А. В. Кононов, П. А. Кононова

ПРИБЛИЖЕННЫЕ АЛГОРИТМЫ ДЛЯ NP-ТРУДНЫХ ЗАДАЧ

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Механико-математический факультет
Кафедра теоретической кибернетики

А. В. Кононов, П. А. Кононова

ПРИБЛИЖЕННЫЕ АЛГОРИТМЫ ДЛЯ NP-ТРУДНЫХ ЗАДАЧ

Учебно-методическое пособие

Новосибирск
2014

ББК В173
УДК 519.854.2
К 647

Издание подготовлено в рамках реализации *Программы развития государственного образовательного учреждения высшего профессионального образования «Новосибирский государственный университет»* на 2009–2018 годы.

Кононов, А. В.

К 647 Приближенные алгоритмы для NP-трудных задач : учеб.-метод. пособие / А. В. Кононов, П. А. Кононова ; Новосиб. гос. ун-т. — Новосибирск: РИЦ НГУ, 2014. — 117 с.

Учебное пособие представляет собой изложение курса лекций «Приближенные алгоритмы», которые включены в программу курса магистратуры ММФ НГУ по специальности «Прикладная математика и информатика». В пособии рассматриваются современные методы и подходы к решению фундаментальных NP-трудных задач дискретной оптимизации, таких как задачи о покрытии, задачи упаковки, задачи теории расписаний, задачи размещения и задачи маршрутизации.

Пособие состоит из трех частей. В первой части рассмотрены комбинаторные алгоритмы с гарантированными оценками точности. Вторая часть посвящена основным методам построения приближенных схем. В третьей части представлены приближенные алгоритмы, основанные на решении задач линейного программирования и использовании теории двойственности.

ББК В173
УДК 519.854.2

© Новосибирский государственный университет, 2014
© А. В. Кононов, П. А. Кононова, 2014

Оглавление

Введение

Трудность решения в какой-то мере входит в само понятие задачи: там, где нет трудности, нет и задачи.

Д. Пойа

Комбинаторная оптимизация — одно из самых молодых и бурно развивающихся направлений дискретной математики. Дискретные задачи оптимизации возникают всюду: в традиционных отраслях планирования, таких как теория расписаний, размещение предприятий, дизайн коммуникационных сетей; в современных компьютерных приложениях, связанных с хранением информации в базах данных и быстрым доступом к ней, быстрой передачей пакетов через локальные и глобальные сети; в бесчисленных экономических и финансовых приложениях, таких как планирование инвестиций проектов, разработка стратегий аукционов, или, например, в вирусном маркетинге.

К сожалению, большинство задач, возникающих в комбинаторной оптимизации, являются NP-трудными, и большинство специалистов, работающих в данной области, верят, что для них не существует быстрых эффективных алгоритмов нахождения точного решения. Наше пособие посвящено изучению методов построения приближенных алгоритмов и приближенных схем для труднорешаемых задач.

Что же такое приближенные алгоритмы и приближенные схемы и как они могут помочь в решении трудной вычислительной задачи? Представим себе, что Крош и Ёжик — герои популярного детского сериала

«Смешарики» — решили построить ракету для полета в космос. Строительство космического корабля — серьезное и дорогостоящее мероприятие. Поэтому Крош и Ёжик попросили Пина составить на компьютере календарный план строительства и оценить его стоимость. Пин прикинул свои возможности и сказал, что он может найти оптимальный план, и стоимость проекта, скорее всего, составит миллион монеток. «Отлично, — воскликнул Крош. — Завтра утром мы с Ёжиком придем за планом». «Но я не успею завтра до утра, — ответил Пин. — Я могу составить идеальный план, но для этого мне потребуется двадцать пять лет!»

Крош и Ёжик удивились неспособности Пина решить на компьютере такую пустяковую задачу и обратились к научному гуру Лосяшу. Лосяш подумал и сказал: «Приходите ко мне завтра, и вы получите свой план. Расходы на постройку составят два миллиона монеток!» «Но мы хотим один миллион!» — хором закричали Крош и Ёжик. «Ну тогда приходите через 25 лет!» — ответил Лосяш. «Впрочем, если вы придете ко мне послезавтра, то я смогу найти для вас план стоимостью полтора миллиона, а если вы придете ко мне через три дня, то постройка корабля обойдется вам в один и треть миллиона», — добавил он. «Сколько же будет стоить нам корабль, если мы придем к тебе через x дней?» — спросил любопытный Крош. «Тогда я найду план в $1 + 1/x$ миллионов монеток», — ответил Лосяш.

Что же предложил своим юным друзьям ученый Лосяш? Все просто! Лосяш придумал семейство быстрых приближенных алгоритмов, которое решало задачу Кроша и Ёжика с гарантированной точностью, то есть он придумал быструю приближенную схему.

Так как основная цель нашего пособия — научиться строить приближенные алгоритмы и приближенные схемы для задач комбинаторной оптимизации, то нам потребуется более формально определить, какие задачи мы будем рассматривать и каким свойствам должны удовлетворять алгоритмы, которые мы хотим строить.

0.1 Задачи и алгоритмы

Задачи комбинаторной оптимизации состоят в поиске «наилучшего» решения из огромного, хотя, как правило, конечного множества допу-

стимых решений. Задача оптимизации Π есть либо задача минимизации, либо задача максимизации и состоит из:

- множества Ω_Π индивидуальных задач (примеров);
- конечного множества $Sol_\Pi(I)$ допустимых решений индивидуальной задачи $I \in \Omega_\Pi$;
- целевой функции (критерия) h_Π, сопоставляющей каждой индивидуальной задаче $I \in \Omega_\Pi$ и каждому допустимому решению $\sigma \in Sol_\Pi(I)$ некоторое рациональное число $y(I, \sigma)$, называемое стоимостью решения σ.

Если Π — задача минимизации, то оптимальным решением индивидуальной задачи $I \in \Omega_\Pi$ называется такое допустимое решение $\sigma^* \in Sol_\Pi(I)$, что для всех $\sigma \in Sol_\Pi(I)$ выполнено неравенство $y(I, \sigma^*) \leq y(I, \sigma)$. Алгоритм, который находит оптимальное решение для каждого примера задачи, называется точным.

Типичным примером задачи комбинаторной оптимизации является задача о вершинном покрытии. *Вершинным покрытием* графа $G = (V, E)$ называется такое подмножество вершин $S \subseteq V$, что у каждого ребра графа G хотя бы одна из его вершин (граничных точек) входит в S. Сформулируем задачу в наиболее простой форме.

Задача о вершинном покрытии наименьшей мощности

Дано: неориентированный граф $G = (V, E)$.

Найти вершинное покрытие наименьшей мощности.

Несмотря на простоту своей формулировки, задача о вершинном покрытии наименьшей мощности является NP-трудной, и для нее неизвестны быстрые эффективные алгоритмы. Давайте определимся, какие алгоритмы будем считать быстрыми, а какие — нет. Конечно, время работы любого алгоритма зависит не только от сложности индивидуальной задачи, но и от ее размера. Наивно полагать, что задача о вершинном покрытии для графа с тысячей вершин и десятью тысячами ребер будет решаться так же быстро, как и задача на графе с десятью вершинами и парой десятков ребер. Поэтому для каждого примера определим размер

его входа. Для этого заметим, что все объекты, которые будут рассматриваться в этом пособии (графы, матрицы, слова, рациональные числа), могут быть закодированы последовательностью нулей и единиц. Размер входа примера с рациональными данными равен числу бит, требуемых для его двоичного представления. Например, целое число x требует для своего представления $O(\log x)$ бит.

В свою очередь, все алгоритмы, представленные в этой книге, можно рассматривать как последовательность инструкций, каждая из которых может быть представлена цепочкой элементарных шагов. Примерами таких элементарных шагов являются назначения переменных, условные переходы и простые арифметические операции — сложение, вычитание, умножение, деление или сравнение двух чисел. Алгоритм решения задачи Π называется полиномиальным, если для любого примера $I \in \Omega_\Pi$ алгоритм строит допустимое решение за число элементарных шагов, ограниченное полиномом от размера входа примера I.

Как было упомянуто ранее, задача о вершинном покрытии наименьшей мощности является NP-трудной. Это означает, что к ней сводится некоторая NP-полная задача, и, следовательно, существование точного полиномиального алгоритма для NP-трудной задачи влечет совпадение классов P и NP. Вопрос о совпадении или несовпадении классов P и NP является одной из важнейших открытых проблем современной математики. Многочисленные результаты по теории комбинаторной сложности косвенно подтверждают гипотезу о несовпадении этих классов, и поэтому построение точного полиномиального алгоритма решения NP-трудных задач для большинства специалистов по комбинаторной оптимизации представляется бесперспективным занятием. Более подробно о комбинаторной сложности задач и проблеме совпадения классов P и NP можно прочитать в знаменитой монографии Гэри и Джонсона [1] или в многочисленных учебниках по комбинаторной оптимизации [2], [3], [9].

Большинство задач комбинаторной оптимизации, возникающих в приложениях, являются NP-трудными. Существуют различные подходы к решению таких задач. Один из подходов связан с нахождением точных решений алгоритмами переборного типа. К сожалению, для большинства NP-трудных задач переборные алгоритмы решают только примеры малой размерности. Другой подход связан с поиском приближенных решений. В свою очередь, алгоритмы построения приближенных решений

разделяются на две большие группы. Для первых удается доказать, что они всегда находят решение с гарантированной оценкой точности в худшем случае. Для вторых такие результаты неизвестны, хотя на практике они часто находят решения, близкие к оптимальному. В нашем пособии мы будем изучать приближенные алгоритмы с гарантированной оценкой точности в худшем случае.

Обозначим через $A(I)$ стоимость решения, найденного алгоритмом A, для индивидуальной задачи $I \in \Omega_\Pi$, а через $OPT(I)$ стоимость ее оптимального решения.

Полиномиальный алгоритм A называется *ρ-приближенным алгоритмом* для задачи минимизации Π, если для любого примера $I \in \Omega_\Pi$ алгоритм находит решение, удовлетворяющее неравенству $A(I) \leq \rho OPT(I)$. Величину ρ будем называть *оценкой точности* алгоритма A.

Семейство приближенных алгоритмов $\{A_\varepsilon\}$ называется *полиномиальной приближенной схемой* для задачи оптимизации Π, если для любого фиксированного $\varepsilon > 0$ алгоритм A_ε является $(1+\varepsilon)$-приближенным алгоритмом.

Семейство приближенных алгоритмов $\{A_\varepsilon\}$ называется *вполне полиномиальной приближенной схемой* для задачи оптимизации Π, если для любого $\varepsilon > 0$ алгоритм A_ε является $(1+\varepsilon)$-приближенным алгоритмом, и время его работы ограниченно полиномом от размера входа и величины $\frac{1}{\varepsilon}$.

0.2 Нижние оценки

Перед тем как рассмотреть первый приближенный алгоритм для NP-трудной задачи, давайте зададимся вопросом, как оценить качество построенного алгоритма. Безусловно, было бы идеально сравнить стоимость решений, получаемых алгоритмом, со стоимостью оптимальных решений. Проблема в том, что в рассматриваемых далее задачах не только нахождение оптимального решения является NP-трудной задачей, но и вычисление значения оптимального решения — также NP-трудная задача. Иначе говоря, найти стоимость оптимального решения так же сложно, как и найти само оптимальное решение.

Таким образом, для оценки качества работы приближенного алго-

ритма для NP-трудной задачи минимизации прежде всего требуется построить «хорошую» полиномиально вычислимую нижнюю оценку стоимости оптимального решения этой задачи. Неудивительно, что построение «хорошей» нижней оценки зачастую подсказывает, как построить «хороший» приближенный алгоритм.

Рассмотрим задачу о вершинном покрытии наименьшей мощности. Пусть задан граф $G = (V, E)$. Подмножество ребер $M \subseteq E$ называется *паросочетанием*, если никакие два ребра из M не смежны, то есть не имеют общей граничной точки. Отметим, что нахождение в графе максимального по мощности паросочетания хотя и является нетривиальной задачей, но для нее известны точные полиномиальные алгоритмы. Значительно проще найти в графе паросочетание, максимальное по включению. Начиная с пустого паросочетания, достаточно добавлять к нему в произвольном порядке ребра, у которых нет общих граничных точек с уже включенными в паросочетание ребрами.

Заметим, что мощность максимального по включению паросочетания является нижней оценкой мощности любого вершинного покрытия. Действительно, из определения вершинного покрытия следует, что по крайней мере одна граничная точка каждого ребра, входящего в паросочетание, должна входить в вершинное покрытие. Поскольку ребра, входящие в паросочетание, не имеют общих граничных точек, то число вершин, входящих в покрытие, не может быть меньше числа ребер, входящих в паросочетание.

Алгоритм Простой

1 Найти в графе G максимальное по включению паросочетание M.
2 Пусть S — множество вершин, инцидентных ребрам в M.
3 **return** S

Теорема 1. Алгоритм Простой *является* 2*-приближенным алгоритмом для задачи о вершинном покрытии наименьшей мощности.*

Доказательство. Число вершин в множестве S равно удвоенному числу ребер в M, которое является нижней оценкой мощности любого вершинного покрытия. □

Итак, мы построили первый приближенный алгоритм. Как видим, это оказалось не слишком трудной задачей. Однако, возникает резонный вопрос, а нельзя ли улучшить полученный нами результат. Следующие три вопроса позволяют выделить три основных пути к улучшению аппроксимируемости рассматриваемой задачи.

- Можно ли улучшить анализ качества алгоритма?
- Можно ли для некоторого $\rho < 2$ построить алгоритм, который всегда находит решение, стоимость которого не более чем в ρ раз больше нижней оценки, используемой в доказательстве теоремы 1?
- Можно ли придумать другую нижнюю оценку и разработать на ее основе ρ-приближенный алгоритм с $\rho < 2$?

Давайте рассмотрим по порядку эти вопросы применительно к Алгоритму Простой. Следующий пример показывает, что улучшить анализ качества Алгоритма Простой нельзя. Построим бесконечное семейство примеров, при решении которых он ошибается в два раза. Рассмотрим полный двудольный граф $K_{n,n}$, изображенный на рис. 1. Алгоритм Простой в этом графе выберет в решение все $2n$ вершин, в то время как оптимальное решение состоит из n вершин одной доли.

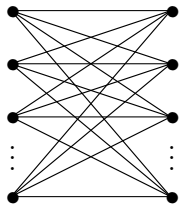


Рис. 1. Полный двудольный граф

Перейдем к следующему вопросу. Пусть для оценки качества решения используется нижняя оценка из теоремы 1. Приведем пример, на котором значение нижней оценки отличается от оптимума в два раза. Рассмотрим полный граф на n вершинах, где n нечетно. Мощность максимального паросочетания равна $(n-1)/2$, а мощность оптимального

вершинного покрытия $n-1$. Следовательно, стоимость решения любого (даже не полиномиального) алгоритма может быть почти в 2 раза хуже нижней оценки.

Труднее всего ответить на вопрос о существовании другого ρ-приближенного алгоритма, основанного на другой нижней оценке, с $\rho < 2$. Только недавно индийский математик Субхаш Кхот выдвинул правдоподобную гипотезу [7], [8], из которой следует, что существование для задачи о вершинном покрытии наименьшей мощности полиномиального алгоритма с гарантированной оценкой $2-\varepsilon$, при любом фиксированном $\varepsilon > 0$, влечет совпадение классов P и NP.

0.3 Упражнения

1. Предложить $\frac{1}{2}$-приближенный алгоритм для решения следующей задачи. В ориентированном графе $G = (V, E)$ выбрать максимальное по мощности множество дуг, такое, что полученный подграф не содержит циклов. (Для произвольного множества вершин выбрать наибольшее из множеств, содержащее либо входящие, либо выходящие дуги. Какую схему можно использовать для верхней оценки оптимума?)

2. Разработать 2-приближенный алгоритм для задачи поиска максимального паросочетания минимального веса в неориентированном графе. (Использовать тот факт, что произвольное максимальное паросочетание не меньше половины максимального паросочетания.)

3. Рассмотрим следующий 2-приближенный алгоритм для задачи о вершинном покрытии наименьшей мощности. С помощью поиска в глубину построить в графе G дерево и выдать множество внутренних вершин S этого дерева. Показать, что S действительно является вершинным покрытием в G и $|S| \leq 2 \cdot OPT$. (Показать, что G имеет паросочетание мощности $|S|/2$.)

4. Рассмотрим следующий жадный алгоритм для решения задачи о вершинном покрытии наименьшей мощности. На каждом шаге

выбираем вершину максимальной степени, добавляем ее в покрытие и удаляем ее и все смежные ей ребра. Продолжаем, пока не удалим все ребра. Доказать, что этот алгоритм для графа с n вершинами всегда выдает решение, стоимость которого не более чем в $\ln n$ раз больше оптимального. Построить пример, на котором решение, полученное жадным алгоритмом, в 100 раз хуже, чем оптимальное.

5. В задаче о максимальном разрезе задан граф $G = (V, E)$ и требуется разбить множество вершин графа V на два непересекающихся множества (две доли) V_1 и V_2 так, чтобы максимизировать число ребер, соединяющих вершины разных множеств.

   - Выбрать произвольное разбиение множества вершин V.
   - Если существует вершина $v \in V$, такая, что число ее соседей в доле, которой она принадлежит больше, чем число ее соседей в другой доле, то переместить эту вершину из одной доли в другую.
   - Если таких вершин нет, то выдать полученное разбиение.

   Докажите, что алгоритм остановится через конечное число шагов и решение, найденное алгоритмом, будет содержать по крайней мере половину всех ребер графа G.

Часть I

Комбинаторные алгоритмы

1 *Задача о покрытии множествами*

— Пух, тебе что намазать, меду или сгущенного молока?
— (Пятачку) Тебе меду, или того и другого? (Кролику) И того, и другого!.. И можно без хлеба!

диалог Кролика и Винни Пуха
из мультфильма «Винни Пух идет
в гости»

Задача о вершинном покрытии наименьшей мощности — это частный случай значительно более общей задачи о покрытии множествами, которая является одной из важнейших задач в комбинаторной оптимизации. Многие интересные проблемы, возникающие в приложениях, могут быть сформулированы как задача о покрытии множествами. Например, представьте себе следующую ситуацию. Сотовая телефонная компания планирует охватить своими услугами новый регион. Для этого на территории региона в доступных для обслуживания местах нужно разместить набор вышек, которые будут получать и передавать радиосигналы. Вышки должны быть размещены так, чтобы сигнал был доступен из каждого населенного пункта региона. При этом компания хочет минимизировать общие затраты на их установку. В общем случае, такая задача может быть сформулирована как задача о покрытии множествами.

Задача о покрытии множествами

Дано: конечное множество U из n элементов, набор его подмножеств $\mathcal{S} = \{S_1, \ldots, S_k\}$ и веса подмножеств $c : \mathcal{S} \to \mathbf{Q}^+$.

Найти минимальный по весу поднабор из $\mathcal{S}$, покрывающий множество U.

Поднабор $S' \subseteq \mathcal{S}$ называется *покрытием*, если любой элемент из U принадлежит хотя бы одному элементу из S'.

1.1 Жадный алгоритм для задачи о покрытии множествами

В 1979 году Хватал [4] предложил жадный алгоритм для задачи о покрытии множествами.

Алгоритм Хватала $(U, S, c : S \to Q^+)$

1 $C \leftarrow \emptyset, Sol \leftarrow \emptyset$
2 **while** $C \neq U$
3 **do**
Найти множество $S_i \in S \setminus Sol$, у которого
$\alpha_i = \frac{c(S_i)}{|S_i \setminus C|}$ минимально.
$Sol \leftarrow Sol \bigcup \{S_i\}$
$price(e) := \alpha_i$ для всех $e \in S_i \setminus C$
$C \leftarrow C \bigcup S_i$
4 **return** Sol

На каждой итерации выбираем самое эффективное множество, удаляем покрытые элементы и продолжаем до тех пор, пока не будут покрыты все элементы. Пусть C — это множество элементов, уже покрытых на предыдущих итерациях. Для каждого множества S_i определим его *эффективность* как $\alpha_i = \frac{c(S_i)}{|S_i \setminus C|}$. Эффективность множества равна средней стоимости, с которой покрываются элементы этого множества, еще не покрытые на предыдущих итерациях. *Ценой элемента* назовем среднюю стоимость, с которой он был покрыт. Для всех вновь покрытых элементов множества S_i цена будет одинаковой.

Занумеруем элементы из множества U в порядке $e_1, \ldots, e_n$, в котором они были покрыты в процессе работы алгоритма.

Лемма 1. *Для каждого $k \in \{1, \ldots, n\}$ верно, что $price(e_k) \leq \frac{OPT}{n-k+1}$.*

Доказательство. Рассмотрим итерацию, на которой был покрыт элемент e_k. Обозначим через $\overline{C}$ — множество еще непокрытых элементов. На каждой итерации все непокрытые элементы могут быть покрыты множествами, общий вес которых не превосходит OPT. Следовательно, среди этих множеств существует множество с эффективностью не более чем $OPT/|\overline{C}|$. Так как элемент e_k еще не покрыт, то множество $\overline{C}$ состоит из не менее чем $n - k + 1$ элементов. Поскольку на этой итерации элемент e_k был покрыт множеством с минимальной эффективностью, то:

$$price(e_k) \leq \frac{OPT}{|\overline{C}|} \leq \frac{OPT}{n-k+1}.$$

□

Теорема 2. Алгоритм Хватала *является H_n-приближенным алгоритмом для задачи о покрытии, где $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}$.*

Доказательство. Поскольку вес каждого покрытого множества поровну распределен по ценам каждого нового покрытого им элемента, то общий вес покрытия равен $\sum_{k=1}^{n} price(e_k)$. По лемме 1 эта величина не превышает $(1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n}) \cdot OPT$. □

На рис. 1.1 приведен пример, на котором эта оценка достигается. Заданы n одноэлементных множеств с весами $\frac{1}{n-i+1}, i = 1, \ldots, n$ и одно множество веса $1 + \varepsilon$ для некоторого $\varepsilon > 0$, содержащее все элементы.

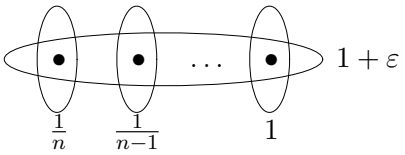


Рис. 1.1. Точный пример к Алгоритму Хватала

В результате работы Алгоритма Хватала множество элементов будет покрыто одноэлементными множествами, каждое из которых является наиболее эффективным на соответствующей итерации. Таким образом, алгоритм найдет покрытие веса:

$$\frac{1}{n} + \frac{1}{n-1} + \ldots + 1 = H_n,$$

в то время как оптимальное покрытие имеет вес $1 + \varepsilon$.

1.2 Уровневый алгоритм для задачи о вершинном покрытии

Рассмотрим задачу, которая является частным случаем задачи о покрытии множествами.

Задача о вершинном покрытии

Дано: граф $G = (V, E)$, веса вершин $w : V \to \mathbf{Q}^+$.

Найти вершинное покрытие наименьшего веса.

Пусть $deg(v)$ обозначает число ребер, инцидентных вершине v. Назовем функцию, соответствующую весам вершин, *пропорционально-степенной*, если существует константа $c > 0$, такая, что вес каждой вершины $v \in V$ равен $c \cdot deg(v)$.

Лемма 2. *Пусть $w : V \to \mathbf{Q}^+$ пропорционально-степенная функция. Тогда $w(V) \leq 2OPT$.*

Доказательство. Пусть $c > 0$ — константа, такая, что $w(v) = c \cdot deg(v)$. Пусть U — оптимальное вершинное покрытие в графе G. Поскольку U покрывает все ребра, то:

$$\sum_{v \in U} deg(v) \geq |E|.$$

Из этого следует, что $w(U) \geq c|E|$. Вспоминая, что $\sum_{v \in V} deg(v) = 2|E|$, получим $w(V) = 2c|E|$. Лемма доказана. □

Пусть $w : V \to \mathbf{Q}^+$ — произвольная функция. Определим *наибольшую пропорционально-степенную функцию относительно* w в графе G следующим образом:

- Удалим из графа все вершины степени 0.
- Среди оставшихся вершин вычислим величину $c = \min\{\frac{w(v)}{deg(v)}\}$.

Тогда функция $t(v) = c \cdot deg(v)$ — искомая.

Определим через $w'(v) = w(v) - t(v)$ *остаточную функцию весов.*

Представим алгоритм, который разбивает исходную весовую функцию на пропорционально-степенные функции и строит вершинное покрытие.

Алгоритм Уровневый $(G = (V, E), w : V \to Q^+)$

1 $Sol \leftarrow \emptyset, i \leftarrow 0, w'(v) \leftarrow w(v)$, $D_0 = \{v \in V | deg(v) = 0\}$
$V_0 \leftarrow V \setminus D_0$ (удаляем вершины степени 0)
2 **while** $V_i \neq \emptyset$
do $c \leftarrow \min\{w(v)/deg(v)\}$
$t(v) \leftarrow c \cdot deg(v)$ (находим наибольшую пропорционально-степенную функцию относительно w')
$w'(v) \leftarrow w'(v) - t_i(v)$ (вычисляем остаточную функцию весов)
$W_i = \{v \in V_i | w'(v) = 0\}$
$Sol \leftarrow Sol \cup W_i$ (пополняем решение)
$V_i \leftarrow V_i \setminus W_i$
$i \leftarrow i + 1$
$D_i = \{v \in V | deg(v) = 0\}$
$V_i \leftarrow V_i \setminus D_i$ (удаляем вершины степени 0)
3 **return** Sol

Схематично процесс работы алгоритма представлен на рис. 1.2.

Теорема 3. Алгоритм Уровневый *является 2-приближенным алгоритмом для задачи о вершинном покрытии.*

Доказательство. Пусть $t_0, \ldots, t_{k-1}$ — пропорционально-степенные функции на графах $G_0, \ldots, G_{k-1}$. Тогда $Sol = W_0 \cup \ldots \cup W_{k-1}$ — решение, полученное алгоритмом, и $V \setminus Sol = D_0 \cup \ldots \cup D_k$.

Сначала покажем, что множество Sol является вершинным покрытием для графа G. Предположим, что это не так. Тогда существует ребро (u, v), такое, что $u \in D_i,\ v \in D_j$ для некоторых i, j. Пусть $i \leq j$, тогда (u, v) должно лежать в G_i. Это противоречит предположению, что степень u равна нулю.

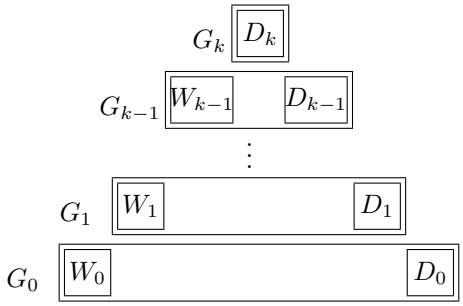


Рис. 1.2. Схема работы Алгоритма Уровневый

Пусть C^* — оптимальное вершинное покрытие. Докажем, что $w(Sol) \leq 2 \cdot OPT$. Рассмотрим вершину $v \in Sol$. Пусть $v \in W_j$ для некоторого j, тогда ее вес равен

$$w(v) = \sum_{i \leq j} t_i(v).$$

Теперь рассмотрим вершину $v \in V \setminus Sol$. Пусть $v \in D_j$ для некоторого j, тогда:

$$w(v) \geq \sum_{i < j} t_i(v).$$

Заметим, что в каждом слое i множество вершин $C^* \cap G_i$ является покрытием для G_i, поскольку G_i — вершинно-индуцированный подграф. Из леммы 2 имеем $t_i(Sol \cap G_i) \leq 2 \cdot t_i(C^* \cap G_i)$. Окончательно получим:

$$w(Sol) = \sum_{t=0}^{k-1} t_i(Sol \cap G_i) \leq 2 \sum_{i=0}^{k-1} t_i(C^* \cap G_i) \leq 2 \cdot w(C^*).$$

□

Приведем пример, на котором достигается эта оценка. Пусть задан полный двудольный граф $K_{n,n}$, в котором все вершины имеют единичные веса. Алгоритм Уровневый возьмет в покрытие все $2n$ вершин графа $K_{n,n}$, тогда как оптимальное решение состоит из вершин одной доли.

1.3 Сведение задачи о кратчайшей суперстроке к задаче о покрытии множествами

Задача о кратчайшей суперстроке

Дано: конечный алфавит Σ и множество строк $U = \{s_1, \ldots, s_n\} \subseteq \Sigma^+$.

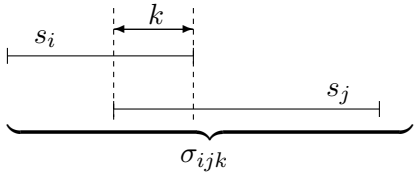
Найти кратчайшую суперстроку s, которая содержит каждую строку s_i, как подстроку.

Без ограничения общности будем считать, что никакая строка s_i не содержит другую строку $s_j, i \neq j$, как подстроку. Через Σ^* обозначим множество всевозможных строк из алфавита Σ. Через Σ^+ обозначим множество всевозможных строк из алфавита Σ, не содержащее пустое слово.

Задача о кратчайшей суперстроке является NP-трудной. Для нее известен следующий жадный алгоритм. Определим *перекрытие* двух строк $s, t \in \Sigma^*$ как максимальную длину суффикса строки s, который является префиксом строки t. Алгоритм работает с множеством строк T. Для начала $T = U$. На каждом шаге алгоритм выбирает из множества T две строки, которые имеют максимальное перекрытие, и заменяет их строкой, которая получается из них перекрытием. После $n - 1$ шага T будет состоять из одной строки. Очевидно, что эта строка будет содержать каждую из строк s_i как подстроку. Есть гипотеза, что этот алгоритм является 2-приближенным. На следующем примере можно увидеть, как алгоритм находит решение в 2 раза длиннее оптимального. Рассмотрим 3 строки: ab^k, b^kc, b^{k+1}. Если сначала выбрать две первые строки, то жадный алгоритм получит строку ab^kcb^{k+1}, это в два раза длиннее, чем кратчайшая суперстрока $ab^{k+1}c$.

Следующий $2H_n$-приближенный алгоритм основан на сведении задачи о кратчайшей суперстроке к задаче о покрытии множествами. Используя исходные данные задачи о кратчайшей суперстроке, построим

пример $\mathcal{S}$ задачи о покрытии. Для строк $s_i, s_j \in U$ и некоторого $k > 0$, если последние k символов строки s_i совпадают с первыми k символами строки s_j, получим строку σ_{ijk} совмещением строк s_i и s_j по k позициям.



Пусть множество M состоит из всех возможных допустимых строк σ_{ijk}. Для каждой строки $\pi \in \Sigma^+$ определим множество $set(\pi) = \{s \in U | s \text{ является подстрокой } \pi\}$. Множество элементов U требуется покрыть подмножествами $set(\pi)$, где $\pi \in U \cup M$. Вес множества $set(\pi)$ положим равным $|\pi|$, т. е. длине строки π.

Алгоритм Ли

1 По индивидуальной задаче о кратчайшей суперстроке
построить индивидуальную задачу о покрытии множествами.
2 С помощью Алгоритма Хватала найти покрытие.
Пусть оно состоит из множеств $set(\pi_1), \ldots, set(\pi_k)$.
3 Соединить строки $\pi_1, \ldots, \pi_k$ в произвольном порядке в суперстроку s.
4 **return** s

Пусть $OPT_{\mathcal{S}}$ и OPT означают стоимость оптимального решения примера $\mathcal{S}$ задачи о покрытии и длину кратчайшей суперстроки соответственно. Покажем, что эти величины могут отличаться друг от друга не более, чем в 2 раза.

Лемма 3. $OPT \leq OPT_{\mathcal{S}} \leq 2 \cdot OPT$.

Доказательство. Рассмотрим оптимальное покрытие $\{set(\pi_i) | 1 \leq i \leq l\}$ и получим строку s соединением строк $\pi_i, 1 \leq i \leq l$ в произвольном порядке. Очевидно, что $|s| = OPT_{\mathcal{S}}$. Так как каждая строка из U является подстрокой некоторой строки π_i, то она также является подстрокой s. Поэтому полученная строка является суперстрокой, и $OPT_{\mathcal{S}} = |s| \geq OPT$.

Для доказательства второго неравенства рассмотрим кратчайшую суперстроку s^* для строк $s_1, \ldots, s_n$, $|s^*| = OPT$. Покажем, что для примера $\mathcal{S}$ задачи о покрытии существует некоторое покрытие веса не более $2 \cdot OPT$.

Рассмотрим самое левое появление строк $s_1, \ldots, s_n$ в строке s^*. Так как никакие из строк $s_1, \ldots, s_n$ не являются подстроками друг друга, то крайние левые n строк начинаются с различных позиций. По той же самой причине они заканчиваются в различных позициях. Перенумеруем строки в порядке их появления. Так как никакая из строк не является подстрокой другой, то они будут заканчиваться в том же порядке.

Разобьем упорядоченный список строк $s_1, \ldots, s_n$ на группы, как показано на рис. 1.3. Каждая группа будет состоять из множества попарно пересекающихся строк. Обозначим через b_i и e_i индексы первой и последней строки в i-ой группе. Если группа i состоит только из одной строки, то $b_i = e_i$. Положим $b_1 = 1$. Пусть e_1 — это наибольший индекс строки, которая перекрывается со строкой s_1 (существует как минимум одна такая строка — s_1). В общем случае: если $e_i < n$, то положим $b_{i+1} = e_i + 1$ и определим e_{i+1} как наибольший индекс строки, которая перекрывается с b_{i+1}. В конце процедуры получим $e_t = n$ для некоторого $t \leq n$.

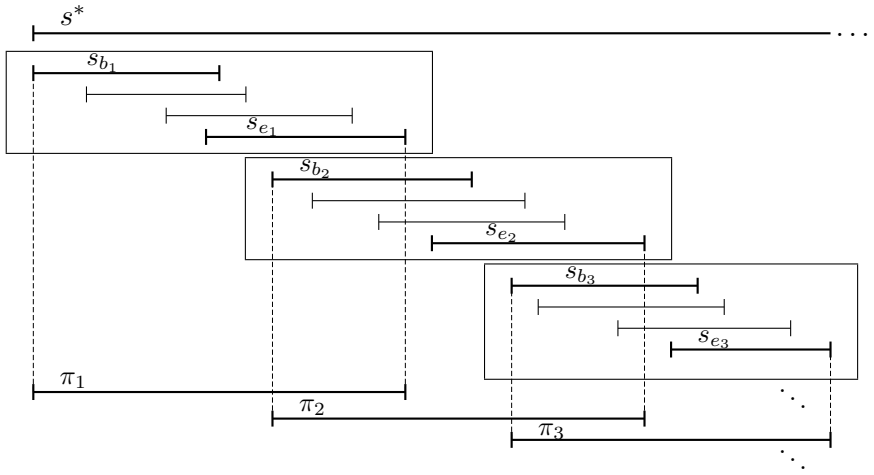


Рис. 1.3. Разбиение строк на группы

Для каждой пары строк (s_{b_i}, s_{e_i}) существует $k_i > 0$, определяющее длину перекрытия этих строк в строке s (оно может отличаться от их максимального перекрытия). Пусть $\pi_i = \sigma_{b_i e_i k_i}$. Тогда набор подмножеств $\{set(\pi_i) | 1 \leq i \leq t\}$ является допустимым решением примера $\mathcal{S}$, и его стоимость равна $\sum_i |\pi_i|$.

Теперь заметим, что π_i не пересекается с π_{i+2}. Докажем это утверждение для $i = 1$. Предположим, что строки π_1 и π_3 перекрываются. Тогда получается, что s_{b_3} перекрывается с s_{e_1} в строке s^*. Однако по построению s_{b_3} не перекрывается с s_{b_2} (поскольку s_{b_3} относится к следующей группе). Следовательно, s_{e_1} заканчивается позже чем s_{b_2}. Получаем противоречие с тем, что строки должны заканчиваться в том же порядке, как и начинаются. Доказательство для других i строится аналогично.

В итоге получим, что каждый символ строки s^* входит не более чем в два множества π_i. Отсюда получаем, что $OPT_{\mathcal{S}} \leq \sum_i |\pi_i| \leq 2 \cdot OPT$. $\square$

Мощность множества элементов U в примере $\mathcal{S}$ равна n, то есть числу строк в задаче о суперстроке. Поэтому из леммы 3 и теоремы 2 получаем следующую теорему.

Теорема 4. Алгоритм Ли *является $2H_n$-приближенным алгоритмом для задачи о суперстроке, где n равно числу строк в заданном примере.*

1.4 Упражнения

1. Из слов *сыроежка, опёнок, трюфель, шампиньон, мухомор, белянка, рыжик, эвкалипт, берёза, туя, секвойя, липа, крыжовник* требуется выбрать набор слов так, чтобы в них содержались все гласные буквы и общее число букв было минимальным. Сформулировать задачу как задачу о покрытии и решить Алгоритмом Хватала.

2. Рассмотрим следующую жадную стратегию для задачи о вершинном покрытии наименьшей мощности. Возьмем в решение произвольную вершину максимальной степени и удалим ее вместе с инцидентными ей ребрами. Продолжим эту процедуру до тех пор,

пока в графе не останется ребер. Докажите, что полученное решение не превосходит $H_n \cdot OPT$.

3. Постройте пример, на котором достигается оценка Алгоритма Хвatала для задачи о покрытии множествами, даже если вес каждого множества равен 1.

4. Рассмотрим примеры задачи о покрытии, в которых каждый элемент входит не больше чем в 3 подмножества. Постройте для таких примеров 3-приближенный уровневый алгоритм.

5. Полным антисимметрическим графом называется ориентированный граф $G = (V, E)$, в котором для любой пары вершин $u, v \in V$ существует ровно одна из дуг $(u, v) \in E$ либо $(v, u) \in E$. Обратным вершинным покрытием в G называется множество вершин, после удаления которых граф становится ациклическим. Предложить 3-приближенный алгоритм для поиска обратного вершинного покрытия в антисимметрическом графе. (Показать, что для этого достаточно убить циклы длины 3.)

2 *Задачи с метрикой*

Ходы кривые роет
Подземный умный крот.
Нормальные герои
Всегда идут в обход!

строки песни из кинофильма
«Айболит-66»

В этой главе представим приближенные алгоритмы для метрической задачи Штейнера и метрической задачи коммивояжера. Причины рассмотрения метрических вариантов этих задач разные. Для задачи Штейнера метрическая задача является ядром общей задачи, и общая задача может быть сведена к метрической. Для задачи коммивояжера без этого ограничения не существует «хороших» приближенных алгоритмов, в предположении, что $P \neq NP$. Тем не менее, приближенные алгоритмы для этих задач используют общую идею, которая и будет рассмотрена в этой главе.

В конце главы покажем, что задача о кратчайшей суперстроке, введенная в предыдущей главе, может быть сведена к задаче коммивояжера со специальной метрикой, и для нее существует 4-приближенный алгоритм.

2.1 Метрическая задача Штейнера

Задача о дереве Штейнера была поставлена Гауссом в письме Шумахеру. На плоскости задано n городов, необходимо соединить их ломаны-

ми линиями, чтобы каждый город был соединен с каждым и суммарная длина всех проведенных линий была минимальна. Заметим, что в оптимальное решение могут входить промежуточные точки, а все соединения должны быть отрезками, соединяющими точки (исходные или промежуточные). Далее рассмотрим задачу Штейнера на графах.

Задача Штейнера

Дано: граф $G = (V, E)$, стоимости ребер $cost : E \to \mathbf{Q}^+$, подмножество $R \subseteq V$.

Найти дерево T наименьшей стоимости, такое, что $R \subseteq T$, т. е. содержит все вершины из R.

Множество R называется *множеством требований*.
Множество $V \setminus R$ называется *множеством Штейнера*.

Метрическая задача Штейнера

Дано: полный граф $G = (V, E)$, стоимости ребер $cost : E \to \mathbf{Q}^+$, такие, что для любых трех вершин u, v и w выполнено неравенство: $cost(u, v) \leq cost(u, w) + cost(w, v)$, подмножество $R \subseteq V$.

Найти дерево T наименьшей стоимости, такое, что $R \subseteq T$.

Теорема 5. *Существование ρ-приближенного алгоритма для метрической задачи Штейнера влечет существование ρ-приближенного алгоритма для задачи Штейнера.*

Доказательство. Пусть I — пример задачи Штейнера с графом $G = (V, E)$. По графу G построим полный граф G' для примера I' метрической задачи Штейнера. Определим стоимость ребра (u, v) в G', как стоимость кратчайшего $u - v$-пути в G. Граф G' называется *метрическим замыканием* графа G. Множество требований и множество Штейнера в обоих примерах совпадают.

Стоимость любого ребра $(u, v) \in E$ в графе G' не превосходит стоимости этого ребра в графе G. Поэтому стоимость оптимального решения примера I не превосходит стоимости оптимального решения примера I'.

Пусть задано дерево Штейнера T' в примере I'. Покажем, как за полиномиальное время построить в примере I дерево Штейнера не большей

стоимости. Стоимость ребра (u, v) в графе G' соответствует стоимости пути в графе G. Заменим каждое ребро дерева T' на соответствующий путь для получения подграфа графа G. Очевидно, что в этом подграфе все вершины множества требований соединены. Однако, этот подграф в общем случае может содержать цикл. Если это так, то удалим лишние ребра, чтобы получить дерево T.

Пусть C' — решение, найденное ρ-приближенным алгоритмом для метрической задачи Штейнера. Способом, описанным выше, найдем решение стоимости C в задаче Штейнера, такое, что

$$C \leq C' \leq \rho OPT' \leq \rho OPT.$$

□

Пусть R — множество требований. Очевидно, что минимальное остовное дерево в R является допустимым решением. Так как задача поиска минимального остовного дерева принадлежит классу P, а метрическая задача Штейнера NP-трудна, то, по-видимому, нельзя утверждать, что оптимальное решение задачи о минимальном остовном дереве всегда является оптимальным решением задачи Штейнера. Действительно, рассмотрим следующий пример, который изображен на рис. 2.1.

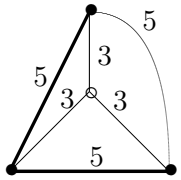


Рис. 2.1. Минимальное остовное дерево и минимальное дерево Штейнера

Пусть задан полный граф на четырех вершинах. Множество требований состоит из трех вершин, которые выделены черным цветом (•), а множество Штейнера — из одной вершины белого цвета (○). Стоимость ребер между каждой парой требований равна 5, а стоимость ребер, связывающих вершину Штейнера с остальными вершинами, равна 3. Оптимальное решение задачи о минимальном остовном дереве нарисовано

жирными линиями, его стоимость равна 10. Оптимальное решение метрической задачи Штейнера нарисовано тонкими линиями, и его стоимость равна 9. Дугой изображено ребро, не попавшее в эти деревья.

Итак, минимальное остовное дерево не всегда является оптимальным решением задачи Штейнера. Однако стоимость минимального остовного дерева является хорошей оценкой на стоимость оптимального дерева Штейнера.

Теорема 6. *Пусть $R \subseteq V$ — множество требований в задаче Штейнера. Тогда стоимость минимального остовного дерева в R не превосходит двух стоимостей оптимального дерева Штейнера в графе G.*

Доказательство. Рассмотрим дерево Штейнера стоимость которого равна OPT (рис. 2.2). Дублируя ребра, получим Эйлеров граф, связывающий все вершины из множества R, а также, возможно, и некоторые Штейнеровы вершины. Найдем Эйлеров обход в этом графе.

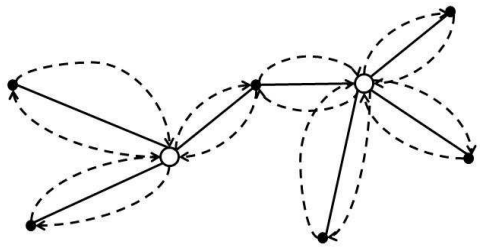


Рис. 2.2. Двойной обход

Стоимость этого обхода равна $2 \cdot OPT$. Затем, используя порядок вершин в Эйлеровом обходе, получим Гамильтонов цикл методом срезания углов, пропуская вершины Штейнера и уже пройденные вершины (рис. 2.3).

Из неравенства треугольника следует, что стоимость нового цикла не может превышать стоимости Эйлерова обхода. Удалив одно ребро из Гамильтонового цикла, получим путь P по вершинам из R стоимости, не превышающей $2 \cdot OPT$. Путь P является остовным деревом на R. Поэтому стоимость минимального остовного дерева не превышает $2 \cdot OPT$. $\square$

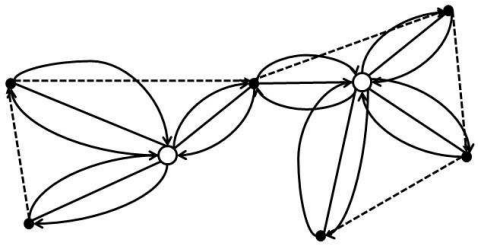


Рис. 2.3. Метод срезания углов

Используя результат теоремы 6, построим простой приближенный алгоритм для задачи Штейнера.

Алгоритм Остовное дерево $(G = (V, E), R, cost : E \to \mathbf{Q}^+)$

1 Найти минимальное остовное дерево T на множестве вершин R
2 **return** T

Следствие 1. Алгоритм Остовное дерево *является 2-приближенным алгоритмом для задачи Штейнера.*

На рис. 2.4 приведен пример, на котором эта оценка достигается. Рассмотрим граф, в котором n требований и одна вершина Штейнера. Стоимость ребра между вершиной Штейнера и требованиями равна 1, стоимость ребра между любой парой требований равна 2. В этом графе остовное дерево на множестве вершин R имеет стоимость $2(n-1)$, а оптимальное дерево Штейнера имеет стоимость n.

2.2 Метрическая задача коммивояжера

Задача коммивояжера — одна из самых известных задач комбинаторной оптимизации, заключающаяся в отыскании самого выгодного маршрута, проходящего через указанные города ровно по одному разу с последующим возвратом в исходный город.

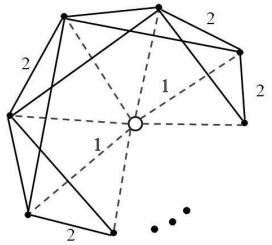


Рис. 2.4. Пример точности оценки Алгоритма Остовное дерево

Задача коммивояжера

Дано: полный граф $G = (V, E)$, стоимости ребер $cost : E \to \mathbf{Q}^+$.

Найти гамильтонов цикл C минимальной стоимости.

Цикл, который содержит все вершины, называется *гамильтоновым.*

Теорема 7. *Для любой полиномиально вычислимой функции $\alpha(n)$, для задачи коммивояжера не существует ρ-приближенного алгоритма с $\rho = \alpha(n)$, если $P \neq NP$.*

Доказательство. Докажем от противного. Предположим, что существует ρ-приближенный алгоритм $\mathcal{A}$ с $\rho = \alpha(n)$ для задачи коммивояжера. Покажем, что с помощью полиномиального алгоритма $\mathcal{A}$ можно решать NP-трудную задачу Гамильтонов цикл. Это будет означать, что $P = NP$.

Основная идея сведения — построить по графу G полный взвешенный граф G', такой, что:

- если в G есть гамильтонов цикл, то стоимость оптимального решения задачи коммивояжера в графе G' равна n, и
- если в G нет гамильтонова цикла, то оптимальное решение задачи коммивояжера для графа G' имеет стоимость не меньше, чем $n \cdot \alpha(n)$.

Заметим, что алгоритм $\mathcal{A}$ для графа G' должен найти решение стоимости $cost \leq \alpha(n) \cdot n$ в первом случае и стоимости $cost > \alpha(n) \cdot n$ во втором случае.

По графу G построим полный взвешенный граф G'. Назначим стоимости, равные 1, тем ребрам, которые есть в графе G, и стоимости, равные $\alpha(n) \cdot n$, тем ребрам, которых нет в графе G. Если в графе G есть гамильтонов цикл, то соответствующий ему обход в графе G' имеет стоимость n. С другой стороны, если в G нет гамильтонова цикла, то любой обход в графе G' должен использовать ребро стоимости $\alpha(n) \cdot n$, а значит, стоимость обхода будет $cost > \alpha(n) \cdot n$. □

Заметим, что в доказательстве такой сильной неаппроксимируемости используются стоимости ребер, которые нарушают неравенство треугольника. Если ограничиться только теми графами, на которых это неравенство выполняется, т. е. *метрической задачей коммивояжера*, то задача останется NP-трудной, но не будет столь трудна для аппроксимации.

Метрическая задача коммивояжера

Дано: полный граф $G = (V, E)$, стоимости ребер $cost : E \to \mathbf{Q}^+$, такие, что для любых трех вершин u, v и w выполняется $cost(u, v) \leq cost(u, w) + cost(w, v)$.

Найти гамильтонов цикл C минимальной стоимости.

Представим простой 2-приближенный алгоритм для метрической задачи коммивояжера. В качестве нижней оценки используем оптимальное решение задачи о минимальном остовном дереве на графе G. Это решение является нижней оценкой для оптимального решения задачи коммивояжера, поскольку при удалении любого ребра из гамильтонова цикла получается остовное дерево в графе G.

Алгоритм Двойное остовное дерево $(G = (V, E), cost : E \to \mathbf{Q}^+)$

1 Найти минимальное остовное дерево T в графе G.
2 Удвоить каждое ребро в дереве T и получить Эйлеров граф H.
3 Найти Эйлеров обход R в графе H.
4 Построить Гамильтонов цикл C, посещая вершины графа G в том порядке, в котором они встречаются в R.
5 **return** C

Заметим, что выполнение шага 4 аналогично методу срезания углов из теоремы 6.

Теорема 8. Алгоритм Двойное остовное дерево *является 2-приближенным алгоритмом для метрической задачи коммивояжера.*

Доказательство. Как было замечено выше, $cost(T) \leq OPT$. Поскольку граф H содержит каждое ребро из дерева T дважды, то $cost(R) = 2 \cdot cost(T)$. Тогда из неравенства треугольника после процедуры срезания углов получим $cost(C) \leq cost(R)$ и $cost(C) \leq 2 \cdot OPT$. □

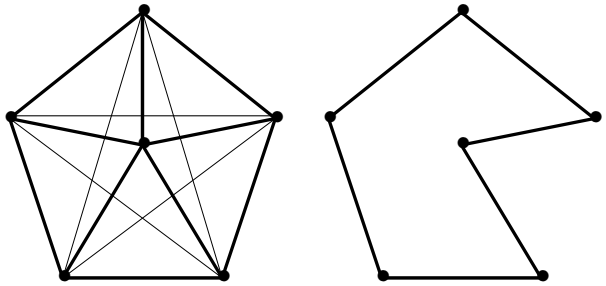


Рис. 2.5. Пример точности оценки Алгоритма Двойное остовное дерево

Приведем пример, который показывает, что эта оценка точна. Пусть задан полный граф на n вершинах, стоимости ребер равны 1 и 2. Для $n = 6$ граф изображен на рис. 2.5. Все толстые ребра имеют стоимость, равную 1, а остальные — стоимость, равную 2. В случае произвольного n граф будет иметь $2n - 2$ ребра стоимости 1. Эти ребра образуют звезду $K_{1,n-1}$ и цикл на $n-1$ вершине, проходящий через центр звезды. Стоимости остальных ребер равны 2. Оптимальное решение задачи коммивояжера имеет стоимость n. На рисунке 2.5 изображен оптимальный обход коммивояжера для $n = 6$.

Предположим, что Алгоритм Остовное дерево найдет остовное дерево в виде звезды, составленной из ребер стоимости 1. Далее предположим, что Эйлеров обход посетит вершины в порядке, показанном на рис. 2.6.

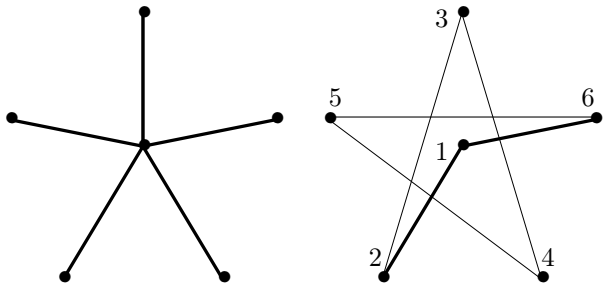


Рис. 2.6. Оптимальное остовное дерево и цикл, полученный из него

Тогда цикл, полученный методом срезания углов, содержит $n-2$ ребра стоимости 2 и имеет общую стоимость $2n-2$. При $n \to \infty$ получим, что стоимость такого цикла в два раза больше оптимального.

Алгоритм Двойное остовное дерево сначала находит недорогой Эйлеров обход из остовного дерева графа G, а затем из него делает обход коммивояжера методом срезания углов. Существует ли Эйлеров обход, отличный от найденного с помощью двойного остовного дерева? Напомним, что граф имеет Эйлеров обход тогда и только тогда, когда степень каждой его вершины четна. Пусть множество V' содержит все вершины, которые имеют нечетную степень в остовном дереве. Тогда количество таких вершин $|V'|$ должно быть четно, поскольку сумма степеней всех вершин в любом графе четна. Если мы добавим к минимальному остовному дереву минимальное совершенное паросочетание для множества V', то степень каждой вершины будет четна, и мы получим Эйлеров граф. Напомним, что задача нахождения совершенного паросочетания минимальной стоимости разрешима за полиномиальное время.

Алгоритм Кристофидеса – Сердюкова $(G = (V, E), cost : E \to \mathbf{Q}^+)$

1 Найти минимальное остовное дерево T в графе G.
2 Найти паросочетание минимальной стоимости M
на множестве вершин в T с нечетными степенями.
3 Добавить ребра M к T и получить Эйлеров граф H.
4 Найти Эйлеров обход R в графе H.
5 Построить Гамильтонов цикл C, посещая вершины графа G
в том порядке, в котором они встречаются в R.
6 **return** C

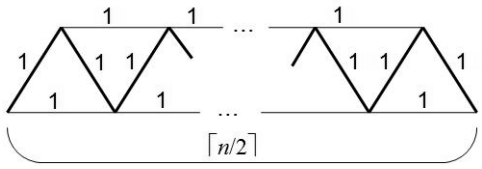


Рис. 2.7. Точность Алгоритма Кристофидеса – Сердюкова

Для оценки точности этого алгоритма приведем новую нижнюю оценку на стоимость оптимального решения в задаче коммивояжера.

Лемма 4. *Пусть $V' \subseteq V$, такое, что $|V'|$ четно, и пусть M — минимальное по стоимости совершенное паросочетание для V'. Тогда $cost(M) \leq OPT/2$.*

Доказательство. Рассмотрим оптимальный цикл τ в задаче коммивояжера на графе G. Пусть цикл τ' получается из τ на множестве вершин V' методом срезания углов. Из неравенства треугольника имеем, что $cost(\tau') \leq cost(\tau)$. Но τ' является объединением двух совершенных паросочетаний на V', каждое из которых содержит альтернативное ребро τ. Таким образом, наименьшее из двух паросочетаний имеет стоимость не более $cost(\tau')/2 \leq OPT/2$. Следовательно, оптимальное паросочетание также имеет стоимость не более $OPT/2$. □

Теорема 9. Алгоритм Кристофидеса – Сердюкова *является 3/2-приближенным алгоритмом для метрической задачи коммивояжера.*

Доказательство. Оценим стоимость Эйлерова обхода R

$$cost(R) \leq cost(T) + cost(M) \leq OPT + \frac{1}{2}OPT = \frac{3}{2}OPT.$$

Используя неравенство треугольника, получим $cost(C) \leq cost(R)$. Теорема доказана. □

На рис. 2.7 приведен пример графа на n вершинах для некоторого нечетного числа n, который показывает, что эта оценка точна. Жирным цветом отмечены ребра, которые входят в минимальное остовное дерево T. Это остовное дерево имеет только две вершины нечетной степени, и,

соединяя их ребром, получим обход коммивояжера стоимости $(n-1)+\lceil n/2 \rceil$. Оптимальный обход имеет стоимость n.

Поиски лучших приближенных алгоритмов с гарантированной оценкой точности для метрической задачи коммивояжера продолжаются. Многие ученые считают, что должен существовать 4/3-приближенный алгоритм.

2.3 Кратчайшая суперстрока

В первой главе рассматривалась задача о кратчайшей суперстроке. Приближенный алгоритм для ее решения был основан на сведении ее к задаче о покрытии множествами. В этой главе опишем другой подход к решению задачи о кратчайшей суперстроке и получим для ее решения 4-приближенный алгоритм.

Кратчайшая суперстрока

Дано: конечный алфавит Σ и множество строк $U = \{s_1, \ldots, s_n\} \subseteq \Sigma^+$.

Найти кратчайшую суперстроку s, которая содержит каждую строку $s_i \in U$, как подстроку.

Начнем с построения хорошей нижней оценки на оптимальное решение.

Пусть s^* — кратчайшая суперстрока. Перенумеруем строки так, чтобы они появлялись в s^* в порядке $s_1, \ldots, s_n$ (рис. 2.8).

Перекрытием (s_i, s_j) назовем максимальное перекрытие строк s_i и s_j, то есть наибольший суффикс s_i, который является префиксом s_j, и обозначим $overlap(s_i, s_j)$.

Префиксом (s_i, s_j) назовем префикс строки s_i, полученный после удаления из нее перекрытия (s_i, s_j), и обозначим $prefix(s_i, s_j)$. Перекрытие в суперстроке s^* между двумя последовательными строками s_i и s_{i+1} максимально, так как иначе можно получить более короткую суперстроку. Поскольку никакая из строк s_i не может быть подстрокой другой строки, получим

$$OPT = |prefix(s_1, s_2)| + |prefix(s_2, s_3)| + \ldots + |prefix(s_n, s_1)| + |overlap(s_n, s_1)|$$

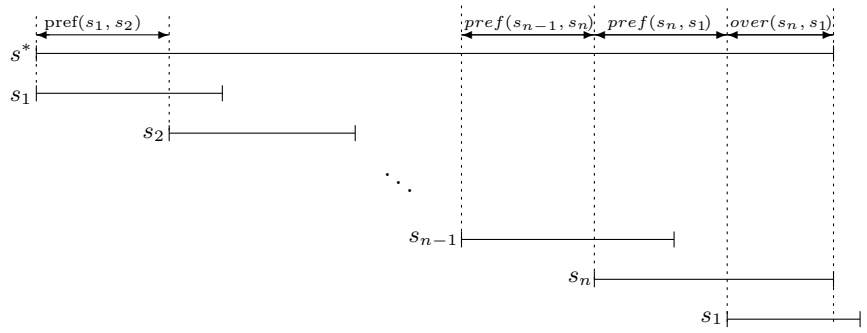


Рис. 2.8. Разбиение суперстроки на префиксы

Графом префиксов G_{pref} на множестве вершин $V = \{s_1, \ldots, s_n\}$ называется полный ориентированный граф, в котором каждая дуга (s_i, s_j) имеет вес $|prefix(s_i, s_j)|$. В построенном графе вес цикла $s_1 \to s_2 \to \ldots \to s_n \to s_1$ равен $|prefix(s_1, s_2)| + |prefix(s_2, s_3)| + \ldots + |prefix(s_n, s_1)|$. Следовательно, минимальный по весу гамильтонов цикл в графе префиксов является нижней оценкой на длину кратчайшей суперстроки. К сожалению, вычисление этой нижней оценки также является трудноразрешимой задачей. Чтобы обойти эту трудность, вместо одного цикла рассмотрим циклическое покрытие.

Циклическое покрытие — набор непересекающихся циклов, покрывающих все вершины. Заметим, что гамильтонов цикл также является циклическим покрытием, а значит, циклическое покрытие минимального веса определяет нижнюю оценку на длину кратчайшей суперстроки.

В отличие от задачи коммивояжера, задача о циклическом покрытии разрешима за полиномиальное время сведением к задаче о совершенном паросочетании минимального веса. Действительно, для графа префиксов построим соответствующий ему двудольный граф H с множеством вершин $U = \{u_1, \ldots, u_n\}$ в одной доле и $V = \{v_1, \ldots v_n\}$ в другой. Для любых $i \neq j \in \{1, \ldots, n\}$ определим ребро (u_i, v_j) веса $|prefix(s_i, s_j)|$. Нетрудно заметить, что каждому циклическому покрытию в графе префиксов G_{pref} соответствует совершенное паросочетание того же веса в графе H и наоборот.

Пусть $c = (i_1 \to i_2 \to \ldots \to i_l \to i_1)$ — цикл в графе G_{pref}. Обозначим через $w(c)$ вес цикла c и определим

$$\alpha(c) = \text{prefix}(s_{i_1}, s_{i_2}) \circ \ldots \circ \text{prefix}(s_{i_{l-1}}, s_{i_l}) \circ \text{prefix}(s_{i_l}, s_{i_1}),$$

$$w(c) = |\text{prefix}(s_{i_1}, s_{i_2})| + \ldots + |\text{prefix}(s_{i_{l-1}}, s_{i_l})| + |\text{prefix}(s_{i_l}, s_{i_1})|.$$

Через $(\alpha(c))^k$ для $k = 1, \ldots, \infty$ обозначим конкотенацию k строк $\alpha(c)$ в одну. Если $\alpha(c)$ длиннее s_{i_1}, то s_{i_1} содержится в $\alpha(c)$. Если $\alpha(c)$ короче, чем s_{i_1}, то s_{i_1} содержится в $(\alpha(c))^k$ для некоторого фиксированного k. Более того, каждая строка $s_{i_1}, s_{i_2}, \ldots, s_{i_l}$ является подстрокой $(\alpha(c))^\infty$. Пусть $\sigma(c) = \alpha(c) \circ s_{i_1}$. Тогда $\sigma(c)$ — суперстрока для $s_{i_1}, s_{i_2}, \ldots, s_{i_l}$.

В качестве примера рассмотрим цикл c, образованный из четырех строк, изображенных на рис. 2.9. В данном примере $\alpha(c) =$ abcde и $w(c) = 5$, строка $\alpha(c)$ короче, чем строка c_1. Кроме того, нетрудно проверить, что строки $s_{i_1}, s_{i_2}, s_{i_3}$ и s_{i_4} лежат в $(\alpha(c))^4 =$ abcdeabcdeabcdeabcde и $\sigma(c) =$ abcdeabcdeabcdeabcde.

Пусть $c = (i_1 \to i_2 \to i_3 \to i_4 \to i_1)$.

s_{i_1} abcdeabcdeabcde
s_{i_2} bcdeabcdeabcdea
s_{i_3} cdeabcdeabcdeab
s_{i_4} deabcdeabcdeabc
s_{i_1} abcdeabcdeabcde

Рис. 2.9. Пример цикла из четырех строк

Построим цикл c, начиная с произвольной строки s_{i_1}, которую будем называть *представителем цикла c*.

Алгоритм Суперстрока $(s_1, \ldots, s_n)$

1 Построить граф префиксов G_{pref} для строк $s_1, \ldots, s_n$.
2 Найти минимальное по весу циклическое покрытие графа G_{pref}.
Пусть это будет $C = \{c_1, \ldots, c_k\}$.
3 **return** $\sigma(c_1) \circ \ldots \circ \sigma(c_k)$.

Очевидно, что $\sigma(c_1) \circ \ldots \circ \sigma(c_k)$ является суперстрокой. Более того, если в каждом цикле есть представитель, длина которого не превосходит вес цикла, то решение не превосходит $2 \cdot OPT$. Рассмотрим решение, в котором строки длинные, а цикл короткий. Каждая такая строка является подстрокой $(\alpha(c))^\infty$. Следовательно, длинные строки должны быть периодическими. Это наблюдение позволяет построить новую нижнюю оценку на величину оптимального решения.

Лемма 5. *Если каждая строка в $U' \subseteq U$ является подстрокой t^∞ для строки t, то существует цикл веса не больше $|t|$ в графе префиксов, покрывающий все вершины, соответствующие строкам в U'.*

Доказательство. Упорядочим строки из U' по моментам их первого появления в t^∞. Все эти моменты различны и появляются в первой копии t. Рассмотрим цикл в графе префиксов, посещающий вершины в заданном порядке. Ясно, что вес этого цикла не превосходит $|t|$. □

Лемма 6. *Пусть $\mathcal{C}$ — циклическое покрытие минимального веса, c и c' — два цикла в $\mathcal{C}$ и r, r' — первые строки этих циклов. Тогда $|overlap(r, r')| < w(c) + w(c')$.*

Доказательство. Предположим, что $|overlap(r, r')| \geq w(c) + w(c')$. Отложим в перекрытии (r, r') префикс длины $w(c)$ и обозначим его α. Обозначим через α' префикс длины $w(c')$ в перекрытии (r, r') (рис. 2.10).

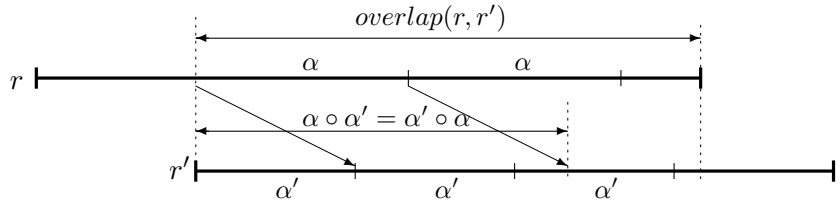


Рис. 2.10. Перекрытие (r, r')

Очевидно, что перекрытие (r, r') является префиксом и для α^∞ и для $(\alpha')^\infty$. Заметим также, что α является префиксом для $(\alpha')^\infty$, а α' — для α^∞. Так как $|overlap(r, r')| \geq |\alpha| + |\alpha'|$ (см. рис. 2.10), то это означает,

что α и α' коммутативны, то есть $\alpha \circ \alpha' = \alpha' \circ \alpha$. Но тогда $\alpha^\infty = (\alpha')^\infty$. Это следует из того, что для любого $k > 0$

$$\alpha^k \circ (\alpha')^k = (\alpha')^k \circ \alpha^k.$$

Поэтому для любого $N > 0$ префикс длины N для α^∞ совпадает с префиксом для $(\alpha')^\infty$.

Тогда по лемме 5 существует цикл веса не более $w(c)$ в графе префиксов, покрывающий все строки в c и c', но это противоречит тому факту, что $\mathcal{C}$ — циклическое покрытие минимального веса.

□

Теорема 10. Алгоритм суперстрока *является 4-приближенным алгоритмом для задачи о кратчайшей суперстроке.*

Доказательство. Пусть $w(\mathcal{C}) = \sum_{i=1}^{k} w(c_i)$. Алгоритм находит решение длины

$$\sum_{i=1}^{k} |\sigma(c_i)| = w(\mathcal{C}) + \sum_{i=1}^{k} |r_i|,$$

где r_i — первая строка в c_i. Понятно, что $w(\mathcal{C}) \leq OPT$. Покажем, что сумма длин первых строк не превосходит $3 \cdot OPT$.

Перенумеруем первые строки $r_1, \ldots, r_k$ в порядке их появления в кратчайшей суперстроке S. Используя лемму 6, получим

$$OPT \geq \sum_{i=1}^{k} |r_i| - \sum_{i=1}^{k-1} |overlap(r_i, r_{i+1})| \geq \sum_{i=1}^{k} |r_i| - 2\sum_{i=1}^{k} w(c_i).$$

Следовательно,

$$\sum_{i=1}^{k} |r_i| \leq OPT + 2\sum_{i=1}^{k} w(c_i) \leq 3 \cdot OPT.$$

□

2.4 Упражнения

1. Трудность задачи Штейнера состоит в поиске оптимального подмножества вершин Штейнера, которые необходимо включить в де-

рево. Показать, что если это множество известно, то задача Штейнера решается за полиномиальное время.

2. Найти в полном графе на рис. 2.11 приближенное решение задачи коммивояжера, используя Алгоритм Двойное остовное дерево и Алгоритм Кристофидеса – Сердюкова. Стоимости ребер (A, E), (B, F), (C, G) равны 19.

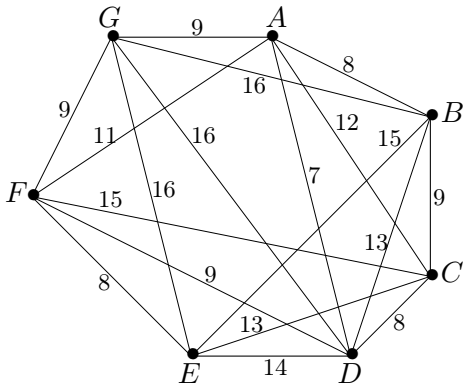


Рис. 2.11. Граф

3. Рассмотрим вариант метрической задачи коммивояжера, в котором необходимо найти простой путь, содержащий все вершины графа. В зависимости от числа заданных концов пути (0, 1 или 2) возникает три задачи.

   А. Построить 3/2-приближенный алгоритм для случая, если задана только одна конечная точка или ни одной.

   Б. Построить 5/3-приближенный алгоритм для случая, если заданы обе конечные точки.

4. Рассмотрим следующий алгоритм решения метрической задачи коммивояжера. Сначала найдем два самых близких города, обозначим их i и j. Построим цикл S, проходящий через них: $i \to j \to i$. На каждой следующей итерации найдем пару городов $i \in S$ и $j \notin S$ с минимальным расстоянием c_{ij}. Добавим город j в цикл

S сразу после города i. Доказать, что этот алгоритм является 2-приближенным.

5. Придумать приближенные алгоритмы для вариантов задачи о кратчайшей суперстроке. Обозначим за s^R — строку обратную к s.

   А. Найти кратчайшую строку, которая содержит все $s_i \in S$ и s_i^R как подстроки.

   Б. Найти кратчайшую строку, которая содержит либо $s_i \in S$, либо s_i^R как подстроку.

6. Пусть $\sigma(c_1) \circ \ldots \circ \sigma(c_k)$ — решение, найденное Алгоритмом суперстрока. Доказать, что если в каждом цикле есть представитель, длина которого не превосходит вес цикла, то решение не превосходит $2 \cdot OPT$.

3 *Задача о k-центрах*

— А ты уверен, что середина сосиски была именно на этом месте?

Шарик из мультфильма «Котенок по имени Гав»

Рассмотрим следующую задачу. В сельском районе расположено несколько небольших населенных пунктов, связанных дорогами. Необходимо построить k больниц так, чтобы минимизировать максимальное расстояние от каждого населенного пункта до ближайшей больницы. Назовем эту задачу задачей о k-центрах. Далее в главе рассматривается задача о k-центрах и ее взвешенная версия при условии, что расстояния между городами удовлетворяют неравенству треугольника. Без такого ограничения для задачи о k-центрах не существует $\alpha(n)$-приближенных алгоритмов для любой полиномиально вычислимой функции $\alpha(n)$ при условии, что $P \neq NP$.

Метрическая задача о k-центрах

Дано: полный граф $G = (V, E)$, стоимости ребер $cost : E \to \mathbf{Q}^+$ такие, что для любых трех вершин u, v и w выполняется $cost(u, v) \leq cost(u, w) + cost(w, v)$. Для любого множества $S \subseteq V$ определим $connect(v, S) = \min\{cost(u, v) | u \in S\}$.

Найти множество $S \subseteq V$, с $|S| = k$, которое минимизирует величину $\max_v\{connect(v, S)\}$.

3.1 Параметрическое сокращение для метрической задачи о k-центрах

Зачастую знание стоимости оптимального решения упрощает поиск хорошего приближенного решения. В частности, это позволяет отсечь заведомо неподходящие входные данные и сузить область поиска. К сожалению, вычисление стоимости оптимального решения имеет ту же сложность, что и нахождение самого решения NP-трудной оптимизационной задачи. Техника параметрического сокращения позволяет обходить эту трудность для некоторых задач. Выберем параметр t, который является предположением о стоимости оптимального решения. Для выбранного значения t из заданного примера I удалим те данные, которые не будут использоваться в любом решении стоимостью не больше t. Обозначим такой сокращенный пример как $I(t)$.

Алгоритм состоит из двух шагов: на первом шаге семейство примеров $I(t)$ используется для подсчета нижней оценки на OPT, назовем ее t^*, на втором шаге находится допустимое решение на примере $I(\alpha \cdot t^*)$, для некоторого подходящего параметра α.

Покажем, как технику параметрического сокращения применить к задаче о k-центрах. Занумеруем ребра графа G по неубыванию стоимостей, т. е. $cost(e_1) \leq cost(e_2) \leq \ldots \leq cost(e_m)$, и пусть $G_i = (V, E_i)$, где $E_i = \{e_1, e_2, \ldots, e_i\}$.

Доминирующим множеством в неориентированном графе $H = (U, F)$ называется подмножество $S \subset U$, такое, что каждая вершина из множества $U \setminus S$ смежна с некоторой вершиной из множества S.

Обозначим через $dom(H)$ мощность минимального (по мощности) доминирующего множества в H. Вычисление $dom(H)$ является NP-трудной задачей. Задача о k-центрах эквивалентна поиску минимального индекса i, такого, что G_i содержит доминирующее множество мощности не более k, т. е. G_i содержит k звезд, охватывающих все вершины, где *звезда* — это граф $K_{1,p}$, $p \geq 1$. Если i^* — это минимальный такой индекс, тогда величина $cost(e_{i^*})$ является оптимальным значением решения задачи о k-центрах. Обозначим эту величину как OPT.

Независимым множеством в графе $H = (U, F)$ называется подмножество вершин $I \subset U$, такое, что в нем нет смежных вершин.

Назовем *квадратом графа* $H = (V, F)$ граф $H^2 = (V, E)$, такой, что ребро $(u, v) \in E$ тогда и только тогда, когда граф H содержит путь между вершинами u и v, $u \neq v$, длины не более 2.

Лемма 7. *Пусть дан граф H, и пусть I — независимое множество в H^2. Тогда $|I| \leq dom(H)$.*

Доказательство. Пусть D — минимальное доминирующее множество в H. Тогда H содержит $|D|$ звезд, охватывающих все вершины, $D = dom(H)$. Поскольку каждая из звезд будет кликой в H^2, то H^2 содержит $|D|$ клик, содержащих все вершины. Так как в множестве I может лежать не более одной вершины из каждой клики H^2, то $|I| \leq dom(H)$. □

Алгоритм Хошбаум – Шмойса $(G = (V, E), cost : E \to \mathbf{Q}^+)$

1 Построить семейство графов $G_1^2, G_2^2, \ldots, G_m^2$.
2 Найти максимальное независимое множество I_r в каждом графе G_r^2.
3 Вычислить наименьший индекс r, такой, что $|I_r| \leq k$.
  Пусть это будет j.
4 **return** I_j.

Лемма 8. *Для j, полученного* алгоритмом Хошбаум – Шмойса, *верно, что $cost(e_j) \leq OPT$.*

Доказательство. Для каждого $i < j$ имеем, что $|I_i| > k$. По лемме 7 получаем $dom(G_i) \geq |I_i| > k$, поэтому $i^* > i$ и $j \leq i^*$. Отсюда $cost(e_j) \leq OPT$. □

Теорема 11. Алгоритм Хошбаум – Шмойса *является 2-приближенным алгоритмом для метрической задачи о k-центрах.*

Доказательство. Для начала заметим, что максимальное независимое множество I в графе G является также и доминирующим (поскольку, если некоторая вершина v не доминируется никакой из вершин из множества I, то $I \cup \{v\}$ является независимым множеством, а это противоречит тому, что I — максимальное независимое множество). В графе G_j^2 найдутся звезды с центрами в вершинах I_j, которые охватывают все вершины G. Из неравенства треугольника стоимость ребер в графе G_j^2 не превосходит $2 \cdot cost(e_j)$. Тогда из леммы 8 следует, что $2 \cdot cost(e_j) \leq 2 \cdot OPT$. □

Приведем пример, который показывает, что эта оценка точна. Пусть задан граф на $n+1$ вершинах, с одной центральной вершиной. Ребра, смежные с центральной вершиной, имеют стоимость 1, все остальные ребра имеют стоимость 2. Пример графа изображен на рис. 3.1.

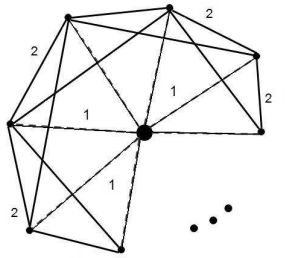


Рис. 3.1. Пример точности оценки Алгоритма Хошбаум – Шмойса

Для $k = 1$ оптимальное решение — это центральная вершина, и $OPT = 1$. Алгоритм Хошбаум – Шмойса получит индекс $j = n$. Граф G_n^2 — это полный граф, и если в качестве максимального независимого множества выбрана не центральная вершина, то стоимость решения, полученного алгоритмом, будет 2.

Теорема 12. *Для любого $\varepsilon > 0$ не существует $(2-\varepsilon)$-приближенного алгоритма для метрической задачи о k-центрах в предположении, что $P \neq NP$.*

Доказательство. Покажем, что если такой алгоритм существует, то он может решать задачу нахождения доминирующего множества за полиномиальное время. Пусть граф $G = (V, E)$ и число k являются входными данными задачи доминирующее множество. Построим полный граф $G' = (V, E')$ с функцией стоимостей

$$cost(u, v) = \begin{cases} 1, & \text{если } (u, v) \in E, \\ 2, & \text{если } (u, v) \notin E. \end{cases}$$

Заметим, что граф G' удовлетворяет неравенству треугольника и следующим условиям:

- если $dom(G) \leq k$, то G' имеет решение задачи о k-центре стоимости 1,
- если $dom(G) > k$, то стоимость оптимального решения задачи о k-центре в графе G' равна 2.

В первом случае $(2-\varepsilon)$-приближенный алгоритм должен найти решение стоимости 1, поскольку в $(2-\varepsilon)$-приближенном решении не могут использоваться ребра стоимости 2. Следовательно, такой алгоритм будет решать NP-полную задачу о доминирующем множестве за полиномиальное время. □

3.2 Задача о взвешенных центрах

Технику параметрического сокращения можно использовать и для решения метрической задачи о взвешенных центрах.

Метрическая задача о взвешенных центрах

Дано: полный граф $G = (V, E)$, стоимости ребер $cost : E \to \mathbf{Q}^+$ такие, что для любых трех вершин u, v и w выполняется $cost(u, v) \leq cost(u, w) + cost(w, v)$, веса вершин $w : V \to \mathbf{R}^+$ и верхняя граница $W \in \mathbf{R}^+$.

Найти множество $S \subseteq V$ суммарного веса $w(S) \leq W$, которое минимизирует величину $\max_v\{connect(v, S)\}$.

Обозначим через $wdom(G)$ вес доминирующего множества минимального веса в графе G. Для решения метрической задачи о взвешанных центрах требуется найти наименьший индекс i, такой, что $wdom(G_i) \leq W$. Пусть это будет индекс i^*, тогда стоимость оптимального решения $OPT = cost(e_{i^*})$.

Рассмотрим вершинно-взвешенный граф H, и пусть I — независимое множество в графе H^2. Для каждого $u \in I$ обозначим через $s(u)$ самого легкого соседа вершины u в графе H (вершина u также считается сама для себя соседом). Пусть $S = \{s(u)|u \in I\}$.

Лемма 9. *Для любого вершинно-взвешенного графа H верно неравенство $w(S) \leq wdom(H)$.*

Доказательство. Пусть D — минимальное по весу доминирующее множество в графе H. Тогда в H существует множество несвязных звезд с центрами в вершинах из D, покрывающих все вершины. Поскольку каждая из звезд становится кликой в графе H^2, то I содержит не более одной вершины из каждой звезды. При этом каждая вершина из I имеет центр соответствующей звезды как соседнюю вершину в графе H. Для каждой вершины из I множество S содержит самого легкого соседа. Следовательно, $w(S) \leq w(D)$. □

В Алгоритме Хошбаум – Шмойса–2 вершина $s_i(u)$ обозначает самого легкого соседа вершины u в графе G_i.

Алгоритм Хошбаум – Шмойса–2 $(G = (V, E), cost : E \to \mathbf{Q}^+)$

1 Построить семейство графов $G_1^2, G_2^2, \ldots, G_m^2$.
2 Найти максимальное независимое множество I_r в каждом графе G_r^2.
3 Построить $S_r = \{s_r(u) | u \in I_r\}$.
4 Вычислить наименьший индекс r, такой, что $w(S_r) \leq W$.
Пусть это будет j.
5 **return** S_j.

Теорема 13. Алгоритм Хошбаум – Шмойса–2 *является 3-приближенным алгоритмом для метрической задачи о взвешенных центрах.*

Доказательство. Покажем, что $cost(e_j)$ является нижней оценкой стоимости OPT. Действительно, для каждого $i < j$, согласно шагу 4 алгоритма, $w(S_i) > W$. По лемме 9 $wdom(G_i) \geq w(S_i) > W$. Следовательно, $i^* > i$ и $j \leq i^*$.

Так как I_j является доминирующим множеством в G_j^2, то можно покрыть все вершины V звездами в G_j^2 с центрами в вершинах I_j. Из неравенства треугольника вес ребер в звездах не превосходит $2 \cdot cost(e_j)$.

Каждый центр звезды связан с вершиной из S_j ребром стоимости не более чем $cost(e_j)$ (рис. 3.2). Сдвинем каждый центр в вершину из множества S_j, смежную с ним. Из неравенства треугольника получим, что $\max_v\{connect(v, S_j)\} \leq 3 \cdot cost(e_j)$. □

Покажем, что эта оценка достигается. Рассмотрим граф с $n + 4$ вершинами. Веса вершин и стоимости ребер представлены на рис. 3.3, все пропущенные ребра имеют стоимость, равную длине минимального пути.

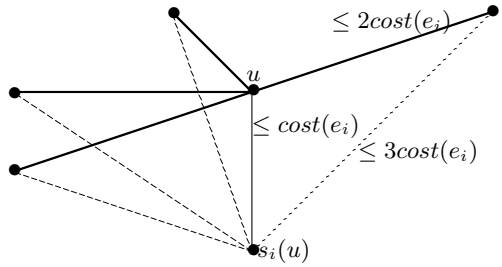


Рис. 3.2. Перераспределение звезд

Заметим, что при $W = 3$ оптимальное решение задачи о взвешенных центрах имеет вес $1 + \varepsilon$: в качестве центров выбираются вершины $\{a, c\}$. Для любого $i < n + 3$ в графе G_i^2 существует хотя бы одна изолированная вершина бесконечного веса, поэтому множество S_i, полученное алгоритмом, содержит вершины бесконечного веса. Предположим, что для $i = n + 3$ алгоритм выбрал $I_{n+3} = \{b\}$ в качестве максимального независимого множества. Тогда алгоритм выдаст $S_{n+3} = \{a\}$ в качестве решения. Вес такого решения равен 3.

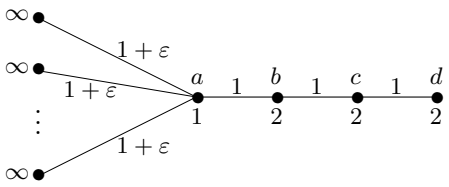


Рис. 3.3. Точность оценки Алгоритма Хошбаум – Шмойса–2

3.3 Упражнения

1. Рассмотрим следующую задачу. Расположить на шахматной доске двух королей так, чтобы минимизировать число ходов от одного из королей до любой клеточки доски. Сформулировать эту задачу как задачу о k-центрах и решить ее Алгоритмом Хошбаум – Шмойса.

2. Показать, что если стоимости ребер в графе не удовлетворяют неравенству треугольника, то для задачи о k-центрах не существует $\alpha(n)$-приближенного алгоритма для любой полиномиально вычислимой функции $\alpha(n)$.

3. Рассмотрим шаг 2 Алгоритма Хошбаум – Шмойса, на котором требуется найти максимальное независимое множество в графе G_i^2. Известны различные способы выбора минимального доминирующего множества. Изменим алгоритм так, чтобы I_r содержало минимальное доминирующее множество в графе G_i^2. Показать, что новый алгоритм не гарантирует оценку 2 для задачи о k-центрах. Какой гарантированной оценкой точности он обладает?

4. Рассмотрим следующий жадный алгоритм для задачи о k-центрах.

   Алгоритм Гонзалеза $(G = (V, E), cost : E \to \mathbf{Q}^+)$

   1 Выбрать произвольную вершину v и положить $S = \{v\}$.
   2 **while** $|S| < k$
   найти вершину v, максимально удаленную от множества S,
   т.е. $connect(v, S) = \max_{u \in V} connect(u, S)$.
   Добавить v в S.
   3 **return** S.

   Докажите, что Алгоритм Гонзалеза также является 2-приближенным алгоритмом для задачи о k-центрах.

Часть II

Приближенные схемы

4 *Общие принципы построения приближенных схем*

План, что и говорить, был превосходный; простой и ясный, лучше не придумать. Недостаток у него был только один: было совершенно неизвестно, как привести его в исполнение.

Льюис Кэрролл «Алиса в стране чудес»

В этой главе рассматриваются основные идеи и подходы к построению приближенных схем. В качестве примера, на котором будут иллюстрироваться эти идеи, выбрана задача минимизации длины расписания выполнения множества работ на двух параллельных идентичных машинах. Эту задачу принято обозначать $P2||C_{\max}$.

Задача $P2||C_{\max}$

Дано: множество работ $J = \{1, \ldots, n\}$, две идентичные параллельные машины M_1 и M_2, каждая работа должна быть выполнена на одной из двух машин, длительность каждой работы $p_j > 0$. Прерывания в выполнении работ запрещены, каждая машина обслуживает не более одной работы одновременно.

Найти расписание, имеющее минимальную длину $C_{\max}$.

Задача $P2||C_{\max}$ является NP-трудной в обычном смысле.

Процесс решения любой задачи можно описать следующей схемой. Алгоритм A получает входные данные примера I, обрабатывает их и получает решение $A(I)$ (рис. 4.1).

Рис. 4.1. Алгоритм A решает пример I и находит допустимое решение $A(I)$

Если оптимизационная задача X является трудной, то время работы точного алгоритма A может оказаться очень большим. В худшем случае оно оценивается экспоненциальной функцией от размера входа. Возникает вопрос: за счет чего можно улучшить поведение алгоритма, чтобы он находил хорошее решение за разумное время?

Представьте, что вы стоите в центре огромного лабиринта и вам требуется найти единственный выход из него. Ваша задача очень сложная: в лабиринте сотня комнат, и они связаны между собой разветвленной сетью коридоров. Вам потребуется уйма времени, чтобы обследовать его целиком.

Ваша задача заметно упростилась бы, если:

- в лабиринте было бы только несколько комнат (лучше всего одна) или один коридор, пусть даже и длинный,
- или лабиринт имел бы много выходов, равномерно разбросанных по всему лабиринту,
- или, например, все коридоры, ведущие в одно и то же место, были бы выкрашены в один цвет. Тогда, обследовав один из коридоров, можно было бы получить информацию обо всех коридорах того же цвета.

Первая ситуация соответствует упрощению исходных данных задачи, вторая ситуация описывает разбиение пространства решений, в третьей — предлагается упростить работу самого алгоритма. Каждому из предложенных способов будет посвящен отдельный раздел этой главы.

4.1 Упрощение исходных данных

Первая идея состоит в том, чтобы превратить трудный пример в более простой, в котором легко найти оптимальное решение, а затем использовать оптимальное решение простого примера для получения приближенного решения трудного примера. Более формально такой метод можно описать трехшаговой процедурой, схематично изображенной на рис. 4.2.

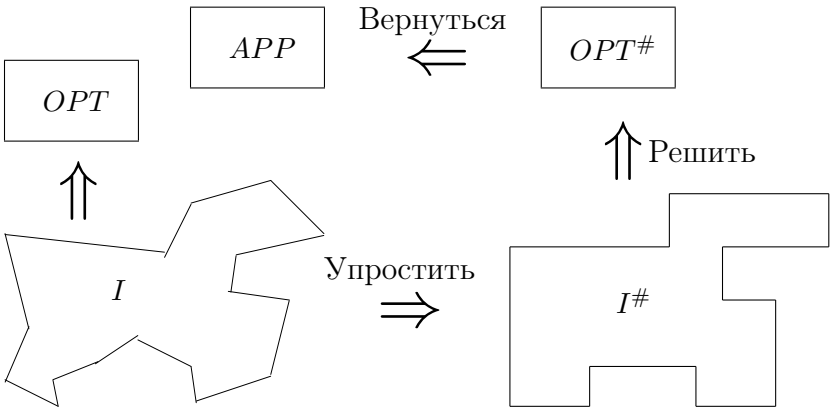


Рис. 4.2. Упрощение исходных данных

(A) Упрощение. Преобразовать пример I в более легкий пример $I^{\#}$. Степень упрощения зависит от желаемой точности приближения ε. Чем ближе будет ε к 0, тем больше пример $I^{\#}$ должен быть похож на пример I. Время, требуемое для построения упрощенного примера, должно быть ограничено полиномом от размера входа примера I.

(B) Решение. Найти оптимальное решение $OPT^{\#}$ упрощенного примера $I^{\#}$ за полиномиальное время.

(C) Возвращение. Перестроить решение $OPT^{\#}$ примера $I^{\#}$ в допустимое решение APP исходного примера I. Такое перестроение ис-

пользует сходство между примерами I и $I^{\#}$. В идеальном случае решение APP будет близким к решению $OPT^{\#}$, которое в свою очередь близко к OPT.

Конечно, поиск хорошего упрощения на шаге **(A)** — непростая задача. Если пример $I^{\#}$ будет слишком похож на исходный пример I, то поиск оптимального решения в $I^{\#}$ останется NP-трудной задачей. С другой стороны, если пример $I^{\#}$ выбран слишком далеко от I, то оптимальное решение $I^{\#}$ не поможет найти хорошее приближенное решение для исходного примера I. Недостаточное упрощение (например, если $I^{\#} = I$) или чрезмерное упрощение (например, если $I^{\#} = \emptyset$) одинаково опасны. Ниже приведем несколько известных приемов упрощения входа.

Округление. Простейший способ улучшить структуру данных — округлить их. Например, можно округлить длины всех работ к ближайшей степени числа 2 или заменить дробные величины целыми.

Слияние. Другим способом улучшения является слияние маленьких объектов в один или несколько больших. Например, можно объединить большое число маленьких работ в одну большую работу.

Отбрасывание. Следующий способ — это отбрасывание несущественных данных из примера. Например, можно выделить небольшое множество работ, удаление которых не влияет на структуру расписания.

Усреднение. Другой способ — это замена нескольких похожих объектов одинаковыми. Например, можно заменить 36 работ с почти одинаковыми длительностями на 36 одинаковых работ со средней длительностью.

Используя идею упрощения исходных данных, построим полиномиально приближенную схему для задачи $P2||C_{\max}$.

Длина расписания в задаче $P2||C_{\max}$ определяется общим временем работы наиболее загруженной машины. Длину оптимального расписания обозначим OPT. Общее время выполнения работ обозначим через $p_{sum} = \sum_{j=1}^{n} p_j$, а длительность самой большой работы обозначим через $p_{max} = \max_{j=1}^{n} p_j$. Пусть $L = \max\{\frac{1}{2}p_{sum}, p_{max}\}$, тогда

$$L \leq OPT.$$

Действительно, p_{max} является нижней оценкой, так как каждая работа должна целиком выполняться на одной машине. Кроме того, каждая машина за единицу времени выполняет единицу работы, поэтому $\frac{1}{2}p_{sum}$ также является нижней оценкой, которая достигается, когда нагрузка распределена поровну между машинами.

(А) Как упростить пример? Рассмотрим произвольный пример I задачи $P2||C_{\max}$, и пусть $\varepsilon > 0$ — заданное фиксированное число. В зависимости от параметра ε разделим все работы на большие и маленькие.

Работа j называется *большой*, если ее длительность $p_j > \varepsilon L$. Упрощенный пример $I^{\#}$ содержит все большие работы из примера I.

Работа j называется *маленькой*, если ее длительность $p_j \leq \varepsilon L$. Обозначим через S общее время выполнения всех маленьких работ в примере I. Тогда пример $I^{\#}$ будет содержать $\lfloor S/(\varepsilon L) \rfloor$ работ длительности εL. То есть маленькие работы как бы склеиваются вместе и разрезаются на одинаковые кусочки.

Лемма 10. $OPT(I^{\#}) \leq (1+\varepsilon)OPT(I)$.

Доказательство. Пусть S_i — общая длительность всех маленьких работ, выполняемых на машине M_i в оптимальном расписании $\sigma^*(I)$ для примера I. По расписанию $\sigma^*(I)$ построим допустимое расписание $\sigma(I^{\#})$ для примера $I^{\#}$. Все большие работы будут выполняться на тех же машинах, где они были в расписании $\sigma^*(I)$. Все маленькие работы на машине M_i заменим на $\lceil S_i/(\varepsilon L) \rceil$ одинаковых работ длительности εL. Поскольку

$$\lceil S_1/(\varepsilon L) \rceil + \lceil S_2/(\varepsilon L) \rceil \geq \lfloor S_1/(\varepsilon L) + S_2/(\varepsilon L) \rfloor = \lfloor S/(\varepsilon L) \rfloor,$$

то все работы длины εL назначены на машины. В результате будет получено допустимое расписание для примера $I^{\#}$. При этом нагрузка машины M_i увеличилась не более чем на

$$\lceil S_i/(\varepsilon L) \rceil \varepsilon L - S_i \leq (S_i/(\varepsilon L) + 1)\varepsilon L - S_i = \varepsilon L.$$

Из этого следует, что

$$OPT^{\#} \leq OPT + \varepsilon L \leq (1+\varepsilon)OPT.$$

□

(B) Как решать упрощенный пример? Оценим число работ в примере $I^{\#}$. При замене маленьких работ в примере I на одинаковые работы в примере $I^{\#}$ общее время их выполнения не увеличивается. Следовательно, оно не превосходит $p_{sum} \leq 2L$. Так как длительность каждой работы из примера $I^{\#}$ не больше чем εL, то их общее количество не превосходит $2L/(\varepsilon L) = 2/\varepsilon$. Следовательно, число работ в примере $I^{\#}$ не зависит от n. Переберем все возможные расписания. Их не более чем $2^{2/\varepsilon}$, и длину каждого такого расписания можно определить за время $O(2/\varepsilon)$. Следовательно, общее время решения примера $I^{\#}$ ограничено константой при фиксированном ε. Эта «константа» очень большая и растет экспоненциально от $1/\varepsilon$.

(C) Как вернуться к исходной задаче? Пусть $\sigma^{\#}$ — оптимальное расписание упрощенного примера $I^{\#}$. Через $L_i^{\#}$ обозначим загрузку машины M_i, через $B_i^{\#}$ обозначим суммарную длительность больших работ на машине M_i, и через $S_i^{\#}$ — общий размер всех маленьких работ на машине M_i в расписании $\sigma^{\#}$. Очевидно, что $L_i^{\#} = B_i^{\#} + S_i^{\#}$ и

$$S_1^{\#} + S_2^{\#} = \varepsilon L \cdot \lfloor \frac{S}{\varepsilon L} \rfloor > S - \varepsilon L.$$

Построим расписание σ для исходного примера I. Каждая большая работа выполняется на той же машине, что и в расписании $\sigma^{\#}$. Для маленьких работ выделим интервал длины $S_1^{\#} + 2\varepsilon L$ на машине M_1 и интервал длины $S_2^{\#}$ на машине M_2. Будем последовательно упаковывать маленькие работы в выделенный интервал на M_1, пока не встретим работу, которая туда не поместится. Оставшиеся маленькие работы упакуем в выделенный интервал на машине M_2. Поскольку размер маленькой работы не превышает εL, то общий размер упакованных работ на машине M_1, по крайней мере, — $S_1^{\#} + \varepsilon L$. Тогда размер оставшихся неупакованных работ не больше, чем $S - S_1^{\#} - \varepsilon L$, и не превышает $S_2^{\#}$. Следовательно, все они поместятся в выделенный интервал на машине M_2. Таким образом, получим расписание σ для исходного примера I.

Сравним загрузку машин L_1 и L_2 в расписании σ с загрузкой $L_1^{\#}$ и $L_2^{\#}$ в расписании $\sigma^{\#}$. Поскольку общий размер маленьких работ на машине M_i не превосходит $S_i^{\#} + 2\varepsilon L$, то, используя неравенства $L \leq OPT$ и $L_i^{\#} \leq OPT^{\#} \leq (1+\varepsilon)OPT$, получим

$$L_i \leq B_i^{\#} + (S_i^{\#} + 2\varepsilon L) = L_i^{\#} + 2\varepsilon L \leq (1+\varepsilon)OPT + 2\varepsilon OPT = (1+3\varepsilon)OPT.$$

Значит, длина расписания σ не более чем в $1 + 3\varepsilon$ раза превосходит длину оптимального расписания. Поскольку можно выбрать 3ε сколь угодно близким к 0, то описанная процедура позволяет построить полиномиально приближенную схему для задачи $P2||C_{\max}$.

Алгоритм Схема–1 $(J = \{1, \ldots, n\}, p : J \to \mathbf{Z}^+, \varepsilon)$

1 Определить множества работ **Big** $= \{j \in J | p_j \geq \varepsilon L\}$ и **Small** $= \{j \in J | p_j < \varepsilon L\}$.
2 Заменить все маленькие работы на машине M_i на $\lceil X/\varepsilon L \rceil$ работ длины εL.
3 Найти оптимальное расписание $\sigma^{\#}$ в новом примере $I^{\#}$, перебрав все допустимые назначения по машинам.
4 По расписанию $\sigma^{\#}$ построить допустимое расписание σ исходной задачи, используя описанную выше процедуру распределения маленьких работ по машинам.
5 **return** σ.

Теорема 14. Алгоритм Схема–1 *является* $(1 + \varepsilon)$*-приближенным алгоритмом для задачи* $P2||C_{\max}$.

4.2 Разбиение пространства решений

Вторая идея построения приближенной схемы состоит в разбиении пространства решений на несколько меньших областей, в которых проще найти хорошее приближенное решение. Если решить задачу отдельно для каждой маленькой области и взять среди них лучшее, есть шанс получить решение, близкое к оптимальному.

(A) Разбиение. Разобьем множество допустимых решений $\mathcal{F}$ примера I на d подмножеств $\mathcal{F}^{(1)}, \mathcal{F}^{(2)}, \ldots, \mathcal{F}^{(d)}$, таких, что $\bigcup_{\ell=1}^{d} \mathcal{F}^{(\ell)} = \mathcal{F}$. Такое разбиение зависит от заданной точности ε. При этом число d должно быть полиномиально ограничено от размера входа.

(B) Выбор представителя. Для каждого множества $\mathcal{F}^{(\ell)}$ определим *хорошего представителя* со значением целевой функции $App^{(\ell)}$, которое является хорошим приближением к оптимальному значению $OPT^{(\ell)}$. Время поиска представителя должно быть полиномиально ограничено от размера входа.

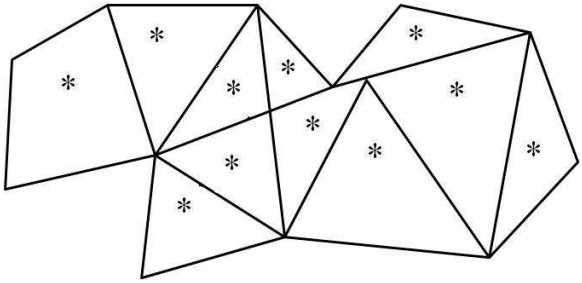


Рис. 4.3. Разбиение области допустимых решений

(C) Выбор наилучшего представителя. Из множества найденных хороших решений выбирается наилучший представитель со значением целевой функции *App* для примера I.

Основная трудность в реализации описанного подхода состоит в аккуратном разбиении на шаге **(A)**. Если разбиение выбрать слишком мелким (например, сформировать для каждого допустимого решения отдельную область $\mathcal{F}$), то получится экспоненциальное число таких областей и шаг **(C)** не удастся выполнить за полиномиальное время. Если же разбиение будет слишком грубым (например, область $\mathcal{F}$ будет содержать все или почти все решения), то выбор представителя на шаге **(B)** будет так же труден, как и решение исходной задачи. Таким образом, необходимо создать достаточно небольшое число областей, а точнее, полиномиально ограниченное от размера входа примера I, как и требуется на шаге **(A)**. При этом каждая область должна содержать все допустимые расписания, обладающие заданным общим свойством. Например, им может быть требование выполнять работу 3 в момент времени 7 на машине M_5. Заданное свойство фиксирует часть параметров, оставляя остальные свободными. Остается найти хорошее решение, используя только свободные параметры.

Вернемся к задаче построения расписания работ на двух идентичных машинах. Будем использовать технику разбиения пространства решений, как это было предложено Грэхамом в 1969 году.

(A) Как определить области? Пусть I — пример задачи $P2||C_{\max}$ и $\varepsilon > 0$ — заданная точность. Определим множество *маленьких работ*, длительность которых не превосходит εL, и множество *больших работ*, длительность которых строго больше εL. Заметим, что число больших работ в примере I не превосходит $2/\varepsilon$. Рассмотрим множество $\mathcal{F}$ допустимых решений примера I. Каждое допустимое решение $\sigma \in \mathcal{F}$ однозначно определяется назначением n работ по двум машинам.

Определим различные области $\mathcal{F}^{(1)}, \mathcal{F}^{(2)}, \ldots$, соответствующие одинаковым назначениям больших работ по машинам: два допустимых решения σ_1 и σ_2 лежат в одной области тогда и только тогда, когда все большие работы в расписании σ_1 назначены на те же машины, что и в расписании σ_2. Заметим, что назначение маленьких работ может быть произвольным. Поскольку больших работ не больше чем $2/\varepsilon$, то число их возможных назначений на две машины также не больше $2^{2/\varepsilon}$. Следовательно, число областей разбиения ограничено константой, не зависящей от размера входа.

(B) Как выбрать хорошего представителя? Рассмотрим некоторую область $\mathcal{F}^{(\ell)}$ и обозначим через $OPT^{(\ell)}$ длину лучшего расписания в этой области. Во всех расписаниях из области $\mathcal{F}^{(\ell)}$ назначения больших работ по машинам одинаковые. Обозначим через $B_i^{(\ell)}$ общую длительность больших работ, выполняющихся на машине M_i. Заметим, что

$$T = \max\{B_1^{(\ell)}, B_2^{(\ell)}\} \leq OPT^{(\ell)}.$$

Так как назначение больших работ фиксировано, то осталось распределить по машинам только маленькие работы. Начальная загрузка машин M_1 и M_2 равна $B_1^{(\ell)}$ и $B_2^{(\ell)}$ соответственно. Назначим маленькие работы одну за другой по следующему правилу: каждый раз работа назначается на машину с меньшей нагрузкой. В результате за линейное от числа работ время будет получено расписание $\sigma^{(\ell)}$ из заданной области $\mathcal{F}^{(\ell)}$ с длиной $APP^{(\ell)}$.

Сравним $APP^{(\ell)}$ и $OPT^{(\ell)}$. Если $APP^{(\ell)} = T$, то $\sigma^{(\ell)}$ является оптимальным расписанием в области $\mathcal{F}^{(\ell)}$. Рассмотрим случай, когда $APP^{(\ell)} > T$, и пусть загрузка машины M_i задает длину расписания $\sigma^{(\ell)}$. Рассмотрим последнюю работу, назначенную на машину M_i. Это должна быть маленькая работа. Ее длительность не превосходит εL. В тот момент, когда происходило назначение этой маленькой работы, загрузка машины

M_i не превышала $\frac{1}{2}p_{sum}$. Отсюда получим:

$$APP^{(\ell)} \leq \frac{1}{2}p_{sum} + \varepsilon L \leq (1+\varepsilon)L \leq (1+\varepsilon)OPT \leq (1+\varepsilon)OPT^{(\ell)}.$$

На шаге **(C)** просто выберем лучшее из найденных решений. Время каждого шага описанной процедуры ограничено полиномом от числа работ и $1/\varepsilon$, в результате получим полиномиальную приближенную схему.

Алгоритм Схема–2 ($J = \{1, \ldots, n\}, p : J \to \mathbf{Z}^+$)

1 Определить множества работ **Big** $= \{j \in J | p_j \geq \varepsilon L\}$
и **Small** $= \{j \in J | p_j < \varepsilon L\}$.
2 Определить области $\mathcal{F}^{(1)}, \mathcal{F}^{(2)}, \ldots \mathcal{F}^{(h)}$
согласно назначениям больших работ по машинам.
3 Сформировать задачи $I^{(1)}, I^{(2)}, \ldots, I^{(h)}$, в которых назначение
больших работ по машинам фиксировано и задано на входе.
4 Решить каждую из задач $I^{(\ell)}, \ell = 1, \ldots h$, назначая маленькие
работы одну за другой на машину с меньшей нагрузкой.
5 Пусть σ^* — лучшее из полученных расписаний на шаге 4.
6 **return** σ^*.

Теорема 15. Алгоритм Схема–2 *является* $(1+\varepsilon)$*-приближенным алгоритмом для задачи* $P2||C_{\max}$.

4.3 Структурирование работы алгоритма

Третьим стандартным способом построить приближенную схему является *структурирование работы алгоритма*. Основная идея — рассмотреть точный, но медленный алгоритм. Такой алгоритм получает и перерабатывает огромное число различных значений. Можно отбросить часть из них и ускорить работу алгоритма. В результате алгоритм становится быстрее, но при этом получает неточные решения. В идеальном случае алгоритм станет полиномиальным, а полученное решение — хорошим приближением оптимума.

Применим этот подход к задаче $P2||C_{\max}$. Обозначим через σ_k допустимое расписание из k первых работ. Закодируем расписание σ_k с на-

грузками машин L_1 и L_2 двумерным вектором $[L_1, L_2]$. Пусть V_k обозначает множество векторов, соответствующих допустимым расписаниям для k первых работ.

Алгоритм Динамическое программирование
$(J = \{1, \ldots, n\}, p : J \to \mathbf{Z}^+)$

1 Положить $V_0 = \{[0, 0]\}, i := 0$.
2 **while** $i \neq n$
для каждого вектора $[x, y] \in V_i$ добавить вектора
$[x + p_i, y]$ и $[x, y + p_i]$ в V_{i+1},
$i := i + 1$;
3 Найти $[x^*, y^*] \in V_n$, который минимизирует величину
$\max_{[x,y] \in V_n} \{x, y\}$.
4 **return** $[x^*, y^*]$.

Покажем, что Алгоритм Динамическое программирование находит точное решение задачи $P2||C_{\max}$. На итерации i шага 2 алгоритм назначает работу J_i либо на машину M_1, либо на машину M_2. В каждом случае нагрузка соответствующей машины увеличивается на величину p_i, и соответствующий новому распределению нагрузок вектор добавляется в V_i. Таким образом, алгоритм вычисляет вектора нагрузок для всех возможных назначений работ по машинам и на последнем шаге выбирает из них оптимальное.

Оценим теперь трудоемкость алгоритма. Поскольку координаты всех векторов — это целые числа от 0 до p_{sum}, то число различных векторов множества V_i не превосходит $(p_{sum})^2$. Общее количество векторов, вычисляемых алгоритмом, не превосходит $n(p_{sum})^2$. Трудоемкость алгоритма — $O(n(p_{sum})^2)$. В свою очередь, размер входа $|I|$ ограничен $O(\log(p_{sum})) = O(\ln(p_{sum}))$ или $O(n \log p_{max})$. Следовательно, трудоемкость алгоритма не ограничена полиномом от размера входа.

Как упростить множество векторов? Все вектора соответствуют точкам плоскости в квадрате $[0, p_{sum}] \times [0, p_{sum}]$. Разделим этот квадрат вертикальными и горизонтальными линиями на клетки. В обоих направлениях линии имеют координаты Δ^i для $i = 1, 2, \ldots, K$, где

$$\Delta = 1 + \frac{\varepsilon}{2n},$$

$$K = \lceil \log_\Delta(p_{sum}) \rceil = \lceil \ln(p_{sum}) / \ln \Delta \rceil \le \lceil (1 + 2n/\varepsilon) \ln(p_{sum}) \rceil.$$

Пусть два вектора $[x_1, y_1]$ и $[x_2, y_2]$ попали в одну клетку, тогда для них выполняется

$$x_1/\Delta \le x_2 \le x_1\Delta \quad \text{и} \quad y_1/\Delta \le y_2 \le y_1\Delta.$$

Поскольку Δ очень маленькое, то вектора в одной клетке располагаются близко друг к другу. Для каждого множества V_i выберем один вектор из каждой клетки, имеющей пересечение с V_i, и добавим его в *урезанное* множество векторов $V_i^{\#}$. Все остальные вектора из рассмотрения выбывают.

Алгоритм Схема–3 ($J = \{1, \ldots, n\}, p : J \to \mathbf{Z}^+$)

1 Положить $V_0^{\#} = \{[0, 0]\}, i := 0$.
2 **while** $i \ne n$
для каждого вектора $[x, y] \in V_i^{\#}$ добавить вектора
$[x + p_i, y]$ и $[x, y + p_i]$ в V_{i+1};
$i := i + 1$;
Преобразовать V_i в $V_i^{\#}$.
3 Найти $[x^*, y^*] \in V_n^{\#}$, который минимизирует величину
$\max_{[x,y] \in V_n^{\#}} \{x, y\}$.
4 **return** $[x^*, y^*]$.

Что можно сказать о трудоемкости алгоритма? Множество векторов в $V_i^{\#}$ содержит не более одного вектора из каждой клетки. Всего клеток — K^2. Трудоемкость Алгоритма Схема–3 — $O(nK^2)$. Поскольку $nK^2 = n\lceil (1 + 2n/\varepsilon) \ln(p_{sum}) \rceil^2$, то получим, что время работы Алгоритма Схема–3 ограничено полиномом от размера входа и $1/\varepsilon$. Осталось оценить точность решений, получаемых алгоритмом.

Лемма 11. *Для каждого вектора $[x, y] \in V_i$ существует вектор $[x^{\#}, y^{\#}] \in V_i^{\#}$, такой, что $x^{\#} \le \Delta^i x$ и $y^{\#} \le \Delta^i y$.*

Доказательство. Докажем по индукции. При $i = 1$ имеем $x_1/\Delta \le x_2 \le x_1\Delta$ и $y_1/\Delta \le y_2 \le y_1\Delta$. Теперь предположим, что утверждение леммы верно для $i-1$ и рассмотрим произвольный вектор $[x, y] \in V_i$. Для любого вектора из V_i существует вектор $[a, b] \in V_{i-1}$, такой, что либо $[x, y] =$

$[a + p_i, b]$, либо $[x, y] = [a, b + p_i]$. Будем считать, что $[x, y] = [a + p_i, b]$. По индукционному предположению существует вектор $[a^{\#}, b^{\#}] \in V^{\#}_{i-1}$, для которого $a^{\#} \leq \Delta^{i-1}a$ и $b^{\#} \leq \Delta^{i-1}b$. Алгоритм Схема–3 строит вектор $[a^{\#} + p_i, b^{\#}]$ и выбирает вектор $[\alpha, \beta]$, такой, что $\alpha \leq \Delta(a^{\#} + p_i)$ и $\beta \leq \Delta b^{\#}$. Для α имеем:

$$\alpha \leq \Delta(a^{\#} + p_i) \leq \Delta^i a + p_i \leq \Delta(a + p_i) = \Delta^i x.$$

Повторяя эти рассуждения для β, получим $\beta \leq \Delta^i y$. □

Теорема 16. Алгоритм Схема–3 — $(1 + \varepsilon)$-*приближенный алгоритм для задачи* $P2||C_{\max}$.

Доказательство. Для вектора $[x^{\#}, y^{\#}] \in V^{\#}_n$ имеем:

$$\max\{x^{\#}, y^{\#}\} \leq \max\{\Delta^n x, \Delta^n y\} = \Delta^n \max\{x, y\} = \Delta^n OPT.$$

Следовательно, алгоритм является Δ^n-приближенным алгоритмом для решения задачи $P2||C_{\max}$. Оценим, как велико Δ^n. По определению $\Delta = 1 + \frac{\varepsilon}{2n}$. Используем неравенство $(1 + \frac{z}{n})^n \leq 1 + 2z$, которое выполняется для $0 \leq z \leq 1$. Полагая $z = \varepsilon/2$, получим $\Delta^n \leq 1 + \varepsilon$.

□

4.4 Упражнения

1. Рассмотрим задачу построения расписания на двух параллельных машинах со скоростями $s_1 = 1$ и $s_2 = 2$. Для каждой работы J_i задан ее объем w_i. Время выполнения p_i работы J_i на k-ой машине равно $p_i = w_i/s_k$. Пусть Σ — сумма объемов всех работ, а W — объем максимальной работы. Укажите, какие из следующих величин являются нижними оценками на величину оптимального решения, и обоснуйте свое решение.

   (a) $\max\{\Sigma, W\}$,

   (b) $\max\{\Sigma/2, W/2\}$,

   (c) $\max\{\Sigma/3, W\}$,

   (d) $\max\{\Sigma/3, W/2\}$,

(e) $\max\{\Sigma/3, W/3\}$.

2. В Алгоритме Схема–1 для задачи $P2||C_{\max}$ на втором шаге все маленькие работы заменялись на $\lceil X/\varepsilon L \rceil$ одинаковых работ длины εL. Рассмотрим другой способ изменения маленьких работ. Возьмем две маленькие работы $p', p'' \leq \varepsilon L$ и склеим их в одну работу длительностью $p' + p''$. Будем склеивать маленькие работы, пока в примере есть по крайней мере две маленькие работы. Обозначим полученный пример $I^{\#}_{alt}$. Можно ли на основе этого построить приближенную схему для задачи $P2||C_{\max}$? Правда ли, что $OPT^{\#}_{alt} \leq (1+\varepsilon)OPT$? Можно ли ограничить число работ в полученном примере $I^{\#}_{alt}$? Как из оптимального расписания примера $I^{\#}_{alt}$ построить приближенное решение примера I?

3. Рассмотрим Алгоритм Схема–2 для задачи $P2||C_{\max}$. Определим области $\mathcal{F}^{(\ell)}$, как на шаге 2. Сначала выберем область $\mathcal{F}^{(\ell)}$, в которой достигается минимум величины $\max\{B^{\ell}_1, B^{\ell}_2\}$, а затем добавим маленькие работы одну за другой на машину с меньшей нагрузкой. Проверить, является ли полученный алгоритм приближенной схемой.

4. Рассмотрим n работ $J_1, \ldots, J_n$ с целыми положительными длительностями p_j и две машины. Необходимо найти расписание с загрузками машин L_1 и L_2, минимизирующих следующие целевые функции:

   (a) $|L_1 - L_2|$

   (b) $\max\{L_1, L_2\}/\min\{L_1, L_2\}$

   (c) $(L_1 - L_2)^2$

   (d) $L_1 + L_2 + L_1 \cdot L_2$

   (e) $\max\{L_1, L_2/2\}$

   Для каких случаев можно построить полиномиальную приближенную схему? Для каких нельзя?

5 *Асимптотическая приближенная схема для задачи об упаковке*

А десять мулов по два сундука — не мало ль будет?..

Касым из спектакля «Али-Баба и сорок разбойников»

Допустим, вы задумали переезд и решили упаковать свою библиотеку по коробкам. Объем коробок одинаковый, формат книг тоже одинаковый, но их толщина разная. На дно коробки помещается ровно одна книга. Необходимо минимизировать число используемых коробок. В итоге, получается классическая задача об упаковке.

Задача об упаковке

Дано: n предметов и их размеры $a_1, \ldots, a_n \in (0, 1]$.

Найти упаковку всех предметов в ящики размера 1, чтобы минимизировать число использованных ящиков.

Известно много простых эвристических алгоритмов для задачи об упаковке. Например, можно упаковывать предметы в произвольном порядке, используя следующее правило. Положим очередной предмет в первый ящик, в который он войдет. Если такого нет, то откроем новый ящик и положим предмет в него. Такое правило называют *Первый подходящий.*

Алгоритм Первый подходящий $(a_1, \ldots, a_n)$

1 $i \leftarrow 1$, $k \leftarrow 1$, $bin(1) \leftarrow 1$, $bin(2) \leftarrow 1$.
2 **while** $i \leq n$
$r \leftarrow \min\{j \leq k+1 | a_i \leq bin(j)\}$
if $r \leq k$
then $bin(r) \leftarrow bin(r) - a_i$
else $k \leftarrow k+1$
$bin(k) \leftarrow 1 - a_i$
$bin(k+1) \leftarrow 1$
3 **return** k

Теорема 17. Алгоритм Первый подходящий *является 2-приближенным алгоритмом для задачи об упаковке.*

Доказательство. Пусть открыто k ящиков. Следовательно, по крайней мере, $k-1$ ящик заполнен хотя бы наполовину. Отсюда,

$$\sum_{i=1}^{n} a_i > \frac{k-1}{2}.$$

Поскольку сумма размеров предметов является нижней оценкой на OPT, то $2 \cdot OPT > k-1$. Получим $2 \cdot OPT \geq k$. □

Рассмотрим известную NP-трудную задачу.

Задача о разбиении

Дано: n предметов и их веса $a_1, \ldots, a_n$, такие, что $\sum_i a_i = 2$.

Найти разбиение всех предметов на два множества, такое, что сумма весов предметов в каждом множестве одинакова.

Теорема 18. *Для любого $\varepsilon > 0$ не существует ρ-приближенного алгоритма для задачи об упаковке с $\rho = \frac{3}{2} - \varepsilon$, если $P \neq NP$.*

Доказательство. Предположим, что такой алгоритм существует. Покажем, как NP-трудную задачу о разбиении свести к проверке, можно ли разместить все предметы в два ящика. Заметим, что задача о разбиении имеет решение тогда и только тогда, когда все предметы можно упаковать ровно в два ящика. Если задача о разбиении не имеет решения, то

для упаковки понадобится не менее трех ящиков. Таким образом, если задача о разбиении имеет решение, то $(\frac{3}{2} - \varepsilon)$-приближенный алгоритм найдет упаковку всех предметов в 2 ящика, а значит, и решение задачи о разбиении. □

5.1 Асимптотическая приближенная схема

Заметим, что пример, построенный в доказательстве теоремы 18, — очень частный, в нем подразумевается, что OPT равен 2 или 3 даже при неограниченном числе предметов. Интересно, можно ли построить хороший алгоритм, когда OPT растет при увеличении n? Покажем, что для задачи об упаковке можно построить асимптотическую приближенную схему.

Семейство приближенных алгоритмов $\{A_\varepsilon\}$ называется *ассимптотической полиномиально приближенной схемой* для задачи оптимизации П, если для любого $\varepsilon > 0$ существует константа c_ε такая, что алгоритм A_ε находит решение, удовлетворяющее неравенству

$$A_\varepsilon(I) \leq (1 + \varepsilon)OPT + c_\varepsilon.$$

Лемма 12. *Пусть $\varepsilon > 0$ — некоторая константа и K — фиксированное положительное целое число. Рассмотрим примеры задачи об упаковке, в которых размеры предметов не меньше ε, и число различных размеров равно K. Тогда существует полиномиальный алгоритм, который точно решает такие примеры.*

Доказательство. Рассмотрим множество предметов, помещенных в один ящик. Пусть x_i — число предметов i-го размера в ящике. Тогда вектор $(x_1, \ldots, x_K)$ определяет тип возможной загрузки ящика. Число предметов, помещающихся в один ящик, не превосходит $\lfloor \frac{1}{\varepsilon} \rfloor$. Положим $M = \lfloor \frac{1}{\varepsilon} \rfloor$. Тогда число различных типов ящиков не превосходит $R = C^M_{M+K}$. Общее число ящиков не превосходит n. Значит, число возможных допустимых упаковок не превосходит $P = C^{n+R-1}_R$ и ограничено полиномом от n. Переберем все упаковки и выберем среди них оптимальную. □

Рассмотрим задачу упаковки больших предметов.

Лемма 13. *Пусть $\varepsilon > 0$ — некоторая константа. Рассмотрим примеры задачи об упаковке, в которых размеры предметов не меньше ε. Тогда существует полиномиальный алгоритм, который находит упаковку в не более чем $(1+\varepsilon)OPT$ ящиков, где OPT — число ящиков в оптимальной упаковке.*

Доказательство. Пусть I — пример задачи, удовлетворяющий условиям леммы. Упорядочим все предметы по неубыванию размеров. Разобьем их на $K = \lceil 1/\varepsilon^2 \rceil$ групп так, что все группы кроме последней содержат $Q = \lfloor n\varepsilon^2 \rfloor$ предметов. Таким образом, для $1 \le i \le K-1$ в i-ю группу попадут предметы с номерами $(i-1)Q+1, (i-1)Q+2, \ldots, iQ$. Все оставшиеся предметы попадут в последнюю K-ю группу. Заметим, что две группы могут содержать элементы одного размера. Построим пример J, округляя размер каждого предмета до размера наибольшего в группе. Пример J имеет не более K различных размеров предметов. Тогда по лемме 12 можно найти оптимальную упаковку для примера J за полиномиальное время. Заметим, что такая упаковка будет допустимой и для исходного примера. Осталось показать, что $OPT(J) \le (1+\varepsilon)OPT(I)$.

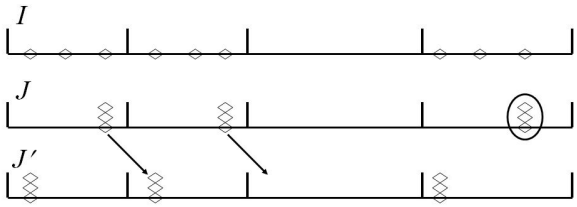


Рис. 5.1. Преобразование примера I в примеры J и J' и соотношения между размерами в множествах J и J'

Построим новый пример J', округляя размер каждого элемента до размера наименьшего элемента в группе. Очевидно, что выполняется $OPT(J') \le OPT(I)$. Заметим, что размер предметов в i-ой группе в примере J не больше, чем размер предметов в $i+1$-ой группе в примере J' для всех $i = 1, \ldots, K-1$. Назовем предметы, попавшие в K-ую группу, *крупными*, а остальные предметы — *мелкими*.

Каждому мелкому предмету в группе i из примера J инъективно сопоставим некоторый предмет из группы $i+1$ из примера J'. Соотношение между размерами предметов из различных групп для примеров I, J, J' проиллюстрировано на рис. 5.1. Овалом выделено множество крупных предметов. Рассмотрим оптимальную упаковку U' в примере J'. Поместим каждый мелкий предмет i в примере J в ящик, где лежал предмет $f(i)$ в упаковке U' для примера J'. Так как каждый предмет в упаковке U' либо будет заменен на предмет меньшего размера, либо вообще удален, то общий размер предметов в каждом ящике может только уменьшаться. Таким образом, число ящиков, в которые помещены маленькие предметы из примера J, не превосходит $OPT(J')$. Поместим каждый крупный предмет в отдельный ящик. Получим

$$OPT(J) \leq OPT(J') + Q \leq OPT(I) + Q.$$

Так как каждый элемент в примере I имеет размер не меньше ε, то $OPT \geq n\varepsilon$. Значит, $Q = \lfloor n\varepsilon^2 \rfloor \leq \varepsilon OPT$. Отсюда следует, что $OPT(J) \leq (1+\varepsilon)OPT(I)$. □

Алгоритм Фернандес де ла Вега – Луекера $(a_1, \ldots, a_n)$

1 Определить множества **Big** $= \{j | a_j \geq \varepsilon\}$ и **Small** $= \{j | a_j < \varepsilon\}$.
2 Разбить большие предметы на $K = \lceil \frac{1}{\varepsilon^2} \rceil$ групп
по не более чем $Q = \lfloor n\varepsilon^2 \rfloor$ предметов в каждой.
3 Округлить размер каждого большого предмета
к размеру наибольшего в группе.
4 Упаковать большие предметы оптимальным образом
относительно новых размеров.
5 Упаковать маленькие предметы, используя правило
первый подходящий. Пусть k — число использованных ящиков
в полученной упаковке.
6 **return** k

Теорема 19. *Для любого фиксированного* $0 < \varepsilon < 1/2$ Алгоритм Фернандес де ла Вега – Луекера *находит упаковку, использующую не более* $(1 + 2\varepsilon)OPT + 1$ *ящиков, и время его работы ограничено полиномом от* n.

Доказательство. Пусть I' — пример, полученный из заданного примера I удалением работ из множества **Small**, т. е. тех, длительность которых меньше ε. По лемме 13 на шаге 4 алгоритм найдет упаковку предметов примера I' в не более чем $(1 + \varepsilon)OPT(I')$ ящиков. Если на шаге 5, при вставке маленьких работ, не потребуется ни одного нового ящика, то $k = (1 + \varepsilon)OPT(I') \leq (1 + \varepsilon)OPT(I)$. В противном случае, по крайней мере, $k-1$ ящик должен быть заполнен не меньше, чем на $1-\varepsilon$. Следовательно, сумма размеров предметов в примере I, по крайней мере, — $(k-1)(1-\varepsilon) \leq OPT(I)$. Используя это как нижнюю оценку на OPT, получим

$$k \leq OPT(I)/(1 - \varepsilon) + 1 \leq (1 + 2\varepsilon)OPT(I) + 1,$$

где $0 < \varepsilon < 1/2$. Таким образом, для каждого $0 < \varepsilon < 1/2$ получен полиномиальный алгоритм с гарантированной оценкой $(1 + 2\varepsilon)OPT + 1$. □

5.2 Упражнения

1. Построить пример, на котором Алгоритм Первый подходящий находит решение хуже, чем $5/3 \cdot OPT$.
2. Рассмотрим вариант Алгоритма Первый подходящий, называемый Алгоритм Следующий подходящий. Если очередной предмет не помещается в последний открытый ящик, то открываем новый пустой ящик. Показать, что этот алгоритм является 2-приближенным, и оценка 2 является точной.
3. Назовем алгоритм упаковки *монотонным*, если число использованных ящиков, найденное алгоритмом для подмножества предметов, не превосходит числа использованных ящиков для упаковки всего множества предметов. Определить, являются ли Алгоритм Первый подходящий и Алгоритм Следующий подходящий монотонными.

6 *Построение расписания работ на идентичных машинах*

— Ну, граждане алкоголики, хулиганы, тунеядцы... Кто хочет сегодня поработать? На сегодня наряды: песчаный карьер — два человека...

милиционер из кинофильма
«Операция "Ы" и другие
приключения Шурика»

В разделе 4 рассматривалась задача $P2||C_{\max}$ построения минимального по длине расписания работ на двух идентичных параллельных машинах. Для нее были построены две полиномиальные приближенные схемы и одна вполне полиномиальная приближенная схема. Заметим, что такие же схемы будут работать, когда число машин равно 3, 4 или другой фиксированной константе.

В этом разделе построим полиномиальную приближенную схему для более общей задачи, в которой число машин произвольно и неизвестно заранее.

Задача $P||C_{\max}$

Дано: множество работ $J = \{1, \ldots, n\}$, m идентичных параллельных машин, каждая работа должна быть выполнена на одной из этих машин, длительность каждой работы $p_j > 0$. Прерывания запрещены, каждая машина обслуживает не более одной работы одновременно.

Найти расписание, имеющее минимальную длину $C_{\max}$.

6.1 Жадный алгоритм

Как и в задаче с двумя машинами, длина расписания полностью определяется распределением работ по машинам. Пусть $\sigma(j)$ обозначает номер машины, на которую назначена работа j. Тогда *расписанием* будем называть n-мерный вектор $\bar{\sigma} = (\sigma(1), \ldots, \sigma(n))$.

Обозначим через L_i загрузку машины M_i.

Алгоритм Грэхема $(m, p_1, \ldots, p_n)$

1 $L_i \leftarrow 0$ для всех $i = 1, \ldots, n$.
2 **for** $j = 1$ to n
  do Найти машину M_i, у которой L_i минимально.
  Назначить работу j на машину M_i: $\sigma(j) \leftarrow i$,
  $L_i \leftarrow L_i + p_i$.
3 $C_{\max} \leftarrow \max_{i=1,\ldots,m} L_i$.
4 **return** $\sigma, C_{\max}$.

Теорема 20 (Грэхем). Алгоритм Грэхема *является* $(2 - \frac{1}{m})$-*приближенным алгоритмом для задачи* $P||C_{\max}$.

Доказательство. Пусть M_i — машина, на которой выполняется работа j, определяющая длину расписания. Обозначим через s_j время, когда работа j начинает выполняться на машине M_i. Поскольку алгоритм назначает работу на ближайшую свободную машину, это означает, что все машины заняты до момента s_j, и в промежутке $[0, s_j]$ они выполняют

работы, отличные от работы j. Следовательно,

$$s_j \leq \frac{1}{m}\sum_{i=1}^{n} p_i - p_j \leq OPT - \frac{p_j}{m}.$$

Учитывая, что $p_j \leq OPT$, получим оценку на длину расписания, построенного алгоритмом Грэхема $s_j + p_j \leq (2 - \frac{1}{m})OPT$. □

Покажем, что эта оценка точна. Рассмотрим пример с m^2 единичными работами и одной работой длины m. Если рассмотреть работы в указанном порядке, то Алгоритм Грэхема найдет расписание длины $2m$, в то время как длина оптимального расписания $m + 1$.

6.2 Связь задачи $P||C_{\max}$ с задачей об упаковке

Заметим, что в задаче $P||C_{\max}$ существует расписание длины t тогда и только тогда, если n предметов веса $p_1, \ldots, p_n$ можно упаковать в m контейнеров вместимости не более t каждый. Рассмотрим пример I задачи об упаковке, в котором задано n предметов и их размеры $p_1, \ldots, p_n$. Через $\text{bins}(I, t)$ обозначим минимальное число ящиков размера t, в которые можно разложить эти n предметов. Тогда минимальную длину расписания можно определить как

$$OPT = \min\{t : \text{bins}(I, t) \leq m\}. \tag{6.1}$$

Чтобы осуществить это сведение, необходимо знать или угадать значение $C_{\max}$ в оптимальном решении и уметь оптимально упаковывать работы по ящикам фиксированного размера. Сначала детально рассмотрим задачу упаковки.

Пусть число различных размеров предметов фиксировано и равно k. Зададим их порядок и предположим, что каждый ящик имеет вместимость 1. Каждый пример задачи об упаковке можно представить как вектор из k компонент, $(i_1, i_2, \ldots, i_k)$, указывающий число предметов каждого размера. Тогда через $\text{BINS}(i_1, \ldots, i_k)$ обозначим минимальное число ящиков, требуемое, чтобы упаковать это множество предметов.

Для данного примера $(n_1, n_2, \ldots, n_k)$ вычислим множество $\mathcal{Q}$ всех векторов $(q_1, \ldots, q_k)$, таких, что $\text{BINS}(q_1, \ldots, q_n) = 1$ и $0 \leq q_i \leq n_i, 1 \leq i \leq k$. Поскольку $\sum_{i=1}^{k} n_i = n$, то $\mathcal{Q}$ содержит не более n^k элементов.

Вычислим все значения k-размерной таблицы $\mathrm{BINS}(i_1, \ldots, i_k)$ для каждого $(i_1, \ldots, i_k) \in \{0, \ldots, n_1\} \times \{0, \ldots, n_2\} \times \{0, \ldots, n_k\}$. Сначала заполним эту таблицу значениями $\mathrm{BINS}(q) = 1$ для каждого $q \in \mathcal{Q}$. Затем вычислим остальные значения по формуле:

$$\mathrm{BINS}(i_1, \ldots, i_k) = 1 + \min_{q \in \mathcal{Q}} \mathrm{BINS}(i_1 - q_1, \ldots, i_k - q_k).$$

Все значения в таблице могут быть вычислены за $O(n^{2k})$ элементарных операций.

6.3 Полиномиальная приближенная схема для задачи $P||C_{\max}$

Зафиксируем параметры $\varepsilon > 0$ и $t \in [LB, 2LB]$.

Алгоритм Упаковка работ $(p_1, \ldots, p_n, \varepsilon, t)$

1 Определить множества работ **Big** $= \{j | p_j \geq t\varepsilon\}$
и **Small** $= \{j | p_j < t\varepsilon\}$.
2 Округлить размер каждой большой работы:
if $(p_j \in [t\varepsilon(1+\varepsilon)^i, t\varepsilon(1+\varepsilon)^{i+1}))$
then $p'_j \leftarrow t\varepsilon(1+\varepsilon)^i$.
3 Используя динамическое программирование, найти оптимальную упаковку U больших предметов размера p'_j в ящики размера t.
4 Расширить ящики до размера $t(1+\varepsilon)$.
Вернуть большим предметам исходные размеры p_j.
5 Упаковать маленькие предметы в свободное пространство, используя правило *Первый подходящий*.
Пусть $\alpha(I, \varepsilon, t)$ — число использованных ящиков в этой упаковке.
6 **return** $\alpha(I, \varepsilon, t)$

Число различных значений размеров p'_j можно ограничить величиной $k = \lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil$. Трудоемкость основной процедуры определяется трудоемкостью динамического программирования и равна $O(n^{2k})$. При фиксированном ε трудоемкость процедуры полиномиально зависит от n.

Лемма 14. $\alpha(I, \varepsilon, t) \leq bins(I, t)$.

Доказательство. Пусть на шаге 5 алгоритма не был открыт ни один новый ящик. Утверждение леммы верно, так как в этом случае $\alpha(I, \varepsilon, t)$ является оптимальным решением задачи без маленьких предметов и с уменьшенными размерами больших предметов.

В противном случае все ящики, кроме одного, должны быть заполнены не меньше чем на t. Следовательно, сумма размеров предметов больше, чем $t(\alpha(I, \varepsilon, t) - 1)$, и нужно как минимум $\alpha(I, \varepsilon, t)$ ящиков, чтобы упаковать все предметы. □

Из леммы 14 и равенства 6.1 получим нижнюю оценку на длину расписания.

Следствие 2. $\min\{t : \alpha(I, \varepsilon, t) \leq m\} \leq OPT$.

Доказательство. $OPT = \min\{t : bins(I, t) \leq m\}$. По лемме 14 для любого $t : \alpha(I, \varepsilon, t) \leq bins(I, t)$. Отсюда, $\min\{t : \alpha(I, \varepsilon, t) \leq m\} \leq OPT$.

□

Осталось определить, как найти t. Воспользуемся для этого тривиальной нижней оценкой

$$LB = \max\left\{\frac{1}{m}\sum_i p_i, \max_i\{p_i\}\right\}.$$

Из доказательства теоремы 20 получим

$$LB \leq OPT \leq 2LB \quad \text{и} \quad \alpha(I, \varepsilon, 2LB) \leq m.$$

Поэтому можно угадать значение оптимального решения, устроив бинарный поиск в интервале $[LB, 2LB]$ и используя в качестве проверки Алгоритм Упаковка работ. Пусть для двух значений $t_0 \leq t_1$ из интервала $[LB, 2LB]$ известно, что $\alpha(I, \varepsilon, t_0) > m$ и $\alpha(I, \varepsilon, t_1) \leq m$. Рассмотрим $t = \frac{t_0+t_1}{2}$. Если, применив Алгоритм Упаковка работ, получим $\alpha(I, \varepsilon, t) > m$, то продолжаем бинарный поиск в интервале $[t, t_1]$, в противном случае — в интервале $[t_0, t]$.

На каждой итерации длина интервала уменьшается вдвое. Остановим процедуру бинарного поиска, когда длина интервала станет εLB, и обозначим его $[T - \varepsilon LB, T]$. Это потребует $\lceil \log_2 \frac{1}{\varepsilon} \rceil$ итераций. Пусть T определяет правую точку интервала.

Лемма 15. $T \leq (1+\varepsilon)OPT$.

Доказательство. Заметим, что значение $\min\{t : \alpha(I, \varepsilon, t) \leq m\}$ должно быть в интервале $[T - \varepsilon LB, T]$. Следовательно,

$$T \leq \min\{t : \alpha(I, \varepsilon, t) \leq m\} + \varepsilon LB.$$

Используя следствие 2 и ограничение $LB \leq OPT$, получим требуемое. □

Напомним, что для $t = T$ алгоритм дает расписание длины не более $(1+\varepsilon)T$. В итоге, получаем следующий результат.

Теорема 21. *Для любого $\varepsilon > 0$ существует алгоритм A_ε, который находит расписание длины не больше $(1+\varepsilon)^2 OPT \leq (1+3\varepsilon)OPT$ за $O(n^{2k}\lceil \log_2 \frac{1}{\varepsilon} \rceil)$ операций, где $k = \lceil \log_{1+\varepsilon} \frac{1}{\varepsilon} \rceil$.*

6.4 Упражнения

1. С помощью полиномиальной приближенной схемы решить задачу с 15 работами и тремя машинами с точностью $\varepsilon = \frac{1}{3}$. Длительности работ считать равными 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15.
2. В примере, показывающем точность Алгоритма Грэхема, последней идет самая длинная работа. Показать, что если отсортировать работы в порядке невозрастания длительностей, то алгоритм будет давать оценку 4/3. Доказать, что эта оценка точна.
3. Построить вполне полиномиальную приближенную схему для задачи минимизации длины расписания на m параллельных машинах, где m — заданная константа.

Часть III

Линейное программирование

7 *Введение в линейное программирование*

— Шурик! Вы комсомолец?
— Да.
— Это же не наш метод!

диалог Феди и Шурика из
кинофильма «Операция "Ы" и
другие приключения Шурика»

Большая часть известных приближенных алгоритмов для задач комбинаторной оптимизации основана на решении задач линейного программирования. Задача линейного программирования (задача ЛП) — это оптимизационная задача с линейной целевой функцией $z(x)$ и линейными ограничениями:

$$z(x) = c_1 x_1 + \cdots + c_n x_n \to \min$$

$$a_{11} x_1 + \ldots + a_{1n} x_n \geq b_1$$

$$\vdots$$

$$a_{m1} x_1 + \ldots + a_{mn} x_n \geq b_m$$

$$x_i \geq 0 \;\; i = 1, \ldots, n.$$

Существуют различные формы представления задачи линейного программирования как системы неравенств и равенств с заданной целевой

функцией. Представление, приведенное выше, называется задачей линейного программирования в стандартной форме. Путем нехитрых алгебраических преобразований любую задачу линейного программирования, в которой требуется минимизировать некоторую целевую функцию, можно представить как задачу линейного программирования в стандартной форме.

Проблема решения задачи линейного программирования долгое время была краеугольным камнем в дискретной оптимизации. Еще в классической монографии М. Гэри и Д. Джонсона, вышедшей в 70-х годах XX века, она фигурировала как одна из важнейших открытых задач в теории комбинаторной сложности. Многочисленные теоретические результаты, полученные к тому времени, указывали на то, что задача не должна быть NP-трудной, однако точный полиномиальный алгоритм ее решения не был известен.

Прорыв состоялся в 1979 году, когда советский математик Л. Хачиян показал, как метод эллипсоидов, первоначально разработанный для решения задач нелинейной недифференцируемой оптимизации, можно использовать для проверки допустимости системы линейных ограничений за полиномиальное время. Этот результат имел огромное значение в области математического программирования, так как из него следовала разрешимость задачи линейного программирования за время, ограниченное полиномом от размера входных данных.

При этом необходимо отметить, что, несмотря на теоретическую эффективность метода эллипсоидов, его практическая реализация сопряжена со значительными трудностями. Почти для всех примеров задачи линейного программирования метод эллипсоидов находит решение медленнее, чем симплекс-алгоритм, разработанный Г. Данцигом в конце 40-х годов прошлого столетия. В свою очередь, симплекс-алгоритм, хотя и не относится к полиномиальным алгоритмам, позволяет решать на современных компьютерах практические задачи большой размерности за очень короткое время. Таким образом, задача линейного программирования эффективно разрешима как с теоретической, так и с практической точек зрения. Кроме того, в рассматриваемой задаче есть много интересных теоретических аспектов, особенно в связи с комбинаторными задачами. Большинство из них так или иначе связаны с идеей двойственности, речь о которой пойдет в следующем разделе.

7.1 Крайние точки и двойственность

Рассмотрим задачу ЛП в стандартной форме:

$$\sum_{j=1}^{n} c_j x_j \to \min$$

$$\sum_{j=1}^{n} a_{ij} x_j \geq b_i, \;\; i = 1, \ldots, m,$$

$$x_j \geq 0, \;\; j = 1, \ldots, n,$$

где a_{ij}, b_i и c_j – рациональные числа.

Пусть $x = \{x_1, \ldots, x_n\}$ — вектор в n-мерном евклидовом пространстве R^n. Множество $\mathcal{S}$ всех векторов x, удовлетворяющих указанным ограничениям, называется *допустимым множеством*, а любое $x \in \mathcal{S}$ называется допустимым решением или допустимой точкой. Если x^1 и x^2 принадлежат $\mathcal{S}$, то множеству $\mathcal{S}$ принадлежит и $\alpha x^1 + (1-\alpha)x^2$, $0 \leq \alpha \leq 1$, то есть $\mathcal{S}$ представляет собой выпуклое многогранное множество, образованное системой линейных неравенств.

Точка $x \in \mathcal{S}$ называется *крайней точкой* выпуклого множества, если не существует двух отличных от x точек $y, z \in \mathcal{S}$, таких, что $x = \alpha y + (1-\alpha)z$ для некоторого $0 \leq \alpha \leq 1$.

Теорема 22. *Если задача ЛП в стандартной форме имеет оптимальное решение, то существует оптимальное решение, которое является крайней точкой.*

Отметим, что при решении задачи ЛП и алгоритм эллипсоидов, и симплекс-метод находят решения, которые являются крайними точками.

Назовем рассмотренную выше задачу ЛП прямой задачей ЛП.

Вводя переменные y_i для i-го неравенства, получим двойственную задачу ЛП:

$$\sum_{i=1}^{m} b_i y_i \to \max$$

$$\sum_{i=1}^{m} a_{ij} y_i \leq c_j, \;\; j = 1, \ldots, n,$$

$$y_i \geq 0, \;\; i = 1, \ldots, m.$$

Заметим, что задача, двойственная к двойственной задачи ЛП, совпадает с прямой задачей ЛП. Для прямой и двойственной задач выполняются основные теоремы двойственности.

Теорема 23 (1-ая теорема двойственности). *Прямая и двойственная к ней задачи либо одновременно разрешимы, либо одновременно неразрешимы. При этом в первом случае значения целевых функций этих задач совпадают, а во втором случае, по крайней мере, одна из задач неразрешима в силу несовместности ее ограничений.*

Теорема 24 (2-ая теорема двойственности). *Допустимые решения $x = \{x_1, \ldots, x_n\}$ и $y = \{y_1, \ldots, y_m\}$ прямой и двойственной задачи соответственно оптимальны тогда и только тогда, когда*

$$x_j(c_j - \sum_{i=1}^{m} a_{ij}y_i) = 0 \;\; \textit{для всех } j, \tag{7.1}$$

$$y_i(\sum_{j=1}^{n} a_{ij}x_j - b_i) = 0 \;\; \textit{для всех } i. \tag{7.2}$$

Из 1-ой теоремы двойственности непосредственно вытекает, что стоимость любого допустимого решения двойственной задачи является нижней оценкой стоимости оптимального решения прямой задачи. Условия (7.1) и (7.2) называют прямым и двойственным условиями дополняющей нежесткости. Эти условия часто используются для доказательства того, что решение, полученное алгоритмом, является точным. Для построения приближенных алгоритмов удобнее использовать их в ослабленной форме. Пусть $x = \{x_1, \ldots, x_n\}$ и $y = \{y_1, \ldots, y_m\}$ — допустимые решения прямой и двойственной задачи соответственно, и заданы два параметра $\alpha \geq 1$ и $\beta \geq 1$.

α-Прямое условие дополняющей нежесткости:

$$x_j = 0 \text{ или } \frac{c_j}{\alpha} \leq \sum_{i=1}^{m} a_{ij}y_i \leq c_j \;\; \text{для всех } j.$$

β-Двойственное условие дополняющей нежесткости:

$$y_i = 0 \text{ или } b_i \leq \sum_{j=1}^{n} a_{ij}x_j \leq \beta b_i \text{ для всех } i.$$

Предложение 1. *Если x и y — прямое и двойственное допустимые решения, удовлетворяющие ослабленным условиям дополняющей нежесткости, то*

$$\sum_{j=1}^{n} c_j x_j \leq \alpha \cdot \beta \sum_{i=1}^{m} b_i y_i.$$

Доказательство. Из условий дополняющей нежесткости следует

$$\sum_{j=1}^{n} c_j x_j \leq \alpha \sum_{j=1}^{n} (\sum_{i=1}^{m} a_{ij} y_i) x_j = \alpha \sum_{i=1}^{m} (\sum_{j=1}^{n} a_{ij} x_j) y_i \leq \alpha\beta \sum_{i=1}^{m} b_i y_i.$$

□

7.2 Основные методы построения приближенных алгоритмов с использованием линейного программирования

Многие задачи комбинаторной оптимизации могут быть сформулированы, как задачи целочисленного линейного программирования (задача ЦЛП). При ослаблении ограничений целочисленности задача ЦЛП трансформируется в задачу линейного программирования. Полученную задачу ЛП принято называть ЛП-релаксацией исходной задачи. Решение ЛП-релаксации является естественной нижней оценкой стоимости оптимального решения задачи комбинаторной оптимизации. Отметим, что одна и та же задача комбинаторной оптимизации может быть описана разными задачами ЦЛП, линейные программы которых могут давать различные нижние оценки, что, как мы увидим впоследствии, влияет на качество приближенных алгоритмов.

Как упоминалось во введении, хорошая нижняя оценка зачастую помогает построить хорошее приближенное решение. Существует две основные техники построения приближенных алгоритмов, использующих

линейное программирование: ЛП-округление и прямо-двойственная схема. Первый метод, наиболее естественный, заключается в том, чтобы точно решить соответствующую задачу ЛП и затем аккуратно преобразовать полученное дробное решение в целочисленное. Оценка точности такого алгоритма основана на сравнении стоимостей дробного и целочисленного решений.

Второй, возможно, более сложный метод использует идею двойственности. В нем итеративно вычисляются целочисленное решение прямой задачи и допустимое дробное решение двойственной задачи. Любое допустимое решение двойственной задачи является нижней оценкой на оптимум прямой. Качество алгоритма оценивается сравнением этих двух решений. Оба подхода будут рассмотрены при построении приближенных алгоритмов для задачи о покрытии в разд. 9.

На первый взгляд может показаться, что первый метод всегда предпочтительнее второго. Действительно, оптимальное решение прямой задачи всегда дает более точную нижнюю оценку, чем допустимое решение двойственной задачи. Тем не менее, как правило, оба метода дают одинаковые по точности результаты. Более того, оценка точности алгоритмов, построенных при помощи как первой, так и второй техники, зависит от разрыва целочисленности (разрыва двойственности) линейной релаксации рассматриваемой задачи.

Рассмотрим ЛП-релаксацию некоторой задачи минимизации П. Пусть $OPT_f(I)$ — стоимость оптимального дробного решения примера I. **Разрывом целочисленности** называется следующая величина:

$$\sup_I \frac{OPT(I)}{OPT_f(I)}.$$

Напомним, что для оценки точности алгоритма стоимость полученного им решения сравнивается с нижней оценкой оптимального решения, то есть со стоимостью оптимального решения задачи ЛП или допустимого двойственного решения. Следовательно, наилучшая оценка точности, которую можно получить, используя линейное программирование, равна разрыву целочисленности в используемой задачи ЛП. Для многих известных труднорешаемых задач известны алгоритмы, основанные на двух подходах, которые достигают такой точности.

Основное отличие двух техник определяется не точностью получае-

мых алгоритмов, а их трудоемкостью. В алгоритмах, основанных на ЛП-округлении, сначала требуется найти точное решение задачи ЛП. Хотя, как было отмечено выше, эта задача хорошо разрешима как с теоретической, так и с практической стороны, она все-таки требует значительно больших вычислительных усилий, чем чисто комбинаторные алгоритмы, которые можно использовать при реализации прямо-двойственной схемы.

7.3 Упражнения

1. Рассмотрим следующую задачу линейного программирования, которую назовем P:

$$x_1 + x_3 \to min,$$
$$x_1 + 2x_2 \leq 5,$$
$$x_2 + 2x_3 = 6,$$
$$x_1, x_2, x_3 \geq 0.$$

   Запишите задачу D, двойственную к P. Запишите условия дополняющей нежесткости для этой задачи и используйте их для решения двойственной задачи D.

2. Сформулируйте следующие задачи как задачи ЦЛП:

   (a) задача о покрытии;

   (b) задача Штейнера;

   (c) задача коммивояжера.

   Ослабьте в них условие целочисленности и запишите двойственные к ним задачи.

3. Истинны или ложны следующие утверждения:

   (a) если задача линейного программирования недопустима, то двойственная к ней задача должна быть неограниченной;

   (b) если задача линейного программирования неограничена, то двойственная к ней задача должна быть недопустимой?

4. Покажите, что множество оптимальных точек индивидуальной задачи ЛП является выпуклым множеством.

8 *Минимизация длины расписания на различных машинах*

Штирлиц никогда не торопил события. Выдержка, считал он, оборотная сторона стремительности. Все определяется пропорциями: искусство, разведка, любовь, политика.

из кинофильма «Семнадцать мгновений весны»

В главах 4 и 6 изучались задачи построения минимального по длине расписания на параллельных идентичных машинах. В этой главе рассмотрим задачу, в которой длительность каждой работы зависит от того, на какую машину она назначена.

Задача $R||C_{\max}$

Дано: множество работ $J = \{1, \ldots, n\}$ и множество параллельных машин $M = \{M_1, \ldots, M_m\}$, каждая работа должна быть выполнена на одной из этих машин, $p_{ij} \geq 0$ — длительность выполнения работы j на машине M_i. Прерывания запрещены, каждая машина обслуживает не более одной работы одновременно.

Найти расписание, имеющее минимальную длину $C_{\max}$.

8.1 ЦЛП-формулировка и параметрическое сокращение

Сформулируем задачу $R||C_{\max}$ как задачу ЦЛП. Введем переменные x_{ij}, такие, что $x_{ij} = 1$, если работа j выполняется на машине M_i, и $x_{ij} = 0$ в противном случае. Требуется минимизировать длину расписания C_{max}.

$$C_{\max} \to \min$$

$$\sum_{M_i \in M} x_{ij} = 1, \;\; j = 1, \ldots, n,$$

$$\sum_{j \in J} x_{ij} p_{ij} \leq C_{max}, \quad i = 1, \ldots, m,$$

$$x_{ij} \in \{0, 1\}, \;\; i = 1, \ldots, m, \;\; j = 1, \ldots, n.$$

Первое ограничение означает, что каждая из работ должна быть назначена на некоторую машину, второе ограничение говорит о том, что длина расписания определяется временем работы наиболее загруженной машины.

Покажем, что ЦЛП($R||C_{\max}$) имеет неограниченный разрыв целочисленности. Для этого рассмотрим пример, в котором есть одна работа и m машин. Длительность выполнения этой работы на любой машине равна m. Стоимость оптимального решения в таком примере также равна m, а в оптимальном решении линейной релаксации работа будет поровну разделена между всеми машинами, и длина дробного расписания будет равна 1.

Возникает вопрос: почему в рассмотренном примере стоимость решения задачи ЛП во много раз меньше, чем стоимость решения задачи ЦЛП? Дело в том, что ЦЛП автоматически полагает $x_{ij} = 0$, если $p_{ij} > C_{\max}$, а ЛП распределяет по $\frac{1}{m}$-й части работы на каждую из машин. Этого можно было бы избежать, если добавить в ЛП ограничение:

$$\text{для любых } M_i \in M, \; j \in J, \;\; \text{если } \; p_{ij} > t, \;\; \text{то } x_{ij} = 0.$$

К сожалению, это нелинейное ограничение не может быть использовано в ЛП.

Чтобы обойти эту трудность, используем технику параметрического сокращения. Предположим, что $T \in \mathbf{Z}^+$ — верхняя оценка длины расписания. Удалим все пары работа – машина, такие, что $p_{ij} > T$. То есть будем считать, что в этом случае работа j не может выполняться на машине M_i. Определим $S_T = \{(i,j)|p_{ij} \leq T\}$. Для каждого значения параметра $T \in \mathbf{Z}^+$ рассмотрим линейную программу $LP(T)$. В линейной программе $LP(T)$ присутствуют только переменные x_{ij}, у которых $(i,j) \in S_T$. Требуется проверить, существует ли дробное расписание, длина которого не превосходит T.

$$\sum_{i:(i,j)\in S_T} x_{ij} = 1, \quad j \in J,$$

$$\sum_{j:(i,j)\in S_T} x_{ij} p_{ij} \leq T, \quad M_i \in M,$$

$$x_{ij} \geq 0, \quad (i,j) \in S_T.$$

8.2 Свойства экстремальных решений

Используя бинарный поиск, можно найти наименьшее значение T, при котором $LP(T)$ имеет допустимое решение. Пусть T^* и есть такое значение. Очевидно, что T^* является нижней оценкой. Если существует допустимое решение в $LP(T)$, то в процессе проверки будет найдено допустимое решение, соответствующее крайней точке выпуклого многогранника, образованного ограничениями задачи $LP(T)$. Назовем такие решения экстремальными. Для того, чтобы оценить, насколько хороша нижняя оценка, и построить хороший приближенный алгоритм, определим, какими полезными свойствами обладают экстремальные решения.

Лемма 16. *Любое допустимое решение $LP(T)$, являющееся крайней точкой, имеет не более $n+m$ ненулевых переменных.*

Доказательство. Пусть $r = |S_T|$ — число переменных в $LP(T)$. Допустимое решение является крайней точкой тогда и только тогда, когда r линейно независимых ограничений обращаются в равенство. По крайней мере, $r-(m+n)$ из них будут выбраны из третьего множества ограничений (т. е. $x_{ij} \geq 0$). Значит, $r-(n+m)$ соответствующих переменных

равны 0. Получим, что любое экстремальное решение $LP(T)$ имеет не более $n + m$ ненулевых переменных. □

Пусть $\boldsymbol{x}$ — экстремальное решение $LP(T)$. Будем называть работу в решении $\boldsymbol{x}$ *целой*, если она назначена ровно на одну машину. В противном случае, будем называть работу *дробной*.

Следствие 3. *В любом экстремальном решении $LP(T)$ есть, по крайней мере, $n - m$ целых работ.*

Доказательство. Пусть $\boldsymbol{x}$ — экстремальное решение $LP(T)$, α — число целых работ, β — число дробных работ. Каждая дробная работа назначается, по крайней мере, на 2 машины, а значит, дает не менее двух ненулевых входов в $\boldsymbol{x}$. Имеем

$$\alpha + 2\beta \leq n + m.$$

Отсюда следует, что $\beta \leq m$ и $\alpha \geq n - m$. □

Другие свойства экстремального решения $LP(T)$ основаны на представлении этого решения двудольным графом.

8.3 Представление экстремального решения двудольным графом

Пусть $\boldsymbol{x}$ — экстремальное решение $LP(T)$. Определим двудольный граф $G = (M, J, E)$ на множестве вершин $M \cup J$, в котором существует ребро $(M_i, j) \in E$ тогда и только тогда, когда $x_{ij} \neq 0$. Пусть $F \subset J$ является множеством дробных работ решения $\boldsymbol{x}$. Обозначим через H подграф G, индуцированный на множество вершин $M \cup F$. Заметим, что ребро (M_i, j) принадлежит графу H тогда и только тогда, когда $0 < x_{ij} < 1$.

Связный граф на множестве вершин V называется *псевдодеревом*, если он имеет не более $|V|$ ребер.

Граф называется *псевдолесом*, если каждая его связная компонента является псевдодеревом.

Лемма 17. *Граф G — псевдолес.*

Доказательство. Покажем, что число ребер в каждой связной компоненте графа G ограничено числом вершин в ней.

Рассмотрим в графе G связную компоненту G_C. Пусть $M_C \subseteq M$ — подмножество машин и $J_C \subseteq J$ — подмножество работ, попавших в G_C. Определим $S_{T,C} = \{(M_i, j) \in S_T | M_i \in M_C, j \in J_C\}$. Рассмотрим задачу линейного программирования $LP_C(T)$ с множеством переменных и ограничений, связанных с работами и машинами из G_C.

$$\sum_{i:(M_i,j)\in S_{T,C}} x_{ij} = 1, \;\; j \in J_C,$$

$$\sum_{j:(M_i,j)\in S_{T,C}} x_{ij} p_{ij} \leq T, \quad M_i \in M_C,$$

$$x_{ij} \geq 0, \quad (M_i, j) \in S_{T,C}.$$

Выберем точку $\boldsymbol{y}$ с координатами из множества $S_{T,C}$, такую, что $\boldsymbol{y}_{i,j} = \boldsymbol{x}_{i,j}$ для всех $(M_i, j) \in S_{T,C}$. Тогда $\boldsymbol{y}$ — экстремальная точка в $LP_C(T)$. Применяя лемму 16, получим, что число ребер не превосходит $n_C + m_C$, следовательно, G_C — псевдодерево, а G — псевдолес.

□

Паросочетание в H назовем *совершенным*, если оно покрывает каждую работу $J_i \in F$.

Лемма 18. *Граф H имеет совершенное паросочетание.*

Доказательство. Удаляя из графа G все вершины, соответствующие целым работам, и инцидентные им ребра, получим граф H. Поскольку удалено одинаковое количество вершин и ребер, то граф H — псевдолес. Все листья (если они есть) в H соответствуют машинам. Выберем произвольный лист (машину) и добавим инцидентное ему ребро в паросочетание. Удалим обе вершины (машину и работу) этого ребра из H. Повторим эту процедуру, пока не получим либо набор четных циклов, либо пустой граф. Из каждого цикла добавим набор чередующихся ребер в паросочетание. В результате получим совершенное паросочетание, покрывающее все работы. □

Для описания алгоритма осталось определить границы бинарного поиска. Назначим каждую работу на ту машину, на которой она имеет наименьшее время выполнения. Пусть α — длина полученного расписания, тогда

$$\frac{\alpha}{m} \leq C_{\max}(\sigma^*) \leq \alpha.$$

Для нахождения T^* устроим бинарный поиск в интервале $[\frac{\alpha}{m}, \alpha]$, используя задачу $LP(T)$ для проверки существования дробных расписаний длины не меньше T.

Алгоритм Ленстра – Шмойс – Тардош
$(J = \{1, \ldots, n\}, M = \{M_1, \ldots, M_m\}, p : J \times M \to \mathbf{Q}^+)$

1 Используя бинарный поиск в интервале $[\frac{\alpha}{m}, \alpha]$, найти наименьшее допустимое значение $T \in \mathbf{Z}^+$, для которого $LP(T)$ имеет допустимое решение. Обозначить его T^*.
2 Найти $\boldsymbol{x}$ — экстремальное решение $LP(T)$.
3 Назначить все целые работы на те же машины, что и в $\boldsymbol{x}$.
4 Построить граф H и найти в нем совершенное паросочетание.
5 Назначить дробные работы на машины (согласно этому паросочетанию). Обозначить полученное расписание σ.
6 **return** σ

Теорема 25. Алгоритм Ленстра – Шмойс – Тардош *является 2-приближенным алгоритмом для задачи* $R||C_{\max}$.

Доказательство. Согласно шагу 1 алгоритма имеем, что $T^* \leq OPT$. Длина расписания на множестве целых работ не превосходит T^*. При распределении дробных работ, согласно совершенному паросочетанию, на каждую машину добавляется не больше одной работы. Так как $p_{ij} \leq T^*$, то общая длина полученного расписания не превосходит $2T^* \leq 2 \cdot OPT$. □

Покажем, что эта оценка достигается. Рассмотрим пример $\hat{I}_m$, в котором задано m параллельных машин и $n = m^2 - m + 1$ работ. Длительность первой работы равна m, а длительность остальных работ равна 1. В оптимальном расписании первая работа назначена на одну из машин, а остальные работы поровну распределены по другим машинам.

Длина такого расписания равна m. Легко заметить, что $LP(T)$ не имеет допустимого решения для $T < m$. Предположим, что для $T = m$ выбрано следующее экстремальное решение $\hat{x}$. Одинаковые части первой работы размера 1 назначены на каждую машину. Кроме того, на каждой машине целиком выполняется $m-1$ работа единичной длительности. После округления решения $\hat{x}$ получим расписание длины $2m-1$.

8.4 Упражнения

1. С помощью алгоритма Ленстра – Шмойс – Тардош решить задачу с 15 работами и тремя машинами. Длительности работ заданы в таблице:

	J_1	J_2	J_3	J_4	J_5	J_6	J_7	J_8	J_9	J_{10}	J_{11}	J_{12}	J_{13}	J_{14}	J_{15}
M_1	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
M_2	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5
M_3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1

2. Доказать, что решение $\hat{x}$, описанное в примере, является экстремальным решением задачи $LP(T)$ при $t = m$.

3. Рассмотрим следующее решение x задачи $LP(m)$ для примера $\hat{I}_m$. Первая работа поровну поделена между всеми машинами, то есть $x_{i1} = \frac{1}{m}$ для всех i. Вторая и третья работа выполняются на машинах 1 и 2 так, что $x_{12} = 1/2$, $x_{22} = 1/2$, $x_{13} = 1/2$, и $x_{23} = 1/2$. Все остальные работы целиком выполняются на какой-либо из машин, и длина расписания равна m. Докажите, что решение x не является экстремальной точкой.

4. Найти оценку точности алгоритма Ленстра – Шмойс – Тардош для задачи, в которой все машины являются идентичными.

5. Доказать, что точка $\boldsymbol{y}$, выбранная в доказательстве леммы 17, является экстремальной точкой в $LP_C(T)$.

9 *Приближенные алгоритмы для задачи о покрытии множествами*

— А ты умеешь считать?
— Нет.
— Отлично... два сольдо плюс два сольдо — будет десять сольдо. Десять сольдо плюс десять сольдо — будет сто сольдо. Плати один золотой.

диалог хозяина харчевни и
Буратино из кинофильма
«Приключения Буратино»

Вернемся к задаче о покрытии множествами, которая была рассмотрена в главе 1. Напомним, что Алгоритм Хватала находит в худшем случае H_n-приближенное решение. Достаточно удивительно, но этот наивный алгоритм дает результат, который вряд ли может быть улучшен. В частности, Фейдж доказал, что для любой константы ε существование $(1-\varepsilon)\ln n$-приближенного алгоритма влечет существование алгоритма трудоемкости $O(n^{O(\log\log n)})$ для всех задач в классе NP. Эта гипотеза представляется такой же неправдоподобной, как и гипотеза о совпадении классов P и NP. Таким образом, для любой константы ρ для задачи о покрытии, скорее всего, не существует ρ-приближенного алгоритма.

Пусть f_i — кратность элемента e_i, то есть число множеств из $\mathcal{S}$,

в которые он входит. Положим $f = \max_{i=1,\ldots,n} f_i$ и построим хорошие приближенные алгоритмы для примеров задачи о покрытии, в которых кратность элементов ограничена.

9.1 ЛП-округление для задачи о покрытии множествами

Сформулируем задачу о покрытии в виде задачи целочисленного линейного программирования MIP_{cov}. Пусть $x_S = 1$, если подмножество S входит в покрытие, и $x_S = 0$ в противном случае.

$$\sum_{S \in \mathcal{S}} c(S) x_S \to \min$$
$$\sum_{S | e \in S} x_S \geq 1, \;\; e \in U,$$
$$x_S \in \{0, 1\} \;\; S \in \mathcal{S}.$$

Перейдем к линейной релаксации LP_{cov}, для этого все переменные сделаем непрерывными. Получим:

$$\sum_{S \in \mathcal{S}} c(S) x_S \to \min$$
$$\sum_{S | e \in S} x_S \geq 1, \;\; e \in U,$$
$$x_S \geq 0 \;\; S \in \mathcal{S}.$$

Оптимальное решение задачи LP_{cov} является нижней оценкой на величину оптимума исходной задачи MIP_{cov}.

Алгоритм Округление ЛП-релаксации $(U, \mathcal{S}, c : \mathcal{S} \to \mathbf{Q}^+)$

1 Найти оптимальное решение задачи LP_{cov}.
2 $Sol \leftarrow \emptyset$.
3 **for** $S \in \mathcal{S}$
  do if $x_S \geq 1/f$
    then $Sol := Sol \cup \{S\}$.
4 **return** Sol

Теорема 26. Алгоритм Округление ЛП-релаксации *является f-приближенным алгоритмом для задачи о покрытии.*

Доказательство. Рассмотрим некоторый элемент $e \in U$. Ограничения в линейной программе гарантируют, что существует такое множество S, что $e \in S$ и $x_S \geq 1/f$. Согласно шагу 3 Алгоритма Округление ЛП-релаксации, оно войдет в покрытие. Следовательно, полученное решение является допустимым.

Так как $x_S \geq 1/f$ для всех выбранных S, то стоимость найденного покрытия превышает стоимость решения линейной релаксации не больше чем в f раз. □

Следствие 4. Алгоритм Округление ЛП-релаксации *является 2-приближенным алгоритмом для задачи о вершинном покрытии.*

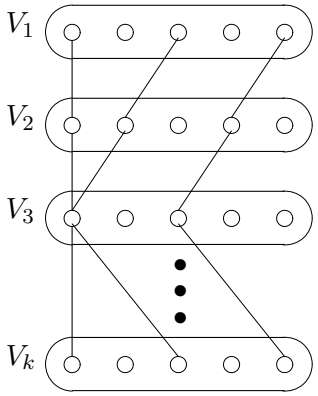


Рис. 9.1. Точность алгоритма

Покажем, что эта оценка достигается. Рассмотрим полный k-дольный гиперграф, изображенный на рис. 9.1. Пусть дано k непересекающихся множеств $V_1, \ldots, V_k$. Каждое множество содержит n вершин. Гиперграф состоит из множества вершин $V = V_1 \cup \ldots \cup V_k$ и n^k различных гиперребер: каждое гиперребро соединяет по одной вершине из каждого V_i.

В задаче о покрытии элементы соответствуют гиперребрам, и множества соответствуют вершинам. Элемент принадлежит множеству, если гиперребро, соответствующее элементу, проходит через вершину, соответствующую множеству. Кратность каждого элемента равна k. Каждое множество имеет стоимость 1. Полагая значение каждой переменной x_S равным $1/k$, получим оптимальное дробное покрытие веса n. Округляя это решение, алгоритм построит покрытие, состоящее из всех nk множеств. С другой стороны, очевидно, что набор множеств, соответствующих вершинам в V_1, задает оптимальное покрытие, и его вес равен n.

9.2 Полуцелочисленность ЛП-релаксации задачи о вершинном покрытии

Рассмотрим задачу о вершинном покрытии, введенную в параграфе 1.2, которая является частным случаем задачи о покрытии множествами. Пусть в графе $G = (V, E)$ заданы неотрицательные веса вершин $w : V \to \mathbf{Q}^+$. Тогда для задачи о вершинном покрытии можно переписать задачу MIP_{cov} как

$$\sum_{v \in V} w(v) x_v \to \min$$

$$x_u + x_v \geq 1, \;\; (u, v) \in E,$$

$$x_v \in \{0, 1\} \;\; v \in V.$$

При этом ее линейная релаксация $LP_{vert-cov}$ примет вид:

$$\sum_{v \in V} w(v) x_v \to \min$$

$$x_u + x_v \geq 1, \;\; (u, v) \in E,$$

$$x_v \geq 0 \;\; v \in V.$$

Допустимое решение задачи линейного программирования называется полуцелочисленным, если значение каждой ее переменной равно либо 0, либо 1, либо $\frac{1}{2}$.

Лемма 19. *Пусть x — допустимое решение задачи $LP_{vert-cov}$, которое не является полуцелочисленным. Тогда x не является крайней точкой множества решений, описываемого неравенствами задачи $LP_{vert-cov}$.*

Доказательство. Рассмотрим произвольное допустимое решение x задачи $LP_{vert-cov}$, которое не является полуцелочисленным. Покажем, что x может быть представлено как выпуклая комбинация двух отличных от него допустимых решений задачи $LP_{vert-cov}$. Рассмотрим множество вершин, на котором x имеет значения, отличные от 0, 1 и $\frac{1}{2}$. Разобьем это множество на два:

$$V_+ = \left\{ v \middle| \frac{1}{2} < x_v < 1 \right\} \text{ и } V_- = \left\{ v \middle| 0 < x_v < \frac{1}{2} \right\}.$$

Для произвольного $\varepsilon > 0$ определим два решения

$$y_v = \begin{cases} x_v + \varepsilon, & x_v \in V_+ \\ x_v - \varepsilon, & x_v \in V_- \\ x_v, & \text{иначе} \end{cases}, \quad z_v = \begin{cases} x_v - \varepsilon, & x_v \in V_+ \\ x_v + \varepsilon, & x_v \in V_- \\ x_v, & \text{иначе.} \end{cases}$$

По предположению $V_+ \bigcup V_- \neq \emptyset$. Следовательно, x отличен от y и z, и при этом $x = \frac{1}{2}(y + z)$. Покажем, что можно выбрать такое $\varepsilon > 0$, что решения y и z являются допустимыми. Пусть $\varepsilon_1 = \min_{v \in V} \frac{x_v}{2}$. Тогда для любого положительного $\varepsilon \leq \varepsilon_1$ все координаты векторов y и z неотрицательны. Пусть $E' \subseteq E$ обозначает подмножество ребер $(u, v) \in E'$, для которых выполнено $x_u + x_v > 1$. Рассмотрим $\varepsilon_2 = \min_{(u,v) \in E'} \frac{x_u + x_v - 1}{2}$. Заметим, что эта величина строго больше 0. Ясно, что при выборе любого положительного $\varepsilon \leq \varepsilon_2$ получим $y_u + y_v \geq 1$ и $z_u + z_v \geq 1$ для всех $(u, v) \in E'$. Осталось проверить, что эти неравенства выполняются для всех ребер $(u, v) \in E \setminus E'$. Для каждого такого ребра либо $x_u = x_v = \frac{1}{2}$, либо $x_u = 0$, $x_v = 1$, либо $x_u \in V_+$ и $x_v \in V_-$. В каждом из трех случаев для любого выбора ϵ получим

$$x_u + x_v = y_u + y_v = z_u + z_v = 1.$$

Следовательно, при $\varepsilon = \min\{\varepsilon_1, \varepsilon_2\}$ решения y и z являются допустимыми. □

Непосредственно из леммы 19 вытекает следующий результат.

Теорема 27. *Любое экстремальное решение задачи $LP_{vert-cov}$ является полуцелочисленным.*

Из установленного свойства легко построить 2-приближенный алгоритм для задачи о вершинном покрытии, аналогичный Алгоритму Округление ЛП-релаксации: найти экстремальную точку задачи $LP_{vert-cov}$ и взять все вершины, для которых значение соответственной переменной равно $\frac{1}{2}$ или 1.

9.3 Прямо-двойственная схема для задачи о покрытии множествами

Рассмотрим двойственную задачу к LP_{cov}. Введем переменные y_e, соответствующие элементам $e \in U$, и получим задачу:

$$\sum_{e \in U} y_e \to \max$$

$$\sum_{e:e \in S} y_e \leq c(S), \;\; S \in \mathcal{S},$$

$$y_e \geq 0 \;\; e \in U.$$

Для построения f-приближенного алгоритма зададим параметры $\alpha = 1$ и $\beta = f$. Ослабленные условия дополняющей нежесткости примут вид:
1-Прямое условие

$$\forall S \in \mathcal{S} : x_S \neq 0 \Rightarrow \sum_{e:e \in S} y_e = c(S).$$

Множество S будем называть *строгим*, если $\sum_{e:e \in S} y_e = c(S)$.
f-Двойственное условие

$$\forall e : y_e \neq 0 \Rightarrow \sum_{S:e \in S} x_S \leq f.$$

Прямо-двойственный алгоритм $(U, \Omega, c : \Omega \to \mathbf{Q}^+)$

1 $x \leftarrow 0,\ y \leftarrow 0.$
2 **while** покрыты не все элементы
do Выбрать непокрытый элемент, например, e.
Увеличить y_e до тех пор, пока какое-либо множество
не станет строгим. При этом решение двойственной задачи
должно остаться допустимым.
Поместить все строгие множества в покрытие и обновить x.
Объявить все элементы, появляющиеся в этих множествах,
«покрытыми».
3 **return** x

Алгоритм стартует с нулевого решения $x = 0$ и $y = 0$. Нулевое решение является допустимым в двойственной задаче и недопустимым в прямой. При этом нулевое решение очевидно удовлетворяет ослабленным условиям дополняющей нежесткости. На каждой итерации алгоритм улучшает допустимость прямого решения и оптимальность двойственного, не нарушая при этом ослабленные условия дополняющей нежесткости. В ходе работы алгоритма все переменные в прямой задаче равны либо 0, либо 1. Тогда из 1-прямого условия следует, что все множества, выбранные в покрытие, являются строгими. В свою очередь f-двойственное условие эквивалентно условию, что *каждый элемент, имеющий ненулевое значение двойственной переменной, может быть покрыт не более чем f раз*. Поскольку каждый элемент содержится не более чем в f множествах, то это условие выполняется для всех элементов автоматически.

Теорема 28. Прямо-двойственный алгоритм *является f-приближенным алгоритмом для задачи о покрытии.*

Доказательство. Заметим, что алгоритм останавливается, только когда все элементы покрыты. Следовательно, решение, полученное алгоритмом, является допустимым. В покрытие выбираются только строгие множества, и значения y_e элементов, которые в них входят, больше не изменяются. Следовательно, 1-прямое условие выполняется на каждой итерации алгоритма. Напомним, что выполнение f-двойственного условия сле-

дует из определения параметра f. Таким образом, алгоритм находит допустимое решение и, по предложению 1, является f-приближенным. $\square$

Покажем, что оценка, полученная в теореме 28, является точной. Пусть $\mathcal{S}$ состоит из $n-1$ множеств веса 1, $\{e_1, e_n\}, \ldots, \{e_{n-1}, e_n\}$, и одного множества веса $1+\varepsilon$, $\{e_1, \ldots, e_{n+1}\}$, где $\varepsilon > 0$ (рис. 9.2). Поскольку e_n принадлежит всем n множествам, то $f = n$.

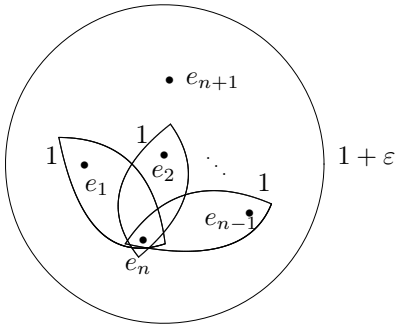


Рис. 9.2. Точность Прямо-двойственного алгоритма

Предположим, что на первой итерации алгоритм выбрал элемент e_n. Тогда переменная y_{e_n} примет значение 1, и все множества $\{e_i, e_n\}, i = 1, \ldots, n-1$ станут строгими. Следовательно, все они попадут в покрытие и покроют элементы $e_1, \ldots, e_n$.

На второй итерации будет выбран оставшийся элемент e_{n+1}. $y_{e_{n+1}}$ увеличится на ε, и множество $\{e_1, \ldots, e_{n+1}\}$ станет строгим. В результате, в покрытие попадут все множества, и его вес составит $n+\varepsilon$, в то время как вес оптимального покрытия равен $1+\varepsilon$.

9.4 Упражнения

1. Изменим шаг 3 в Алгоритме Округление ЛП-релаксации. Пусть в покрытие войдут все множества S с ненулевым значением переменных x_S в ЛП-релаксации. Показать, что и в этом случае

алгоритм является f-приближенным алгоритмом для задачи о покрытии.

2. Для графа, изображенного на рис. 9.3, решить задачу о вершинном покрытии Алгоритмом Округление ЛП-релаксации и Прямо-двойственным алгоритмом.

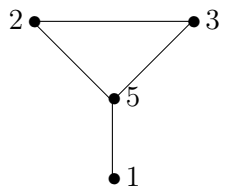


Рис. 9.3. Граф

3. Показать, что рассмотренный в главе 1 для задачи о вершинном покрытии Алгоритм Уровневый также является прямо-двойственным алгоритмом. Как будут выглядеть ослабленные условия дополняющей нежесткости в этом случае?

4. Доказать или опровергнуть следующее утверждение. Рассмотрим пример задачи о покрытии множествами, в котором каждый элемент входит ровно в три множества. Тогда существует оптимальное решение линейной релаксации LP_{cov}, в котором значения всех переменных кратны величине $\frac{1}{3}$.

10 *Вероятностные алгоритмы для задачи о максимальной выполнимости*

— Это случайность!
— Случайности не случайны!

диалог мастера Шифу и мастера Угвэй из мультфильма «Кунг-фу панда»

В 1971 году С. Кук доказал, что задача о выполнимости является полной в классе задач NP [5], положив начало теории NP-полноты. В этой главе рассмотрим оптимизационную версию задачи о выполнимости.

Задача о максимальной выполнимости

Дано: набор f дизъюнкций на булевых переменных $x_1, \ldots, x_n$ и веса дизъюнкций $w : f \to \mathbf{Q}^+$.

Найти назначение истинности для булевых переменных, максимизирующее общий вес выполненных дизъюнкций.

Дизъюнкция c *выполнена*, если она принимает значение «истина». Напомним, что каждая дизъюнкция состоит из набора переменных или их отрицаний, называемых литералами. Все переменные и литералы принимают значение 0 или 1 (*истина* или *ложь*). Для каждой переменной x_i выполнено $x_i + \overline{x}_i = 1$. Дизъюнкция c выполнена, если значение хотя бы

одного из ее литералов равно 1. Обозначим через $size(c)$ число литералов в дизъюнкции c.

Для задачи о максимальной выполнимости рассмотрим три вероятностных алгоритма. Идея вероятностного алгоритма состоит из задания функции распределения для каждой переменной и в случайном выборе ее значений согласно функции распределения. Выбор функции распределения, как правило, основывается либо на дробном решении линейной релаксации, либо из учета комбинаторных свойств задачи.

Примером последнего может служить представленный в следующем разделе простой алгоритм, предложенный Джонсоном.

10.1 Алгоритм Джонсона и дерандомизация

Алгоритм Джонсона с равной вероятностью назначает каждой переменной либо значение 1, либо значение 0.

Вероятностный алгоритм Джонсона
$(x_1, \ldots, x_n, f, w : f \to \mathbf{Q}^+)$

1 Независимо для каждого i:
  $x_i \leftarrow 1$ с вероятностью 0,5
  и $x_i \leftarrow 0$ иначе.
  Назовем полученное назначение τ.
2 **return** τ.

Пусть случайная величина W обозначает общий вес выполненных дизъюнкций. Для каждой дизъюнкции $c \in f$ введем случайную величину W_c, которая обозначает вес, вносимый дизъюнкцией c в W.

$$W = \sum_{c \in f} W_c,$$

$$\mathbf{E}[W_c] = w_c \cdot \mathbf{Pr}[c = 1]$$

Для $k \geq 1$ обозначим $\alpha_k = 1 - 2^{-k}$.

Лемма 20. *Если $size(c) = k$, то $E[W_c] = \alpha_k w_c$.*

Доказательство. Дизъюнкция c не учитывается в τ тогда и только тогда, когда значения всех литералов равны 0. Вероятность этого события 2^{-k}. □

Теорема 29. $E[W] \geq \frac{1}{2}OPT$.

Доказательство. Поскольку для $k \geq 1$ верно, что $\alpha_k \geq 1/2$, то

$$E[W] = \sum_{c \in f} E[W_c] \geq \frac{1}{2} \sum_{c \in f} w_c \geq \frac{1}{2} OPT.$$

□

Итак, если выбирать значение каждой переменной случайно с равной вероятностью, то математическое ожидание веса выполненных дизъюнкций не меньше, чем $\frac{OPT}{2}$. Повторяя эту процедуру многократно, можно добиться выполнения этого свойства с большой вероятностью. На самом деле, можно показать, что существует детерминированная процедура, строящая решение веса не меньше, чем $\frac{OPT}{2}$. Такую процедуру принято называть дерандомизацией вероятностного алгоритма.

Зафиксируем назначение истинности $a_1, \ldots, a_i$ для переменных $x_1, \ldots, x_i$ и рассмотрим условное математическое ожидание $E[W | x_1 = a_1, \ldots, x_i = a_i]$. Если $i = n$, то условное математическое ожидание равно весу дизъюнкции, соответствующей этому назначению истинности.

Лемма 21. *Условное математическое ожидание $E[W | x_1 = a_1, \ldots, x_i = a_i]$ может быть вычислено за время, ограниченное полиномом от размера входа.*

Доказательство. Зафиксируем значения переменных $x_1 = a_1, \ldots, x_i = a_i$. Удалим из всех дизъюнкций литералы, которые приняли значение «ложно». Если значение хотя бы одного литерала в ϕ с фиксированной переменной получило значение «истина», то добавим вес дизъюнкции ϕ к W и удалим ее из набора f. Получим новый набор дизъюнкций на переменных $x_{i+1}, \ldots, x_n$, математическое ожидание общего веса выполненных дизъюнкций которого можно легко вычислить за время, ограниченное полиномом от числа литералов и дизъюнкций.

□

Теорема 30. *Существует такое назначение истинности* $x_1 = a_1, \ldots, x_n = a_n$, *что* $W(a_1, \ldots, a_n) \geq E[W]$, *и оно может быть вычислено за полиномиальное время.*

Доказательство. По теореме 29 имеем $E[W] \geq \frac{1}{2}OPT$. Докажем теорему, используя индукцию. Пусть $E[W|x_1 = a_1, \ldots, x_i = a_i] \geq \frac{1}{2}OPT$ для некоторого $i \geq 0$. Тогда

$$E[W|x_1 = a_1, \ldots, x_i = a_i] =$$

$$E[W|x_1 = a_1, \ldots, x_i = a_i, x_{i+1} = 1]/2$$

$$+E[W|x_1 = a_1, \ldots, x_i = a_i, x_{i+1} = 0]/2$$

Следовательно,
либо $E[W|x_1 = a_1, \ldots, x_i = a_i, x_{i+1} = 1] \geq E[W|x_1 = a_1, \ldots, x_i = a_i]$,
либо $E[W|x_1 = a_1, \ldots, x_i = a_i, x_{i+1} = 0] \geq E[W|x_1 = a_1, \ldots, x_i = a_i]$.

Нетрудно проверить, что искомое назначение можно найти за $O(n)$ шагов, на каждом шаге выбирая назначение с большим средним. Получим $E[W|x_1 = a_1, \ldots, x_{i+1} = a_{i+1}] \geq \frac{1}{2}OPT$. □

Отметим, что подобная техника может быть использована и в более общем случае, когда для всех переменных заданы различные распределения, и даже если значения переменных зависят друг от друга. Действительно,

$$E[W|x_1 = a_1, \ldots, x_i = a_i] =$$

$$E[W|x_1 = a_1, \ldots, x_i = a_i, x_{i+1} = 1] \cdot Pr[x_{i+1} = 1|x_1 = a_1, \ldots, x_i = a_i]$$

$$+E[W|x_1 = a_1, \ldots, x_i = a_i, x_{i+1} = 0] \cdot Pr[x_{i+1} = 0|x_1 = a_1, \ldots, x_i = a_i].$$

Так как переменная может принимать только два значения (1 или 0), то $Pr[x_{i+1} = 1|x_1 = a_1, \ldots, x_i = a_i] + Pr[x_{i+1} = 0|x_1 = a_1, \ldots, x_i = a_i] = 1$. Следовательно, в правой части выражения для $E[W|x_1 = a_1, \ldots, x_i = a_i]$ стоит выпуклая комбинация от $E[W|x_1 = a_1, \ldots, x_i = a_i, x_{i+1} = 1]$ и $E[W|x_1 = a_1, \ldots, x_i = a_i, x_{i+1} = 0]$ и либо
$E[W|x_1 = a_1, \ldots, x_i = a_i, x_{i+1} = 1] \geq E[W|x_1 = a_1, \ldots, x_i = a_i]$,
либо
$E[W|x_1 = a_1, \ldots, x_i = a_i, x_{i+1} = 0] \geq E[W|x_1 = a_1, \ldots, x_i = a_i]$.

Правда, если распределение задано слишком сложно, то могут возникнуть проблемы с леммой 21, то есть с быстрым вычислением условных математических ожиданий.

Пусть x_i — случайная величина, которая равна 1 с вероятностью p_i и 0 с вероятностью $1 - p_i$, и $W(x_1, \ldots, x_n)$ — произвольная целевая функция. Тогда следующий алгоритм находит решение со значением не меньше чем $E(W)$.

Алгоритм Наибольшее среднее $(p_1, \ldots, p_n)$

1 **for** $i = 1$ **to** n
do
2 Найти $E[W|x_1 = a_1, \ldots, x_{i-1} = a_{i-1}|x_i = 0]$
и $E[W|x_1 = a_1, \ldots, x_{i-1} = a_{i-1}|x_i = 1]$.
3 **if** $E[W|x_1 = a_1, \ldots, x_{i-1} = a_{i-1}|x_i = 0] \geq$
$E[W|x_1 = a_1, \ldots, x_{i-1} = a_{i-1}|x_i = 1]$,
then положить $a_i = 0$,
else положить $a_i = 1$.
4 **return** $x_1 = a_1, \ldots, x_n = a_n$.

Заметим, что решение, получаемое алгоритмом, зависит от значений величин $p_1, \ldots, p_n$.

10.2 Вероятностное ЛП-округление

Сформулируем задачу о максимальной выполнимости как задачу целочисленного линейного программирования IP_{sat}. Пусть переменная $z_c = 1$, если дизъюнкция c выполняется при назначении переменных $y_1, \ldots, y_n$. Через S_c^+ обозначим множество переменных, входящих в дизъюнкцию c без отрицания, а через S_c^- — подмножество переменных $y_1, \ldots, y_n$, входящих в дизъюнкцию c с отрицанием. Тогда задача о максимальной выполнимости может быть записана как

$$\sum_{c \in f} w_c z_c \to \max$$

$$\sum_{i \in S_c^+} y_i + \sum_{i \in S_c^-} (1 - y_i) \geq z_c, \quad c \in f,$$

$$z_c \in \{0,1\}, \;\; c \in f,$$
$$y_i \in \{0,1\}, \;\; i = 1, \ldots, n.$$

Перейдем к линейной релаксации этой задачи LP_{sat}:

$$\sum_{c \in f} w_c z_c \to \max$$
$$\sum_{i \in S_c^+} y_i + \sum_{i \in S_c^-} (1 - y_i) \geq z_c, \;\; c \in f,$$
$$0 \leq z_c \leq 1, \;\; c \in f,$$
$$0 \leq y_i \leq 1, \;\; i = 1, \ldots, n.$$

Следующий вероятностный алгоритм полагает значение каждой переменной равным 1 с вероятностью, равной значению этой переменной в оптимальном решении линейной релаксации.

Вероятностный алгоритм ЛП $(x_1, \ldots, x_n, f, w : f \to \mathbf{Q}^+)$

1 Решить задачу LP_{sat}.
Пусть (y^*, z^*) — оптимальное решение.
2 Независимо для каждого i :
$x_i \leftarrow 1$ с вероятностью y_i^*
$x_i \leftarrow 0$, иначе.
Назовем полученное назначение τ.
3 **return** τ.

Для $k \geq 1$ определим $\beta_k = 1 - (1 - \frac{1}{k})^k$.

Лемма 22. *Если $size(c) = k$, то $E[W_c] \geq \beta_k w_c z^*(c)$.*

Доказательство. Без ограничения общности можно считать, что в дизъюнкцию c все литералы входят без отрицания. Предположим, что $c = x_1 \vee \ldots \vee x_k$.

Дизъюнкция c выполнена, если не все ее переменные получили значение 0. Вероятность этого события:

$$\mathbf{Pr}[c \text{ выполнена}] = 1 - \prod_{i=1}^{k} (1 - y_i) \geq 1 - \left(\frac{\sum_{i=1}^{k} (1 - y_i)}{k} \right)^k =$$

$$1 - \left(1 - \frac{\sum_{i=1}^{k} y_i}{k}\right)^k \geq 1 - (1 - \frac{z_c^*}{k})^k.$$

Первое неравенство следует из неравенств $\frac{a_1+\ldots+a_k}{k} \geq \sqrt[k]{a_1 \cdot \ldots \cdot a_k}$, а последнее неравенство вытекает из ограничения $y_1 + \ldots + y_k \geq z_c$. Рассмотрим функцию $g(z) = 1 - (1 - \frac{z}{k})^k$. Это вогнутая функция, у которой $g(0) = 0$ и $g(1) = \beta_k$ (рис. 10.1). Для $z \in [0, 1]$ имеем $g(z) \geq \beta_k z$. Следовательно, $Pr[c \text{ выполнена}] \geq \beta_k z_c^*$. □

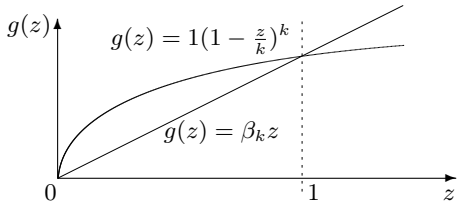


Рис. 10.1. Поведение функций $\beta_k z$ и $(1 - \frac{z}{k})^k$

Следствие 5. *$E[W] \geq \beta_k OPT$, если размер дизъюнкций не превосходит k.*

Доказательство. Замечая, что β_k — убывающая функция от параметра k, получим $E[W] = \sum_{c \in f} E[W] \geq \beta_k \sum_{c \in f} w_c z^* = \beta_k OPT$. □

Теорема 31. Вероятностный алгоритм ЛП *является* $(1 - \frac{1}{e})$-*приближенным алгоритмом.*

10.3 $\frac{3}{4}$-приближенный алгоритм

С равной вероятностью применим один из двух описанных алгоритмов. Пусть $b = 0$, если был применен алгоритм Джонсона, и $b = 1$, если был применен вероятностный алгоритм ЛП. Пусть z^* обозначает оптимальное решение ЛП.

Лемма 23. $E[W_c] \geq \frac{3}{4} w_c z^*(c)$.

Доказательство. Пусть $size(c) = k$. Тогда из леммы 20 и из неравенства $z^*(c) \leq 1$ получим

$$E[W_c | b = 0] = \alpha_k w_c \geq \alpha_k w_c z^*(c),$$

а из леммы 22 —

$$E[W_c | b = 1] \geq \beta_k w_c z^*(c).$$

Объединяя эти два неравенства, получим

$$E[W_c] = \frac{1}{2}(E[W_c | b = 0] + E[W_c | b = 1]) \geq w_c z^*(c) \frac{(\alpha_k + \beta_k)}{2}.$$

Заметим, что $\alpha_1 + \beta_1 = \alpha_2 + \beta_2 = 3/2$, и $\alpha_k + \beta_k \geq 7/8 + (1 - 1/e) \geq 3/2$, для $k \geq 3$, и, следовательно,

$$E[W_c] \geq \frac{3}{4} w_c z^*(c).$$

□

Алгоритм Гоеманса – Вильямсона $(x_1, \ldots, x_n, f, w : f \to \mathbf{Q}^+)$

1 Положить $p_i = \frac{1}{2}$ для всех i.
2 Применить Алгоритм Наибольшее среднее $(p_1, \ldots, p_n)$.
Назовем полученное назначение τ_1.
3 Решить задачу LP_{Sat}.
Пусть (y^*, z^*) — оптимальное решение.
4 Положить $p_i = y_i^*$ для всех i.
5 Применить Алгоритм Наибольшее среднее $(p_1, \ldots, p_n)$.
Назовем полученное назначение τ_2.
6 **return** лучшее из τ_1 и τ_2.

Теорема 32. Алгоритм Гоеманса – Вильямсона *является* $\frac{3}{4}$*-приближенным алгоритмом для задачи о максимальной выполнимости.*

Доказательство. Суммируя по среднему ожидаемому вкладу от каждой дизъюнкции, получим

$$E[W] = \sum_{c \in f} E[W_c] \geq \frac{3}{4} \sum_{c \in f} w_c z^*(c) = \frac{3}{4} OPT_{LP} \geq \frac{3}{4} OPT.$$

□

10.4 Упражнения

1. Рассмотрим пример задачи о максимальной выполнимости на булевых переменных x_1, x_2, x_3 при весе каждой дизъюнкции, равном 1:

   $x_1 \vee \overline{x}_2$,

   $x_2 \vee x_3$,

   $x_3 \vee \overline{x}_1$,

   $x_1 \vee x_2 \vee x_3$,

   $\overline{x}_1 \vee \overline{x}_2 \vee \overline{x}_3$.

   Решить этот пример Вероятностным алгоритмом Джонсона и Алгоритмом Гоеманса – Вильямсона. Сравнить полученные решения.

2. В лемме 20 показано, что если число литералов в каждой дизъюнкции не превосходит k, то Вероятностный алгоритм Джонсона является α_k-приближенным алгоритмом для задачи о максимальной выполнимости. Привести пример, показывающий, что эта оценка точна.

3. Предположим, что в примере задачи о максимальной выполнимости нет одноэлементных дизъюнкций, и в каждой дизъюнкции из двух элементов по крайней мере один элемент входит без отрицания. Рассмотрим для таких примеров следующий алгоритм. Независимо для каждого $i : x_i \leftarrow 1$ с вероятностью p и $x_i \leftarrow 0$, иначе. Подберите значение параметра p так, чтобы максимизировать стоимость математического ожидания получаемых решений в худшем случае.

4. Рассмотрим следующий алгоритм. Выбрать произвольное назначение истинности τ и пусть τ' — его дополнение, то есть переменная истинна в τ тогда и только тогда, когда она ложна в τ'. Подсчитать вес выполненных дизъюнкций при назначениях τ и τ'. Выдать лучшее назначение. Доказать, что этот алгоритм является $\frac{1}{2}$-приближенным алгоритмом для задачи о максимальной выполнимости.

5. Пусть (y^*, z^*) — оптимальное решение задачи LP_{Sat}. Определим величины $p_i = \frac{1}{2}y_i^* + \frac{1}{4}$ и применим Алгоритм Наибольшее среднее $(p_1, \ldots, p_n)$. Доказать, что полученный алгоритм является $\frac{3}{4}$-приближенным алгоритмом для задачи о максимальной выполнимости.

Литература

[1] Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М., Мир. 1982.

[2] Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. 1984.

[3] Ausiello G., Crescenzi P., Gambosi G., et al. Complexity and approximation: combinatorial optimization problems and their aproximability properties. Berlin: Springer-Verlag, 1999.

[4] Chvatal V. A greedy heuristic for the set covering problem. Mathematics of Operations Research, 4, 1979, pp. 233–235.

[5] Cook S. A. The complexity of theorem proving procedures. Proceeding of the 3th Annual ACM Symposium on the Theory of Computing, 1971, pp. 151–158.

[6] Feige U. A treshold of $\ln n$ for approximating set cover. Journal of the ACM, 45, 1998, pp. 634–652.

[7] Khot S. On the power of unique 2-prover 1-round games. In Proceedings of the 34th Annual ACM Symposium on the theory of Computing, 2002, pp. 767–775.

[8] Khot S., Regev O. Vertex cover might be hard to approximate to with $2-\varepsilon$. Journal of Computer and System Sciences, 74, 2008, pp. 335–349.

[9] Korte B., Vygen J. Combinatorial optimization. Theory and Algorithms. Berlin: Springer-Verlag, 2000.

Учебное издание

Кононов Александр Вениаминович,
Кононова Полина Александровна

ПРИБЛИЖЕННЫЕ АЛГОРИТМЫ
ДЛЯ NP-ТРУДНЫХ ЗАДАЧ

Учебно-методическое пособие

Редактор А.В. Грасмик

Подписано в печать 18.07.2014.
Формат 60×84 1/16. Печать офсетная.
Уч.-изд. л. 7,3. Усл. печ. л. 6,8. Тираж 60 экз.

Заказ №

Редакционно-издательский центр НГУ.
630090, г. Новосибирск, ул. Пирогова, 2.