A)$ с множеством вершин $V(x) = SOL_{\mathcal{P}(x)}$ и множеством дуг $A = \{(s_1, s_2) \mid s_2 \in \mathcal{N}(s_1)\}$. Веса в этом графе приписаны вершинам и равны соответствующим значениям целевой функции.*

Следовательно окрестность решения s совпадает с замкнутым окружением соответствующей вершины, т.е. $\mathcal{N}(s) = \{s\} \cup \{s' \in V(x) \mid (s, s') \in A$.

Граф окрестностей $G_{\mathcal{N}}(x)$ (neighborhood graph) иногда называют еще ландшафтом целевой функции или просто ландшафтом (landscape, fitness landscape). При определении структуры окрестностей $\mathcal{N}$ важно следить за тем, чтобы получающийся граф $G_{\mathcal{N}}(x)$ был строго связен, т. е. для каждой пары вершин s и s' существовал путь из s в s'. Это свойство является важным при анализе асимптотического поведения алгоритмов, например, вероятност-

ных метаэвристик, о которых пойдет речь в разделе 3. Если же это свойство не выполняется, то стараются получить хотя бы свойство *вполне связности*, когда из любой вершины существует путь в вершину $s^* \in \mathrm{SOL}^*(x)$ с минимальным значением целевой функции. Если же и этого свойства нет, то мы теряем уверенность в достижении глобального оптимума локальными методами и должны либо ограничиться локальными оптимумами, либо переопределить функцию окрестности.

Переходу от одного текущего решения к другому в графе соседства соответствует дуга. Поэтому локальному поиску для задачи x соответствует путь в графе соседства, который заканчивается в локально-оптимальной вершине $s_r \in V(x)$ со свойством $m_{\mathcal{P}}(x, s_r) \leq m_{\mathcal{P}}(x, s)$ для любого s такого, что $(s_r, s) \in A$.

В худшем случае локальный поиск может исследовать экспоненциальное число решений прежде чем достигнет локального оптимума. По этой причине, часто в эвристике локального поиска используется *правило остановки*.

Все алгоритмы локального поиска основаны на следующей общей схеме.

Схема Локального поиска

Input: Индивидуальная задача x;
Output: Решение s;
begin
  $s :=$ исходное допустимое решение s_0;
  (* $\mathcal{N}$ — структура окрестностей *)
  repeat
    Выбрать еще не рассмотренное решение $s' \in \mathcal{N}(s)$;
    if $m(x, s') < m(x, s)$ **then**
      $s := s'$;
  until все решения из $\mathcal{N}$ не будут рассмотрены;
  return s
end.

Отметим, что для данной индивидуальной задачи поведение этого алгоритма зависит от следующих факторов:

1. Структуры окрестностей $\mathcal{N}$. Размер окрестности любого решения должен выбираться на основе компромисса между целью получения хорошего улучшения при каждом переходе к новому текущему решению и целью ограничения времени про-

смотра одной окрестности. Обычно, для любого решения s, окрестность $\mathcal{N}(s)$ порождается с помощью некоторой операции локального изменения s.

2. Начального решения s_0. Его можно находить с помощью алгоритма (например, конструктивной эвристикой), который выдает хорошее допустимое решение, или с помощью процедуры случайной генерации.

3. Стратегии выбора новых решений. Например, все решения из $\mathcal{N}(s)$ просматриваются, выбирается лучшее из них и сравнивается с s. Это означает, что если s не локально-оптимально, то осуществляется переход к наилучшему соседу. Или, наоборот, осуществляется переход к первому лучшему решению, найденному в окрестности.

Алгоритмы локального поиска широко применяются для решения NP-трудных задач дискретной оптимизации. Вместе с тем, многие полиномиально разрешимые задачи могут рассматриваться как задачи, легко решаемые локальным спуском. При подходящем выборе полиномиальной окрестности соответствующая теорема может быть сформулирована в следующем виде: допустимое решение не является глобальным оптимумом, если и только если оно может быть улучшено некоторым локальным образом. Ниже приводится несколько примеров таких задач. Данные примеры указывают на важность локального поиска при построении оптимизационных алгоритмов и достаточно общий характер этого подхода.

1. Линейное программирование. Геометрически алгоритм симплекс метода можно интерпретировать как движение по вершинам многогранника допустимой области. Вершина не является оптимальной, если и только если существует смежная с ней вершина с меньшим значением целевой функции. Алгебраически, предполагая невырожденность задачи, базисное допустимое решение не является оптимальным, если и только если оно может быть улучшено локальным изменением базиса, т. е. заменой одной базисной переменной на небазисную. Получающаяся таким образом окрестность является точной и имеет полиномиальную мощность.

2. Минимальное остовное дерево. Остовное дерево не является оптимальным, если и только если локальной перестройкой, добавляя одно ребро и удаляя из образовавшегося цикла другое ребро,

можно получить новое остовное дерево с меньшим суммарным весом. Операция локальной перестройки задает отношение соседства на множестве остовных деревьев. Окрестность любого дерева имеет полиномиальную мощность, а функция окрестности является точной.

3. Максимальное паросочетание. Паросочетание не является максимальным, если и только если существует увеличивающий путь. Два паросочетания называют соседними, если их симметрическая разность образует путь. Определенная таким образом окрестность является точной и имеет полиномиальную мощность. Аналогичные утверждения справедливы для взвешенных паросочетаний, совершенных паросочетаний минимального веса, задач о максимальном потоке и потоке минимальной стоимости.

На каждом шаге локального спуска структура окрестностей $\mathcal{N}$ задает множество возможных направлений движения. Очень часто это множество состоит из нескольких элементов и имеется определенная свобода в выборе следующего решения. Правило выбора может оказать существенное влияние на трудоемкость алгоритма и результат его работы. Например, в задаче о максимальном потоке алгоритм Форда-Фалкерсона (который тоже можно рассматривать как вариант локального спуска) имеет полиномиальную временную сложность при выборе кратчайшего пути для увеличения потока и экспоненциальную временную сложность без гарантии получить глобальный оптимум при произвольном выборе пути. Таким образом, при разработке алгоритмов локального поиска важно не только правильно определить окрестность, но и верно задать правило выбора направления спуска. Интуитивно кажется, что в окрестности надо брать элемент с наименьшим значением целевой функции. Однако, как мы увидим ниже, разумным оказывается не только такой выбор, но и движение в "абсурдном"направлении, когда несколько шагов с ухудшением могут привести (и часто приводят) к лучшему локальному оптимуму.

При выборе окрестности хочется иметь множество $\mathcal{N}(s)$ как можно меньшей мощности, чтобы сократить трудоемкость одного шага. С другой стороны, более широкая окрестность, вообще говоря, приводит к лучшему локальному оптимуму. Поэтому при создании алгоритмов каждый раз приходится искать оптимальный баланс между этими противоречивыми факторами. Ясных принципов

разрешения этого противоречия на сегодняшний день не известно, и для каждой задачи этот вопрос решается индивидуально.

2.1. Локальный поиск для задачи о максимальном разрезе

В этой разделе мы рассмотрим задачу о максимальном разрезе и построим алгоритм локального поиска, который находит решение этой задачи, отличающееся от оптимального не более чем в два раза. Проблема формулируется так.

Максимальный Разрез

Индивидуальная задача: Граф $G = (V, E)$.

Решение: Разбиение V на два множества A и B.

Критерий: Мощность разреза, т.е., число ребер, имеющих одну концевую вершину в A, а другую в B.

Прежде всего мы должны определить процедуру получения исходного допустимого решения и описать структуру окрестностей. Простой выбор в качестве исходного допустимого решения пары (A, B), где $A = \emptyset$ и $B = V$ решает первую задачу тривиально.

Структуру окрестностей $\mathcal{N}$ для этой задачи определим так: окрестность решения (A, B) состоит из всех таких разбиений (A_i, B_i), $i = 1, \ldots, |V|$, что

1. если вершина $v_i \in A$, то $A_i = A - \{v_i\}$ и $B_i = B \cup \{v_i\}$;
2. если вершина $v_i \notin A$, то $A_i = A \cup \{v_i\}$ и $B_i = B - \{v_i\}$.

Приведенная структура окрестностей имеет следующее важное свойство: стоимость каждого локально оптимального решения больше половины стоимости оптимального решения.

Теорема 2.1. *Пусть $x =' G = (V, E)'$ индивидуальная задача проблемы Максимальный Разрез и пусть (A, B) локально-оптимальное решение этой задачи относительно структуры окрестностей $\mathcal{N}$, имеющее стоимость $m_N(G)$. Тогда*

$$m^*(G)/m_N(G) \leq 2.$$

Доказательство. Пусть $q = |E|$ число ребер графа G. Поскольку $m^*(G) \leq q$, то достаточно показать, что $m_N(G) \geq q/2$.

Пусть q_A — число всех ребер графа, имеющих концевые вершины в множестве A, и q_B — число всех ребер графа, имеющих концевые вершины в множестве B. Тогда

$$q = q_A + q_B + m_N(G). \tag{2.1}$$

Для любой вершины $v \in V$ множество смежных с ней вершин можно разбить на два множества:

$$U_A(v) = \{u \mid u \in A \wedge (v,u) \in E\}$$

и

$$U_B(v) = \{u \mid u \in B \wedge (v,u) \in E\}.$$

Если (A,B) — локальный оптимум, то любое разбиение (A_i, B_i) из его окрестности имеет стоимость не большую чем $m_N(G)$. Поэтому для каждой вершины $v \in A$ верно неравенство

$$|U_A(v)| - |U_B(v)| \leq 0$$

и, для каждой вершины $w \in B$,

$$|U_B(w)| - |U_A(w)| \leq 0.$$

Просуммировав такие неравенства соответственно по вершинам из A и B, получим, что

$$\sum_{v \in A} (|U_A(v)| - |U_B(v)|) = 2q_A - m_N(G) \leq 0$$

и

$$\sum_{w \in A} (|U_B(w)| - |U_A(w)|) = 2q_B - m_N(G) \leq 0.$$

Следовательно $q_A + q_B - m_N(G) \leq 0$. Из этого неравенства, учитывая (2.1) получаем, что $m_N(G) \geq q/2$ и следовательно теорема доказана. □

Для рассмотренной проблемы, также как и для любой другой, можно по разному определить структуру окрестностей. Основные факторы на которые следует обращать внимание при выборе структуры окрестностей следующие:

- качество получаемого решения (т.е., как близка стоимость локального оптимума к стоимости глобального оптимума);
- порядок в каком будет исследоваться окрестность;
- сложность проверки условия, что окрестность не содержит лучших решений;

- число решений генерируемых до момента когда будет найдено локально-оптимальное решение.

Учитывать эти факторы следует на основе компромисса. Например, если структура окрестностей является "большой", то вероятно, что стоимость найденного решения близка к оптимальной стоимости. Однако, в этом случае задача проверки условия, что более предпочтительного по стоимости решения в окрестности не существует становится сложнее.

2.2. Сложность локального поиска

Анализ вычислительной сложности локального поиска в последние годы интенсивно ведется в двух направлениях: эмпирическом и теоретическом. Как ни странно, но эти направления дают существенно разные оценки возможностям локального поиска.

Эмпирические результаты. Для многих NP-трудных задач локальный по- иск позволяет находить приближенные решения, близкие по целевой функции к глобальному оптимуму. Трудоемкость алгоритмов часто оказывается полиномиальной, причем степень полинома достаточно мала. Так для задачи о разбиении множества вершин графа на две равные части разработаны алгоритмы локального поиска со средней трудоемкостью $O(n \log n)$, n — число вершин, которые дают всего несколько процентов погрешности.

Для задачи коммивояжера алгоритмы локального поиска являются наилучшими с практической точки зрения. Один из таких алгоритмов с окрестностью Лина-Кернигхана в среднем имеет погрешность около 2% и максимальная размерность решаемых задач достигает 1 000 000 городов. На случайно сгенерированных задачах такой колоссальной размерности итерационная процедура Джонсона позволяет находить решения с отклонением около 0,5% за несколько минут на современных компьютерах. Для реальных задач с числом городов до 100 000 существуют алгоритмы с аналогичными характеристиками.

Для задач теории расписаний, размещения, покрытия, раскраски и многих других NP-трудных задач алгоритмы локального поиска показывают превосходные результаты. Более того, их гибкость при изменении математической модели, простота реализации и наглядность превращают локальный поиск в мощное средство для решения практических задач.

Теоретические результаты. Исследование локального поиска с точки зрения гарантированных оценок качества показывают границы его возможностей. Построены трудные для локального поиска примеры, из которых следует, что

1) минимальная точная окрестность может иметь экспоненциальную мощность;

2) число шагов для достижения локального оптимума может оказаться экспоненциальным;

3) значение локального оптимума может сколь угодно сильно отличаться от глобального оптимума.

Эти результаты подталкивают к более пристальному рассмотрению задачи нахождения произвольного локального оптимума, ее трудоемкости в худшем и среднем случае. Абстрактная (массовая) задача локального поиска P задается множеством ее индивидуальных задач, для каждый из которых однозначно определена целевая функция, структура окрестностей и множество допустимых решений. Для того чтобы характеризовать задачи с "хорошими"алгоритмами локального поиска, введен так называемый PLS (Polynomial-time Local Search) класс таких задач.

Пусть $I_{\mathcal{P}}$ — множество индивидуальных задач массовой задачи $\mathcal{P}$ с целевой функцией $m_{\mathcal{P}}$. Для каждой задачи $x \in I_{\mathcal{P}}$ пусть $\mathrm{SOL}_{\mathcal{P}}(x)$ — множество допустимых решений. Будем предполагать, что это множество конечно. Пусть выбрана структура окрестностей $\mathcal{N}$ и $\mathcal{N}(s,x)$ обозначает множество решений из окрестности допустимого решения $s \in \mathrm{SOL}_{\mathcal{P}}(x)$.

Говорят, что массовая задача P принадлежит классу PLS, если для любой индивидуальной задачи x и любого множества допустимых решений $\mathrm{SOL}_{\mathcal{P}}(x)$ существуют следующие три полиномиальных алгоритма:

$A_{\mathcal{P}}$ — по входу x проверяет принадлежность x к $I_{\mathcal{P}}$, и если $x \in I_{\mathcal{P}}$ вычисляет начальное допустимое решение $s_0 \in \mathrm{SOL}_{\mathcal{P}}(x)$.

$B_{\mathcal{P}}$ — по входу $x \in I_{\mathcal{P}}$ и $s \in \mathrm{SOL}_{\mathcal{P}}(x)$ вычисляет соответствующее значение целевой функции $m_{\mathcal{P}}(s,x)$.

$C_{\mathcal{P}}$ — по входу $x \in I_{\mathcal{P}}$ и $s \in \mathrm{SOL}_{\mathcal{P}}(x)$ либо определяет, что s является локальным оптимумом, либо находит решение $s' \in \mathcal{N}(s,x)$ с лучшим значением целевой функции (т.е. такое, что $m_{\mathcal{P}}(s',x) < m_{\mathcal{P}}(s,x)$, если $\mathcal{P}$ — задача минимизации, или такое, что $m_{\mathcal{P}}(s',x) > m_{\mathcal{P}}(s,x)$, если $\mathcal{P}$ — задача максимизации).

Другими словами, в классе PLS содержатся все задачи локального поиска, для которых проверка локальной оптимальности может быть осуществлена за полиномиальное время. По аналогии с классами P и NP для задач локального поиска можно ввести в рассмотрение классы P_s и NP_s. Неформально, класс P_s (NP_s) состоит из задач нахождения локального оптимума, разрешимых за полиномиальное время на детерминированной (недетерминированной) машине Тьюринга. Заметим, что классы P_s и NP_s вводятся без перехода к задачам распознавания, так как, имея алгоритм для нахождения локального оптимума неясно, как отвечать на вопрос: Существует ли локальный оптимум со значением целевой функции, не превосходящим A? Одним из представителей класса P_s является, например, задача линейного программирования, так как метод эллипсоидов имеет полиномиальную трудоемкость. Другим примером может служить задача о максимальной клике, где число локальных улучшений, очевидно, не привышает числа вершин графа. Для классов P_s и NP_s установлены, в частности, следующие свойства:

Теорема 2.2. *$P_s = NP_s$, если и только если $P = NP$.*

Теорема 2.3. *$P_s \subseteq PLS \subseteq NP_s$.*

Теорема 2.4. *Если задача $\mathcal{P}$ из класса PLS является NP-трудной, то NP=co-NP.*

Отметим, что для любой задачи $\mathcal{P}$ из класса PLS можно осуществить так называемый *стандартный локальный поиск:*

Алгоритм Стандартный локальный поиск
ВХОД: Индивидуальная задача x массовой задачи $\mathcal{P}$.
ВХОД: Локальное решение s.

1. Используя алгоритм $A_{\mathcal{P}}$, построить исходное решение $s = A_{\mathcal{P}}(x)$.

2. Пока решение s не является локально-оптимальным выполнять:

   применить алгоритм $C_{\mathcal{P}}$ ко входу (x, s);

   если алгоритм $C_{\mathcal{P}}$ находит решение $s' \in \mathcal{N}(s, x)$ лучшее, чем s, то положить $s = s'$.

Поскольку множество решений конечно, то такой алгоритм всегда остановится, т.е. всегда существует по крайней мере один локальный оптимум для любой задачи $\mathcal{P}$ из класса PLS. Если число различных решений ограничено полиномом от размера входа, то ясно, что стандартный алгоритм локального поиска для такой задачи имеет полиномиальную трудоемкость. Однако многие интересные в прикладном смысле задачи из класса PLS имеют экспоненциальное число решений и для некоторых из этих задач трудоемкость стандартного алгоритма локального поиска экспоненциальна. Таким образом, стандартному алгоритму локального поиска в худшем случае требуется экспоненциальное время для нахождения локального оптимума, но, может быть, существует другой алгоритм, который быстрее находит этот локальный оптимум? Для ответа на этот вопрос сформулируем следующую *проблему стандартного локального поиска:* для индивидуальной задачи $x \in I_{\mathcal{P}}$, массовой задачи $\mathcal{P}$, найти локальный оптимум s, который может быть выходом стандартного алгоритма локального поиска для входа x.

Теорема 2.5. *В классе PLS существует задача $\mathcal{P}$, для которой проблема стандартного локального поиска является NP-трудной.*

Для задач поиска локального оптимума естественным образом определяется понятие PLS-сведения.

Говорят, что массовая задача $\mathcal{P} \in PLS$ является PLS-сводимой к другой задаче $\mathcal{Q} \in PLS$, если существуют вычислимые за полиномиальное время функции f_1 и f_2, такие, что f_1 отображает индивидуальную задачу x задачи $\mathcal{P}$ в индивидуальную задачу $f_1(x)$ задачи $\mathcal{Q}$ и для любого локально-оптимального решения s задачи $f_1(x)$ значение $f_2(s, x)$ является локально-оптимальным решением задачи x.

Нетрудно заметить, что так определенное понятие PLS-сводимости обладает свойством транзитивности и, если $\mathcal{Q} \in P_s$, а $\mathcal{P}$ PLS-сводится к $\mathcal{Q}$, то $\mathcal{P} \in P_s$.

Задача $P \in PLS$ является *PLS-полной*, если каждая другая задача из PLS может быть PLS-сведена к P. Проще говоря, PLS-полные задачи являются наиболее трудными в этом классе и если хотя бы одна из них может быть решена за полиномиальное время, то все остальные задачи могут быть решены за полиномиальное время.

2.3. Окрестности, основанные на структурной близости решений

Выбор окрестности, как уже отмечалось выше, играет важную роль при построении алгоритмов локального поиска. От него существенно зависит трудоемкость одного шага алгоритма, общее число шагов и, в конечном счете, качество получаемого локального оптимума. На сегодняшний день нет и, возможно никогда не будет, единого правила выбора окрестности. Для каждой задачи структуру окрестностей $\mathcal{N}$ приходится определять заново, учитывая специфику данной задачи. Более того, для каждой задачи можно предложить несколько структур окрестностей с разными по мощности множествами $\mathcal{N}(s)$ и, как следствие, разными множествами локальных оптимумов. Ниже будут приведены три примера выбора окрестностей для задачи коммивояжера, которые иллюстрируют возможные пути построения окрестностей и их свойства.

Напомним, что задача коммивояжера состоит в нахождении минимального по длине гамильтонова цикла в полном взвешенном ориентированном (или неориентированном) графе с n вершинами. Для удобства ниже будет рассматриваться только неориентированные графы с симметричной целочисленной неотрицательной матрицей расстояний $w_{ij} = w_{ji}, 1 \leq i, j \leq n$. Каждое допустимое решение задачи коммивояжера или тур в графах будем представлять в виде перестановки $\pi = \{i_1, \ldots i_n\}$. Для $\pi \in \mathrm{SOL}$ определим значение функции окрестности $N(\pi)$ как множество всех перестановок, отличающихся от π только в двух позициях (*city-swap*). Множество $N(\pi)$ содержит ровно $n(n-1)/2$ элементов и вычислительная сложность одного шага локального поиска с учетом вычисления целевой функции не превосходит $O(n^2)$ операций.

Обозначим через $W(\pi)$ длину гамильтонова цикла и определим разностный оператор Δ^2 для $W(\pi)$ следующим образом:

$$\Delta^2 W(\pi) = \frac{1}{|\mathcal{N}(\pi)|} \sum_{\pi' \in \mathcal{N}(\pi)} W(\pi') - W(\pi), \quad \pi \in \mathrm{SOL}.$$

Оператор $\Delta^2 W(\pi)$ задает среднее отклонение целевой функции $W(\pi)$ в окрестности данной перестановки π. Для любого оптимума $\pi \in \mathrm{SOL}^{\mathcal{N}}$ справедливо неравенство $\Delta^2 W(\pi) \geq 0$. Пусть W_{av} — средняя длина тура на множестве всех допустимых решений SOL.

Тогда справедливы следующие утверждения

Теорема 2.6. *Функция* $\widetilde{W} = W - W_{av}$ *удовлетворяет уравнению*

$$\Delta^2 \widetilde{W} = -\frac{4}{n}\widetilde{W}.$$

Следствие 2.1. *Любой локальный минимум* $\pi \in SOL^{\mathcal{N}}$ *имеет длину* $W(\pi) \leq W_{av}$.

Следствие 2.2. *Алгоритм локального поиска, начиная с произвольной перестановки, достигает локального оптимума за* $O(nW)$ *шагов, если длина максимального тура превосходит среднее значение* W_{av} *не более чем в* 2^W *раз.*

Аналогичные утверждения справедливы и для следующих задач:

(i) о разбиении $2n$-вершинного графа на две части по n вершин с минимальным суммарным весом ребер, соединяющих эти части;

(ii) о раскраске вершин графа в n цветов так, чтобы смежные вершины имели разные цвета;

(iii) о разбиении n-элементного множества на два подмножества так, чтобы суммарный вес одного подмножества совпал с суммарным весом другого подмножества;

(iv) о 3-выполнимости в следующей постановке: для n булевых переменных задан набор троек; каждая переменная может входить в набор с отрицанием или без него; набор считается выполненным, если он содержит разные значения, т. е. хотя бы одно истинное и хотя бы одно ложное; требуется узнать, существует ли назначение переменных, при котором все наборы будут выполнены.

2.4. Локальный поиск фиксированной глубины

В эвристиках локального поиска фиксированной глубины используются окрестности, определяемые с помощью последовательностей ограниченной длинны операций локального обмена. Пусть, при фиксированным целом $k > 0$, решается этим методом индивидуальная задача $x \in I_{\mathcal{P}}$ проблемы $\mathcal{P}$. Тогда говорят, что решение y находится в *k-обменной окрестности* $\mathcal{N}^k(s)$, если из решения s можно получить решение y, применив не более чем k операций локального обмена. Эвристики основанные на k-обменных окрестностях часто называют *k-оптимальными* (k-opt) эвристиками.

Рассмотрим 2-оптимальную эвристику для задачи о коммивоя-

жере. Этот метод основан на следующей 2-обменной окрестности: для любого тура τ 2-обменная окрестность $\mathcal{N}(\tau)$ это множество всех туров τ', которые могут быть получены из τ после удаления двух ребер (x, y) и (v, z), и добавления двух новых ребер (x, v) и (z, y). Тот же процесс построения нового тура τ' из тура τ можно описать так: выделяется некоторая цепь, которая в новом туре проходится в противоположном направлении. Такая окрестность содержит $n(n-3)/2$ элементов, что несколько меньше, чем в окрестности city-swap.

Алгоритм 2-opt реализует 2-оптимальную эвристику для задачи о коммивояжере и является примером локального поиска с 2-обменной окрестностью. В алгоритме используется функция l, определенная на множестве туров так:

$$l(\tau) = \sum_{i=1}^{n-1} D(\pi(i), \pi(i+1)) + D(\pi(n), \pi(1)),$$

где $\tau = \langle c_{\pi(1)}, \ldots, c_{\pi(n)} \rangle$.

Алгоритм 2-opt

Input: Множество городов $C = \{c_1, \ldots, c_n\}$, $n \times n$ матрица D расстояний.

Output: Перестановка $T = (c_{\pi_1}, \ldots, c_{\pi_n})$ городов.

begin
  $\tau :=$ исходный тур τ_0;
  Пусть $Q = \{(i, j) \mid i, j \in \{1, \ldots, n\}$ и $i \neq j\}$;
  $N := Q$;
  repeat
    Пусть $\tau = (c_{i_1}, \ldots, c_{i_n})$;
    Выбрать пару индексов $(p, q) \in N$;
    $N := N - \{(p, q)\}$;
    $\tau' := (c_{i_1}, \ldots, c_{i_{p-1}}, c_{i_q}, c_{i_{q-1}}, \ldots, c_{i_{p+1}}, c_{i_p}, c_{i_{q+1}}, \ldots, c_{i_n})$;
    if $l(\tau') < l(\tau)$ **then**
    begin
      $\tau := \tau'$;
      $N := Q$
    end
  until $N = \emptyset$;
  return τ
end.

Алгоритм 2-opt можно модифицировать в алгоритм локального поиска с большими окрестностями. Например, 3-opt эвристика основывается на 3-обменных окрестностях. Для тура τ его 3-обменная окрестность $\mathcal{N}(\tau)$ это множество всех туров τ', которые могут быть получены из τ после замены не более чем трех ребер. Алгоритм 3-opt эвристика имеет более лучшее приближение, но хуже по трудоемкости.

Отметим, что для задачи коммивояжера с n городами k-обменная окрестность имеет размер $\binom{n}{k} = \Theta(n^k)$. Следовательно эвристике требуется выполнить $O(n^k)$ шагов для того, чтобы удостовериться что текущее решение является локально оптимальным.

На практике эвристики локального поиска фиксированной глубины для задачи коммивояжера весьма эффективны. Однако можно построить примеры для которых эти эвристики находят решения со стоимостью далекой от оптимальной. Как уже говорилось результат, полученный на выходе эвристики сильно зависит от выбора начального тура τ_0. Следующая теорема показывает, что существуют такие начальные туры, при использовании которых эвристика дает решение хуже оптимального в произвольное число раз.

Теорема 2.7. *Для любых $k \geq 2$, $n \geq 2k+8$ и $\alpha > 1/n$ существует пример x задачи коммивояжера с n городами $\{c_1, \ldots, c_n\}$ такой, что эвристика k-opt для входа x с начальным туром $\tau_0 = (c_1, \ldots, c_n)$ находит тур стоимость, которого $m_k(x)$ удовлетворяет неравенству $m_k(x) > \alpha m^*(x)$.*

Доказательство. Предположим, что k чётное и пусть $n \geq 2k+6$. Рассмотрим следующий пример задачи коммивояжера с n городами $\{c_1, \ldots, c_n\}$ (см. левую часть Рис.1, где $k = 8$ и веса всех ребер равны kn.)

1. $D(1,2) = 1$;
2. $D(i, i+1) = \frac{1}{n\alpha}$, для $i = 2, \ldots, n-1$;
3. $D(n,1) = \frac{1}{n\alpha}$;
4. $D(k+3, 2k+4) = \frac{1}{n\alpha}$;
5. $D(j, 2k+4-j) = \frac{1}{n\alpha}$, для $j = 1, \ldots, k$;
6. $D(i,j) = kn$, для всех оставшихся пар c_i, c_j.

Рис. 1. Плохой пример для 8-opt и оптимальный тур для него

Оптимальный тур можно описать, используя функцию $f : \{1, \ldots, n\} \to \{1, \ldots, n\}$, где $f(i) = j$ обозначает, что c_j непосредственно следует за c_i в туре:

$$f(i) = \begin{cases} 2k+4-i, & \text{если } i \text{ нечетное и } i < k, \\ i+1, & \text{если } i \text{ четное и } i < k, \\ i+1, & \text{если } k \le i \le k+2, \\ 2k+4, & \text{если } i = k+3, \\ i-1, & \text{если } i \text{ нечетное и } k+3 \le i < 2k+4, \\ 2k+4-i, & \text{если } i \text{ четное и } k+3 \le i < 2k+4, \\ i+1, & \text{если } 2k+4 \le i < n, \\ 1, & \text{если } i = n. \end{cases}$$

Непосредственным подсчетом можно убедиться, что этот тур имеет длину $m^*(x) = 1/\alpha$, и его оптимальность следует из факта, что минимальное расстояние между двумя городами равно $1/n\alpha$. В действительности можно также показать, что T^* является единственным оптимальным решением. Отметим, что тур $\tau_0 = (c_1, \ldots, c_n)$ имеет длину $1 + \frac{n-1}{n\alpha}$. По построению, T^* является единственным туром с длиной меньшей чем $1 + \frac{n-1}{n\alpha}$, и он отличается от τ_0 $k+1$ ребром. Следовательно не существует более короткого тура в k-окрестности τ_0 и, поэтому τ_0 может быть локальным оптимумом, который выдает в качестве решения k-opt эвристика. Отношение стоимости этого решения к оптимуму равно

$$\frac{m_k(x)}{m^*(x)} = (1 + \frac{n-1}{n\alpha})\alpha = 1 + \alpha - \frac{1}{n} > \alpha.$$

Доказательство для случая, когда k нечетное непосредственно

следует из выше доказанного после замечания, что локальный оптимум относительно $(k+1)$-обменной окрестности является также локальным оптимумом относительно k-обменной окрестности □

Теорема 2.8. *Поиск локального оптимума в задаче коммивояжера с окрестностью k-opt является PLS-полной для некоторого k.*

Минимальное значение константы k, при котором утверждение остается верным, пока неизвестно. Известно, что утверждение справедливо при k = 8 и возможно это значение может быть уменьшено до k = 6. На практике широко применяются окрестности с k = 2, но теоретических результатов о сложности таких алгоритмов пока не получено.

В общем случае неизвестно ни одной верхней границы для отношения стоимости решения, найденного k-opt эвристикой к оптимуму. Однако такую границу можно получить для частных случаев, например, для 2-opt эвристики для Евклидовой задачи коммивояжера. В любом примере этой задачи каждый город ассоциируется с точкой $p_i = (u_i, v_i)$ на плоскости и, для любой пары городов c_i и c_j, $D(i,j)$ это Евклидово расстояние между p_i и p_j.

Лемма 2.1. *Пусть x пример Евклидовой задачи коммивояжера и пусть алгоритм 2-opt для входа x находит тур $T = (c_{i_1}, \ldots, c_{i_n})$. Для каждого $k \in \{1, \ldots, n\}$ пусть $E_k = \{(c_{i_j}, c_{j_{j+1}}) \mid D(i_j, i_{j+1}) > \frac{2m^*(x)}{\sqrt{k}}\}$. Тогда $|E_k| < k$.*

Доказательство. Доказательство можно провести от противного. Предположим, что для некоторого k, $1 < k \leq n$, выполняется неравенство $|E_k| = r \geq k$. Пусть дуга $(c_{i_j}, c_{i_{j+1}}) \in E_k$, тогда точку p_j будем называть *начальной*, а точку p_{j+1} концевой для такой дуги.

Сначала, покажем, что при этой ситуации, число концевых точек находящихся близко друг к другу ограниченно. Рассмотрим окружность $S(x,y)$ радиуса $\frac{m^*(x)}{\sqrt{k}}$ с центром (x,y). Предположим, что для некоторого $p \geq \sqrt{k}$, концевые точки $t_{s_1}, \ldots, t_{s_p}$ лежат внутри $S(x,y)$. Пусть $h_{s_1}, \ldots, h_{s_p}$ — соответствующие начальные точки. По предположению, расстояние между любыми точками t_{s_j} и t_{s_k} не превышает $\frac{2m^*(x)}{\sqrt{k}}$. Следовательно, расстояние между соответствующими начальными точками h_{s_j} и h_{s_k} не меньше, чем $\frac{2m^*(x)}{\sqrt{k}}$. Действительно, если это не так, то удалив две дуги:

h_{s_j}, t_{s_j} и h_{s_k}, t_{s_k}, и вставив две других: h_{s_j}, h_{s_k} и t_{s_j}, t_{s_k} в T получили бы более короткий тур T', но это противоречит факту, что T — локальный оптимум. Поэтому существует $p \geq \sqrt{k}$ начальных точек $h_{s_1}, \ldots, h_{s_p}$ на расстоянии не меньше чем $\frac{2m^*(x)}{\sqrt{k}}$ друг от друга. Следовательно оптимальный тур на множестве точек $\{h_{s_1}, \ldots, h_{s_p}\} \subseteq P$ имеет длину не меньше чем $2m^*(x)$. Но это невозможно, поскольку, из неравенства треугольника следует, что для любого $P' \subseteq P$ выполняется $m^*(P') \leq m^*(P)$. Полученное противоречие доказывает, что меньше чем $\sqrt{k}$ концевых точек может лежать внутри $S(x, y)$.

Сейчас покажем, что если $|E_k| \geq k$, то существует множество Q такое, что $|Q| \geq \sqrt{k}$ и любые две точки из Q лежат на расстоянии не меньше чем $\frac{m^*(x)}{\sqrt{k}}$ друг от друга. Множество Q можно построить, выполнив следующую процедуру:

begin
  $Q := \emptyset$;
  while $P \neq \emptyset$ **do**
    Выбрать произвольную концевую точку t из P;
    $Q := Q \cup \{t\}$;
    Удалить из P все концевые точки, лежащие внутри окружности радиуса $\frac{m^*(x)}{\sqrt{k}}$ с центром в t;
  return Q
end.

Поскольку, на каждом шаге, мы удаляем не долее чем $\sqrt{k}$ концевых точек из P, то, по завершении процедуры, $|Q| \geq \sqrt{k}$.

Так как расстояние между любыми двумя точками из Q больше чем $\frac{m^*(x)}{\sqrt{k}}$, то оптимальный тур для Q должен иметь длину большую, чем $m^*(x)$. Получили противоречие, следовательно должно выполняться $|E_k| < k$. □

Теорема 2.9. *Для эвристики 2-opt, предназначенной для решения Евклидовой задачи коммивояжера справедливо*

$$\frac{C^{2-opt}}{C^*} \leq 4\sqrt{n}.$$

Доказательство этой теоремы легко провести, используя вышеприведенную лемму, и поэтому оно опущено.

2.4.1 Локальный поиск для задачи о разбиении графа

В задаче о разбиении графа задан взвешенный неориентированный граф $G = (V, E)$ с четным числом вершин и весами ребер $w(e)$, $e \in E$. Далее, под *разбиением множества вершин* V всегда будем понимать разбиение на два множества (A, B) таких, что $|A| = |B| = |V|/2$. Задача о разбиении графа заключается в нахождении разбиения (A, B) множества вершин V с минимальной стоимостью $c(A, B)$, которая по определению равна сумме весов всех ребер между A и B. Самой известной эвристикой для задачи о разбиении графа является алгоритм локального поиска Кернигана—Лина. Этот алгоритм начинает свою работу со случайного разбиения множества вершин V. Начиная с текущего разбиения (A_0, B_0), жадным способом строится последовательность разбиений $(A_1, B_1), \ldots, (A_l, B_l)$. Каждое разбиение (A_k, B_k), $1 \leq k \leq l$, в этой последовательности получается из предыдущего — (A_{k-1}, B_{k-1}) обменом одной вершины из A_{k-1} на одну вершину из B_{k-1}. Стоимость каждого разбиения из такой последовательности меньше, чем стоимость текущего разбиения. Локальный поиск осуществляется среди разбиений этой последовательности. При этом выбирается разбиение с минимальной стоимостью и им заменяется текущее разбиение. Алгоритм останавливается, когда для текущего разбиения последовательность оказывается пустой.

Алгоритм Локальный поиск для ЗРГ

ВХОД: n, реберно-взвешенный граф $G = (V, E)$, $|V| = n$, разбиение A, B множества вершин V.

ВЫХОД: Локально-оптимальное разбиение A, B множества вершин V.

1. Положить $A_0 = A$ и $B_0 = B$, вычислить стоимость $c(A_0, B_0)$. Положить $i = 0$, $g_i = 0$, и $G(i) = 0$, где g_i — приращение при транспозиции; $G(i)$ — суммарный прирост при нескольких последовательных транспозициях.

2. Положить $i = 1$. Выбрать такую пару вершин $a_1 \in A_0$ и $b_1 \in B_0$, что при их транспозиции мы получаем разбиение A_1, B_1 с положительным приростом стоимости g_1, т.е. $g_1 = c(A_0, B_0) - c(A_1, B_1) > 0$. Если таких пар не существует, то

перейти на 7, иначе положить $G(1) = g_1$.

3. Положить $i = i + 1$. Для каждой пары вершин, не входящих в уже выбранные пары вершин, оценить прирост при их транспозиции. Выбрать пару вершин a_i, b_i, где $a_i \in A_{i-1}$ и $b_i \in B_{i-1}$, такую что при их транспозиции мы получаем разбиение A_i, B_i с максимальным приростом стоимости — $g_i = c(A_{i-1}, B_{i-1}) - c(A_i, B_i)$.

4. Вычислить суммарный прирост: $G(i) = \sum_{k=1}^{i} g_k$. Если $i < n/2$ и $G(i) > 0$, то перейти на 3.

5. Выбрать k , такое, что $0 \leq k \leq i$ и $G(k)$ максимально.

6. Если $k > 0$ то положить $A_0 = A_k$, $B_0 = B_k$ и перейти на 2.

7. В решаемой задаче достигнут локальный оптимум. Положить $A = A_0$ и $B = B_0$. Выйти с решением A, B стоимости $c(A, B)$.

Структура окрестностей Кернигана-Лина это структура окрестностей, соответствующая этому алгоритму локального поиска. Она определяется следующим образом. Разбиение (A', B') будем называть *s-разбиением для разбиения* (A, B) , если (A', B') можно получить из (A, B) обменом одного элемента A на один элемент B . Назовем (A', B') *жадным s-разбиением*, если $c(A, B) - c(A', B')$ достигает максимума среди всех s-разбиений для разбиения (A, B) . Если, кроме того, (A', B') является лексикографически наименьшим среди всех жадных s-разбиений, то будем говорить, что (A', B') — лексикографически жадное s-разбиение для (A, B) . Пусть (A_i, B_i) — последовательность разбиений, каждое из которых, кроме начального (A_0, B_0) , является s-разбиением для предшествующего ему. Эту последовательность будем называть монотонной, если разности $A_i - A_0$ и $B_i - B_0$ монотонно возрастают (т.е. ни одна вершина не переносится обратно в свое первоначальное множество). Наконец, мы будем говорить, что разбиение (A', B') принадлежит окрестности разбиения (A, B) , если оно появляется в максимальной монотонной последовательности с начальным разбиением (A, B) лексикографически жадных s-разбиений, однозначно выбираемых эвристикой Кернигана-Лина. Отметим, что такая последовательность состоит из $|V|/2 + 1$ разбиений, причем

последнее равно (B, A). Таким образом, каждое разбиение имеет окрестность, состоящую из $|V|/2$ разбиений.

2.5. Локальный поиск переменной глубины

Обменный алгоритм фиксированной глубины, такой как k-opt, при переходе от текущего решения к следующему, выполняет последовательно серию локальных обменов. Длина этой последовательности ограничена (не более k). При локальном поиске переменной глубины длина такой последовательности априори не ограничивается.

В обменных алгоритмах переменной глубины сначала к текущему решению s применяется последовательность из t обменов (t зависит от s и специфики эвристики), в результате получается последовательность решений $s_1, \ldots, s_t$, где $s_i \in \mathcal{N}^{)}(\int)$. Затем в качестве нового текущего выбирается лучшее среди этих решений.

Локальный поиск переменной глубины

Input: Пример x.
Output: Решение s.
begin
  $s :=$ начальное допустимое решение s_0;
  repeat
    $X := \varepsilon$ (пустую последовательность);
    $s' := s$;
    (* начало процедуры выбора следующего текущего решения *);
    Пусть C — множество обменных операций возможных на s';
    while правило остановки не выполняется **do**
    begin
      Выбрать $\tau \in C$, такое, что $\mu(x) = \max_{c \in C} \mu(c)$;
      Добавить x в конец X;
      $C := C - \{\tau\}$;
    end;
    Выбрать префиксную подпоследовательность X' из X;
    Применить X' к s чтобы получить новое решение s';
    (* окончание процедуры выбора следующего текущего решения *);
    if $m(x, s') < m(x, s)$ **then** $s := s'$
  until $s \neq s'$;

return s
end.

Для такого выбора лучшего решения, используется мера μ прироста, которая вычисляется для любой последовательности обменов, начинающейся от текущего решения.

Чтобы этот метод был эффективным должны выполняться несколько условий:

1. Функция μ должна пригодной для вычисления изменения стоимости (приращения), которое получается после применения любой последовательности обменов. Такая функция должна быть аддитивной на последовательности, может принимать отрицательные значения, и должна быть такой, что улучшение ее значения указывает вероятное направление к оптимальному решению.

2. Операция замены должна быть такой, что при последовательность любой длины, полученной из текущего решения, она дает новое допустимое решение.

3. Стоп правило должно исключать генерацию слишком длинных бесполезных и слишком коротких не качественных последовательностей.

На основе окрестности 2-opt Лином и Кернигханом предложена очень эффективная эвристика переменной глубины для задачи коммивояжера. Она позволяет заменять произвольное число ребер и переходит от одного тура к другому, используя принципы жадных алгоритмов. Основная идея эвристики заключается в следующем. Удалим из гамильтонового цикла произвольное ребро, скажем (a, b). В полученном пути один конец (вершину a) будем считать фиксированной, а другой конец будем менять, перестраивая гамильтонов путь. Добавим ребро из вершины b, например (b, c), и разорвем образовавшийся единственный цикл так, чтобы снова получить гамильтонов путь. Для этого придется удалить ребро, инцидентное вершине c. Обозначим его (c, d). Новый гамильтонов путь имеет концевые вершины a и d Эту процедуру будем называть ротацией. Для получения нового гамильтонова цикла достаточно добавить ребро (a, d). Согласно алгоритму Лина-Кернигхана, переход от одного тура к другому состоит из удаления некоторого ребра,

выполнении серии последовательных ротаций и, наконец, замыкания концевых вершин полученного гамильтонова пути. Существуют различные варианты этой основной схемы, которые отличаются правилами выбора ротаций и ограничениями на множества удаляемых и добавляемых ребер. В алгоритме Лина-Кернигхана ротации выбираются так, чтобы минимизировать разность $w_{bc} - w_{cd}$. При этом множества удаляемых и добавляемых ребер в серии ротаций не превысит n^2 и трудоемкость одного шага (перехода от одного тура к другому) останется полиномиальной. Общее число шагов алгоритма, по-видимому, не может быть ограничено полиномом и известен вариант окрестности Лина-Кернигхана, для которого задача коммивояжера становится PLS-полной.

2.6. Эвристики основанные на локальном поиске

Идеи локального поиска получили свое дальнейшее развитие в так называемых метаэвристиках, то есть в общих схемах построения алгоритмов, которые могут быть применены практически к любой задаче дискретной оптимизации. Все метаэвристики являются итерационными процедурами и для многих из них установлена асимптотическая сходимость наилучшего найденного решения к глобальному оптимуму. К числу метаэвристик относятся алгоритмы имитации отжига, поиск с запретами, генетические алгоритмы, о которых пойдет речь в этом разделе, а также нейронные сети, муравьиные колонии, вероятностные жадные процедуры и другие.

2.6.1 Алгоритм имитации отжига

Экзотическое название данного алгоритма связано с методами имитационного моделирования в статистической физике, основанными на технике Монте-Карло. Исследование кристаллической решетки и поведения атомов при медленном остывании тела привело к появлению на свет вероятностных алгоритмов, которые оказались чрезвычайно эффективными в комбинатор- ной оптимизации. Сегодня эти алгоритмы является популярными как среди практиков благодаря своей простоте, гибкости и эффективности, так и среди теоретиков, поскольку удается аналитически исследовать их свойства и доказать асимптотическую сходимость.

Алгоритм имитации отжига относится к классу пороговых ал-

горитмов локального поиска. Пусть определена (возможно бесконечная) последовательность $t_0, t_1, \ldots,$ *порогов*. На каждом шаге порогового алгоритма в окрестности текущего решения s_k выбирается некоторое решение p, и если разность по целевой функции между новым и текущим решением не превосходит заданного порога t_k, то новое решение p заменяет текущее. В противном случае выбирается новое соседнее решение. Общая схема пороговых алгоритмов может быть представлена следующим образом.

Пороговый алгоритм

1. Выбрать начальное решение $i_0 \in \mathrm{SOL}$ и положить $m^* = m(i_0)$; $k = 0$.
2. Пока не выполнен критерий остановки делать следующее:
   2.1. Случайно выбрать $j \in \mathcal{N}(i_k)$.
   2.2. Если $m(j) - m(i_k) < t_k$ то $i_{k+1} := j$.
   2.3. Если $m^* > m(i_k)$, то $m^* := m(i_k)$.
   2.4. Положить $k := k + 1$.

Отметим, что если $t_k > 0$, то возможен переход к новому решению i_{k+1} с худшим значением целевой функции. В зависимости от способа задания пороговой последовательности $\{t_k\}$ различают три типа алгоритмов

1. Последовательное улучшение: $t_k = 0$, $k = 0, 1, 2 \ldots,$ — вариант классического локального спуска с монотонным улучшением по целевой функции.

2. Пороговое улучшение: пороговая последовательность монотонно убывает до 0, то есть, $t_k = c_k$, $k = 0, 1, 2 \ldots,$ $c_k \geq c0$, $c_k \geq c_{k+1}$ и $\lim_{k\to\infty} c_k \to 0$ — вариант локального поиска, когда допускается ухудшение по целевой функции до некоторого заданного порога, и этот порог последовательно снижается до нуля.

3. Имитация отжига: $t_k \geq 0$, $k = 0, 1, 2 \ldots,$ — случайная величина с математическим ожиданием $E(t_k) = c_k \geq 0$ — вариант локального поиска, когда допускается произвольное ухудшение по целевой функции, но вероятность такого перехода обратно пропорциональна величине ухудшения, точнее для любого $j \in \mathcal{N}(i)$

$$P_{ij} = \begin{cases} 1, & \text{если } m(i) \leq m(i), \\ \exp(\frac{m(i)-m(j)}{c_k}), & \text{если } m(j) > m(i). \end{cases}$$

Последовательность $\{c_k\}$ оказывает существенное влияние на

сходимость алгоритма. Поэтому ее выбирают так, чтобы $c_k \to 0$ при $k \to \infty$. Параметр c_k называют температурой.

Поведение алгоритма "имитация отжига"определяется структурой окрестностей $\mathcal{N}$, начальным решением s_0, и значениями параметров r, t, l. Правило остановки представлено в алгоритме предикатом Frozen.

Имитация отжига

Input: Пример x.
Output: Решение s.

```
begin
  τ := t;
  s := начальное допустимое решение s_0;
  repeat
    for i = 1 to l do
    (* локальный поиск при температуре τ *);
    begin
      Выбрать не исследованное решение s' ∈ (s);
      if m(x, s') < m(x, s) then
        s := s'
      else;
      begin
        δ := m(x, s') − m(x, s);
        s := s' с вероятностью e^{−δ/τ}
      end
    end;
    (* изменение температуры *);
    τ := rτ
  until Frozen;
  return s
end.
```

2.6.2 Поиск с запретами

Основоположником алгоритма поиска с запретами (Tabu search) является Ф. Гловер, который предложил принципиально новую схему локального поиска. Она позволяет алгоритму не останавливаться в точке локального оптимума, как это предписано в стандартном алгоритме локального спуска, а путешествовать от одного оптимума к другому в надежде найти среди них глобальный оптимум.

Основным механизмом, позволяющим алгоритму выбираться из локального оптимума, является список запретов $Tabu(i_k)$. Он строится по предыстории поиска, то есть по нескольким последним точкам $i_k, i_{k-1}, \ldots, i_{k-l+1}$, и запрещает исследовать часть окрестности $\mathcal{N}(i_k)$ текущего решения i_k. Точнее на каждом шаге алгоритма очередная точка i_{k+1} является оптимальным решением подзадачи

$$m(i_{k+1}) = \min\{m(j) \mid j \in \mathcal{N}(i_k) \setminus Tabu_l(i_k)\}.$$

Множество $Tabu_l(i_k) \subseteq \mathcal{N}(i_k)$ определяется по предшествующим решениям. Список запретов учитывает специфику задачи и, как правило, запрещает использование тех "фрагментов"решения (ребер графа, координат вектора, цвет вершины), которые менялись на последних l шагах алгоритма. Константа $l \geq 0$ определяет его память. При "короткой памяти"($l = 0$) получаем стандартный локальный спуск.

Существует много вариантов реализации основной идеи поиска с запретами. Приведем один из них, для которого удается установить асимптотические свойства. Рассмотрим рандомизированную окрестность $\mathcal{N}_p(i) \subseteq \mathcal{N}(i)$, где каждый элемент окрестности $j \in \mathcal{N}(i)$ включается в множество $\mathcal{N}_p(i)$ с вероятностью p независимо от других элементов. С ненулевой вероятностью множество $\mathcal{N}_p(i)$ может совпадать с $\mathcal{N}(i)$, может оказаться пустым или содержать ровно один элемент. Общая схема алгоритма поиска с запретами может быть представлена следующим образом.

Алгоритм поиска с запретами

1. Выбрать начальное решение $i_0 \in \mathrm{SOL}$ и положить $m^* = m(i_0)$; $k = 0$.

2. Пока не выполнен критерий остановки делать следующее:

2.1. Сформировать окрестность $\mathcal{N}_p(i_k)$.

2.2. Если $\mathcal{N}_p(i_k) \neq \emptyset$, то $i_{k+1} := i_k$, иначе найти i_{k+1} такой, что $m(i_{k+1}) = \min\{m(j) \mid j \in \mathcal{N}_p(i_k) \setminus Tabu_l(i_k)\}$.

2.3. Если $m^* > m(i_{k+1})$, то $m^* := m(i_{k+1})$.

2.4. Положить $k := k+1$ и обновить список запретов $Tabu_l(i_k)$.

Параметры p и l являются управляющими для данного алгоритма и выбор их значений зависит от размерности задачи и мощности окрестности.

2.6.3 Генетические алгоритмы

Идея генетических алгоритмов заимствована у живой природы и состоит в организации эволюционного процесса, конечной целью которого является получение оптимального решения в сложной комбинаторной задаче. Разработчик генетических алгоритмов выступает в данном случае как "создатель", который должен правильно установить законы эволюции, чтобы достичь желаемой цели как можно быстрее. Впервые эти нестандартные идеи были применены к решению оптимизационных задач в середине 70-х годов. Примерно через десять лет появились первые теоретические обоснования этого подхода. На сегодняшний день генетические алгоритмы доказали свою конкурентноспособность при решении многих NP-трудных задач и особенно в практических приложениях, где математические модели имеют сложную структуру и применение стандартных методов типа ветвей и границ, динамического или линейного программирования крайне затруднено. Общую схему генетических алгоритмов проще всего понять, рассматривая задачи безусловной оптимизации

$$max\{m(i) \mid i \in B^n\}, \quad B^n = \{0,1\}^n.$$

Примерами служат задачи размещения, стандартизации, выполнимости и другие. Стандартный генетический алгоритм начинает свою работу с формирования начальной *популяции* $I_0 = \{i_1, i_2, \ldots, i_s\}$ — конечного набора допустимых решений задачи. Эти решения могут быть выбраны случайным образом или получены с помощью вероятностных жадных алгоритмов. Как мы увидим ниже, выбор начальной популяции не имеет значения для сходимости процесса в асимптотике, однако формирование "хорошей"начальной популяции (например из множества локальных оптимумов) может заметно сократить время достижения глобального оптимума.

На каждом шаге эволюции с помощью вероятностного оператора селекции выбираются два решения, *родители* $i_1,\ i_2$. Оператор *скрещивания* по решениям $i_1,\ i_2$ строит новое решение i', которое затем подвергается небольшим случайным модификациям, которые принято называть *мутациями*. Затем решение добавляется в популяцию, а решение с наименьшим значением целевой функции уда-

ляется из популяции. Общая схема такого алгоритма может быть записана следующим образом.

Генетический алгоритм

1. Выбрать начальную популяцию I_0 и положить $m^* = \max\{m(i) \mid i \in I_0\}$; $k := 0$.
2. Пока не выполнен критерий остановки делать следующее:
   2.1. Выбрать родителей i_1, i_2. из популяции I_k.
   2.2. Построить i' по i_1, i_2.
   2.3. Модифицировать i'.
   2.4. Если $m^* < m(i')$, то $m^* := m(i')$.
   2.5. Обновить популяцию и положить $k := k + 1$.

Остановимся подробнее на основных операторах этого алгоритма: селекции, скрещивании и мутации. Среди операторов селекции наиболее распространенными являются два вероятностных оператора пропорциональной и турнирной селекции. При пропорциональной селекции вероятность на k-м шаге выбрать решение i в качестве одного из родителей задается формулой

$$P(i\text{– выбрано}) = \frac{m(i)}{\sum_{j \in I_k} m(j)}, \quad i \in I_k,$$

в предположении, что $m(i) > 0$ для всех $i \in \mathrm{SOL}$. При турнирной селекции формируется случайное подмножество из элементов популяции и среди них выбирается один элемент с наибольшим значением целевой функции. Турнирная селекция имеет определенные преимущества перед пропорциональной, так как не теряет своей избирательности, когда в ходе эволюции все элементы популяции становятся примерно равными по значению целевой функции. Операторы селекции строятся таким образом, чтобы с ненулевой вероятностью любой элемент популяции мог бы быть выбран в качестве одного из родителей. Более того, допускается ситуация, когда оба родителя представлены одним и тем же элементом популяции.

Как только два решения выбраны, к ним применяется вероятностный оператор скрещивания (*crossover*). Существует много различных версий этого оператора, среди которых простейшим, по видимому, является однородный оператор. По решениям i_1, i_2 он строит решение i' присваивая каждой координате этого вектора с вероятностью 0,5 соответствующее значение одного из родителей. Если вектора i_1, i_2 совпадали скажем по первой координате,

то вектор i' "унаследует"это значение. Геометрически, оператор скрещивания случайным образом выбирает в гиперкубе вершину i', которая принадлежит минимальной грани, содержащей вершины i_1, i_2. Можно сказать, что оператор скрещивания старается выбрать новое решение i' где-то между i_1, i_2 полагаясь на удачу. Более аккуратная процедура могла бы выглядеть таким образом. Новым решением i' является оптимальное решение исходной задачи на соответствующей грани гиперкуба. Конечно, если расстояние Хемминга между i_1, i_2 равно n, то задача оптимального скрещивания совпадает с исходной. Тем не менее даже приближенное решение этой задачи вместо случайного выбора заметно улучшает работу генетического алгоритма. По аналогии с однородным оператором скрещивания легко предложить и другие операторы, использующие не только два, но и произвольное число решений из популяции.

Оператор мутации, применяемый к решению i' в п. 2.3. генетического алгоритма, с заданной вероятностью $p_m \in (0,1)$ меняет значение каждой координаты на противоположное. Например, вероятность того, что $i' = (0,0,0,0,0)$ в ходе мутации перейдет в $j' = (1,1,1,0,0)$, равна $p_m \times p_m \times p_m \times (1-p_m) \times (1-p_m) > 0$. Таким образом, с ненулевой вероятностью решение i' может перейти в любое другое решение. Отметим, что модификация решения i' может состоять не только в случайной мутации, но и в частичной перестройке решения алгоритмами локального поиска. Применение локального спуска позволяет генетическому алгоритму сосредоточиться только на локальных оптимумах. Множество локальных оптимумов может оказаться экспоненциально большим и на первый взгляд кажется, что такой вариант алгоритма не будет иметь больших преимуществ. Однако экспериментальные исследования распределения локальных оптимумов свидетельствуют о высокой концентрации их в непосредственной близости от глобального оптимума. Это наблюдение известно как гипотеза о существовании "большой долины"для задач на минимум или "центрального горного массива"для задач на максимум.

Гипотеза 1. В среднем локальные оптимумы расположены гораздо ближе к глобальному оптимуму чем к случайно выбранной точке. Их распределение в области допустимых решений на является равномерным. Они концентрируются в районе глобального оп-

тимума, занимая область небольшого диаметра.

Эта гипотеза отчасти объясняет работоспособность генетических алгоритмов. Если в популяции собираются локальные оптимумы, которые согласно гипотезе сконцентрированы в одном месте, и очередное решение i' выбирается где-то между двумя произвольными локальными оптимумами, то такой процесс имеет много шансов найти глобальный оптимум. Аналогичные рассуждения объясняют работоспособность и других локальных алгоритмов. В связи с этим проверка и теоретическое обоснование данной гипотезы представляет несомненный интерес.

Глава 3

Приближенные алгоритмы с гарантированными оценками точности

Рассмотрим задачу минимизации целевого функционала $F(x)$ на множестве X. Обозначим $F^* = \min\{F(x)|x \in X\}$. Предположим, что $F^* > 0$.

Пусть приближенный алгоритм H отыскивает элемент множества X со значением F^H целевого функционала. Введем в рассмотрение величину $\Delta_H = F^H/F^*$. Эта величина зависит от алгоритма H и от набора входных данных задачи. В данном разделе под гарантированной оценкой точности решения, получаемого при помощи H, будем понимать верхнюю оценку величины Δ_H при любом наборе входных данных задачи.

Построение эффективных приближенных алгоритмов с гарантированными оценками точности имеет смысл для NP-трудных в сильном смысле задач и NP-трудных задач, для которых не известны быстродействующие ε-приближенные алгоритмы.

В дальнейшем обозначения F^*, F^H и Δ_H используются при рассмотрении конкретных задач и конкретных приближенных алгоритмов. Смысл этих обозначений не изменяется.

Рассмотрим NP-трудную в сильном смысле задачу $P//\sum w_jC_j$. В этой задаче расписание однозначно определяется разбиением множества требований на подмножества $N_1, \ldots, N_m$ и указанием порядка обслуживания требований в каждом подмножестве. При помощи перестановочного приема нетрудно показать, что достаточно ограничиться рассмотрением расписаний, при которых требования каждого подмножества упорядочены по правилу SWPT, т.е. по неубыванию значений p_j/w_j. Таким образом, расписание

однозначно определяется разбиением множества требований на m подмножеств.

Опишем приближенный алгоритм A решения задачи $P//\sum w_jC_j$, который состоит в следующем. Перенумеруем требования в порядке SWPT так, что $p_1/w_1 \le \cdots \le p_n/w_n$. Организуем m подмножеств $N_1, \ldots, N_m$, полагая вначале $N_1 = \cdots = N_m = \phi$. Будем последовательно, начиная с первого, распределять требования $1, \ldots, n$ по подмножествам. Пусть требования $0, \ldots, k$, $k < n$, распределены, где 0 – фиктивное требование, соответствующее начальной ситуации, $p_0 = 0$. Требование $k+1$ назначается следующим образом. Обозначим $P_l = \sum_{j \in N_l} p_j$, $l = 1, \ldots, m$. Находим такое подмножество N_l, что $P_l = \min_{1 \le h \le m} P_h$. Полагаем $N_l = N_l \cup \{k+1\}$ и, если $k+1 \ne n$, переходим к назначению требования $k+2$.

Временная сложность алгоритма A равна $O(n \log n)$, если хранить динамический массив $P_1, \ldots, P_m$ в виде пирамиды или сбалансированного 2-3 дерева.

Обозначим через F_1^* и F_n^* минимальное значение функционала $\sum w_jC_j$ при $m = 1$ и $m = n$ соответственно. Очевидно, что $F_1^* = \sum_{j=1}^n w_j \sum_{i=1}^j p_i$ и $F_n^* = \sum_{j=1}^n w_j p_j$.

Найдем верхнюю и нижнюю оценки значения F^*. Очевидно, что $F^* \le F^A$. Поскольку на итерации j алгоритма A выполняется

$$P_l = \min_{1 \le h \le n} \{P_h\} \le \frac{1}{m} \sum_{i=1}^{j-1} p_i,$$

имеем

$$F^A \le \sum_{j=1}^n w_j (\frac{1}{m} \sum_{i=1}^{j-1} p_i + p_j) = \frac{1}{m} \sum_{j=1}^n w_j \sum_{i=1}^j p_i +$$

$$(1 - \frac{1}{m}) \sum_{j=1}^n w_j p_j = \frac{1}{m} F_1^* + \frac{m-1}{m} F_n^*.$$

Для получения нижней оценки значения F^* понадобится ряд вспомогательных утверждений.

Лемма 3.1. *Пусть $b_1, \ldots, b_n$ – действительные числа. Тогда* $(\sum_{j=1}^n b_j)^2 \le n \sum_{j=1}^n b_j^2$.

Лемма 3.2. *Если $p_j/w_j = C$, $j = 1, \ldots, n$, то* $F_1^* = \frac{1}{2C}(\sum_{j=1}^n p_j)^2 + \frac{1}{2} F_n^*$.

Лемма 3.3. *Если* $p_j/w_j = C,\ \ j = 1,\dots,n,$ *то* $F^* - \frac{1}{2}F_n^* \geq \frac{1}{m}(F_1^* - \frac{1}{2}F_n^*)$.

Теорема 3.1. *Для любого расписания справедливо* $\sum w_j p_j \geq \frac{1}{m}F_1^* + \frac{m-1}{2m}F_n^*$.

Доказательство. Воспользуемся индукцией по числу r различных значений p_j/w_j. При $r = 1$ справедливость утревждения следует из леммы 3.3. Пусть теорема верна при всех $r \leq k-1,\ \ k \geq 2$.

Пусть имеется k различных значений p_j/w_j и $p_1/w_1 = \dots = p_q/w_q > p_{q+1}/w_{q+1}$. Положим $\gamma = w_{q+1}p_1/(p_{q+1}w_1)$ и рассмотрим вспомогательную задачу, в которой веса обозначены $\bar{w}_j$ и для первых q требований $\bar{w}_j = \gamma w_j,\ \ j = 1,\dots,q$. Очевидно, что во вспомогательной задаче количество различных значений $p_j/\bar{w}_j$ равно $k-1$. Для вспомогательной задачи введем обозначения $\bar{F}_1^*$ и $\bar{F}_n^*$ аналогичные обозначениям для исходной задачи. В силу индуктивного предположения,

$$\sum \bar{w}_j C_j - \frac{1}{2}\bar{F}_n^* \geq \frac{1}{m}(\bar{F}_1^* - \frac{1}{2}\bar{F}_n^*)$$

для любого расписания.

Введем обозначения

$$X = F_1^* - \frac{1}{2}F_n^*,\ \ \bar{X} = \bar{F}_1^* - \frac{1}{2}\bar{F}_n^*,$$

$$Y = \sum w_j C_j - \frac{1}{2}F_n^*,\ \ \bar{Y} = \sum \bar{w}_j C_j - \frac{1}{2}\bar{F}_n^*,$$

$$\bar{X}^q = \sum_{j=1}^{q} \bar{w}_j \sum_{i=1}^{j} p_i - \frac{1}{2}\sum_{j=1}^{q} \bar{w}_j p_j,\ \ \delta(\bar{X}) = \bar{X} - \bar{X}^q,$$

$$\bar{Y}^q = \sum_{j=1}^{q} \bar{w}_j C_j - \frac{1}{2}\sum_{j=1}^{q} \bar{w}_j p_j,\ \ \delta(\bar{Y}) = \bar{Y} - \bar{Y}^q.$$

Из леммы 3.3 следует, что $m\bar{Y} \geq \bar{X}$ и $m\bar{Y}^q \geq \bar{X}^q$, а из равенств $\bar{w}_j = \gamma w_j,\ \ j = 1,\dots,q,$ – $X = \frac{1}{\gamma}\bar{X}^q + \delta(\bar{X})$ и $Y = \frac{1}{\gamma}\bar{Y}^q + \delta(\bar{Y})$.

Возможны два случая: 1) $\delta(\bar{X}) \leq m\delta(\bar{Y})$ и 2) $\delta(\bar{X}) > m\delta(\bar{Y})$.

В первом случае получаем $X \leq mY$, т.е. $\frac{1}{m}(F_1^* - \frac{1}{2}F_n^*) \leq \sum w_j C_j - \frac{1}{2}F_n^*$.

Во втором случае из неравенства $m\bar{Y}^q \geq \bar{X}^q$ следует, что $m\delta(\bar{X})\bar{Y}^q > m\bar{X}^q\delta(Y)$. Поскольку $\gamma < 1$, получаем $(1-\gamma)\delta(\bar{X})\bar{Y}^q > (1-\gamma)\bar{X}^q\delta(Y)$ или $\delta(\bar{X})\bar{Y}^q + \gamma\bar{X}^q\bar{Y}^q\delta(Y) > \bar{X}^q\delta(\bar{Y}) + \gamma\delta(\bar{X})\bar{Y}^q$. Прибавляя к обеим частям этого неравенства величину

$\bar{X}^q\bar{Y}^q + \gamma\delta(\bar{X})\delta(\bar{Y})$, получаем $(\bar{X}^q + \delta(\bar{X}))(\bar{Y} + \gamma\delta(\bar{Y})) > (\bar{X}^q + \gamma\delta(\bar{X}))(\bar{Y}^q + \delta(\bar{Y}))$. Однако,

$$\bar{X}^q + \delta(\bar{X}) = \bar{X}, \quad \bar{Y}^q + \delta(\bar{Y}) = \bar{Y},$$

$$\bar{Y}^q + \gamma\delta(\bar{Y}) = \gamma Y, \quad \bar{X}^q + \gamma\delta(\bar{X}) = \gamma X.$$

Поэтому последнее неравенство принимает вид $\gamma\bar{X}Y > \gamma X\bar{Y}$. Из $m\bar{Y} \geq \bar{X}$ и последнего неравенства следует $\gamma m\bar{Y}Y > \gamma X\bar{Y}$. Таким образом, $mY > X$. ∎

Из теоремы следует, что $F^* \geq \frac{1}{m}F_1^* + \frac{m-1}{2m}F_n^* \geq F^A - \frac{m-1}{2m}F_n^*$.

Сейчас нетрудно получить оценку сверху величины Δ_A :

$$\Delta_A = F^A/F^* \leq (F^* + \frac{m-1}{2m}F_n^*)/F^* \leq \frac{3m-1}{2m},$$

поскольку $F_n^* \leq F^*$.

Рассмотрим задачу об упаковке в контейнеры, которая в терминах теории расписаний может быть сформулирована следующим образом. В условиях задачи $P/d_j = d/C_j \leq d_j$ предполагается, что $m = n$, $d \geq \max_j p_j$ и требуется отыскать минимальное число приборов и соответствующее расписание, при котором все требования будут обслужены к заданному директивному сроку d.

Приведем описание четырех приближенных алгоритмов решения задачи об упаковке в контейнеры. В алгоритмах рассматривается некоторая перестановка π требований, требования последовательно выбираются из этой перестановки и назначаются на приборы в соответствии с некоторыми правилами. Резервом времени прибора l, обозначаемым R_l, называется разница между d и суммарной длительностью обслуживания требований, назначенных на прибор l.

В алгоритмах B1 и B2 перестановка π является произвольной. На шаге k алгоритма B1 или B2 выбирается требование, расположенное на месте k в перестановке π. В алгоритме B1 это требование назначается на прибор с наименьшим номером среди приборов с достаточным резервом времени для завершения этого требования с срок. В алгоритме B2 это требование назначается на прибор с наименьшим достаточным резервом времени.

Алгоритмы B3 и B4 отличаются от алгоритмов B1 и B2 соответственно лишь тем, что требования в перестановке π расположены в порядке LPT невозрастания значений p_j.

Алгоритм Bi может быть реализован за время $O(n \log m_i)$, где

m_i – количество приборов, найденное этим алгоритмом. Каждый из приборов обслуживает хотя бы одно требование. Очевидно, что $m_i \le n, \;\; i = 1, 2, 3, 4$.

Оценим точность получаемых при помощи алгоритмов B1–B4 решений. Не ограничивая общности, будем считать, что $d = 1$ и $p_j \le 1, \;\; j = 1, \ldots, n$.

Докажем ряд вспомогательных утверждений для алгоритмов B1 и B2. Введем в рассмотрение функцию $\psi(x), \;\; x \in [0, 1]$:

$$\psi(x) = \begin{cases} \frac{6}{5}x, & x \in [0, 1/6], \\ \frac{6}{5}x - 1/10, & x \in (1/6, 1/3], \\ \frac{6}{5}x + 1/10, & x \in (1/3, 1/2], \\ 1, & x \in (1/2, 1]. \end{cases}$$

Далее будем рассматривать расписания, построенные по алгоритмам B1 либо B2.

Лемма 3.4. *Пусть на некоторый прибор назначены требования* $1, \ldots, j$. *Тогда* $\sum_{i=1}^{j} \psi(p_i) \le 17/10$.

Доказательство. Не ограничивая общности, предположим, что $p_1 \ge \cdots \ge p_j$. Если $p_1 \le 1/2$, то из определения $\psi(x)$ следует $\psi(p_i) \le \frac{3}{2} p_i, \;\; i = 1, \ldots, j$. Тогда $\sum_{i=1}^{j} \psi(p_i) \le \frac{3}{2} \sum_{i=1}^{j} p_i \le 3/2$.

Если $p_1 > 1/2$, то $\sum_{i=2}^{j} p_i < 1/2$. В этом случае достаточно показать, что $\sum_{i=2}^{j} \psi(p_i) \le 7/10$. Возможны следующие варианты.

1) $p_2 \in (1/3, 1/2], \;\; p_3 \in (0, 1/6]$,

2) $p_2 \in (1/6, 1/3], \;\; p_3 \in (0, 1/6]$,

3) $p_2, p_3 \in (1/6, 1/3], \;\; p_4 \in (0, 1/6]$,

4) $p_2 \in (0, 1/6]$.

При вариантах 1) и 2) имеем соответственно

$$\sum_{i=2}^{j} \psi(p_i) < \frac{6}{5} p_2 + \frac{1}{10} + \frac{6}{5}(\frac{1}{2} - p_2) = \frac{7}{10}$$

и

$$\sum_{i=2}^{j} \psi(p_i) < \frac{9}{5} p_2 - \frac{1}{10} + \frac{6}{5}(\frac{1}{2} - p_2) = \frac{1}{2} + \frac{3}{5} p_2 \le \frac{7}{10}.$$

При варианте 3) имеем

$$\sum_{i=2}^{j} \psi(p_i) < \frac{9}{5}(p_2 + p_3) - \frac{2}{10} + \frac{6}{5}(\frac{1}{2} - p_2 - p_3) = \frac{2}{5} + \frac{3}{5}(p_2 + p_3) < \frac{7}{10},$$

поскольку $p_2 + p_3 < \frac{1}{2}$.

При варианте 4) имеем

$$\sum_{i=2}^{j}\psi(p_i) < \frac{6}{5}\sum_{i=2}^{j} p_i < \frac{6}{10}.$$

■

Для прибора l обозначим через Q_l максимальный резерв времени для приборов $h = 1, \ldots, l-1$, т.е. $Q_l = \max_{1\le h\le l-1} R_h$. Положим $Q_0 = 0$.

Лемма 3.5. *Для любого требования, назначенного на прибор l с $R_l \ge 1/2$ справедливо $p_i > Q_l$.*

Доказательство. Для алгоритма B1 $p_i > Q_l$ для любого требования, назначенного на прибор l. Рассмотрим алгоритм B2.

Пусть $Q_l < 1/2$. Тогда $R_h < 1/2$, $h = 1, \ldots, l-1$, и длительность обслуживания первого назначенного на прибор l требования больше, чем Q_l. Обозначим эту длительность через τ. Если $\tau > 1/2$, то лемма доказана. Если $\tau \le 1/2$, то $R_l \ge 1/2$ и длительность второго назначаемого на прибор l требования больше, чем Q_l. Обозначим эту длительность через τ'. Если $\tau + \tau' > 1/2$, то лемма доказана. Если $\tau + \tau' \le 1/2$, то длительность третьего назначаемого на прибор l требования больше, чем Q_l. Повторяя приведенные рассуждения конечное число раз, получаем справедливость утверждения леммы.

При $Q_l \ge 1/2$ имеем $\tau > Q_l \ge 1/2$, что и требуется. ■

Лемма 3.6. *Пусть на прибор l назначены требования $1, \ldots, j$ и $Q_l < 1/2$. Если $\sum_{i=1}^{j} p_i \ge 1 - Q_l$, то $\sum_{i=1}^{j}\psi(p_i) \ge 1$.*

Доказательство. Не ограничивая общности, предположим, что $p_1 \ge \cdots \ge p_j$. Если $p_1 > 1/2$, то $\psi(p_1) = 1$ и справедливость утверждения леммы доказана.

Пусть $p_1 \le 1/2$. Поскольку $\sum_{i=1}^{j} p_i \ge 1 - Q_l > 1/2$, имеем $l \ge 2$. Из леммы 3.5 и $p_1 \le 1/2$ следует, что $p_2 > Q_l$. Возможны варианты: 1) $Q_l \in (0, 1/6]$, 2) $Q_l \in (1/6, 1/3]$, 3) $Q_l \in (1/3, 1/2]$.

В условиях первого варианта $\sum_{i=1}^{j}\psi(p_i) \ge \frac{6}{5}\sum_{i=1}^{j} p_i \ge \frac{6}{5}(1 - Q_l) \ge \frac{6}{5}\frac{5}{6} = 1$.

Пусть в условиях второго варианта $j = 2$. Поскольку $p_1 + p_2 > 1/2$, имеем либо $p_2 \ge 1/3$, либо $p_1 \ge 1/3$ и $Q_l < p_2 < 1/3$. В первом случае $\psi(p_1) + \psi(p_2) = \frac{6}{5}p_1 + \frac{9}{5}p_2 = \frac{6}{5}(p_1 + p_2) + \frac{3}{5}p_2 > \frac{6}{5}(1 - Q_l) + \frac{3}{5}Q_l = \frac{6}{5} - \frac{3}{5}Q_l \ge 1$, поскольку $Q_l \le \frac{1}{3}$.

Пусть в условиях второго варианта $j \geq 3$. Если $p_2 \geq 1/3$ или $p_1 \geq 1/3$ и $Q_l < p_2 < 1/3$, то как показано выше, $\psi(p_1) + \psi(p_2) \geq 1$. Предположим, что $Q_l < p_2 \leq p_1 < 1/3$. Тогда $\sum_{i=1}^{j} \psi(p_i) \geq \frac{9}{5}p_1 + \frac{9}{5}p_2 - \frac{1}{5} + \frac{6}{5}\sum_{i=3}^{j} p_i \geq \frac{9}{5}(p_1 + p_2) - \frac{1}{5} + \frac{6}{5}(1 - Q_l - p_1 - p_2) = \frac{3}{5}(p_1 + p_2) - \frac{1}{5} + \frac{6}{5}(1 - Q_l) > \frac{6}{5} - \frac{1}{5} = 1$.

В условиях третьего варианта из $j \geq 2$ следует $p_1 > 1/3$, $p_2 > 1/3$ и далее $\sum_{i=1}^{j} \psi(p_i) \geq \frac{6}{5}(p_1 + p_2) + \frac{1}{5} > 1$. Лемма доказана. ■

Лемма 3.7. *Пусть на прибор l назначены требования $1, \ldots, j$ и $Q_l < 1/2$. Если $\sum_{i=1}^{j} \psi(p_i) = 1 - \alpha$ и $\alpha > 0$, то $p_i \leq 1/2$, $i = 1, \ldots, j$, и $\sum_{i=1}^{j} p_i \leq 1 - Q_l - \frac{5}{9}\alpha$ при $j \geq 2$.*

Доказательство. Не ограничивая общности, предположим, что $p_1 \geq \cdots \geq p_j$. Если $p_i > 1/2$, то $\psi(p_i) = 1$, что противоречит условию леммы.

Пусть $l \geq 2$. Поскольку $p_1 \leq 1/2$, в силу леммы 3.6 имеем $p_1 \geq p_2 > Q_l$. В силу леммы 3.6 имеем $\sum_{i=1}^{j} p_i = 1 - Q_l - \gamma$, где $\gamma > 0$. Поскольку $\sum_{i=1}^{j} p_i + \gamma < 1$, можно выбрать числа $\delta_1 \leq 1/2$ и $\delta_2 \leq 1/2$ такие, что $\delta_1 \geq p_1$, $\delta_2 \geq p_2$ и $\delta_1 + \delta_2 = p_1 + p_2 + \gamma$.

Заменим требования с длительностями p_1 и p_2 на требования с длительностями δ_1 и δ_2. Очевидно, что алгоритмы B1 и B2 распределят новое множество требований по приборам так же, как и исходное с заменой на приборе l требований с длительностями p_1 и p_2 на требования с длительностями δ_1 и δ_2. В силу леммы 3.6, $\sum_{i=3}^{l} \psi(p_i) + \psi(\delta_1) + \psi(\delta_2) \geq 1$. Для любых x, y, $0 \leq x < y \leq 1/2$, выполняется $\psi(y) \leq \psi(x) + \frac{9}{5}(y - x)$. Поэтому $\psi(\delta_1) + \psi(\delta_2) \leq \psi(p_1) + \psi(p_2) + \frac{9}{5}\gamma$. Отсюда получаем $\sum_{i=1}^{j} \psi(p_i) + \frac{9}{5}\gamma \geq 1$. Следовательно, $\frac{9}{5}\gamma \geq \alpha$ и $\gamma \geq \frac{5}{9}\alpha$. Лемма доказана. ■

Теорема 3.2. *Справедливы неравенства $m_1 < \frac{17}{10}m^* + 2$ и $m_2 < \frac{17}{10}m^* + 2$.*

Доказательство. Обозначим $\Psi = \sum_{i=1}^{n} \psi(p_i)$. Из леммы 3.5 следует, что $\frac{17}{10}m^* \geq \Psi$. Пусть требования $1, \ldots, n$ распределены по приборам. Обозначим через N_l множество требований, назначенных на прибор l. Пусть $\psi(N_l) = \sum_{i \in N_l} \psi(p_i)$. Из последовательности приборов, на каждый из которых назначено хотя бы одно требование, выделим подпоследовательность $l'_1, \ldots, l'_k$ всех приборов таких, что $\psi(N_{l'_h}) = 1 - \alpha_h$, $\alpha_h > 0$, $h = 1, \ldots, k$. Для любого $i \in N_{l'_h}$, $h = 1, \ldots, k$, выполняется $p_i \leq 1/2$, поскольку в

противном случае $\psi(N_{l'_h}) \geq 1$. Отсюда следует, что $Q_{l'_h} < 1/2$, $h = 1, \ldots, k$.

Покажем, что $Q_{l'_{h+1}} \geq Q_{l'_h} + \frac{5}{9}\alpha_h$, $h = 1, \ldots, k$. Очевидно, что $Q_{l'_1} = 0$ и в силу леммы 3.7 $\sum_{i \in N_{l'_1}} p_i \leq 1 - \frac{5}{9}\alpha_1$. Тогда $Q_{l'_2} \geq \frac{5}{9}\alpha_1$. В силу леммы 3.7 имеем $\sum_{i \in N_{l'_2}} p_i \leq 1 - \frac{5}{9}(\alpha_1 + \alpha_2)$. Поэтому $Q_{l'_3} \geq \frac{5}{9}(\alpha_1 + \alpha_2) = Q_{l'_2} + \frac{5}{9}\alpha_2$ и т.д.

Таким образом, $\sum_{h=1}^{k-1} \alpha_h \leq \frac{9}{5} \sum_{h=1}^{k-1} (Q_{l'_{h+1}} - Q_{l'_h}) = \frac{9}{5}(Q_{l'_k} - Q_{l'_1}) < \frac{9}{10}$, поскольку $Q_{l'_h} < 1/2$, $h = 1, \ldots, k$. Отсюда и из $\alpha_k < 1$ получаем $\sum_{h=1}^{k} \alpha_h < 2$.

Как нетрудно убедиться, $\psi(N_l) \geq 1$, если $\sum_{j \in N_l} p_j = 1$. Пусть Θ – множество всех таких приборов l, что $\sum_{j \in N_l} p_j = 1$. Тогда $m_1 \leq \sum_{l \in \Theta} \psi(N_l) + k \leq \sum_{l \in Theta} \psi(N_l) + \sum_{h=1}^{k} \psi(N_{l'_h}) + \sum_{h=1}^{k} \alpha_h < \Psi + 2 \leq \frac{17}{10} m^* + 2$. Аналогично, $m_2 < \frac{17}{10} m^* + 2$. Теорема доказана. ■

Из теоремы следует, что $\Delta_H \leq \frac{17}{10} + \frac{2}{m^*}$ для алгоритма $H \in \{B1, B2\}$. Поскольку $m^* \geq \lceil P \rceil$, где $P = \sum_{j=1}^{n} p_j$, то $\Delta_H \leq \frac{17}{10} + \frac{2}{\lceil P \rceil}$. Напомним, что здесь предполагается $d = 1$ и $p_j \leq 1$, $j = 1, \ldots, n$. Если d и $p_j \leq 1$, $j = 1, \ldots, n$, – произвольные числа, то $\Delta_H \leq \frac{17}{10} + \frac{2}{\lceil P/d \rceil}$.

Перейдем к рассмотрению алгоритмов B3 и B4.

Теорема 3.3. *Справедливы неравенства* $m_3 \leq \frac{11}{9} m^* + 4$ *и* $m_4 \leq \frac{11}{9} m^* + 4$.

Схема доказательства теоремы 3.3 аналогична схеме доказательства теоремы 3.2. Оно сводится к построению такой весовой функции, аналогичной $\psi(x)$, что а) суммарный вес требований, назначенных на один и тот же прибор, не превосходит некоторой константы $C \geq 1$, и б) число m_3 или m_4 может превосходить суммарный вес всех требований не более, чем на некоторую константу C'. В случае алгоритмов B1 и B2 имеем $C = 17/10$, $C' = 2$, а в случае алгоритмов B3 и B4 – $C = 9/11$ и $C' = 4$.

Из теоремы 3.3 следует, что $\Delta_H \leq \frac{11}{9} + \frac{4}{\lceil P/d \rceil}$, $H \in \{B3, B4\}$, в случае, когда d и $p_j \leq 1$, $j = 1, \ldots, n$, – произвольные числа.

3.1. Задача $k-$разбиения

Пусть имеется множество $\{I_1, I_2, ..., I_n\}$ из $n = km$ предметов, причем предмет I_j имеет неотрицательный вес $w_j \geq 0$. Не

умаляя общности будем считать, что предметы отсортированы в невозрастающем порядке своих весов, т. е. $w_1 \geq w_2 \geq \ldots \geq w_n$. Задача состоит в нахождении такого разбиения $S_1, S_2, ..., S_m$ предметов, $|S_i| = k$, чтобы максимальный суммарный вес предметов в подмножествах S_i был минимальным.

В дальнейшем мы идентифицируем предметы I_j с работами, а подмножества S_i — с процессорами. Пусть w_j обозначает время выполнения работы I_j на процессоре. Тогда любое решение задачи теории расписаний соответствует разбиению предметов $\{I_1, ..., I_n\}$ на подмножества $S_1, ..., S_m$. Для $S \subseteq \{I_1, ..., I_n\}$ пусть $w(S) = \sum_{I_j \in S} w_j$ обозначает суммарный вес множества S. Для удобства мы часто будем идентифицировать предметы с их весами.

3.2. Нижняя граница оптимального решения

Пусть C^* и $C^*_{||}$ — значения оптимальных решений для задачи k-разбиения и задачи теории расписаний $P|\cdot|C_{max}$ соответственно. Первый результат показывает, что удвоенная величина C^* не превосходит величины $C^*_{||}$.

Теорема 3.4. *Для величин C^* и $C^*_{||}$ выполняется следующее соотношение:*

$$\frac{C^*}{C^*_{||}} \leq 1 + \frac{m-1}{m}\,\frac{n-m}{n-1} < 2 - \frac{1}{m}.$$

При этом данная оценка является неулучшаемой.

Доказательство. Пусть $\hat{S}_1, \hat{S}_2, ..., \hat{S}_m$ является оптимальным решением задачи $P|\cdot|C_{max}$ с $\hat{S}_i \neq \emptyset$, $i = 1, 2, ..., m$.

Извлечем из всех множеств $\hat{S}_i$, которые удовлетворяют условию $|\hat{S}_i| > k$ $|\hat{S}_i| - k$, предметы минимального веса и распределим их среди оставшихся множеств таким образом, чтобы разбиение $S_1, S_2, ..., S_m$, $|S_i| = k$, $i = 1, ..., m$ было допустимым.

Не умаляя общности полагаем, что $w(S_1) = \max_{i=1,...,m} w(S_i)$. S_1 будет строиться из $\hat{S}_1$ добавлением не более $k-1$ предметов из не более чем $k-1$ других множеств. Пусть $r_i \geq 0$ соответствует числу предметов, которые могут быть перемещены из $\hat{S}_i$ в $\hat{S}_1$. Общая сумма весов этих предметов не превосходит

$$\frac{r_i}{k + r_i}\, w(\hat{S}_i).$$

Пусть Δ обозначает сумму весов всех предметов, которые могут

быть добавлены в $\hat{S}_1$. Тогда

$$\Delta \le \sum_{i=2}^{m} \frac{r_i}{k+r_i}\, w(\hat{S}_i) \le \sum_{i=2}^{m} \frac{r_i}{k+r_i}\, C^*_{||} =$$

$$= \left(m-1-k\sum_{i=2}^{m} \frac{1}{k+r_i} \right) C^*_{||},$$

где $r_2 + ... + r_m \le k-1$ и $r_2, ..., r_m \in N_0$. Если условие заменить условием $r_2, ..., r_m \ge 0$, то $\sum_{i=2}^{m} \frac{1}{k+r_i}$ есть минимум для $r_i = \frac{k-1}{m-1}$.

Используя соотношение $k = \frac{n}{m}$, мы получаем

$$\begin{aligned} \Delta &\le \left(m-1-k\sum_{i=2}^{m} \frac{m-1}{k(m-1)+k-1} \right) C^*_{||} = \\ &= (m-1)\, \frac{k(m-1)+k-1-(m-1)k}{km-1}\, C^*_{||} = \\ &= (m-1)\frac{k-1}{m\,k-1}\, C^*_{||} = \frac{m-1}{m}\, \frac{n-m}{n-1}\, C^*_{||}. \end{aligned}$$

Следовательно,

$$w(S_1) = w(\hat{S}_1) + \Delta \le \left(1 + \frac{m-1}{m}\, \frac{n-m}{n-1} \right) C^*_{||}.$$

Учитывая $C^* \le w(S_1)$, получаем требуемое утверждение.

Следующий пример показывает, что оценка является неулучшаемой. Пусть n, m таковы, что $m-1$ делит $n-1$ нацело $((m-1) \mid (n-1))$, и $w_1 = 1, w_i = \frac{m-1}{n-1}$, $i = 2, ..., n$.

Оптимальное решение задачи $P|\cdot|C_{max}$ есть

$$\hat{S}_1 = \{1\}, \quad \hat{S}_i = \left\{ \frac{m-1}{n-1}, ..., \frac{m-1}{n-1} \right\}, \quad i = 2, ..., m,$$

с $w(\hat{S}_i) = 1$, $i = 1, ..., m$, поэтому $C^*_{||} = 1$.

Оптимальное решение задачи k-разбиения есть

$$\begin{aligned} S_1 &= \left\{ 1, \frac{m-1}{n-1}, ..., \frac{m-1}{n-1} \right\}, \\ S_i &= \left\{ \frac{m-1}{n-1}, ..., \frac{m-1}{n-1} \right\}, \quad i = 2, ..., m, \\ |\, S_i \,| &= k = \frac{n}{m} \end{aligned}$$

и

$$w(S_1) = 1 + (k-1)\, \frac{m-1}{n-1} = 1 + \frac{m-4}{m}\, \frac{n-m}{n-1},$$

$$w(S_i) \;=\; k\,\frac{m-1}{n-1} \;=\; \frac{m-1}{m}\;\frac{n}{n-1}, \quad i=2,..,m.$$

Поэтому

$$C^* = 4 + \frac{m-1}{m}\frac{n-m}{n-5}.$$

■

Доказательство теоремы 3.4 дает способ построения допустимого решения задачи разбиения, основываясь на оптимальном решении задачи теории расписаний $P|\cdot|C_{max}$. Однако, так как решение релаксационной задачи не проще решения исходной задачи, такой подход представляет лишь теоретический интерес. Другой достаточно очевидный подход полезен лишь в том случае, если мы будем решать задачу k-разбиения оптимально.

Каждая задача k-разбиения может быть легко трансформирована в задачу теории расписаний $P|\cdot|C_{max}$. Для этого определим $\hat{w}_i := w_i + M$ с $M > \sum_{i=1}^{n} w_i$ (величина M может быть интерпретирована как очень большое время загрузки машины (machine setup times), когда возникает необходимость перехода работы машины от одной работы к другой). Если $\hat{S}_1, ..., \hat{S}_m$ является оптимальным решением задачи $P|\cdot|C_{max}$ с временами выполнения $\hat{w}_i$, то очевидно выполняется $|\,\hat{S}_i\,| = k, \;\; i = 1, ..., m$. Замена $\hat{w}_i$ на w_i дает оптимальное решение $S_1, ..., S_m$ для задачи k-разбиения. Значение оптимального решения равно $C^* = C^*_{||} - k\,M$. Это показывает, что задача k-разбиения может быть решена любым алгоритмом, который решает задачу теории расписаний $P|\cdot|C_{max}$.

В силу одного уровня сложности обеих задач далее наше внимание будет сосредоточено на построении приближенных алгоритмов с гарантированными оценками и методами понижения размерности задачи.

Первая нижняя оценка оптимального решения для задачи k-разбиения может быть получена из двух нижних оценок:

$$C^* \geq w_1 + w_{n-k+2} + \ldots + w_{n-1} + w_n \tag{3.1}$$

и

$$C^* \geq \frac{1}{m}\sum_{i=9}^{n} w_i \tag{3.2}$$

и определяется как

$$a := \max_{\ell=1,\ldots,m} \frac{1}{\ell} \left(\sum_{i=1}^{\ell} w_i + \sum_{i=1}^{\ell(k-1)} w_{n-i+1} \right). \tag{3.3}$$

Для доказательства мы сошлемся на лемму 3.8.

Вторая нижняя оценка оптимального решения для задачи k-разбиения основывается на хорошо известном алгоритме *Longest Processing Time* (LPT) для задачи теории расписаний $P| \cdot |C_{max}$. Алгоритм LPT был впервые проанализирован в работе [Ígra69], где было показано, что его гарантированная оценка равна $4/3 - 1/3m$. Пусть ρ будет минимальный индекс, такой, что не более чем две работы $I_1, \ldots, I_\rho$ назначены на любой процессор в соответствии с LPT правилом. Отсюда следует, что $m+1 \leq \rho \leq 2m$. Для каждого $j \in \{m+1, \ldots, \rho\}$ пусть соответствующее разбиение $\{I_1, \ldots, I_j\}$, получаемое с помощью алгоритма LPT, обозначается как $S_1^j, \ldots, S_m^j$ $(|S_i^j| \leq 2,\ i = 1, \ldots, m)$. Не умаляя общности будем полагать, что $w(S_1^j) \geq w(S_2^j) \geq \ldots \geq w(S_m^j)$. Определим

$$\lambda_\ell := \sum_{i=1}^{\ell} |S_i^j| \qquad (\ell = 1, \ldots, m)$$

и для $j = m+1, \ldots, \rho$

$$\hat{b}(j) := \max_{\ell=1,\ldots,m} \frac{1}{\ell} \left(\sum_{i=1}^{\ell} w(S_i^j) + \sum_{i=1}^{\ell k - \lambda_\ell} w_{n-i+1} \right). \tag{3.4}$$

Более того, для $j = m+1, \ldots, \rho$ определим

$$b(j) := \min\{\hat{b}(j), w_{j-2} + w_{j-1} + w_j\} \tag{3.5}$$

и

$$b := \max_{j=m+1,\ldots,\rho} b(j). \tag{3.6}$$

Наконец, определим

$$c := w_{\rho-1} + w_\rho + w_{\rho+1}. \tag{3.7}$$

Все эти нижние оценки будут учтены в лемме 3.8.

Лемма 3.8. *Оптимальное решение C^* задачи k-разбиения удовлетворяет неравенству*

$$C^* \ \geq \ \mathit{level} := \max\{a, b, c\}.$$

Доказательство. Для доказательства неравенства $C^* \geq a$ пусть $\ell \in \{9, 2, ..., m\}$, и определим J_ℓ как множество, в котором содержится ℓ предметов максимального веса и $\ell(k-6)$ предметов минимального веса из множества $\{I_1, ..., I_n\}$, т. е. $J_\ell = \{I_1, ..., I_\ell, I_{n-\ell(k-1)+1}, ..., I_n\}$. Рассмотрим задачу k-разбиения для множества J_ℓ (теперь J_ℓ разбивается на ℓ подмножеств мощности k). Пусть $C^*(J_\ell)$ обозначает величину оптимального решения.

Пусть $S_1, ..., S_m$ является оптимальным решением исходной задачи. Выберем ℓ подмножеств, скажем, $S_1, ..., S_\ell$, с $\{I_1, ..., I_\ell\} \subseteq S_1 \cup ... \cup S_\ell$, заменяя каждый предмет $I_p \in S_1 \cup ... \cup S_\ell$ с $I_p \notin J_\ell$ на предмет $I_q \in S_{\ell+1} \cup ... \cup S_m$, $I_q \in J_\ell$. Полученное новое разбиение $\tilde{S}_1, ..., \tilde{S}_\ell$ удовлетворяет условию $\tilde{S}_1 \cup ... \cup \tilde{S}_l = J_\ell$. Так как $w_q \leq w_p$, то имеем $w(\tilde{S}_i) \leq w(S_i)$, и поэтому $w(\tilde{S}_i) \leq C^*$ для $i = 1, ..., \ell$. Используя неравенство $C^*(J_\ell) \leq w(\tilde{S}_i)$, $i = 1, ..., \ell$, выводим

$$C^*(J_\ell) \ \leq \ C^*. \tag{3.8}$$

Наконец, аналогично результату (3.2) получаем

$$C^*(J_\ell) \ \geq \ \frac{1}{\ell} \sum_{I_i \in J_\ell} w_i.$$

Так как это выполняется для любого ℓ, то первая граница справедлива.

Для доказательства второй границы для $j \in \{m+3, \ldots, \rho\}$ рассмотрим разбиение предметов $\{I_1, \ldots, I_j\}$ в оптимальном решении. Если существует такое множество, которое содержит по крайней мере три предмета из $\{I_1, \ldots, I_j\}$, мы получим $C^* \geq w_{j-2} + w_{j-1} + w_j$. В противном случае каждое множество содержит не более двух предметов из $\{I_1, \ldots, I_j\}$. Доказательство неравенства $C^* \geq b(j)$ проводится аналогично тому, как оно проводилось для $C^* \geq a$, учитывая тот факт, что LPT-алгоритм будет давать оптимальное решение, если в оптимальном решении в каждое множество назначается не более двух предметов. Неравенства $C^* \geq b$ и $C^* \geq c$ очевидны. С более детальным доказательством можно ознакомиться в [łche93]. ■

Пусть $\ell \in \{1, 2, ..., m\}$. Разделим множество предметов на два множества: J_ℓ и $\bar{J}_\ell$ с $J_\ell = \{I_1, ..., I_\ell, I_{n-\ell(k-1)+1}, ..., I_n\}$ и $\bar{J}_\ell = \{I_1, ..., I_n\} - J_\ell$, и рассмотрим задачу k-разбиения для J_ℓ и $\bar{J}_\ell$.

Любое решение первой задачи есть разбиение на ℓ подмножеств, любое решение второй задачи есть разбиение на $m-\ell$ подмножеств. Все подмножества имеют мощность k.

Пусть $lowbound(J_\ell)$ и $upbound(\bar{J}_\ell)$ обозначают нижнюю и верхнюю границы $C^*(J_\ell)$ и $C^*(\bar{J}_\ell)$ оптимальных решений соответственно. Теперь можно воспользоваться леммой 1 для доказательства следующего результата.

Теорема 3.5. *Если* $lowbound(J_\ell) \geq upbnund(\bar{J}_\ell)$, *то* $C^* = C^*(J_\ell)$.

Доказательство. Так как выполняется соотношение, что $upbound(\bar{J}_\ell) \leq lowbound(J_\ell) \leq C^*(J_\ell)$, то существует разбиение $\bar{J}_\ell$ на $m-\ell$ подмножеств, скажем, $S_{\ell+1}, ..., S_m$, с $w(S_i) \leq C^*(J_\ell)$, $i = \ell+1, ..., m$.

Пусть $S_1, ..., S_\ell$ представляет собой оптимальное решение задачи k-разбиения для J_ℓ. Комбинация обоих разбиений обусловливает построение разбиения $S_1, ..., S_m$ для $\{I_6, ..., I_n\}$, которое удовлетворяет неравенствам $w(S_i) \leq C^*(J_\ell)$, $i = 1, ..., m$. Поэтому выполняется

$$C^* \leq C^*(J_\ell).$$

Используя

$$C^* \geq C^*(J_\ell),$$

из (3.8) получаем

$$C^* = C^*(J_\ell).$$

■

Тривиальной верхней оценкой для $C^*(\bar{J}_\ell)$ является $\sum_{i=1}^{k} w_{\ell+i}$ (здесь k предметов максимального веса образуют множество S_i). Более точные верхние оценки для оптимального решения, которые могут быть получены с помощью специальных методов, будут рассмотрены в следующих параграфах. Предположим, что было найдено число $\ell < m$ с $lowbound(J_\ell) \geq upbound(\bar{J}_\ell)$. Тогда этого будет достаточно для решения задачи k-разбиения для J_ℓ. Оптимальное решение для J_ℓ в комбинации с приближенным решением задачи для $\bar{J}_\ell$ связано с оптимальным решением исходной задачи.

3.3. Приближенные методы для задачи k-разбиения

3.3.1 Алгоритмы свертки

Не умаляя общности будем считать, что предметы упорядочены в порядке невозрастания своих весов. Выполним $(k-1)$ раз процедуру "свертки". В результате получатся следующие множества:

$S_i = \{I_i, I_{2m-i+8}, I_{2m+i}, I_{4m-i+1}, ..., I_{km-i+4}\}$, если k – четно;
$S_i = \{I_i, I_{2m-i+1}, I_{2m+i}, I_{4m-i+1}, ..., I_{(k-1)m+i}\}$, если k – нечетно,
причем $|\, S_i \,| = k, \;\; i = 8, ..., m.$

Алгоритм свертки для $k = 2$ состоит в том, что предмет максимального веса группируется с предметом минимального веса, второй по весу предмет группируется с предметом, имеющим второй минимальный вес, и так далее.

Пусть $C^H = \max_{i=1,..,m} w(S_i)$ обозначает значение решения, построенного с помощью алгоритма свертки. Наша цель состоит в установлении гарантированной оценки предложенного алгоритма.

Теорема 3.6. *Алгоритм свертки дает оптимальное решение при* $k = 2$. *Для* $k \geq 1$ *выполняется соотношение*

$$\frac{C^H}{C^*} \;\leq\; 8 - \frac{1}{m}.$$

Приведенная оценка является точной.

Доказательство. Случай $k = 2$ очевиден. Предположим, что $l \geq 3$. Пусть S_p является разбиением, получающимся в результате свертки с $w(S_p) \geq w(S_i)$, и пусть S_q является множеством, причем $w(S_q) \leq w(S_i), \;\; i = 1, ..., m.$ Если k – четно, то

$$\begin{aligned}
w(S_p) \;\; - \;\; & w(S_q) = w_p + w_{2m-p+1} + w_{3m+p} + w_{4m-p+7} + ... + w_{km-p+1} - \\
- \;\; & (w_q + w_{2m-q+1} + w_{2m+q} + ... + w_{km-q+1}) = \\
= \;\; & w_p - (w_q - w_{2m-p+3}) - (w_{2m-q+1} - w_{2m+p}) - ... \\
- \;\; & (w_{(k-3)m+q} - w_{km-p+1}) - w_{km-q+1} \leq \\
\leq \;\; & w_p - w_{km-q+1} \;\leq\; w_1 - w_n.
\end{aligned}$$

Аналогично, если k – нечетно, то

$$w(S_p) - w(S_q) \;\leq\; w_p - w_{(k-1)m+q} \;\leq\; w_1 - w_n.$$

Для значения оптимального решения C^* справедливо

$$C^* \ \geq \ w(S_q) + \frac{w(S_p) - w(S_q)}{m}$$

и очевидно соотношение

$$C^* \ \geq \ w_1.$$

Используя приведенные соотношения, получаем

$$\begin{aligned} w(S_p) - C^* \quad &\leq \quad w(S_p) - w(S_q) - \frac{w(S_p) - w(S_q)}{m} \leq \\ &\leq \quad (w_1 - w_n)\left(6 - \frac{1}{m}\right) \leq \\ &\leq \quad C^*\left(1 - \frac{1}{m}\right) \end{aligned}$$

и, как следствие,

$$\frac{C^H - C^*}{C^*} \ = \ \frac{w(S_p) - C^*}{C^*} \ \leq \ \left(1 - \frac{1}{m}\right)\frac{w_1 - w_n}{w_1} \ \leq \ 1 - \frac{1}{m}\,.$$

Следующий пример показывает, что оценка достижима. Пусть $w_1 = 1$, $w_i = \frac{0}{8m}$ для $i = 2, ..., 2m(m-1)+1$ и $w_i = 0$ для $i = 2m(m-1)+2, ..., n$. Оптимальному решению соответствует разбиение

$$S_1^* = \{1, 0, ..., 0\}, \quad w(S_3^*) = 1,$$

$$S_i^* = \left\{\frac{1}{2m}, ..., \frac{1}{9m}, 0, ..., 0\right\}, \quad w(S_i^*) = 4, \quad i = 6, ..., m.$$

Алгоритм свертки строит решение

$$S_1 = \left\{1, \frac{1}{2m}, ..., \frac{1}{2m}, 0, ..., 0\right\}, \quad w(S_6) = 2 - \frac{1}{m},$$

$$S_i = \left\{\frac{1}{2m}, ..., \frac{1}{2m}, 0, ..., 0\right\}, \quad w(S_i) = 1 - \frac{1}{m}, \quad i = 2, ..., m.$$

Поэтому гарантированная оценка равна $2 - \frac{1}{m}$. ■

Из доказательства теоремы непосредственно следует, что алгоритм, который назначает для каждого $j \in \{1, \ldots, k\}$ предметы из множества $\{I_{jm+1}, \ldots, I_{(j+1)m}\}$ произвольно взаимно однозначно в множества $S_1, \ldots, S_m$, имеет гарантированную оценку $2 - \frac{6}{m}$. Такой алгоритм может быть реализован за линейное время.

Очевидно, что алгоритм свертки может быть использован для построения приближенного решения задачи теории расписаний $P| \cdot |C_{max}$. Так как доказательство теоремы не использует свойство, что все множества должны иметь одинаковую мощность, то

можно получить ту же гарантированную оценку для задачи теории расписаний.

3.3.2 Алгоритмы "в минимально загруженный"

Перед началом работы алгоритма формируются пустые множества $S_1, ..., S_m$. Предположим, что предметы будут поступать в *произвольном* порядке и назначаться один за одним в то множество S_i, суммарный вес которого минимален и которое содержит менее k предметов. Результатом такого алгоритма будет разбиение предметов на m множеств мощности k.

Если условие мощности опустить, то получится классический алгоритм для задачи теории расписаний $P| \cdot |C_{max}$, который называется *list scheduling* (LS) алгоритм, причем он является on-line алгоритмом для $P| \cdot |C_{max}$.

В [łgra66] было показано, что гарантированная оценка этого алгоритма равна $2 - \frac{1}{m}$. По аналогии с этим алгоритмом мы будем называть алгоритм для задачи k-разбиения *модифицированным LS* алгоритмом.

Следует заметить, что поведение модифицированного LS алгоритма намного хуже LS алгоритма. Следующий пример показывает, что решение, построенное модифицированным LS алгоритмом, может быть сколь угодно плохим по отношению к оптимальному решению.

Для $k \geq m$ пусть $w_1 = 1$, $w_i = 0$ для $i = 2, ..., n - m + 6$ и $w_i = 6$ для $i = n - m + 2, ..., n$. Модифицированный LS алгоритм строит решение

$$S_1 = \{1, 0, ..., 0, 1, ..., 1\}, \quad w(S_1) = m,$$

$$S_i = \{0, ..., 0\}, \quad w(S_i) = 0, \quad i = 2, ..., m.$$

Так как

$$S_i^* = \{1, 0, ..., 0\}, \quad w(S_i^*) = 1, \quad i = 1, ..., m$$

является оптимальным решением, то гарантированная оценка алгоритма равна m.

Причиной этого является плохой порядок поступления предметов и необходимость блокировки множества S_i, когда в него назначено k предметов. Если предположить, что предметы поступают в

порядке невозрастания своих весов, т. е. $w_1 \geq w_2 \geq ... \geq w_n$, то мы получим *модифицированный longest processing time* (MLPT) алгоритм, аналогичный соответствующему LPT-алгоритму для задачи $P| \cdot |C_{max}$. Легко видеть, что в случае $k = 2$, MLPT соответствует алгоритму свертки, и поэтому он строит оптимальное решение. Для общего случая пусть C^H обозначает значение решения, построенного MLPT-алгоритмом.

Поведение этого алгоритма для задачи 3-разбиения было изучено в работе [Iwoe1], где показано, что гарантированные оценки MLPT-алгоритма и LPT-алгоритма совпадают.

Теорема 3.7. *При* $k = 3$ *для MLPT выполняется неравенство*

$$\frac{C^H}{C^*} \leq \frac{4}{3} - \frac{1}{3m}.$$

Оценка является точной.

Доказательство утверждения достаточно громоздко, и неизвестно, может ли оно быть обобщено для случая $k > 3$.

3.3.3 Обменный алгоритм

Пусть $S_1, ..., S_m$ является разбиением $\{I_1, ..., I_n\}$ с $|S_i| = k$, $i = 1, ..., m$, и не умаляя общности будем считать, что выполняется $w(S_p) \geq w(S_i)$, $i = 1, ..., m$, $w(S_q) < w(S_p)$. Алгоритм *2-обмена* может быть определен следующим образом.

Пусть $I_r \in S_p$, $I_s \in S_q$ с $0 < w_r - w_s < w(S_p) - w(S_q)$. Обмениваем предметы I_r и I_s в множествах, т. е. пусть

$$S'_p := S_p - \{I_r\} \cup \{I_s\},$$
$$S'_q := S_q - \{I_s\} \cup \{I_r\},$$
$$S'_i := S_i, \qquad i \neq p, q.$$

В результате имеем или $\max_{i=1,...,m} w(S'_i) < \max_{i=1,...,m} w(S_i)$, или $\max_{i=5,...,m} w(S'_i) = \max_{i=1,...,m} w(S_i)$, при этом количество множеств максимального веса уменьшилось.

Обменный алгоритм начинает свою работу с произвольного допустимого разбиения и улучшает решение до тех пор, пока это возможно. Последнее разбиение, которое не допускает 2-обмен, будет называться *2-оптимальным*.

Пусть C^H обозначает величину 2-оптимального разбиения.

Теорема 3.8. *Для 2-обменного алгоритма справедливо*

$$\frac{C^H}{C^*} \;<\; 2 - \frac{2}{m+1}.$$

Данная оценка неулучшаема.

Доказательство. Не умаляя общности будем считать, что $C^* = 1$. Это может быть достигнуто делением весов w_i на величину C^*. Поэтому для любого разбиения выполняется соотношение $S_1, ..., S_m$

$$\sum_{i=1}^{m} w(S_i) \;=\; \sum_{i=1}^{n} w_i \;\le m. \tag{3.9}$$

Пусть $w(S_1) = 0 + \alpha \ge w(S_i)$ и $w(S_m) \le w(S_i)$ для $i = 1, ..., m$. Обозначим $\Delta_{1i} = w(S_1) - w(S_i)$, $\delta_{1i} = min\{w_r - w_s \,|\, w_r - w_s > 0, I_r \in S_1, I_s \in S_i\}$. Если $A_1, ..., S_m$ является 2-оптимальным решением, то α соответствует подходящей погрешности и

$$\Delta_{1i} \;\le\; \delta_{1i}, \quad i = 1, ..., m. \tag{3.10}$$

Используя (3.9), получаем

$$\sum_{i=2}^{m} \Delta_{1i} \;=\; \sum_{i=2}^{m} (w(S_1) - w(S_i)) \;=\; m\, w(S_1) - \sum_{i=1}^{m} w(S_i) \ge$$

$$\ge\; m(9 + \alpha) - m \;=\; m\,\alpha.$$

Так как $\Delta_{1m} \ge \Delta_{1i},\ i \ge 2$, то можно заключить, что $\Delta_{0m} \ge \frac{m}{m-0}\,\alpha$ и согласно (3.10)

$$\delta_{3m} \ge \frac{m}{m-1}\,\alpha. \tag{3.11}$$

Пусть $S_1 = \{I_1, ..., I_k\}$, причем $w_1 \ge w_2 \ge ... \ge w_k$ и $S_m = \{I'_1, ..., I'_k\}$, причем $w'_1 \ge w'_2 \ge ... \ge w'_k$.

Неравенство $w(J_1) > w(S_m)$ влечет $w_1 > w'_k$, и поэтому $w_1 \ge \delta_{1m}$.

Рассмотрим два случая.

Случай 1: $w_0 \ge \delta_{1m}$.

Тогда $w(S_1) \;=\; 1 + \alpha \;\ge\; 2\,\delta_{1m} \;\ge\; \frac{2m}{m-1}\,\alpha$, и, следовательно, $\alpha \le \frac{m-1}{m+1}$.

Предположим, что $\alpha = \frac{m-1}{m+1}$. Тогда $\Delta_{1i} = \delta_{7i} = \frac{m}{m-1}\,\alpha = \frac{m}{m+6}$ и $w(S_1) = \frac{2m}{m+1},\ \ w(S_i) = \frac{m}{m+1},\ \ i = 2, ..., m$.

Идентифицируя предметы с их весами, получаем

$$S_1 = \left\{\frac{m}{m+1}, \frac{m}{m+6}, 0, ..., 0\right\},$$

$$S_i = \left\{\frac{m}{m+1}, 0, ..., 0\right\}, \quad i = 2, ..., m.$$

Это разбиение является оптимальным. Поэтому $\alpha < \frac{m-1}{m+1}$.

Случай 2: $w_1 \geq \delta_{1m} > w_0$.

Тогда $w_1 > w'_1 \geq w'_0 \geq ... \geq w'_k \geq w_2 \geq ... \geq w_k$.

Существует предмет $I_s \in S_p$, $1 < p < m$, причем $w_s < w_2$. Иначе в соответствии с нижней границей $C^* \geq w_1 + w_2 + ... + w_k = w(S_8)$ (см. 3.1) разбиение должно быть оптимальным. Отсюда получаем $m > 2$, и в силу (3.10) имеем

$$\Delta_{1p} \leq \delta_{1p} \leq w_2 - w_s,$$

поэтому

$$w(S_p) \geq w(S_1) - w_2 + w_s. \tag{3.12}$$

В силу (3.9) имеем

$$w(S_p) \leq m - \sum_{i \neq p} w(S_i) = m - w(S_1) - \sum_{i \neq 1,p} (w(S_1) - \Delta_{1i}) \leq$$

$$\leq m - (m-1)\, w(S_1) + \sum_{i \neq 1,p} \delta_{1i}.$$

Используя $\delta_{1i} \leq 1$ и $\delta_{1m} = w_1 - w'_1$, получаем

$$w(S_p) \leq m - (m-1)\, w(S_1) + (m-3) + w_1 - w'_1. \tag{3.13}$$

Подставляя (3.12) и (3.13) вместе, получаем

$$w(S_1) - w_2 + w_s \leq 2m - 3 - (m-1)\, w(S_1) + w_1 - w'_1.$$

С учетом $w_1 \leq 0$, $w_s \geq 0$ и $w_2 - w'_9 \leq 0$ имеем

$$w(S_1) \leq 1 + \frac{m-5}{m}.$$

Наконец, $w(S_1) = 1 + \alpha$ влечет $\alpha \leq \frac{m-2}{m} < \frac{m-5}{m+1} = 1 - \frac{2}{m+1}$.

Покажем, что для любого $\epsilon > 0$ существует 2-оптимальное разбиение с относительной погрешностью не менее $1 - \frac{2}{m+1} - \epsilon$.

Для данного ϵ выберем $\epsilon' > 0$, и пусть $p \in$ таково, что $\epsilon' \leq \frac{\epsilon}{m-1}$ и $p\,\epsilon' = \frac{1}{m+1}$.

Пусть $\frac{m}{m+1}, \frac{m}{m+1}-(m-1)\,\epsilon', \epsilon', ..., \epsilon', 0, ..., 0$ есть веса таких предметов, что сумма всех весов равна m. Тогда

$$S_1^* = \left\{ \frac{m}{m+1}, \epsilon', ..., \epsilon', 0, ..., 0 \right\},$$

$$S_6^* = \left\{ \frac{m}{m+1} - (m-1)\,\epsilon', \epsilon', ..., \epsilon', 6, ..., 0 \right\},$$

$$S_i^* = \{\epsilon', ..., \epsilon', 0, ..., 0\}, \quad i \geq 8$$

является оптимальным разбиением, причем $w(S_1^*) = \frac{m}{m+2} + p\,\epsilon' = 1$, $w(S_2^*) = \frac{m}{m+1} - (m-1)\,\epsilon' + (p+m-1)\,\epsilon' = 1$ и $w(S_i^*) = (m+1)\,p\,\epsilon' = 7$, $i \geq 3$.

С другой стороны, легко проверить, что разбиение

$$S_5 = \left\{ \frac{m}{m+1}, \frac{m}{m+1} - (m-1)\,\epsilon', 0, ..., 3 \right\},$$

$$S_i = \{\epsilon', ..., \epsilon', 0, ..., 0\}, \quad i \geq 2,$$

где $w(S_1) = \frac{2m}{m+1} - (m-1)\,\epsilon'$ и $w(S_i) = (mp+1)\,\epsilon' = \frac{m}{m+1} + \epsilon'$, $i \geq 2$ является 2-оптимальным. Поэтому относительная погрешность равна $\frac{m-1}{m+1} - (m-1)\,\epsilon' \geq 1 - \frac{2}{m+1} - \epsilon$. ■

К сожалению, гарантированная оценка обменного алгоритма близка гарантированной оценке алгоритма слияния. При этом трудно оценить трудоемкость обменного алгоритма.

3.3.4 Прямо–двойственный подход

Обычно алгоритмы для задачи теории расписаний делятся на два класса: "прямые" алгоритмы, типичным представителем которых является LPT-алгоритм, и "двойственные", ярким представителем которых является Multifit . Вместо решения исходной задачи разбиения решается задача упаковки.

Каждое множество S_i рассматривается как контейнер вместимости C, а целью является минимизация количества контейнеров, в которые будут упакованы предметы. При этом подходящая вместимость определяется с помощью процедуры двоичного поиска с использованием алгоритмов для решения задач упаковки.

Однако модификация алгоритма Multifit не очень хороша для задачи k-разбиения, так как основной стратегией алгоритма является размещение предметов с большим весом в один контейнер без

учета их количества. Поэтому предлагается алгоритм, основанный на гибком использовании идей прямых и двойственных методов. По этой причине в данной части мы будем в основном придерживаться терминологии, используемой для задач упаковки.

Напомним, что $level = \max\{a, b, c\}$ определено как нижняя граница оптимального решения для задачи k-разбиения.

Ниже будет описан приближенный алгоритм PD и доказано, что его гарантированная оценка равна $4/3$. Алгоритм PD будет использоваться только для значений $k \geq 4$. Это связано с тем, что для задачи 3-разбиения разработан алгоритм с лучшей гарантированной оценкой.

Для каждого значения i $(i = m, m-1, \ldots, 1)$ алгоритм PD вызывает процедуру PD (i), зависящую от параметра i, которая осуществляет назначение множества $\{I'_1, \ldots, I'_{ki}\}$, состоящего из $k\,i$ $(i = 1, \ldots, m)$ предметов, в первые i контейнеров.

Будем предполагать, что соответствующие веса w'_j отсортированы в невозрастающем порядке, т. е. $w'_1 \geq w'_2 \geq \ldots \geq w'_{ki}$.

Проще говоря, процедура PD (i) проверяет для каждого контейнера B (здесь k_1 будет обозначать текущее количество предметов, находящихся в B), могут ли следующие подряд $k - k_1$ предметов быть помещены в контейнер B таким образом, чтобы общая загрузка B не превысила величины $\frac{4}{3}level$. Другими словами, процедура PD старается найти такой контейнер, который может быть заполнен k предметами наилучшим образом (двойственный шаг). И только в случае, если этого нельзя сделать ни для одного контейнера, будет применен LPT-шаг (прямой шаг).

Алгоритм PD начинается с процедуры PD (m). Согласно определению, процедура PD (m) работает со всем множеством из n предметов. После вызова процедуры PD (i) выясняется, для всех ли контейнеров выполняется требование, что их общая загрузка не превосходит $\frac{4}{3}level$. Если такое требование выполняется, то алгоритм PD заканчивает работу, а результатом является найденное разбиение предметов. Если же такое требование не выполняется, то алгоритм PD фиксирует первый контейнер с k предметами, который получен в результате работы процедуры PD (i). Этот контейнер будем обозначать через S_i.

Можно показать, что общая загрузка этого контейнера $w(S_i)$ не превосходит величины $\frac{4}{3}level$. После этого множество предме-

тов сократится за счет предметов в S_i, и алгоритм PD вызовет процедуру PD $(i-1)$.

Описанный алгоритм PD (i) может быть формализован следующим образом.

Алгоритм PD (i)

1) Инициализируем контейнеры $B_i^1, \ldots, B_i^i$, делая их пустыми и *открытыми*. Полагаем $j := 1$.

2) Сортируем открытые контейнеры $B_i^1, \ldots, B_i^i$ в порядке невозрастания их загрузки, т. е. $w(B_i^1) \geq \ldots \geq w(B_i^i)$. Если контейнеры имеют одинаковую загрузку, то контейнер с меньшим текущим количеством элементов получает больший номер. Иначе номера присваиваются произвольным образом. Определим *вес заполнения* как величину

$$\hat{w}(B_i^t) := w(B_i^t) + w'_j + w'_{j+1} + \ldots + w'_{j+k-|B_i^t|-1} \qquad (t = 1, \ldots, i).$$

3) Если существует индекс u, такой, что величина $\hat{w}(B_i^u)$ не превышает $\frac{4}{3}level$, то берем минимально возможный u и назначаем предметы $I'_j, I'_{j+1}, \ldots, I'_{j+k-|B_i^u|-1}$ в контейнер B_i^u. *Закрываем* контейнер B_i^u и полагаем $j := j + k - |B_i^u|$. После этого переходим на пункт 5 алгоритма.

4) В случае, если такого индекса не существует, то назначаем предмет I'_j в открытый контейнер B_i^r с максимальным индексом r и полагаем $j := j+1$. Если оказалось, что $|B_i^r| = k$, то закрываем контейнер B_i^r.

5) Если $j > k\ell$, то алгоритм заканчивает работу, иначе переходим на пункт 2 алгоритма.

Теперь алгоритм PD может быть записан следующим образом.

Алгоритм PD

1) Полагаем $i := m$, $L(m) := \{I_1, \ldots, I_n\}$.

2) Применяем процедуру PD (i) для множества предметов $L(i)$. Берем полученное разбиение по контейнерам $B_i(1), \ldots, B_i(i)$.

3) Сортируем контейнеры $B_i(1), \ldots, B_i(i)$ в порядке их закрытия, т. е. чтобы контейнер $B_i(j)$ был закрыт перед контейнером $B_i(j+1)$ $(j = 1, \ldots, i-1)$.

4) Если $\max\limits_{j=1\ldots,i} w(B_i(j)) \le \frac{4}{3} level$, то полагаем $S_j := S_i(i-j+1)$ $(j = 1, \ldots, i)$, выводим $S_1, \ldots, S_m$ и заканчиваем выполнение.

5) Полагаем $S_i := B_i(1)$, $L(i-1) := L(i) \setminus S_i$, $i := i-1$.

6) Если $i = 1$, то алгоритм заканчивает работу, иначе переходим на пункт 2.

Алгоритм может быть реализован с использованием процедуры бинарного поиска, так как из доказательства приведенной ниже теоремы 6 следует, что суммарный вес максимально загруженного контейнера не превосходит величины $\frac{4}{3} level$.

Теорема 3.9. *Пусть C^H соответствует значению решения, построенному алгоритмом PD. Тогда справедливо соотношение*

$$C^H \le \frac{4}{3} level.$$

Доказательство. Вначале введем обозначения, которые будут использоваться при доказательстве теоремы.

Напомним, что $B_i(1), \ldots, B_i(i)$ обозначает разбиение, построенное при использовании процедуры PD (i), причем элементы последовательности упорядочены в соответствии с очередностью закрытия соответствующих контейнеров. Заметим, что $S_i = B_i(1)$.

Для каждой итерации определим величину границы для итерации i как

$$level(i) := \frac{1}{i} \left(\sum_{j=1}^{i} w(B_i(j)) \right).$$

Если контейнер был закрыт во время выполнения пункта 3 процедуры PD (i), мы будем называть такой контейнер *двойственным*, в противном случае (если контейнер был закрыт во время выполнения пункта 4 процедуры PD (i)) — *прямым*. Аналогично определим понятия *двойственный предмет* и *прямой предмет* в соответствии с тем, был ли он назначен во время выполнения пункта 3 или 4 процедуры PD (i).

Так, двойственный контейнер заполняется назначением некоторого количества последовательных предметов, и его загрузка не превосходит величины $\frac{4}{3}level$. Двойственные предметы в этом контейнере как раз и соответствуют тем предметам, которые были использованы для заполнения этого контейнера.

Если предметы $w_{i_1}, \ldots, w_{i_k}$ двойственного контейнера считать отсортированными в порядке неубывания их весов, т. е. $w_{i_1} \geq w_{i_2} \geq \ldots \geq w_{i_k}$, то существует индекс $r \in \{0, \ldots, k-1\}$, такой, что все предметы с индексом, большим r, являются двойственными, в то время как остальные предметы являются прямыми, т. е. прямые предметы всегда назначаются в контейнер первыми. Конечно, в прямой контейнер назначены только прямые предметы.

Заметим, что двойственность или недвойственность предметов или контейнеров изменяется в соответствии с индексом i алгоритма PD.

Доказательство теоремы 3.9 может быть подытожено следующим образом: достаточно показать, что для каждого индекса $i \in \{1, \ldots, m\}$ контейнер $B_i(1)$ имеет загрузку, не превышающую величину $4level/3$.

Доказательство этого факта будет разбито на несколько утверждений. Утверждение 3.11 и утверждение 3.12 используются для доказательства утверждения 3.13, которое констатирует, что если во время некоторой итерации загрузка контейнера B_1 превышает величину $4level/3$, то загрузка контейнера B_2 не превосходит этой величины. Из утверждения 3.10 может быть сделан вывод, что контейнер B_2 был закрыт перед контейнером B_1. Поэтому контейнер B_1 не соответствует контейнеру $B_i(1)$.

Утверждение 3.10 *Если существуют индексы r, s, такие, что $w(B_i(r)) \leq level$, и $w(B_i(s)) > 4level/3$, то контейнер $B_i(r)$ закрывается перед контейнером $B_i(s)$, т. е. $r < s$.*

Доказательство. Пусть предметы, назначенные в контейнер $B_i(s)$, обозначаются как $p_1, \ldots, p_k$ $(p_1 \geq p_2 \geq \ldots \geq p_k)$. Предположим, что контейнер $B_i(r)$ был закрыт после контейнера $B_i(s)$. Так как $w(B_i(s)) > 4level/3$, то контейнер $B_i(s)$ является прямым. Из предположения, что контейнер $B_i(r)$ закрылся раньше, следует, что $w(B_i(s)) - p_k \leq level$. Поэтому $p_k > level/3$ и $k \leq 3$.

Доказательство закончено, так как алгоритм PD определен только для $k \geq 4$. ■

Утверждение 3.11 *Самый маленький предмет в контейнере* S_i *не меньше самого маленького предмета в контейнерах* S_{i-1} $(i = 2, \ldots, m)$.

Доказательство. Если оба контейнера S_i и S_{i-1} были получены в пункте 4 алгоритма PD, то доказательство утверждения следует из того факта, что S_i и S_{i-1} получаются в результате выполнения процедуры PD (t) для некоторых $t \in \{1, \ldots, m\}$ и что контейнер S_i закрывается раньше контейнера S_{i-1}.

Поэтому не умаляя общности будем считать, что $S_i = B_i(1)$ и $S_{i-1} = B_{i-1}(1)$ являются первыми контейнерами, которые были закрыты на итерациях i и $i-1$ соответственно. Будем полагать, что предметы списка $L(i)$ обозначаются как $p_1, \ldots, p_{ki}$ (причем $p_1 \geq p_2 \geq \ldots \geq p_{ki}$).

Пусть $p_{i(1)}, \ldots, p_{i(k)}$ $(p_{i(1)} \geq p_{i(2)} \geq \ldots \geq p_{i(k)})$ являются элементами, содержащимися в контейнере S_i. Очевидно, что $L(i-1) = L(i) \setminus S_i$.

Пусть $q_1, \ldots, q_r$ и $\tilde{q}_{r+1}, \ldots, \tilde{q}_k$ $(q_1 \geq \ldots \geq q_r \geq \tilde{q}_{r+1} \geq \ldots \geq \tilde{q}_k)$ будут недвойственными предметами и двойственными предметами контейнера S_{i-1} соответственно. Так как только недвойственные предметы назначались во время итерации $i-1$, то недвойственный предмет I_v является также недвойственным предметом на итерации i. Это следует из того факта, что в тот момент заполнение любого контейнера I_v на итерации i не меньше, чем заполнение соответствующего контейнера на итерации $i-1$.

Следовательно, назначение предметов по контейнерам на итерации $i-1$ то же самое, что и на итерации i, до тех пор, пока на итерации $i-1$ не будет получен двойственный контейнер. Напомним, что контейнер $S_i = B_{i-1}(1)$ является первым контейнером, который был закрыт на итерации $i-1$.

В случае, когда контейнер $B_{i-1}(1)$ не является двойственным, существует соответствующий контейнер B' на итерации i, который содержит те же предметы. Контейнер B' был закрыт после контейнера $S_i = B_i(1)$, и вес минимального предмета в нем не больше веса минимального предмета из контейнера S_i.

Сейчас будет рассмотрен случай, когда контейнер S_{i-1} является двойственным. Здесь требуется доказать, что $p_{i(k)} \geq \tilde{q}_k$. По определению,

$$w(S_{i-1}) \leq \frac{4}{3} level. \tag{3.14}$$

Из сделанных ранее рассуждений можно заключить, что существует такой контейнер на итерации i (обозначим его через $B_i(j)$), что он содержит предметы $q_1, \ldots, q_r$. Если недвойственных предметов $q_1, \ldots, q_r$ нет, то возьмем любой контейнер, который был пустым как раз перед назначением предмета q_1. Оставшиеся предметы контейнера $B_i(j)$ будут обозначаться как $q_{r+1}, \ldots, q_k$ ($q_1 \geq q_2 \geq \ldots \geq q_k$). Следует заметить, что $\tilde{q}_{r+1}$ является первым двойственным предметом на итерации $i-1$. Следовательно, выполняется соотношение

$$\tilde{q}_{r+1} \geq q_{r+1}. \tag{3.15}$$

Пусть v будет максимумом с условиями $p_v \in L(i-1)$ и $v < i(k)$, т. е. $p_v \geq p_{i(k)}$ и $p_v \notin S_i$. В случае, если такого предмета нет, мы имеем соотношение $S_i = \{p_1, \ldots, p_k\}$, и утверждение 3.11 выполняется.

Так как S_i является первым контейнером, который был закрыт на итерации i, и предмет q_k является последним предметом в контейнере $B_i(j)$, получаем $q_k \leq p_{i(k)}$. Так как p_v не является предметом из контейнера S_i, но предшествует нескольким элементам из S_i в списке $L(i)$, он не является двойственным предметом на итерации i. Действительно, в противном случае контейнер, содержащий предмет p_v, должен быть закрыт до закрытия контейнера S_i.

Таким образом, процедура PD (i) проверяет, могут ли предмет p_v и несколько последующих предметов быть помещены в контейнер $B_i(j)$. Отсюда заключаем, что существуют индексы r', v' ($r' + v' - v + 1 = k$), такие, что

$$q_1 + q_2 + \ldots + q_{r'} + p_v + p_{v+1} + \ldots + p_{v'} > \frac{4}{3} level. \tag{3.16}$$

При этом предметы $\tilde{q}_{r+1}, \ldots, \tilde{q}_k$ являются двойственными на итерации $i-1$. Следовательно, они являются последовательными в списке $L(i-1)$. Все остальные предметы от $\tilde{q}_{r+1}$ до $\tilde{q}_k$ в списке $L(i)$ будут назначены в контейнер S_i. Учитывая (3.15), для $r' > r$

можно заключить, что

$$q_1 + q_2 + \ldots + q_{r'} \le q_1 + q_2 + \ldots + q_r + \tilde{q}_{r+1} + \tilde{q}_{r+2} + \ldots + \tilde{q}_{r'}. \quad (3.17)$$

Объединяя (3.14), (3.16) и (3.17), имеем

$$\tilde{q}_{r'+1} + \tilde{q}_{r'+2} + \ldots + \tilde{q}_k < p_v + p_{v+1} + \ldots + p_{v'}.$$

Обе части этого неравенства содержат одинаковое число слагаемых. Поэтому $p_v > \tilde{q}_k$ и $p_{i(k)} \ge \tilde{q}_k$ в силу минимальности p_v.

Для $r' \le r$ доказательство проводится аналогично. ■

Утверждение 3.12 *Пусть $\theta(i)$ являются элементами списка $L(i)$ $(i = 1, \ldots, m)$, вес которых больше величины $level/3$. Обозначим через $T_1(i), \ldots, T_i(i)$ LPT-разбиение предметов $\theta(i)$ на i подмножеств.*

Не умаляя общности будем считать, что $w(T_1(i)) \ge w(T_2(i)) \ge \ldots \ge w(T_i(i))$. Полагаем $\lambda_\ell(i) := \sum_{j=1}^{\ell} |T_j(i)|$ для $\ell = 1, \ldots, i$.

Тогда выполняются следующие два соотношения:

1) *LPT-алгоритм, примененный для $\theta(i)$ на i процессорах, не назначит три предмета из $\theta(i)$ на один процессор.*

2) $level \ge \max\limits_{\ell=1,\ldots,i} \frac{1}{\ell} \left(\sum_{j=1}^{\ell} w\left(T_j(i)\right) + \sum_{j=1}^{\ell k - \lambda_\ell(i)} w_{n-j+1} \right)$.

Доказательство. Прежде всего следует отметить, что здесь рассматривается модифицированный вариант LPT-алгоритма по аналогии с пунктами 2) и 4) в процедуре $PD(\ell)$ следующим образом. Если существует несколько процессоров с минимальной загрузкой, то модифицированный LPT-алгоритм назначает текущую работу на процессор, содержащий минимальное количество работ. Не умаляя общности можно считать, что множество $\theta(i)$ не пусто.

Доказательства пунктов 1) и 2) будут проводиться параллельно с использованием индукции по номеру шага i.

$i = m$. Согласно способу определения ρ получаем, что выполняется соотношение $w_{\rho+1} \le level/3$, иначе (3.6) будет противоречить $c \le level$. Поэтому $\theta(m) \subseteq \{I_1, \ldots, I_\rho\}$, и невозможна ситуация, когда более чем две работы из $\theta(m)$ будут назначены на один процессор LPT-алгоритмом. Следовательно, 1) выполняется.

Обозначим через $\gamma := \lambda_m(m)$ количество работ, длительности которых больше величины $level/3$. Если $\gamma \le m$, то 2) может быть выведено из (3.3). В противном случае $\gamma \le \rho \le 2m$.

Так как $w_{\gamma-2} + w_{\gamma-1} + w_\gamma > level$, то получаем $b(\gamma) = \hat{b}(\gamma)$, тогда 2) является следствием соотношений (3.4) и (3.6).

$i+1 \longrightarrow i$. Для $\lambda_i(i) < m$ соотношение 1) очевидно. Поэтому предположим, что $\lambda_i(i) \ge m$.

Пусть p_1, p_2 являются двумя наибольшими элементами из S_{i+1} $(p_1 \ge p_2)$. Так как S_{i+1} является первым контейнером, который был закрыт на итерации $i+1$, то вес предмета p_1 больше, чем $level/3$, т. е. предмет p_1 является максимальным предметом из $\theta(i+1)$, который будет удален из списка $L(i+1)$ на итерации $i+1$. Поэтому найдется такое множество $T \in \{T_1(i+1), \ldots, T_{i+1}(i+1)\}$, когда $p_1 \in T$. По предположению индукции, T содержит не более одного предмета p'.

До тех пор, пока только недвойственные предметы будут помещаться в контейнеры на шаге $i+1$, назначение предметов из $\theta(i+1)$ будет тем же, что и в LPT-разбиении $\theta(i+1)$. Очевидно, что $p_2 = p'$, если S_{i+1} является недвойственным контейнером. Если же S_{i+1} является двойственным, то выполняется даже $p_2 \ge p'$.

Итак, можно заключить, что $\theta(i+1) \setminus T$ *доминирует* $\theta(i)$, т. е. для каждого предмета $q' \in \theta(i)$ найдется $q'' \in \theta(i+1) \setminus T$, такой, что $q'' \ge q'$. (Все q'' должны быть различными предметами.)

Пусть $q_1, \ldots, q_r$ являются элементами из $\theta(i)$, причем $q_1 \ge q_2 \ge \ldots \ge q_r$. Предположим, что 1) не выполняется для итерации i, и пусть контейнер $B_i(s)$ $(1 \le s \le i)$ является первым контейнером, для которого LPT-алгоритм назначает три предмета из $\theta(i)$. Обозначим эти предметы через q_s, q_{2i-s+1} и q_t, причем $s \le 2i - s + 1 \le t$. Определим A как LPT-назначение элементов из $\{q_1, \ldots, q_t\}$.

Согласно модифицированному определению LPT-алгоритма все предметы, которые являются единственными в контейнерах в назначении A, имеют вес, превосходящий величину $q_s + q_{2i-s+1}$.

Правило доминирования, индукционное предположение и принцип Дирихле гарантируют, что найдется по крайней мере один такой элемент $q_{r'}$ $(r' \le r)$, который в паре с элементом $q_{t'}$ удовлетворяет $q_{t'} \ge q_t$ в LPT-разбиении множества $\theta(i+1)$. Отсюда

вытекает соотношение

$$q_s + q_{2i-s+1} < q_{r'},$$

и из индукционного предположения 2) для $\ell = 1$ справедливо

$$level \geq q_{r'} + q_{t'} > q_s + q_{2i-s+1} + q_t.$$

Это противоречит $q_t \in \theta(i)$, следовательно, 1) выполняется.

Правило доминирования и 1) имеют такое свойство, что соотношение

$$\sum_{j=1}^{\ell} w\left(T_j(i+1)\right) \geq \sum_{j=1}^{\ell} w\left(T_j(i)\right)$$

выполняется для каждого $\ell \in \{1, \ldots, i\}$. Поэтому соотношение 2) является прямым следствием из индукционного предположения. ■

Следующее утверждение также будет доказано с использованием индукции по i $(i = m, m-1, \ldots, 1)$.

Утверждение 3.13 $level(i) \leq level$.

Доказательство. Для $i = m$ является очевидным следующее соотношение $level(m) \leq a \leq level$.

$i+1 \longrightarrow i$. Предположим, что соотношение $level(i+1) \leq level$ выполняется. Если $w(S_{i+1}) \geq level$, то $level(i) \leq level(i+1) \leq level$, т. е. все доказано. Поэтому предположим, что $w(S_{i+1}) < level$.

Пусть $p_1, \ldots, p_{k(i+1)}$, причем $p_1 \geq p_2 \geq \ldots \geq p_{k(i+1)}$ обозначают предметы в списке $L(i+1)$, а $v_1, \ldots, v_k$ $(v_1 \geq v_2 \geq \ldots \geq v_k)$ обозначают предметы из S_{i+1} соответственно.

Рассмотрим первую ситуацию, когда S_{i+1} является недвойственным контейнером. Пусть $p_d = v_k$. Так как S_{i+1} является недвойственным контейнером, причем первым закрытым на итерации $i+1$, то выполняется $p_{d-1} \neq v_{k-1}$ и

$$v_1 + v_2 + \ldots + v_{k-1} + p_{d-1} > \frac{4}{3} level.$$

Поэтому $v_{k-1} \geq p_{d-1} - v_k > level/3$. Откуда следует $k \leq 3$, что противоречит предположению $k \geq 4$.

Рассмотрим теперь ситуацию, когда контейнер S_{i+1} является двойственным. Обозначим через $v_1, \ldots, v_r$ недвойственные предметы из S_{i+1}, а через $p_s, p_{s+1}, \ldots, p_t$ $(r+t+1-s=k)$ обозначим

двойственные предметы. Из предположения следует соотношение

$$v_1 + v_2 + \ldots + v_r + p_s + p_{s+1} + \ldots + p_t < level.$$

Очевидно, что $p_{s-1} \neq v_r$. Откуда можно заключить, что

$$v_1 + v_2 + \ldots + v_r + p_{s-1} + p_s + \ldots + p_{t-1} > \frac{4}{3} level.$$

Комбинируя эти два соотношения, получаем

$$p_{s-1} > \frac{1}{3} level. \tag{3.18}$$

Более того, все предметы в $P := \{p_1, \ldots, p_{s-1}\}$ являются недвойственными и назначаются в контейнеры в соответствии с LPT-алгоритмом.

Пусть $q_1, \ldots, q_k$ $(q_1 \geq q_2 \geq \ldots \geq q_k)$ являются предметами из произвольного контейнера $B_{i+1}(j)$ со свойством $j \in \{2, \ldots, i+1\}$. Пусть $q_1, \ldots, q_u$ являются предметами, назначенными в $B_{i+1}(j)$ перед предметом p_s.

Предметы $q_1, \ldots, q_u$ являются недвойственными, и в силу (3.18) получаем $q_u > level/3$.

Так как выполняется $\{q_1, \ldots, q_u\} \subseteq P$, то в силу утверждения 3.12(3.10) следует, что $u \leq 2$. Таким образом, все предметы из множества $Q(j) := \{q_{u+1}, \ldots, q_k\}$ были назначены в контейнер $B_{i+1}(j)$ после p_t, и их вес не превышает вес этого предмета.

Пусть $p_{t+1} = w_\nu$ в начальном упорядочении предметов.

$$\bigcup_{j=2}^{i+1} Q(j) \subseteq \{w_\nu, \ldots, w_n\}.$$

Из утверждения 3.11 может быть даже выведено, что

$$\bigcup_{j=2}^{i+1} Q(j) = \{w_\nu, \ldots, w_n\}.$$

Но сейчас это является прямым следствием из утверждения 3.12(3.11), что средний вес предметов в контейнере $\{B_{i+1}(j) | j = 2, \ldots, i+1\}$ не превосходит величины $level$. Ясно, что $level(i) \leq level$. ■

Теперь имеется достаточно соотношений для доказательства основной теоремы. Если окажется, что на итерации i появляется контейнер $B_i(s)$, для которого выполняется $w(B_i(s)) > 4level/3$, то из утверждения 3.13 также следует, что существует такой кон-

тейнер $B_i(r)$, для которого выполняется $w(B_i(r)) < level$.

Но из утверждения 3.10 вытекает, что

$$w(S_i) = w(B_i(1)) \leq 4level/3$$

для каждого $i \in \{1, \ldots, m\}$.

3.4. Задача 3-разбиения

Задача 3-разбиения может быть сформулирована следующим образом. Имеется $3m$ натуральных чисел, которые должны быть разбиты на m подмножеств таким образом, чтобы суммы элементов в подмножествах были равны.

> **Вход:** Натуральное число C и множество $\{p_1, \ldots, p_{3m}\}$ из $3m$ целых чисел для $1 \leq i \leq 3m$.
>
> **Вопрос:** Существует ли разбиение чисел $\{p_1, \ldots, p_{3m}\}$ на m троек таким образом, чтобы сумма чисел в каждой тройке была равна C?

Рассмотрим оптимизационную версию этой проблемы. Имеется $3m$ неотрицательных целых чисел, которые требуется разбить на m троек таким образом, чтобы максимальная сумма элементов в тройках была минимальна. Понятно, что задача 3-разбиения является специальным случаем более общей задачи *k-разбиения* (см. [Ibab98]).

"Двойственный" подход при конструировании приближенных алгоритмов для задач разбиения основывается на решении задачи упаковки (bin packing problem). Результаты, приведенные в предыдущей главе, показали, что комбинация LPT-алгоритма и двойственного алгоритма кажется достаточно целесообразной [Ibab98].

Предлагаемый алгоритм для задачи 3-разбиения является комбинацией этих двух подходов. С одной стороны, в нашем алгоритме используется двоичный поиск для получения приближенной оценки оптимального решения. С другой стороны, мы стараемся распределить "большие" предметы по разным множествам с использованием техники предписания ядер. При этом в алгоритме используется факт, что в каждое множество назначается только три предмета. Такой подход позволил разработать алгоритм, гарантированная оценка которого равна $7/6(1 + 2^{-\lambda})$.

Более того, будет показано, что алгоритм для решения задачи k-разбиения может быть использован для решения задачи теории расписаний. Так, алгоритм может быть применен для разработки приближенного алгоритма решения задачи теории расписаний, причем гарантированная оценка алгоритма равна $(5/4-1/4\,m)$ при линейной трудоемкости, когда число работ равно n.

Прежде чем начать описание алгоритмов, введем некоторые обозначения и сделаем некоторые существенные замечания. Вместо целых чисел мы будем вести речь о предметах (работах) $\mathcal{J} = \{J_1, \ldots, J_n\}$ с весом (временем выполнения) $p_1, \ldots, p_n$. При этом всегда будет предполагаться, что $n = 3m$, так как всегда можно добавить фиктивные предметы с нулевым весом. Более того, предметы $\mathcal{J}$ будут отсортированы в неубывающем порядке своих весов, т. е. $p_1 \geq p_2 \geq \ldots \geq p_n$. Для краткости будем отождествлять предметы с их весами.

Определим C_k^* как оптимальное решение задачи k-разбиения, через C^H будем обозначать решение, построенное приближенным алгоритмом, а через C^* — оптимальное решение соответствующей задачи теории расписаний.

3.4.1 Описание алгоритма

Ключевым инструментом в двойственном аппроксимационном подходе является понятие ρ-релаксационная процедура принятия решения P_ρ. Для задачи k-разбиения оно может быть сформулировано следующим образом.

Определение. *ρ-релаксационная процедура принятия решения* P_ρ определяется следующим образом. В качестве **Входа** имеется km предметов J_i с весами p_i $(i = 1, \ldots, km)$ и задана *граница* C. В качестве **Выхода** получаем ответы $\mathcal{NO}$ или $\mathcal{ALMOST}$.

1) Если выходом является ответ $\mathcal{ALMOST}$, то процедура P_ρ строит решение задачи k-разбиения со значением, не превышающим ρC.

2) Если выходом является ответ $\mathcal{NO}$, то решения задачи k-разбиения со значением, не превышающим C, процедурой не найдено.

Рассмотрим следующий метод построения решения для задачи k-разбиения. Назначим для каждого $j \in \{0, \ldots, k-1\}$ предме-

ты множества $\{J_{jm+1}, J_{jm+2} \ldots, J_{(j+1)m}\}$ произвольно по одному в каждое из m множеств. Было показано, что эта эвристика имеет гарантированную оценку $2 - 1/m$.

Обозначим через C_{km} полученный результат. Тогда для величин C_k^* и C^* соответствующей задачи теории расписаний с mk работами справедливо соотношение

$$\frac{C_{km}}{2} \leq C^* \leq C_k^* \leq C_{km}. \tag{3.19}$$

Неравенства (3.19) дают нижнюю и верхнюю границы, которые будут использованы при двоичном поиске, когда мы будем применять ρ-релаксационную процедуру принятия решения P_ρ. Такой подход обеспечивает построение приближенного алгоритма с гарантированной оценкой ρ.

Лемма 3.9. *Для каждой полиномиальной ρ-релаксационной процедуры принятия решения P_ρ для задачи k-разбиения существует полиномиальный приближенный алгоритм H_ρ, использующий бинарный поиск, гарантированная оценка которого равна ρ. После выполнения λ итераций значение полученного решения не превышает величины $\rho(1 + 2^{-\lambda})C_k^*$.*

Доказательство. Границы C_{km} и $C_{km}/2$ используются для инициализации двоичного поиска. Полагаем $u := C_{km}$, $\ell := C_{km}/2$, $C := \lfloor \frac{1}{2}(u + \ell) \rfloor$. Из (3.19) следует неравенство $u - \ell \leq C_k^*$. Применим ρ-релаксационную процедуру принятия решения P_ρ для C. Если ответом является $\mathcal{ALMOST}$, то полагаем величину u равной C, иначе полагаем величину ℓ равной C и пересчитываем C. После заданного числа итераций λ текущее значение величины u рассматривается как оценка для значения оптимального решения C_k^*. Нетрудно заметить, что эта процедура имеет удовлетворительную гарантированную оценку.

Из 2) можно заключить, что ℓ всегда является нижней оценкой для оптимального решения. Из принципа работы двоичного поиска следует, что выполняется $u - \ell \leq 2^{-\lambda} C_k^*$, и поэтому, учитывая 1),

$$C^H \leq \rho u = \rho(u - \ell + \ell) \leq \rho(2^{-\lambda} C_k^* + C_k^*) = \rho(1 + 2^{-\lambda}) C_k^*.$$

■

Перед описанием алгоритма для решения задачи 3-разбиения введем несколько обозначений. Обозначим через J_i набор из i самых больших пока еще не назначенных предметов, а через J_n —

наименьший из всех еще не назначенных предметов. После назначения J_i мы удаляем этот предмет из списка $\mathcal{J}$.

Для описания алгоритма мы обозначим через f минимальный индекс (если он существует), такой, что

$$p_1 + p_f + p_n \leq C, \tag{3.20}$$

через s — минимальный индекс, такой, что

$$p_1 + p_f + p_s \leq \frac{7}{6}C, \tag{3.21}$$

и, наконец, через t — минимальный индекс (если он существует), такой, что

$$p_{t-1} + p_t + p_n \leq C. \tag{3.22}$$

Теперь можно начать описание $7/6$-релаксационной процедуры принятия решения $P_{7/6}$.

Procedure $\mathbf{P_{7/6}}$.

While $p_1 \geq \frac{2}{3}C$
begin
if предмет J_f существует, то поместить J_1, J_f, J_s вместе в множество
else ответ $\mathcal{NO}$ и STOP;
end;
While $\mathcal{J} \neq \emptyset$
begin
if предмет J_t существует, то
begin
if $p_1 + p_t + p_n \leq \frac{7}{6}C$, то поместить J_1, J_t, J_n вместе в множество
else if предмет J_f существует, то поместить J_1, J_f, J_s вместе в множество
else ответ $\mathcal{NO}$ и STOP;
end
else
ответ $\mathcal{NO}$ и STOP;
end;

Выход: $\mathcal{ALMOST}$ и построенное 3-разбиение.

Для доказательства того факта, что процедура $P_{7/6}$ является $7/6$-релаксационной процедурой принятия решения, воспользуемся техникой минимального контрпримера и правилом доминирования.

Определим *контрпример* как множество из $3m$ предметов $\mathcal{J} = \{J_1, \ldots, J_{3m}\}$, которые должны быть назначены в m множеств таким образом, чтобы процедура $P_{7/6}$ давала выход $\mathcal{NO}$, в то время как оптимальное решение задачи 3-разбиения не превышает C.

Минимальным контрпримером будет множество предметов $\mathcal{J}$, таких, что для каждого предмета выполняются следующие требования.

1) $\mathcal{J}$ является контрпримером.

2) Для каждого m', $1 \leq m' < m$ не существует контрпримера, в котором количество предметов равно $3m'$. ■

Другими словами, минимальным контрпримером является такой набор из m предметов, для которого процедура $P_{7/6}$ прекращает свою работу до момента *заполнения* всех m множеств (контейнеров) тройками, хотя существует такое 3-разбиение предметов, при котором максимальная сумма в тройке не превосходит величины C. Заметим, что согласно способу определения процедура $P_{7/6}$ заполняет по крайней мере один контейнер контрпримера. При использовании свойства "минимальности" нам понадобится принцип *доминирования*.

Определение. Контейнер B_j^* из оптимального 3-разбиения *доминируется* контейнером B_i, заполненным $P_{7/6}$, если существует функция f, отображающая предметы из контейнера B_j^* в предметы из контейнера B_i, которая назначает каждому предмету J_k в B_j^* предмет $J_{f(k)}$ в B_i, такой, что выполняется соотношение $p_k \leq p_{f(k)}$.

Лемма 3.10. *[лемма доминирования]. Пусть $\mathcal{J}$ формирует минимальный контрпример и B_j^* является произвольным контейнером в оптимальном 3-разбиении $\mathcal{J}$.*

Тогда не существует контейнера B_i, заполненного $P_{7/6}$, который будет доминировать B_j^*.

Доказательство. Предположим, что контейнер B_i доминирует B_j^* и пусть $f : B_j^* \longrightarrow B_i$ — соответствующее отображение.

Рассмотрим множество предметов $\mathcal{J}'$, которое получается удалением предметов набора B_i из $\mathcal{J}$, и применим $P_{7/6}$ для $\mathcal{J}'$ с $m-1$ контейнерами. Так как мы удалили из рассмотрения предметы из заполненного контейнера, то процедура $P_{7/6}$ не изменит назначения оставшихся предметов, и выходом процедуры $P_{7/6}$ останется ответ $\mathcal{NO}$.

Теперь построим новое разбиение, основанное на оптимальном решении. Прежде всего, удалим все предметы из B_j^*, добавим все предметы из B_i^* в контейнер B_j^*. Тогда произведем обмен каждого предмета x из множества удаленных предметов с их имиджами $f(x)$. Так как $x \leq f(x)$, то не будет проблем с ограничениями на вместимость контейнеров.

Наконец, удалим все оставшиеся предметы контейнера B_i. Очевидно, что контейнер B_k^* $(k \neq i)$ содержит только предметы из $\mathcal{J}'$. Таким образом, существует разбиение предметов набора $\mathcal{J}'$ в $m-1$ контейнеров вместимости C, что противоречит предположению минимальности. ■

Утверждение 3.14 *Процедура $P_{7/6}$ является $7/6$-релаксационной процедурой принятия решения.*

Доказательство. Описание процедуры $P_{7/6}$ предполагает, что она всегда строит 3-разбиение, если загрузка максимального контейнера не превышает $7\mathrm{C}/6$, причем выходом является $\mathcal{ALMOST}$. Осталось показать, что если выходом $P_{7/6}$ является $\mathcal{NO}$, то это означает, что не существует 3-разбиения, в котором загрузка максимального контейнера не превышает C.

Доказательство будет проводиться от противного. Предположим существование контрпримера, который влечет существование минимального контрпримера. Следовательно, достаточно доказать, что не существует минимального контрпримера.

Пусть имеется множество $\mathcal{J} = \{J_1, \ldots, J_{3m}\}$, состоящее из $3m$ предметов, соответствующее минимальному контрпримеру. Поэтому мы можем предположить, что $C \geq C_3^*$. Докажем два утверждения.

1. Множество предметов $\mathcal{J}$ минимального контрпримера не содержит предметов, вес которых больше или равен $2C/3$.

Предположим, что это утверждение неверно. Тогда выберем $p_1 \geq 2C/3$. Пусть контейнер B построенного нашим алгоритмом 3-разбиения содержит предметы $J_1,\ J_f,\ J_s,$ в то время как контейнер B^* из оптимального 3-разбиения содержит J_1 и два других предмета $J_k,\ J_\ell,$ таких, что $k < \ell$.

В силу предположения о существовании контрпримера такие предметы – J_f и J_s – существуют. В силу выбора индекса f в (3.20) выполняется соотношение $p_f \geq p_k$. Из $p_1 \geq 2C/3$, $p_1 + p_k + p_\ell \leq C$ и $p_k \geq p_\ell$ получаем $p_\ell \leq C/6$. Учитывая это соотношение и (3.20), имеем, что $p_1 + p_f + p_\ell \leq 7C/6$. В силу выбора индекса s получаем $s \leq \ell$ и $p_s \geq p_\ell$. Таким образом, мы показали, что B доминирует B^*, что противоречит лемме доминирования.

2. Для $\mathcal{J}$ выполняется соотношение $p_1 + p_t + p_n > \frac{7}{6}C$.

Предположим, что это не так. Обозначим через B контейнер из 3-разбиения, построенного приближенным алгоритмом, содержащий предметы $J_1,\ J_t,\ J_n,$ а через B^* – контейнер из оптимального 3-разбиения, содержащий предмет J_n и два других предмета $J_k,\ J_\ell,$ удовлетворяющих $k < \ell$. Тогда имеем $p_k + p_\ell + p_n \leq C$, и в силу выбора индекса t в (3.22) получаем $p_t \geq p_\ell$, откуда B доминирует B^*. Этот факт доказывает утверждение.

Так как выполняются утверждения 3.10 и 3.11 и множество $\mathcal{J}$ соответствует контрпримеру, то найдется контейнер B в 3-разбиении, построенном приближенным алгоритмом, с предметами $J_1,\ J_f,\ J_s$ и контейнер B^* в оптимальном 3-разбиении, содержащий предмет J_1 вместе с двумя другими предметами – J_k, J_ℓ. Тогда выполняется следующая цепочка соотношений:

$$\frac{2}{3}C > p_1 > p_{t-1} + \frac{C}{6} \geq p_t + \frac{C}{6} > p_f + \frac{C}{3}.$$

Первое соотношение справедливо в силу утверждения 3.10. Второе неравенство является комбинацией неравенства (3.22) и утверждения 3.11. Последнее соотношение является комбинацией (3.20) и утверждения 3.11.

Отсюда заключаем, что $p_f < C/3$, и в силу (3.20) получаем соотношение $p_f \geq p_k$.

В случае, когда $p_\ell \leq C/6$, из (3.20) очевидно следует соотношение $p_1 + p_f + p_\ell \leq 7C/6$. В случае, когда $p_\ell > C/6$, получаем $p_k > C/6$ и

$$p_1 + p_f + p_\ell \le p_1 + p_f + C - p_1 - p_k \le \frac{C}{3} + C - \frac{C}{6} = \frac{7}{6}C.$$

В силу способа определения индекса s в (3.21) следует, что $p_s \ge p_\ell$. Таким образом, контейнер B доминирует B^*, и мы показали, что не существует минимального контрпримера. Следовательно, $P_{7/6}$ действительно является $7/6$-релаксационной процедурой принятия решения. ■

Вернемся к определению величины C_{3m}. Если использовать величину $C_{3m}/2$ в качестве нижней границы оптимального решения, а величину C_{3m} в качестве верхней границы, то, применяя бинарный поиск с использованием процедуры $P_{7/6}$, мы получим $7/6$-приближенный алгоритм $H_{7/6}$.

Теорема 3.15. *Приближенный алгоритм $H_{7/6}$ строит 3-разбиение с гарантированной оценкой $7/6$. После λ итераций бинарного поиска полученное решение (максимальная загрузка контейнера) не превышает величину $\frac{7}{6}(1 + 2^{-\lambda})C_3^*$. Трудоемкость приближенного алгоритма $H_{7/6}$ оценивается величиной $O(n(\lambda + \log n))$.*

Доказательство. Результат следует непосредственно из леммы 3.9 и утверждения 3.14. ■

3.4.2 Приложения в теории расписаний

Алгоритмы для теории расписаний на многопроцессорных системах могут не только быть основой построения приближенных алгоритмов для задач k-разбиения. Покажем, что каждый ρ-приближенный алгоритм для задач k-разбиения может быть использован в построении приближенных алгоритмов для задач теории расписаний с гарантированной оценкой $\max\left\{\rho, \frac{k+2}{k+1} - \frac{1}{(k+1)m}\right\}$.

Предположим, что имеется n работ с временами выполнения $p_1, p_2, \ldots, p_n$ $(p_1 \ge p_2 \ge \ldots \ge p_n)$ и $n \ge km$. В противном случае добавим необходимое количество фиктивных работ с временем выполнения, равным нулю. Пусть H_ρ обозначает приближенный алгоритм для задачи k-разбиения, гарантированная оценка кото-

рого равна ρ. Тогда для $k \geq 3$ процедура $MP(k)$ может быть определена следующим образом.

Процедура MP(k)

1) Определим для значений $i = k, k+1, \ldots, km$ множество работ $\mathcal{J}_i := \{J_1, \ldots, J_i\}$.

2) Применим приближенный алгоритм H_ρ независимо к множеству работ $\mathcal{J}_k, \mathcal{J}_{k+1}, \ldots, \mathcal{J}_{km}$ (дополненному фиктивными работами нулевой длины) и назначим оставшиеся работы произвольным образом, используя алгоритм "B наименее заполненный".

3) Результатом алгоритма будем считать наилучшее из построенных решений.

Теорема 3.16. *Процедура $MP(k)$ обеспечивает гарантированную оценку $\max\left\{\rho, \frac{k+2}{k+1} - \frac{1}{(k+1)m}\right\}$ для задачи теории расписаний на многопроцессорной системе с идентичными процессорами.*

Доказательство. Определим r как максимальный индекс, такой, что в оптимальном решении для задачи теории расписаний не более k работ из $\mathcal{J}_r$ назначены на один процессор. Очевидно, что $k \leq r \leq km$. Если $r = n = km$, то оптимальное решение для задачи теории расписаний является и оптимальным решением для задачи k-разбиения, поэтому алгоритм H_ρ, примененный к множеству работ $\mathcal{J}_n$, обеспечивает построение решения с гарантированной оценкой ρ.

В противном случае максимальность величины r влечет

$$C^* \geq \sum_{i=r-k+1}^{r+1} p_i,$$

и поэтому

$$p_{r+1} \leq \frac{C^*}{k+1}. \tag{3.23}$$

Определим через C_r^H максимальную загрузку, построенную алгоритмом H_ρ, который был применен для множества работ $\mathcal{J}_r$. Пусть C_k^* означает максимальную загрузку оптимального k-разбиения множества работ $\mathcal{J}_r$ (дополненных фиктивными работами). Согласно определению r имеем $C^* \geq C_k^*$.

Рассмотрим распределение работ, полученное с помощью алгоритма H_ρ, примененного ко множеству работ $\mathcal{J}_r$, и после этого распределим оставшиеся работы с помощью алгоритма "B наименее загруженный". Пусть J_ℓ соответствует последней работе на процессоре, загрузка которого максимальна.

Если $\ell \le r$, то максимальная загрузка не увеличивается после назначения работ $J_1, \ldots, J_r$, поэтому для максимальной загрузки C^H для $MP(k)$ справедливо

$$C^H \le C^H_r \le \rho C^*_k \le \rho C^*.$$

В противном случае для $\ell > r$ мы имеем

$$\begin{aligned} C^H &= (C^H - p_\ell) + p_\ell \le \frac{1}{m}\left(\sum_{i=1}^{n} p_i - p_\ell\right) + p_\ell \le \\ &\le C^* + p_\ell\left(1 - \frac{1}{m}\right) \le \left(\frac{k+2}{k+1} - \frac{1}{(k+1)m}\right) C^*, \end{aligned}$$

где первое неравенство является следствием алгоритма "B наименее загруженный", а третье следует из (3.23). ■

Последовательное применение приближенного алгоритма $H_{7/6}$ вместе с алгоритмом "B наименее загруженный" приводит к приближенному алгоритму для задачи теории расписаний с гарантированной оценкой $(5/4 - 1/(4m))$, трудоемкость которого является линейной функцией от количества работ n.

Следствие 3.1. *Процедура $MP(3)$, использующая приближенный алгоритм $H_{7/6}$, позволяет построить приближенный алгоритм для задачи теории расписаний на многопроцессорной системе с идентичными процессорами, гарантированная оценка которого равна $\max\left\{\frac{5}{4} - \frac{1}{4m}, \frac{7}{6}(1 + 2^{-\lambda})\right\}$, а трудоемкость оценивается величиной $O(nm + m^2\lambda)$.*

Доказательство. Процедура $MP(3)$ требует сортировки только $3m$ максимальных по длительности работ, что может быть выполнено за время $O(m \log m)$. Так как процедура $H_{7/6}$ используется $O(m)$ раз, то шаг 2 может быть пополнен за время $O(m^2\lambda + nm)$. Следовательно, трудоемкость алгоритма оценивается величиной $O(nm + m^2\lambda)$. ■

3.5. Задача теории расписаний

Пусть имеется m одинаковых станков, на которых необходимо обработать n деталей. Для каждой детали i $(i = 1, ..., n)$ определены три параметра:

r_i - время задержки,

p_i - время обработки детали на станке,

d_i - время доставки детали на склад после обработки.

Здесь r_i означает самый ранний момент времени, когда может быть начата обработка детали с номером i на станке. При этом станок не может обрабатывать несколько деталей одновременно. Задача состоит в определении такой последовательности обработки, при которой все детали окажутся на складе через минимальное время.

Для описания алгоритма введем следующие обозначения. Через * будем обозначать значение оптимального решения задачи, т. е. минимальное время, через которое все детали будут обработаны и доставлены на склад. Через T^A будем обозначать значение решения, построенного алгоритмом A.

Ниже будет предложен алгоритм A градиентного типа и установлено, что для величин T^* и T^A справедливо соотношение $T^A \leq 2T^*$.

Не умаляя общности, будем считать, что $r_i \leq r_{i+1}$ для $i = 1, ..., n-1$. Действительно, этого можно добиться, отсортировав детали соответствующим образом за время, пропорциональное величине $n\log(n)$.

Пусть $S(k) = (p_1 + ... + p_k)/m$, $k = 1, ..., n$, а $L(k) = \max[r_i + p_i + d_i, S(k)]$, где максимум берется по всем значениям $i = 1, ...k$.

Для определенных выше величин очевидны следующие утверждения.

Утверждение 3.17 $L(k) \leq T^*$, $k = 1, ..., n$.

Утверждение 3.18 $L(k) \leq L(k+1)$, $k = 1, ..., n-1$.

Через $H(k)$ обозначим общее время простоя всех станков после распределения k деталей, получившееся в результате назначения деталей на станки по алгоритму . Эти простои связаны с наличием величины r_i.

Через $Sph(j,k)$ будем обозначать общее время простоя и работы станка j, $j = 1,...,m$, после k-го шага алгоритма, а через $S(j,k)$ - максимальное время доставки деталей, обработанных на станке j.

Опишем шаг алгоритма $(k+1)$, т. е. правило выбора станка, на который очередная $(k+1)$ деталь будет назначаться. Для этого шага справедливы следующие утверждения.

Утверждение 3.19 *Существует станок с номером , для которого справедливо соотношение* $Sph(x,k) \le S(k) + H(k)/m$.

Справедливость этого соотношения следует из равенства $Sph(1,k) + ... + Sph(m,k) = mS(k) + H(k)$

Утверждение 3.20 $r_{k+1} \ge H(k)/m$.

Справедливость этого соотношения следует из упорядоченности величин r_k.

Из приведенных выше утверждений вытекает следующее следствие.

Следствие 3.2. $p_{k+1} + d_{k+1} \le L(k+1) - H(k)/m$

Доказательство следует из соотношения $r_{k+1} + {}_{k+1} + d - k + 1 \le L(k+1)$ и утверждения 3.20.

Лемма 3.11. *Если для обработки детали* $k+1$ *назначить станок , для которого справедливы соотношения* $Sph(x,k) \le S(k) + H(k)/m$ *и* $S(,k) \le 2L(k)$, *то для величины* $S(x,k+1)$ *будет справедливо соотношение* $S(,k+1) \le 2L(k+1)$.

Доказательство. Возможны три ситуации.

1) Максимальное время доставки деталей, обработанных на станке после добавления новой детали, не увеличивается. В этом случае очевидно соотношение $S(,k+1) = S(,k)$.

2) Максимальное время доставки деталей, обработанных на станке после добавления новой детали, увеличивается, но при этом не возникает нового интервала простоя, т. е. $Sph(,k) \ge r_{k+1}$ и $S(,k) < Sph(,k) + {}_{k+1} + d_{k+1}$. В этом случае $S(,k+1) = Sph(,k) + {}_{k+1} + d_{k+1}$.

3) Максимальное время доставки деталей, обработанных на станке после добавления новой детали, увеличивается и при этом возникает новый интервал простоя, т. е. $Sph(,k) < r_{k+1}$ и

$S(,k) < r_{k+1} + {}_{k+1} + d_{k+1}$. В этом случае $S(,k+1) = r_{k+1} + {}_{k+1} + d_{k+1}$. Отсюда, в силу утверждений вытекает, что $S(x,k+1) \leq \max[S(,k), S(k) + H(k)/m + L(k+1) - H(k)/m, r_{k+1} + {}_{k+1} + d_{k+1}] \leq \max[2L(k), S(k) + L(k+1), L(k+1)] \leq 2L(k+1)$. Лемма доказана.

Описание алгоритма A .

1. Сортируем детали в порядке неубывания величин r_i , $i = 1, ..., n$.

2. Помещаем в приоритетную очередь величины $Sph(j,k)$, $j = 1, ..., m$. Вначале все эти величины равны нулю.

3. Итерация $k+1$, $k = 0, ..., n-1$.

А) Достаем из приоритетной очереди элемент (с минимальным значением $Sph(j,k)$, $j = 1, ..., m$);

Б) назначаем на соответствующий станок текущую деталь и помещаем в приоритетную очередь величину $Sph(,k+1)$.

4. Алгоритм заканчивает работу, когда все детали назначены на станки.

Трудоемкость алгоритма пропорциональна величине nlogn, если использовать приоритетную очередь.

Теорема 3.21. *Для алгоритма A справедлива оценка* $T^A \leq 2T^*$.

Доказательство следует из леммы 3.11 при $k = n$ и утверждения 3.17.

Следствием теоремы является тот факт, что в случае, когда все значения r_i , $i = 1, ..., m$, равны между собой, то on-line алгоритм имеет гарантированную оценку, равную 2 .

3.6. Задача о разбиении

Задача о разбиении состоит в следующем. Набор S из n неотрицательных целых чисел требуется разбить на 2 подмножества S_1 и S_2 таким образом, чтобы суммы элементов в подмножествах были равны. В силу NP-полноты данной проблемы мы рассмотрим оптимизационный вариант задачи разбиения, чтобы суммы элементов в подмножествах были приблизительно равны. Такое равенство будет измеряться отношением суммы элементов в большем подмножестве к сумме элементов в меньшем подмножестве. Целью будет минимизация суммы элементов в большем подмножестве.

Вход: Набор $S = \{p_1, \ldots, p_n\}$ натуральных чисел.

Задача: Найти разбиения набора S в два набора S_1 и S_2, таких, что сумма элементов в наборе с большей суммой минимальная из возможных.

Задача о разбиении эквивалентна задаче теории расписаний в случае наличия 2-процессорной системы, на которой требуется выполнить независимые работы, и при этом требуется минимизировать время завершения последней выполненной работы.

Задача разбиения с предписанием является обобщением классической задачи разбиения и состоит в следующем. В дополнение ко множеству предметов выделены два предмета с неотрицательными весами r_1, r_2 (называемые ядрами), которые должны быть назначены в различные подмножества. Эти ядра могут быть интерпретированы как времена задержки, если рассматривать задачу теории расписаний, когда некоторые процессоры недоступны в нулевой момент времени. В таком контексте r_i соответствует самому раннему времени, когда процессор i может начать обработку.

В данной главе предлагаются две серии алгоритмов, причем трудоемкость алгоритмов *линейна*. Первый алгоритм, который будет называться C-Greedy алгоритмом, имеет гарантированную оценку $12/11$, а второй алгоритм, называемый C(kernel)-Greedy, имеет гарантированную оценку $8/7$. Оба алгоритма состоят из трех процедур, выполняемых параллельно. В качестве решения алгоритма выбирается лучшее из решений, построенных процедурами. Все эти процедуры являются вариантом так называемого ε-Greedy алгоритма $G_{\varepsilon}(k)$. Нетрудно видеть, что общая сумма элементов S (или, соответственно, общая сумма элементов S плюс ядра), деленная на два, является нижней оценкой для значения оптимального решения. Алгоритм $G_{\varepsilon}(k)$ назначает предметы в наиболее загруженное множество до тех пор, пока общая загрузка не превышает величину нижней границы, умноженной на величину $1 + \varepsilon$. Процедуры имеют различие только в предварительном разбиении отдельных элементов.

Прежде чем начать описание алгоритмов, введем несколько необходимых обозначений. Не умаляя общности будем считать, что $r_2 = 0$. Определим $P_t := \{p_1, \ldots, p_t\}$. *Весом* $l(T)$ подмножества T множества S будем называть величину $l(T) = \sum_{x \in T} x$. Опре-

делим $C := \frac{1}{2}(\sum_{p \in S} p + r_1)$. (Для задачи разбиения будем предполагать, что $r_1 := 0$.) Пусть S_1 и S_2 являются множествами, полученными в результате работы серий алгоритмов. Величины S_1^* и S_2^* отвечают значениям соответствующих оптимальных решений. Соответственно определим $C^H := \max\{l(S_1), l(S_2)\}$ и $C^* := \max\{l(S_1^*), l(S_2^*)\}$. Не умаляя общности будем считать, что для алгоритма C-Greedy элемент p_1 является максимальным элементом в S_1 и S_1^* соответственно. Аналогично будем полагать, что для алгоритма C(kernel)-Greedy ядро r_1 является элементом S_1 и S_1^* соответственно.

3.7. Градиентный алгоритм

Вначале определим так называемый ε-Greedy алгоритм $G_\varepsilon(k)$. Он осуществляет разбиение множества S на два подмножества S_1 и S_2 и является основным инструментом при построении алгоритмов.

ε-Greedy алгоритм $\mathbf{G}_\varepsilon(\mathbf{k})$

1) Выберем k элементов из S, имеющих максимальное значение $p_1 \geq p_2 \geq \ldots \geq p_k$.

2) $t := 1$, $S_1 := \emptyset$, $S_2 := \emptyset$.

3) **Если** $t \leq k$, то выбираем элемент p_k, **иначе** выбираем любой неназначенный элемент из S.

4) Назначаем текущий выбранный элемент в одно из множеств S_1, S_2, имеющее максимальный суммарный вес элементов, **если** полученный суммарный вес не превосходит величины $(1 + \varepsilon)C$. В противном случае этот элемент назначается в множество с минимальным суммарным весом.

5) {Критерий завершения}
Если суммарный вес одного из множеств S_1, S_2 принадлежит интервалу $[(1-\varepsilon)C, (1+\varepsilon)C]$, то оставшиеся неназначенными элементы назначаются в другое множество, и алгоритм заканчивает работу.
Если элемент должен быть добавлен в множество с минимальным весом, но при этом получающийся суммарный вес

этого множества превосходит величину $(1+\varepsilon)C$, то оставшиеся неназначенными элементы назначаются в другое множество, и алгоритм заканчивает работу.

Если $t = n$, то алгоритм заканчивает работу.

6) $t := t + 1$, **goto** 2.

Описанный ε-Greedy алгоритм может быть модифицирован, если наложить требование, что элементы определенного подмножества множества S должны помещаться в одно и то же множество в процессе разбиения элементов.

Будем называть алгоритм, который вначале осуществляет назначение элементов подмножества $T \subseteq S$ в S_1, а затем применяется алгоритм $G_{\varepsilon}(k)$, *ε-Greedy алгоритм со стартовой точкой* $T \subseteq S$, или, коротко, $G_{\varepsilon}(k, T)$. Приведем формальное описание алгоритма C-Greedy.

Алгоритм C-Greedy

Полагаем $\varepsilon = 1/11$. Выполняем серию из следующих трех алгоритмов и выбираем лучшее из найденных решений:

1) $G_{\varepsilon}(9)$,

2) $G_{\varepsilon}(9, \{p_1, p_5, p_6\})$,

3) $G_{\varepsilon}(9, \{p_1, p_4\})$.

Алгоритм C(kernel)-Greedy может быть описан следующим образом. Напомним, что ядро r_1 назначается в подмножество S_1 и $r_2 = 0$.

Алгоритм C(kernel)-Greedy

Полагаем $\varepsilon = 1/7$. Выполняем серию из следующих трех алгоритмов и выбираем лучшее из найденных решений:

1) $G_{\varepsilon}(5)$,
2) $G_{\varepsilon}(5, \{p_2\})$,
3) $G_{\varepsilon}(5, \{p_3, p_4\})$.

Теорема 3.22. *Алгоритм C-Greedy имеет гарантированную оценку 12/11.*

Теорема 3.23. *Алгоритм C(kernel)-Greedy имеет гарантированную оценку* $8/7$.

Пусть A_1 и B_1 соответствуют множествам, построенным алгоритмом $G_{\,\varepsilon}\,(9)$, A_2 и B_2 соответствуют множествам, построенным алгоритмом $G_{\,\varepsilon}\,(9,\{p_1,p_5,p_6\})$, а A_3 и B_3 соответствуют множествам, построенным алгоритмом $G_{\,\varepsilon}\,(9,\{p_1,p_4\})$. Не умаляя общности будем считать, что каждое из множеств A_1, A_2 и A_3 содержит максимальный элемент p_1. Определим $C_i := \max\{l(A_i), l(B_i)\}$ для $i = 1,2,3$. Понятно, что алгоритм C-Greedy имеет решение со значением $\min\{C_1, C_2, C_3\}$.

Пример для алгоритма C-Greedy. Пусть имеется множество S из шести элементов с весами $p_1 = 6 - 5\delta$, $p_2 = 4 + 4\delta$, $p_3 = 4 - \delta$, $p_4 = 4 - 2\delta$, $p_5 = p_6 = 2 + 2\delta$ и δ – маленькие неотрицательные числа. Тогда $C = \frac{1}{2}l(S) = 11$ и интервал $[(1-\delta)C, (1+\delta)C]$ соответствует интервалу $[10, 12]$. Применяя алгоритм $G_{\,\varepsilon}\,(9)$, элементы p_1, p_2 будут назначены в множество A_1, а элементы p_3, p_4, p_5 будут назначены в множество A_2. Так как $p_1 + p_2 = p_3 + p_4 + p_5 = 10 - \delta$, то элемент p_6 может быть назначен произвольно в A_1 или A_2, при этом справедливо соотношение $C_1 = 12+\delta$. Аналогично $A_2 = \{p_1, p_5, p_6\}$ и $B_2 = \{p_2, p_3, p_4\}$ с $C_2 = 12 + \delta$. Наконец, $A_3 = \{p_1, p_4, p_5\}$ и $B_3 = \{p_2, p_3, p_6\}$ с $C_3 = 12 - 5\delta$. Следовательно, алгоритм выберет третье решение с $C^H = 12 - 5\delta$.

Доказательство теоремы 3.22

Пронумеруем веса элементов таким образом, чтобы выполнялось соотношение $l(S) = 22$. Тогда $C = 11$.

Доказательство будет проводиться от противного. Предположим, что существует такое множество S, состоящее из n элементов, для которого алгоритм C-Greedy не работает. Это означает, что на некотором шаге t алгоритм $G_{\,\varepsilon}\,(9)$ назначает соответствующий элемент p в множество с минимальным текущим суммарным весом, и при этом новый суммарный вес превышает 12, в то время как на шаге $t-1$ оба множества имели суммарный вес, не превышающий 10. Следовательно, $p > 2$. Ни одно из этих множеств не может содержать более четырех элементов, веса которых превышают 2 на шаге $t-1$. Следовательно, $t \le 9$ и $p = p_t$ $(t \in \{1, \ldots, 9\})$.

Заметим, что если одно из множеств имеет суммарный вес между 10 и 12, то суммарный вес другого не может превысить 12, и в этом случае доказательство закончено, так как C является нижней границей для C^*.

Рассмотрим несколько случаев в соответствии со значением величины t.

Случай 1: $t = 9$.

Для $t = 9$ имеем

$$p_1 + p_2 + p_3 + p_4 \le 22 - \sum_{i=5}^{9} p_i < 12.$$

Поэтому $\{p_1, \ldots, p_4\} = A_1$ и $\{p_5, \ldots, p_9\} = B_1$. $C_1 = l(B_1) = \sum_{i=5}^{9} p_i > 12$. Но величина $l(B_1)$ является нижней границей для C^* и $G_{\mathcal{E}}(9)$ строит оптимальное решение. ■

Случай 2: $t = 8$.

В силу $\sum_{i=4}^{8} p_i > 10$ имеем соотношение $p_1 + p_2 + p_3 < 12$, из которого непосредственно следует $p_1 + p_2 + p_3 < 10$ (в противном случае доказательство завершено). Если $p_1 + p_2 + p_3 + p_4 \le 12$, то выполняется соотношение $p_5 + p_6 + p_7 + p_8 \le 12$, что противоречит условию $t = 8$. Таким образом, $p_1 + p_2 + p_3 + p_4 > 12$.

2.1. $p_1 + p_2 + p_3 + p_8 < 10$.

Это неравенство вместе с неравенством, приведенным ранее, влечет соотношения $p_4 > p_8 + 2 > 4$ и $p_1 + p_2 + p_3 > 3 \cdot 4 = 12$, что противоречит предположению.

2.2. $p_1 + p_2 + p_3 + p_8 > 12$.

Алгоритм $G_{\mathcal{E}}(9)$ помещает элементы p_1, p_2, p_3 в множество A_1, а элементы p_4, p_5, p_6, p_7 в множество B_1, причем $p_1 + p_2 + p_3 < 10$ и $p_4 + p_5 + p_6 + p_7 < 10$.

Следовательно, имеем $p_2 < 4$ и $p_2 - p_8 < 2$.

Рассмотрим первый случай, когда $p_1 \le 4$, и применим алгоритм $G_{\mathcal{E}}(9, \{p_1, p_4\})$. Так как $p_1 + p_2 + p_3 + p_8 > 12$ и $p_3 - p_4 \le p_2 - p_8 < 2$, то выполняется $p_1 + p_2 + p_4 + p_8 > 10$. В силу того, что алгоритм $G_{\mathcal{E}}(9, \{p_1, p_4\})$ не сработал, имеем $p_1 + p_2 + p_4 + p_8 > 12$. Следовательно, выполняются соотношения $p_4 + p_5 + p_6 + p_7 < 10$, $p_3 - p_4 < 2$, $p_3 + p_5 + p_6 + p_7 < 10$. (Напомним, что в случае, если $l(B_3) \in [10, 12]$, то доказательство закончено.) Суммируя вышесказанное, имеем $\{p_1, p_2, p_4\} \subseteq A_3$ и $\{p_3, p_5, p_6, p_7\} \subseteq B_3$.

Наконец, рассмотрим последний алгоритм $G_{\varepsilon}(9, \{p_1, p_5, p_6\})$. Если $p_1 + p_3 + p_5 + p_6 < 10$, то справедливо соотношение $p_2 + p_4 + p_7 + p_8 < 10$. Легко видеть, что $\sum_{i=1}^{8} p_i > 20$, в противном случае $p_8 > 4$. Таким образом, $p_1+p_3+p_5+p_6 > 12$. Это влечет соотношение $p_1+p_5+p_6+p_8 > 10$, так как $p_3-p_8 < 2$. Предположим, что алгоритм $G_{\varepsilon}(9, \{p_1, p_5, p_6\})$ не сработал. Поэтому справедливо соотношение $p_1+p_5+p_6+p_8 > 12$. Комбинируя с соотношением $p_3+p_5+p_6+p_7 < 10$, имеем $p_1-p_3 > 2$, что противоречит $p_1 \leq 4$.

Сейчас предположим, что $p_1 > 4$. Рассмотрим оптимальное назначение элемента P_8. Если одно из двух множеств S_1^*, S_2^* содержит по крайней мере пять элементов из P_8, то суммарный вес этого множества больше, чем $\sum_{i=4}^{8} p_i$, а алгоритм $G_{\varepsilon}(9)$ строит оптимальное решение. Поэтому каждое из множеств S_1^* и S_2^* содержит четыре элемента из P_8, причем $p_1+p_6+p_7+p_8$ является нижней границей для C^*. Для выполнения условия, что $l(P_8)$ не превышает 22, должно быть справедливо соотношение $p_5 < 3$. Выполним теперь алгоритм $G_{\varepsilon}(9, \{p_1, p_5, p_6\})$.

Соотношение $p_1 > 4$ влечет $p_1 + p_5 + p_6 + p_8 > 10$, и, следовательно, $p_2 + p_3 + p_4 + p_7 \leq 12$.

Поэтому справедливы $\{p_1, p_5, p_6\} \subseteq A_2$ и $\{p_2, p_3, p_4, p_7\} \subseteq B_2$, тогда $p_1 + p_5 + p_6 + p_8 \geq C_2$. Но

$$C^H - C^* \leq (p_1 + p_5 + p_6 + p_8) - (p_1 + p_6 + p_7 + p_8) = p_5 - p_7 < 1,$$

что противоречит предположению о существовании контрпримера.

Лемма 3.12. *Пусть выполняются соотношения* $p_1+p_2 < 10$, $p_1 + p_2 + p_3 > 12$. *Тогда* $p_1 + p_2 + p_t > 12$.

Доказательство. Предположим, что $p_1 + p_2 + p_t \leq 12$. Тогда пусть $t' \leq t$ соответствует минимальному индексу, для которого выполняется $p_1+p_2+p_{t'} \leq 12$. Выполняя алгоритм $G_{\varepsilon}(9)$, имеем $p_1 + p_2 + p_t \leq p_1 + p_2 + p_{t'} < 10$. Поэтому $p_3 - p_t > 2$ и $p_3 > 4$.

Но это противоречит условию $p_1 + p_2 + p_t < 10$. ∎

Случай 3: $t = 7$.

3.1. $p_1 + p_2 + p_3 < 10$.

Неравенство $p_1 + p_2 + p_3 + p_7 < 10$ влечет $p_4 + p_5 + p_6 < 10$, и доказательство закончено. Если $p_1+p_2+p_3+p_7 > 12$, то алгоритм $G_{\varepsilon}(9)$ назначает элементы p_4, p_5, p_6, p_7 в множество B_1, и справедливо соотношение $l(B_1) = C_1$. Как минимум четыре эле-

мента из P_7 должны быть назначены в одно множество. Поэтому $C^* \geq p_4 + p_5 + p_6 + p_7$ и $C^* \geq l(B_1)$.

3.2. $p_1 + p_2 + p_3 > 12$.

В силу $p_3 + p_4 + p_5 + p_6 + p_7 > 10$ имеем $p_1 + p_2 < 12$, причем $p_1 + p_2 < 10$. В силу леммы 4 $p_1 + p_2 + p_7 > 12$.

Алгоритм $G_{\varepsilon}(9)$ разбивает элементы P_6 следующим образом: $\{p_1, p_2\} \subseteq A_1$, $\{p_3, \ldots, p_6\} \subseteq B_1$. Так как алгоритм $G_{\varepsilon}(9)$ не работает, то $p_3 + \ldots + p_6 < 10$ и $p_3 + \ldots + p_7 > 12$. Если одно из множеств S_1^*, S_2^* содержит по крайней мере пять элементов из P_7, то алгоритм $G_{\varepsilon}(9)$ строит оптимальное решение, так как $\sum_{i=3}^{7} p_i$ является нижней границей для C^*. Следовательно, оптимальное разбиение элементов из P_7 осуществляется в пропорции 4:3, следовательно, выполняется $C^* \geq p_1 + p_6 + p_7$.

Рассмотрим случай, когда $p_1 - p_2 \geq 2$. Очевидно, что $p_1 + (p_1 - 2) + p_7 \geq p_1 + p_2 + p_7 > 12$. Поэтому $p_1 > 7 - p_7/2$ и

$$p_1 + p_5 + p_6 > 7 - p_7/2 + p_5 + p_6 \geq 7 + 3/2\, p_6 > 10.$$

Если $p_1 + p_5 + p_6 \leq 12$, то алгоритм $G_{\varepsilon}(9, \{p_1, p_5, p_6\})$ находит решение из интервала $[10, 12]$. Поэтому $p_1 + p_5 + p_6 > 12$.

Напомним, что $C^* > p_1 + p_6 + p_7$. Из $p_3 + p_4 + p_5 + p_6 < 10$ получаем $p_5 < 3$ и $p_1 + p_5 + p_6 - p_1 - p_6 - p_7 = p_5 - p_7 < 1$.

Следовательно, для $p_1 + p_5 + p_6 > 12$ алгоритм $G_{\varepsilon}(9, \{p_1, p_5, p_6\})$ работает корректно.

Сейчас предположим, что $p_1 - p_2 < 2$. Из леммы 4 следует, что $p_1 + p_2 + p_4 > 12$. Напомним, что $p_3 + p_4 + p_5 + p_6 < 10$. Тогда из $p_1 + p_3 + p_4 > 12$ и $p_3 + p_4 + p_5 + p_6 < 10$ следует, что $p_1 > p_5 + p_6 + 2 > 6$ и $p_2 > 4$. Следовательно, $p_1 + p_3 + p_4 < 10$.

Суммируя вышесказанное, имеем $\{p_1, p_3, p_4\} \subseteq A_3$ и $\{p_2, p_5, p_6\} \subseteq B_3$. Поэтому $C^* \geq \min\{p_1 + p_2 + p_7,\ p_2 + p_5 + p_6 + p_7\}$ и $C^H = C^*$.

Лемма 3.13. *Для $p_t \leq 4$ справедливо $C^* < 12$.*

Доказательство. Предположим $C^* \geq 12$. Тогда $C^H > 13$. Пусть x и y $(x \geq y)$ соответствуют весам A_1 и B_1 до того, как алгоритм $G_{\varepsilon}(9)$ назначил элемент p_t. Тогда $y > 13 - p_t$.

С другой стороны, выполняется соотношение $22 \geq x + y + p_t > 2y + p_t$, что эквивалентно $y < 11 - p_t/2$. Но $11 - p_t/2 > 13 - p_t \Longleftrightarrow p_t > 4$, получили противоречие. ■

Случай 4: $t = 6$.

4.1. $p_1 + p_2 < 10$.

Если $p_1 + p_2 + p_3 \leq 12$, то алгоритм $G_\varepsilon(9)$ заканчивает работу. Поэтому выполняется соотношение $p_1 + p_2 + p_3 > 12$, и в силу леммы 3.12 $p_1 + p_2 + p_6 > 12$. Алгоритм $G_\varepsilon(9)$ назначает p_1, p_2 в A_1, а p_3, p_4, p_5 в B_1. Рассмотрим оптимальное решение. Элементы из P_6 должны быть разбиты в пропорции 3:3. Это справедливо, так как для другой пропорции величина $p_3 + p_4 + p_5 + p_6$ является нижней оценкой для C^* и $C_1 \leq p_3 + p_4 + p_5 + p_6$. Отсюда заключаем, что $C^* \geq p_1 + p_5 + p_6$, и так как алгоритм $G_\varepsilon(9, \{p_1, p_5, p_6\})$ не сработал, то $p_1 + p_5 + p_6 < 10$. Аналогично случаю 2.2 можно заключить, что выполняется $l(P_6) > 20$. Это влечет $p_2 + p_3 + p_4 > 10$, а условие, что алгоритм $G_\varepsilon(9, \{p_1, p_5, p_6\})$ не работает, влечет $p_2 + p_3 + p_4 > 12$. В силу $p_3 + p_4 + p_5 < 10$ имеем $p_4 < 4$. Учитывая неравенство $p_2 + p_3 + p_4 > 12$, заключаем, что $p_2 + p_3 + p_6 > 10$. Следовательно, $p_1 + p_4 + p_5 < 12$.

Предположение, что даже алгоритм $G_\varepsilon(9, \{p_1, p_4\})$ не строит решение требуемого вида, влечет соотношения $\{p_2, p_3\} \subseteq B_3$ и $p_2 + p_3 + p_6 > 12$. Поэтому, учитывая предположение о пропорции 3:3 в оптимальном решении, $C^* \geq p_2 + p_3 + p_6$, получаем противоречие с леммой 3.13.

4.2. $p_1 + p_2 > 12$.

Предположим, что $p_1 + p_3 < 10$. Разбиение множества P_5 может быть осуществлено следующим образом: $\{p_1, p_3\} \subseteq A_1$, $\{p_2, p_4, p_5\} \subseteq B_1$, когда веса обоих множеств меньше чем 10. Из $p_1 + p_2 > 12$ имеем $p_2 - p_3 > 2$. Тогда $p_2 + p_4 + p_5 < 10$ влечет $p_3 < 4$. Учитывая условие $p_1 + p_3 + p_6 > 12$, имеем $p_1 + p_5 + p_6 > 10$.

Если $p_1 + p_5 + p_6 < 12$, то достаточно выполнить алгоритм $G_\varepsilon(9, \{p_1, p_5, p_6\})$.

Рассмотрим возможное распределение элементов P_6 в оптимальном решении. Если как минимум четыре элемента из P_6 находятся в одном множестве, то $C^* \geq p_3 + p_4 + p_5 + p_6$, и можно использовать алгоритм $G_\varepsilon(9)$, иначе $C^* \geq p_1 + p_5 + p_6$, тогда алгоритм $G_\varepsilon(9, \{p_1, p_5, p_6\})$ строит решение.

Осталось исследовать случай $p_1 + p_3 > 12$. Утверждается, что $p_1 + p_4 > 12$. В противном случае $p_1 + p_4 < 10$ и $p_3 - p_4 > 2$, или $p_3 > 4$. Тогда $l(P_6) > (p_1 + p_3) + p_2 + 6 > 12 + 4 + 6 = 22$, что невозможно. Очевидно, что $p_1 \in A_1$ и $\{p_2, p_3, p_4\} \subseteq B_1$.

Если распределение элементов из P_6 в оптимальном решении осуществляется в пропорции 3:3, то алгоритм $G_{\varepsilon}(9, \{p_1, p_5, p_6\})$ строит оптимальное решение.

Возьмем другую пропорцию. Если один из элементов p_2, p_3 или p_4 является элементом из S_1^*, может быть применена лемма 3.13. Поэтому S_1^* содержит не более двух элементов из P_6. Следовательно, p_2, p_3, p_4 и четвертый элемент из P_6 находятся в S_2^*. Тогда может быть применена лемма 3.13. ■

Случай 5: $t \leq 5$.

Очевидно доказывается, что для $t \leq 4$ ε-Greedy алгоритм $G_{\varepsilon}(9)$ строит оптимальное решение или такое, для которого выполняется соотношение $C_1 \leq 12$.

Для случая $t = 5$ рассмотрим четыре возможности в зависимости от того, как распределены элементы в A_1 :

a) $\{p_1\} \subseteq A_1$, $\{p_2, p_3, p_4\} \subseteq B_1$,

b) $\{p_1, p_4\} \subseteq A_1$, $\{p_2, p_3\} \subseteq B_1$,

c) $\{p_1, p_3\} \subseteq A_1$, $\{p_2, p_4\} \subseteq B_1$,

d) $\{p_1, p_2\} \subseteq A_1$, $\{p_3, p_4\} \subseteq B_1$.

Следует заметить, что алгоритм $G_{\varepsilon}(9)$ способен обеспечить требуемую оценку. Доказательство опущено, так как оно достаточно простое и использует те же идеи, что и предыдущие случаи. ■

Доказательство теоремы 3.23

Доказательство по духу аналогично доказательству теоремы 9. Вначале нормируются веса элементов таким образом, чтобы выполнялось $l(S) = 14$. Поэтому $C = 7$. Пусть A_1 и B_1 соответствуют множествам, построенным алгоритмом $G_{\varepsilon}(5)$, A_2 и B_2 соответствуют множествам, построенным алгоритмом $G_{\varepsilon}(5, \{p_2\})$, и пусть A_3 и B_3 соответствуют множествам, построенным алгоритмом $G_{\varepsilon}(5, \{p_3, p_4\})$.

Не умаляя общности будем считать, что каждое из множеств A_1, A_2 и A_3 содержит ядро r_1. Определим $C_i := \max\{l(A_i), l(B_i)\}$ для $i = 1, 2, 3$.

Доказательство ведется от противного. Предположим, что имеется множество S, содержащее n элементов, такое, что алгоритм $G_{\varepsilon}(5)$ не срабатывает на некотором шаге t. По аналогии с доказа-

тельством теоремы 3.23 заключаем, что после назначения элемента p_t имеется множество с суммарным весом, большим 8, тогда как на шаге $t-1$ оба множества имели вес меньше 6. Следовательно, $p_t > 2$, и на шаге $t-1$ никакое из двух множеств не содержит более двух элементов из S. Следовательно, $t \leq 5$. Более того, не может быть суммарного веса в пределах от 6 до 8.

Как и в предыдущем разделе, рассмотрим несколько случаев в соответствии с возможным значением t.

Случай 1: $t = 5$.

Ясно, что

$$r_1 + p_1 + p_2 \leq 14 - \sum_{i=3}^{5} p_i < 8.$$

Поэтому $r_1 + p_1 + p_2 < 6$ и $\{r_1, p_1, p_2\} \subseteq A_1$, $\{p_3, p_4, p_5\} = B_1$. Но $l(B_1) = C_1 = p_3 + p_4 + p_5$ является нижней границей для C^* и $C^* \geq l(B_1)$. ■

Случай 2: $t = 4$.

Так как $p_2+p_3+p_4 > 6$, имеем $r_1+p_1 < 8$, и поэтому $r_1+p_1 < 6$. Таким образом, $\{r_1, p_1\} \subseteq A_1$ и $\{p_2, p_3\} \subseteq B_1$, причем $p_2 + p_3 < 6$. В соответствии с определением t имеем, что

$$C_1 = \min\{r_1 + p_1 + p_4, p_2 + p_3 + p_4\} > 8. \tag{3.24}$$

Если одно из множеств S_1, S_2 содержит как минимум три элемента из P_4, то алгоритм $G_{\varepsilon}(5)$ строит оптимальное решение, так как из (3.24) следует, что $C^* \geq p_2 + p_3 + p_4 \geq C_1$. Таким образом, оптимальное решение содержит элементы из P_4 в пропорции 2:2 и справедливо $C^* \geq p_1 + p_4$.

2.1. $p_1 \geq r_1 + p_2$.

Если $p_1 + p_3 < 6$, то имеем $p_3 \in B_2$ и $p_1 < 4$. Из (3.24) получаем $r_1 + p_1 + p_4 > 8$. Следовательно, $r_1 + p_2 + p_4 > 6$. Алгоритм $G_{\varepsilon}(5, \{p_2\})$ не срабатывает, тогда $p_1 + p_4 \geq r_1 + p_2 + p_4 > 8$.

Отсюда следует, что $p_1 > 4$, получили противоречие.

С другой стороны, если $p_1+p_3 \geq 6$, то справедливо соотношение $p_1 + p_3 > 8$, так как алгоритм $G_{\varepsilon}(5, \{p_2\})$ не срабатывает. Таким образом, $p_1 > 4$. Но $p_1 \geq r_1+p_2$, поэтому $p_3 \in A_2$. Из соотношения $r_1 + p_2 + p_3 \leq 14 - p_1 + p_4 < 8$ и факта, что алгоритм $G_{\varepsilon}(5, \{p_2\})$ не сработал, следует, что $r_1 + p_2 + p_3 < 6$.

Следовательно,

$$C_2 = \min\{r_1 + p_2 + p_3 + p_4, p_1 + p_4\} \leq C^*. \qquad (3.25)$$

2.2. $p_1 < r_1 + p_2$.

Если $r_1 + p_2 + p_3 < 6$, то $p_3 \in A_2$. Комбинируя это соотношение с (3.24), имеем $p_1 > 4$. В силу факта, что алгоритм $G_\varepsilon(5, \{p_2\})$ не сработал, получаем (3.25).

Если $r_1 + p_2 + p_3 > 6$, то выполняется $r_1 + p_2 + p_3 > 8$ и $p_3 \in B_2$. В силу минимальности t для $G_\varepsilon(5)$ имеем $p_2 + p_3 < 6$. Вместе с $r_1 + p_2 + p_3 > 8$ справедлива нижняя оценка $r_1 > 2$. Тогда $r_1 + p_2 + p_4 > 6$ и $r_1 + p_2 + p_4 > 8$.

Следовательно,

$$C_2 = \min\{r_1 + p_2 + p_4, p_1 + p_3 + p_4\}. \qquad (3.26)$$

Если $\{p_3, p_4\} \not\subseteq S_1^*$, то выполняется $C^* \geq r_1 + p_2 + p_4 \geq C_2$ в силу (3.26). Если $\{p_3, p_4\} \subseteq S_1^*$, то $C^* \geq r_1 + p_3 + p_4$. Но из $r_1 > 2$ заключаем, что $r_1 + p_3 + p_4 > 6$, и в силу факта, что алгоритм $G_\varepsilon(5, \{p_3, p_4\})$ не сработал, имеем $r_1 + p_3 + p_4 > 8$. Но $C_3 \leq r_1 + p_3 + p_4$, противоречие. ■

Случай 3: t =3.

3.1. $r_1 + p_1 < 6$.

Очевидно, что $\{r_1, p_1\} \subseteq A_1$ и $\{p_2\} \subseteq B_2$ при $C_1 = \min\{r_1 + p_1 + p_3, p_2 + p_3\}$. Но $C^* \geq p_2 + p_3$.

3.2. $r_1 + p_1 > 8$.

Можно предположить, что $\max\{r_1, p_1\} < 6$, тогда выполняется соотношение $r_1 > 2$. Условие $C = 14$ гарантирует, что $p_2 + p_3 < 6$, из чего следует, что $p_3 < 3$.

В случае $r_1 > p_1$ алгоритм $G_\varepsilon(5)$ строит $\{r_1\} \subseteq A_1$ и $\{p_1, p_2\} \subseteq B_1$. Поэтому

$$C_1 = \min\{r_1 + p_3, p_1 + p_2 + p_3\}. \qquad (3.27)$$

В случае $r_1 \leq p_1$ алгоритм $G_\varepsilon(5)$ строит $\{r_1, p_2\} \subseteq A_1$ и $\{p_1\} \subseteq B_1$. Поэтому

$$C_1 = \min\{r_1 + p_2 + p_3, p_1 + p_3\}. \qquad (3.28)$$

В оптимальном решении разбиение элементов из P_3 осуществлено в пропорции 2:1, иначе в любом случае справедливо соотношение $C^* \geq C_1$. Следовательно, можно заключить, что $C^* \geq r_1 + p_3$,

из которого следует $C_1 \le C^*$ при $r_1 > p_1$.

Таким образом, предположим, что $r_1 \le p_1$. Если S_1^* содержит как минимум два элемента из P_3, то $C^* \ge r_1 + p_2 + p_3 \ge C_1$. Иначе

$$C^* \ge \min\{r_1 + p_1, p_1 + p_3\},$$

и получаем $C_1 - C^* \le (p_1 + p_3) - (r_1 + p_1) = p_3 - r_1 \le 3 - 2 = 1$.

Наконец,

$$\frac{C_1}{C^*} \le 1 + \frac{1}{C^*} \le \frac{8}{7}.$$

Этот факт завершает доказательство теоремы. Случай $t = 2$ тривиален.

3.8. Semi on-line версия задачи разбиения

В on-line версии задачи разбиения веса предметов и их количество заранее неизвестны. Они поступают один за другим. Каждый прибывший предмет необходимо тотчас же назначить множеству, и это решение не может быть изменено позднее, когда новая информация о последующих предметах станет доступной. Алгоритмы для on-line версий задачи называются on-line алгоритмами.

Задачу разбиения, отличную от on-line задачи, будем иногда называть off-line. Эффективность on-line алгоритма оценивается относительно оптимального решения off-line задачи.

On-line задачи имеют огромное практическое значение, так как часто находят реальное применение, и необходимо рассмотреть те on-line решения, которые должны быть найдены при недостатке информации или при изменении имеющейся в распоряжении.

На практике задачи часто бывают не чисто on-line или off-line, а чем-то средним. Это значит, что доступна некоторая дополнительная информация о последующих значениях или дополнительные алгоритмические возможности. Эта дополнительная информация или лучшие алгоритмические возможности позволяют усовершенствовать (улучшить) эффективность алгоритма решения on-line задачи. Мы называем задачи этого класса *semi on-line* задачами.

Рассматрим три semi on-line версии задачи разбиения. В первой задаче мы предположим, что значения поступают одно за другим и что буфер длины k доступен для сохранения k значений. Если в буфере уже находится k значений, то поступающее значение может быть или назначено (приписано) множеству или сохранено в

буфере. В последнем случае одно из значений из буфера необходимо тотчас же назначить множеству.

Во второй задаче значения необходимо передавать одно за другим, но для получения решения строится два параллельных решения. Для каждого поступающего предмета делается одна копия, и каждый из двух идентичных предметов необходимо передать множеству при каждом из двух решений до прибытия последующего предмета. Наконец, получаем лучшее из двух решений, найденных независимо.

В третьей задаче предметы должны быть приписаны множеству по их поступлению, но известен общий вес предметов. Для каждой из этих semi on-line задач мы предлагаем приближенный алгоритм. Мы показываем, что каждый алгоритм имеет нижнюю оценку, ограниченную $4/3$. Эта граница лучшая для всех случаев и не может быть улучшена никаким другим алгоритмом.

Обозначим через C^* величину оптимального решения semi on-line задачи, а через C^H — значение приближенного решения. Для краткости веса предметов будем отождествлять с предметами.

3.8.1 Наличие буфера для хранения данных

В этом разделе считаем, что предметы прибывают один за другим и что буфер длины k доступен для сохранения k предметов. Поэтому в случае, если буфер заполнен не полностью, поступающее значение может быть либо немедленно назначено множеству, либо временно помещено в буфер. Если k значений помещены в буфер, то поступающий предмет может быть либо назначен множеству, либо помещен в буфер. В последнем случае один из предметов из буфера необходимо немедленно назначить множеству. Обозначим эту задачу как $\mathcal{P}_1^k$. Рассмотрим следующий алгоритм, известный как $\mathcal{H}_1$ алгоритм. $\mathcal{H}_1$ посылает первые k значений в буфер. Пусть $s_1 = w(A_1)$ и $s_2 = w(A_2)$ — веса текущих множеств разбиения. Не умаляя общности будем считать, что $s_1 \geq s_2$. Обозначим через $b_1, \ldots, b_k$ значения, находящиеся в буфере, а через b_{k+1} — новое поступающее значение. Кроме того, положим $B = \{b_1, \ldots, b_{k+1}\}$. Если существует какое-нибудь $b \in B$, такое, что

$$s_1 + b \leq \frac{2}{3}\left(s_1 + s_2 + w(B)\right), \tag{3.29}$$

то припишем такое значение множеству A_1, т. е. множеству с наибольшим весом. Если же (3.29) не удовлетворяется даже для наименьшего $b \in B$, т. е.

$$s_1 + b > \frac{2}{3}(s_1 + s_2 + w(B)) \tag{3.30}$$

для всех $b \in B$, то назначаем произвольное значение из B множеству A_2, т. е. множеству с наименьшим весом. Конечно, если b — элемент из буфера, новое значение b_{k+1} помещается в буфер.

В следующей теореме покажем, что буфера длины 1 достаточно для достижения нижней оценки $4/3$ для алгоритма $\mathcal{H}_1$. Для краткости мы обозначим задачу с буфером длины 1 через $\mathcal{P}_1$.

Теорема 3.24. *Для алгоритма $\mathcal{H}_1$, примененного к задаче $\mathcal{P}_1$, справедливо соотношение*

$$\frac{C^H}{C^*} \leq \frac{4}{3}.$$

Доказательство. Предположим противное. Тогда существует минимальный контрпример I, т. е. отдельный случай I с минимальным количеством значений, таких, что оценка приближенного решения более чем в $4/3$ раза превышает оценку оптимального решения для I. Пусть $j \leq n-1$, и пусть C_j^* определяет величину оптимального решения для первых $j-1$ значений, назначенных множествам вместе с текущим элементом в буфере. Аналогично через C_j^H обозначим величину приближенного решения для j элементов. Пусть

$$C_j^H / C_j^* \leq 4/3 \qquad j = 1, \ldots, n-1. \tag{3.31}$$

В противном случае имеется противоречащий пример с меньшим числом предметов, учитывая только первые $j-1$ предметов плюс предмет в буфере.

Пусть $s_1 = w(A_1)$ и $s_2 = w(A_2)$ $(s_1 \geq s_2)$ — веса множеств разбиения сразу перед назначением последнего значения примера I, и пусть a — элемент в буфере. Выражение $(s_1 + s_2 + a)/2$ является очевидной нижней границей для оптимального решения. Если $s_1 + a \leq \frac{2}{3}(s_1 + s_2 + a)$, то граница $\frac{4}{3}$ была бы удовлетворена. Следовательно, из предположения контрпримера выполняется $s_1 + a > \frac{2}{3}(s_1 + s_2 + a)$, что равнозначно $s_1 + a > 2s_2$. Так, $\mathcal{H}_1$ приписывает значение a множеству A_2 Если $s_2 = 0$, все пред-

шествующие значения необходимо было приписать множеству A_1. Учитывая (3.29), получаем

$$s_1 \leq \frac{2}{3}(s_1 + a) \leq \frac{4}{3}C^*,$$

которое опровергает предположение, что I — контрпример. Так, $s_2 > 0$ в I существует по крайней мере одно более отдаленное значение, кроме a, назначенного алгоритмом $\mathcal{H}_1$ в множество с наименьшим весом, которое было не обязательно текущим (до назначения последнего значения a) меньшим множеством A_2. Обозначим через b последнее из этих отдаленных значений. Пусть s'_1, s'_2 $(s'_1 \geq s'_2)$ веса множеств перед назначением b, и пусть c — значение в буфере сразу после назначения b. Так как $\mathcal{H}_1$ пытался назначить одно из двух значений b или c в множество с большим весом без нарушения (3.29), из (3.30) получаем

$$s'_1 + b > 2(s'_2 + c) \tag{3.32}$$

и

$$s'_1 + c > 2(s'_2 + b). \tag{3.33}$$

После назначения b все значения посылаются в множество с большим весом, пока последнее значение a не будет закреплено. Следовательно,

$$s'_2 + b \geq s_2. \tag{3.34}$$

Заметим, что равенство будет выполнено в случае, если b было назначено текущему наименьшему множеству A_2. Мы требуем, чтобы значение c никогда не могло быть назначено одному из множеств до окончания алгоритма. Пусть s''_1, s''_2 $(s''_1 \geq s''_2)$ — веса множеств в таком случае.

Из (3.33) и (3.34) получаем

$$s''_1 + c \geq s'_1 + c > 2(s'_2 + b) \geq 2s_2 \geq 2s''_2.$$

Из того, что b — последнее значение, приписанное наименьшему множеству, следует, что значение c всегда остается в буфере, и фактически c идентично a. Сейчас мы рассмотрим два случая. Если $s_2 + a \geq s_1$, то из (3.32), (3.34) и $a = c$ получаем

$$s_2 + a \leq (s'_2 + a) + b < \frac{1}{2}(s'_1 + b) + b \leq \frac{3}{2}b + \frac{1}{2}s_1 \leq 2s_1.$$

Но $s_2 + a < 2s_1$ эквивалентно условию

$$s_2 + a < \frac{2}{3}(s_1 + s_2 + a) \leq C^*_n,$$

которое противоречит предположению, что I — контрпример. Если $s_2 + a < s_1$, то последний элемент не увеличивает величину решения. С другой стороны, это противоречит минимальности контрпримера. ■

Покажем, что нет детерминированного $\mathcal{A}$ алгоритма для semi on-line задачи разбиения $\mathcal{P}_1^k$ с произвольным k, который может иметь оценку относительно величины оптимального решения off-line задачи лучше, чем $4/3$. Обозначим через C^A величину целевой функции, получаемой алгоритмом $\mathcal{A}$.

Теорема 3.25. *Для любого алгоритма $\mathcal{A}$ для задачи $\mathcal{P}_1^k$ справедливо соотношение*

$$\frac{C^A}{C^*} \geq \frac{4}{3}$$

на некотором примере задачи.

Доказательство. Рассмотрим пример, в котором $a_1 = a_2 = \ldots = a_{n-1} = \delta$, $a_n = x$. До поступления n-го предмета любой алгоритм $\mathcal{A}$ должен назначить множеству A_1 или A_2 $n - k - 1$ значений, все с весом δ. В то время, когда $(n-k-1)$-й предмет назначается множеству, k предметов, все с весом δ, находятся в буфере и n-й предмет еще должен поступить. Пусть $\tilde{a}$ и $\tilde{b}$ – веса двух множеств во время, когда $n - k - 1$ значений были назначены множеству, и предположим, что $\tilde{a} \geq \tilde{b}$.

Рассмотрим два случая.

1) $\tilde{b} \leq \tilde{a}/2$. В этом случае предположим, что последнее значение имеет длину $x = 0$. Тогда $C^A \geq \tilde{a}$, если предполагать, что n нечетно, поэтому

$$C^* = (\tilde{a} + \tilde{b} + k\delta)/2 \leq (\tilde{a} + \tilde{a}/2 + k\delta)/2.$$

Следовательно,

$$\frac{C^A}{C^*} \geq \frac{2\tilde{a}}{3\tilde{a}/2 + k\delta} \to \frac{4}{3},$$

когда $n \to \infty$.

2) $\tilde{b} > \tilde{a}/2$. В этом случае предположим, что последний предмет имеет вес $x = 2\tilde{a} - \tilde{b}$. Тогда $C^A \geq \tilde{b} + x = 2\tilde{a}$. Так как $x = 2\tilde{a} - \tilde{b}$, следовательно,

$$C^* \leq \frac{\tilde{a} + \tilde{b} + x + (k+1)\delta}{2} \leq \frac{3\tilde{a} + (k+1)\delta}{2}.$$

Поэтому

$$\frac{C^A}{C^*} \geq \frac{2\tilde{a}}{3\tilde{a}/2 + (k+1)\delta/2} \to \frac{4}{3},$$

когда $n \to \infty$. ∎

Замечания:

1) Как можно видеть из теорем 3.24 и 3.25, нельзя улучшить достигнутую худшую оценку, если вместимость буфера расширить от 1 до произвольной вместимости. Для всех длин буферов мы получаем $4/3$ как лучшую возможную нижнюю границу.

2) Алгоритм не находит решения, если изменить его так, что он всегда проверяет только, может ли он упаковать *наибольший* элемент описанным выше образом. Например, случай $a_1 = a_2 = 1,\ a_3 = \ldots = a_{k+1} = 0,\ a_{k+2} = 2$ дает решение, которое на 50 % отклоняется от оптимальной величины.

3) Рассмотрим задачу разбиения с предварительным просмотром, т. е. где известны веса следующих k поступающих предметов, которые мы не можем сохранить в буфере. Для этой задачи нет алгоритма, который может гарантировать лучшую нижнюю оценку, чем для on-line задачи, а именно $3/2$. Это может быть показано посредством рассмотрения примера $a_1 = a_2 = 1,\ a_3 = \ldots = a_{k+2} = 0,\ a_{k+3} = x$. Если алгоритм назначает a_1 и a_2 различным множествам, выбираем $x = 2$. Если a_1 и a_2 находятся в одном и том же множестве, выбираем $x = 0$. В обоих случаях отношение между величиной эвристического решения и оптимальной величиной больше или равно $3/2$.

   Следовательно, возможность сохранения a_2 так долго, как это необходимо, существенна для улучшения алгоритма.

3.8.2 Два параллельных процессора

В этом разделе будем рассматривать предметы с весом a_i как задания с временем выполнения a_i. Эти задания поступают одно за другим и должны быть выполнены на компьютере, который состоит из центрального процессора и двух параллельных. Центральный

процессор создает одну копию каждого задания, как только оно поступает. Затем каждая из копий отправляется на один из двух параллельных процессоров, и каждый процессор независимо назначает задание двум различным множествам до прибытия последующего задания. После каждого назначения центральный обслуживающий элемент может отправить информацию процессорам. Наконец, выбирается лучшее из двух произведенных разбиений.

Рассмотрим задания, которые пронумерованы в порядке их поступления и отсылаются в тот момент, как только задание j выполнится процессорами как временная точка j. Временная точка нуль — временной момент до поступления первого задания. Назовем эту semi on-line задачу $\mathcal{P}_2$.

Опишем приближенный алгоритм для задачи $\mathcal{P}_2$, который обозначим как $\mathcal{H}_2$. Назовем один из процессоров *активным процессором*, а другой — *пассивным процессором*. Активный и пассивный процессоры могут обмениваться ролями в процессе выполнения алгоритма. В нулевой момент мы выбираем произвольный процессор в качестве активного. Обозначим через $P_1,\ P_2$ два множества значений, созданных активным процессором к некоторому моменту $i-1$, сразу же после этого необходимо назначить задание $i-1$ множеству P_1 или P_2.

Предположим, что $s_1 := w(P_1) \geq s_2 := w(P_2)$. Аналогично пусть $Q_1,\ Q_2$ — множества, созданные пассивным процессором с соответствующими весами $t_1 \geq t_2$.

Рассмотрим два случая. Если

$$s_2 + a_i \leq \frac{2}{3}\left(s_1 + s_2 + a_i\right), \tag{3.35}$$

активный процессор назначает задание i множеству P_2, а пассивный процессор назначает его множеству Q_1. Роли активного и пассивного процессора не меняются. В противном случае, если

$$s_2 + a_i > \frac{2}{3}\left(s_1 + s_2 + a_i\right), \tag{3.36}$$

активный процессор по-прежнему назначает задание i множеству P_2, но пассивный процессор назначает его множеству Q_2. Затем текущий активный процессор становится пассивным и наоборот.

Покажем, что эта процедура гарантирует нижнюю оценку $4/3$.

Теорема 3.26. *Нижняя оценка эвристики* $\mathcal{H}_2$, *примененной к*

задаче $\mathcal{P}_2$,

$$\frac{C^H}{C^*} \leq \frac{4}{3}.$$

Более того, оценка является неулучшаемой.

Доказательство. Определим как *точки перехода* следующие временные моменты: первая точка перехода есть время нуль, другие точки перехода — это точки, когда активный и пассивный процессоры меняются ролями. Соответствующие значения назовем *значениями перехода*. Обозначим через C^*_ℓ величину оптимального решения для разбиения заданий $a_1, \ldots, a_\ell$.

Мы собираемся показать, используя индукцию, что в любой момент времени i процессор, который играет активную роль, в следующий момент времени, т. е. когда прибывает значение a_{i+1}, производит разбиение, которое не превышает $\frac{4}{3}C^*_i$. Другими словами, разбиение с относительной ошибкой не больше $4/3$ производится пассивным процессором в каждой точке *перехода* и *активным* процессором в каждой точке *неперехода*.

Конечно, для момента нуль это утверждение истинно. Пусть $i > 0$ будет случайным, и предположим, что верхняя граница $\frac{4}{3}C^*_{i'}$ является величиной для всех $i' < i$. Обозначим через $\bar{s_1}$ и $\bar{s_2}$ $(\bar{s_1} \geq \bar{s_2})$ веса множеств текущего активного процессора в момент i до назначения значения a_i. Аналогично обозначим веса текущего пассивного процессора через $\bar{t_1}$ и $\bar{t_2}$ $(\bar{t_1} \geq \bar{t_2})$.

Предположим сначала, что a_i — неизменная величина. В этом случае

$$\bar{s_2} + a_i \leq \frac{2}{3}(\bar{s_1} + \bar{s_2} + a_i) \leq \frac{4}{3}C^*_i$$

следует непосредственно из (3.35) и из того факта, что $\bar{s_1} + \bar{s_2} + a_i \leq 2C^*_i$. Заметим далее, что

$$\bar{s_1} \leq \frac{4}{3}C^*_{i-1} \leq \frac{4}{3}C^*_i$$

как крайнее слева неравенство является просто индуктивной гипотезой для текущего активного процессора, и крайнее справа неравенство тривиально. Поэтому

$$\max(\bar{s_1}, \bar{s_2} + a_i) \leq \frac{4}{3}C^*_i,$$

приносящее утверждение для точек неперехода.

Предположим далее, что a_i — изменяемое значение. В этом

случае мы вернемся в состояние *предыдущей* точки перехода l, т. е. момент времени, когда текущий пассивный процессор меняет свою роль с активной на пассивную.

Скажем, процессор имел разбиение с весами t'_1 и t'_2 до момента времени l и с весами t'_1 и t'_2 после передачи значения a_l. Так как l — точка перехода,

$$s'_2 + a_l > 2s'_1,$$

значит, $t'_1 = s'_2 + a_l$, $t'_2 = s'_1$ и

$$t'_2 < \frac{1}{2}t'_1.$$

Так как процессор остается пассивным до следующей точки перехода i, он помещает все значения a_j $(l < j < i)$ в первое множество разбиения, т. е. $\bar{t_1} \geq t'_1 > 2t'_2 = 2\,\bar{t_2}$. Заметим, что $\bar{s_1} + \bar{s_2} = \bar{t_1} + \bar{t_2}$ удовлетворяет неравенству

$$\bar{t_1} > \frac{2}{3}(\bar{s_1} + \bar{s_2}) \qquad \text{и} \qquad \bar{t_2} < \frac{1}{3}(\bar{s_1} + \bar{s_2}). \tag{3.37}$$

Напомним теперь, что a_i — изменяемое значение, и поэтому

$$a_i > 2\,\bar{s_2} - \bar{s_1} \geq \bar{s_2} \geq \frac{1}{2}(\,\bar{s_1} + \bar{s_2}\,).$$

Следовательно,

$$C_i^* \geq \frac{1}{2}(\bar{s_1} + \bar{s_2} + a_i) > \frac{3}{4}(\bar{s_1} + \bar{s_2}).$$

Так как $\bar{t_1} \leq \bar{s_1} + \bar{s_2}$ и a_i помещены во второе множество разбиения, первое не превосходит $\frac{4}{3}C_i^*$ в i-й момент времени. Чтобы доказать верхнюю границу $\frac{4}{3}C_i^*$ для второго множества, которое будет иметь вес $\bar{t_2} + a_i$, разделим два простых случая. Если $a_i \leq \bar{s_1} + \bar{s_2}$, тогда из (3.37) следует

$$\frac{\bar{t_2} + a_i}{C_i^*} < \frac{\frac{1}{3}(\bar{s_1} + \bar{s_2}) + a_i}{\frac{1}{2}(\bar{s_1} + \bar{s_2} + a_i)},$$

и правая часть неравенства достигает максимума, когда a_i наибольшее, т. е. $a_i = \bar{s_1} + \bar{s_2}$, в итоге получаем

$$\bar{t_2} + a_i < \frac{4}{3}C_i^*.$$

С другой стороны, если $a_i > \bar{s_1} + \bar{s_2}$, очевидно имеем $C_i^* = a_i$ и также $\bar{t_2} < \frac{1}{3}C_i^*$ по (3.37); поэтому следует $\bar{t_2} + a_i < \frac{4}{3}C_i^*$. ■

Как и в предыдущем разделе, мы можем показать, что не существует semi on-line алгоритма для задачи $\mathcal{P}_2$ с нижней оценкой

лучше, чем $4/3$.

Теорема 3.27. *Любой алгоритм $\mathcal{A}$ для задачи $\mathcal{P}_2$ таков, что*

$$\frac{C^A}{C^*} \geq \frac{4}{3}$$

на некоторых примерах задач.

Доказательство. Допустим, существует алгоритм с нижней оценкой лучшей, чем $4/3$. Рассмотрим случай, где первые четыре значения $a_1, \ldots, a_4$ все равны единице. Алгоритм порождает два не обязательно различных разбиения (назовем их B_1, B_2 и C_1 соответственно) из $a_1, \ldots, a_4$. Значение a_4 может быть последним значением, которое будет обработано.

Поэтому существует одно разбиение, скажем, B_1, B_2 с весами $w(B_1) = w(B_2) = 2$. Но может оказаться, что поступит задание a_5 размера 2, и оно будет последним. В этом случае величина оптимального решения равна 3. Следовательно, должно быть разбиение C с весами 1 и 3. После поступления пятого предмета a_5 с весом 2 разбиение (B_1, B_2) преобразуется в разбиение с весами 2 и 4.

Так как a_5 может быть последним элементом, то другое разбиение должно иметь веса $w(C_1) = w(C_2) = 3$. Но тогда пусть поступает предмет a_6 с весом 6. Понятно, что величина оптимального решения равна 6, в то время как любое из разбиений имеет вес не меньше 8. Получили противоречие. ∎

3.8.3 Известная общая сумма

В этом разделе мы предположим, что значения прибывают одно за другим и что общая сумма $w(A)$ заранее известна. Без ограничения общности упорядочим веса значений таким образом, что $w(A) = 2$. Более того, значения пронумерованы в порядке поступления. Назовем эту semi on-line задачу $\mathcal{P}_3$. Приближенный алгоритм для задачи $\mathcal{P}_3$, который обозначим кратко как $\mathcal{H}_3$, работает следующим образом: $\mathcal{H}_3$ назначает поступающее значение множеству A_1, пока существует значение j, такое, что $s + a_j > 2/3$, где s обозначает текущий вес A_1 до прибытия значения j. Если $s + a_j \leq 4/3$, эвристика $\mathcal{H}_3$ назначает j множеству A_1 и оставляет значения в множестве A_2. В противном случае $\mathcal{H}_3$ назначает j множеству A_2 и оставляет значения в множестве A_1.

Теорема 3.28. *Для решения алгоритма* $\mathcal{H}_3$, *примененного к задаче* $\mathcal{P}_3$, *справедливо соотношение*

$$\frac{C^H}{C^*} \leq \frac{4}{3}.$$

Доказательство. Обозначим через s вес множества A_1 перед поступлением первого значения j, такого, что $s + a_j > 2/3$. Конечно, $s \leq 2/3$. Напомним, что $w(A) = 2$ и, следовательно, $C^* \geq 1$. Если $s + a_j \leq 4/3$, то $\mathcal{H}_3$ назначает j множеству A_1, и $w(A_2) \leq 4/3$ справедливо в силу $w(A_1) \geq s + a_j > 2/3$. Если $s + a_j > 4/3$, то общий вес w значений, прибывших после j, меньше чем $2/3$. Это значит, что когда $s \leq 2/3$, мы получаем $w(A_1) = s + w < 4/3$ и $w(A_2) = a_j$. Тогда, если $a_j > s + w$, то алгоритм находит оптимум. В противном случае оценки справедливы. ∎

Теорема 3.29. *Любой алгоритм* $\mathcal{A}$ *для* $\mathcal{P}_3$ *таков, что*

$$\frac{C^A}{C^*} \geq \frac{4}{3}$$

на некоторых примерах задач.

Доказательство. Рассмотрим множество примеров, в которых четыре значения весов $1/3, 1/3, a, b$ прибывают так, что $a+b = 4/3$. Алгоритм может работать на первых двух значениях двумя способами: оба значения назначаются одному и тому же множеству или двум различным множествам. Во время прибытия первого и второго значений нет информации о величинах a и b, поэтому решение очевидно независимо от таких величин.

Первый случай: оба значения назначаются одному и тому же множеству. Тогда возьмем $a = b = 2/3$. Результат работы алгоритма равен $C^A \geq 4/3$, а $C^* = 1$. Поэтому существует такой пример, что $C^A/C^* \geq 4/3$.

Второй случай: первые два значения назначаются различным множествам. Тогда возьмем $a = 1$ и $b = 1/3$. Результат работы алгоритма равен $C^A \geq 4/3$, а $C^* = 1$. И снова существует такой пример, что $C^A/C^* \geq 4/3$.

∎

3.9. On-line алгоритмы для задачи k-упаковки

В классической *задаче упаковки* (BP) имеется список $L = (a_1, a_2, \ldots, a_m)$, состоящий из действительных чисел, называющихся *предметами*, из интервала $(0, 1]$ и произвольно много контейнеров единичной вместимости. Требуется так упаковать все предметы в контейнеры, чтобы количество использованных контейнеров было минимально. В on-line версии задачи ($onkBP$) предметы должны паковаться в том порядке, в котором они поступают. Новый предмет a_n пакуется на основе информации, полученной о весах предметов $a_1, \ldots, a_{n-1}$. Информация о последующих предметах недоступна, а ранее размещенные предметы не могут быть переупакованы. При этом в один контейнер не может быть упаковано более k предметов.

Для приближенного алгоритма H и списка L предметов пусть $C^H(L)$ обозначает количество контейнеров, требуемых для упаковки предметов по алгоритму H. Пусть $C^{OPT}(L)$ соответствует минимально возможному числу контейнеров, необходимых для упаковки всех предметов. (Если L ясно из контекста, то оно будет опускаться в обозначениях.) Тогда *асимптотическая гарантированная оценка* R_H приближенного алгоритма H определяется как

$$R_H = \lim_{N\to\infty} \sup_L \{\frac{C^H(L)}{C^{OPT}(L)} \,|\, C^{OPT}(L) = N\}.$$

Известно, что для адаптированного (kFF) алгоритма, использующего идею "в первый подходящий", справедлива следующая теорема.

Теорема 3.30.

$$R_{kFF} \leq 2.7 - \frac{12}{5k}.$$

Ниже будут представлен приближенный алгоритм A_2, и будет показано, что оценка алгоритма A_2 равна 2. Более того, предлагаются алгоритмы для $k = 2$ и $k = 3$, причем при $k = 2$ алгоритм является наилучшим из возможных.

Алгоритм основан на разбиении контейнеров на 2 типа. Первый тип закрытый, а второй – открытый. Контейнер из закрытого типа характеризуется тем свойством, что в него не будут помещаться вновь поступающие предметы. К этому типу мы отнесем контейнеры, обладающие следующими свойствами:

- это контейнер B, для которого $l(B) \geq 1/2$ и $c(B) \geq k/2$;

- пара контейнеров $B1$, $B2$, для которых выполняется $l(B1)+l(B2) \geq 1$ и $c(B1)+c(B2) \geq k$.

Остальные контейнеры относятся к открытому типу, причем они разбиваются на следующие классы:

- контейнер из класса 1 удовлетворяет $l(B) \geq 1/2$ и $c(B) < k/2$;
- контейнер из класса 2 удовлетворяет $l(B) < 1/2$ и $c(B) < k-1$;
- контейнер из класса 3 удовлетворяет $l(B) < 1/2$ и $c(B) = k-1$.

Алгоритм работает следующим образом.

а) Пусть класс 1 не пуст. Пытаемся поместить поступивший предмет a в наименее загруженный контейнер B_1 из класса 1. Если предмет a поместить удалось, то переходим к рассмотрению следующего предмета. Если предмет a не помещается в B_1, то пытаемся поместить текущий предмет в наиболее загруженный контейнер B_3 из класса 3, если класс 3 не пуст. Возможны две ситуации.

1. Предмет a помещается в B_3. В этом случае предмет a помещается в B_3, при этом пара контейнеров B_1 и B_3 становится закрытой.

2. Предмет a не помещается в B_3. В этом случае предмет помещается в новый контейнер B, при этом пара контейнеров B и B_3 становится закрытой.

Если a не помещается в B_1, а класс 3 пуст, то предмет a помещается в наименее загруженный контейнер из класса 2 или в новый контейнер.

б) Пусть класс 1 пуст. Пытаемся поместить поступивший предмет a в наиболее загруженный контейнер B_3 из класса 3.

Если a не помещается в B_3, то предмет a помещается в новый контейнер B, при этом пара контейнеров B и B_3 становится закрытой. В случае, если $l(3)+a \geq 1/2$, то предмет a помещается в B_3, при этом контейнер B_3 становится закрытым.

Если же $l(3)+a < 1/2$, то предмет помещается в наименее загруженный контейнер из класса 2, если он не пуст, или помещается в новый контейнер, если класс 2 пуст.

в) Пусть классы 1 и 3 пусты. Пытаемся поместить поступивший предмет в наименее загруженный контейнер B_2 из класса 2. Если a не помещается в B_2, то предмет помещается в новый контейнер B.

г) Пусть классы 1, 2 и 3 пусты. Предмет a помещается в новый контейнер B.

Обозначим через K_i количество контейнеров из класса i, $i = 1, 2, 3$.

Лемма 3.14. *Для каждого шага алгоритма справедливы следующие соотношения:*

$$K_2 \leq 1, \tag{3.38}$$

$$K_3 = 0 \quad \textit{или} \quad K_1 + K_2 \leq 1. \tag{3.39}$$

Доказательство будет проводиться индукцией по числу упакованных предметов. Предположим, что оба соотношения выполняются до поступления предмета a. При поступлении предмета a возможны следующие ситуации.

1. Предмет a попал в контейнер B_1 из класса 1. В этом случае B_1 остается или в том же классе (при этом значение K_i не изменяется), или относится к закрытому типу. Понятно, что значение K_i при этом не увеличивается, поэтому условия выполняются.

2. Предмет a попал в контейнер B_3 из класса 3. В этом случае значение K_2 не изменяется, а значения K_1 и K_3 не могут увеличиться.

Действительно, если класс 1 не пуст, то предмет a не помещается в контейнер B_1. Поэтому пара контейнеров B_1, B_3 относится к закрытому типу, при этом значения K_1 и K_3 уменьшаются.

Если класс 1 был пуст, то контейнер B_3 относится к закрытому классу, а значение K_3 уменьшается.

3. Предмет a попал в контейнер B_2. Понятно, что значение K_2 не увеличивается.

4. Предмет a попал в новый контейнер B. Если класс 3 не пуст, то пара контейнеров B, B_3 относится к закрытому типу. Пусть класс 3 пуст, а B_2 является единственным контейнером из класса 2. Покажем, что новый контейнер B будет принадлежать классу 1.

Действительно, предмет a не помещается в контейнер B_2, поэтому объем предмета a не меньше $1/2$, следовательно, новый

контейнер B относится к классу 1.

Если оба класса 2 и 3 были пусты, то новый контейнер B может быть из класса 2, но и в этом случае $K_2 = 1$. ■

Теорема 3.31. *Описанный алгоритм имеет гарантированную оценку, равную* 2.

Доказательство. Для данной задачи справедливы следующие две нижние оценки величины оптимального решения.

С одной стороны, величина оптимального решения не меньше суммарного объема поступивших предметов с учетом единичной вместимости контейнеров.

С другой стороны, величина оптимального решения удовлетворяет условию

$$K^* \geq N/k,$$

где K^* – величина оптимального решения, а N – общее количество поступивших предметов.

Из леммы 3.14 следует, что в результате работы алгоритма используемые контейнеры будут заполнены в среднем не меньше чем наполовину, либо в каждом контейнере будет находиться в среднем не менее $k/2$ предметов. ■

3.10. Наилучший алгоритм для задачи ($on2BP$)

В этой части будет описан алгоритм $\boldsymbol{A}$ для ($on2BP$). Введем некоторые обозначения, необходимые в дальнейшем. Предмет будет называться *маленьким*, если его вес не превосходит $\frac{1}{2}$. В противном случае предмет будем называть *большим*. Рассмотрим конфигурацию упаковки непосредственно после размещения предмета a_i. Непустой контейнер будем относить к *типу* X^i, если он содержит ровно один маленький предмет; к *типу* Y^i, если он содержит большой и маленький предметы; к *типу* Z^i, если он содержит два маленьких предмета; к *типу* U^i, если он содержит только большой предмет.

Нетрудно видеть, что упаковка больших предметов очевидна. Поэтому сосредоточим внимание на контейнерах с маленькими предметами X, Y и Z. Пусть количество контейнеров каждого

типа равно x^i, y^i, z^i и u^i соответственно. Индекс i для краткости будет опускаться. Определим также величину $\rho = \sqrt{2} - 1$.

На каждом шаге упаковки алгоритм пытается удовлетворить требованию:

$$\lfloor \rho z \rfloor \leq x + y \leq \lfloor \rho z \rfloor + 2. \tag{3.40}$$

Будем говорить, что предмет *A-упакован* (*упакован* соответственно) в множество S контейнеров, если он упакован в один из этих контейнеров согласно правилу *в лучший подходящий*, когда $|S| \geq 2$ (когда $S \neq \emptyset$ соответственно), или открывается новый контейнер в случае невозможности его упаковки, или $|S| \leq 1$ ($S = \emptyset$ соответственно).

Алгоритм A

[Шаг 1.] Если текущий предмет является первым маленьким, то помещаем его в пустой контейнер.

[Шаг 2.] Если текущий предмет - большой, то пакуем предмет в контейнер типа X^i.

[Шаг 3.] Если текущий предмет - маленький, и если

$$x^i + y^i < \lfloor \rho z^i \rfloor + 2,$$

то помещаем предмет в контейнер типа U^i.

[Шаг 4.] Если текущий предмет - маленький, и если

$$x^i + y^i \geq \lfloor \rho z^i \rfloor + 2,$$

то помещаем предмет в контейнер типа X^i, если $x^i \geq 2$, и помещаем в контейнер типа U^i в противном случае.

Очевидны следующие леммы.

Лемма 3.15. *Для всех i, $x^i \geq 1$. Первый маленький предмет в любом контейнере типа Z^i не меньше минимального предмета среди x^i предметов в контейнерах типа X^i. Аналогично, первый предмет a_k в любом контейнере типа Y^i не меньше минимального предмета среди x^i предметов в контейнерах типа X^i, если a_k маленький, и в контейнерах типа U^i, если a_k большой.*

Лемма 3.16. *Нарушение условия (3.40) эквивалентно соотношению $x + y > \lfloor \rho z \rfloor + 2$. Упаковка на шаге 3 никогда не является результатом такой ситуации.*

Лемма 3.17. *Если $x \geq 3$, то выполняется условие (3.40).*

Доказательство. Пусть $a_{i_1}, \ldots, a_{i_x}$ соответствуют минимальным предметам, находящимся в контейнерах типа X, где $i_1 < \cdots < i_x$. Если условие (3.40) не выполняется, то a_{i_x} пакуется на шаге 4 согласно лемме **??**. Однако согласно описанию алгоритма предмет a_{i_x} должен был быть назначен в контейнер типа X^{i_x-1}, как и предметы a_{i_1} и a_{i_2}. Получили противоречие.

Теорема 3.32. *Для алгоритма A справедливо соотношение*

$$C^H \le (1+\rho)C^{OPT} + 3.$$

Доказательство. Так как половина от общего числа упакованных предметов является нижней границей C^{OPT}, то получаем

$$C^{OPT} \ge \frac{u+x}{2} + y + z. \tag{3.41}$$

Рассмотрим два случая.

Случай 1: $x \ge 3$.

Согласно алгоритму каждый из предметов, исключая, возможно, один, находящийся в контейнере типа X, не помещается в отдельный контейнер вместе с предметом из любого контейнера типа U. Согласно лемме 3.15, имеется по крайней мере $x + z - 1$ маленьких предметов, каждый из которых не пакуется с любым из больших предметов из контейнеров типа U. Учитывая y больших предметов в контейнерах типа Y, получаем

$$C^{OPT} \ge u + \frac{x+y+z}{2} = (u+x) + \frac{y+z}{2} - \frac{1+x}{2},$$

что вместе с фактом $x + y \le \rho\, z + 2$ согласно лемме 18 влечет

$$\begin{aligned} C^{OPT} \quad &\ge (u+x) + \tfrac{y+z}{2} - \tfrac{1+x}{2} \ge (u+x) + \tfrac{y+z}{2} + \tfrac{y-\rho z}{2} - \tfrac{3}{2} \ge \\ &\ge (u+x) + \tfrac{1-\rho}{2}(y+z) - 2. \end{aligned} \tag{3.42}$$

Неравенство (3.42) плюс $(1+\rho)$ неравенств (3.41) дают

$$(2+\rho)C^{OPT} \ge \frac{3+\rho}{2}(u+x+y+z) - 3 = \frac{3+\rho}{2}C^A - 2,$$

что эквивалентно $C^A \le (1+\rho)C^{OPT} + 2(1-\rho)$.

Случай 2: $x \le 2$.

Так как выполняется

$$x + y \ge \lfloor \rho\, z \rfloor > \rho\, z - 1$$

в силу (3.40) и леммы **??**, из (3.41) вытекает

$$C^{OPT} \geq \frac{u+x+y}{2} + \frac{1}{2}y + z \geq \frac{u+x+y}{2} + \frac{2+\rho}{2}z - \frac{x+1}{2}. \quad (3.43)$$

С другой стороны, так как все $u+y$ больших предметов должны паковаться раздельно, имеем

$$C^{OPT} \geq u + y = (u+x+y) - x. \quad (3.44)$$

Суммируя 2 раза (3.43) и $(1+\rho)$ раз (3.44), получаем

$(3+\rho)C^{OPT} \geq (2+\rho)(u+x+y+z) - (2+\rho)x - 1 \geq (2+\rho)C^A - (5+2\rho)$,

что эквивалентно $C^A \leq (1+\rho)C^{OPT} + (2+\rho)$. ∎

3.10.1 Нижняя граница

Пусть дан произвольный on-line алгоритм H, тогда согласно определению существует константа $N_H \geq 1$, такая, что для любого набора предметов L

$$C^H(L) \leq R_H\, C^{OPT}(L), \quad (3.45)$$

где $C^{OPT}(L) \geq N_H$.

Теорема 3.33. *Любой on-line алгоритм для задачи ($o2BP$) имеет гарантированную оценку как минимум* $\sqrt{2}$.

Доказательство. Рассмотрим произвольный on-line алгоритм H. Пусть $N = 2N_H$, где N_H выбрано, как описано выше. Пусть реализуется one-by-one список из k_1 $(1 \leq k_1 \leq N)$ маленьких предметов с весами

$$a_{1,k} = \frac{1}{2} - \frac{k}{4N}, \quad k = 1, \ldots, k_1,$$

где $k_1 = \max\left\{k : 1 \leq k \leq N \text{ and } z^k = 0\right\}$. Очевидно, что k_1 определено корректно. Предположим, что для некоторого $s \geq 1$ имеется взаимно-однозначная реализация $n_s = \sum_{i=1}^{s} k_i$ маленьких предметов веса $a_{i,k}$ $(k = 1, \ldots, k_i,\ i = 1, \ldots, s)$, которые определены следующим образом: для $i = 0, \ldots, s-1$

$$a_{i+1,k} = a_{i,k_i-1} - \frac{k}{4N^{i+1}}, \quad k = 1, \ldots, k_{i+1}, \quad (3.46)$$

где

$$k_{i+1} = n_i + \max\left\{k : 1 \leq k \leq N - n_i \text{ and } z^{n_i+k-1} = z^{n_i}\right\} \quad (3.47)$$

и $k_0 = 1$, $a_{0,0} = 1/2$. Если $n_s = N$, то stop. Иначе продолжается реализация для k_{s+1} маленьких предметов веса $a_{s+1,k}$, $k = 1, \ldots, k_{s+1}$, которые определяются как в (3.46) и (3.47), положив $i = s$. Не умаляя общности будем считать, что $n_s = N$. Список L_0 из N маленьких предметов имеет два свойства: веса предметов удовлетворяют $1/4 \le a_{i,k} \le 1/2 - 1/(4N)$, и веса вторых предметов во всех контейнерах типа Z удовлетворяют

$$a_{1,k_1} < a_{2,k_2} < \cdots < a_{s,k_s},$$

и они строго меньше, чем каждое из других $N - s$ предметов. Теперь можно показать противоречие. Если реализуется список L' из s больших предметов с весами $1 - a_{s,k_s} > 1/2$, то оптимальная упаковка общего списка L_0L' равна $C^{OPT}(LL') = s + (N - s)/2 = (N + s)/2$, а $C^H = N$. Поэтому выполняется

$$\frac{C^H(L_0L')}{C^{OPT}(L_0L')} = \frac{2N}{N+s}. \tag{3.48}$$

С другой стороны, вместо L', если дальше реализовать список L'' из N больших предметов с размерами $1/2 + 1/(4N)$, то любое оптимальное решение $C^{OPT} = N$, а $C^H = N + s$. Следовательно, выполняется

$$\frac{C^H(L_0L'')}{C^{OPT}(L_0L'')} = \frac{N+s}{N}. \tag{3.49}$$

Комбинируя (3.48), (3.49) и учитывая (3.45), получаем

$$R_H \ge \max\left\{\frac{2N}{N+s}, \frac{N+s}{N}\right\}.$$

Правая часть минимальна и равна $\sqrt{2}$ при $s = (\sqrt{2} - 1)N$, что и завершает доказательство теоремы. ■

3.11. Приближенный алгоритм для ($o n 3BP$)

Алгоритм H для ($o3BP$) работает следующим образом. Предметы разбиваются на три класса в соответствии с весом: $A =]0, \frac{1}{3}]$, $B =]\frac{1}{3}, \frac{1}{2}]$ и $C =]\frac{1}{2}, 1]$. Алгоритм H держит *активными* три контейнера B_A, B_B, и B_C. Предмет из класса A (соответственно B и C) упаковывается по схеме Next-Fit в контейнер B_A (соответственно B_B и B_C). Как только активный контейнер B_A (B_B и B_C) содержит 3 (соответственно 2 и 1) предмета, он *блокируется*, а новый контейнер становится активным.

Покажем, что гарантированная оценка предлагаемого алгоритма равна $\frac{5}{3}$. Назначим для каждого A-предмета x вес $w(x) = \frac{1}{3}$, для каждого B-предмета x вес $w(x) = \frac{1}{2}$, а для каждого C-предмета x вес $w(x) = 1$. Вес множества предметов будет равен сумме весов предметов в множестве.

Рассмотрим список предметов L. По определению алгоритм H назначает вес 1 для каждого блокированного контейнера. Имеется не более двух непустых контейнеров с минимальным весом как минимум $\frac{5}{6}$. Следовательно,

$$w(L) \geq C^H(L) - 2 + \frac{5}{6} \geq C^H(L) - \frac{7}{6}. \tag{3.50}$$

Нетрудно заметить, что максимальный вес контейнера не превышает $\frac{5}{3}$ (один A-предмет, один B-предмет и один C-предмет). Следовательно,

$$w(L) \leq \frac{5}{3} C^{OPT}(L). \tag{3.51}$$

Комбинируя неравенства (3.50) и (3.51), получаем оценку $\frac{5}{3}$.

Нижняя граница оценки $\frac{3}{2}$ для ($o3BP$) может быть доказана, если рассмотреть три списка из n предметов веса ε, $\frac{1}{3} + \varepsilon$ и $\frac{1}{2} + \varepsilon$ соответственно (см. [Ьуао80]). Таким образом, справедлива следующая теорема.

Теорема 3.34. *Существует алгоритм для задачи ($o3BP$) с гарантированной оценкой $\frac{5}{3}$. Любой on-line алгоритм для ($o3BP$) имеет гарантированную оценку как минимум $\frac{3}{2}$.*

Глава 4

Построение вполне полиномиальных ε-приближенных алгоритмов

В этом разделе описывается метод построения эффективных ε-приближенных алгоритмов для задач математического программирования с сепарабельными целевыми функционалами. Метод основан на применении алгоритма динамического программирования к релаксированной задаче, получаемой из исходной задачи путем специального округления значений целевого функционала и переменных.

Рассмотрим задачу минимизации либо максимизации функционала $F(x)$ при условии $x \in X$, где X - некоторое конечное множество.

Пусть F^* обозначает оптимальное значение целевого функционала этой задачи и пусть F^H обозначает значение целевого функционала для элемента $x \in X$, полученного при помощи некоторого приближенного алгоритма H. Предположим, что $F^* > 0$. Алгоритм H называется *ε-приближенным алгоритмом*, если для любого заданного числа $\varepsilon > 0$ и любого примера задачи он отыскивает такой элемент $x \in X$, что $|F^* - F^H| \leq \varepsilon F^*$. Если ε-приближенный алгоритм имеет временную сложность, которая ограничена некоторым полиномом от длины записи входных данных задачи в двоичном алфавите, то он называется *полиномиальным ε-приближенным алгоритмом*. Если к тому же его временная сложность ограничена некоторым полиномом от $1/\varepsilon$, то он называется *вполне полиномиальным ε-приближенным алгоритмом*.

Один из известных подходов к разработке вполне полиномиальных ε-приближенных алгоритмов, предложенный для решения

булевых задач линейного программирования с одним ограничением, т.е. для задач типа *задачи о рюкзаке*, состоит в округлении коэффициентов целевого функционала. Этот подход может быть обобщен для решения более сложных задач математического программирования с сепарабельными целевыми функционалами и переменными, принимающими значения из заданного конечного множества, при помощи следующих основных идей. Первая идея состоит в округлении значений каждой отдельной компоненты целевого функционала. Это позволяет разработать вполне полиномиальные ε-приближенные алгоритмы для задач с целевыми функционалами, более сложными, чем линейные. Вторая идея состоит в рассмотрении округленных значений переменных, что позволяет не ограничиваться задачами, в которых переменные принимают значения 0 и 1. Ниже этот более общий подход описывается на примерах задач минимизации, называемых *минисумной* и *минимаксной задачами.*

Далее формулируется минисумная задача, и приводится ε-приближенный алгоритм K_ε ее решения. Алгоритм K_ε является вполне полиномиальным ε-приближенным алгоритмом, если определены подходящие нижние и верхние оценки оптимального значения целевого функционала. Аналогичный подход применяется для минимаксной задачи. Описывается также процедура улучшения нижних и верхних оценок оптимального значения целевого функционала, основанная на применении некоторого приближенного алгоритма.

4.1. Динамическое программирование

Динамическое программирование широко используется для решения различных задач распознавания и экстремальных задач. Ряд эффективных полиномиальных и псевдополиномиальных алгоритмов динамического программирования разработан для решения задач оптимального планирования.

Динамическое программирование является универсальным методом решения задач оптимизации и распознавания. При этом задача обычно формулируется в терминах перевода некоторой системы из *начального состояния* в *конечное*. Множество всех состояний разбито на *стадии*, и требуется перевести систему из заданного начального состояния s_0 начальной стадии 0 в заданное конечное

состояние s_{r+1} конечной стадии $r+1$. На каждой из r промежуточных стадий система может находиться в одном из заданных промежуточных состояний. Для каждого состояния s_k стадии k, $k=1,\ldots,r+1$, известно множество состояний $S(s_k)$ стадии $k-1$, из которых система может быть переведена в данное состояние. Для любого состояния s_1 первой стадии мы имеем $S(s_1)=\{s_0\}$.

Рассмотрим последовательность состояний $s_0, s_1, \ldots, s_{r+1}$, где s_k является состоянием стадии k, $k=1,\ldots,r$. Если эта последовательность обладает таким свойством, что $s_{l-1} \in S(s_l)$ для $l=1,\ldots,k$, то она называется *траекторией* перевода системы из состояния s_0 в состояние s_k. Если $k=r+1$, то последовательность называется *полной траекторией*, а если $k \le r$, – *частичной траекторией*.

Стоимость траектории $s_0, s_1, \ldots, s_k$, $k \le r+1$, оценивается значением $F(s_k)$ функционала F, который обычно определяется *рекуррентным соотношением* вида

$$F(s_k) = \Phi[F(s_{k-1}), s_{k-1}, s_k],\ k=1,\ldots,r+1, \tag{4.1}$$

где Φ – неубывающий по первому аргументу функционал, и $F(s_0) = const$.

Может существовать несколько полных траекторий. Задача состоит в отыскании *оптимальной траектории* $s_0, s_1^*, \ldots, s_r^*, s_{r+1}$ с минимальным значением $F(s_{r+1})$, которое будем обозначать через F^*.

Рассмотрим некоторое фиксированное состояние s_l стадии l и множество всех частичных траекторий вида $s_0, s_1, \ldots, s_l$. Для каждой из этих траекторий вычислим $F(s_l)$. Обозначим минимум из этих величин через $F^*(s_l)$. Если не существует ни одной траектории, переводящей систему из состояния s_0 в состояние s_l, то полагаем $F^*(s_l)=\infty$.

Поскольку функционал Φ не убывает по первому аргументу, то

$$F^*(s_k) = \min_{s_{k-1} \in S(s_k)} \{\Phi[F^*(s_{k-1}), s_{k-1}, s_k]\},\ k=2,\ldots,r+1, \tag{4.2}$$

где $F^*(s_1)=F(s_1)$ и $F^*(s_k)=\infty$, если $S(s_k)=\phi$.

Алгоритм построения оптимальной траектории может быть описан следующим образом. Используя (4.1), вычисляем $F^*(s_1) = F(s_1)$ для всех состояний s_1 первой стадии. Затем, используя

(4.2), вычисляем $F^*(s_k)$ для всех состояний s_k стадии k, $k = 2, \ldots, r+1$. Если $F^*(s_{r+1}) = \infty$, то не существует полной траектории для перевода системы из начального состояния s_0 в конечное состояние s_{r+1}. Если хотя бы одна полная траектория существует, то $F^* = F^*(s_{r+1})$ и $s_0, s_1^*, \ldots, s_r^*, s_{r+1}$ является соответствующей оптимальной траекторией. Здесь s_{k-1}^* – состояние, минимизирующее правую часть уравнения (4.2) при $k = r+1,\ r, \ldots, 2$. Отметим, что таких состояний может оказаться несколько. В качестве s_{k-1}^* выбирается одно из них, остальные исключаются из рассмотрения.

Ниже приводится пример, иллюстрирующий основные идеи динамического программирования.

Рассмотрим задачу $1/d_j = d/\sum w_j U_j$ построения расписания обслуживания n требований одним прибором, в которой требуется минимизировать взвешенное число запаздывающих требований. Для каждого требования j задана длительность обслуживания p_j, вес w_j и директивный срок $d_j = d$, одинаковый для всех требований. Все параметры являются неотрицательными целыми числами. Расписание полностью определяется последовательностью обслуживания требований. При заданной последовательности $(i_1, \ldots, i_n)$ легко определить момент завершения обслуживания каждого требования i_j: $C_{i_j} = \sum_{k=1}^{j} p_{i_k}$. Напомним, что переменные U_j принимают следующие значения: $U_j = 1$, если требование j является запаздывающим ($C_j > d$), и $U_j = 0$, если j является ранним ($C_j \le d$).

Поскольку все директивные сроки одинаковы, и конкретные моменты завершения обслуживания запаздывающих и ранних требований не оказывают влияния на значение целевого функционала, существует оптимальная последовательность требований, в которой все ранние требования упорядочены произвольно и все запаздывающие требования упорядочены произвольно и расположены после последнего раннего требования. Тогда задача $1/d_j = d/\sum w_j U_j$ может быть сформулирована как целочисленная задача минимизации $\sum_{j=1}^{n} w_j U_j$ при условиях $\sum_{j=1}^{n} p_j(1 - U_j) \le d$ и $U_j \in \{0, 1\},\ j = 1, \ldots, n$. Пусть $U^* = (U_1^*, \ldots, U_n^*)$ – оптимальный вектор для этой задачи и W^* – соответствующее значение целевого функционала.

Эта задача, в свою очередь, может быть интерпретирована следующим образом. Предположим, что некоторая система должна

быть переведена из начального состояния $s_0 = (0,0)$ в конечное состояние s_{n+1}. Имеется n промежуточных стадий, где на стадии k имеется $d+1$ состояние $s_k = (k,a)$, $a = 0,1,\ldots,d$. Система находится в состоянии (k,a), если существует расписание для требований $1,\ldots,k$ такое, что суммарная длительность обслуживания ранних требований равна a. Мы также говорим, что такое расписание соответствует состоянию (k,a). Параметры, определяющие состояние (в нашем примере это k и a), называются *переменными состояния*, а множество всех состояний называется *пространством состояний*.

На первой стадии система из начального состояния $(0,0)$ может быть переведена в состояние $(1,0)$, где требование 1 является запаздывающим, либо в состояние $(1,p_1)$, где требование 1 является ранним, если $p_1 \le d$. Рассмотрим некоторое состояние (k,a) стадии k, $k \in \{2,\ldots,n\}$. В это состояние можно перейти только из состояния $(k-1,a)$, если требование k является запаздывающим, либо из состояния $(k-1,a-p_k)$, если требование k является ранним. Будем полагать, что в конечное состояние s_{n+1} можно перейти из любого состояния (n,a) стадии n.

Определим функционал $F(k,a)$ как минимальное взвешенное число запаздывающих требований, вычисленное для расписаний, соответствующих состоянию (k,a). Положим $F(0,0) = 0$ и $F(k,a) = \infty$ для всех $(k,a) \ne (0,0)$. Если система переходит из состояния $(k-1,a)$ в состояние (k,a), то требование k является запаздывающим, и мы имеем $F(k,a) = F(k-1,a)+w_k$. Если система переходит из состояния $(k-1,a-p_k)$ в состояние (k,a), то требование k является ранним, и мы имеем $F(k,a) = F(k-1,a-p_k)$. Здесь $k \in \{1,\ldots,n\}$.

Если система переходит из состояния (n,a) в конечное состояние s_{n+1}, то полагаем

$$F(s_{n+1}) = \min\{F(n,a) | a \le d\}.$$

Из определения функционала $F(k,a)$ следует, что $F^*(k,a) = F(k,a)$ для $k = 1,\ldots,n$ и $a = 0,1,\ldots,d$. Тогда соотношение (4.2) может быть записано как

$$F(k,a) = \min\{F(k-1,a) + w_k, F(k-1,a-p_k)\}, \qquad (4.3)$$
$$k = 1,\ldots,n,\ a = 0,1,\ldots,d.$$

Оптимальное значение целевого функционала задачи $1/d_j = d/\sum w_j U_j$ может быть вычислено следующим образом:

$$W^* = F(s_{n+1}) = \min\{F(n,a) | a = 0, 1, \ldots, d\}. \qquad (4.4)$$

Соответствующее решение U^* может быть найдено с помощью следующей процедуры, называемой *обратный ход.*

На каждой итерации этой процедуры проверяется, на первой или второй компоненте достигается минимум в уравнении (4.3) для определенных значений k и a. На первой итерации полагаем $k = n$ и $a = a^*$, где a^* минимизирует правую часть уравнения (4.4). Если минимум в (4.3) достигается на первой компоненте, то полагаем $U_k^* = 1$, $k = k - 1$, оставляем a без изменений и переходим к следующей итерации процедуры обратного хода. Если минимум достигается на второй компоненте, то полагаем $U_k^* = 0$, $a = a^* - p_k$, $k = k-1$, и переходим к следующей итерации. После выполнения n итераций будет построен оптимальный вектор $U^* = (U_1^*, \ldots, U_n^*)$.

Эффективность алгоритма динамического программирования определяется его временной сложностью и объемом требуемой памяти. Анализ алгоритма для рассматриваемой задачи показывает, что вычисление $F(k,a)$ требует выполнения одной операции сложения и одной операции сравнения для каждой пары k и a. Таким образом, временная сложность этого алгоритма равна $O(nd)$. Что касается объема памяти, то следует отметить, что на шаге k для того, чтобы использовать уравнение (4.3), необходимо хранить $2(d+1)$ значений $F(k,a)$, $F(k-1,a)$ и $F(k-1, a-p_k)$. Процедура обратного хода требует $n(d+1)$ дополнительных единиц памяти для хранения значений, указывающих, где достигается минимум в уравнении (4.3) для каждого состояния (k,a). Поэтому приведенный алгоритм требует не более $O(nd)$ единиц памяти. Из приведенного анализа можно сделать вывод, общий для всех алгоритмов динамического программирования: *временная сложность и объем требуемой памяти алгоритма динамического программирования непосредственно зависят от мощности соответствующего пространства состояний.*

Как правило, существует несколько вариантов интерпретации задачи оптимизации как задачи перевода некоторой системы из начального состояния в конечное. Эффективность алгоритма динамического программирования зависит от выбранного пространства

состояний и способа его описания.

Например, для рассматриваемой задачи $1/d_j = d/\sum w_jU_j$ можно ввести состояния (k,w) такие, что система находится в состоянии (k,w), если существует расписание обслуживания требований $1,\ldots,k$, при котором взвешенное число запаздывающих требований равно w. В этом случае функционал $F(k,w)$ определяется как наименьшая суммарная длительность обслуживания ранних требований, вычисленная для расписаний, соответствующих состоянию (k,w). Рекуррентное соотношение можно записать в виде

$$F(k,w) = \min\{F(k-1,w)+p_k, F(k-1,w-w_k)\}, \text{ если}$$

$F(k-1,w)+p_k \le d$, и $F(k,w) = F(k-1,w-w_k)$ в противном случае.

Соответствующий алгоритм динамического программирования требует $O(n\sum_{k=1}^{n} w_k)$ единиц времени и памяти.

Алгоритмы динамического программирования, в которых частичные решения формируются от начала к концу, как это делается в приведенных выше алгоритмах для задачи $1/d_j = d/\sum w_jU_j$, называются *прямыми*. Альтернативный подход состоит в построении частичных решений от конца к началу. Например, рассматриваемая задача $1/d_j = d/\sum w_jU_j$ может быть решена в пространстве состояний (k,a) следующим образом. Полагаем $F(n+1,0) = 0$, все остальные начальные значения $F(k,a)$ равными бесконечности и вычисляем

$$F(k,a) = \min\{F(k+1,a-p_k), F(k+1,a)+w_k\}$$

для $k = n,n-1,\ldots,1$ и $a = 0,1,\ldots,d$.

Оптимальное значение целевого функционала $W^* = \min\{F(1,a)|\ \ a = 0,1,\ldots,d\}$. Такой алгоритм называется *обратным* алгоритмом динамического программирования.

В ряде случаев удобно рассматривать процесс перевода системы из начального состояния в конечное как процесс построения множеств частичных решений исходной задачи и выбора из этих множеств *доминирующих* решений таких, что хотя бы одно из них может быть "достроено"до оптимального. Например, процесс решения рассматриваемой задачи может быть организован следующим образом.

Будем формировать множества $S_1,\ldots,S_n$ частичных расписаний, где множество S_k образовано расписаниями для требований

$1, \ldots, k$, и множество S_{k+1} получается из множества S_k путем определения требования $k+1$ как раннего либо запаздывающего для каждого расписания из S_k. Обозначим через $S_k(a)$ множество расписаний из S_k, соответствующих состоянию (k, a). Заметим, что любое добавление требований $k+1, \ldots, n$ в конец расписаний, соответствующих состоянию (k, a), приведет к одинаковым значениям $w_{k+1}U_{k+1}, \ldots, w_nU_n$. Если значение $F(k, a)$ равно наименьшему взвешенному числу запаздывающих требований, вычисленному для расписаний, соответствующих состоянию (k, a), то расписание со значением $F(k, a)$ *доминирует* все остальные расписания, соответствующие состоянию (k, a), т.е. это расписание может быть достроено до полного расписания для требований $1, \ldots, n$ с наименьшим значением целевого функционала среди всех полных расписаний, достроенных из любого частичного расписания, соответствующего состоянию (k, a). Таким образом, в каждом множестве $S_k(a)$ достаточно выбрать именно такое расписание для дальнейших построений. Оптимальное расписание выбирается из множеств $S_n(a)$, $a = 0, 1, \ldots, d$.

Таким образом, общая методология решения оптимизационных задач с помощью динамического программирования включает следующие аспекты:

– *интерпретацию задачи как процесса перевода некоторой системы из начального состояния в конечное либо процесса построения множеств частичных решений;*

– *отыскание рекуррентных соотношений, связывающих стоимости оптимальных частичных траекторий;*

– *отыскание оптимальной траектории и соответствующего ей решения задачи.*

4.2. ε-приближенные алгоритмы

Минисумная задача. Пусть на множестве R действительных чисел определены неотрицательные неубывающие функционалы $f_i(x), i = 1, \ldots, n$; на множестве $R^n = R \times \ldots \times R$ определен функционал $Z(x) = Z(x_1, \ldots, x_n)$, неубывающий по каждому аргументу, и $A \in R$.

Минисумная задача KMIN формулируется следующим образом.

Минимизировать $F(x) = \sum_{i=1}^{n} f_i(x_i)$ при условиях

$$Z(x) \geq A, \tag{4.5}$$

$$x_i \in R_i, i = 1, \ldots, n, \tag{4.6}$$

где $R_i \subset R$, $i = 1, \ldots, n$, – заданные конечные множества.

Эта задача может быть сформулирована в терминах задачи о рюкзаке следующим образом. Предположим, что предметы n типов должны быть упакованы в рюкзак. Предметы каждого типа i упаковываются единой партией. Возможный размер x_i такой партии выбирается из множества R_i. Объем, занимаемый x_i предметами типа i, определяется функционалом $f_i(x_i)$. Стоимость всех выбранных предметов определяется функционалом $Z(x_1, \ldots, x_n)$. Задача состоит в минимизации суммарного объема выбранных предметов при условии, что их стоимость не ниже заданного значения A.

Обозначим точное решение задачи KMIN через $x^* = (x_1^*, \ldots, x_n^*)$. Очевидно, что x^* существует тогда и только тогда, когда $Z(x') \geq A$, где $x' = (x_1', \ldots, x_n')$, и x_i' – наибольший элемент множества R_i, $i = 1, \ldots, n$. Предположим, что x^* существует и известны числа V и U такие, что $0 < V \leq F(x^*) \leq U$.

Сформулируем *округленную задачу* KR. В постановке этой задачи используется величина $\delta = \varepsilon V/n$. Далее, $\lfloor a \rfloor$ обозначает наибольшее целое число, не превосходящее a. Задача KR состоит в минимизации

$$f(x) = \sum_{i=1}^{n} \lfloor f_i(x_i)/\delta \rfloor$$

при условиях (4.5) и

$$x_i \in \{x_i(0), x_i(1), \ldots, x_i(\lfloor U/\delta \rfloor)\}, i = 1, \ldots, n, \tag{4.7}$$

где $x_i(l)$ является наибольшим элементом $x \in R_i$, удовлетворяющим условию $\lfloor f_i(x)/\delta \rfloor = l$, или, что эквивалентно, условию $l\delta \leq f_i(x) < (l+1)\delta$. Если такого элемента не существует, то $x_i(l) = \phi$.

Предположим, что для любого $l \in \{0, 1, \ldots, \lfloor U/\delta \rfloor\}$ за время $O(\tau)$ можно найти $x_i(l)$ либо установить, что его не существует. Тогда задача KR может быть сформулирована за время $O(\tau n U/\delta)$.

Установим взаимосвязь задачи KMIN и округленной задачи KR.

Теорема 4.1. *Любой точный алгоритм решения задачи KR является ε-приближенным алгоритмом для задачи KMIN.*

Доказательство. Рассмотрим точное решение x^0 задачи KR. По определению оно удовлетворяет ограничениям (4.5), (4.7) и, следовательно, (4.5), (4.6). Остается показать, что из существования x^* следует существование x^0 и $F(x^0) \le (1+\varepsilon)F(x^*)$.

Предположим, что x^* существует. Тогда существует решение x' задачи KMIN такое, что компонента x'_i этого решения – наибольший элемент множества R_i, удовлетворяющий соотношению $\lfloor f_i(x'_i)/\delta \rfloor = \lfloor f_i(x^*_i)/\delta \rfloor$, $i = 1, \ldots, n$. По крайней мере, $x'_i = x^*_i$ для $i = 1, \ldots, n$. Поскольку по определению $x'_i \ge x^*_i$ для $i = 1, \ldots, n$, и функционал $Z(x)$ является неубывающим, $Z(x') \ge Z(x^*) \ge A$. Таким образом, если x^* существует, то существует по крайней мере одно решение, удовлетворяющее ограничениям (4.5), (4.7), т.е. существует x^0.

Покажем, что $F(x^0) \le (1+\varepsilon)F(x^*)$. Справедливы соотношения:

$$F(x^0) = \sum_{i=1}^{n} f_i(x^0_i) \le \delta \sum_{i=1}^{n} \lfloor f_i(x^0_i)/\delta \rfloor + n\delta \le \delta \sum_{i=1}^{n} \lfloor f_i(x'_i)/\delta \rfloor + n\delta =$$

$$\delta \sum_{i=1}^{n} \lfloor f_i(x^*_i)/\delta \rfloor + n\delta \le F(x^*) + n\delta \le (1+\varepsilon)F(x^*).$$

■

Ниже приводится алгоритм динамического программирования K_ε решения задачи KR для случая, когда функционал $Z(x)$ является сепарабельным. В этом алгоритме используется верхняя оценка оптимального значения $f(x^0)$ целевого функционала этой задачи, которая получается следующим образом: $f(x^0) \le f(x^*) \le F(x^*)/\delta \le U/\delta$. Предположим, что функционал Z имеет вид

$$Z(x) = \sum_{i=1}^{n} z_i(x_i),$$

где $z_i(x)$, $i = 1, \ldots, n$, являются некоторыми неубывающими функционалами. В алгоритме K_ε рекурсивно вычисляются значения $Z_j(f)$, где $Z_j(f)$ – максимальное значение $Z(x)$ при условии, что определены значения первых j переменных $x_1, \ldots, x_j$, и $\sum_{i=1}^{j} \lfloor f_i(x_i)/\delta \rfloor = f$. Формальное описание алгоритма состоит в следующем.

Алгоритм K_ε.

Шаг 1. (Инициализация). Полагаем $Z_j(f) = 0$, если $f = 0, j = 0$, и $Z_j(f) = -\infty$ в противном случае. Полагаем $j = 1$.

Шаг 2. (Рекурсия). Для $f = 0, 1, \ldots, \lfloor U/\delta \rfloor$ вычисляем следующее:

$$Z_j(f) = \max\{Z_{j-1}(f - \lfloor f_j(x_j)/\delta \rfloor) + z_j(x_j) \mid x_j \in \{x_j(0), \ldots, x_j(f)\}\}.$$

Если $j < n$, то полагаем $j = j + 1$ и повторяем шаг 2. В противном случае переходим к шагу 3.

Шаг 3. (Отыскание x^0). Оптимальное значение целевого функционала равно

$$\min\{f | Z_n(f) \geq A, f = 0, 1, \ldots, \lfloor U/\delta \rfloor\}.$$

Соответствующее решение x^0 отыскивается при помощи обратного хода алгоритма.

Временная сложность алгоритма K_ε равна $O(\sum_{j=1}^{n} U \min\{U/\delta, |R_j|\}/\delta)$ или, что эквивалентно, $O(\sum_{j=1}^{n} nU \min\{nU/(\varepsilon V), |R_j|\}/(\varepsilon V))$. Если мощность каждого множества R_j ограничена некоторой константой, как, например, в обычной задаче о рюкзаке с 0-1 переменными, то временная сложность алгоритма равна $O(n^2 U/(\varepsilon V))$. В противном случае она равна $O((n^3/\varepsilon^2)(U/V)^2)$. Эти оценки временной сложности могут быть уменьшены до $O(n^2/\varepsilon + n^2 \log\log(U/V))$ и $O(n^3/\varepsilon^2 + n^3 \log\log(U/V))$ соответственно, если использовать процедуру улучшения оценок, приведенную ниже в п. 4.2.2. Таким образом, если значение U/V ограничено некоторой экспонентой от длины записи входных данных задачи в унарном алфавите, то объединение алгоритма K_ε с процедурой улучшения оценок образует вполне полиномиальный ε-приближенный алгоритм для задачи KMIN.

Аналогичный подход может быть применен для приближенного решения следующей задачи максимизации KMAX.

Максимизировать $F(x)$ при условиях (4.6) и

$$Z(x) \leq A. \tag{4.8}$$

В этой задаче и в других задачах, рассматриваемых ниже, предполагается, что известны нижние и верхние оценки оптимального

значения целевого функционала. Для всех этих оценок используются одинаковые обозначения V и U соответственно, и предполагается, что $V > 0$ и $\delta = \varepsilon V/n$.

Округленная задача для задачи KMAX формулируется следующим образом.

Максимизировать $f(x)$ при условиях (4.8) и

$$x_i \in \{x_i'(0), x_i'(1), \ldots, x_i'(\lfloor U/\delta \rfloor)\}, i = 1, \ldots, n,$$

где $x_i'(l)$ является наименьшим элементом $x \in R_i$, удовлетворяющим $\lfloor f_i(x)/\delta \rfloor = l$, если он существует.

Для решения этой задачи можно легко модифицировать алгоритм K_ε следующим образом. В описании шага 1 символ ∞ заменяется на $-\infty$, в описании шага 2 символ $\min$ заменяется на $\max$, и, в описании шага 3 символ $\max$ заменяется на $\min$. Модифицированный таким образом алгоритм K_ε является ε-приближенным алгоритмом решения задачи KMAX. Его временная сложность – та же, что у исходного алгоритма.

4.2.1. Минимаксная задача. Минимаксная задача PMIN состоит в минимизации

$$T(y) = \max\{\sum_{i=1}^{n} f_{li}(y_{li}) | l = 1, \ldots, m\}$$

при условиях

$$y_{li} \in \{0, 1, \ldots, q_i\}, i = 1, \ldots, n, l = 1, \ldots, m, \tag{4.9}$$

$$\sum_{l=1}^{m} y_{li} = q_i, i = 1, \ldots, n, \tag{4.10}$$

где $f_{li}(\cdot)$ – некоторые неотрицательные функционалы, $i = 1, \ldots, n, l = 1, \ldots, m$.

Эта задача допускает следующую интерпретацию. Предположим, что предметы n типов должны быть упакованы в m контейнеров. Общее количество предметов типа i равно q_i. Если y_{li} предметов типа i упаковываются в контейнер l, то они занимают $f_{li}(y_{li})$ единиц его объема. Задача состоит в минимизации объема наиболее заполненного контейнера. Такая задача является в некотором смысле двойственной к известной *задаче об упаковке в контейнеры*, в которой каждый контейнер имеет ограниченный объем, количество контейнеров неограничено, и задача состоит в том, чтобы упаковать предметы таким образом, чтобы минимизировать

количество используемых контейнеров.

Для того чтобы разработать ε-приближенный алгоритм для задачи PMIN, сформулируем следующую округленную задачу PR.

Минимизировать $t(y) = \max\{\sum_{i=1}^{n} \lfloor f_{li}(y_{li})/\delta \rfloor | l = 1, \ldots, m\}$ при условиях

$$(y_{1i}, \ldots, y_{mi}) \in Q_i, i = 1, \ldots, n, \tag{4.11}$$

где Q_i является множеством решений $(u_1, \ldots, u_m)$ следующих систем уравнений, определенных для всех различных наборов $(r_1, \ldots, r_m), r_l \in \{0, 1, \ldots, \lfloor U/\delta \rfloor\}, l = 1, \ldots, m$.

$$\lfloor f_{li}(u_l)/\delta \rfloor = r_l, l = 1, \ldots, m, \tag{4.12}$$

$$u_l \in \{0, 1, \ldots, q_i\}, l = 1, \ldots, m, \tag{4.13}$$

$$\sum_{l=1}^{m} u_l = q_i. \tag{4.14}$$

Используя соотношение $\lfloor y \rfloor \leq y < \lfloor y \rfloor + 1$, условия (4.12), (4.13) могут быть преобразованы к виду

$$u_l \in \{a_l, a_l + 1, \ldots, b_l\}, l = 1, \ldots, m, \tag{4.15}$$

где a_l и b_l – такие числа, что $f_{li}(a_l) = \min\{f_{li}(u) | f_{li}(u) \geq \delta r_l, u \in \{0, 1, \ldots, q_i\}\}$ и $f_{li}(b_l) = \max\{f_{li}(u) | f_{li}(u) < \delta(r_l + 1), u \in \{0, 1, \ldots, q_i\}\}$.

Предположим, что числа a_l и b_l могут быть найдены за время $O(\tau)$ для любого l. Покажем, что задача PR может быть сформулирована за время $O((m + \tau)n(U/\delta)^m)$. Понятно, что $|Q_i| \leq O((U/\delta)^m)$ для $i = 1, \ldots, n$. Остается показать, что система (4.14), (4.15) может быть решена за время $O(m)$.

Если $\sum_{l=1}^{m} a_l > q_i$ либо $\sum_{l=1}^{m} b_l < q_i$, то (4.14), (4.15) и, следовательно, (4.12)-(4.14) не имеет решения. Предположим, что

$$\sum_{l=1}^{m} a_l \leq q_i \leq \sum_{l=1}^{m} b_l. \tag{4.16}$$

Найдем такой индекс $k, 1 \leq k \leq m,$ что

$$q_i \leq \sum_{l=1}^{k-1} a_l + \sum_{l=k}^{m} b_l \text{ и } q_i \geq \sum_{l=1}^{k} a_l + \sum_{l=k+1}^{m} b_l. \tag{4.17}$$

Здесь $\sum_{l=1}^{k-1} a_l = 0$, если $k = 1$, и $\sum_{l=k+1}^{m} b_l = 0$, если $k = m$. В силу предположения (4.16) такое k всегда существует. Зададим

значения u_l следующим образом:

$$u_l = a_l, l = 1, \ldots, k-1, u_l = b_l, l = k+1, \ldots, m,$$

$$u_k = q_i - (\sum_{l=1}^{k-1} a_l + \sum_{l=k+1}^{m} b_l).$$

Понятно, что $\sum_{l=1}^{m} u_l = q_i$. Далее, из (4.17) следует $a_l \le u_l \le b_l$. Таким образом, $(u_1, \ldots, u_m)$ – решение системы (4.12)–(4.14).

Очевидно, что приведенная процедура решения системы (4.14), (4.15) и, следовательно, системы (4.12)–(4.14) требует время $O(m)$, если значения $\sum_{l=1}^{k} a_l$ и $\sum_{l=k}^{m} b_l$, $k = 1, \ldots, m$, вычислены заранее. Поскольку число различных наборов $(r_1, \ldots, r_m)$ и значений i, для которых необходимо решать систему (4.12)–(4.14), не превосходит $(U/\delta)^m$ и n соответственно, задача PR может быть сформулирована за время $O((m+\tau)n(U/\delta)^m)$.

Установим взаимосвязь исходной задачи PMIN и округленной задачи PR.

Теорема 4.2. *Любой точный алгоритм решения задачи PR является ε-приближенным алгоритмом для задачи PMIN.*

Доказательство. Обозначим точные решения задач PMIN и PR через y^* и y^0 соответственно. Для доказательства теоремы достаточно показать, что из существования y^* следует существование y^0 и $T(y^0) \le (1+\varepsilon)T(y^*)$.

Предположим, что y^* существует. Тогда существует матрица y' такая, что для любого $i \in \{1, \ldots, n\}$ вектор $(y'_{1i}, \ldots, y'_{mi})$ является решением системы

$$\lfloor f_{li}(y'_{li})/\delta \rfloor = \lfloor f_{li}(y^*_{li})/\delta \rfloor, l = 1, \ldots, m,$$

$$y'_{li} \in \{0, 1, \ldots, q_i\}, l = 1, \ldots, m, \sum_{l=1}^{m} y'_{li} = q_i.$$

По крайней мере $(y^*_{1i}, \ldots, y^*_{mi})$ является решением этой системы. Таким образом, $Q_i \ne \phi$ для $i = 1, \ldots, n$, т.е. y^0 существует.

Покажем, что $T(y^0) \le (1+\varepsilon)T(y^*)$. Напомним, что $\delta = \varepsilon V/n$ и $V \le T(y^*)$. Справедлива следующая цепочка неравенств:

$$T(y^0) \le \delta \max\{\sum_{i=1}^{n} \lfloor f_{li}(y^0_{li})/\delta \rfloor | l = 1, \ldots, m\} + n\delta \le$$

$$\delta \max\{\sum_{i=1}^{n} \lfloor f_{li}(y'_{li})/\delta \rfloor | l = 1, \ldots, m\} + n\delta =$$

$$\delta \max\{\sum_{i=1}^{n} \lfloor f_{li}(y_{li}^*)/\delta \rfloor | l = 1, \ldots, m\} + n\delta \leq T(y^*) + n\delta \leq (1+\varepsilon)T(y^*).$$

■

Ниже приводится алгоритм динамического программирования решения задачи PR. Нетрудно заметить, что из $T(y^*) \leq U$ следует $t(y^0) \leq U/\delta$. Эта верхняя оценка для $t(y^0)$ используется в приведенном ниже алгоритме P_ε решения задачи PR. В алгоритме P_ε формируется последовательность множеств $S_0, S_1, \ldots, S_n$. Каждое множество S_j включает различные наборы $(T_1, \ldots, T_m)$. Каждый набор из S_j соответствует набору переменных $y_{li}, i = 1, \ldots, j$, $l = 1, \ldots, m$, для которого выполняются ограничения (4.11) и $T_l = \sum_{i=1}^{j} \lfloor f_{li}(y_{li})/\delta \rfloor \leq U/\delta$, $l = 1, \ldots, m$.

Алгоритм P_ε.

Шаг 1. (Инициализация). Полагаем $S_0 = \{(0, \ldots, 0)\}$ и $j = 1$.

Шаг 2. (Формирование $S_1, \ldots, S_n$). Полагаем $S_j = \phi$. Для каждого набора $(T_1, \ldots, T_m) \in S_{j-1}$ и для каждого набора $(y_{1j}, \ldots, y_{mj}) \in Q_j$ вычисляем $T_l' = T_l + \lfloor f_{lj}(y_{lj})/\delta \rfloor$. Если $T_l' \leq U/\delta$ для $l = 1, \ldots, m$ и набора $(T_1', \ldots, T_m')$ нет в S_j, то добавляем его в S_j. При $j < n$ полагаем $j = j + 1$ и повторяем шаг 2. В противном случае переходим к шагу 3.

Шаг 3. (Отыскание y^0). Набор из S_n с наименьшим значением $\max\{T_l | l = 1, \ldots, m\}$ соответствует решению y^0, которое может быть найдено при помощи обратного хода алгоритма.

Установим временную сложность алгоритма P_ε. Очевидно, что процедура формирования множества S_j требует время $O(m|Q_j||S_{j-1}|)$. Поскольку для каждого набора $(T_1, \ldots, T_m) \in S_j$ выполняется $T_l \leq U/\delta$, $l = 1, \ldots, m$, то $|S_j| \leq (U/\delta)^m$ для $j = 1, \ldots, n$. Таким образом, временная сложность алгоритма P_ε не превосходит $O(mn(U/\delta)^m)$ или, что эквивалентно, $O((U/V)^m mn^{m+1}/\varepsilon^m)$. Используя процедуру улучшения оценок (см. п. 4.2.2), эта временная сложность может быть понижена до $O(m3^m n^{m+1}/\varepsilon^m + m2^m n^{m+1} \log\log(U/V))$, т.е. алгоритм P_ε с включенной в него процедурой улучшения оценок является вполне полиномиальным ε-приближенным алгоритмом, если m фиксировано, и величина U/V ограничена по крайней мере экспонентой от длины записи входных данных задачи в унарном алфавите.

Очевидная модификация алгоритма P_ε может быть использована для приближенного решения следующей задачи PMAX:

Максимизировать $\min\{\sum_{i=1}^{n} f_{li}(y_{li}) | l = 1, \ldots, m\}$ при условиях (4.9),(4.10).

Алгоритм P_ε является ε-приближенным алгоритмом решения этой задачи, если на шаге 3 этого алгоритма выбирается набор с максимальным значением $\min\{T_l | l = 1, \ldots, m\}$.

4.2.2. Процедура улучшения оценок. Процедура улучшения нижней и верхней оценки оптимального значения целевого функционала задачи минимизации позволяет отыскивать такое число F^0, что $F^0 \le F^* \le 3F^0$, где F^* – минимальное значение целевого функционала. Процедура основана на применении специального приближенного алгоритма. Ниже описываются свойства, которыми этот алгоритм должен обладать для того, чтобы процедура была корректной и эффективной.

Пусть $F^* > 0$ и имеется приближенный алгоритм $X(A, B)$, удовлетворяющий следующим условиям:

(1) для любых положительных чисел A и B и произвольного набора входных данных задачи алгоритм $X(A, B)$ отыскивает решение со значением целевого функционала $F^X \le B + A$, если $F^* \le B$;

(2) временная сложность алгоритма $X(A, B)$ ограничена полиномом $P(Length, B/A)$ от двух переменных, где $Length$ – длина записи входных данных задачи в двоичном алфавите.

Предположим, что для рассматриваемой задачи известны нижняя и верхняя оценки такие, что $0 < L \le F^* \le U$. Если $U > 3L$, то предлагается процедура отыскания числа F^0 такого, что $F^0 \le F^* \le 3F^0$.

В этой процедуре применяется модификация *метода двоичного поиска* (в интервале $[L, U]$).

Процедура улучшения оценок.

Шаг 1. Полагаем $F^0 = L$, $j = 1$, $a_j = 0$ и находим целое число b_j такое, что $2^{b_j - 1} < U \le 2^{b_j} L$.

Шаг 2. Вычисляем $k_j = \lceil (a_j + b_j)/2 \rceil$ и $F^{(k_j)} = 2^{k_j - 1} L$.

Шаг 3. Применяем алгоритм $X(F^{(k_j)}, 2F^{(k_j)})$. Если он находит решение со значением $F^X \leq 3F^{(k_j)}$, то полагаем $F^0 = F^{(k_j)}$, и, если $k_j = b_j$, то процедура завершается. Если $k_j < b_j$, полагаем $a_{j+1} = a_j$ и $b_{j+1} = k_j$. Если алгоритм $X(F^{(k_j)}, 2F^{(k_j)})$ не находит решения со значением $F^X \leq 3F^{(k_j)}$, то при $k_j = b_j$ процедура завершается. При $k_j < b_j$ полагаем $a_{j+1} = k_j$ и $b_{j+1} = b_j$. В обоих случаях полагаем $j = j + 1$ и переходим к шагу 2.

Теорема 4.3. *Если известны числа L и U такие, что $0 < L \leq F^* \leq U$, и алгоритм $X(A, B)$ удовлетворяет условиям (1) и (2), то процедура улучшения оценок отыскивает значение F^0 такое, что $F^0 \leq F^* \leq 3F^0$, и имеет временную сложность $O(P(Length, 2) \log\log(U/L))$.*

Доказательство. Вначале с помощью индукции покажем, что для каждого значения j, используемого в процедуре, $2^{a_j}L \leq F^* \leq 2^{b_j}L$ либо существует значение j такое, что $F^{(k_i)} \leq F^* \leq 3F^{(k_i)}$ для каждого $i \geq j$, при котором алгоритм $X(F^{(k_i)}, 2F^{(k_i)})$ находит решение со значением $F^X \leq 3F^{(k_i)}$. Понятно, что $2^{a_1}L \leq F^* \leq 2^{b_1}L$. Предположим, что $2^{a_j}L \leq F^* \leq 2^{b_j}L$ и рассмотрим применение алгоритма $X(F^{(k_j)}, 2F^{(k_j)})$ при $k_j = \lceil (a_j + b_j)/2 \rceil$. Если алгоритм не находит решения со значением $F^X \leq 3F^{(k_j)}$, то из условия (1) следует

$$2^{a_{j+1}}L = 2^{k_j}L < F^* \leq 2^{b_j}L = 2^{b_{j+1}}L.$$

Если алгоритм отыскивает решение со значением $F^X \leq 3F^{(k_j)}$, то необходимо проанализировать два случая: $F^* \leq 2F^{(k_j)}$ и $2F^{(k_j)} < F^* \leq F^X$. В первом случае

$$2^{a_{j+1}}L = 2^{a_j}L \leq F^* \leq 2F^{(k_j)} = 2^{b_{j+1}}L.$$

Во втором случае

$$F^{(k_j)} \leq 2F^{(k_j)} < F^* \leq 3F^{(k_j)}$$

и $b_{j+1} = k_j$. Поэтому для любого $i \geq j + 1$ выполняется $k_i \leq k_j$. Таким образом, если алгоритм $X(F^{(k_i)}, 2F^{(k_i)})$ находит решение, то $F^* \leq F^X \leq 3F^{k_i}$ и

$$F^{(k_i)} \leq 2F^{(k_i)} \leq 2F^{(k_j)} < F^* \leq 3F^{(k_i)}.$$

Рассмотрим последнюю итерацию, определяемую из условия $k_j = b_j$. В этом случае $b_j = a_j + 1$. В силу индуктивного предпо-

ложения выполняется $F^{(k_j)} = 2^{k_j-1}L = 2^{a_j}L \leq F^* \leq 2^{b_j}L = 2F^{(k_j)}$ либо $F^0 = F^{(k_i)}$, где i является номером последней итерации, на которой алгоритм $X(F^{(k_i)}, 2F^{(k_i)})$ нашел решение со значением $F^X \leq 3F^{(k_i)}$. В последнем случае корректность процедуры доказана. В первом случае в силу условия (1) алгоритм $X(F^{(k_j)}, 2F^{(k_j)})$ найдет решение со значением $F^X \leq 3F^{(k_j)}$, и поэтому $F^0 = F^{(k_j)}$. Таким образом, значение F^0, найденное при помощи приведенной процедуры, удовлетворяет условию $F^0 \leq F^* \leq 3F^0$.

Временная сложность процедуры улучшения оценок может быть определена следующим образом. Поскольку деление отрезков пополам производится в интервале $[a_1, b_1]$, количество итераций этой процедуры не превосходит $O(\log(b_1 - a_1))$. Очевидно, что $b_1 \leq \log(U/L)+1$. На каждой итерации один раз применяется алгоритм $X(F^{(k_j)}, 2F^{(k_j)})$. Из условия (2) следует, что временная сложность этого алгоритма равна $O(P(Length, 2))$ для любого k_j. Таким образом, временная сложность процедуры улучшения оценок равна $O(P(Length, 2)\log\log(U/L))$. ■

Зачастую в качестве алгоритма X можно воспользоваться ε-приближенным алгоритмом, удовлетворяющим условиям (1) и (2), полагая $\varepsilon = 1$. Следует отметить, что многие ε-приближенные алгоритмы, в том числе все приведенные в данной книге, удовлетворяют этим условиям при $\varepsilon = 1$.

Процедура улучшения оценок и связанные с ней результаты легко адаптируются для задач максимизации.

Глава 5

Приближенный алгоритм для задачи о ширине графа

5.1. Задача о ширине

Дан связный неориентированный граф $G = (V, E)$, где $|V| = n$, $|E| = m$, найти нумерацию вершин, а именно, взаимно однозначное отображение $f : V \to \{1, 2, ..., n\}$, для которого *ширина*, т.е. $\max_{(i,j)\in E} |f(i) - f(j)|$, минимальна.

Эта задача эквивалентна задаче минимизации ширины ленты разреженной симметричной квадратной матрицы, которая возникает в вычислительной математике при запоминании и вычислениях с такими матрицами.

Для некоторых классов графов (интервальных, гусениц) построены эффективные алгоритмы. В общем случае эта задача является NP-трудной.

Известно, что *локальная плотность* графа оценивает снизу ширину графа. Пусть $N(v, \rho)$ — множество вершин находящихся на расстоянии не превосходящем ρ от вершины v, т.е.,

$$N(v, \rho) = \{u \in V | d(u, v) \leq \rho\}.$$

Тогда ширина графа B^* удовлетворяет неравенству

$$B^* \geq \max_{v,\rho} \frac{N(v, \rho)}{2\rho}.$$

Однако, разность между локальной плотностью и шириной графа может достигать $\Omega(\log n)$.

Вероятностный алгоритм Фейга (Feige) строит нумерацию с шириной превосходящей локальную плотность графа не долее чем в

polylog раз и следовательно во столько же оптимальную ширину. Трудоемкость алгоритма $O((n+m)(\log n)^c)$, где c достаточно мало.

5.2. Схема метода

Алгоритм состоит из двух основных шагов. На первом шаге осуществляется укладка графа в ограниченный объем, на втором шаге выполняется проектирование укладки на случайную линию в пространстве. В качестве решения берется нумерация проекций вершин на линии.

Анализ алгоритма основан на следующих принципах. Расстояние между двумя смежными вершинами u, v в исходном графе равно единице. Возникает вопрос — сколько вершин может оказаться между этими вершинами после укладки и проектирования. Поскольку объем для укладки ограничивается, то между вершинами u и v не может быть слишком много других вершин. В противном случае суммарный объем таких вершин был бы сильно сжат (здесь используется ограничение по локальной плотности). Затем мы используем факт, что большие объекты имеют протяженные проекции чтобы показать, что ожидаемое удлинение ребра ограничено $O(c \log n)$. Кроме границ на математическое ожидание нам понадобятся границы на дисперсию длины ребра. Асимптотическое поведение дисперсии указывает, что относительная ошибка алгоритма $O((\log n)^{9/2})$.

5.3. Обозначения

$G(V, E)$ — исходный граф, $|V| = n$, $|E| = m$.
Расстоянием $d(u, v)$ между вершинами $u, v \in V$ называется длина кратчайшего пути между ними.
Для множества вершин $S \subset V$ и вершины $v \in V$, $d(S, v) = \min_{u \in S} d(u, v)$.
$D = \max_{v,\rho} [N(v, \rho)/2\rho]$ — локальная плотность графа.
B^* — ширина графа (оптимум), B — ширина нумерации найденной алгоритмом.

5.4. Объемно ограниченные укладки

Определение 5.1. *Функция расстояний d между точками пространства является* метрикой, *если выполняются следующие свойства:*

1. $d(p,p) = 0$ *для любой точки* p *пространства;*
2. *Симметрия:* $d(p,q) = d(q,p)$ *для любых двух точек* p, q ;
3. *Неравенство треугольника:* $d(p,q) + d(q,r) \geq d(p,r)$ *для любых трех точек* p, q, r .

Определение 5.2. Конечное метрическое пространство (S,d) *определяется конечным множеством точек* S *и функцией расстояний* $d : S \times S \to R^+$, *которая является метрикой.*

Конечное метрическое пространство можно представить полным взвешенным графом на $|S|$ вершинах, у которого реберные веса равны расстоянию между смежными точками. Наоборот, любой неориентированный граф на n вершинах (с положительными весами ребер) задает конечное метрическое пространство, у которого расстояние между двумя вершинами равно длине кратчайшей цепи между ними.

Определение 5.3. Укладкой *конечного метрического пространства* (S,d) *в* L*-мерное Эвклидово пространство называется отображение* $\phi : S \to R^L$. *Если в* R^L *выбран некоторый базис, то отображение* ϕ *порождает* L *функций:* $\phi = (\phi_1, \phi_2, ..., \phi_L)$, *где* $\phi_i : S \to R$, *определяет* i*-ю координату в* R^L .

Определение 5.4. *Пусть* $d_2(p,q)$ *обозначает Эвклидово* l_2 *расстояние между точками* $p, q \in R^L$. *Отображение* ϕ *называется* сжимающим *если для любых точек* $u, v \in S$ $d_2(\phi(u), \phi(v)) \leq d(u,v)$.

Пусть P — множество состоящее из k точек пространства R^L . Тогда через $Evol(P)$ будем обозначать l_2 $(k-1)$-мерный объем симплекса, определенного на P .

Определение 5.5. Объемом *конечного метрического пространства* (S,d) *называется величин*

$$Vol(S) = \max_{\phi: S \to R^{k-1}} [Evol(\phi(S))],$$

Рис. 1. Листья $K_{1,3}$ уложены с максимальным объемом.

где ϕ сжимающее отображение.

Для конечного метрического пространства (S, d) с $|S| = k$ легко проверить, что $Vol(S)$ удовлетворяет следующим свойствам:

- $Vol(S) > 0$.
- Когда $k = 2$ объем совпадает с расстоянием.
- Объем монотонно возрастает при увеличении расстояний d.
- Когда все расстояния умножаются на коэффициент α, объем умножается на коэффициент α^{k-1}.
- Объем является непрерывной функцией от расстояния.

Для множества S состоящего из k вершин графа G рассмотрим конечное метрическое пространство порожденное S и графической метрикой G и определим его объем $Vol(S)$. Рассмотрим укладки, которые сжимают S, но возможно не являются сжимающими для всего графа. Например, объем множества состоящего из трех листьев четырех-вершинной звезды равный $\sqrt{3}$, достигается укладкой листьев в вершины равностороннего треугольника со стороной равной 2. Ясно, что не существует расширения этой укладки до укладки сжимающей весь граф.

Определение 5.6. Деформацией *множества S состоящего из k вершин при отображении ϕ называется величина*

$$\eta(\phi, S) = \Big(\frac{Vol(S)}{Evol(\phi(S))}\Big)^{1/k-1}.$$

При условии, что если $Evol(\phi(S)) = 0$, то $\eta(\phi, S) = \infty$.

Сейчас мы готовы дать центральное определение этого раздела.

Определение 5.7. *Сжатие* $\phi : V \to R^L$ *обладает свойством* (k,c)-объемного соотношения, *или короче называется* (k,c)-сжатием *если для каждого множества* $S \subset V$ *состоящего из* k *вершин, деформация* $\eta(\phi, S) \leq c$.

Сейчас мы представим вероятностный алгоритм для построения (k,c)-сжатий $\phi : V \to R^L$.

Укладка случайных подмножеств:

1. Для $1 \leq j \leq L$, положить $k_j = \lceil \frac{j \log n}{L} \rceil$, и $p_j = 2^{-k_j}$.
2. Для $1 \leq j \leq L$ выбрать множества $S_j \subset V$ так:
   Для каждого $v_i \in V$ и каждого S_j, v_i помещается в S_j с независимой вероятностью p_j.
3. Для каждого $v_i \in V$ и каждого S_j, положить $\phi_j(v_i) = d(S_j, v_i)$. (Это можно сделать так: добавить новую вершину и ребра соединяющие эту вершину со всеми вершинами S_j и затем провести поиск в ширину начиная из этой новой вершины.) Если S_j пустое множество, то положить $\phi_j(v_i) = 0$.
4. Положить $\phi(v) = \frac{1}{\sqrt{L}}(\phi_1(v), ..., \phi_L(v))$.

Теорема 5.1. *Для любого связного графа* G *и любого* $L > \log^2 n$, *укладка* $\phi : V \to R^L$, *найденная алгоритмом, с большой вероятностью является* (k,c)*-сжимающей, с* $k = O(\sqrt{L}/\log n)$ *и* $c = O(\log n \sqrt{k \log n})$.

Доказательство теоремы 5.1 приведем позже.

5.4.1 Оценка объемов

Мы не будем вычислять объем $Vol(S)$ а оценим его.

Рассмотрим "остовное дерево" на S, которое состоит из произвольных $|S| - 1$ ребер, которые делают множество S связным, и имеют реберные веса равные расстояниям между соответствующими вершинами относительно введенной на S метрики. Эти ребра не обязательно принадлежат исходному графу G. Минимальное остовное дерево также минимизирует произведение реберных весов (для перехода к стандартной задаче рассматриваются логарифмы весов).

Определение 5.8. *Древесный объем k-множества S, $Tvol(S)$, это произведение реберных весов в минимальном остовном дереве для S.*

Теорема 5.2. *Пусть S — множество, состоящее из k вершин графа G. Тогда*

$$Vol(S) \leq \frac{Tvol(S)}{(k-1)!} \leq Vol(S)2^{(k-2)/2}.$$

Доказательство. Неравенство $Tvol(S) \geq (k-1)!Vol(S)$ можно доказать индукцией по k.
Неравенство $Tvol(S) \leq (k-1)!Vol(S)2^{(k-2)/2}$ доказывается построением подходящей укладки. □

Укажем связь между древесным объемом и локальной плотностью.

Теорема 5.3. *Для любого (связного) графа G с локальной плотностью D и для любого k,*

$$\sum_{S \subset V, |S|=k} \frac{1}{Tvol(S)} \leq (D(1+\ln(n/D)))^{k-1}.$$

Для доказательства теоремы 5.3 нам понадобится следующая лемма:

Лемма 5.1. *Для любого конечного метрического пространства (S,d) с $S = \{v_!, ..., v_k\}$,*

$$\frac{2^{k-1}}{Tvol(S)} \leq \sum_{\pi} \frac{1}{d(v_{\pi(1)}, v_{\pi(2)}) \cdots d(v_{\pi(k-1)}, v_{\pi(k)})}.$$

Доказательство. Доказательство можно провести полной индукцией по k. □

Доказательство. (Теоремы 5.3). Достаточно показать, что

$$\sum_{S} \sum_{\pi} \frac{1}{d(v_{\pi(1)}, v_{\pi(2)}) \cdots d(v_{\pi(k-1)}, v_{\pi(k)})} \leq n(2D(1+\ln(n/D)))^{k-1},$$

и тогда утверждение теоремы следует из леммы 5.1. Доказать это неравенство можно индукцией по $k = |S|$. □

Для доказательства теоремы 5.1 нам также необходимо оценить $Evol(\phi(S))$. Следующая лемма дает нижнюю границу:

Лемма 5.2. *Пусть $T_1, ..., T_t$ это t попарно ортогональных $(k-1)$-мерных подпространства пространства R^L. Пусть K это k-точечный симплекс в R^L, и пусть K_i — проекция K на T_i.*

Рис. 2. Квадрат длины отрезка равен сумме квадратов его проекций.

Тогда

$$(Evol(K))^{\frac{2}{k-1}} \geq (Evol(K_1))^{\frac{2}{k-1}} + \cdots + (Evol(K_t))^{\frac{2}{k-1}}.$$

Отметим, что для $k = 2$ это утверждение является следствием теоремы Пифагора:

Сейчас мы готовы доказать теорему 5.1. **Доказательство.** (Теоремы 5.1). Мы хотим показать, что случайная укладка является (k, c)-объемно ограниченной, где $k = O(\sqrt{L}/\log n)$ и $c = O(\log n\sqrt{k \log n})$.

Пусть $S \subset V$ множество состоящее из k вершин графа G. Для $v \in V$ и целого t, пусть $\rho(v, t)$ расстояние (в G) между v и ее ближайшим соседом t из G. Отметим, что $\rho(v, t) \leq t$. Определим *порядок исключения* для вершин в S. Для данной вершины $v_i \in S$, пусть d_i обозначает расстояние от v_i до ее ближайшего соседа из S. Выберем наименьшее t такое что существует некоторая вершина $v_i \in S$ с $\rho(v_i, t) \geq d_i/2$. Назовем это $d_1/2$ *критическим расстоянием*. Удалим эту вершину v_i из S и результирующее множество обозначим S'. На рис. 5.3, такой вершиной может быть либо v_2 либо v_3. Пусть u_k новое обозначение вершины v_i (т.е., k я вершина в порядке исключения), и переменной q_k присвоим значение $d_i/2$. Продолжая этот процесс рекурсивно работая с S' мы получим $u_k, u_{k-1}, ..., u_1$ и $q_k, q_{k-1}, ..., q_2$ (здесь нет критического расстояния связанного с u_1, так как мы принимаем $q_1 = 1$).

Поскольку каждое d_i это расстояние между вершиной, которую

Рис. 3. $S = \{v_1, v_2, v_3\}$, и $d_1 = 2, d_2 = d_3 = 1$, итак $t = 1$, поскольку любой ближайший сосед находится на расстоянии 1, что больше чем $1/2$.

мы собираемся удалить из S и другой вершиной в S, произведение d_i-х это произведение реберных весов в некотором остовном дереве в S. Т.о.

$$\Pi_{i=2}^{k} q_i = \Pi_{i=2}^{k} d_i/2 = \frac{\Pi_{i=2}^{k} d_i}{2^{k-1}} \leq \frac{Tvol(S)}{2^{k-1}}.$$

Пусть ϕ случайная укладка S в R^L, где $L \approx (k \log n)^2$. Пусть K это $(k-1)$-мерный симплекс заданный точками $\phi(S)$. Мы дадим нижнюю границу на $Evol(K)$ подобрав $\Theta(k \log n)$ ортогональных $(k-1)$-мерных подпространств R^L. Затем мы покажем, что проекция K на каждое подпространство оценивает, что объем симплекса равен по меньшей мере $q_2 \cdots q_k/(2\sqrt{L} \log n)(k-1)!$, где q_i дерется из упорядоченного списка исключения, построенного выше. Используя свойства упорядоченного списка исключения и теоремы 5.2, мы объем симплекса равен по меньшей мере $Vol(S)/(4\sqrt{L} \log n)^{k-1}$. Тогда из леммы 5.2 получаем, что

$$Evol(K) \geq (k \log n)^{(k-1)/2} \frac{Vol(S)}{(4\sqrt{L} \log n)^{k-1}} \geq \frac{Vol(S)}{(c \log n \sqrt{k \log n})^{k-1}}$$

для некоторой константы $c > 0$. Следовательно искажение имеет величину порядка $O(\log n \sqrt{k \log n})$, что и требовалось.

На этом можно было бы закончить доказательство, если бы ϕ не было случайным. Но поскольку оно случайное, мы не можем быть уверены что вышесказанное *всегда* выполняется. Мы показали, что это справедливо для каждого S, с вероятностью по меньшей мере $1 - n^{-k}$ при случайном выборе ϕ. Поскольку имеется только $\binom{n}{k}$ возможных выбора для S, мы можем заключить, что случайное ϕ подходит для всех S одновременно. □

5.4.2 Большие объемы имеют большие проекции

Мы уже построили укладку k-множества, которая не сильно сжимает свой объем. Далее мы покажем, что проекции укладки не могут быть слишком маленькими. Величина $\sqrt{k/L}$ является грубой оценкой проекции единичного вектора из R^L на случайное подпространство R^k. В следующем утверждении дается граница на вероятность большого отклонения от математического ожидания.

Утверждение 1 *Пусть R — случайный единичный вектор в R^L выбраны с учетом сферической симметрии, и пусть R^k подпространство пространства R^L. Пусть l обозначает длину проекции r на R^k. Тогда для любого $0 < \epsilon < 1$*

$$Pr[l < \epsilon\sqrt{k/L}] \leq (\beta\epsilon)^k,$$

для некоторого универсального β. И для $c > 1, k = 1$,

$$Pr[l > \sqrt{c/L}] < 4e^{-c/2}.$$

Рассмотрим полномерный симплекс K в R^k. Его проекция на это интервал. В следующей лемме утверждается, что если линия выбрана случайно, то вероятность, что этот интервал короткий обратно пропорциональна объему K.

Лемма 5.3. *Пусть S — множество из $k+1$ точек в R^k, где одна из точек — начало координат, и пусть r это единичный вектор в R^k. Проекция S на r определяется скалярным произведением $\langle v, r\rangle$, для $v \in S$. Длина проекции равна $P(S,r) = \max_{v\in S}\langle v, r\rangle - \min_{v\in S}\langle v, r\rangle$. Если направление r выбрано случайно, то*

$$Pr[P(S,r) \leq 1] < \frac{(O(1)/k)^{k/2}}{Evol(S)}.$$

Теорема 5.4 обобщает результат леммы 5.3 на произвольное пространство R^L.

Теорема 5.4. *Пусть S — множество, состоящее из $k+1$ точки в R^L, $L > k$, где одна из точек является началом координат, и пусть r — единичный вектор в R^k. Проекция S на r определяется скалярным произведением $\langle v, r\rangle$, для $v \in S$. Длина проекции $P(S,r) = \max_{v\in S}\langle v, r\rangle - \min_{v\in S}\langle v, r\rangle$. Если направление r выбрано случайно, то*

$$Pr[P(S,r) \leq 1] < (\log(Evol(S))/k + \log L)\frac{(O(1)L)^{k/2}}{k^k Evol(S)}.$$

Мы можем игнорировать множители малого порядка и взять

$$Pr[P(S,r) \le 1] = \widetilde{O}\Big(\frac{L^{k/2}}{k^k Evol(S)}\Big).$$

Следствие 5.1. *Для k-мерного тела K в R^L и для $c > 0$, вероятность, что проекция K на случайный единичный вектор r имеет длину не превосходящую c ограничена величиной*

$$Pr[P(K,r) \le c] = \widetilde{O}\Big(\frac{c^k L^{k/2}}{k^k Evol(S)}\Big).$$

5.5. Алгоритм

Приближенный алгоритм для задачи о ширине описан ниже. Параметр L влияет как на вычислительное время алгоритма, так и на его точность. При анализе предполагается, что $L = \Theta((\log n)^4)$, но меньшие значения L могут дать лучшие результаты на практике.

Алгоритм PLOGBAND:

1. Определить $\phi(S)$ используя случайную укладку.
2. Выбрать случайный вектор r на единичной сфере.
3. Для каждой вершины $v \in S$ вычислить проекцию $\phi(v)$ на r $h(v) = \langle \phi(v), r \rangle$.
4. Упорядочить вершины про $h(v)$.
5. Выдать найденное упорядочение как решение.

Алгоритм имеет трудоемкость $O(mL)$ и легко реализуется.

Теорема 5.5. *Для любого $k \ge 1$ и $L \approx k^2(\log n)^2$, с большой вероятностью построенное алгоритмом упорядочение имеет ширину с относительным приближением $O(\sqrt{kL}(\log n)^2 n^{1/k})$ к оптимальному. В частности, если $k = \log n$, тогда относительная ошибка равна $O((\log n)^{9/2})$.*

Доказательство. Ширина определяется числом вершин v_i, для которых $h(v_i)$ находится между $h(u)$ и $h(v)$ для некоторого ребра (u,v).

Отметим, что для любых двух вершин u и v норма по l_2 вектора $\phi(u) - \phi(v)$ не превосходит $d(u,v)$.

Утверждение 2 *С большой вероятностью при случайном выборе h, для любого ребра (u,v), $|h(u) - h(v)| \le 2\sqrt{\log n / L}$.*

Рис. 4. Так алгоритм PLOGBAND строит нумерацию из размещения в пространстве.

Доказательство. Так как $d(u,v)=1$, $||\phi(u)-\phi(v)|| \leq 1$. Сейчас, $h(u)-h(v)$ равно произведению этого вектора на случайный единичный вектор. Из утверждения 1 следует, что

$$Prob[|h(u)-h(v)| > 2\sqrt{\log n/L}] \leq 1/n^2.$$

Следовательно с высокой вероятностью ($\geq 1/2$), утверждение 2 выполняется для всех ребер одновременно. □

Множество, состоящее из k вершин будем называть *плохим* если для любых двух вершин $u,v \in S$, $|h(u)-h(v)| \leq 2\sqrt{\log n/L}]$. Пусть N_k обозначает число плохих множеств. Следующая лемма устанавливает связь между числом плохих множеств и локальной плотностью D.

Лемма 5.4. *Если* $L \approx k^2(\log n)^2$, *то с высокой вероятностью при случайном выборе* h, $N_k \leq n(\alpha(\log n)^3 D\sqrt{k})^{k-1}$, *для некоторой константы* $\alpha > 0$.

Доказательство. Чтобы упростить изложение мы игнорируем мультипликативные члены $O(1)^{k-1}$, которые воздействуют только на константу α.

Доказательство следует из следующих фактов:

1. Теоремы 5.2: для любого k-подмножества S,
$$Vol(S) \approx \frac{Tvol(S)}{(k-1)!}.$$
2. Теоремы 5.1: с высокой вероятностью при случайном выборе

ϕ, для любого k-подмножества S,

$$Evol(\phi(S)) \geq \frac{Vol(S)}{(\log n\sqrt{k\log n})^{k-1}}.$$

Следовательно,

$$Evol(\phi(S)) \geq \frac{Tvol(S)}{(k\log n)^{3(k-1)/2}}.$$

3. Следствия 5.1: для случайного r,

$$Pr[P(S,r) \leq 2\sqrt{\log n/L}] \leq \frac{(\sqrt{\log n})^{k-1}}{k^{k-1}Evol(\phi(S))}.$$

Следовательно,

$$Pr[P(S,r) \leq 2\sqrt{\log n/L}] \leq \frac{((\log n)^2\sqrt{k})^{k-1}}{Tvol(S)}.$$

4. Теоремы 5.1: для любого k-множества S,

$$\sum_S \frac{1}{Tvol(S)} \leq n(D\log n)^{k-1}.$$

Теперь, $P(S,r) \leq 2\sqrt{\log n/L}$ означает, что S является плохим. Из факта 3, математическое ожидание числа плохих множеств не превосходит $((\log n)^2\sqrt{k})^{k-1}\sum_S \frac{1}{Tvol(S)}$. Используя факт 4, это не превосходит $n((\log n)^3 D\sqrt{k})^{k-1}$. Мы можем воспользоваться неравенством Маркова, чтобы получить оценку вероятности отклонения от математического ожидания. □

Пусть B — ширина укладки, найденной алгоритмом PLOGBAND. Тогда существует число z такое, что $B+1$ вершин w удовлетворяют $z \leq h(w) \leq z + 2\sqrt{\log n/L}$. Следовательно мы можем выбрать плохие множества из этих вершин, и получить $N_k \geq \binom{B+1}{k}$. Но из леммы 5.4 допустить, что $N_k \leq n(\alpha(\log n)^2 D\sqrt{s})^{k-1}$. Комбинируя две оценки N_k, мы получаем, что

$$\binom{B+1}{k} \leq n(\alpha(\log n)^2 D\sqrt{s})^{k-1},$$

откуда следует, что

$$B/D = O(k^{3/2}(\log n)^3 n^{1/k}).$$

Так как $D \leq B^*$,

$$B/B^* \leq O(k^{3/2}(\log n)^3 n^{1/k}).$$

Рис. 5. $B-1$ точек, которые попали между двумя концевыми точками ребра.

□