МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

А.А. Тюхтина

МЕТОДЫ ДИСКРЕТНОЙ ОПТИМИЗАЦИИ

Часть 2

Учебно-методическое пособие

Рекомендовано методической комиссией
Института экономики и предпринимательства ННГУ для студентов, обучающихся
по направлению подготовки 38.04.05 «Бизнес – информатика»
(академическая магистерская программа
«Информационные технологии и аналитические методы моделирования
и оптимизации бизнес-процессов»)

Нижний Новгород
2015

УДК 519.854
ББК 22.18
Т 98

Т 98 Тюхтина А.А. Методы дискретной оптимизации. Часть 2: Учебно-методическое пособие. – Нижний Новгород: Нижегородский госуниверситет, 2015. – 72 с.

Рецензент: кандидат физ.-мат. наук, доцент **А.В. Калинин**

В учебно-методическом пособии рассматриваются метод динамического программирования и приближенные алгоритмы для решения задач дискретной оптимизации, изучаются подходы к получению оценок для оптимального значения целевых функций.

Данное пособие предназначено для студентов Нижегородского госуниверситета, обучающихся по направлению «Бизнес – информатика» (академическая магистерская программа «Информационные технологии и аналитические методы моделирования и оптимизации бизнес-процессов»). Содержание пособия соответствует программе дисциплины «Теория игр и исследование операций (продвинутый уровень)».

Ответственный за выпуск:
председатель методической комиссии
Института экономики и предпринимательства ННГУ,
к.э.н., доцент **Е.Н. Летягина**

УДК 519.854
ББК 22.18

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Методы исследования операций используются в настоящее время как для решения задач управления организационными системами различной природы, так и в теоретических исследованиях, например, при анализе экономико-математических моделей. Дисциплина Б1.В.ДВ.2 «Теория игр и исследование операций (продвинутый уровень)», преподаваемая студентам Института экономики и предпринимательства ННГУ им. Н.И. Лобачевского, обучающимся по направлению подготовки 38.04.05 «Бизнес - информатика» (академическая магистерская программа «Информационные технологии и аналитические методы моделирования и оптимизации бизнес-процессов») ориентирована на формирование у выпускника следующих компетенций:

– способность к творческой адаптации к конкретным условиям выполняемых задач и их инновационным решениям (ОПК-3);

– способность проводить научные исследования для выработки стратегических решений в области ИКТ (ПК-12);

– способность применять информационные технологии и аналитические методы для моделирования и оптимизации бизнес-процессов (ДКМП-1).

Многие задачи управления организационными системами и бизнес-процессами сводятся к выбору из конечного множества возможных решений наилучшего по определенному критерию, то есть допускают постановку в виде задач дискретной оптимизации.

В рамках дисциплины «Теория игр и исследование операций (продвинутый уровень)» рассматриваются как классические, так и современные подходы к решению задач дискретной оптимизации.

Настоящее учебно-методическое пособие посвящено изучению метода динамического программирования для задач многошагового оптимального управления, метода решения задач целочисленного линейного программирования, матрица системы ограничений которых имеет блочную структуру, а также некоторых эвристических алгоритмов.

1. ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Динамическое программирование – подход, позволяющий решать задачи оптимизации, которые могут быть сформулированы как задачи многошагового оптимального управления некоторой системой. Как правило, такие задачи обладают аддитивной или мультипликативной целевой функцией, то есть

$$f(x)=\sum_{k=1}^{n} f_k(x_k) \text{ или } f(x)=\prod_{k=1}^{n} f_k(x_k),\ x=(x_1,...,x_n).$$

Обычно предполагается, что все функции f_k принимают только положительные значения, но никакие условия регулярности (непрерывность, дифференцируемость, выпуклость и т.п.) на них не накладываются.

Как научное направление динамическое программирование сформировалось в начале 50-х годов XX-го века благодаря работам американского математика Ричарда Беллмана и его сотрудников [2]. Суть метода заключается в сведении исходной задачи к упорядоченной совокупности задач меньшей размерности и последовательного их решения. При этом на каждом шаге при решении очередной задачи используются результаты решения предыдущих задач. Поскольку любая подзадача определяет некоторое множество допустимых решений, метод динамического программирования относится к комбинаторным методам оптимизации, то есть представляет собой метод упорядоченного перебора вариантов.

Рассмотрим сначала общую схему динамического программирования для оптимизации управляемой системы, затем приведем примеры применения метода к решению некоторых задач дискретной оптимизации. Так как логарифм мультипликативной функции является аддитивной функцией, ограничимся изучением задач с аддитивным критерием.

1.1. Общая схема динамического программирования

В обобщенном виде метод динамического программирования заключается в следующем [6].

Рассматривается некоторый объект (управляемая система), *текущее состояние* которого описывается с помощью набора параметров ξ. Множество всех возможных состояний Ξ конечно. Изменение состояний системы осуществляется посредством управляющих воздействий, которые представляют собой выбор элемента *x* (*управления*) из некоторого множества *X*, называемого множеством управлений. Без ограничения общности можно считать, что $X \subseteq R^1$. В состоянии ξ доступно конечное множество управлений $X(\xi) \subseteq X$. Предполагается, что в системе отсутствует последействие, то есть состояние ξ', в которое переходит система после выбора управления, зависит только от этого управления и непосредственно предшествующего состояния ξ:

$$\xi' = \varphi(x, \xi),\ x \in X(\xi). \qquad (1.1)$$

Уравнения (1.1) называются *уравнениями состояния*.

Траекторией системы называется конечная последовательность $\xi = (\xi_1, ..., \xi_m)$ состояний системы такая, что

$$\xi_{k+1} = \varphi(x, \xi_k),\ x \in X(\xi_k),\ k = 1, ..., m-1.$$

Управление, посредством которого система переходит из состояния ξ_k в состояние ξ_{k+1}, обозначим через x_k. *Стратегией* управления для траектории $\xi = (\xi_1, ..., \xi_m)$ называется вектор допустимых управлений $x = (x_1, ..., x_{m-1})$. Очевидно, траектория однозначно определяется своим начальным состоянием и выбором стратегии.

Эффективность управления $x \in X(\xi)$ оценивается с помощью неотрицательной функции стоимости $f(x, \xi)$, зависящей не только от применяемого управляющего воздействия, но и от текущего состояния системы. Стоимость траектории определяется как сумма стоимостей отдельных её этапов, то есть

$$f(\xi) = \sum_{k=1}^{m-1} f(x_k, \xi_k).$$

По определению траектории, её стоимость зависит только от её начального состояния и набора пошаговых управлений.

Пусть заданы начальное состояние системы ξ_0 и конечное состояние ξ_T. Связывающая их траектория $(\xi_0, ..., \xi_T)$ называется *полной*. Обычно требуется построить полную траекторию, имеющую наибольшую или наименьшую стоимость. Для определенности будем рассматривать задачу на минимум. Стратегия, определяющая оптимальную полную траекторию, называется оптимальной стратегией.

Метод динамического программирования – это метод пошаговой оптимизации, поэтапного построения оптимального управления. В его основе лежит принцип оптимальности, впервые сформулированный Р. Беллманом в 1953 г. следующим образом:

Оптимальное поведение обладает тем свойством, что каковы бы ни были первоначальное состояние и решение в начальный момент, последующие решения должны составлять оптимальное поведение относительно состояния, получающегося в результате первого решения.

Принцип может быть обоснован доказательством от противного. Пусть, далее, $(\xi_0, \xi_1, \xi_2, ..., \xi_T)$ – полная оптимальная траектория. В соответствии с принципом оптимальности, траектория $(\xi_1, \xi_2, ..., \xi_T)$ будет иметь оптимальную стоимость среди всех траекторий, соединяющих ξ_1 и ξ_T. Приняв ξ_1 за начальное состояние и применяя принцип оптимальности получим, что траектория $(\xi_2, ..., \xi_T)$ – лучшая среди траекторий, соединяющих ξ_2 и ξ_T, и так далее: *если траектория оптимальна, то любой её заканчивающийся в* ξ_T *отрезок также*

оптимален. Аналогично обосновывается оптимальность любого отрезка оптимальной траектории, начинающегося с ξ_0.

Пусть ξ – произвольное состояние системы, через $F^T(\xi)$ обозначим стоимость оптимальной траектории, соединяющей ξ с финальным состоянием ξ_T. Очевидно, $F^T(\xi_T)=0$, $F^T(\xi_0)$ – стоимость оптимальной полной траектории.

Из принципа оптимальности Беллмана вытекает, что

$$F^T(\xi)=\min_{x\in X(\xi)}\left\{f(x,\xi)+F^T(\varphi(x,\xi))\right\}. \qquad (1.2)$$

Обозначим теперь через $F^0(\xi)$ стоимость оптимальной траектории, соединяющей начальное состояние ξ_0 с состоянием ξ. Очевидно, $F^0(\xi_0)=0$, а стоимость оптимальной полной траектории равна $F^0(\xi_T)$. Пусть $\Gamma(\xi)$ – множество состояний, непосредственно предшествующих состоянию ξ, то есть таких состояний ξ', что существует управление $x(\xi',\xi)\in X(\xi')$, переводящее систему из ξ' в ξ, $\xi=\varphi(x(\xi',\xi),\xi')$.

В соответствии с принципом оптимальности,

$$F^0(\xi)=\min_{\xi'\in\Gamma(\xi)}\left\{f(x(\xi',\xi),\xi')+F^0(\xi')\right\}. \qquad (1.3)$$

Функции $F^T(\xi)$ и $F^0(\xi)$ называются *функциями Беллмана*, а рекуррентные соотношения (1.2) и (1.3) – *уравнениями Беллмана*.

В основе алгоритма динамического программирования могут лежать либо соотношения (1.2), тогда он называется методом обратной прогонки, либо соотношения (1.3), и алгоритм называют методом прямой прогонки.

Метод обратной прогонки (обратный метод Беллмана) осуществляется следующим образом. Последовательно вычисляются значения функции Беллмана:

$$F^T(\xi_T)=0,\ F^T(\xi)=f(x(\xi,\xi_T),\xi) \text{ для } \xi\in\Gamma(\xi_T),$$

далее, с помощью соотношений (1.2) постепенно определяются $F^T(\xi)$ для таких состояний ξ, что найдены значения функции Беллмана для всех состояний $\varphi(x,\xi)$, $x\in X(\xi)$, и, в конце концов, вычисляется $F^T(\xi_0)$.

Пусть $x^*(\xi)\in X(\xi)$ – управление, минимизирующее правую часть в (1.2), оно называется *условным оптимальным управлением*. Тогда оптимальная полная траектория $(\xi_0,\xi_1,\xi_2,...,\xi_T)$ восстанавливается следующим образом:

$$\xi_0,\ \xi_1=\varphi(x^*(\xi_0),\xi_0),\ \xi_2=\varphi(x^*(\xi_1),\xi_1),\ \ldots$$

При осуществлении **метода прямой прогонки** определяются значения функции $F^0(\xi)$ в следующем порядке:

$$F^0(\xi_0)=0,\ F^0(\xi)=f(x,\xi) \text{ для } x\in X(\xi_0),\ \xi=\varphi(x,\xi_0),$$

далее, на основании соотношений (1.3), последовательно вычисляются $F^0(\xi)$ для таких состояний ξ, что найдены значения функции Беллмана для всех состояний из $\Gamma(\xi)$, и, в конце концов, определяется $F^0(\xi_T)$.

Пусть $\gamma(\xi) \in \Gamma(\xi)$ – состояние, минимизирующее правую часть в (1.3). Тогда условные оптимальные управления – $x(\gamma(\xi),\xi)$. Оптимальная траектория восстанавливается начиная с финального состояния:

$$\xi_T,\ \xi_{T-1} = \gamma(\xi_T),\ \ldots.$$

Таким образом, во время реализации алгоритма динамического программирования дважды осуществляется многошаговый процесс: сначала находятся условные оптимальные управления и значения функции Беллмана от конечного состояния до начального или от начального состояния до конечного, затем восстанавливается оптимальная траектория и оптимальная стратегия в обратном направлении – от начала к концу или от конца к началу соответственно. Число элементарных операций, необходимых для реализации обратного и прямого методов Беллмана, квадратично зависит от числа состояний системы, причем большая часть вычислений приходится на этап условной оптимизации.

Несмотря на одинаковую вычислительную сложность, алгоритм обратной прогонки используется чаще, чем алгоритм прямой прогонки.

Представим рассматриваемую управляемую систему ориентированным графом, вершины которого соответствуют состояниям системы, а дуги – управлениям следующим образом: дуга, отвечающая управлению $x \in X(\xi)$, идет из вершины ξ в вершину $\varphi(x,\xi)$ и имеет длину $f(x,\xi)$. Любой траектории системы соответствует ориентированный путь в этом графе и стоимость траектории равна длине этого пути. Таким образом, задача поиска оптимальной траектории (оптимальной стратегии управления) сводится к задаче поиска кратчайшего пути в графе, соединяющего начальную вершину ξ_0 и конечную вершину ξ_T. Естественно считать, что граф ациклический – система не может вернуться в уже пройденное состояние. Кроме того, не существует дуг, входящих в вершину ξ_0 и дуг, исходящих из ξ_T. Принцип оптимальности утверждает ни что иное, что *любой подпуть кратчайшего или самого длинного пути сам является кратчайшим или самым длинным*.

Многие задачи динамического программирования формулируются таким образом, что система переводится из начального состояния в финальное за фиксированное число n шагов (этапов), то есть стратегия управления – n-мерный вектор $x = (x_1,\ldots,x_n)$, где x_k – управление, выбираемое на k-м шаге, $k = 1,\ldots,n$. Каждый этап определяется множеством состояний, в которых система может находится перед выбором управления, либо в которые может перейти в результате управления, причем в начале первого этапа система находится в начальном состоянии ξ_0, а в конце последнего – в финальном состояния ξ_T. Таким образом, все полные траектории включают $n+1$ состояние

$(\xi_0,...,\xi_n \equiv \xi_T)$. Можно считать, что функции состояния (1.1) и эффективность управления изменяются от этапа к этапу, то есть

$$\xi_k = \varphi_k(x_k, \xi_{k-1}),\ k = 1,...,n, \tag{1.4}$$

стоимость шага $k = 1,...,n$ равна $f_k(x_k, \xi_{k-1})$, а стоимость стратегии определяется аддитивной функцией $f(x) = \sum_{k=1}^{n} f_k(x_k, \xi_{k-1})$.

Пусть в начале шага t система находится в состоянии ξ. Обозначая через $F_t(\xi)$ минимальное значение $\sum_{k=t}^{n} f_k$, то есть $F^T(\xi)$, получаем, что соотношение (1.2) примет вид

$$F_t(\xi) = \min_{x_t}\{f_t(x_t, \xi) + F_{t+1}(\xi_t)\},\ \xi_t = \varphi_t(x_t, \xi), \tag{1.5}$$

где минимум берется по всем допустимым на этапе t управлениям. Решение задачи (1.5) – условное оптимальное управление на шаге t – обозначается $\hat{x}_t(\xi)$. Процедура обратной прогонки динамического программирования осуществляется следующим образом.

Пусть $F_{n+1} \equiv 0$. Используя рекуррентные соотношения (1.5), последовательно определяем $F_t(\xi)$ и $\hat{x}_t(\xi)$ для всех состояний шага t, $t = n,...,1$, причем $F_n(\xi) = \min_{x_n}\{f_n(x_n, \xi)\}$. Далее поэтапно восстанавливаются оптимальная траектория и оптимальная стратегия:

$$\xi_0,\ x_1^* = \hat{x}_1(\xi_0),\ \xi_1 = \varphi_1(x_1^*, \xi_0),\ \xi_t = \varphi_t(x_t^*, \xi_{t-1}),\ x_t^* = \hat{x}_t(\xi_{t-1}),\ t = 2,....,n.$$

При использовании процедуры прямой прогонки через $F_t(\xi)$ обозначается $F^0(\xi)$, то есть минимальное значение $\sum_{k=1}^{t} f_k$ при условии, что в конце этапа t система находится в состоянии ξ. Условное оптимальное управление на этапе t выбирается в предположении, что предыдущие этапы пройдены оптимально, и рекуррентное соотношение (1.3) принимает вид

$$F_t(\xi) = \min_{x_t}\{f_t(x_t, \xi_{t-1}) + F_{t-1}(\xi_{t-1})\},\ \xi = \varphi_t(x_t, \xi_{t-1}). \tag{1.6}$$

Полагая $F_0 \equiv 0$, имеем

$$F_1(\xi) = \min_{x_1}\{f_1(x_1, \xi_0)\},\ \xi = \varphi_1(x_1, \xi_0).$$

Последовательно вычисляя $F_t(\xi)$, $\hat{x}_t(\xi)$ для всех возможных в конце этапа t состояний, $t = 1,...,n$, на последнем шаге получим $x_n^* = \hat{x}_n(\xi_T)$. Пользуясь уравнениями состояния, восстановим в обратном порядке оптимальный план и оптимальную траекторию:

$$\xi_T,\ x_n^* = \hat{x}_n(\xi_T),\ \xi_t = \varphi_t(x_t^*, \xi_{t-1}),\ x_t^* = \hat{x}_t(\xi_t),\ \ t = n-1,...,1.$$

Общий алгоритм постановки и решения задачи динамического программирования может быть следующим.

– Разделить процедуру решения задачи на этапы. Иногда это разбиение обеспечивается постановкой задачи, иногда требуется вводить его искусственно. Обычно, чем больше количество шагов, тем проще реализация каждого отдельного шага, однако, с другой стороны, слишком мелкое разбиение может приводить к лишним расчетам, усложняющим процедуру поиска оптимального решения.

– Выбрать параметры, характеризующее состояние управляемой системы перед каждым шагом. Набор этих параметров не должен зависеть от этапа и их количество должно быть по возможности небольшим. Если состояние системы описывается многими параметрами (так называемыми фазовыми координатами), то становится трудно перед каждым шагом перебрать все их варианты и для каждого найти оптимальное условное управление. Последнее ещё больше затрудняется если и число возможных вариантов управления велико. В этих случаях применение метода динамического программирования может стать нерациональным, эту проблему Р. Беллман назвал «проклятием многомерности».

– Определить допустимые управления для каждого состояния на каждом этапе.

– Определить функции состояния (1.4), связывающие этапы между собой, и функции стоимости управлений.

– Записать основное рекуррентное уравнение динамического программирования (1.5) или (1.6).

– Найти, пользуясь составленным уравнением, значения соответствующей функции Беллмана на каждом шаге и условные оптимальные управления. При решении конкретной задачи оптимизации часто оказывается возможным сократить объём вычислений за счёт состояний, заведомо не принадлежащих оптимальной траектории.

– Восстановить оптимальную траекторию и оптимальное управление.

Замечание 1.1. Метод динамического программирования может применяться только для управления системами, в которых множества допустимых в каком–либо состоянии управлений и, соответственно, выбор оптимального управления, не зависят от того, каким путём система пришла в это состояние, то есть не зависят от предыстории управляемого процесса. Иногда выполнение этого условия можно обеспечить, увеличив размерность задачи или количество состояний системы.

Замечание 1.2. Задачи динамического программирования с мультипликативным критерием можно решать непосредственно, без логарифмирования целевой функции, заменив в используемых рекуррентных соотношениях сумму на умножение и соответственно скорректировав начальные условия.

Замечание 1.3. Алгоритм динамического программирования может быть обобщен на случай задач «на узкое место», в которых требуется найти полную траекторию с минимальным значением максимального из пошаговых стоимостей или максимальным значением минимальной из пошаговых стоимостей.

1.2. Кратчайшие пути в сети

Как указано в предыдущем параграфе, задача о поиске кратчайшего пути в ориентированном графе является, в определённом смысле, основной задачей динамического программирования. Пусть в графе имеется одна начальная вершина – вершина, в которую не входит не одна дуга, и одна вершина, из которой ни одна дуга не выходит – конечная. Длиной пути называется сумма длин образующих его дуг. Требуется найти ориентированный путь из начальной вершины в конечную, имеющий наименьшую длину.

Если граф не содержит ориентированных циклов, его вершины можно перенумеровать так, чтобы для любой дуги (i, j) выполнялось неравенство $i < j$. Одновременно можно разбить вершины на подмножества (*слои*) таким образом, что вершины слоя не связаны дугами друг с другом, входящими дугами с вершинами следующих слоёв и исходящими дугами с вершинами предыдущих слоёв.

Один из возможных и простейших алгоритмов указанного упорядочения вершин следующий. Начальной вершине присваивается номер 1 и она образует первый слой. Из рассмотрения удаляется начальная вершина с исходящими из неё дугами, вершины, в которые теперь не входят дуги, относят ко второму слою и нумеруются в произвольном порядке последовательными числами 2, 3, …. Удаляются уже пронумерованные вершины с исходящими из них дугами и так далее.

Длину дуги (i, j) будем обозначать далее c_{ij}. В матрице длин дуг будут определены элементы, лежащие выше главной диагонали. Длины отсутствующих дуг можно положить равной бесконечности.

Обозначим через r_i длину кратчайшего пути из вершины i до финальной вершины n. Тогда соотношение (1.2) можно переписать в виде

$$r_i = \min_j \{c_{ij} + r_j\},\ i = n-1,...,1, \tag{1.7}$$

где минимум берется по всем дугам, исходящим из вершины i. Значения r_i вычисляются в порядке убывания номеров, начиная с $r_n = 0$, причём r_1 - длина искомого пути. Для восстановления оптимального пути либо для каждой вершины i на этапе условной оптимизации запоминается следующая вершина j, решение задачи (1.7), либо вершина, следующая за i на оптимальном пути, находится из условия

$$r_i - r_j = c_{ij}. \tag{1.8}$$

Аналогично, если r_i - длина кратчайшего пути из вершины 0 в вершину i, то соотношение (1.3) для прямой прогонки примет вид

$$r_i = \min_j \{c_{ji} + r_j\},\ i = 2,...,n, \tag{1.9}$$

где минимум берется по всем дугам, входящим в вершину i. Значения r_i вычисляются в порядке возрастания номеров, начиная с $r_1 = 0$.

Пример 1.1. Найдём кратчайший путь из вершины 1 в вершину 8 в графе, изображенном на рис. 1.1.

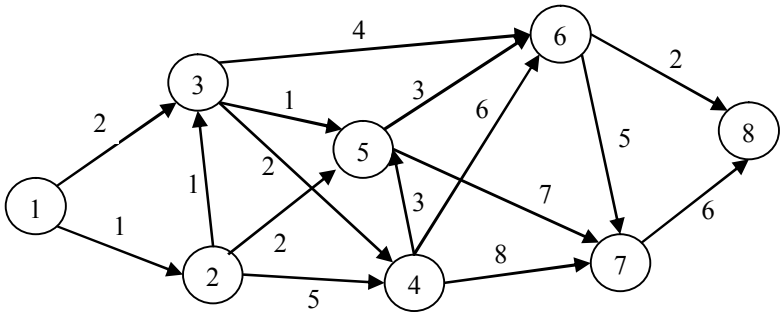


Рис.1.1. Граф для примера 1.1

Нетрудно проверить, что граф удовлетворяет условиям применения алгоритма. Вычисления по формуле (1.7) или (1.9) можно проводить, используя рисунок:

$$r_8 = 0,\ r_7 = 6,\ r_6 = \min\{2,5+6\} = 2,\ r_5 = \min\{3+2,7+6\} = 5,$$
$$r_4 = \min\{3+5,6+2,8+6\} = 8,\ r_3 = \min\{2+8,1+5,4+2\} = 6,$$
$$r_2 = \min\{1+6,5+8,2+6\} = 7,\ r_1 = \min\{1+7,2+6\} = 8;$$

или

$$r_1 = 0,\ r_2 = 1,\ r_3 = \min\{2,1+1\} = 2,\ r_4 = \min\{1+5,2+2\} = 4,$$
$$r_5 = \min\{1+2,2+1,4+3\} = 3,\ r_6 = \min\{2+4,6+6,3+3\} = 6,$$
$$r_7 = \min\{6+8,3+7,6+5\} = 10,\ r_8 = \min\{6+2,10+6\} = 8.$$

Приведем матричную форму алгоритма прямой прогонки. Метка r_i, $i = 2,3,...$, определяется как минимальная сумма элементов столбца i матрицы длин дуг, стоящих в строках $1,...,i-1$, и соответствующих элементов столбца меток. Элементы, на которых достигается минимум в (1.9), подчеркнуты (рис. 1.2). Кратчайшие пути восстанавливаются в обратном направлении:

8 – 6 (строка с подчеркнутым элементом в столбце 8) – 5 (строка с подчеркнутым элементом в столбце 6) – 3 (строка с подчеркнутый элементом в столбце 5) – 2 (строка с подчеркнутым элементом в столбце 3) – 1.

Аналогично получаем пути 8–6–5–3–1, 8–6–5–2–1, 8–6–3–2–1, 8–6–3–1, все длиной 8.

	2	3	4	5	6	7	8	r_i
1	1	2	–	–	–	–	–	0
2	0	1	5	2	–	–	–	1
3	–	0	2	1	4	–	–	2
4	–	–	0	3	6	8	–	4
5	–	–	–	0	3	7	–	3
6	–	–	–	–	0	5	2	6
7	–	–	–	–	–	0	6	10
8	–	–	–	–	–	–	0	8

Рис. 1.2. Матрица длин дуг для примера 1.1

Пусть в графе имеется m слоёв. По построению, путь, связывающий вершины слоя j и $j-1$, $j=2,...,m$, состоит не более чем из одной дуги, следовательно, путь из начальной вершины (образующей первый слой) в конечную (слой m) включает не более чем $m-1$ дугу. Таким образом, процесс поиска кратчайшего пути по формулам (1.7) или (1.9) можно разбить на $m-1$ этап, на каждом из которых рассматриваются вершины одного слоя.

Обозначим слой k через S_k. Пусть $i \in S_k$, $k=2,...,m$, $R_j(i)$ – длина кратчайшего пути из вершины 1 в вершину i, проходящего через вершину слоя j, $j<k$. Очевидно,

$$R_j(i)=\min_{l\in S_j}\{r_l+c_{li}\},\ r_i=\min_{j<k}R_j(i). \qquad (1.10)$$

Полагаем, как и ранее, $r_1=0$, тогда $R_1(i)=c_{1i}$ для $i=2,...,n$.

Аналогично может быть модифицирован и алгоритм обратной прогонки.

Вычисления на основании соотношений (1.10) эффективны, если вершины каждого слоя связаны только с вершинами непосредственно предшествующего и следующего слоёв. В этом случае на шаге k требуется определять только $R_{k-1}(i)$, а любой путь из первой вершины в конечную содержит $m-1$ дугу.

Пример 1.2. Найдем путь минимальной длины из вершины 1 в вершину 10 для сети, изображенной на рисунке 1.3. Имеется 5 слоёв, $S_1=\{1\}$, $S_2=\{2,3,4\}$, $S_3=\{5,6,7\}$, $S_4=\{8,9\}$, $S_5=\{10\}$. Полагаем $r_1=0$. Далее используем соотношение (1.9):

$$k=2,\ R_1(2)=2,\ R_1(3)=4,\ R_1(4)=3;$$

$$k=3,\ R_2(5)=\min\{r_2+c_{25},r_3+c_{35},r_4+c_{45}\}=\min\{8,12,9\}=8,$$

$$R_2(6)=\min\{r_2+c_{26},r_3+c_{36},r_4+c_{46}\}=4,\ R_2(7)=\min\{r_3+c_{37},r_4+c_{47}\}=11;$$

аналогично, при $k=4$, $R_3(8)=10$, $R_3(9)=8$ и, наконец, $R_4(10)=16$. Кратчайший путь длиной 16: 1–4–6–9–10.

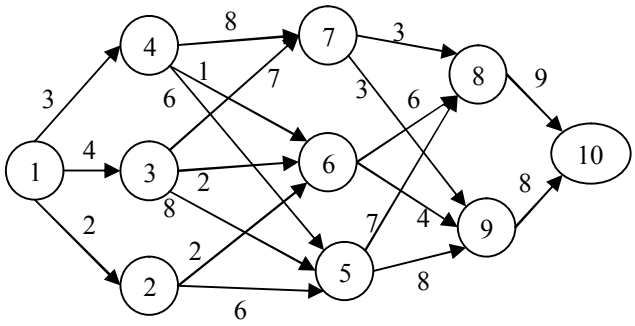


Рис. 1.3. Граф для примера 1.2

Алгоритмы прямой прогонки поиска кратчайшего пути из начальной вершины в конечную находят также кратчайшие пути из начальной вершины во все остальные. Два предыдущих алгоритма предназначены для поиска кратчайших путей в ациклическом графе. Рассмотрим более общий случай.

Пусть требуется найти кратчайшие пути из одной вершины, например, вершины 1, во все остальные в связном графе с положительными длинами дуг.

Алгоритм Дейкстры [14] для решения этой задачи определяет кратчайшие пути в порядке увеличения их длины. На этапе k, $k=1,...,n-1$, осуществляется поиск пути, состоящего не более чем из k дуг и образованного, в соответствии с принципом оптимальности, присоединением одной дуги к какому-либо уже найденному кратчайшему пути. Пусть M_k – множество вершин, для которых найдены длины кратчайших путей к концу *k*-го этапа, R_k – длина *k*-го кратчайшего пути, c_i^k – длина дуги, связывающей конечную вершину *k*-го пути с вершиной *i*. Тогда, полагая $M_0=\{1\}$, можем записать

$$R_k=\min_{l<k,\, j\notin M_{k-1}}\{R_l+c_j^l\}=\min_{i\in M_{k-1},\, j\notin M_{k-1}}(r_i+c_{ij}). \qquad (1.11)$$

Если минимум в (1.11) достигается на паре (i,j), то *k*-й кратчайший путь получается присоединением дуги (i,j) к пути до вершины i.

Итерации алгоритма могут быть осуществлены с помощью расстановки меток вершин – временных l_j, равных длине текущего пути в вершину, и постоянных r_j, обозначающих, как и ранее, длину кратчайшего пути в вершину. Изначально первая вершина получает постоянную метку $r_1=0$, остальные вершины – временные метки $l_j=c_{1j}$. Среди всех временных меток находится минимальная и делается постоянной, допустим, это метка вершины i. Метки смежных с ней вершин пересчитываются по формуле

$$l_j=\min\{l_j, r_i+c_{ij}\}.$$

Далее опять находится минимальная временная метка, делается постоянной и процесс повторяется, пока все метки не станут постоянными. Сами кратчайшие пути восстанавливаются по формуле (1.8).

Матричная форма алгоритма Дейкстры. В матрице длин дуг вычеркнуть первый столбец, присвоить первой строке метку $r_1=0$. На шаге k ($k=1,...,n-1$) найти $R_k=\min\{r_i+c_{ij}\}$ по всем невычеркнутым элементам всех отмеченных строк i. Пусть минимум в определении R_k достигается на элементе (i,j), вычеркнуть столбец j и положить $r_j=R_k$.

Пример 1.3. Найдём кратчайшие пути из вершины 1 во все остальные в графе на рис. 1.4.

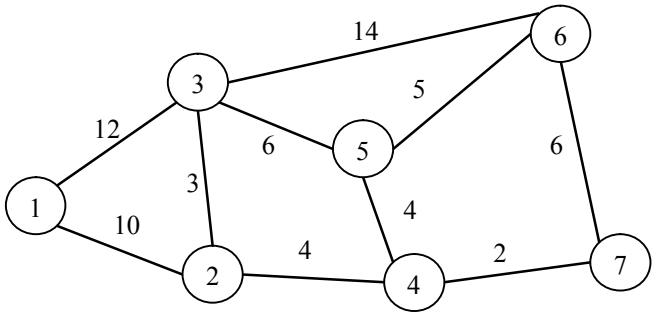


Рис.1.4. Граф для примера 1.3

Составим матрицу длин дуг (рис. 1.5). $R_1 = \min\{c_{1j}\} = c_{12} = 10$, $r_2 = 10$. Продолжаем, подчеркивая элементы матрицы, на которых достигается минимум в определении R_k.

	1	2	3	4	5	6	7	r_i
1	0	10	12	–	–	–	–	0
2	10	0	3	4	–	–	–	10
3	12	3	0	–	6	14	–	12
4	–	4	–	0	4	–	2	14
5	–	–	6	4	0	5	–	18
6	–	–	14	–	5	0	6	22
7	–	–	–	2	–	6	0	16

Рис. 1.5. Матрица длин дуг для примера 1.3

Восстановим, например, кратчайший путь до вершины 4. Подчеркнутый элемент в четвертом столбце находится в строке 2, подчеркнутый элемент второго столбца – в строке 1, следовательно, искомый путь – 1–2–4. Кратчайшие пути в вершины указаны на рисунке 1.6.

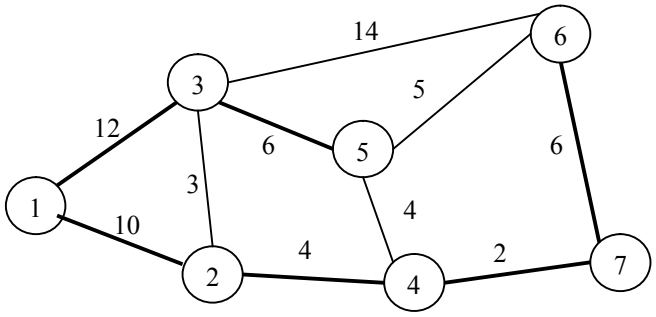


Рис.1.6. Кратчайшие пути для примера 1.3

Рассмотренные алгоритмы можно применять и для поиска самого длинного пути, заменив в соответствующем рекуррентном соотношении min на max и положив длину отсутствующих дуг равной $-\infty$.

1.3. Метод динамического программирования для задач о рюкзаке

Рассмотрим следующую задачу целочисленной оптимизации с аддитивной целевой функцией, обобщающую задачу о рюкзаке:

$$\sum_{j=1}^{n} f_j(x_j) \to \max \tag{1.12}$$

$$\sum_{j=1}^{n} c_j(x_j) \le b, \tag{1.13}$$

$$0 \le x_j \le d \text{ и целые, } j = 1,...,n. \tag{1.14}$$

Содержательно задача (1.12)–(1.14) может быть интерпретирована следующим образом. Имеется n видов активов (инвестиционных проектов, акций

предприятий и т.п.), причём x единиц j-го актива имеют стоимость $c_j(x)$ и приносят прибыль $f_j(x)$. Требуется распределить между этими активами средства в объеме b так, чтобы суммарная прибыль была максимальна.

Предполагаем, что все функции c_j неотрицательны, принимают целые значения и $c_j(0)=0$, f_j – произвольные функции такие, что $f_j(0)=0$.

Разобьем процесс решения на n шагов, на каждом шаге j принимается решение о вложениях в один актив x_j, стоимость шага j равна $f_j(x_j)$. В качестве состояния системы рассматривается количество $\xi = 0,...,b$ распределяемых средств. Обозначим через $F_k(\xi)$ прибыль от оптимального вложения средств ξ в первые k активов, то есть максимальное значение целевой функции в задаче

$$\sum_{j=1}^{k} f_j(x_j) \to \max$$

$$\sum_{j=1}^{k} c_j(x_j) \le \xi,$$

$$0 \le x_j \le d \text{ и целые, } j=1,...,k.$$

Оптимальное значение целевой функции исходной задачи равно $F_n(b)$, функции состояния имеют вид

$$\varphi_j(x_j,\xi)=\xi + c_j(x_j),$$

где ξ – средства, распределяемые между ресурсами $1,...,j-1$. Запишем рекуррентное соотношение для процедуры прямой прогонки:

$$F_k(\xi)=\max\{f_k(x_k)+F_{k-1}(\xi - c_k(x_k))\}, \qquad (1.15)$$

где максимум берется по всем целым $0 \le x_k \le d$ таким, что $c_j(x_j) \le \xi$.

Условия $F_0 \equiv 0$ и $F_k(0)=0$, k=1,…,n, означают, что прибыль равна нулю, если средства не распределяются. Начиная с $F_1(\xi)=\max_{x_1}\{f_1(x_1)\}$, определяются $F_k(\xi)$ и условные оптимальные управления $\hat{x}_k(\xi)$ для $k=1,...,n$ и $\xi=0,...,b$. На последнем шаге достаточно найти только $F_n(b)$ и $\hat{x}_n(b)$. С использованием соотношений

$$x_n^* = \hat{x}_n(b),\ \xi_{n-1} = b - c_n(x_n^*),\ x_{n-1}^* = \hat{x}_{n-1}(\xi_{n-1}),\ \xi_{n-2} = \xi_{n-1} - c_{n-1}(x_{n-1}^*),\ldots$$

вычисляются все компоненты оптимальной стратегии.

Рекуррентные соотношения вида (1.15) позволяют получить оптимальное решение рассматриваемой задачи вида (1.12) – (1.14) для любого целочисленного значения $0 \le b' \le b$.

Применим изложенный алгоритм для решения **задач о рюкзаке**. Положим $f_j(x)=c_j x$, $c_j(x)=a_j x$. Как и ранее, предполагается, что $c_j>0$, $0<a_j \le b$ ($j=1,...,n$), все параметры целые. $F_k(\xi)$ теперь означает максимальную

ценность рюкзака вместимостью ξ, заполненного предметами первых k типов, то есть оптимальное значение целевой функции в задаче

$$\sum_{j=1}^{k} c_j x_j \to \max$$

$$\sum_{j=1}^{k} a_j x_j \le \xi,$$

$$x_j \ge 0 \text{ и целые, } j = 1,...,k.$$

Уравнение (1.15) принимает вид

$$F_k(\xi) = \max\{c_k x_k + F_{k-1}(\xi - a_k x_k)\},\ x_k \le \xi / a_k. \qquad (1.16)$$

При k=1 $F_1(\xi) = \max\limits_{x_1 \le \xi / a_1} \{c_1 x_1\}$, то есть $\hat{x}_1(\xi) = [\xi / a_1]$, где через $[x]$ обозначается наименьшее целое число, большее или равное x. На шаге $k = 2,...,n$ удобно значения максимизируемой в (1.16) функции заносить в таблицу вида 1.1. Тогда $F_k(\xi)$ – максимальное значение в строке ξ, столбец, в котором оно расположено, определяет $\hat{x}_k(\xi)$.

Таблица 1.1

Этап условной оптимизации

$\xi \backslash x_k$	0	1	…	$[\xi / a_к]$
0				
1				
…				
b				

В процессе условной оптимизации постепенно формируется таблица (табл. 1.2), на основании которой восстанавливается оптимальный план: если найдено значение $x_k^* = \hat{x}_k(\xi_k)$, то $\xi_{k-1} = \xi_k - a_k x_k^*$ определяет строку, в которой расположено значение x_{k-1}^*.

Таблица 1.2

Итоговая таблица

ξ	$F_1(\xi)$	$\hat{x}_1(\xi)$	…	$F_n(\xi)$	$\hat{x}_n(\xi)$
0					
1					
…					
b					

Пример 1.4. Решим следующую задачу:

$$4x_1 + 2x_2 + 3x_3 \to \max$$

$$3x_1 + 2x_2 + 2x_3 \le 7,$$

$$x_j \ge 0 \text{ и целые, } j = 1,2,3.$$

$k = 1$. Вычисляем $\hat{x}_1(\xi) = [\xi / 3]$, $F_1(\xi) = 4\hat{x}_1(\xi)$ и заполняем первые два столбца таблицы 1.5.

$k=2$. $F_2(\xi)=\max\{2x_2+F_1(\xi-2x_2)\}$, $x_2\le\xi/2$.

Вычисляем $2x_2+F_1(\xi-2x_2)$ для всех $\xi=0,...,7$ и $x_2\le[\xi/2]$ (табл. 1.3).

Таблица 1.3

Второй шаг алгоритма

$x_2\backslash\,\xi$	0	1	2	3	4	5	6	7
0	0	0	0	4	4	4	8	8
1			2	2	2	6	6	6
2					4	4	4	8
3							6	6

Находим значения функции Беллмана (максимальные элементы столбцов) и заполняем третий и четвертый столбцы таблицы 1.5.

$k=3$. $F_3(\xi)=\max\{3x_3+F_2(\xi-2x_3)\}$, $x_3\le[\xi/2]$.

Вычисляем $3x_3+F_2(\xi-2x_3)$ для всех $\xi=0,...,7$ и $x_3\le[\xi/2]$ (табл. 1.4). Заканчиваем формировать таблицу 1.5.

Таблица 1.4

Третий шаг алгоритма

$x_3\backslash\,\xi$	0	1	2	3	4	5	6	7
0	0	0	2	4	4	6	8	8
1			3	3	5	7	7	9
2					6	6	8	10
3							9	9

Таблица 1.5

Итоговая таблица

ξ	$F_1(\xi)$	$\hat{x}_1(\xi)$	$F_2(\xi)$	$\hat{x}_2(\xi)$	$F_3(\xi)$	$\hat{x}_3(\xi)$
0	0	0	0	0	0	0
1	0	0	0	0	0	0
2	0	0	2	1	3	1
3	4	1	4	0	4	0
4	4	1	4	0 или 2	6	2
5	4	1	6	1	7	1
6	8	2	8	0	9	3
7	8	2	8	0 или 2	10	2

Максимальное значение целевой функции равно 10. Восстановим решение: $x_3=2$, $x_2=\hat{x}_2(7-4)=0$, $x_1=\hat{x}_1(3)=1$.

Можно использовать следующую модификацию решения (1.16) [14]. Если $x_k=0$, то $F_k(\xi)=F_{k-1}(\xi)$, случай $x_k\ge1$ возможен только при $\xi\ge a_k$ и тогда $F_k(\xi)=c_k+F_k(\xi-a_k)$. Таким образом, соотношение (1.16) принимает вид

$$F_k(\xi)=\max\{c_k+F_k(\xi-a_k);F_{k-1}(\xi)\}\text{ при }\xi\ge a_k,$$

$$F_k(\xi)=F_{k-1}(\xi),\text{ если }\xi<a_k. \qquad (1.17)$$

Значения $F_k(\xi)$ вычисляются последовательно для $\xi=0,...,b$. В качестве управления на k-м шаге выступает решение брать или не брать k-й предмет, $x_k\ge1$ или $x_k=0$.

Пусть $i_k(\xi)$ – максимальный индекс ненулевой переменной, использующейся в $F_k(\xi)$, то есть если $i_k(\xi)=j$, то $x_j \ge 1$ и $x_l = 0$ для $l > j$. Полагаем $i_1(\xi)=0$, если $F_1(\xi)=0$ и $i_1(\xi)=1$, если $F_1(\xi)\ne 0$.

В общем случае, если $c_k + F_k(\xi - a_k) \ge F_{k-1}(\xi)$, то есть $x_k \ge 1$, то $i_k(\xi)=k$. Если же $c_k + F_k(\xi - a_k) < F_{k-1}(\xi)$, то, очевидно, $i_k(\xi)=i_{k-1}(\xi)$.

Для определения значений формирующих $F_k(\xi)$ переменных $x_j(\xi)$, $j \le k$, поступают следующим образом. Пусть $i_k(\xi)=l$. Тогда

$$x_j(\xi)=0,\ j \ge l,\ x_l(\xi)\ge 1.$$

Если $i_k(\xi - a_l)=l$, то $x_l(\xi)\ge 2$ и нужно перейти к $i_k(\xi - 2a_l)$, если же $i_k(\xi - a_l)=i<l$, то $x_l(\xi)=1$, $x_j(\xi)=0$ для $l>j>i$, $x_i(\xi)\ge 1$ и необходимо рассмотреть $i_k(\xi - a_l - a_i)$.

Эти же алгоритмы можно применять и для задачи о рюкзаке с ограничением типа равенства, полагая $F_k(\xi)=-\infty$, если соответствующая задача не имеет допустимых решений.

Пример 1.5. Рассмотрим следующую задачу:

$$5x_1 + 4x_2 + 8x_3 + 2x_4 \to \max$$
$$2x_1 + 4x_2 + 3x_3 + x_4 = 7,$$
$$x_1,\ x_2,\ x_3,\ x_4 \ge 0 \text{ и целые.}$$

Применим рекуррентные соотношения (1.17).

$k=1$. Значения функции Беллмана $F_1(\xi)$ определены только для четных значений ξ и равны $5\xi/2$, $i_1(\xi)=0$, если $F_1(\xi)=0$ и $i_1(\xi)=1$ в противном случае.

$k=2$. $F_2(\xi)=\max\{4+F_2(\xi-4);F_1(\xi)\}$ при $\xi\ge 4$, $F_2(\xi)=F_1(\xi)$, если $\xi<4$.

Таким образом,

$$F_2(4)=\max\{4+F_2(0);F_1(4)\}=\max\{4,10\}=10,$$

$$F_2(5)=\max\{4+F_2(1);F_1(5)\}=-\infty,\ F_2(6)=\max\{4+F_2(2);F_1(6)\}=12,$$

$$F_2(7)=\max\{4+F_2(3);F_1(7)\}=-\infty.$$

Таблица 1.6

Пример 1.5

ξ	$F_1(\xi)$	$i_1(\xi)$	$F_2(\xi)$	$i_2(\xi)$	$F_3(\xi)$	$i_3(\xi)$	$F_4(\xi)$	$i_4(\xi)$
0	0	0	0	0	0	0	0	0
1	-∞	-	-∞	-	-∞	-	2	4
2	5	1	5	1	5	1	5	1
3	-∞	-	-∞	-	8	3	8	3
4	10	1	10	1	10	1	10	1 или 4
5	-∞	-	-∞	-	13	3	13	3
6	12	1	12	1	16	3	16	3
7	-∞	-	-∞	-	18	3	18	3 или 4

Продолжаем вычисления аналогично.

$k=3$. $F_3(\xi)=\max\{8+F_3(\xi-3);F_2(\xi)\}$ при $\xi\ge 3$, $F_3(\xi)=F_2(\xi)$, если $\xi<3$.

$k=4$. $F_4(\xi)=\max\{2+F_4(\xi-1);F_3(\xi)\}$ при $\xi\ge 1$.

Результаты вычислений заносим в таблицу 1.6.

Возьмем $i_4=4$. Так как $i_4(6)=3$, $x_4=1$, $x_3\ge 1$. Рассмотрим $i_3(3)=3$, далее – $i_3(0)=0$, то есть $x_3=2$, $x_1=x_2=0$. Если $i_4=3$, то $x_4=0$, $i_3(4)=1$, то есть $x_3=1$, $x_2=0$, и, так как $i_1(2)=1$, $i_1(0)=0$, $x_1=2$.

Задача о бинарном рюкзаке – это задача о рюкзаке, в которой $d=1$. Рекуррентное соотношение (1.15) принимает вид

$$F_k(\xi)=\max_{x_k\in\{0,1\}}\{c_k x_k+F_{k-1}(\xi-a_k x_k)\},\ \text{если}\ \xi\ge a_k,$$

$$F_k(\xi)=F_{k-1}(\xi),\ \text{если}\ \xi<a_k. \qquad (1.18)$$

Это соотношение, очевидно, совпадает с (1.17): если $x_k=0$, то $F_k(\xi)=F_{k-1}(\xi)$, и $F_k(\xi)=c_k+F_k(\xi-a_k)$ при $x_k=1$, но, в отличие от целочисленной задачи о рюкзаке, возможно непосредственное определение управляющих переменных.

Пример 1.6. Решим следующую задачу:

$$5x_1+3x_2+4x_3+4x_4\to\max$$

$$2x_1+2x_2+x_3+2x_4\le 4,$$

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

Полагаем $F_0(\xi)=0$, $\xi=0,...,4$, $F_1(0)=0$, $F_1(1)=0$, $F_1(2)=F_1(3)=F_1(4)=5$. Далее $F_2(\xi)=\max\{3+F_1(\xi-2),F_1(\xi)\}$ при $\xi\ge 2$, $F_2(1)=F_1(1)$,

$$F_3(\xi)=\max\{4+F_2(\xi-1),F_2(\xi)\},\ \xi\ge 1,$$

$$F_4(\xi)=\max\{4+F_3(\xi-2),F_3(\xi)\},\ \xi\ge 2,\ F_4(1)=F_3(1).$$

Результаты вычислений заносим в таблицу 1.7.

Оптимальное значение целевой функции равно 9.

При $x_4=0$ $x_3=\hat{x}_3(4)=1$, $x_2=\hat{x}_2(4-1)=0$, $x_1=\hat{x}_1(3)=1$. Альтернативное решение – $(1,0,0,1)$.

Таблица 1.7

Пример 1.6

ξ	$F_1(\xi)$	$\hat{x}_1(\xi)$	$F_2(\xi)$	$\hat{x}_2(\xi)$	$F_3(\xi)$	$\hat{x}_3(\xi)$	$F_4(\xi)$	$\hat{x}_4(\xi)$
0	0	0	0	0	0	0	0	0
1	0	0	0	0	4	1	0	0
2	5	1	5	0	5	0	5	0
3	5	1	5	0	9	1	9	0
4	5	1	8	1	9	1	9	0 или 1

2. ЭВРИСТИЧЕСКИЕ АЛГОРИТМЫ

Время и вычислительные ресурсы, которые требуются для решения многих имеющих прикладное значение задач дискретной оптимизации, резко (экспоненциально) увеличиваются с ростом размера задачи. Таким образом, на практике получение оптимального решения задачи может оказаться неоправданно трудоёмким. В этом случае важнее найти пусть и не самое лучшее решение, но сравнительно быстро и просто. Кроме того, ценность оптимального решения снижается при наличии погрешностей в исходных данных задачи, да и сама математическая модель является, как правило, только приближенным описанием изначальной проблемы.

Получение хороших допустимых решений задачи дискретной оптимизации имеет не только практическое значение, но и является важным элементом многих методов поиска оптимального решения, например, метода ветвей и границ.

При численном решении выпуклых задач математического программирования, как правило, на каждом шаге алгоритма определяется некоторое допустимое решение задачи, отклонение которого от оптимального можно оценить и остановить алгоритм при достижении желаемой точности. В случае дискретного программирования задача поиска приближенного решения, обладающего заданной погрешностью, например такого $x \in D$, что $f(x) - f(x^*) < \varepsilon$, где x^* – оптимальное решение, может оказаться столь же сложной, как и исходная задача.

Приближенные алгоритмы решения задач дискретной оптимизации являются, в основном, *эвристическими* (от греч. heuristiko — отыскиваю, открываю), то есть в их основе лежит анализ структуры задачи, предположения о свойствах оптимального решения, часто не имеющие строгого формального обоснования.

Обычно не представляется возможным определить, насколько решение, полученное применением эвристики, отличается от оптимального. Возникает, следовательно, проблема оценки эффективности приближенных алгоритмов. Традиционно используются следующие методы [15].

Эмпирические сравнения. При этом подходе сравнивается работа различных алгоритмов на некотором множестве задач. В качестве критериев сравнения могут выступать качество полученного решения, время вычисления, трудоёмкость. Указанный способ оценки алгоритмов используется, например, при разработке методов решения той или иной задачи для обоснования того, что новый алгоритм лучше уже известных. Основной проблемой подхода является, разумеется, выбор репрезентативного множества тестовых задач. Так как эвристика может хорошо работать на одном наборе задач и плохо на другом, эмпирические сравнения помогают в построении или отборе алгоритмов, эффективных для решения задач, возникающих в определенной практической ситуации.

Анализ наихудшего случая. Иногда удаётся установить максимальное

отклонение (в терминах относительной погрешности) значения целевой функции задачи, достигаемое применением эвристики, от оптимального значения при любых входных данных. Алгоритмы, для которых возможно получить такой результат, называются алгоритмами гарантированного функционирования. Вообще говоря, эвристика может выполняться очень хорошо на большинстве задач, возникающих в приложениях, и чрезвычайно плохо в некоторых частных случаях. Следовательно, алгоритм с лучшей оценкой работы в наихудшем случае не обязательно более эффективен на практике.

Анализ среднего случая. Данный подход, применяемый при исследовании задач большого размера, заключается в определении ожидаемой относительной погрешности решения, полученного эвристикой, при некоторых предположениях о распределении входных данных задачи. Основная трудность в применении этого подхода на практике связана с установлением того, какие вероятностные предположения соответствуют конкретным реальным условиям.

Обычно для наиболее эффективных на практике алгоритмов хотя бы два из указанных подходов приводят к хорошим показателям.

2.1. Жадные алгоритмы

Под алгоритмами типа “greedy” (жадными, пожирающими) понимаются эвристики, в которых на каждом этапе решения задачи осуществляется действие, оптимальное для этого этапа. То есть оптимизация целевой функции заменяется локальной оптимизацией на каждом шаге алгоритма. Приведём несколько примеров.

Задача о рюкзаке

Рассмотрим задачу

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

$$\sum_{j=1}^{n} a_j x_j \le b,$$

$$x_j \ge 0 \text{ и целые, } j = 1,...,n.$$

Критерием «выгодности» упаковки предмета типа j в рюкзак является отношение c_j / a_j. Предположим, что

$$c_1 / a_1 \ge c_2 / a_2 \ge ... \ge c_n / a_n. \qquad (2.1)$$

Тогда жадная стратегия состоит в том, чтобы взять максимально возможное количество предметов первого типа, то есть $x_1 = \lfloor b / a_1 \rfloor$, затем, если $b - a_1 x_1 > 0$ (в рюкзаке осталось место), упаковать максимально возможное количество предметов второго типа, то есть $x_2 = \lfloor ((b - a_1 x_1)/ a_2 \rfloor$, и так далее.

Пример 2.1. Рассмотрим задачу из примера 1.4:

$$4x_1 + 2x_2 + 3x_3 \to \max$$

$$3x_1 + 2x_2 + 2x_3 \le 7,$$
$$x_j \ge 0 \text{ и целые, } j = 1,2,3.$$

Имеем $c_1/a_1 = 4/3$, $c_2/a_2 = 1$, $c_3/a_3 = 3/2$. На основании жадной стратегии, $x_3 = 3$, $x_1 = x_2 = 0$, значение целевой функции равно 9.

Пример 2.2. Рассмотрим задачу из примера 1.5:

$$5x_1 + 4x_2 + 8x_3 + 2x_4 \to \max$$
$$2x_1 + 4x_2 + 3x_3 + x_4 = 7,$$
$$x_1,\ x_2,\ x_3,\ x_4 \ge 0 \text{ и целые.}$$

Находим, что $\max\{c_j/a_j\} = c_3/a_3 = 8/3$. Полагаем $x_3 = 2$, и, так как $7 - 3x_3 = 1$, можно взять $x_4 = 1$, $x_1 = x_2 = 0$. Полученное решение оптимально.

Справедлива следующая теорема [11].

Теорема 2.1. *Пусть x^* – оптимальное решение задачи о рюкзаке, x' – решение, полученное применением жадного алгоритма. Тогда*

$$\frac{\langle c, x^* \rangle - \langle c, x' \rangle}{\langle c, x^* \rangle} \le 0{,}5.$$

Доказательство. Без ограничения общности можно считать, что выполнено условие (2.1). Так как $x_1' = [b/a_1]$,

$$c_1\left[\frac{b}{a_1}\right] \le \langle c, x' \rangle \le \langle c, x^* \rangle = \sum_{j=1}^{n} c_j x_j^* = \sum_{j=1}^{n} \frac{c_j}{a_j} a_j x_j^* \le \frac{c_1}{a_1} b.$$

Если $a_1 = b$, то x' – оптимальное решение. Пусть $a_1 < b$. Тогда $[b/a_1] \ge 1$, $b/a_1 < [b/a_1] + 1 \le 2[b/a_1]$. Таким образом,

$$\frac{\langle c, x' \rangle}{\langle c, x^* \rangle} \ge \frac{[b/a_1]}{b/a_1} > \frac{1}{2},$$

откуда и следует доказываемая оценка.

Для задачи о бинарном рюкзаке

$$\sum_{j=1}^{n} c_j x_j \to \max$$
$$\sum_{j=1}^{n} a_j x_j \le b,$$
$$x_j \in \{0,1\},\ j = 1,\ldots,n.$$

жадным является, например, алгоритм, основанный на правиле Данцига [11]. Пусть выполнено условие (2.1). Тогда $x_1 = 1$, обозначаем $b_1 = b - a_1$ и последовательно для $k = 2,\ldots,n$ полагаем $x_k = 1$, если $a_k \le b_{k-1}$ и $x_k = 0$ в противном случае, $b_k = b_{k-1} - a_k x_k$.

Применяя к задаче о бинарном рюкзаке рассуждения из теоремы 2.1 получаем, что при выполнении (2.1) справедлива оценка

$$\frac{\langle c, x^* \rangle - \langle c, x' \rangle}{\langle c, x^* \rangle} \le 1 - \frac{a_1}{b}.$$

Пример 2.3. Рассмотрим задачу из примера 1.6:

$$5x_1 + 3x_2 + 4x_3 + 4x_4 \to \max$$
$$2x_1 + 2x_2 + x_3 + 2x_4 \le 4,$$
$$x_i \in \{0,1\},\ i = 1,...,4.$$

Находим, что $c_1 / a_1 = 2{,}5$, $c_2 / a_2 = 1{,}5$, $c_3 / a_3 = 4$, $c_4 / a_4 = 2$. Таким образом, получаем оптимальное решение $(1,0,1,0)$.

Минимальное остовное дерево

Для некоторых классов задач дискретной оптимизации жадные алгоритмы всегда гарантируют получение оптимального решения. Пусть $G = (V, E)$ – связный неориентированный граф, $|V| = n$, длины ребер заданы матрицей C с элементами c_{ij}. Рассмотрим задачу поиска минимального остовного дерева графа G, то есть связного ациклического подграфа, содержащего все вершины графа, имеющего наименьший вес.

Лемма 2.1. *Любые два из следующих трёх условий определяют остовное дерево* $T = (V, E')$*:*

– граф T *связный;*

– граф T *не содержит циклов;*

– $|E'| = |V| - 1$.

Сформулируем задачу о минимальном остовном дереве как задачу дискретной оптимизации, используя первое и третье свойства. Пусть переменная x_e принимает значение 1, если ребро $e \in E$ включается в дерево и 0 в противном случае. Для $S \subseteq V$ обозначим через $E(S)$ множество ребер, обе конечные вершины которых принадлежат S. Пусть c_e – стоимость (длина) ребра $e \in E$. Получаем задачу

$$\sum_{e \in E} c_e x_e \to \min \tag{2.2}$$
$$\sum_{e \in E} x_e = n - 1, \tag{2.3}$$
$$\sum_{e \in E(S)} x_e \le |S| - 1,\ S \subseteq V,\ S \ne \varnothing, \tag{2.4}$$
$$x_e \in \{0,1\},\ e \in E. \tag{2.5}$$

Жадная стратегия для решения поставленной задачи заключается в присоединении на каждом шаге к уже найденной части дерева кратчайшего возможного ребра. Оптимальность такого подхода вытекает из следующих утверждений.

Лемма 2.2. *Пусть i — произвольная вершина графа, ребро $[i, j]$ – кратчайшее из всех инцидентных i. Тогда существует минимальное остовное дерево, содержащее ребро $[i, j]$.*

Доказательство. Пусть T — минимальное остовное дерево, ребро $[i, j]$ не принадлежит T. В дереве существует единственный путь из j в i, допустим, он проходит через некоторую смежную i вершину k. Заменив ребро $[i,k]$ на $[i, j]$ снова получим остовное дерево, поскольку количество ребер не изменилось, вершины i и k соединены путём, проходящим через вершину j. Если $c_{ij} < c_{ik}$, то построенное дерево имеет меньший вес, чем T, что противоречит минимальности T. Поэтому рассматриваемая ситуация возможна только если $c_{ij} = c_{ik}$, вес нового дерева равен весу дерева T и, следовательно, минимален.

Лемма 2.3. *Пусть F – поддерево минимального остовного дерева. Существует минимальное остовное дерево, содержащее F вместе с кратчайшим ребром из соединяющих вершину из F и вершину, не принадлежащую F.*

Доказательство повторяет доказательство леммы 2.2 с заменой i на F.

Алгоритм Прима построения минимального остовного дерева начинает работу с кратчайшего ребра, выходящего из произвольной вершины графа, скажем, $[i, j]$. На основании леммы 2.3 к нему присоединяется кратчайшее из исходящих из вершин i и j ребер, и так далее. Для поиска кратчайших рёбер на каждом шаге алгоритма вершины графа получают метки – временные, равные длине ребра, соединяющего вершину с вершиной уже найденного поддерева, и постоянные, равные длине кратчайшего такого ребра.

Начальный шаг. Пусть алгоритм начинается с вершины 1, она получает постоянную метку $l_1^* = 0$, а все остальные вершины – временные метки $l_j = c_{1j}$.

Общий шаг. Среди всех временных меток выбирается одна с наименьшим значением, например, l_j и делается постоянной. К дереву присоединяется ребро $[i, j]$ такое, что вершина i имеет постоянную метку и $c_{ij} = l_j^*$. Временные метки соседей вершины j пересчитываются по формуле

$$l_k = \min\{l_k, c_{jk}\}.$$

Матричная форма алгоритма Прима

В матрице длин дуг вычеркнуть первый столбец, пометить первую строку. На шаге k $(k = 1,...,n-1)$ выбрать минимальный элемент среди всех невычеркнутых элементов в помеченных строках, пусть это c_{ij}. Вычеркнуть столбец j и пометить строку j.

Пример 2.4. Найдём минимальное остовное дерево для графа на рисунке 1.4. Матричная форма алгоритма Прима реализована на рис. 2.1, метки строк указывают порядок, в котором они присваивались. Итоговое дерево изображено на рис. 2.2, его вес равен 28. Заметим, что кратчайшие пути из одной вершины

во все остальные также образуют остовное дерево. В данном примере (рис. 1.6) вес этого дерева равен 40.

	1	2	3	4	5	6	7	
1	0	10	12	–	–	–	–	1
2	10	0	3	4	–	–	–	2
3	12	3	0	–	6	14	–	3
4	–	4	–	0	4	–	2	4
5	–	–	6	4	0	5	–	6
6	–	–	14	–	5	0	6	7
7	–	–	–	2	–	6	0	5

Рис. 2.1. Матрица длин дуг для примера 2.4

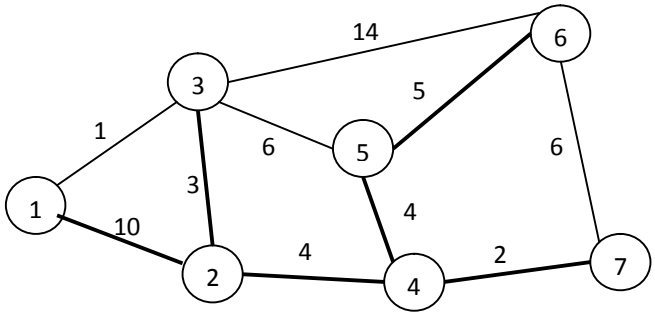


Рис.2.2. Минимальное остовное дерево

1-дерево минимальной стоимости

1-деревом в неориентированном графе называется связный остовный подграф, состоящий из дерева на множестве вершин $\{2,...,n\}$ и двух ребер, инцидентных вершине 1. Таким образом, 1-дерево содержит точно один цикл, проходящий через вершину 1.

Вес 1-дерева определяется как сумма стоимостей всех его ребер.

Для $S \subseteq V$ обозначим через $\delta(S)$ множество ребер, одна из вершин которых принадлежит множеству S, а вторая – нет. Формулировка задачи поиска 1-дерева минимальной стоимости, аналогичная (2.2)-(2.5), имеет вид

$$\sum_{e \in E} c_e x_e \to \min \tag{2.6}$$

$$\sum_{e \in E} x_e = n, \tag{2.7}$$

$$\sum_{e \in \delta(1)} x_e = 2, \tag{2.8}$$

$$\sum_{e \in E(S)} x_e \le |S| - 1,\ S \subseteq V \setminus \{1\},\ S \ne \varnothing, \tag{2.9}$$

$$x_e \in \{0,1\},\ e \in E. \tag{2.10}$$

Для построения кратчайшего 1-дерева нужно построить остовное дерево минимального веса на вершинах $\{2,...,n\}$ и добавить к нему два кратчайших ребра, инцидентных вершине 1.

Задача коммивояжера

Жадными алгоритмами для задачи коммивояжера являются алгоритмы перехода в ближайшую точку, в которых на каждом шаге к найденному частичному маршруту присоединяется самая короткая из еще не выбранных дуг. Например, начиная с произвольной вершины i_1, определяют вершину i_2 из условия $c_{i_1,i_2} = \min\limits_j c_{i_1 j}$, вершину i_3 – из условия $c_{i_2,i_3} = \min\limits_{j \neq i_1} c_{i_2 j}$, и так далее. Вершина i_n определяется при этом однозначно, дуга (i_n, i_1) замыкает маршрут.

Улучшить результат работы указанного алгоритма можно, применяя его несколько раз, выбирая различные начальные вершины в качестве начальной.

Третья модификация алгоритма следующая. Алгоритм начинает работы с кратчайшей дуги графа. Далее, если $(i_1,...,i_k)$ – частичный маршрут, к нему добавляется кратчайшая из дуг $c_{i,i_1} = \min\limits_s c_{si_1}$ и $c_{i_k,j} = \min\limits_s c_{i_k s}$.

Решение, порожденное жадным алгоритмом, может сколь угодно сильно отличаться от оптимального.

Пример 2.5. Рассмотрим задачу коммивояжера с расстояниями между вершинами

$$c_{ij} = \begin{cases} q, & i = 1,...,n-1, j = i+1, \\ A, & i = n, j = 1, \\ p, & \textit{в остальных случаях}, \end{cases}$$

где $q < p < A$.

Алгоритм перехода в ближайшую точку, начинающий работу с первой вершины, порождает маршрут $z = (1,2,...,n,1)$ длиной $l(z) = q(n-1) + A$. Оптимальный маршрут $z^* = (1,2,...,n-2,n,n-1,1)$ имеет длину $q(n-3) + 3p$.

Оценим погрешность полученного решения.

$$\frac{l(z) - l(z^*)}{l(z^*)} = \frac{2q + A - 3p}{(n-3)q + 3p} = \frac{A - q - 3(p-q)}{nq + 3(p-q)} < \frac{A-q}{nq}.$$

С другой стороны,

$$\frac{l(z) - l(z^*)}{l(z^*)} = \frac{2q + A - 3p}{(n-3)q + 3p} = \frac{A - p - 2(p-q)}{np + (n-3)(q-p)} > \frac{A-p}{np} - \frac{2(p-q)}{np} > \frac{A-p}{np} - 1.$$

Таким образом,

$$\frac{A-p}{np} - 1 < \frac{l(z) - l(z^*)}{l(z^*)} < \frac{A-q}{nq},$$

то есть погрешность неограниченно растет при $A \to \infty$.

2.2. Алгоритмы гарантированного функционирования

Целью анализа наихудшего случая для эвристического алгоритма является установление максимального отклонения от оптимальности, которое может возникнуть при применении этого алгоритма. Эвристики, для которых возможно оценить это отклонение, называются *алгоритмами гарантированного функционирования*.

Рассмотрим задачу оптимизации, определяемую набором входных данных I. Обозначим через $f^*(I)$ стоимость оптимального решения и пусть $f^H(I)$ – стоимость решения, полученного применением эвристики H. Абсолютная оценка для H определяется как

$$R^H = \inf\left\{ r \ge 1 : \frac{f^H(I)}{f^*(I)} \le r \text{ для всех } I \right\}.$$

Наличие такой оценки означает, что стоимость решения, порожденного эвристикой, даже в худшем случае не более чем в R^H раз превосходит минимальную стоимость.

Для задач большого размера используется также асимптотическая оценка, определяемая как

$$R_\infty^H = \inf\left\{ r \ge 1 : \exists n \text{ такое что } \frac{f^H(I)}{f^*(I)} \le r \text{ для всех } I \text{ с } f^*(I) \ge n \right\}.$$

Эта величина иногда дает более точную картину качества эвристики. Заметим, что $R_\infty^H \le R^H$.

Предположим, алгоритм H является алгоритмом гарантированного функционирования, то есть $R^H < \infty$. Положим $\varepsilon = R^H - 1 > 0$. Тогда для всех наборов данных I относительная погрешность работы алгоритма

$$\frac{f^H(I) - f^*(I)}{f^*(I)} \le \varepsilon .$$

Эвристика H называется в этом случае ε-оптимальной.

Например, теорема 2.1 означает, что жадный алгоритм для решения задачи о рюкзаке является ε-оптимальным при $\varepsilon = 0{,}5$.

Иногда работа алгоритма оценивается не для всех возможных наборов данных, а только для некоторого их семейства. Например, алгоритм, основанный на правиле Данцига для задачи о бинарном рюкзаке, имеет, в общем случае, оценку гарантированного функционирования равную 2. Для задач же, удовлетворяющих условию

$$\frac{a_j}{b} \ge \eta > 0, \ \frac{c_j}{a_j} = \max_i \frac{c_i}{a_i},$$

этот алгоритм является ε-оптимальным при $\varepsilon = 1 - \eta$.

Рассмотрим далее примеры алгоритмов гарантированного функционирования для двух задач дискретной оптимизации – задачи об упаковке и задачи коммивояжера.

Задача об упаковке

Пусть задан набор n действительных чисел $L=(w_1,...,w_n)$, где $w_i \in (0,1]$ интерпретируется как размер предмета i. Требуется разбить множество L на подмножества, так, чтобы сумма элементов каждого подмножества не превосходила 1 (разместить предметы по контейнерам единичной вместимости) и количество подмножеств (использованных контейнеров) было минимальным.

Можно считать, что имеется n контейнеров. Содержимым контейнера называется сумма упакованных в него предметов. Если c_j – содержимое контейнера j, то свободное пространство, равное $1-c_j$, называется резервом контейнера.

Минимальное количество контейнеров, необходимое для упаковки предметов из списка L, то есть оптимальное значение целевой функции задачи об упаковке, определенной списком L, будет обозначаться $b^*(L)$. Аналогично, через $b^H(L)$ обозначается количество контейнеров, требуемых при применении к списку L эвристики H.

Алгоритмы решения задачи об упаковке можно разделить на так называемые *онлайн алгоритмы*, или алгоритмы реального времени, и *оффлайн алгоритмы*. Эвристики первой группы упаковывают предметы в порядке их появления в списке, решение об упаковке каждого предмета принимается до того, как станет известной следующая компонента списка.

Простейшим онлайн алгоритмом для задачи об упаковке является алгоритм «следующего подходящего» (Next-Fit algorithm). Первый предмет помещается в первый контейнер. Далее, на шаге $i=2,...,n$ предмет i упаковывается в последний непустой контейнер j, если $w_i \le 1-c_j$, и в контейнер $j+1$ в противном случае.

Теорема 2.2. *Для любых* L

$$\frac{b^{NF}(L)}{b^*(L)} \le 2. \qquad (2.11)$$

Доказательство. Пусть $b^{NF}(L)=m$. Очевидно, $c_j + c_{j+1} > 1$ для всех $j=1,...,m-1$, поэтому

$$c_1 + 2(c_2 + ... + c_{m-1}) + c_m > m-1 ,$$

то есть суммарное содержимое контейнеров $c_1 + ... + c_m > (m-1)/2$. Таким образом, $b^*(L) > (m-1)/2$ и, следовательно, $m < 2b^*(L)+1$,

$$\frac{b^{NF}(L)}{b^*(L)} = \frac{m}{b^*(L)} \le \frac{2b^*(L)}{b^*(L)} = 2 .$$

Для доказательства точности оценки (2.11) рассмотрим следующий пример.

Пример 2.6. Пусть $n = 4k - 1$,

$$L = \left(\frac{1}{2}, \frac{1}{2k}, \frac{1}{2}, \frac{1}{2k}, \ldots, \frac{1}{2} \right).$$

Таким образом, список содержит $2k$ предметов размера 0,5, для упаковки которых необходимо k контейнеров, и $2k - 1$ предмет размера $1/(2k)$, которые помещаются в один контейнер. Следовательно, $b^*(L) = k + 1$. При применении NF-алгоритма в каждый контейнер кроме последнего упаковывается по два предмета, поэтому $b^{NF}(L) = 2k - 1 + 1 = 2k$,

$$\frac{b^{NF}(L)}{b^*(L)} = \frac{2k}{k+1} = 2 - \frac{2}{k+1}.$$

Увеличивая k получаем, что можно построить пример задачи об упаковке, для которой рассматриваемое отношение сколь угодно близко к 2.

Самыми популярными онлайн алгоритмами для задачи об упаковке являются алгоритмы «первого подходящего» (First-Fit rule) и «наилучшего подходящего» (Best-Fit rule). Согласно правилу «первого подходящего» очередной предмет i помещается в контейнер, имеющий резерв не менее w_i, с наименьшим номером. Алгоритм «наилучшего подходящего» упаковывает предмет i с наибольшим из не превышающих $1 - w_j$ содержимым.

Установлено [15], что для задач большого размера

$$b^{FF}(L) \le \left[\frac{17}{10} b^*(L) \right], \; b^{BF}(L) \le \left[\frac{17}{10} b^*(L) \right],$$

причём можно привести пример задач, для которых отношения $b^{FF}(L)/b^*(L)$ и $b^{BF}(L)/b^*(L)$ сколь угодно близки к 1,7.

Оффлайн алгоритм «первого подходящего в порядке убывания» (First-Fit-Decreasing rule) сначала упорядочивает элементы списка по убыванию, затем применяется алгоритм «первого подходящего». Аналогично, правило «наилучшего подходящего в порядке убывания» состоит в применении к упорядоченному списку алгоритма «наилучшего подходящего». Для этих алгоритмов справедливы асимптотические оценки

$$b^{FFD}(L) \le \frac{11}{9} b^*(L) + 3, \; b^{BFD}(L) \le \frac{11}{9} b^*(L) + 3.$$

Пусть XF обозначает либо FF, либо BF, а пусть XFD – либо FFD, либо BFD. В [15] доказан следующий результат.

Теорема 2.3. *Для всех* L

$$\frac{b^{XF}(L)}{b^*(L)} \le \frac{7}{4}, \; \frac{b^{XFD}(L)}{b^*(L)} \le \frac{3}{2}.$$

Рассмотрим алгоритм реального времени, называемый улучшенным правилом «первого подходящего». Элементы списка L классифицируются в

соответствии с их величиной следующим образом. Предмет i называется предметом типа α, если $0{,}5 < w_i \le 1$, предметом типа β_1, если $0{,}4 < w_i \le 0{,}5$, предметом типа β_2, если $1/3 < w_i \le 0{,}4$ и, наконец, предметом типа δ, если $0 < w_i \le 1/3$.

Контейнеры разделяются на четыре класса. Предметы типа α упаковываются в контейнеры первого класса, предметы типа β_1 – в контейнеры второго класса, предметы типа δ – в контейнеры четвёртого класса. Предметы типа β_2 упаковываются в контейнеры первого и третьего класса, причём в контейнер первого класса может быть упаковано не более одного предмета типа β_2.

Размещение предметов по контейнерам соответствующих типов производится по правилу «первого подходящего».

Теорема 2.4.

$$b^{RFF}(L) \le \frac{5}{3} b^*(L) + 5. \qquad (2.12)$$

Доказательство теоремы приведено в [14].

Пример 2.7. Рассмотрим список, состоящий из $n = 6t + 1$ предметов типа α с размерами $a_j = 0{,}5 + \varepsilon_j$, где $\varepsilon_j = 4^{-(j+2)}$ и $2n$ предметов типа δ с размерами $v_j = 0{,}25 + \varepsilon_j$ и $u_j = 0{,}25 - 2\varepsilon_j$. Список L получен присоединением к списку L_1 списка L_2, где

$$L_1 = (v_1, u_2, u_3, v_3, u_4, u_5, ..., v_{2j-1}, u_{2j}, u_{2j+1}, ..., v_{n-2}, u_{n-1}, u_n),$$

$$L_1 = (v_2, v_4, ..., v_{n-1}, a_1, ..., a_n, u_1, v_n).$$

Поскольку $a_j + v_j + u_j = 1$, $b^*(L) = n$. Указанный алгоритм заполняет сначала контейнеры четвертого типа. При упаковке предметов из списка L_1 формируются контейнеры, содержащие предметы v_{2j-1}, u_{2j}, u_{2j+1}, где $j = 1, ..., (n-1)/2$. Далее, в один контейнер умещаются по три предмета с размерами v_{2j}, $j = 1, ..., (n-1)/2$, всего таких контейнеров $(n-1)/6$. После заполняются n контейнеров первого типа, предмет u_1 может быть помещён в первый контейнер, предмет v_n занимает отдельный контейнер. Таким образом,

$$b^{RFF}(L) = \frac{n-1}{2} + \frac{n-1}{6} + n + 1 = \frac{5}{3} n + \frac{1}{3} = \frac{5}{3} b^*(L) + \frac{1}{3}.$$

Пример 2.8. Применим рассмотренные алгоритмы к задаче об упаковке со списком $L = (0{,}1,\ 0{,}1,\ 0{,}6,\ 0{,}3,\ 0{,}4,\ 0{,}7,\ 0{,}7,\ 0{,}3,\ 0{,}8)$.

Алгоритм «следующего подходящего» требует 5 контейнеров:

$$(0{,}1,\ 0{,}1,\ 0{,}6),\ (0{,}3,\ 0{,}4),\ (0{,}7),\ (0{,}7,\ 0{,}3),\ (0{,}8).$$

Правила «первого подходящего» и «наилучшего подходящего» порождают решение

$$(0{,}1,\ 0{,}1,\ 0{,}6),\ (0{,}3,\ 0{,}4,\ 0{,}3),\ (0{,}7),\ (0{,}7),\ (0{,}8),$$

а улучшенное правило «первого подходящего» – решение

$$(0{,}1,\ 0{,}1,\ 0{,}3,\ 0{,}3),\ (0{,}6,\ 0{,}4),\ (0{,}7),\ (0{,}7),\ (0{,}8).$$

Алгоритмы «первого подходящего в порядке убывания» и «наилучшего подходящего в порядке убывания» формируют оптимальное решение задачи, использующее 4 контейнера:

$$(0{,}8,\ 0{,}1,\ 0{,}1),\ (0{,}7,\ 0{,}3),\ (0{,}7,\ 0{,}3),\ (0{,}6,\ 0{,}4).$$

Задача коммивояжера

Рассмотрим симметричную задачу коммивояжера, матрица длин дуг в которой удовлетворяет неравенству треугольника, то есть

$$c_{ij} \le c_{ik} + c_{kj} \quad \text{для всех попарно различных } i,\ j,\ k.$$

Вначале приведем два приближенных алгоритма для симметричной задачи коммивояжера, основанных на построении эйлерова остовного графа.

Пусть имеется некоторый мультиграф (то есть граф, в котором допускаются кратные ребра) со множеством вершин V.

Эйлеровым маршрутом в мультиграфе называется цикл, содержащий каждое ребро графа точно один раз.

Мультиграф называется *эйлеровым*, если содержит эйлеров маршрут.

Поскольку эйлеров путь проходит через каждую вершину графа, то он является допустимым для общей задачи коммивояжера (задачи, в которой разрешается посещать вершины графа более одного раза). Для того, чтобы преобразовать эйлеров цикл в гамильтонов, применяется следующая процедура, которая называется *спрямлением*: нужно просматривать цикл, начиная с некоторой вершины, и если некоторая вершина встречается повторно, пропускать её, переходя к следующей непосещённой вершине. Полученный гамильтонов цикл называется *вложенным* в эйлеров маршрут. При этом, если матрица длин ребер удовлетворяет неравенству треугольника, длина полученного цикла не превышает длины эйлерова цикла.

Для приближенного решения задачи коммивояжера на основании указанного подхода требуется, прежде всего, построить эйлеров мультиграф.

Теорема 2.5 (теорема Эйлера). *Мультиграф $G = (V, E)$ эйлеров в том и только в том случае, если он связен и каждая вершина из V имеет четную степень*.

Доказательство. Необходимость вытекает из замкнутости маршрута. Доказательство достаточности проводится индукцией по числу ребер в G. Если граф состоит из одной вершины, утверждение тривиально. Предположим теперь, что G удовлетворяет условиям теоремы и, кроме того, все мультиграфы с числом ребер меньшим, чем в G, удовлетворяющие условиям теоремы, эйлеровы. Поскольку из каждой вершины графа выходит, по крайней мере, два ребра, граф содержит циклы. Найдем один из циклов, не содержащих кратных ребер, и удалим его ребра из графа. Так как степень произвольной вершины либо не меняется, либо уменьшается на 2, получаем некоторое число связных компонент, удовлетворяющих условиям теоремы и, следовательно, по предположению индукции, являющихся эйлеровыми графами. Эйлеров маршрут в графе

G можно получить теперь, присоединяя эйлеровы маршруты компонент к исходному маршруту.

Для построения эйлерова маршрута можно использовать поиск в глубину, то есть начиная с произвольной вершины посещать другие вершины, не проходя по одному ребру дважды. Предположим, в процессе построения маршрута пришли в вершину, все ребра, инцидентные которой уже принадлежат маршруту. Если искомый маршрут не найден, обозначим пройденный путь через G', $G'' = G \setminus G'$. В силу связности исходного графа G, G' и G'' имеют хотя бы одну общую вершину. Найдем цикл в G'', проходящий через эту вершину и присоединим его к G'. Действуя таким образом, можно добиться, чтобы $G' = G$.

Вернёмся к задаче коммивояжера в графе с множеством вершин V. Наименьшим по количеству ребер связным подграфом графа является остовное дерево. Так как дерево не содержит циклов, в нем найдутся вершины с нечетными степенями. Простейшим способом преобразовать остовное дерево в остовный эйлеров мультиграф является удвоение всех ребер дерева. Получаем, тем самым, алгоритм приближенного решения задачи коммивояжера, называемый алгоритмом дерева.

Алгоритм дерева

1. В графе G найти минимальное остовное дерево T^*.
2. Построить мультиграф G', удвоив ребра из T^*.
3. Найти эйлеров маршрут в G' и вложенный в него маршрут коммивояжера.

Теорема 2.6. *Если матрица длин дуг C удовлетворяет неравенству треугольника, алгоритм дерева является 1-оптимальным алгоритмом для задачи коммивояжера.*

Доказательство. Мультиграф G' эйлеров по построению. Пусть l^* – длина кратчайшего гамильтонова цикла, $w(T^*)$ – вес минимального остовного дерева, l^{MST} – длина маршрута коммивояжера, порожденного алгоритмом.

Поскольку любой маршрут коммивояжера можно преобразовать в дерево, выбрасывая произвольное ребро, $w(T^*) \le l^*$. Длина полученного эйлерова маршрута равна $2w(T^*)$. Таким образом, поскольку при выполнении неравенства треугольника процедура спрямления не увеличивает длины маршрута,

$$l^{MST} \le 2w(T^*) \le 2l^*,$$

$$\frac{l^{MST}}{l^*} \le 2,$$

то есть гарантированная оценка алгоритма не превышает 2. Из последнего неравенства вытекает

$$\frac{l^{MST} - l^*}{l^*} \le 1$$

и теорема доказана.

Для обоснования того, что полученная в теореме оценка не может быть улучшена, рассмотрим следующий пример.

Пример 2.9. Пусть граф имеет $3n$ вершины, расположенные на плоскости так, как показано на рис. 2.3, в качестве расстояний между вершинами используются евклидовы расстояния. На этом же рисунке изображено минимальное остовное дерево, его вес равен $w(T^*) = n + 1 + (n-2)(1-\varepsilon)$. Алгоритм дерева генерирует маршрут (рис. 2.4a) длиной

$$l^{MST} = 2 + n + (n-2)\left(2(1-\varepsilon) + \sqrt{\varepsilon^2 + (1-\varepsilon)^2}\right) \approx 2 + n + 3(n-2)(1-\varepsilon).$$

Длина оптимального маршрута коммивояжера (рис. 2.4b)

$$l^* = n + 1 + n - 2 + 2\sqrt{1+\varepsilon^2} \approx 2n + 1.$$

Таким образом, при $n \to \infty$ и $\varepsilon \to 0$ $\dfrac{l^{MST}}{l^*} \to 2$.

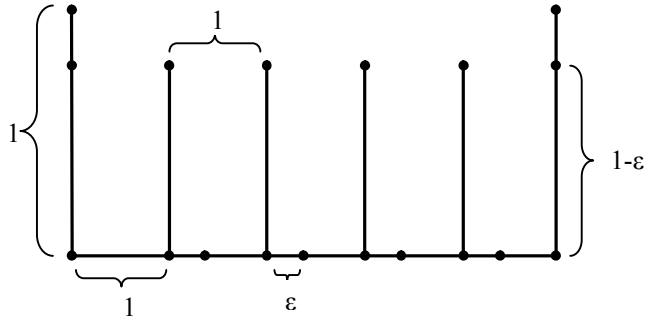
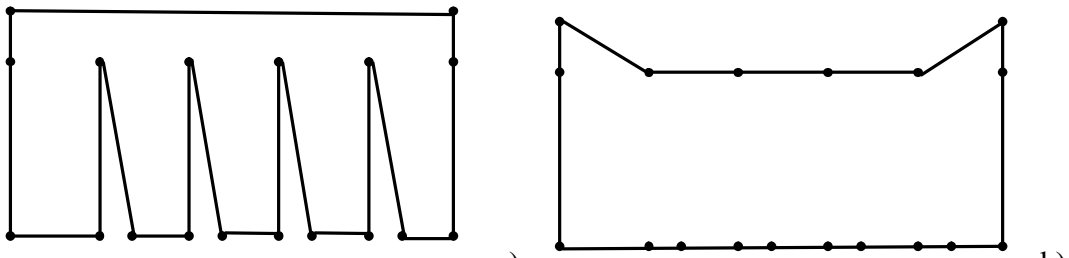


Рис.2.3. Минимальное остовное дерево



a) b)

Рис. 2.4. Маршруты коммивояжера

Второй алгоритм, основанный на построении эйлерова маршрута, разработан в 1976 Н. Кристофидесом и отличается от алгоритма дерева способом преобразования остовного дерева в эйлеров мультиграф.

Лемма 2.4. *Для данного графа с по крайней мере двумя вершинами число вершин нечетной степени четно.*

Совершенным паросочетанием в графе с четным числом вершин называется такое подмножество ребер, что каждая вершина графа является конечной для точно одного ребра в этом подмножестве. Добавление к остовному дереву ребер паросочетания полного графа, построенного на вершинах нечетной степени в дереве, увеличивает степень каждой из этих вершин на единицу и, следовательно, приводит к эйлерову мультиграфу.

Поскольку целью является минимизация стоимости маршрута, выбирается паросочетание, совокупная длина ребер которого минимальна, то есть кратчайшее совершенное паросочетание.

Алгоритм Кристофидеса

1. Найти минимальное остовное дерево T^*.

2. Выделить в T^* вершины нечетной степени и найти кратчайшее совершенное паросочетание M в полном графе, содержащем только эти вершины. Пусть G – мультиграф с множеством вершин V, в который входят все ребра из T^* и M.

3. Найти в G эйлеров маршрут и вложенный в него маршрут коммивояжера.

Пусть l^C – длина маршрута, порожденного эвристикой Кристофидеса.

Теорема 2.7. *Для всех входных данных задачи коммивояжера, удовлетворяющих неравенству треугольника,*

$$l^C \le \frac{3}{2} l^*,$$

то есть алгоритм Кристофидеса является 1/2–приближенным алгоритмом для задачи коммивояжера.

Доказательство.

По построению, граф G – эйлеров. Пусть, как и ранее, $w^* = w(T^*)$ – стоимость минимального остовного дерева. Обозначим через $w(M^*)$ вес кратчайшего паросочетания. Ввиду предположения о неравенстве треугольника,

$$l^C \le w(T^*) + w(M^*).$$

Обозначим вершины нечетной степени в минимальном остовном дереве i_1, $i_2, \ldots, i_{2k}$ согласно их появлению в оптимальном маршруте коммивояжера. Другими словами, $z^* = (\alpha_0 i_1 \alpha_1 i_2 \alpha_{2m-1} i_{2m} \alpha_{2m})$, где все α_k – последовательности (возможно, пустые) вершин из множества V. Рассмотрим два паросочетания на этих вершинах. Первое паросочетание, обозначенное M^1, состоит из ребер (i_1, i_2), $(i_3, i_4), \ldots, (i_{2k-1}, i_{2k})$. Второе паросочетание, M^2, состоит из ребер (i_2, i_3), (i_4, i_5), …, (i_{2k}, i_1).

Поскольку M^* – оптимальное паросочетание,

$$w(M^*) \le 1/2 \left[w(M^1) + w(M^2) \right].$$

По неравенству треугольника имеем

$$l^* \ge w(M^1) + w(M^2).$$

Из двух последних неравенств заключаем, $l^* \ge 2w(M^*)$. Поэтому, так как $w(T^*) \le l^*$,

$$l^C \le w(T^*) + w(M^*) \le l^* + \frac{1}{2} l^* = \frac{3}{2} l^*, \quad \frac{l^C}{l^*} \le \frac{3}{2}.$$

Вычитая по единице из обеих частей, получим $\frac{l^C - l^*}{l^*} \le \frac{1}{2}$ и теорема доказана.

Аналогично алгоритму дерева, алгоритм Кристофидеса может асимптотически достигать своей оценки в худшем случае.

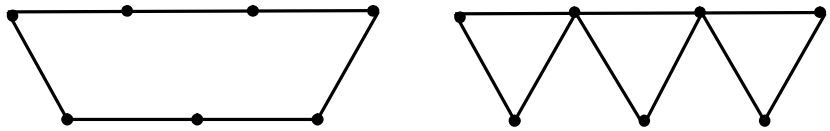


Рис. 2.5. Маршруты коммивояжера для примера 2.10

Пример 2.10. Рассмотрим граф с нечетным числом вершин, расположенных на плоскости так, как на рис. 2.5, расстояние между соседними вершинами равно 1. для которого $L^* = n$, в то время как $L^C = n - 1 + (n-1)/2$.

Пример 2.11. Применим рассмотренные алгоритмы для решения задачи коммивояжера с матрицей длин дуг

$$\begin{pmatrix} - & 28 & 31 & 22 & 36 \\ 28 & - & 40 & 41 & 32 \\ 31 & 40 & - & 14 & 38 \\ 22 & 41 & 14 & - & 24 \\ 36 & 32 & 38 & 24 & - \end{pmatrix}.$$

Минимальное остовное дерево изображено на рис. 2.6. Удваивая его ребра, строим эйлеров маршрут $(1,2,1,4,3,4,5,4,1)$. Таким образом, алгоритм дерева порождает маршрут коммивояжера $(1,2,4,3,5,1)$ длиной 157.

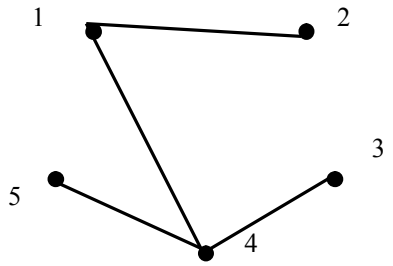


Рис. 2.6. Остовное дерево для примера 2.11

Вершины нечётной степени – 2, 3, 4, 5. Паросочетание минимального веса, построенное на этих вершинах состоит из ребер $[2,5]$ и $[3,4]$. Присоединяя их к дереву, получаем эйлеров маршрут $(1,2,5,4,3,4,1)$. Таким образом, алгоритм Кристофидеса строит маршрут коммивояжера $(1,2,5,4,3,1)$ длиной 129.

Следующий алгоритм – модификация алгоритма перехода в ближайшую точку. Как показывает пример 2.6, в общем случае жадный алгоритм для задачи коммивояжера не имеет гарантированной оценки. Более того, для любого n можно указать такую симметричную матрицу длин дуг порядка n, удовлетворяющую неравенству треугольника, что длина маршрута, порожденного

эвристикой перехода в ближайшую точку, в $O(\ln n)$ раз превышает длину оптимального маршрута.

Будем теперь рассматривать на каждой итерации не часть маршрута коммивояжера, а гамильтонов контур, содержащий некоторое подмножество вершин. Алгоритм включения ближайшего начинается с цикла, состоящего из одной произвольной вершины. Далее выбирается новая вершина, еще не содержащаяся в цикле, ближайшая к нему, и вставляется между двумя сопряженными вершинами цикла. Алгоритм завершает работу, когда все вершины охвачены циклом.

Эвристика включения ближайшего

1. Выбрать произвольную вершину v, цикл S состоит только из v.

2. Найти вершину k вне S, ближайшую к S, то есть $c_{ks} = \min\limits_{i \notin S,\, j \in S} c_{ij}$.

3. Найти ребро $[i, j]$ в S такое, что увеличение длины цикла при включении k между i и j, равное $c_{ik} + c_{kj} - c_{ij}$, минимально.

4. Построить новый цикл S, заменяя $[i, j]$ на $[i,k]$ и $[k, j]$.

5. Если текущий цикл S содержит все вершины, остановиться. Иначе идти на шаг 2.

Пусть l^{NI} – длина решения, полученного эвристикой включения ближайшего.

Теорема 2.8. *Для всех данных задачи коммивояжера, удовлетворяющих неравенству треугольника,*

$$l^{NI} \le 2l^*,$$

то есть алгоритм включения ближайшего является 1-оптимальным.

Доказательство. Докажем сначала, что для любого остовного дерева T

$$l^{NI} \le 2w(T). \qquad (2.13)$$

Чтобы сделать это, установим соответствие между вершинами, включаемыми в цикл, и ребрами данного дерева T. Одновременно с выполнением алгоритма будем строить семейство деревьев $\mathcal{T}$, каждое из которых включает только одну вершину текущего цикла.

Изначально $\mathcal{T} = \{T\}$. Удалим из T ребро $[v,k]$. Дерево T разбивается при этом на два дерева, одно из которых содержит вершину v, а второе – вершину k. Далее, выполняя второй шаг алгоритма, находим единственное дерево $T_k \in \mathcal{T}$, содержащее k. Пусть l – единственная вершина в T_k, принадлежащая текущему циклу. Заменим T_k в $\mathcal{T}$ двумя деревьями, полученными из T_k удалением первого ребра $[l,h]$ на единственном пути от l до k. В конце работы алгоритма $\mathcal{T}$ содержит n деревьев, не имеющих ребер.

Пусть на шаге 2 алгоритма добавляемая вершина k была ближайшей к вершине s текущего цикла. Если цикл содержит точно одну вершину, вставка k в текущий цикл увеличивает длину маршрута на $2c_{sk}$. Предположим, вершин в текущем цикле по крайней мере две. Пусть s' – одна из сопряженных с s

вершин цикла, и пусть ребро $[i, j]$ удаляется из текущего цикла при включении k. Длина маршрута на этом шаге увеличивается на

$$c_{ik} + c_{kj} - c_{ij} \le c_{sk} + c_{s'k} - c_{ss'} \le 2c_{sk},$$

где левое неравенство справедливо по выбору $[i, j]$, а правое – ввиду неравенства треугольника.

Так как l содержится в текущем цикле, а h – нет, $c_{ks} \le c_{lh}$. Следовательно, стоимость текущего цикла возрастает не более, чем на $2c_{lh}$. Наконец, так как это соотношение справедливо для каждого ребра в T и соответствующей вставляемой вершины, получаем (2.13).

Чтобы закончить доказательство, применяем оценку (2.13) для минимального остовного дерева T^*:

$$w(T^*) < l^* \le l^{NI} \le 2w(T^*),$$

откуда и вытекает утверждение теоремы.

Следующий пример показывает, что установленная в теореме оценка не может быть улучшена.

Пример 2.12. Рассмотрим цикл с $n \ge 6$ вершинами и ребрами длиной 1. Расстояние между двумя несмежными вершинами равно длине кратчайшего пути по циклу между ними, то есть

$$c_{ij} = c_{ji} = \min\{j - i, n - j + i\},\ 1 \le i < j \le n.$$

Очевидно, что оптимальный маршрут коммивояжера посещает вершины согласно их расположению в цикле, $l^* = n$. Применим алгоритм включения ближайшего. Алгоритм может начинать работу с вершины 1, затем добавлять к ней вершину 2, затем – 3 и т.д., поскольку каждая добавленная вершина находится на расстоянии 1 (наименьшем возможном) от вершины, уже входящей в маршрут. При этом может быть получен маршрут, состоящий из ребер $[1,2]$, $[n-1, n]$ и ребер $[k, k+2]$ для $1 \le k \le n-2$ (рис. 2.7). Его длина равна $l^{NI} = 2 + 2(n-2) = 2n - 2$.

Таким образом,

$$\frac{l^{NI}}{l^*} = \frac{2n-2}{n} = 2 - \frac{2}{n}.$$

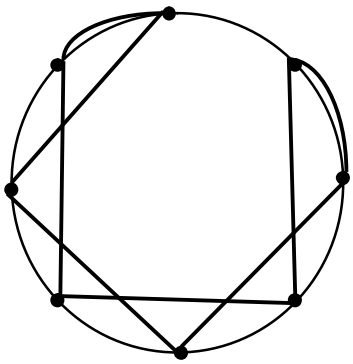


Рис. 2.7. Пример для алгоритма включения ближайшего

Пример 2.13. Применим рассмотренный алгоритм для решения задачи из примера 2.11. Пусть сначала цикл состоит из вершины 1, включаем далее в цикл ближайшую к 1 вершину 4, затем – вершину 3, получаем частичный цикл $(1,3,4,1)$. Ближайшая к вершинам цикла – вершина 5. Она может быть включена в цикл между вершинами 1 и 3 со стоимостью (удлинением маршрута) $36+38-31=43$, между вершинами 3 и 4 со стоимостью $38+24-14=48$, либо между вершинами 4 и 1 со стоимостью $24+36-22=38$. Получаем, таким образом, цикл $(1,3,4,5,1)$. Аналогично добавляя оставшуюся вершину 2, получаем маршрут $(1,3,4,5,2,1)$ длиной 129.

Естественным примером симметричных задач коммивояжера с матрицей длин дуг, удовлетворяющей неравенству треугольника, являются задачи коммивояжера на плоскости, то есть задачи коммивояжера для полных графов, вершины которого соответствуют точкам на плоскости, а длины дуг равны евклидовым расстояниям между этими точками.

Алгоритм включения ближайшего можно применять, в данном случае, начиная с выбора трех вершин графа, образующих треугольник наименьшего периметра.

Модификацией указанного подхода является следующий алгоритм [11]. Сначала строится выпуклая оболочка множества V вершин графа – наименьший по включению выпуклый многоугольник, содержащий V. Граница этого многоугольника образует цикл, проходящий через точки множества $S \subseteq V$. Если $S = V$, этот цикл является оптимальным маршрутом коммивояжера. В противном случае получен частичный цикл, начиная с которого выполняется процедура включения ближайшего.

2.3. Алгоритмы локального поиска

Многие приближенные алгоритмы решения задач дискретной оптимизации включают в себя два этапа – построение допустимого решения, например, с использованием эвристик, и его улучшение. На втором этапе применяются так называемые алгоритмы локального поиска или локальной оптимизации. В их основе лежит понятие окрестности допустимого решения – множества точек, в некотором смысле близких к данной.

В регулярных задачах оптимизации, например, задачах линейного программирования, множество допустимых решений – выпуклая область в R^n. Окрестность точки $x \in R^n$ определяется обычным образом, это множество точек, расстояние которых от x не больше заданного. При этом, поскольку целевая функция непрерывна, чем ближе друг к другу допустимые решения задачи, тем меньше различается их стоимость.

В задачах дискретной оптимизации окрестность точки вводится, как правило, искусственным образом и о близости двух решений можно судить только по значениям целевой функции на этих решениях.

Рассмотрим задачу

$$f(x) \to \min,\ x \in D. \qquad (2.14)$$

Отображение, которое каждому допустимому решению $x \in D$ задачи оптимизации (2.14) ставит в соответствие его окрестность $D(x) \subseteq D$, называется *системой окрестностей* или окрестностной функцией.

Например, окрестность булева вектора может быть получена заменой по некоторому правилу некоторых нулей единицами и наоборот.

Точка $x^* \in D$ называется *локально оптимальным решением* относительно заданной системы окрестностей, если $f(x^*) \le f(x)$ для всех $x \in D(x^*)$.

Пусть $x^0 \in D$. Для осуществления алгоритма локального поиска определяется окрестность $D(x^0)$ и решается задача

$$f(x) \to \min,\ x \in D(x^0). \qquad (2.15)$$

При этом либо просматриваются все элементы $D(x^0)$, либо решение задачи (2.15) продолжается до первой точки $x^1 \in D(x^0)$ такой, что $f(x^1) < f(x^0)$. Затем ведётся поиск в окрестности $D(x^1)$ найденной точки и так далее, пока решение удаётся улучшить.

Вводя новую систему окрестностей, можно повторить алгоритм, начиная с полученного решения. Кроме того, для достижения наилучшего результата локальный поиск можно осуществлять для нескольких начальных точек.

Очевидно, чем больше элементов содержит окрестность, тем, вообще говоря, будет лучше полученное решение. Следует, тем самым, найти компромисс между эффективностью алгоритма и его сложностью, то есть объёмом перебора.

Система окрестностей называется *точной*, если любое локально оптимальное относительно неё решение является оптимальным. В большинстве случаев не представляется возможным установить точность системы окрестностей, не совпадающих со всем множеством D. Кроме того, как правило, не удаётся оценить погрешность полученного решения.

Рассмотрим применение локального поиска для решения симметричной задачи коммивояжера. Пусть $z = (i_1, .., i_n)$ – некоторый маршрут коммивояжера.

Один из вариантов построения окрестности маршрута z – перестановка местами вершин i_k и i_s, $1 \le k < s \le n$. Таким образом, если $s \ne k+1$, дуги (i_{k-1}, i_k), (i_k, i_{k+1}), (i_{s-1}, i_s), (i_s, i_{s+1}) заменяются дугами (i_{k-1}, i_s), (i_s, i_{k+1}), (i_{s-1}, i_k), (i_s, i_{k+1}), в противном случае, при $s = k+1$, дуги (i_{k-1}, i_k), (i_k, i_s), (i_s, i_{s+1}) заменяются дугами (i_{k-1}, i_s), (i_s, i_k), (i_k, i_{s+1}). Такая окрестность содержит $(n-1)n/2$ точек.

Под окрестностью маршрута может пониматься также множество маршрутов, полученных перестановкой s подряд расположенных точек. То есть,

если $\pi = (\pi(1),...,\pi(s))$ – перестановка чисел $1,...,s$, часть маршрута $(i_{k-1}, i_k, ..., i_{k+s})$ заменяется на $(i_{k-1}, i_{k-1+\pi(1)}, ..., i_{k-1+\pi(s)}, i_{k+s})$.

Наиболее эффективными считаются окрестностные функции для симметричной задачи коммивояжера, которые называются *l-заменами*. Под окрестностью маршрута z понимается множество маршрутов, полученных из данного удалением l ребер и добавлением новых l ребер так, чтобы снова был образован маршрут коммивояжера. Например, при $l = 2$ несмежные ребра $[i, j]$ и $[k, s]$ заменяются на $[i, k]$ и $[j, s]$ (рис. 2.8), причём часть маршрута между вершинами j и k проходится в новом цикле в противоположном направлении. Общее число 2-замен равно $(n-3)n/2$, поэтому любая окрестность включает $1 + (n-3)n/2 = (n-1)(n-2)/2$ маршрута.

При $l = 3$ существует 4 варианта замены заданных трех ребер (рис. 2.9).

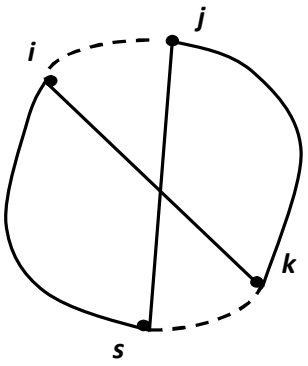


Рис. 2.8. 2-замена

Уже при $l = 4$ решение получается не настолько лучше, чем при $l = 3$, чтобы оправдать возросший объём вычислений. Очевидно, при $l = n$ система окрестностей является точной.

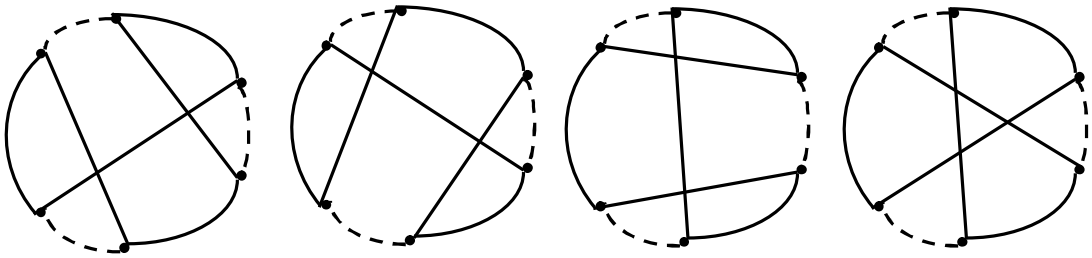


Рис. 2.9. 3-замена

l-замена называется *улучшающей*, если длина нового маршрута меньше исходного.

Маршрут коммивояжера называется *k-оптимальным*, если он является локально оптимальным относительно *l*-замены при всех $l \le k$. Таким образом, для определения *k*-оптимального маршрута последовательно находятся улучшающие замены для $l = 2,...,k$.

Справедливы следующие результаты [15] об оценке наихудшего случая *k*-оптимальных эвристик.

Теорема 2.9. *Пусть* $l^{(k)}$ *– длина k-оптимального маршрута,* l^* *- длина кратчайшего маршрута коммивояжера. Если матрица стоимостей удовлетворяет неравенству треугольника,*

$$\frac{l^{(2)}}{l^*} \le 4\sqrt{n}\,.$$

Существует бесконечное семейство входных данных задачи коммивояжера, удовлетворяющих неравенству треугольника, для которых

$$\frac{l^{(2)}}{l^*} \ge \frac{1}{4}\sqrt{n}\,.$$

Для всех $k \ge 3$ *существует бесконечно большое семейство данных задачи коммивояжера, удовлетворяющих неравенству треугольника, при которых*

$$\frac{l^{(k)}}{l^*} \ge \frac{1}{4}n^{1/(2k)}.$$

Несмотря на то, что *k*-оптимальные эвристики не обладают, вообще говоря, конечной оценкой наихудшего случая, исследования показывают, что во многих случаях эти алгоритмы могут быть высоко эффективны, порождать оптимальные или близкие к оптимальным решения. При этом для достаточно больших окрестностей качество полученного решения не определяется качеством начального решения.

Пример 2.14. Применим 2-оптимальную процедуру для задачи коммивояжера из примеров 2.11, 2.13. В качестве начального выберем маршрут $z^0 = (1,2,4,3,5,1)$ длиной 157, полученный применением алгоритма дерева. Рассмотрим все возможные 2-замены ребер $[i,j]$, $[k,s]$ и подсчитаем увеличение длины маршрута $\delta_{ik} = c_{ik} + c_{js} - c_{ij} - c_{ks}$:

$$\delta_{14} = 20\,,\ \delta_{13} = -3\,,\ \delta_{23} = -15\,,\ \delta_{25} = -23\,,\ \delta_{45} = 5\,.$$

Таким образом, заменяя ребра $[2,4]$, $[5,1]$ на $[2,5]$, $[4,1]$ получаем маршрут $z^1 = (1,2,5,3,4,1)$ длиной 134. Заменяя далее в z^1 ребра $[5,3]$ и $[4,1]$ на $[5,4]$ и $[3,1]$, находим $z^2 = (1,2,5,4,3,1)$ длиной 129. Если же взять $z^0 = (1,2,4,5,3,1)$ длиной 162, то z^2 получается за один шаг заменой ребер $[2,4]$, $[5,3]$.

2.4. Анализ среднего случая

Как показывают исследования, эвристический алгоритм, который хорошо работает на практике, может иметь слабый результат в наихудшем случае если, например, порождает очень плохие решения для одного (или более) патологических наборов данных.

Чтобы преодолеть это затруднение, применяется вероятностный анализ алгоритмов, позволяющий охарактеризовать среднее качество эвристики при некоторых предположениях о распределении данных задачи. При этом,

поскольку вероятностный анализ зачастую достаточно сложен даже для простых алгоритмов, обычно применяется асимптотический анализ, то есть среднее качество эвристики оценивается для очень больших задач. Иногда полученные результаты можно распространить и на задачи более разумного размера.

Пусть рассматривается некоторая задача комбинаторной оптимизации. Обозначим через f_n^* оптимальное значение целевой функции задачи размером n, через f_n^H – стоимость решения этой задачи, полученного применением алгоритма H. Относительная погрешность эвристики H –

$$e_n^H = \frac{f_n^H - f_n^*}{f_n^*}. \qquad (2.16)$$

Предположим, входные данные задачи – независимые случайные величины, имеющие распределение ψ. Алгоритм H называется *асимптотически оптимальным* для ψ, если с вероятностью единица

$$\lim_{n\to\infty} e_n^H = 0.$$

Таким образом, при определенных предположениях о распределении данных, H порождает решения, ожидаемая ошибка которых стремится к нулю при n, стремящемся к бесконечности.

Необходимым этапом анализа среднего случая является исследование поведения при $n \to \infty$ случайной величины f_n^*, позволяющее оценить ожидаемую стоимость оптимального решения задачи и погрешность алгоритма (2.16).

Далее будут рассмотрены некоторые применения указанного подхода к задаче об упаковке и задаче коммивояжера на плоскости. Полученные для этих задач результаты могут быть использованы при анализе сложных практических задач, возникающих, например, при моделировании процессов распределения и транспортировки в логистических сетях.

Задача об упаковке

Рассмотрим задачу об упаковке, в которой размеры предметов w_1, w_2,… – независимые и одинаково распределенные на $(0,1]$ согласно некоторому распределению Ф случайные величины.

Очевидно, что оптимальное решение задачи об упаковке обладает следующим свойством: для любых двух списков L' и L''

$$b^*(L' \cup L'') \le b^*(L') + b^*(L'').$$

Последовательность положительных действительных чисел $\{a_n\}$, $n \ge 1$, называется *субаддитивной*, если $a_n + a_m \ge a_{n+m}$ для любых n и m.

Теорема 2.10. *Если последовательность* $\{a_n\}$, $n \ge 1$, *субаддитивна, тогда существует константа* γ *такая, что*

$$\lim_{n\to\infty} \frac{a_n}{n} = \gamma.$$

Пусть минимальное количество контейнеров, необходимое для упаковки n предметов с размерами, имеющими распределение Φ – случайная величина b_n^*. На основании теоремы 2.10 можно показать, что с вероятностью единица

$$\lim_{n\to\infty}\frac{b_n^*}{n}=\gamma,$$

где γ зависит только от распределения Φ.

Для того, чтобы оценить отклонения величины b_n^* от её среднего значения $E\left(b_n^*\right)$ и величины b_n^*/n от γ, применяется теория случайных процессов. Справедливы следующие результаты [15].

Теорема 2.11.

$$\Pr\left(\left|b_n^*-E\left[b_n^*\right]\right|>t\right)\le 2\exp\left\{-t^2/(2n)\right\}.$$

Для *каждого* $\varepsilon>0$ *найдётся* $n(\varepsilon)>0$ *такое, что для всех* $n\ge n(\varepsilon)$

$$\Pr\left(\left|b_n^*/n-\gamma\right|>\varepsilon\right)<2\exp\left\{-n\varepsilon^2/2\right\}.$$

Из теоремы вытекает, что для любого распределения размеров предметов при увеличении n значение b_n^* быстро становится близким к γn, следовательно, γn может служить хорошей аппроксимацией количества контейнеров, требуемого для упаковки n предметов.

Рассмотрим случай, когда Φ – равномерное распределение на $[0,1]$. Чтобы упаковать множество из n элементов, соответствующих этому распределению, можно использовать эвристику разрезания интервала с параметром r (Sliced Interval Partitioning, SIP(r)). Пусть задано целое число $r\ge 1$. Множество L делится на следующие $2r$ непересекающихся подмножеств, некоторые из которых могут быть пустыми:

$$L_j=\left\{w_k:\frac{1}{2}\left(1-\frac{j+1}{r}\right)<w_k\le\frac{1}{2}\left(1-\frac{j}{r}\right)\right\},\ j=1,...,r-1,$$

$$L^j=\left\{w_k:\frac{1}{2}\left(1+\frac{j-1}{r}\right)<w_k\le\frac{1}{2}\left(1+\frac{j}{r}\right)\right\},\ j=1,...,r-1,$$

$$L_0=\left\{w_k:\frac{1}{2}\left(1-\frac{1}{r}\right)<w_k\le\frac{1}{2}\right\},\ L^r=\left\{w_k:w_k>\frac{1}{2}\left(1+\frac{r-1}{r}\right)\right\}.$$

Обозначаем $n_j=\left|L_j\right|$, $n^j=\left|L^j\right|$, где $j=0,...,r$.

Эвристика SIP(r) сначала упаковывает в контейнеры по два предмета, один из L_j, другой – из L^j, $j=1,...,r-1$. Далее, все предметы из $L_0\cup L^r$, а также оставшиеся $\left|n_j-n^j\right|$ предметов, если $n_j\ne n^j$, помещаются в контейнеры по одному. Таким образом, общее число используемых контейнеров

$$n_0+n^r+\sum_{j=1}^{r-1}\max\left\{n_j,n^j\right\}.$$

Поскольку почти наверное

$$\lim_{n\to\infty}\frac{n_j}{n}=\lim_{n\to\infty}\frac{n^j}{n}=\frac{1}{2r}\ \text{для всех}\ j=1,...,r,$$

$$\gamma=\lim_{n\to\infty}\frac{b_n^*}{n}\le\lim_{n\to\infty}\frac{1}{n}\left[n_0+n^r+\sum_{j=1}^{r-1}\max\left\{n_j,n^j\right\}\right]=\frac{1}{2}+\frac{1}{2r}.$$

Так как это справедливо для любого $r>1$, $\gamma\le 1/2$. С другой стороны, поскольку $\gamma\ge E(w)$ (см. упражнение 27), то $\gamma\ge 1/2$, то есть $\gamma=1/2$ для равномерного распределения на $[0,1]$.

Заметим, что эвристика SIP(r) не является асимптотически оптимальной, так как для любого фиксированного r ошибка стремится к $1/r$.

Один из возможных асимптотически оптимальных алгоритмов следующий. Элементы списка L упорядочиваются по убыванию. Затем наибольший элемент, например, w_j, помещается в один контейнер с наибольшим элементом w_k таким, что $w_j+w_k\le 1$, либо в отдельный контейнер, если такого элемента нет. Далее процедура повторяется с оставшимися элементами списка, пока все предметы не будут упакованы.

Задача коммивояжера на плоскости.

Пусть x_1, …, x_n – множество точек на плоскости. Обозначим через l_n^* длину оптимального маршрута коммивояжера через эти n точек.

Теорема 2.12. *Пусть $a\times b$ – размер наименьшего прямоугольника, содержащего x_1, …, x_n, тогда*

$$l_n^*\le\sqrt{2(n-2)ab}+2(a+b).$$

Доказательство. Возьмём целое число m и разделим прямоугольник размера $a\times b$ (где a – ширина и b – высота) на $2m$ горизонтальных полос равной ширины. Для этого проведем $2m+1$ горизонтальных и две вертикальные (очерчивающие границы прямоугольника) прямые. Пронумеруем горизонтальные прямые 1, 2, …, $2m+1$ сверху вниз.

Временно удалим все прямые с четными номерами. Соединим каждую точку x_i, $i=1,...,n$ двумя вертикальными отрезками с ближайшей горизонтальной прямой, имеющей нечетный номер. Чтобы построить маршрут через x_1, ..., x_n выходим из верхнего левого угла прямоугольника и двигаясь слева направо по первой горизонтальной прямой, собираем все связанные с ней вертикальными отрезками точки, затем спускаемся и проходим аналогично третью горизонтальную прямую справа налево и так далее, до конца $2m+1$–й прямой, и возвращаемся в начальную точку.

Повторяем процедуру для четных прямых и, аналогичным образом, получаем второй маршрут, проходящий через все точки (рис. 2.10).

Сумма длин двух построенных маршрутов равна

$$a(2m+1)+\frac{nb}{m}+2b+a+2\left(b-\frac{b}{m}\right).$$

Так как l_n^* не превосходит длины любого из этих маршрутов,

$$l_n^* \le a + 2b + ma + (n-2)\frac{b}{2m}.$$

Минимизируя правую часть по m, получаем

$$m^* = \left[\sqrt{\frac{b[n-2]}{2a}}\right],$$

то есть

$$L_n^* \le a + 2b + m^* a + \frac{b(n-2)}{2m^*} \le$$

$$\le a + 2b + a\left(\sqrt{\frac{b(n-2)}{2a}} + 1\right) + \frac{b(n-2)}{2}\sqrt{\frac{2a}{(n-2)b}} = \sqrt{2(n-2)ab} + 2(a+b).$$

Из этого результата следует, что длина оптимального маршрута коммивояжера оценивается как $O(\sqrt{n})$. Справедлива, кроме того, следующая теорема.

Теорема 2.13. *Пусть $x_1, \ldots, x_n$ – множество независимых случайных величин, имеющих распределение μ с компактным носителем в R^2. Тогда существует константа $\beta > 0$, не зависящая от распределения μ такая, что с вероятностью единица*

$$\lim_{n\to\infty} \frac{l_n^*}{\sqrt{n}} = \beta \int_{R^2} f^{1/2}(x)dx,$$

где f – плотность распределения μ.

Для решения задачи коммивояжера большого размера на плоскости может быть использован следующий подход. Сначала всё множество точек делится на подмножества, затем решается задача для каждого подмножества и, наконец, полученные маршруты объединяются в один.

Рассмотрим, например, следующий **алгоритм разбиения области**. Наименьший прямоугольник $a \times b$, содержащий множество $x_1, \ldots, x_n$, делится с помощью горизонтальных и вертикальных прямых на подобласти, каждая из которых (за исключением, возможно, одной), включает q точек. Для этого сначала проводится t вертикальных прямых так, что каждый получившийся прямоугольник, кроме, возможно, последнего, содержит $(h+1)q$ точку: точки нумеруются в порядке возрастания их абсциссы, прямая $j \le t$ проходит через точку с номером $j(h+1)q$.

Затем каждый из $t+1$ прямоугольников делится h горизонтальными прямыми на $h+1$ меньших прямоугольников так, что каждый содержит точно q точек кроме, возможно, последнего: точки в каждой вертикальной полосе нумеруются по возрастанию их ординаты, прямая $j \le h$ проходит через точку с номером jq.

Для каждой построенной подобласти решается задача коммивояжера. Например, алгоритм динамического программирования находит оптимальный маршрут через m точек за время $O\left(m^2 2^m\right)$. Если выбрать $q = \left[\log_2 n\right]$, то, так как число подобластей не превышает $1 + n/q$, общее время, затрачиваемое на решение задачи, – $O\left(n^2 \ln n\right)$.

Поскольку построенный набор маршрутов определяет эйлеров граф, в котором степень каждой точки (вершины) либо 2, либо 4 (для точек на границе двух подобластей), можно найти эйлеров маршрут и вложенный в него маршрут коммивояжера.

Чтобы гарантировать, что каждая непустая подобласть содержит точно q точек, за исключением, быть может, одной, h и t должны удовлетворять условиям

$$t = \left[\frac{n}{(h+1)q}\right],$$

$$t(h+1)q < n \le (t+1)(h+1)q .$$

Это достигается выбором $h = \left[\sqrt{n/q}\right]$. На рис. 2.10 приведён пример применения алгоритма для $n = 17$, $q = 4$, $h = 2$ и $t = 1$.

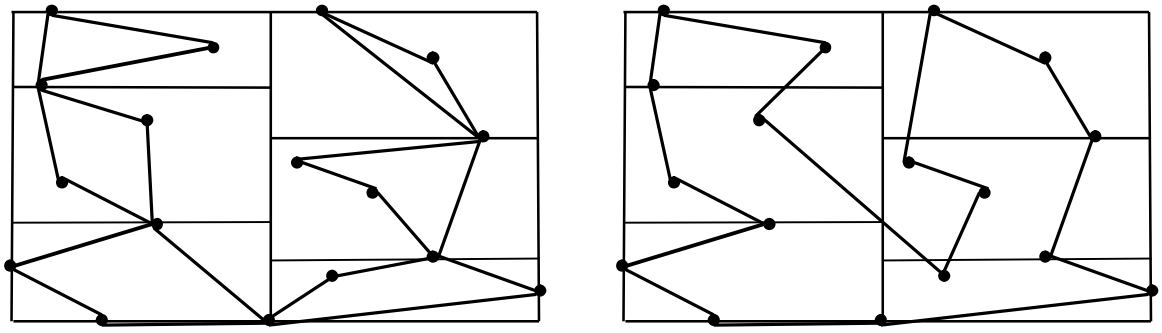


Рис. 2.10. Алгоритм разбиения области

Пусть l^{RP} – длина маршрута, порожденного эвристикой разбиения области. Справедлива следующая теорема, из которой вытекает асимптотическая оптимальность алгоритма.

Теорема 2.14. *При выполнении условий теоремы 2.13 с вероятностью единица*

$$\lim_{n\to\infty} \frac{l_n^*}{\sqrt{n}} = \lim_{n\to\infty} \frac{l}{\sqrt{n}}.$$

3. ОЦЕНКИ СНИЗУ ОПТИМАЛЬНОГО ЗНАЧЕНИЯ ЦЕЛЕВОЙ ФУНКЦИИ

Необходимым этапом решения задачи дискретной оптимизации методом ветвей и границ является оценивание снизу значения целевой функции на некотором множестве. Кроме того, как отмечено в главе 2, важным методом оценки эффективности эвристического алгоритма является сравнение полученного с его помощью значения целевой функции с нижней границей на стоимость оптимального решения. Как правило, для получения оценок решается вспомогательная задача, полученная из исходной, например, отбрасыванием или ослаблением (релаксированием) некоторых ограничений. При применении такого подхода необходимо, в первую очередь, выбрать формулировку задачи как задачи целочисленного программирования, на основании которой будет строиться оценка. Может оказаться, что получившаяся релаксированная задача содержит так много переменных или ограничений, что для её решения практически непригодны стандартные методы и требуется применение специальных алгоритмов.

В данной главе рассматривается метод получения оценок снизу для оптимального значения целевой функции задачи целочисленного линейного программирования, аналогичный методу множителей Лагранжа для задач математического программирования.

3.1. Метод Лагранжевой релаксации

Рассмотрим следующую задачу целочисленного линейного программирования:

$$\langle c, x \rangle \to \min \tag{3.1}$$

$$Ax = b, \tag{3.2}$$

$$Dx \geq e, \tag{3.3}$$

$$x = (x_1, ..., x_n),\ x_i \geq 0,\ i = 1, ..., n, \tag{3.4}$$

$$x_i - \text{целые},\ i = 1, ..., n, \tag{3.5}$$

где $c \in R^n$, $b \in R^m$, $e \in R^k$, A – матрица $m \times n$, D – матрица $k \times n$.

Естественным способом получения оценки снизу для оптимального значения f^* целевой функции задачи (3.1) – (3.5) является отбрасывание условия целочисленности (3.5). Задача линейного программирования (3.1) – (3.4) называется *линейной релаксацией* задачи (3.1) – (3.5). Если f_{LP} – оптимальное значение целевой функции релаксированной задачи, то

$$f_{LP} \leq f^*.$$

Лагранжевой релаксацией ограничений (3.2) с множителями $\lambda \in R^m$ называется задача

$$\langle c,x\rangle - \langle \lambda, Ax-b\rangle \to \min \qquad (3.6)$$

при условиях (3.3)–(3.5).

Функция $L(x,\lambda)=\langle c,x\rangle - \langle \lambda, Ax-b\rangle$ называется *функцией Лагранжа* задачи (3.1) – (3.5).

Обозначим оптимальное значение целевой функции задачи (3.6), (3.3)–(3.5) через $f_D(\lambda)$.

Лемма 3.1. *Для всех* $\lambda \in R^m$

$$f_D(\lambda) \le f^*.$$

Доказательство. Любое допустимое решение x задачи (3.1) – (3.5) будет допустимым и для задачи (3.6), (3.3) – (3.5), поэтому

$$f_D(\lambda) \le \langle c,x\rangle - \langle \lambda, Ax-b\rangle = \langle c,x\rangle .$$

Поскольку это неравенство справедливо и для оптимального решения, лемма доказана.

Лемма 3.1 означает, что $f_D(\lambda)$ представляет собой оценку снизу на оптимальное значение целевой функции при всех λ. Для того, чтобы определить лучшую, то есть наибольшую нижнюю границу, решается двойственная задача

$$f_D(\lambda) \to \max,\ \lambda \in R^m \qquad (3.7)$$

Лемма 3.2. *Функция* $f_D(\lambda)$ *кусочно-линейная и вогнутая.*

Сравним нижнюю границу Лагранжевой релаксации (f_D) с нижней границей, достигнутой решением линейной релаксации задачи (f_{LP}).

Теорема 3.1.

$$f_{LP} \le f_D .$$

Доказательство.

Задача, двойственная к задаче линейного программирования (3.1) – (3.4) имеет вид

$$\langle b,\lambda\rangle + \langle e,\eta\rangle \to \max$$
$$A^T\lambda + D^T\eta \le c,$$
$$\lambda \in R^m,\ \eta \in R^m,\ \eta \ge 0,$$

а задача, двойственная к задаче (3.6), (3.3), (3.4), –

$$\langle e,\eta\rangle \to \max$$
$$D^T\eta \le c - A^T\lambda,$$
$$\eta \in R^m,\ \eta \ge 0.$$

Оптимальное значение целевой функции первой задачи равно, согласно теории двойственности, f_{LP}, а оптимальное значение целевой функций второй задачи, $f_D^*(\lambda)$, удовлетворяет условию

$$f_D^*(\lambda) + \langle \lambda, b\rangle \le f_D(\lambda).$$

Так как при каждом фиксированном $\lambda \in R^m$ сформулированные задачи эквивалентны, получаем

$$f_D = \max_{\lambda} f_D(\lambda) \geq \max_{\lambda}\left(f_D^*(\lambda) + \langle \lambda, b\rangle\right) = f_{LP},$$

что и требовалось доказать.

Замечание 3.1. Лагранжевой релаксацией ограничений (3.3) называется задача

$$\langle c, x\rangle - \langle \lambda, Dx - e\rangle \to \min$$

при условиях (3.2), (3.4), (3.5). Для Лагранжевой релаксации ограничений (3.3) лемма 3.1 справедлива при $\lambda \geq 0$. Аналогично корректируются постановка двойственной задачи и утверждение теоремы 3.1.

Говорят, что задача линейного программирования обладает свойством *целочисленности*, если она имеет целочисленный оптимальный опорный план. Из доказательства теоремы 3.1 вытекает

Замечание 3.2. Если задача (3.1)–(3.4) обладает свойством целочисленности, то $f_{LP} = f_D$.

Утверждение леммы 3.2 означает, что функция $f_D(\lambda)$ достигает максимума в точке разрыва производной. Точку максимума можно найти, например, с использованием метода субградиентной оптимизации [3].

Субградиентная оптимизация

Пусть функция h определена и выпукла на некотором выпуклом множестве $U \subset R^n$. Для любой точки $x \in U$ найдётся такой вектор $p(x) \in R^n$, что при всех $y \in U$

$$h(y) - h(x) \geq \langle p(x), y - x\rangle.$$

Вектор $p(x)$ называется *субградиентом* функции h в точке x. Если функция h непрерывно дифференцируема на U, то для всех $x, y \in U$

$$h(y) - h(x) \geq \langle \operatorname{grad} h(x), y - x\rangle.$$

Таким образом, понятие субградиента является обобщением понятия градиента.

Рассмотрим задачу

$$h(x) \to \min,\ x \in R^n. \tag{3.8}$$

Субградиентный метод решения задачи (3.8) заключается в построении последовательности x_k, $k = 1,2,...$, приближенных решений по формуле

$$x_{k+1} = x_k - \tau_k p(x_k),\ k = 1,2,...,\ x_0 \in R^n. \tag{3.9}$$

Числа $\tau_k > 0$ – размеры шага. Если на некотором этапе $p(x_k) = 0$, то, по определению субградиента

$$h(y) - h(x_k) \geq 0 \text{ для всех } y \in R^n,$$

то есть x_k – решение задачи (3.8).

Теорема 3.2. *Если задача (3.8) имеет решение и* τ_k *удовлетворяют условиям*

$$\sum_{k=0}^{\infty} \tau_k = \infty,\ \sum_{k=0}^{\infty} \tau_k^2 < \infty,$$

итерационный процесс (3.9) сходится к решению задачи (3.8).

Один из способов определения размера шага следующий. Пусть известно минимальное значение h^* функции h. Тогда можно взять

$$\tau_k = \frac{h(x_k) - h^*}{\|p(x_k)\|^2}. \tag{3.10}$$

Применим указанный метод к задаче (3.7). Максимизация вогнутой функции $f_D(\lambda)$ эквивалентна минимизации выпуклой функции $-f_D(\lambda)$.

Пусть $\lambda, \mu \in R^m$, $f_D(\lambda) = L(x, \lambda)$. Так как $f_D(\mu) \le \langle c, x\rangle - \langle \mu, Ax - b\rangle$,

$$-f_D(\mu) + f_D(\lambda) \ge -\langle c, x\rangle + \langle \mu, Ax - b\rangle + \langle c, x\rangle - \langle \lambda, Ax - b\rangle = \langle \mu - \lambda, Ax - b\rangle.$$

Таким образом, $Ax - b$ – субградиент функции $-f_D(\lambda)$ в точке $\lambda \in R^m$.

Итерационный процесс решения задачи (3.7) имеет, согласно (3.9), вид

$$\lambda^{k+1} = \lambda^k - \tau_k(Ax^k - b),\ k = 1,2,\ldots,\ \lambda^0 \in R^m, \tag{3.11}$$

где x^k – оптимальное решение задачи (3.6), (3.3)–(3.5) при $\lambda = \lambda^k$.

По аналогии с (3.10), на практике обычно используется размер шага

$$\tau_k = \frac{\theta_k (UB - f_D(\lambda^k))}{\left\|Ax^k - b\right\|^2},$$

где UB – оценка сверху для оптимального значения целевой функции задачи (3.1) – (3.5), θ_k – параметр, удовлетворяющий условию $0 < \theta_k \le 2$. Можно, например, начинать с $\theta_0 = 2$ и уменьшать его вдвое после того, как на протяжении нескольких итераций значение f_D не увеличивается.

Если максимальное значение двойственной функции совпадает с минимальным значением целевой функции исходной задачи, метод Лагранжевой релаксации позволяет найти оптимальное решение. В противном случае, полученная нижняя граница для оптимального решения целевой функции может быть использована в методе ветвей и границ. Кроме того, для некоторых задач оказывается возможным построить хорошее приближенное решение задачи (3.1)–(3.5) на основании решения релаксированной задачи.

Применение метода Лагранжевой релаксации оправданно, если релаксированная задача решается существенно проще исходной. В дальнейшем будем считать, что можно так перенумеровать переменные, что матрица D – блочно-диагональная:

$$D = \begin{pmatrix} D_1 & & & 0 \\ & D_2 & & \\ & & \ddots & \\ 0 & & & D_T \end{pmatrix}.$$

Полагаем, далее,

$$c = (c_1,\ldots,c_T),\ x = (x_1,\ldots,x_T),\ e = (e_1,\ldots,e_T),\ A = (A_1,\ldots,A_T),$$

где размерности векторов c_t, x_t и количество столбцов матрицы A_t при $t = 1,...,T$ равно количеству столбцов матрицы D_t, а размерность вектора e_t равна количеству строк матрицы D_t.

Лагранжева релаксация ограничений (3.2) принимает тогда вид

$$\sum_{t=1}^{T} \langle c_t, x_t \rangle - \sum_{t=1}^{T} \langle \lambda, A_t x_t - b \rangle \to \min \tag{3.12}$$

$$D_t x_t \geq e_t,\ t = 1,...,T, \tag{3.13}$$

$$x_t \geq 0 \text{ и целочисленные},\ t = 1,...,T. \tag{3.14}$$

Решение задачи (3.12)–(3.14) сводится к решению T задач (для каждого $t = 1,...,T$) меньшего размера

$$\langle c_t - \lambda A_t, x_t \rangle \to \min \tag{3.15}$$

$$D_t x_t \geq e_t, \tag{3.16}$$

$$x_t \geq 0 \text{ и целочисленный.} \tag{3.17}$$

3.2. Подход, основанный на генерации столбцов

Рассмотрим альтернативный подход к решению двойственной задачи (3.7). Пусть X_t – множество векторов, удовлетворяющих условиям (3.16), (3.17). Предположим, все множества X_t конечные и их элементы перенумерованы, $X_t = \{x_1^t,...,x_{r_t}^t\}$. Пусть $\eta_t = \min\limits_j \langle c_t - \lambda A_t, x_j^t \rangle$. Тогда задача (3.7) принимает вид

$$\sum_{t=1}^{T} \eta_t + \sum_{i=1}^{m} \lambda_i b_i \to \max \tag{3.18}$$

$$\eta_t \leq \langle c_t - \lambda A_t, x_j^t \rangle,\ t = 1,...,T,\ j = 1,...,r_t. \tag{3.19}$$

Введём переменные ξ_{tj}, принимающие значение 1, если вектор x_j^t входит в оптимальное решение задачи (3.1) – (3.5), и 0 в противном случае.

Тогда задачу (3.1) – (3.5) можно переписать в виде

$$\sum_{t=1}^{T} \sum_{j=1}^{r_t} c_{tj} \xi_{tj} \to \min \tag{3.20}$$

$$\sum_{j=1}^{r_t} \xi_{tj} = 1,\ t = 1,...,T, \tag{3.21}$$

$$\sum_{t=1}^{T} \sum_{j=1}^{r_t} A_t x_j^t \xi_{tj} = b, \tag{3.22}$$

$$\xi_{tj} \geq 0,\ t = 1,...,T,\ j = 1,...,r_t, \tag{3.23}$$

$$\xi_{tj} - \text{целые},\ t = 1,...,T,\ j = 1,...,r_t. \qquad (3.24)$$

Нетрудно убедиться, что задача, двойственная к линейной релаксации задачи (3.20)–(3.24) – это в точности задача (3.18), (3.19). Таким образом, наилучшая нижняя граница, полученная с помощью Лагранжевых релаксаций, совпадает с оптимальным значением целевой функции задачи (3.20)–(3.23).

Замечание 3.3. Постановку задачи комбинаторной оптимизации в виде (3.20)–(3.24) называют формулировкой разбиения множества (set-partitioning formulation) ввиду сходства задачи (3.20)–(3.24) со следующей проблемой. Дано некоторое множество $S = \{x_1,....,x_n\}$. Требуется разбить его на непересекающиеся подмножества оптимальным по некоторому критерию способом. Пусть $S_k \subseteq S$, $k = 1,...,K$ – допустимые подмножества, $x_{ik} = 1$ если $x_i \in S_k$, и $x_{ik} = 0$ в противном случае, $i = 1,...,n$, c_k – стоимость множества S_k. Вводя переменные ξ_k, принимающие значение 1 или 0 в зависимости от того, входит ли множество S_k в оптимальное разбиение, получаем задачу

$$\sum_{k=1}^{K} c_k \xi_k \to \min$$

$$\sum_{k=1}^{K} x_{ik} \xi_k = 1,\ i = 1,...,n,$$

$$\xi_k \in \{0,1\},\ k = 1,...,K.$$

Задача (3.20)–(3.23) содержит экстремально большое количество неизвестных, а двойственная к ней задача (3.18), (3.19) – такое же число ограничений. Для решения задач линейного программирования большой размерности может быть применён следующий подход, называемый *методом генерации столбцов*.

Заменим в (3.20)–(3.23) и (3.18), (3.19) множества X_t их подмножествами $\overline{X}_t$, так, чтобы задача (3.20)–(3.23) осталась допустимой.

Пусть $\overline{\xi}$ – оптимальное решение полученной ограниченной прямой задачи, а $(\overline{\eta}, \overline{\lambda})$ – двойственной к ней задачи. Тогда, согласно теории двойственности, $\overline{\xi}$ будет оптимальным решением исходной задачи (3.20)–(3.23), если $(\overline{\eta}, \overline{\lambda})$ – допустимое решение задачи (3.18), (3.19), то есть неравенства (3.19) выполнены для всех элементов множеств X_t.

Если для некоторых $t = 1,...,T$ и $j = 1,...,r_t$ условие (3.19) нарушается, то есть

$$\overline{\eta}_t + \left\langle \overline{\lambda}, A_t x_j^t \right\rangle > \left\langle c_t, x_j^t \right\rangle,$$

соответствующий элемент x_j^t добавляется к множеству $\overline{X}_t$ и процесс повторяется. Для того, чтобы обнаружить такие столбцы, решаются вспомогательные задачи

$$\left\langle c_t - \overline{\lambda} A_t, x^t \right\rangle \to \min,\ x^t \in X_t,$$

совпадающие с (3.15)–(3.17). Решение вспомогательной задачи порождает столбец, если оптимальное значение целевой функции этой задачи меньше $\overline{\eta}_t$.

Замечание 3.4. Решение ограниченной двойственной задачи после добавления ограничений можно начинать с решения, полученного на предыдущем этапе, используя двойственный симплекс-метод.

Замечание 3.5. Рассмотренный подход является реализацией для задачи (3.20)–(3.23) метода обобщённого линейного программирования [1], отличающегося от симплекс-метода способом выбора включаемого в базис столбца.

3.3. Асимптотически точные границы для задачи об упаковке

Рассмотрим задачу об упаковке в контейнеры.

$$\sum_{j=1}^{n} y_j \to \min \tag{3.25}$$

$$\sum_{i=1}^{n} w_i x_{ij} \le y_j,\ j = 1,...,n, \tag{3.26}$$

$$\sum_{i=1}^{n} x_{ij} = 1,\ i = 1,...,n, \tag{3.27}$$

$$x_{ij} \in \{0,1\},\ y_j \in \{0,1\},\ i, j = 1,...,n. \tag{3.28}$$

Лагранжева релаксация ограничений (3.27) – задача

$$\sum_{j=1}^{n} y_j - \sum_{i=1}^{n} \lambda_i \left(\sum_{j=1}^{n} x_{ij} - 1 \right) \to \min \tag{3.29}$$

при условиях (3.26), (3.28). Эта задача разбивается на n задач (для $j = 1,...,n$)

$$y_j - \sum_{i=1}^{n} \lambda_i x_{ij} \to \min \tag{3.30}$$

$$\sum_{i=1}^{n} w_i x_{ij} \le y_j, \tag{3.31}$$

$$x_{ij} \in \{0,1\},\ y_j \in \{0,1\},\ i = 1,...,n. \tag{3.32}$$

Если $y_j = 0$, то все $x_{ij} = 0$ и значение целевой функции задачи (3.30)–(3.32) равно 0. При $y_j = 1$ задача (3.30)–(3.32) эквивалентна задаче о бинарном ранце

$$\sum_{i=1}^{n} \lambda_i x_{ij} \to \max \tag{3.33}$$

$$\sum_{i=1}^{n} w_i x_{ij} \le 1, \tag{3.34}$$

$$x_{ij} \in \{0,1\},\ i = 1,...,n. \tag{3.35}$$

Обозначая оптимальное значение целевой функции задачи (3.33)–(3.35) через $g'(\lambda)$, получаем, что оптимальное значение целевой функции задачи (3.30)–(3.32) равно

$$g(\lambda) = \min\{0,1 - g'(\lambda)\},$$

а оптимальное значение целевой функции задачи (3.29), (3.26), (3.28) –

$$f_D(\lambda) = ng(\lambda) + \sum_{i=1}^{n} \lambda_i .$$

Субградиентный алгоритм максимизации $f_D(\lambda)$ реализуется по формуле

$$\lambda^{k+1} = \lambda^k - \tau_k \left(\sum_{j=1}^{n} x_{ij}^k - 1 \right).$$

Сформулируем задачу об упаковке как задачу о разбиении множества. Пусть I – множество всех наборов элементов из списка L, которые можно упаковать в один контейнер, то есть

$$I = \left\{ S \subseteq \{1,...,n\} : \sum_{i \in S} w_i \le 1 \right\}.$$

Для $i = 1,...,n$, $S \in I$, пусть $\alpha_{iS} = 1$, если $i \in S$ и $\alpha_{iS} = 0$ в противном случае.

Введём переменные y_S, принимающие значение 1, если множество S использовано при упаковке, и 0 в противном случае.

Получаем следующую задачу целочисленного линейного программирования.

$$\sum_{S \in F} y_S \to \min \tag{3.36}$$

$$\sum_{S \in F} \alpha_{is} y_S = 1,\ i = 1,...,n, \tag{3.37}$$

$$y_S \in \{0,1\},\ S \in I. \tag{3.38}$$

Задача, двойственная к линейной релаксации задачи (3.36) – (3.38) имеет вид

$$\sum_{i=1}^{n} \lambda_i \to \max \tag{3.39}$$

$$\sum_{i=1}^{n} \alpha_{iS} \lambda_i \le 1,\ S \in I. \tag{3.40}$$

Применим метод генерации столбцов. Заменим в (3.36) – (3.38) и (3.39), (3.40) множество I подмножеством $I' \subset I$. Тогда полученные решения $\overline{y}_S$, $S \in I'$ и $\overline{\lambda}_i$, $i = 1,...,n$ соответствующих задач будут оптимальными, если

$$-\sum_{i=1}^{n} \alpha_{iS} \overline{\lambda}_i \ge -1 \text{ для всех } S \in I. \tag{3.41}$$

Для проверки (3.41) и обнаружения множеств $S \in I$, которые нужно добавить к I', решается задача

$$-\sum_{i=1}^{n}\alpha_{iS}\overline{\lambda}_i \to \min\,,\ S \in I\,,$$

то есть задача (3.33) – (3.35) с $\lambda = \overline{\lambda}$. Соответствующий оптимальному решению вспомогательной задачи вариант упаковки добавляется к I', если оптимальное значение целевой функции больше 1.

Справедлива следующая теорема [15], которая показывает, что разность между оптимальным значением целевой функции линейной релаксации задачи (3.36) – (3.38) и оптимальным количеством контейнеров стремится к нулю при возрастании количества предметов n.

Функция φ называется непрерывной по Липшицу на множестве $\Omega \subseteq R$, если существует константа K такая, что

$$|\varphi(x)-\varphi(y)| \le K|x-y| \text{ для любых } x, y \in \Omega.$$

Теорема 3.3. *Пусть элементы списка L – независимые, одинаково распределенные случайные величины, и плотность их распределения φ – непрерывная по Липшицу на $[0,1]$ функция. Пусть b_n^{LP} – оптимальное значение целевой функции линейной релаксации задачи (3.36) – (3.38), b_n^* – оптимальное значение целевой функции задачи (3.36) – (3.38), то есть – значение оптимального решения задачи об упаковке. Тогда, с вероятностью единица,*

$$\lim_{n\to\infty}\frac{1}{n}b_n^{LP} = \lim_{n\to\infty}\frac{1}{n}b_n^*.$$

Анализ наихудшего случая формулировки (3.36) – (3.38) для задачи об упаковке приводит к следующему результату [15].

Теорема 3.4. *Для любого списка L*

$$b^*(L)/b^{LP}(L) \le 4/3,$$

то есть значение оптимального решения линейной релаксации обеспечивает, по крайней мере, 75% оптимального решения.

3.4. Лагранжевая релаксация для задачи коммивояжера

Рассмотрим симметричную задачу коммивояжера в графе $G = (V, E)$ с матрицей длин дуг $C = (c_{ij})$. Любой маршрут коммивояжера в неориентированном графе – 1-дерево, в котором каждая вершина имеет степень 2. Поэтому вес минимального 1-дерева, не являющегося циклом, оценивает снизу длину оптимального маршрута коммивояжера. Рассмотрим построение наилучшей оценки такого типа.

Справедлива

Лемма 3.3. *Пусть $\lambda \in R^n$ — произвольный вектор. Задача коммивояжера с матрицей длин дуг $C' = (c'_{ij})$, где*

$$c'_{ij} = c_{ij} + \lambda_i + \lambda_j\,,\ i, j = 1,\ldots,n, \qquad (3.42)$$

эквивалентна исходной задаче коммивояжера.

Пусть l^* – длина оптимального маршрута, соответствующего матрице C. После преобразования (3.42), называемого *штрафованием вершин*, оптимальный маршрут остаётся кратчайшим, но его длина становится равной

$$l^* + 2\sum_{i=1}^{n} \lambda_i .$$

В отличие от гамильтоновых циклов, относительные веса 1-деревьев могут измениться при переходе к матрице C'. Перенумеруем все возможные 1-деревья. Пусть d_i^k – степень вершины i в k-м 1-дереве, w_k – его вес до преобразования матрицы длин дуг. Тогда после преобразования (3.42) вес k-го 1-дерева составит

$$w_k + \sum_{i \in V} d_i^k \lambda_i .$$

Таким образом, минимальный вес 1-дерева для преобразованной матрицы равен

$$\min_k \{w_k + \sum_{i \in V} d_i^k \lambda_i\}$$

и, следовательно,

$$l^* + 2\sum_{i \in V} \lambda_i \geq \min_k \{w_k + \sum_{i \in V} d_i^k \lambda_i\}. \qquad (3.43)$$

Определим функцию $w(\lambda)$ выражением

$$w(\lambda) = \min_k \left\{ w_k + \sum_{i \in V} \left(d_i^k - 2\right)\lambda_i \right\}.$$

Так как (3.43) означает, что $l^* \geq w(\lambda)$ для любого $\lambda \in R^n$, лучшая оценка снизу на длину оптимального маршрута, связанная с 1-деревьями, находится решением задачи

$$w(\lambda) \to \max, \ \lambda \in R^n . \qquad (3.44)$$

Свяжем теперь нижнюю границу 1-дерева с Лагранжевой релаксацией следующей формулировки задачи коммивояжера. Как и в п. 2.1 полагаем, что c_e – стоимость ребра $e \in E$ и пусть x_e – переменная, которая принимает значение 1, если оптимальный маршрут включает это ребро, и равна нулю в противном случае. Для заданного подмножества $S \subset V$ через $E(S)$ обозначается множество, имеющих обе граничные точки в S, через $\delta(S)$– разрез, разделяющий S и $V \setminus S$. Задача коммивояжера имеет вид

$$\sum_{e \in E} c_e x_e \to \min \qquad (3.45)$$

$$\sum_{e \in \delta(i)} x_e = 2, \ i = 1,2,...,n, \qquad (3.46)$$

$$\sum_{e \in E(S)} x_e \leq |S| - 1, \ S \subseteq V \setminus \{1\}, \ S \neq \varnothing, \qquad (3.47)$$

$$x_e \in \{0,1\},\ e \in E. \tag{3.48}$$

Ограничения (3.46) гарантируют, что степень каждой вершины равна 2, условия (3.47) – ограничения устранения подциклов.

Условия (3.46) можно заменить следующими ограничениями:

$$\sum_{e \in \delta(i)} x_e = 2,\ i = 2,3,...,n, \tag{3.49}$$

$$\sum_{e \in E} x_e = n. \tag{3.50}$$

В самом деле, достаточно показать, что ограничение

$$\sum_{e \in \delta(1)} x_e = 2 \tag{3.51}$$

эквивалентно ограничению (3.50):

$$\sum_{e \in E} x_e = \frac{1}{2}\sum_{i=1}^{n}\sum_{e \in \delta(i)} x_e = \frac{1}{2}\sum_{i=2}^{n}\sum_{e \in \delta(i)} x_e + \frac{1}{2}\sum_{e \in \delta(1)} x_e = (n-1) + \frac{1}{2}\sum_{e \in \delta(1)} x_e.$$

Таким образом, $\sum_{e \in E} x_e = n$ тогда и только тогда, когда $\sum_{e \in \delta(1)} x_e = 2$.

Добавим ограничение (3.51) к задаче (3.45), (3.47)-(3.50) и рассмотрим Лагранжеву релаксацию ограничений (3.49):

$$\sum_{e \in E} c_e x_e + \sum_{i=2}^{n} \lambda_i \left(\sum_{e \in \delta(i)} x_e - 2 \right) \to \min$$

при условиях (3.47), (3.48), (3.50), (3.51).

Ограничения релаксированной задачи определяют 1-дерево, а целевая функция может быть записана в виде

$$\sum_{i,j} \left(c_{ij} + \lambda_i + \lambda_j\right) x_{ij} - 2\sum_{i=2}^{n} \lambda_i,$$

то есть оптимальное значение целевой функции задачи равно $w(\lambda)$.

Для решения (3.44) может быть использован субградиентный метод. То есть, начиная с некоторого произвольного вектора λ^0, на шаге k метода пересчитываются компоненты вектора λ^k по формуле

$$\lambda_i^{k+1} = \lambda_i^k + t_k\left(d_i^k - 2\right),$$

где d_i^k – степень вершины i в минимальном 1-дереве на шаге k, размер шага равен, например,

$$t_k = \frac{\theta_k\left(UB - w\left(\lambda^k\right)\right)}{\sum_{i=1}^{n}\left(d_i^k - 2\right)^2}.$$

На каждом шаге решения задачи (3.44) субградиентным методом на величину $t_k\left(d_i^k - 2\right)$ штрафуются вершины, степень которых отличается от 2. В [8] предлагаются и другие стратегии штрафования вершин. В частности, рассмотрим *вычисляемые штрафы*.

Для вершины i такой, что $d_i^k > 2$, определяется минимальный положительный штраф, уменьшающий степень вершины на единицу после одной итерации. Предположим, из 1-дерева исключается ребро $[i, j]$.

Если $j \neq 1$, дерево на вершинах $\{2,...,n\}$ распадается на два поддерева – T_1 и T_2, $i \in T_1$. Ребро $[k,s]$, соединяющее T_1 и T_2 ($k \in T_1$, $s \in T_2$), будет включено в дерево, если после преобразования его длина будет не больше длины ребра $[i, j]$. То есть, если $k \neq i$, $c_{ks} \leq c_{ij} + \lambda_i$, $\lambda_i \geq c_{ks} - c_{ij}$. Если $k = i$, $c_{is} + \lambda_i \leq c_{ij} + \lambda_i$ и такая ситуация возможна, только если $c_{is} = c_{ij}$. Следовательно,

$$\lambda_{ij} = \min_{k \in T_1, s \in T_2} c_{ks} - c_{ij}.$$

Аналогично, если $j = 1$, $\lambda_{i1} = \min_{s \neq i} c_{1s} - c_{1i}$. Окончательный штраф определяется как

$$\lambda_i = \min_j \lambda_{ij}.$$

Рассмотрим метод штрафования вершины i такой, что $d_i^k = 1$. В этом случае штраф отрицательный и вычисляется как максимальный штраф, который после одной итерации увеличивает степень вершины на 1. При добавлении ребра $[i, j]$, $j \neq 1$ образуется единственный цикл C_{ij}. Ребро $[k,s] \in C_{ij}$ после преобразования матрицы стоимостей исключается из дерева, если $c_{ij} + \lambda_i \leq c_{ks}$, то есть максимальный штраф, добавляющий ребро $[i, j]$, равен $\lambda_{ij} = \max_{[k,s] \in C_{ij}} c_{ks} - c_{ij}$. Для добавления ребра $[i,1]$ необходим штраф $\lambda_{i1} = \max_s c_{1s} - c_{i1}$. Таким образом,

$$\lambda_i = \max_j \lambda_{ij}.$$

Если все вычисленные указанным образом штрафы накладываются одновременно, то предсказать изменение степеней вершин не представляется возможным.

Исследования показывают, что в большинстве случаев методы штрафования вершин сходятся, то есть найдётся такой вектор $\lambda \in R^n$, что для преобразованной по правилу (3.42) матрицы длин дуг кратчайшее 1-дерево является гамильтоновым циклом. Но даже если $l^* - \max w(\lambda) > 0$, полученная оценка на длину кратчайшего маршрута может быть эффективно использована, например, при реализации метода ветвей и границ.

В [15] отмечается, что линейная релаксация задачи (3.45)–(3.45) обладает свойством целочисленности, следовательно, согласно замечанию 3.2, $f_D \equiv \max w(\lambda) = f_{LP}$. Кроме того, справедливо следующее утверждение.

Теорема 3.5. *Для любой симметричной задачи коммивояжера с матрицей длин дуг, удовлетворяющей неравенству треугольника,*

$$l^* \le \frac{3}{2} f_D .$$

Пример 3.1. Рассмотрим задачу коммивояжера из примера 2.11. Минимальное 1-дерево изображено на рис. 3.1, его вес равен 120.

Вершина 4 имеет степень 3. При исключении ребра $[4,3]$ для объединения двух поддеревьев в одно необходим штраф $\lambda_{43} = \min\{c_{53}, c_{23}\} - c_{43} = 38 - 14 = 24$, при исключении $[4,5]$ – $\lambda_{45} = \min\{c_{35}, c_{32}\} - c_{45} = 38 - 24 = 14$, аналогично, $\lambda_{41} = \min\{c_{13}, c_{15}\} - c_{41} = 31 - 22 = 9$. Таким образом, $\lambda_4 = 9$.

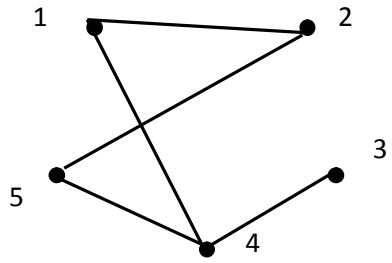


Рис. 3.1. 1-дерево для примера 3.1

Вершина 3 имеет степень 1. Добавление ребра $[3,1]$ осуществится при использовании штрафа $\lambda_{31} = \max\{c_{12}, c_{14}\} - c_{31} = 28 - 31 = -3$, ребра $[3,2]$ – при штрафе $\lambda_{32} = \max\{c_{25}, c_{45}\} - c_{32} = 32 - 40 = -8$, ребра $[3,5]$ – при штрафе $\lambda_{35} = c_{45} - c_{35} = 24 - 38 = -14$. Таким образом, $\lambda_3 = -3$.

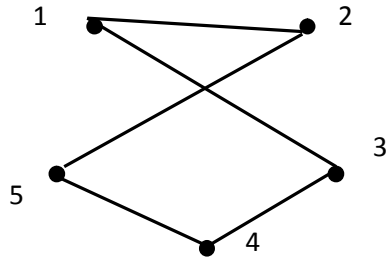


Рис. 3.2. Результат штрафования вершины 4

Штрафуем вершину 4 на 9, прибавляя 9 к элементам четвёртой строки и четвёртого столбца матрицы стоимостей:

$$\begin{pmatrix} - & 28 & 31 & 31 & 36 \\ 28 & - & 40 & 50 & 32 \\ 31 & 40 & - & 23 & 38 \\ 31 & 50 & 23 & - & 33 \\ 36 & 32 & 38 & 33 & - \end{pmatrix}.$$

Минимальное 1-дерево (рис. 3.2) является маршрутом коммивояжера, следовательно, найдено решение задачи. Длина оптимального маршрута равна 137-18=129.

Этот же результат получим, одновременно штрафуя четвертую и третью вершины.

Штрафование только вершины 3 на величину -3 приводит к 1-дереву на рис. 3.3, имеющему вес 117. Таким образом, имеем оценку снизу на оптимальную длину маршрута коммивояжера 117+6=123. Для дальнейшего улучшения этой оценки следует наложить рассчитанный по новой матрице стоимостей штраф на вершину 2 либо на вершину 4.

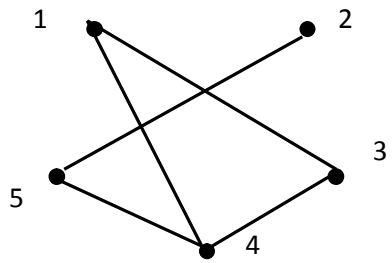


Рис. 3.3. Результат штрафования вершины 3

3.5. Алгоритм решения задачи о p-медиане

Медианой в графе называется вершина, сумма расстояний от которой до всех остальных вершин графа минимально. Обобщением задачи поиска медианы является задача о кратной медиане или *p*-медиане. В графе $G=(V,E)$ требуется выбрать множество $S \subset V$ таким образом, чтобы $|S|=p$ и сумма расстояний от всех вершин до ближайших вершин из множества S была наименьшей.

Основным приложением задач поиска медиан являются задачи размещения пунктов обслуживания, например, складов, почтовых отделений и так далее. Предположим, нужно выбрать p мест из m возможных для размещения в них складов, осуществляющих поставки n потребителям, скажем – розничным магазинам. Стоимость транспортировки продукции от склада в пункте j к потребителю i равна c_{ij} для всех $j=1,...,m$, $i=1,...,n$.

Пусть переменная y_j принимает значение 1, если в пункте j размещается склад (соответствующая вершина принадлежит p-медиане) и 0 в противном случае, $j=1,...,m$. Переменная x_{ij} при $i=1,...,n$, $j=1,...,m$, равна 1, если пункт i прикреплен к вершине j и нулю в противном случае. Задача о *p*-медиане имеет, таким образом, следующий вид:

$$\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} \to \min \tag{3.52}$$

$$\sum_{j=1}^{m} x_{ij}=1,\ i=1,...,n, \tag{3.53}$$

$$\sum_{j=1}^{m} y_j = p, \tag{3.54}$$

$$x_{ij} \le y_j,\ i = 1,...,n,\ j = 1,...,m, \tag{3.55}$$

$$x_{ij}, y_j \in \{0,1\},\ i = 1,...,n,\ j = 1,...,m. \tag{3.56}$$

Ограничения (3.53) гарантируют, что каждая вершина $i = 1,...,n$ прикреплена к одной из вершин $j = 1,...,m$, (3.54) означает, что выбраны p точек из m. Условия (3.55) показывают, что вершины могут прикрепляться только к медианным вершинам (магазин может получать поставки из пункта j, если там есть склад).

Рассмотрим метод решения поставленной задачи, основанный на Лагранжевой релаксации ограничений (3.53). Для произвольного $\lambda \in R^n$ получаем задачу

$$\sum_{i=1}^{n}\sum_{j=1}^{m} c_{ij}x_{ij} - \sum_{i=1}^{n} \lambda_i \left(\sum_{j=1}^{m} x_{ij} - 1 \right) \to \min \tag{3.57}$$

при ограничениях (3.54)–(3.56).

Без учета ограничения (3.54) задача (3.57), (3.55), (3.56) разбивается на m подзадач (для $j = 1,...,m$)

$$\sum_{i=1}^{n} \left(c_{ij} - \lambda_i \right) x_{ij} \to \min \tag{3.58}$$

$$x_{ij} \le y_j,\ i = 1,...,n, \tag{3.59}$$

$$x_{ij} \in \{0,1\},\ i = 1,...,n,\ y_j \in \{0,1\}. \tag{3.60}$$

Если в задаче (3.58)–(3.60) положить $y_j = 0$, то все $x_{ij} = 0$ и, следовательно, равно нулю значение целевой функции. Если же $y_j = 1$, то неравенства (3.59) выполняются автоматически, и следует выбирать $x_{ij} = 1$, если $c_{ij} - \lambda_i < 0$ и $x_{ij} = 0$ в противном случае. Таким образом, минимальное значение целевой функции задачи (3.58)–(3.60)

$$f_\lambda^j = \sum_{i=1}^{n} \min \{ c_{ij} - \lambda_i, 0 \}.$$

Чтобы решить задачу (3.57), (3.54)–(3.56), нужно учесть ограничения (3.54), то есть выбрать p точек из m, имеющих наименьшие значения f_λ^j. Пусть $\pi = (\pi(1),...,\pi(m))$ – перестановка чисел 1,....,m такая, что

$$f_\lambda^{\pi(1)} \le f_\lambda^{\pi(2)} \le ... \le f_\lambda^{\pi(m)}.$$

Тогда оптимальное значение целевой функции задачи (3.57), (3.54)–(3.56) равно

$$f(\lambda) = \sum_{j=1}^{p} f_\lambda^{\pi(j)} + \sum_{j=1}^{n} \lambda_j .$$

$f(\lambda)$ при любом $\lambda \in R^n$ оценивает снизу оптимальное значение целевой функции задачи (3.52)–(3.56). Для определения лучшей такой нижней границы решается двойственная задача

$$f(\lambda) \to \max,\ \lambda \in R^n.$$

На каждом шаге субградиентной процедуры определено множество из p точек $S = \{\pi(1),...,\pi(p)\}$ и матрица $X = (x_{ij})$, где j-й столбец при $j \in S$ – решение задачи (3.58)–(3.60) и $x_{ij} = 0$ при $j \notin S$, $i = 1,...,n$. Если выполнены ограничения (3.53), то получено оптимальное решение задачи о p-медиане. В противном случае можно построить допустимое решение задачи, прикрепляя пункты к ближайшим точкам из множества S. Погрешность этого решения оценивается как $UB - f(\lambda)$.

Алгоритм завершается либо при получении оптимального решения, либо при достижении заданной погрешности, либо после предписанного числа итераций.

Пример 3.2. Найдём 2-медиану в графе на рис. 3.4. Считаем, что любая вершина может быть медианной.

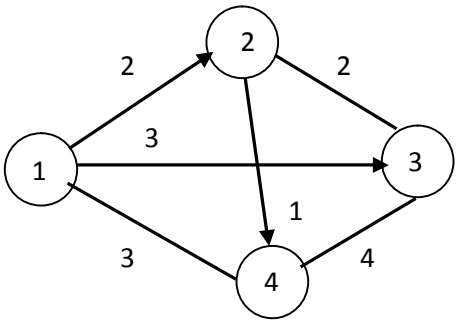


Рис. 3.4. Граф для примера 3.2

В качестве стоимостей c_{ij} рассмотрим расстояния, то есть длины кратчайших путей, между вершинами:

$$C = \begin{pmatrix} 0 & 2 & 3 & 3 \\ 4 & 0 & 2 & 1 \\ 6 & 2 & 0 & 3 \\ 3 & 5 & 4 & 0 \end{pmatrix}.$$

Выберем начальный вектор $\lambda^0 = (4,4,4,4)'$ и преобразуем матрицу стоимостей, вычитая из строки i соответствующий элемент λ_i:

$$C_0 = \begin{pmatrix} -4 & -2 & -1 & -1 \\ 0 & -4 & -2 & -3 \\ 2 & -2 & -4 & -1 \\ -1 & 1 & 0 & -4 \end{pmatrix}.$$

Суммы отрицательных элементов столбцов матрицы C_0, то есть f_λ^j, равны, соответственно, -5, -8, -7, -9. Выбираем вершины 2 и 4, $f\left(\lambda^0\right)=-8-9+16=-1$, матрица X_0 имеет вид

$$X_0=\begin{pmatrix}0&1&0&1\\0&1&0&1\\0&1&0&1\\0&0&0&1\end{pmatrix}.$$

Вектор невязки – $(1,1,1,0)$. Допустимое решение получим, присоединяя каждую вершину к ближайшей из второй или четвёртой, $UB=4$. Выберем размер шага, равный 2. Тогда $\lambda^1=(2,2,2,4)'$,

$$C_1=\begin{pmatrix}-2&0&1&-1\\2&-2&0&-1\\4&0&-2&1\\-1&1&0&-4\end{pmatrix}.$$

Выбранные вершины – 1 и 4, $f\left(\lambda^1\right)=-3-6+10=1$,

$$X_1=\begin{pmatrix}1&0&0&1\\0&0&0&1\\0&0&0&0\\1&0&0&1\end{pmatrix}.$$

Стоимость допустимого решения не изменилась, вектор невязки – $(1,0,-1,1)$. Выбираем шаг, равный 1. $\lambda^2=(1,2,3,5)'$,

$$C_2=\begin{pmatrix}-1&1&2&0\\2&-2&0&-1\\3&-1&-3&0\\0&2&1&-3\end{pmatrix}.$$

Вершины с наименьшей оценкой f_λ^j – 3 и 4, $f\left(\lambda^2\right)=-3-4+11=4=UB$, соответствующее решение является допустимым и, следовательно, оптимальным.

4. УПРАЖНЕНИЯ

1. Найдите кратчайший путь из вершины 1 в вершину 7 в графе на рис. 4.1.

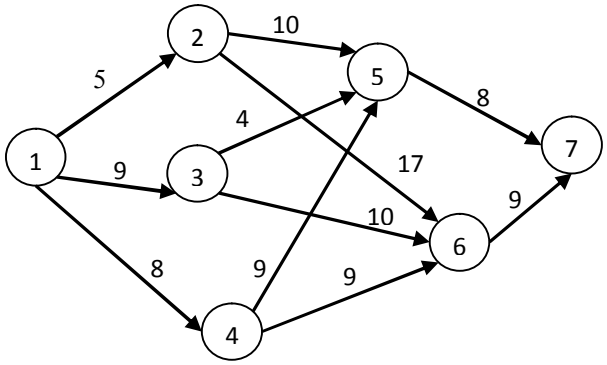


Рис.4.1. Граф для упражнения 1

2. Для графа с n вершинами, в котором допускаются отрицательные длины дуг, обозначим через l_j^k длину кратчайшего пути из начальной вершины в вершину j, использующего не более k дуг. Запишите начальные условия и рекуррентные соотношения для вычисления l_j^k. Покажите, что если при переходе от этапа k к этапу $k+1$ ни одно l_j^k не меняется, найдены кратчайшие пути из начальной вершины во все остальные. Покажите, что если после n этапов алгоритм не завершён в указанном смысле, граф содержит цикл отрицательной длины.

3. Турист собирается в путешествие по дикой местности и должен упаковать в рюкзак предметы трех видов: пищу, средства первой помощи и одежду. Объём рюкзака составляет 3 кубических фута. Каждая единица пищи занимает 1 кубический фут, упаковка средств первой помощи – четверть кубического фута, а отдельный предмет одежды – примерно половину кубического фунта. Турист определил свои предпочтения весовыми коэффициентами 3, 4 и 5 – для пищи, средств первой помощи и одежды соответственно. Опыт подсказывает туристу, что он должен взять не менее одного предмета каждого вида и не более двух комплектов средств первой помощи. Сколько единиц каждого наименования возьмёт турист в поход?

4. Конструируется электронный прибор, состоящий из трёх основных компонентов. Все компоненты соединены последовательно, поэтому выход из строя одного из них влечёт за собой отказ всего прибора. Надёжность (вероятность безаварийной работы) прибора можно повысить путём дублирования компонентов. Допускается использование одного или двух резервных блоков для каждого компонента. В таблице 4.1 содержатся данные о надежности r_i и стоимости c_i компонентов. Надежность прибора оценивается как $r_1 r_2 r_3$. Общая

сумма, выделенная на конструирование прибора, равна 10000 у.е. Как следует сконструировать прибор?

Таблица 4.1

Данные для упражнения 4

Число параллельных блоков	Компонент 1		Компонент 2		Компонент 3	
	r_1	c_1	r_2	c_2	r_3	c_3
1	0.6	1000	0.7	3000	0.5	2000
2	0.8	2000	0.8	5000	0.7	4000
3	0.9	3000	0.9	6000	0.9	5000

5. Фермер имеет k овец. В конце каждого года он принимает решение, сколько овец продать и сколько оставить. Прибыль от продажи одной овцы в год i равна p_i. Количество овец в конце i-го года удваивается к концу (i+1)-го года. В конце n-го года планируется полностью продать овец. Получите рекуррентное уравнение для решения задачи и решите задачу при $n=3$, $k=2$, $p=(100,130,120)$.

6. В начале каждого из следующих четырёх лет необходимо сделать инвестиции в 5, 4, 3 и 2 тыс. у.е. соответственно. Есть возможность вложить капитал в 2 банка, первый выплачивает годовой сложный процент 8.5%, второй – 8%. Для поощрения депозитов оба банка выплачивают новым инвесторам премии в виде процента от вложенной суммы. Премиальные меняются от года к году, и у первого банка равны 1.8, 1.7, 2.1 и 2.5 процентов соответственно, у второго банка премиальные на 0.5% выше. Премиальные выплачиваются в конце года, на протяжении которого сделан вклад, и могут быть инвестированы в один из двух банков на следующий год. Размещённый в банке вклад должен находится там до конца рассматриваемого периода. Необходимо разработать стратегию инвестиций, максимизирующую накопленный капитал к концу четвёртого года.

7. Строительный подрядчик оценивает минимальные потребности в рабочей силе на каждую из последующих пяти недель следующим образом: 6, 5, 3, 6 и 8 рабочих соответственно. Содержание избытка рабочей силы обходится подрядчику в 300 у.е за одного рабочего в неделю, а наём рабочей силы на протяжении одной недели обходится в 400 у.е. плюс 200 у.е. за одного рабочего в неделю. Каждому уволенному рабочему выплачивается выходное пособие в размере 100 у.е. Сформулируйте задачу оптимального найма как задачу о кратчайшем пути и найдите её решение.

8. Рассмотрите задачу замены оборудования на протяжении n лет. Цена новой единицы оборудования равна c, стоимость продажи после t лет оборудования равна $s(t)$, годовая прибыль от эксплуатации является функцией возраста оборудования и равна $r(t)$. Сформулируйте задачу поиска оптимальной стратегии замены оборудования как задачу динамического программирования. Решите задачу, считая, что $n=4$, возраст оборудования составляет 1 год, $c=6000$ у.е., $s(t)=5-t$ при $t<5$ и 0 в противном случае, $r(t)=25-t^2$ при $t<5$ и 0 в противном случае.

9. Начальное количество средств 10 нужно распределять в течение 5 лет между двумя отраслями производства 1 и 2. Средства x, вложенные в первую отрасль, приносят доход x^2, однако уменьшаются при этом до $0.75x$. Аналогично, доход второй отрасли составляет $2x^2$, средства уменьшаются до $0.3x$. По истечении года оставшиеся средства заново распределяются между отраслями. Средств извне не поступает, в производство вкладываются все оставшиеся средства. Доход не вкладывается, накапливается отдельно. Требуется найти способ управления ресурсами, при котором суммарный доход обеих отраслей за 5 лет будет максимальным.

10. Решить задачу с помощью метода динамического программирования

$$x_1 x_2 \ldots x_n \to \max$$
$$x_1 + x_2 + \ldots + x_n = c,$$
$$x_i \geq 0,\ i = 1,\ldots,n.$$

11. Решить задачу с помощью метода динамического программирования

$$x_1^2 + x_2^2 + \ldots + x_n^2 \to \min$$
$$x_1 x_2 \ldots x_n = c,$$
$$x_i \geq 0,\ i = 1,\ldots,n.$$

12. Решить задачу с помощью метода динамического программирования

$$\max\{x_1 + 5{,}5x_2 + 3, x_3 - 2\} \to \min$$
$$x_1 + x_2 + x_3 = c,$$
$$x_i \geq 0,\ i = 1,2,3.$$

13. Рассмотрите следующий алгоритм динамического программирования для решения задачи коммивояжера. Пусть $f_i(j,S)$ – длина кратчайшего пути от вершины 1 до вершины j, проходящего через вершины множества S, $|S| = i$. Определите рекуррентную формулу и решите задачу с матрицей длин дуг

$$\begin{pmatrix} - & 3 & 1 & 5 & 4 \\ 1 & - & 5 & 4 & 3 \\ 5 & 4 & - & 2 & 1 \\ 3 & 1 & 3 & - & 3 \\ 5 & 2 & 4 & 1 & - \end{pmatrix}.$$

14. Докажите, что если граф имеет $2k$ вершины нечётной степени, то множество рёбер разбивается на k путей, так что каждое ребро принадлежит точно одному пути.

15. Докажите лемму 2.1.

16. Пусть T – дерево с n вершинами. Докажите, что существует по крайней мере две вершины со степенью 1.

17. Докажите лемму 2.3.

18. Показать, что минимальное остовное дерево T удовлетворяет следующему свойству. Для любого другого остовного дерева T' k-е кратчайшее ребро в T не длиннее k-го кратчайшего ребра в T', $k = 1,...,n-1$.

19. Компания производит 15 типов изделий на 10 станках. Планируется сгруппировать станки так, чтобы минимизировать «несходство» операций, выполняемых на каждой группе станков. Мерой «несходства» между станками i и j служит величина

$$d_{ij} = 1 - \frac{n_{ij}}{n_{ij} + m_{ij}},$$

где n_{ij} – количество изделий, обрабатываемых как на станке i, так и на станке j, m_{ij} – количество изделий, обрабатываемых только на станке i или только на станке j. На первом станке могут обрабатываться 1 и 6 типы изделий, на втором – 2, 3, 7, 8, 9, 12,13,15, на третьем – 3,5,10,14, на четвёртом – 2, 7, 8, 11, 12, 13, на пятом – 3, 5, 10, 11, 14, на шестом – 1, 4, 5, 9, 10, на седьмом – 2, 5, 7, 9, 10, на восьмом – 3, 4, 15, на девятом – 4 и 10, на десятом – 3, 8, 10, 14, 15 типы изделий. Покажите, что данную задачу можно свести к задаче поиска минимального остовного дерева. Решите для разбиения станков на две и три группы.

20. Задача двух коммивояжеров – задача построения двух маршрутов минимальной совокупной длины, начинающихся в одной точке и таких, что каждая вершина кроме начальной принадлежит точно одному маршруту. Показать, что любой алгоритм для решения задачи коммивояжера может быть приспособлен для решения этой задачи.

21. Рассмотрите задачу коммивояжера, в которой $c_{ij} = a_i + b_j$ для всех i, $j = 1,...,n$. Чему равна длина оптимального маршрута в этой задаче?

22. В открытой задаче коммивояжера требуется построить маршрут минимальной длины, проходящий через все вершины графа и начинающийся и заканчивающийся в любых вершинах. Покажите, как любую открытую задачу коммивояжера свести к обычной задаче коммивояжера и наоборот.

23. Предположим, n различных работ (заданий) требуется выполнить на m машинах. Каждое задание выполняется сначала на машине 1, затем на машине 2 и так далее. Каждая машина может выполнять не более одного задания одновременно, и, начиная работу, работает до её завершения, без прерываний. Количество времени, которое требуется машине j для работы i равно $p_{ij} > 0$. Предполагается, что когда выполнение задания завершается на машине j, то сразу же начинается на машине $j+1$. Покажите, что задача упорядочения работ, при котором выполнение последнего задания заканчивается как можно раньше, может быть сформулирована как задача коммивояжера с $n+1$ вершиной.

24. Докажите лемму 2.4.

25. Рассмотрите задачу коммивояжера с матрицей длин дуг, удовлетворяющей неравенству треугольника, и двумя предписанными вершинами s и t –

каждый маршрут коммивояжера должен содержать дугу (s,t). Модифицировать алгоритм Кристофидеса для этой задачи и показать, что гарантированная оценка равна 3/2.

26. Рассматривается симметричная задача коммивояжера с матрицей длин дуг, удовлетворяющей неравенству треугольника. Пусть длина оптимального маршрута – $l^* > 0$, следующего по величине маршрута – $l' > 0$. Докажите, что $(l'-l^*)/l^* \le 2/n$.

27. Доказать, что константа упаковки γ удовлетворяет неравенству $1 \le \gamma / E(w) \le 2$, где $E(w)$ – ожидаемый размер предметов.

28. Рассмотрите следующий алгоритм для задачи об упаковке, который называется гармоническим алгоритмом с параметром M и обозначается $H(M)$. Для каждого $k = 1,...,M-1$ предметы размера $\frac{1}{k+1} < w_i \le \frac{1}{k}$ упаковываются не более чем по k предметов в контейнер. То есть предметы с размером большим, чем ½, упаковываются по одному, предметы размером $1/3 < w_i \le 1/2$ упаковываются по два в контейнер и так далее. Наконец, предметы размера $w_i \le 1/M$ упаковываются с использованием Правила Первого Подходящего. Даны n предметов, размеры равномерно распределены на $(1/6,1]$. Каково асимптотическое число контейнеров, используемых $H(5)$?

29. Покажите, что нижнюю границу на длину оптимального маршрута коммивояжера можно определить как

$$\frac{2}{n}\max_i \sum_j d_{ij},$$

где d_{ij} – длина кратчайшего пути от вершины i до вершины j.

30. Рассмотрим задачу об упаковке, в которой m_j предметов имеют размер $w_j \in (0,1]$, $j = 1,...,n$. Конфигурацией контейнера назовём целочисленный вектор $\bar{c} = (c_1,...,c_n)$ такой, что $c_i \ge 0$ для $i = 1,...,n$ и $\sum_{j=1}^{n} c_j w_j \le 1$. Пусть всех возможных конфигураций M и они перенумерованы. Через C_{jk} обозначим количество предметов размера w_j в конфигурации k, где $k = 1,...,M$ и $j = 1,...,n$. Сформулируйте задачу целочисленного программирования для решения задачи об упаковке, используя следующие переменные: x_k – число раз использования конфигурации k, $k = 1,...,M$.

31. Рассмотрите список L из n чисел, принадлежащих $(1/3,1/2]$. Пусть b^{LP} – оптимальное дробное решение формулировки разбиения множества для задачи об упаковке, b^* – оптимальное целочисленное решение той же задачи. Докажите, что $b^* \le b^{LP} + 1$.

32. Докажите лемму 3.2.

33. Докажите утверждение замечания 3.1.

34. Докажите теорему 3.1 для релаксации ограничений типа неравенства.

35. Докажите лемму 3.3.

36. Разработайте алгоритм ветвей и границ для симметричной задачи коммивояжера, основанный на нижних границах 1-дерева. Решите с помощью этого алгоритма задачу из примера 2.11.

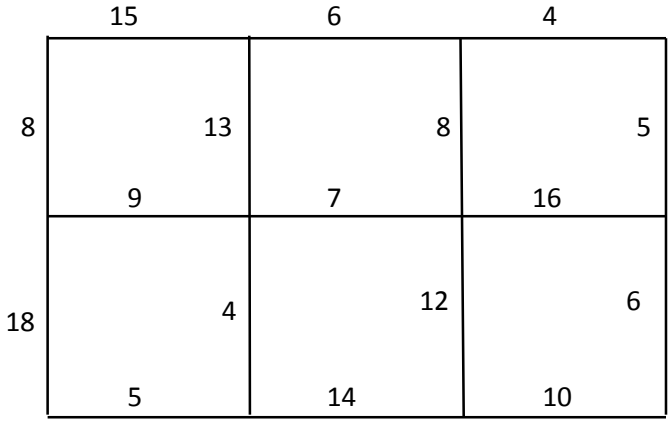


Рис. 4.2. Граф для упражнения 38

37. Разработайте алгоритм решения задачи о *p*-медиане, основанный на генерации столбцов.

38. Найдите 2-медиану и 3-медиану в графе, изображённом на рис. 4.2.

СПИСОК ЛИТЕРАТУРЫ

1. Ашманов С.А. Линейное программирование. – М.: Наука, 1981. – 340 с.
2. Беллман Р., Калаба Р. Динамическое программирование и современная теория управления. – М.: Наука, 1969. – 120 с.
3. Васильев Ф.П. Методы оптимизации. – М.: Факториал Пресс, 2002. – 824 с.
4. Венцель Е.С. Исследование операций. 4-е изд. – М.: Высшая школа, 2007. – 208 с.
5. Зайченко Ю.П., Шумилова С.А. Исследование операций: Сборник задач. – Киев: Выща шк., 1990. – 239 с.
6. Конюховский П.В. Математические методы исследования операций в экономике. – СПб.: Питер, 2002. – 208 с.
7. Корбут А.А., Финкельштейн Ю.Ю. Дискретное программирование. – М.: Наука, 1969. – 368 с.
8. Кристофидес Н. Теория графов. Алгоритмический подход. – М.: Мир, 1978. – 432 с.
9. Мину М. Математическое программирование. Теория и алгоритмы. – М.: Наука, 1990. – 488 с.
10. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. – М.: Мир, 1985. – 512 с.
11. Сигал И.Х., Иванова А.П. Введение в прикладное дискретное программирование: модели и вычислительные алгоритмы. – М.: ФИЗМАТЛИТ, 2007. – 304 с.
12. Таха Х. Введение в исследование операций. – М.: Вильямс, 2005. – 912 с.
13. Тюхтина А.А. Методы дискретной оптимизации: Часть 1: Учебно-методическое пособие. [Электронный ресурс] – Нижний Новгород: Нижегородский госуниверситет, 2014. – 62 с. Режим доступа: http://www.unn.ru/books/met_files/mdo.pdf, свободный.
14. Ху Т.Ч., Шинг М.Т. Комбинаторные алгоритмы. – Нижний Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2004. – 330 с.
15. Bramel J., Simchi-Levi D. The logic of logistics: theory, algorithms, and applications for logistics management. – New York: Springer – Verlag, 1997. – 281 p.

Алла Александровна **Тюхтина**

Методы дискретной оптимизации

Часть 2

Учебно-методическое пособие

Федеральное государственное автономное образовательное учреждение
высшего образования
«Нижегородский государственный университет им. Н.И. Лобачевского».
603950, Нижний Новгород, пр. Гагарина, 23.