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

А. И. Ерзин, Ю. А. Кочетов

ЗАДАЧИ МАРШРУТИЗАЦИИ

Учебное пособие

Новосибирск
2014

ББК В173
УДК 519.8
Е 709

Рецензент
канд. физ.-мат. наук, доц. А. В. Плясунов

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

Е 709 **Ерзин, А. И.**
Задачи маршрутизации : учеб. пособие / А. И. Ерзин, Ю. А. Кочетов; Новосиб. гос. ун-т. – Новосибирск : РИЦ НГУ, 2014. – 95 с.

ISBN 978-5-4437-0275-9

Учебное пособие содержит материал по разделу основного курса «Исследование операций», читаемого авторами на двух потоках механико-математического факультета Новосибирского госуниверситета и посвященного методам поддержки принятия оптимальных решений. Пособие содержит необходимые определения, утверждения, алгоритмы, примеры и упражнения. Пособие включает в себя разделы по синтезу остовных деревьев и деревьев Штейнера, а также по построению оптимальных путей и контуров.

Предназначено для студентов механико-математического факультета НГУ, а также для всех, кто желает освоить курс самостоятельно.

ББК В173
УДК 519.8

ISBN 978-5-4437-0275-9

ОГЛАВЛЕНИЕ

Введение

В 2000 г. профессор Тошиюки Накагаки, биолог и физик из японского университета Хоккайдо, взял образец желтого плесневого гриба Physarum polycephalum и положил его у входа в лабиринт, который используется для проверки интеллекта и памяти мышей. В другой конец лабиринта он поместил кубик сахара.

Physarum polycephalum словно почувствовал запах сахара и начал посылать свои ростки на его поиски. Паутинки гриба раздваивались на каждом перекрестке лабиринта, и те из них, которые попадали в тупик, разворачивались и начинали двигаться в других направлениях. В течение нескольких часов грибные паутинки заполнили проходы лабиринта, и к концу дня одна из них нашла дорогу к сахару.

После этого Тошиюки и группа его исследователей взяли кусочек паутинки гриба, участвовавшей в первом опыте, и положили его у входа копии того же лабиринта, также с кубиком сахара на другом его конце. Произошедшее поразило всех. Паутинка разветвилась на две: один отросток проложил свой путь к сахару, без единого лишнего поворота, другой – вскарабкался по стене лабиринта и пересек его напрямую, по потолку, прямо к цели. Грибная паутинка не только запомнила дорогу, но и изменила правила игры.

Дальнейшие исследования Тошиюки установили, что грибы могут планировать транспортные маршруты не хуже и намного быстрее инженеров-профессионалов. Тошиюки взял карту Японии и поместил кусочки пищи в местах, соответствующих крупным городам страны. Грибы он положил «на Токио». Спустя 23 часа они построили сеть паутинок ко всем кусочкам пищи. В результате получилась почти точная копия железнодорожной сети в окрестностях Токио.

«Не так уж сложно соединить несколько десятков точек; а вот соединить их эффективно и наиболее экономно – это уже совсем не просто. Я верю, что наши исследования не только помогут понять, как улучшать инфраструктуру, но и как строить более эффективные информационные сети», – писал Тошиюки Накагаки.

Данное пособие посвящено изучению алгоритмов, которые позволяют эффективно строить оптимальные маршруты – подграфы

заданных графов, обладающие заданными экстремальными свойствами. Остовные деревья, кратчайшие пути в графах, коммуникационные сети и гамильтоновы циклы будут в этом пособии в центре внимания. Наряду с классическими алгоритмами дискретной оптимизации, мы познакомимся и с новыми подходами к построению алгоритмов, заимствованными у природы: генетические алгоритмы, имитация отжига и др.

Пособие состоит из четырех глав. Первая глава посвящена задачам построения остовных деревьев в заданном взвешенном графе. Во второй главе собраны задачи, связанные с построением кратчайших путей. Третья глава посвящена задаче Штейнера, анализу ее сложности и описанию приближенных алгоритмов ее решения. В заключительной 4 главе исследуется задача коммивояжера и различные подходы для ее решения.

Если не оговорено дополнительно, то далее будут использоваться следующие обозначения:

$\boldsymbol{Z_+}$ – множество целых неотрицательных чисел;

$\boldsymbol{Z_+^n}$ – множество *n*-мерных неотрицательных целочисленных векторов;

$\boldsymbol{N = \{1, 2, \dots, n\} \subset Z_+}$;

$\boldsymbol{R}$ – множество вещественных чисел;

$\boldsymbol{R_+ = \{x \in R,\ x \geq 0\}}$;

$\boldsymbol{R^n}$ – множество *n*-мерных вещественных векторов $\boldsymbol{x = (x_1, \dots, x_n)}$, $\boldsymbol{x_j \in R}$;

$\boldsymbol{R_+^n = \{x \in R^n, x_j \geq 0,\ j = 1, \dots, n\}}$;

$\boldsymbol{B^n}$ – множество *n*-мерных булевых векторов $\boldsymbol{x = (x_1, \dots, x_n)}$, $\boldsymbol{x_j \in \{0, 1\}}$;

$\boldsymbol{|M|}$ – мощность множества $\boldsymbol{M}$;

$\boldsymbol{\lfloor a \rfloor}$ – целая часть числа $\boldsymbol{a}$;

$\boldsymbol{\lceil a \rceil}$ – минимальное целое число, которое не меньше $\boldsymbol{a}$;

$\boldsymbol{a^+ = \max\{0, a\}}$;

$\boldsymbol{x^* = (x_1^*, \dots, x_n^*)}$ – оптимальное решение;

■ – конец доказательства.

Для удобства читателя напомним определения некоторых базовых понятий, которые понадобятся в дальнейшем.

Определение 1. *Графом* $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ называется пара множеств $\boldsymbol{V}$ и $\boldsymbol{E}$, где $\boldsymbol{V} = \{\boldsymbol{1}, \ldots, \boldsymbol{n}\}$ – множество вершин, а $\boldsymbol{E} = \{(\boldsymbol{i}, \boldsymbol{j}) \mid \boldsymbol{i}, \boldsymbol{j} \in \boldsymbol{V}\}$ – множество ребер графа. Для изображения вершины, как правило, используется точка, а для изображения ребра – отрезок линии между двумя точками. Упорядоченная пара вершин определяет *дугу* и изображается стрелкой, которая заканчивается в конечной вершине дуги.

Определение 2. Две вершины, связанные ребром, являются *смежными*. Если вершина является одним из концов ребра, то эти вершина и ребро *инцидентны* друг другу. *Степень вершины* равна числу инцидентных ей ребер. Степень графа совпадает с максимальной степенью вершин.

Определение 3. *Простой цепью* называется ациклический связный подграф степени 2. Ориентированная цепь, в которой все дуги ориентированы от начала к концу пути, называется *путем*.

Определение 4. Связный подграф, все вершины которого имеют степень 2, называется циклом. Цикл, включающий все вершины графа, называется *гамильтоновым*.

Определение 5. Граф, каждому ребру которого $(i, j) \in E$ приписано число – «вес», называется *взвешенным*. Сумма весов ребер, входящих в подграф, называется *весом* подграфа.

Определение 6. Количество символов в стандартном (двоичном) представлении данных индивидуальной задачи $X \in P$ назовем *длиной входа* и обозначим $L = L(X)$.

Определение 7. Пусть алгоритм A решает проблему P и $t_A(X)$ – количество элементарных операций (арифметические операции и операции сравнения), выполняемых алгоритмом A при решении индивидуальной задачи $X \in P$. Тогда функция

$$T_A(n) = \sup_{X \in P}\{t_A(X) : L(X) = n\}$$

называется *трудоемкостью* алгоритма A.

Алгоритм A называется *полиномиальным*, если его трудоемкость $T_A(n) = O(n^d)$, где d – целое положительное число.

Алгоритмы, трудоемкость которых не ограничена полиномом от длины входа, называются *экспоненциальными*.

В теории вычислительной сложности принято рассматривать задачи распознавания (свойств), то есть задачи, в которых возможным ответом является *«Да»* или *«Нет»*. Например, правда ли, что заданный граф является деревом? Среди задач распознавания принято выделять классы P и NP. Напомним, что класс P состоит из задач распознавания, разрешимых за полиномиальное время. Класс NP является более широким. Он включает в себя все задачи распознавания, в которых ответ *«Да»* может быть проверен за полиномиальное время. Задача принадлежит этому классу, если, даже не умея ее решать, можно «легко» проверить ответ, подглядев его или найдя в Интернете. При этом достаточно уметь проверять только ответ *«Да»*. Иногда проверка ответа *«Да»* может быть легче или сложнее проверки ответа *«Нет»*. Рассмотрим задачу о гамильтоновости графа: задан простой неориентированный граф, требуется узнать, содержит ли он гамильтонов цикл? Покажем, что эта задача принадлежит классу NP. Предположим, что граф действительно гамильтонов, и кто-то подсказал нам ответ, указав один из таких циклов. Можно ли проверить эту подсказку за полиномиальное время? Для этого нужно проверить, что указанный набор ребер образует простой цикл, и он покрывает все вершины. Очевидно, что это «легко» сделать и, следовательно, задача принадлежит классу NP. Заметим, что ответ «*Нет*» проверить здесь куда труднее. Говорят, что задача распознавания принадлежит классу co-NP, если ответ «*Нет*» можно проверить за полиномиальное время. Легко показать, что следующая задача принадлежит классу co-NP. Задан простой неориентированный граф, правда ли, что он не является гамильтоновым? Ответ «*Нет*» означает гамильтоновость графа, а это можно легко проверить.

В классе P можно проверить с полиномиальной трудоемкостью любой ответ, и, значит, $P \subseteq NP$. Доказать или опровергнуть обратное включение пока никому не удается. На сегодняшний день это одна из центральных проблем математики – «задача тысячелетия» (http://www.claymath.org/millennium/). За ее решение Американское математическое общество обещает приз – миллион долларов.

Многолетние интенсивные исследования подсказывают, что P ≠ NP. Косвенным доказательством этой гипотезы служит тот факт, что в классе NP обнаружены так называемые NP-полные задачи.

Определение 8. Задачу из класса NP называют *NP-полной*, если существование полиномиального алгоритма для ее решения влечет существование полиномиальных алгоритмов для всех задач из класса NP.

К настоящему времени известно огромное количество NP-полных задач [4, 14]. Однако ни для одной из них не удалось разработать точный полиномиальный алгоритм.

Определение 9. *Приближенным* алгоритмом решения задачи P является алгоритм, строящий допустимое решение. Оценкой точности приближенного алгоритма A назовем такое число $\varepsilon > 0$, что отношение $W_A(I)/W^*(I)$ не превосходит ε для любой индивидуальной задачи $I \in P$ на минимум и не менее ε для любой индивидуальной задачи $I \in P$ на максимум. Здесь $W^*(I)$ – значение целевой функции на оптимальном решении задачи I, а $W_A(I)$ – значение функционала на решении, построенном алгоритмом A. При этом алгоритм A называется ε – *приближенным*.

Глава 1. Построение остовных деревьев

Пусть задан простой неориентированный связный граф $G = (V, E)$ с множеством вершин $V = \{1, \ldots, n\}$ и множеством ребер E, $|E| = m$.

Определение 1.1. *Остовным подграфом* графа G называется *связный* подграф $O = (V, E_O)$, $E_O \subseteq E$, т. е. связный подграф, содержащий *все* вершины графа G.

Определение 1.2. *Остовным деревом* в графе G называется ациклический остовный подграф.

Очевидно, в остовном дереве любая пара вершин из V связана единственной простой цепью, и остовное дерево имеет минимальное число ребер среди всех остовных подграфов.

Число различных остовных деревьев графа экспоненциально зависит от количества вершин n. Например, в полном графе число различных остовных деревьев n^{n-2}. Этот факт доказан десятками различных способов. Если критерий выбора остовного дерева трудно формализуем или субъективен, то для выбора лучшего дерева потребуется перебрать их все. Алгоритм бесповторного просмотра всех остовных деревьев графа представлен, например, в [1].

1.1. Минимальное остовное дерево. Алгоритмы Прима, Краскала, Борувки

Определение 1.3. Остовное дерево, у которого сумма весов ребер минимальна, назовем *минимальным остовным деревом* (МОД).

Многие практические задачи сводятся к построению МОД. Например, пусть требуется связать заданное множество населенных пунктов сетью дорог таким образом, чтобы минимизировать связанные с этим затраты. Если известна стоимость создания дороги между каждой парой пунктов – вес соответствующего ребра, то, найдя МОД в полном графе, вершинам которого соответствуют населенные пункты, мы решим задачу. Известно несколько эффективных (полиномиальных) алгоритмов нахождения МОД. Приведем наиболее популярные.

Алгоритм Прима

Идея алгоритма принадлежит Приму. Эффективную технику реализации предложил Дейкстра [1]. Алгоритм состоит из $\boldsymbol{n-1}$ итераций. На каждой итерации к частично построенному дереву добавляется одна вершина и одно ребро. Сначала строящееся дерево $\boldsymbol{T} = (\boldsymbol{V_T}, \boldsymbol{E_T})$ содержит одну произвольную вершину и не содержит ребер. На каждой итерации ищется ребро минимального веса, связывающее вершину $\boldsymbol{i}$, принадлежащую $\boldsymbol{V_T}$, с вершиной $\boldsymbol{j}$, не принадлежащей $\boldsymbol{V_T}$

$$(\boldsymbol{i},\boldsymbol{j}) = \arg \min_{(\boldsymbol{p},\boldsymbol{q}):\boldsymbol{p}\in\boldsymbol{V_T},\boldsymbol{q}\notin\boldsymbol{V_T}} \boldsymbol{c_{pq}},$$

и добавляется в дерево: $\boldsymbol{V_T} = \boldsymbol{V_T} \cup \{\boldsymbol{j}\}$, $\boldsymbol{E_T} = \boldsymbol{E_T} \cup \{\boldsymbol{i},\boldsymbol{j}\}$,. Когда $\boldsymbol{V_T} = \boldsymbol{n}$, алгоритм останавливается, МОД построено.

Эффективная реализация алгоритма [1] заключается в том, что каждой не принадлежащей $\boldsymbol{V_T}$ вершине $\boldsymbol{j}$ приписывается метка (α_j, β_j), где α_j – номер ближайшей к $\boldsymbol{j}$ вершины из $\boldsymbol{V_T}$, а β_j – вес ребра $(\alpha_j, \boldsymbol{j})$. Тогда после присоединения очередного ребра, метка каждой вершины обновляется с трудоемкостью $\boldsymbol{O(1)}$. Число итераций равно $\boldsymbol{n-1}$, трудоемкость каждой итерации – $\boldsymbol{O(n)}$. Следовательно, общая трудоемкость эффективной реализации алгоритма Прима равна $\boldsymbol{O(n^2)}$.

Алгоритм Краскала [1]

Алгоритм начинает работу с тривиального графа $\boldsymbol{T} = (\boldsymbol{V}, \varnothing)$. Упорядочим ребра в порядке неубывания их весов и будем добавлять ребра в $\boldsymbol{T}$ по порядку. Очередное ребро добавляется в $\boldsymbol{T}$ и исключается из списка, если это не приводит к образованию цикла. В противном случае оно просто удаляется из списка и рассматривается следующее ребро списка. Это повторяется до тех пор, пока число ребер в $\boldsymbol{T}$ не станет $\boldsymbol{n-1}$. Построенное дерево – МОД.

Трудоемкость упорядочивания ребер – $\boldsymbol{O(m \log m)}$. Очевидно, что при построении дерева в худшем случае будут рассмотрены все $\boldsymbol{m}$ ребер графа. Пока не построено МОД, частично построенный граф является несвязным, и добавляемое ребро связывает вершины из разных компонент связности. В процессе рассмотрения ребер упорядоченного списка нужно следить, чтобы не возникало циклов,

т. е. чтобы не связывались вершины одной компоненты связности. Процедура, описанная в [1] (разделы 2.2.1 и 2.2.2), позволяет это сделать с константной трудоемкостью. Компоненты связности удобно хранить в виде системы непересекающихся множеств. Все операции в таком случае выполняются с трудоемкостью $\boldsymbol{O(m\,\alpha(m,\,n))}$, где $\boldsymbol{\alpha}$ – функция, обратная к функции Аккермана [2]. Поскольку для любых практических задач $\boldsymbol{\alpha(m,\,n)} < 5$, то можно принять ее за константу, таким образом, общая трудоемкость алгоритма Краскала равна $\boldsymbol{O(m\,\log m)}$. Следовательно, этот алгоритм более подходит для построения МОД в графе с небольшим количеством ребер.

Алгоритм Борувки

Алгоритм впервые был опубликован в 1926 г. О. Борувкой в качестве метода нахождения оптимальной электрической сети. Работа алгоритма состоит из итераций, каждая из которых заключается в последовательном добавлении ребер к остовному лесу графа, до тех пор, пока не будет построено одно дерево. В алгоритме предполагается, что веса ребер различны, или ребра как-то упорядочены, чтобы выбиралось единственное ребро с минимальным весом (в случае нескольких ребер, имеющих минимальный вес, выбирается, например, ребро с минимальным номером).

Пусть сначала $\boldsymbol{T = (V_T,\, E_T), V_T = V,\, E_T = \varnothing}$, – остовный лес, в котором каждая вершина является деревом. Пока $|\boldsymbol{E_T}| < n - 1$, выполнить:

- для каждой компоненты связности (дерева остовного леса) найти ребро минимального веса, связывающее эту компоненту с некоторой *другой* компонентой связности;
- добавить все найденные ребра в множество E_T.

Полученное дерево T является МОД.

На каждой итерации число деревьев в остовном лесу уменьшается не менее чем в два раза, поэтому алгоритм выполняет $O(\log n)$ итераций. Трудоемкость одной итерации равна $O(m)$, поэтому общая трудоемкость алгоритма – $O(m\,\log n)$.

1.2. Остовные деревья и их приложения

В практических задачах часто требуется построить остовное дерево, удовлетворяющее различным свойствам. В этом разделе приведены некоторые подобные задачи.

МОД ограниченной степени – это минимальное остовное дерево, в котором каждая вершина связана с не более чем d другими вершинами, где d – заданное целое число. Если $d = 2$, то это задача построения минимальной гамильтоновой цепи, откуда следует, что задача построения МОД ограниченной степени NP-трудна в общем случае. Если вершины графа – это точки на плоскости, а веса ребер равны евклидову расстоянию между ними, и требуется построить МОД степени k, то при $k = 5$ задача полиномиально разрешима [3].

В некоторых приложениях требуется построить такое остовное дерево, в котором максимальный вес ребра минимален. Очевидно, в этом случае МОД является одним из искомых деревьев.

При описании алгоритмов Прима и Краскала мы не налагали ограничений на знак весов ребер, поэтому задача построения остовного дерева максимального веса может быть решена алгоритмами Прима или Краскала после умножения весов ребер на –1 и построения МОД в графе с новыми весами.

Рассмотрим подробнее (в качестве примера) задачу построения сети связи с одним источником минимальной стоимости с ограничением на число узлов коммутации в цепях, связывающих пункты с источником. Эта задача в математической постановке является задачей построения МОД ограниченного радиуса на взвешенном графе с выделенной вершиной, которая может быть поставлена следующим образом.

Задан полный неориентированный взвешенный граф $G = (V, E)$, $V = \{0,1, \ldots, n\}$, с неотрицательными весами ребер $a_{ij} \geq 0$. Обозначим $F = F(G)$ – множество остовных деревьев графа G, а $C_k(T)$ – цепь, связывающая вершину k с корнем дерева 0 – источником сигнала в дереве $T \in F$. Требуется построить дерево $T^* \in F$, являющееся решением задачи:

$$\sum_{(i,j)} a_{ij} \to \min_{T \in F}; \qquad (1.1)$$

$$|C_k(T)| \le R, \qquad k = 1, \dots, n, \tag{1.2}$$

где $R \le n$ – заданное положительное целое число, а через $|C_k(T)|$ обозначено количество ребер в цепи $C_k(T)$. Число $\max_{k \in V} |C_k(T)|$ является радиусом дерева T.

Задача (1.1) – (1.2) при $R \ge 2$ является NP-трудной, что естественно вытекает из NP-трудности задачи построения МОД ограниченного диаметра [4]. В работе [5] показана NP-трудность рассматриваемой задачи на максимум, которая тесно связана с задачей (1.1)–(1.2). Действительно, если $a_{ij} \in [a, A]$, то задача

$$\sum_{(i,j) \in T} b_{ij} \to \max_{T \in F};$$

с ограничением (1.2), где $b_{ij} = A - a_{ij}$, эквивалентна задаче (1.1) – (1.2).

В работе [5] для задачи на максимум предложена серия полиномиальных алгоритмов построения решения с гарантированной оценкой относительной погрешности, в наихудшем случае равной 1/2. Если же для весов ребер выполняется неравенство треугольника, то оценка относительной погрешности улучшается до величины $\min\left\{\frac{2}{5}, \frac{4}{R+7}, \frac{2}{R+1}\right\}$.

Для задачи (1.1) – (1.2) полученные априорные оценки точности зависят от параметров задачи и не являются гарантированными [5].

Приведем еще один пример практической задачи выбора дальности передачи элементов радиосети, в которой строится остовное дерево. Эта задача известна в литературе как Min-Power Symmetric Connectivity Problem и заключается в следующем.

Задан простой неориентированный взвешенный граф $G = (V, E)$ с множеством вершин $V, |V| = n$, и множеством ребер E. Пусть $c_{ij} \ge 0$ – вес ребра $(i, j) \in E$. Требуется найти остовное дерево T^* графа G, являющееся решением задачи:

$$W(T) = \sum_{i \in V} \max_{j \in N_i(T)} c_{ij} \to \min_{T}, \tag{1.3}$$

где $N_i(T)$ – множество вершин, смежных с вершиной i в дереве T. Любое допустимое решение задачи (1.3) – остовное дерево – назовем также *коммуникационным деревом*.

Задача (1.3) NP-трудна в сильном смысле уже в случае, когда вершины – это точки в R^2, а вес ребра равен евклидову расстоянию между соответствующими точками. В общем случае NP-трудность естественно следует из сводимости задачи о минимальном покрытии (МП) к задаче (1.3).

Для любого остовного дерева T справедливы очевидные неравенства

$$\sum_{(i,j)\in T} c_{ij} \le W(T) \le 2 \sum_{(i,j)\in T} c_{ij},$$

откуда следует, что МОД является 2-приближенным решением задачи (1.3). Т.е.

$$\frac{W_{\text{МОД}}}{W^*} \le 2,$$

где $W_{\text{МОД}}$ – значение функционала на МОД, а W^* – оптимальное значение целевой функции. Справедлива

Теорема 1.1. Пусть веса ребер, вошедших в МОД, принадлежат отрезку $[a, b]$. Тогда

$$\frac{W_{\text{МОД}}}{W^*} \le 2 - \frac{2a}{a + b + 2b/(n-2)}$$

и при $n \to +\infty$

$$\frac{W_{\text{МОД}}}{W^*} \le \frac{2b}{a+b}.$$

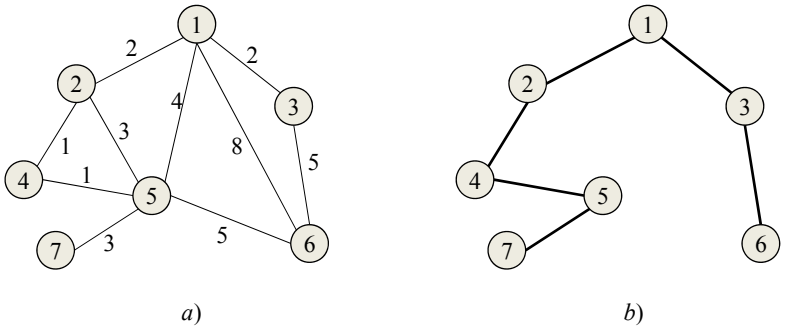
Теорема 1.2. Если задача построения k-приближенного решения задачи МП в графе, степени вершин которого не превосходят k, NP-трудна, то задача построения $\left(1 + \frac{R-1}{k+1}\right)$-приближенного решения задачи (1.3) также NP-трудна.

Следствие 1.1. Известно, что задача построения $\left(1 + \frac{1}{52}\right)$-приближенного решения для проблемы МП при $k = 4$ NP-трудна. Тогда из теоремы 1.2, в частности, следует NP-трудность построения $\left(1 + \frac{1}{260}\right)$-приближенного решения задачи (1.3).

1.3. Примеры и упражнения

Пример 1.1. Построить МОД в графе, изображенном на рис. 1.1*a* (рядом с ребрами указаны их веса), с помощью алгоритма Прима.

Решение. Положим $T = (\{1\}, \varnothing)$. Тогда метки вершин (α_j, β_j), $\alpha_j = 1,\ j = 2, \ldots, 7,\ \beta_2 = \beta_3 = 2,\ \beta_5 = 4$, остальные $\beta_j = +\infty$. Вершины 2 и 3, ближайшие к T, добавим, например, вершину 2, получив $T = (\{1, 2\}, \{(1, 2)\})$, и пересчитаем метки для вершин 3, …, 7. Получим $(\alpha_3, \beta_3) = (1, 2)$, $(\alpha_4, \beta_4) = (2, 1)$, $(\alpha_5, \beta_5) = (2, 3)$, $(\alpha_6, \beta_6) = (1, +\infty)$, $(\alpha_7, \beta_7) = (1, +\infty)$ Теперь ближайшая к T вершина 4, добавим ее: $T = (\{1, 2, 4\}, \{(1, 2), (2, 4)\})$. Продолжая процесс, добавим последовательно вершины 5, 3, 7 и 6 вместе с ребрами (4 ,5), (1, 3), (5, 7) и (3, 6). МОД изображено на рис. 1.1*b*, и его вес равен 14.

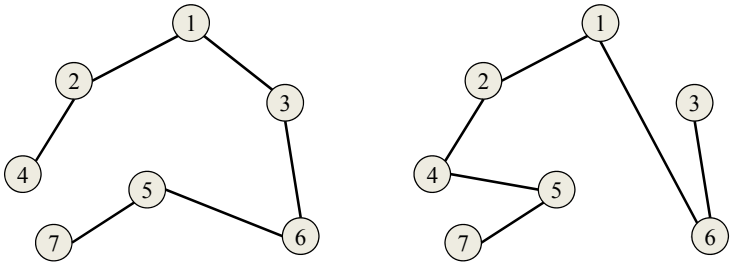


a) *b*)

Рис. 1.1. *a*) граф; *b*) МОД

Пример 1.2. Найти количество остовных деревьев графа, изображенного на рис. 1.1*a*, степени которых не превосходят 2.

Решение. Перенумеруем ребра. Очевидно, ребро 1 = (5, 7) войдет во все остовные деревья. Можно построить двоичное дерево решений, начиная с ребра 1, включая (если это не приводит к циклу или превышению допустимой степени вершин) или не включая очередное ребро. В результате получим три гамильтоновых пути. Один из них изображен на рис. 1.1*b*, два других – на рис. 1.2.



a) *b*)

Рис. 1.2. Два остовных дерева степени 2

Упражнение 1.1. Доказать, что алгоритмы Прима, Краскала и Борувки строят МОД.

Упражнение 1.2. Показать, что в дереве, имеющем две и более вершины, существует как минимум две вершины степени 1.

Упражнение 1.3. Найти такое остовное дерево графа, показанного на рис. 1.1*a*, в котором максимальный вес ребра минимален.

Упражнение 1.4. Найти МОД графа, показанного на рис. 1.1*a*, используя алгоритмы Краскала и Борувки.

Упражнение 1.5. Найти все МОД графа, показанного на рис. 1.3.

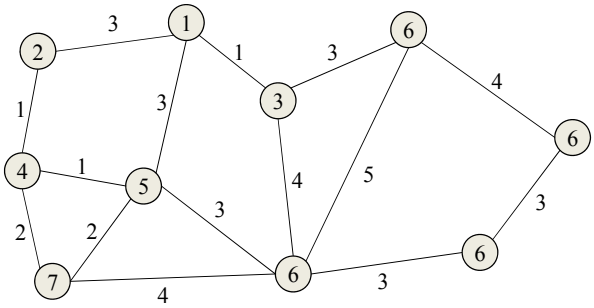


Рис. 1.3. Взвешенный граф
(рядом с каждым ребром указан его вес)

Глава 2. Задачи построения кратчайших путей

Пусть задан *ориентированный* (или неориентированный) граф $G = (V, A)$, в котором $V\{1, \dots, n\}$ – множество вершин, а A – множество дуг. Припишем каждой дуге $(i, j) \in A$ «длину» $d_{ij} \geq 0$.

Определение 2.1. *Путь* P_{ij} из вершины i в вершину j определим как последовательность дуг, которая начинается в i и заканчивается в j, в которой конец предшествующей дуги является началом следующей. Сумму длин дуг, входящих в путь, назовем *длиной* пути.

Задачи построения кратчайших путей естественно возникают во множестве приложений. Например, если в сети связи информация из источника должна достичь получателей за минимальное время, то, построив граф, в котором длина дуги совпадает с временем прохождения сигнала по этой дуге, мы сведем проблему к задаче построения дерева кратчайших путей (ДКП) из источника к приемникам сигнала.

Если требуется построить сеть дорог, связывающих заданное множество городов, таким образом, чтобы время поездки из одного города в другой было минимальным, то нужно найти кратчайшие пути между всеми парами городов.

2.1. Алгоритм Дейкстры

Алгоритм изобретен нидерландским ученым Э. Дейкстрой в 1959 г. [1] и строит ДКП из начальной вершины (корня) графа во все остальные вершины при неотрицательных длинах дуг (ребер).

Пусть s – начальная вершина (источник). Алгоритм на каждом шаге добавляет к частично построенному дереву T с корнем в s одну ближайшую к s вершину, не принадлежащую T. Если требуется найти кратчайший путь до некоторой вершины t, то алгоритм останавливается после добавления этой вершины. Иначе, строится остовное дерево, связывающее все вершины графа кратчайшими путями с s, т. е. ДКП.

Обозначим через $N_j = \{i \in V | (i, j) \in A\}$ – множество вершин, из которых есть дуги в вершину j в графе G. Положим начальное строящееся дерево $T = (s, \varnothing)$. Припишем каждой вершине $i \in V$ метку d_i, соответствующую (текущей) минимальной длине пути из s в i. Мет-

ка источника $d_s = 0$ не меняется в процессе работы алгоритма. Сначала $d_i = +\infty$ для всех $i \neq s$. На каждом шаге алгоритма ищется дуга

$$(i, j) = \arg \min_{p \in N_q;\ p \in T,\ q \notin T} \{d_p + d_{pq}\},$$

которая добавляется в T, и обновляется метка j-ой вершины $d_j = d_i + d_{ij}$. После этого (с константной трудоемкостью) обновляется метка каждой не включенной в T вершины $d_k = \min\{d_k,\ d_j + d_{jk}\}, k \notin T$.

Алгоритм останавливается, когда все вершины графа включены в дерево T (после выполнения n шагов). Трудоемкость одного шага (выбор вершины для включения в T) равна $O(n)$. Следовательно, общая трудоемкость алгоритма Дейкстры – $O(n^2)$.

2.2. Алгоритм Беллмана – Форда

Если длины некоторых ребер (дуг) принимают отрицательные значения, то в случае отсутствия циклов отрицательной длины кратчайшие пути из одной вершины во все остальные строит алгоритм Беллмана – Форда [1].

Введем $p(v)$ – номер вершины, непосредственно предшествующей вершине v в пути P_{sv}. Тогда алгоритм Беллмана – Форда можно записать по шагам следующим образом.

Шаг 1. Для каждой вершины $v \in V$: если $v = s$, то положить $d_v = 0$; иначе положить $d_v = +\infty$ и $p(v) = \text{null}$.

Шаг 2. Для каждой вершины $i = 1, \ldots, n-1$ и для каждого ребра $(u, v) \in A$: если $d_u + d_{uv} < d_v$, то положить $d_v = d_u + d_{uv}$ и $p(v) = u$.

Шаг 3. Для каждого ребра $(u, v) \in A$: если $d_u + d_{uv} < d_v$, то граф содержит отрицательный цикл.

Корректность приведенного алгоритма можно доказать по индукции.

Лемма 2.1. После выполнения i итераций:

1) если $d_u < +\infty$, то d_u – длина некоторого пути из s в u;
2) если существует путь из s в u, содержащий не более i ребер, то d_u не превосходит длины кратчайшего пути из s в u, содержащего не более i ребер.

Доказательство. На нулевой итерации ($i = 0$) длина пути, не содержащего ребер, $d_s = 0$. Для других вершин $d_u = +\infty$, т. к. не существует путей из источника в u с нулевым числом ребер. Докажем первое утверждение. Рассмотрим случай, когда длина пути в вершину v меняется, т. е. выполняется присваивание $d_v = d_u + d_{uv}$. По индуктивному предположению, d_u – это длина некоторого пути из источника в u. Значит, величина $d_u + d_{uv}$ равна длине пути $P_{su} \cup \{(u, v)\}$.

Для доказательства второго утверждения рассмотрим кратчайший путь из s в u, содержащий не более i ребер. Пусть v – предпоследняя вершина этого пути. Тогда часть пути из s в u является кратчайшим путем из источника в v, содержащего не более $i - 1$ ребер. По индуктивному предположению, d_v после $i - 1$ итерации не превосходит длины этого пути. Следовательно, $d_{uv} + d_v$ не превосходит длины пути из s в u. На i-ой итерации величина d_u сравнивается с $d_{uv} + d_v$, и ей присваивается новое значение, если $d_{uv} + d_v$ меньше. Следовательно, после i-ой итерации d_u не превосходит длины кратчайшего пути из источника в u, содержащего не более i ребер. ■

Если в графе нет отрицательных циклов, то в каждую вершину строится единственный путь минимальной длины, и на шаге 3 не может произойти уменьшение длины ни одного пути. В противном случае отрицательный цикл существует.

2.3. Алгоритм Флойда – Уоршелла

Алгоритм Флойда – Уоршелла (Floyd – Warshall) разработан в 1962 г. [1] и используется для нахождения кратчайших расстояний между всеми парами вершин взвешенного ориентированного графа.

Введем обозначение d_{ij}^k для длины кратчайшего пути из i в j, который, кроме самих вершин i, j, проходит только через вершины множества $\{1,2, \dots, k\}$. Тогда $d_{ij}^0 = d_{ij}$ – длина дуги (i, j), если она существует; иначе $d_{ij}^0 = +\infty$.

Если кратчайший путь из i в j не проходит через вершину k, тогда $d_{ij}^k = d_{ij}^{k-1}$. Если существует более короткий путь из i в j, прохо-

дящий через k, тогда $d_{ij}^{k} = d_{ik}^{k-1} + d_{kj}^{k-1}$. Рекуррентная формула для вычисления d_{ij}^{k} имеет вид:

$d_{ij}^{0} = d_{ij}$;

$d_{ij}^{k} = \min\{d_{ij}^{k-1},\ d_{ik}^{k-1} + d_{kj}^{k-1}\}$.

Алгоритм Флойда – Уоршелла последовательно вычисляет все значения d_{ij}^{k} для всех $i,\ j,\ k = 1, \dots, n$. Значения d_{ij}^{n} являются длинами кратчайших путей между вершинами i и j. Несложно понять, что трудоемкость алгоритма – $O(n^3)$. Если дополнительно для каждой пары вершин хранить информацию о первой вершине пути, то, помимо расстояния между двумя узлами, получим возможность восстановить сами кратчайшие пути.

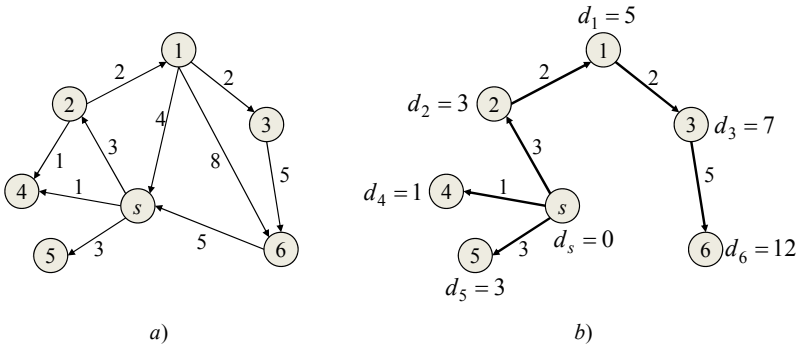
Не существует кратчайшего пути между парой вершин i, j, который является частью отрицательного цикла, т. к. длина пути из i в j может быть сколь угодно мала (отрицательна). По умолчанию, применение алгоритма подразумевает отсутствие отрицательных циклов в графе. Однако если отрицательный цикл есть, алгоритм Флойда – Уоршелла можно использовать для его поиска:

- алгоритм итеративно пересчитывает длины кратчайших путей между каждой парой вершин i, j, в том числе и $i = j$;
- сначала все длины пути из i в i равны нулю;
- длина пути $P_{ik} \cup P_{ki}$ может быть меньше в случае, если она отрицательная, т. е. это отрицательный цикл;
- значит, после остановки алгоритма длина пути из i в i будет отрицательной, если существует цикл отрицательной длины из i в i.

Следовательно, для определения отрицательного цикла с помощью алгоритма Флойда – Уоршелла достаточно просмотреть диагональ матрицы d. Присутствие на диагонали отрицательных чисел говорит о наличии в графе как минимум одного отрицательного цикла. Очевидно, в неориентированном графе ребро отрицательной длины вместе с инцидентными ему вершинами образует отрицательный цикл.

2.4. Примеры и упражнения

Пример 2.1. Построить дерево кратчайших расстояний из вершины s во все вершины ориентированного графа, изображенного на рис. 2.1a (рядом с дугами указаны их длины), с помощью алгоритма Дейкстры.



a) *b*)

Рис. 2.1. Иллюстрация работы алгоритма Дейкстры

Решение. Положим длины кратчайших путей в вершины (метки вершин) равными $d_s = 0,\ d_i = +\infty,\ i \neq s$, и частично построенное дерево кратчайших путей $T = (s, \varnothing)$. Найдем

$$\arg \min_{p \in N_q;\, p \in T;\, q \notin T} \{d_p + d_{pq}\} = \arg \min_{s \in N_q;\, q \notin T} \{d_s + d_{sq}\} = (s, 4)$$

и положим $T = \big(\{s, 4\}, \{(s, 4)\}\big),\ d_4 = 1$. Метки других вершин не изменятся, т. к. из вершины 4 не исходит ни одна дуга.

Найдем

$$\arg \min_{p \in N_q;\, p \in T;\, q \notin T} \{d_p + d_{pq}\} = (s, 2)$$

и положим $T = (\{s, 2, 4\}, \{(s, 4), (s, 2)\}),\ d_2 = 3$. Обновим метку вершины 1: $d_1 = \min\{d_1, d_2 + d_{21}\} = \min\{+\infty, 3 + 2\} = 5$. Метки других вершин не изменятся. Продолжая процесс, добавим последовательно в строящееся дерево ребра $(s,\ 5), (2, 1), (1,\ 3), (3,\ 6)$ и вершины 5, 1, 3, 6. В результате будет построено дерево, показанное на рис. 2.1b, в котором метки рядом с вершинами – длины кратчайших путей из источника s.

Пример 2.2. Найти длины кратчайших путей между всеми парами вершин графа, изображенного на рис. 2.1*a*, с помощью алгоритма Флойда – Уоршелла.

Решение. Зададим начальные длины путей между вершинами матрицей с элементами $d_{ij}^0 = d_{ij}$, полагая $s = 7$ (рис. 2.2*a*, на котором символ «-» соответствует бесконечности). Воспользуемся рекуррентными соотношениями $d_{ij}^1 = \min\{d_{ij}^0, d_{ik}^0 + d_{kj}^0\}$ для вычисления длин путей после первой итерации (рис. 2.2*b*).

0	-	2	-	-	8	4
2	0	-	1	-	-	-
-	-	0	-	-	5	-
-	-	-	0	-	-	-
-	-	-	-	0	-	-
-	-	-	-	-	0	5
-	3	-	1	3	-	0

a) d_{ij}^0

0	7	2	5	8	8	4
2	0	4	1	-	10	6
-	-	0	-	-	5	10
-	-	-	0	-	-	-
-	-	-	-	0	-	-
-	8	-	6	8	0	5
5	3	-	1	3	-	0

b) d_{ij}^1

Рис. 2.2. Матрица начальных расстояний (*a*) и после первой итерации (*b*)

0	7	2	5	8	8	4
2	0	4	1	9	10	6
17	13	0	11	13	5	10
-	-	-	0	-	-	-
-	-	-	-	0	-	-
10	8	12	6	8	0	5
5	3	7	1	3	13	0

*Рис. 2.*3. Матрица кратчайших расстояний между всеми парами вершин

Очевидно, что из вершин 4 и 5 нет путей в другие вершины, поэтому бесконечности в этих строках сохранятся как после первой итерации, так и в дальнейшем.

После 4 итерации получим окончательные значения для минимальных длин путей, связывающих все пары вершин (рис. 2.3).

Упражнение 2.1. Доказать, что алгоритм Дейкстры строит дерево кратчайших расстояний.

Упражнение 2.2. Найти кратчайшие пути между всеми парами вершин графа, изображенного на рис. 2.1*а*.

Упражнение 2.3. Существует ли отрицательный цикл в графе, изображенном на рис. 2.4?

Упражнение 2.4. Воспользуйтесь алгоритмом Беллмана – Форда для построения кратчайших путей из вершины ***s*** в графе, изображенном на рис. 2.4, удалив входящие в ***s*** дуги.

Упражнение 2.5. Доказать корректность алгоритма Флойда – Уоршелла.

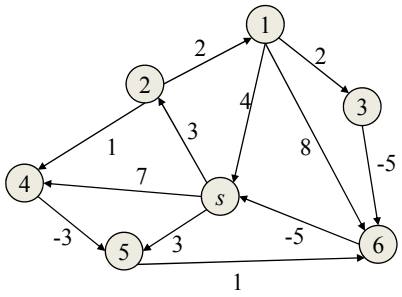


Рис. 2.4. Ориентированный взвешенный граф
(рядом с дугами указаны их длины)

Глава 3. Задача Штейнера

Пусть на плоскости расположены три точки, расстояния между которыми определяются евклидовой метрикой, и требуется связать эти точки с помощью МОД. Построенное дерево показано на рис. 3.1*a*, и его длина (сумма длин ребер) равна $\sqrt{5} + \sqrt{10}$.

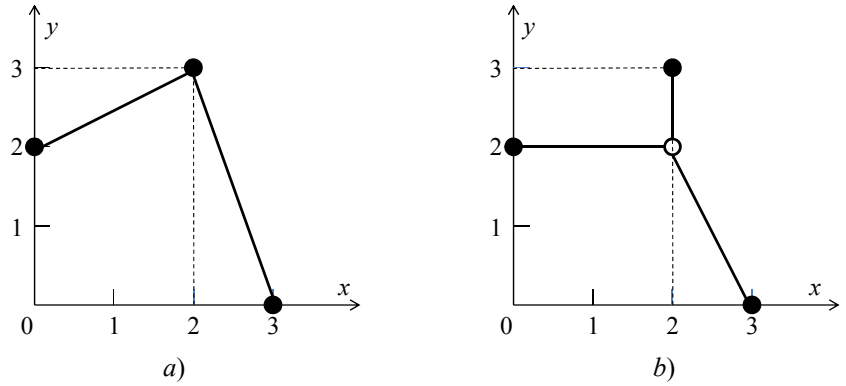


Рис. 3.1. Минимальное остовное дерево (*a*) и дерево Штейнера (*b*)

Введем дополнительную (промежуточную) точку и соединим исходные точки с промежуточной точкой. В результате получим связывающее дерево меньшей длины $\sqrt{5} + 3$ (по сравнению с МОД) (рис. 3.1*b*). Это дерево является *геометрическим* (или *метрическим*) *деревом Штейнера*, и в него вошла одна дополнительная точка (2, 2) – *вершина Штейнера*. При построении метрического дерева Штейнера для уменьшения длины дерева можно использовать любые дополнительные точки на плоскости.

Далее будем рассматривать задачу Штейнера *на графах*, формулировка которой приводится в следующем разделе. В последней задаче точки Штейнера выбираются из заданного множества вершин графа.

3.1. Постановки задачи Штейнера и ее сложность

Пусть задан неориентированный простой взвешенный граф $G = (V, E)$, в котором множество вершин является объединением двух множеств $V = S \cup I$. Вершины множества S будем называть *терминалами*, а вершины из множества I – *промежуточными* вершинами. Каждому ребру $(i, j) \in E,\ i, j \in V$, приписана «длина» $c_{ij} \geq 0$. Требуется связать все вершины множества S деревом минимальной длины, которое называется *минимальным деревом Штейнера* (МДШ). При этом в искомое дерево могут войти промежуточные вершины из множества I.

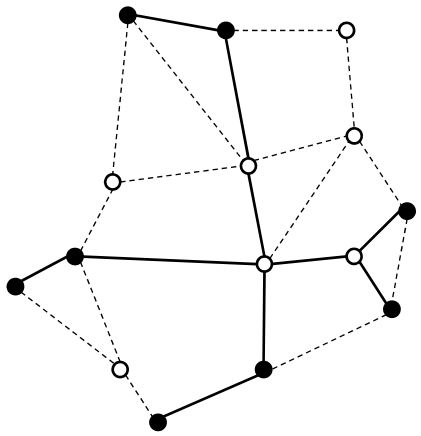


Рис. 3.2. Пример дерева Штейнера на графе

На рис. 3.2 пунктирными линиями изображены ребра графа G, которые соединяют терминалы – закрашенные вершины и промежуточные (не закрашенные) вершины. Сплошными линиями показано дерево Штейнера, в которое вошли три промежуточные вершины.

Для МДШ на плоскости с евклидовой метрикой справедливы следующие свойства.

Свойство 3.1. Точка Штейнера имеет степень 3.

Свойство 3.2. Если вершина i имеет степень 3 в МДШ, то угол между любыми двумя ребрами, инцидентными i, равен 120°.

Свойство 3.3. Число точек Штейнера в МДШ равно $k \in [0, |S| - 2]$.

Известно, что задача Штейнера на графах (а также геометрическая задача Штейнера) NP-трудна в сильном смысле [4], поэтому на практике используются различные приближенные алгоритмы для построения МДШ.

3.2. Приближенные алгоритмы

Очевидно, МОД является приближенным решением задачи Штейнера. При этом для метрического случая, когда точки (вершины) расположены на плоскости, отношение $W_{\text{МОД}}/W^* \leq 2$, где $W_{\text{МОД}}$ – вес минимального остовного дерева, а W^* – минимальная длина дерева Штейнера. Алгоритм Прима (Краскала) строит 2-приближенное решение задачи Штейнера с полиномиальной трудоемкостью.

Известен более сильный результат: $W_{\text{МОД}}/W^* \leq 2/\sqrt{3}$, для метрической задачи с евклидовой метрикой. Предложено также несколько алгоритмов, которые строят 2-приближенное решение задачи Штейнера на произвольных графах. Например, достаточно найти кратчайшие пути между каждой парой вершин графа, перейти к вспомогательному графу без промежуточных вершин с длинами ребер, равными длинам кратчайших путей, и построить МОД. Это и будет 2-приближенное решение для исходной задачи Штейнера.

Опубликован ряд работ, в которых предложены алгоритмы с меньшей оценкой точности, например, алгоритм строящий 1.55-приближенное решение для точек, расположенных на плоскости, расстояния между которыми задаются прямоугольной (манхэттеновской) метрикой L_1 [6]. В этом случае дерево Штейнера состоит из множества вертикальных и горизонтальных отрезков, соединяющих терминалы и промежуточные точки. Однозначно определяется минимальный прямоугольник, в котором находятся все терминалы, и его стороны параллельны осям (прямоугольник *ABCD* на рис. 3.3). Очевидно, все ребра МДШ не выходят за пределы этого прямоугольника. В 1966 г. Ханан показал [7], что существует МДШ, в ко-

торое входят только узлы решетки, полученной от пересечения горизонтальных и вертикальных прямых, проходящих через терминалы, – решетка Ханана (рис. 3.3).

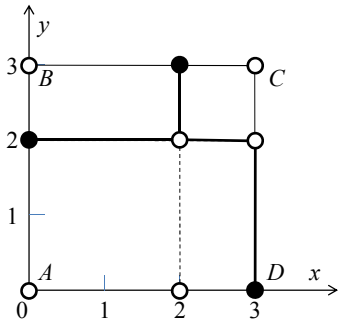


Рис. 3.3. Минимальное дерево Штейнера (жирные линии) на решетке Ханана

В 1976 г. Хванг [8] показал, что МОД является 3/2-приближенным решением. В 1992 г. Зеликовский [9] разработал алгоритм построения прямоугольного дерева Штейнера с оценкой точности 11/8, первый эвристический алгоритм, который строит решение лучше МОД.

3.3. Некоторые задачи синтеза сетей, использующие деревья Штейнера

Задачи построения деревьев Штейнера, обладающих теми или иными свойствами, возникают при синтезе сетей передачи данных, при проектировании коммуникаций, дорог, трубопроводов, при трассировке сверхбольших интегральных схем (СБИС) и т. п.

Рассмотрим для примера следующую задачу построения МДШ с ограничениями на длины путей из заданной вершины (корень или источник сигнала), которая возникает в связи с трассировкой СБИС. В [10] задача поставлена следующим образом.

Задан взвешенный граф $G = (V, E)$, $V = \{0, 1, \ldots, n\}$, где выделенную вершину 0 назовем корнем. Для каждого терминала $k \in S \subseteq V$ задана максимально допустимая длина пути из корня $d_k \geq 0$. Каж-

дому ребру графа $(i,j) \in E$ приписаны целочисленные «вес» $c_{ij} \in [c,\ C]$ и «длина» $d_{ij} \in [d,\ D]$, $c \geq 0, d \geq 0$. Требуется решить задачу

$$W(T) = \sum_{(i,j)\in T} c_{ij} \to \min_{T\in F}; \tag{3.1}$$

$$\sum_{(i,j)\in P_s(T)} d_{ij} \leq d_s, \qquad s \in S, \tag{3.2}$$

где F – множество деревьев Штейнера, связывающих вершины множества S, а $P_s(T)$ – путь из корня в вершину s в дереве T.

Задача (3.1) – (3.2) остается NP-трудной даже в случае $S = V$. В [10] предложен следующий эвристический алгоритм для решения задачи (3.1) – (3.2).

Сначала строящееся дерево состоит из одного корня $T_0 = (\{0\}, \varnothing)$. На каждой итерации алгоритма k к дереву, построенному на предыдущих шагах T_{k-1}, добавляются одна вершина $j \notin T_{k-1}$ и одно ребро $(i,\ j) \in E$:

$$(i,j) = \arg \min_{i\in T_{k-1}, j\notin T_{k-1}} \{c_{ij} + q(d_{ij} + R_i)\},$$

где R_i – длина пути $P_i(T_{k-1})$, $R_j = d_{ij} + R_i$, параметр $q \geq 0$, в результате чего получаем дерево T_k.

Варьирование значений параметра q позволяет контролировать «качество» решения. В частности, доказано, что в случае целочисленных весов и длин ребер приведенный алгоритм строит дерево кратчайших путей при $q \geq Cn$. При $q = 0$ алгоритм, очевидно, строит некое дерево Штейнера без учета ограничений на длины путей. Меняя значения параметра, можно получать разные деревья и запомнить в качестве приближенного решения лучшее допустимое дерево. Показано, что для построения разных деревьев достаточно запустить алгоритм с конечным числом значений q, что приводит к полиномиальной трудоемкости всего алгоритма $O(n^2 \log(CDn))$.

Приведенный алгоритм не имеет гарантированной оценки точности, и для его анализа проведен численный эксперимент, показавший высокую эффективность.

Обратимся к следующему примеру. Одним из этапов проектирования сверхбольших интегральных схем (после размещения элемен-

тов на СБИС) является трассировка соединений или маршрутизация. Этот этап разбивается на глобальную и детальную маршрутизацию.

Глобальная трассировка является одним из важнейших этапов проектирования СБИС, на котором для каждой цепи определяется множество используемых областей маршрутизации в условиях ограничений на трассировочные ресурсы и время прохождения сигнала. В литературе встречается несколько формулировок задачи глобальной трассировки (ЗГТ) с различными критериями и ограничениями. Основной целью глобальной трассировки является маршрутизация всех цепей СБИС без нарушения ограничений. При этом даже простейшая постановка, в которой требуется осуществить маршрутизацию двухтерминальных цепей в условиях ограниченности трассировочных ресурсов (без учета временных задержек), является NP-трудной задачей.

Для решения ЗГТ исследователями предложены различные подходы, в которых трассировка, как правило, осуществляется лишь на двух слоях. В основе этих подходов лежат алгоритмы последовательной маршрутизации, алгоритмы трассировки с разрывом связей и поиском новых соединений, алгоритмы, основанные на решении задач о многопродуктовом потоке, иерархические методы, а также различные метаэвристики.

При проектировании современных СБИС на этапе глобальной маршрутизации, наряду с учетом трассировочных ресурсов, все большее внимание уделяется времени распространения сигнала. При этом плотность соединений и временна́я задержка являются, как правило, конкурирующими критериями, и в литературе практически отсутствуют публикации, в которых эти критерии рассматриваются совместно.

Логическая схема СБИС может быть представлена ациклическим графом с несколькими *основными входами* и *основными выходами*. Она включает в себя разнотипные *элементы*, связанные частично упорядоченными *цепями*. Каждый элемент схемы реализует определенную булеву функцию и имеет несколько входов и один выход, которые наряду с основными входами и выходами называются *терминалами*. Цепь задается одним терминалом-источником и несколькими терминалами – получателями сигнала. Например,

на рис. 3.4 цепь, выделенная жирными линиями, имеет один источник – выход элемента 5 – и четыре терминала, которые являются входами элементов 7, 8, 9 и 11.

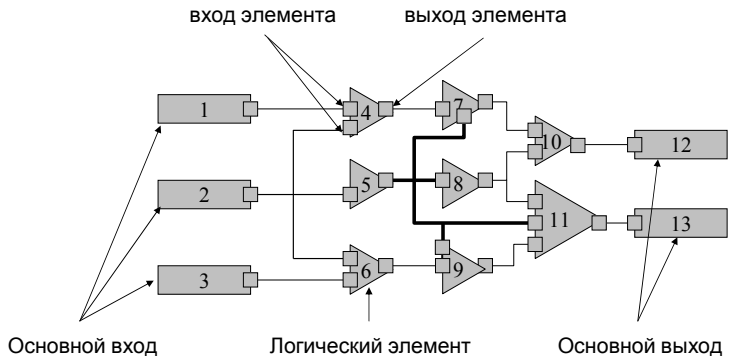


Рис. 3.4. Пример логической сети

Для каждого основного входа схемы задано время поступления сигнала извне (AT – arrival time), а для каждого основного выхода – наиболее позднее допустимое время получения сигнала, которое назовем *директивным* (RT – required time).

Область размещения элементов и трассировки состоит из одинаковых прямоугольников, расположенных друг над другом, которые называются *слоями*. Размещение элементов интегральной схемы осуществляется до трассировки, поэтому координаты всех терминалов предполагаются известными.

Терминалы соседних слоев связываются *межслойными соединениями* (за межслойным соединением в англоязычной литературе закрепился термин «via»), которые будем считать параллельными оси 0*z*. Трассировка на каждом слое осуществляется либо параллельно оси 0*x*, либо параллельно оси 0*y*. При этом некоторые слои могут использоваться только для размещения элементов схемы.

На этапе глобальной маршрутизации все слои СБИС разбиваются на одинаковые *глобальные ячейки*, имеющие форму прямоугольников.

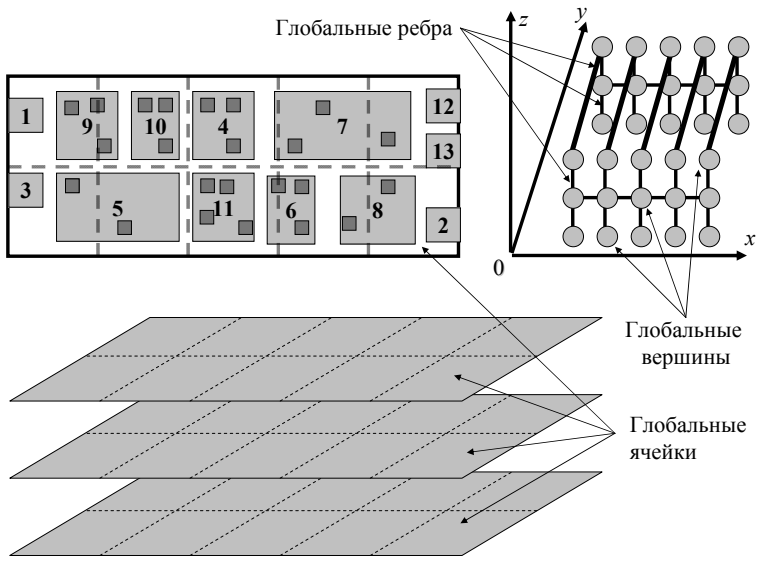


Рис. 3.5. Пример построения глобального графа

В результате каждый терминал попадает в одну из таких ячеек. Затем строится следующий *глобальный граф*. Каждой глобальной ячейке ставится в соответствие *глобальная вершина*. Пара глобальных вершин соединяется *глобальным ребром* в следующих случаях (рис. 3.5):

- если они расположены на одном слое, и соответствующие им глобальные ячейки имеют общую сторону, которая перпендикулярна направлению маршрутизации на данном слое;
- если эти вершины расположены на соседних слоях, и их (x, y)-координаты совпадают.

Для соединения терминалов произвольной цепи используется дерево Штейнера в глобальном графе. Каждому глобальному ребру приписан трассировочный ресурс или *пропускная способность* – максимальное число вхождений этого ребра в совокупность деревьев Штейнера.

Итак, пусть задан глобальный граф $G = (V, E)$ с множеством вершин V и множеством ребер E. Каждому ребру $(i, j) \in E$ приписана

длина $l_{ij} > 0$, емкостное сопротивление (*емкость*) $c_{ij} \geq 0$, электрическое сопротивление (*сопротивление*) $r_{ij} \geq 0$ и пропускная способность $q_{ij} \geq 0$.

Цепь s (будем обозначать ее просто номером $s = 1, \dots, S$) задана множеством терминалов $V^s \subseteq V$. Для терминалов, являющихся основными входами СБИС, заданы времена поступления сигналов извне (ATs). Для каждого из основных выходов задано наиболее позднее допустимое время получения сигнала (RT). Терминал $i \in V^s$ обладает емкостью $c_i^s \geq 0$, которая определяется типом элемента. У каждой цепи s имеется один источник $0^s \in V^s$, имеющий сопротивление $r_0^s \geq 0$.

Для аналитического вычисления времен прохождения сигнала в современных СБИС обычно используется модель Эльмора. Пусть задано дерево T с корнем в вершине 0. Обозначим через:

- $P_k(T)$ – путь из корня в вершину k в этом дереве T;
- C_j – емкость поддерева T_j с корнем в вершине j, то есть $C_j = \sum_{(i,j) \in T_j} c_{ij} + \sum_{i \in T_j} c_i$.

Тогда время прохождения сигнала по дуге $(i, j) \in T$ вычисляется по формуле:

$$d_{ij} = d_{ij}(T) = r_{ij}\left(\frac{c_{ij}}{2} + C_j\right), \tag{3.3}$$

а время прохождения сигнала из корня в произвольную вершину k дерева T вычисляется по формуле:

$$t_k = t_k(T) = r_0 C_0 + \sum_{(i,j) \in P_k(T)} d_{ij}. \tag{3.4}$$

Пусть временные задержки вычисляются по формулам Эльмора (3.3)–(3.4). В заданном n-вершинном дереве трудоемкость этих вычислений ограничена величиной $O(n)$.

В ЗГТ требуется связать вершины каждого множества V^s, $s = 1, \dots, S$, деревом Штейнера таким образом, чтобы каждое ребро глобального графа $(i, j) \in E$ вошло не более чем в q_{ij} различных деревьев, и время прихода сигнала в каждый основной выход СБИС не превышало директивное (RT).

На практике допустимая трассировка может не существовать, либо ее сложно найти из-за большой размерности задачи и ее NP-трудности. Поэтому рассматривается проблема построения трассировки, *близкой к допустимой*, и в дальнейшем именно такую задачу мы будем называть задачей глобальной трассировки. Другими словами, ЗГТ состоит в отыскании трассировки, являющейся компромиссом между превышением трассировочных ресурсов и временем прохождения сигнала.

Для решения ЗГТ предлагается следующая итеративная процедура. Сначала осуществляется сортировка цепей, основанная на одном из эмпирических критериев, в качестве которых могут использоваться количество терминалов цепи, периметр минимального прямоугольника, содержащего все терминалы цепи, и другие. Затем для каждой цепи s из упорядоченного списка, с учетом временных задержек и остаточных пропускных способностей ребер глобального графа, строится множество Q^s *деревьев-кандидатов*. Далее выбирается по одному дереву из каждого множества $Q^s, s = 1, \dots, S$. Для этого рассматривается проблема минимизации суммы штрафов за превышение трассировочных ресурсов, которая сформулирована в форме задачи квадратичного целочисленного программирования. Адаптированный градиентный алгоритм находит решение непрерывной релаксации последней задачи, и затем с помощью различных эвристик строятся целочисленные решения, лучшее из которых выбирается в качестве приближенного решения ЗГТ.

Для построения деревьев с учетом времени прохождения сигнала предложен алгоритм МАД. Рассмотрим произвольную цепь s с источником сигнала в вершине $0 \in V^s$. Алгоритм МАД строит дерево Штейнера для цепи s в некотором связном подграфе $G' = (V', E')$ графа G, где $V' \supseteq V^s$. Из G' удаляются ребра, пропускная способность которых меньше некоторого целого q, являющегося параметром алгоритма. Для каждого оставшегося ребра $(i, j) \in E'$ по формуле (3.3) вычисляется задержка d_{ij}, для чего может использоваться дерево, построенное на предыдущей итерации (вначале это дерево не содержит ребер). Различные способы подсчета d_{ij} обсуждаются ниже.

Описанный алгоритм является модификацией алгоритма Дейкстры, но не гарантирует построение дерева с минимальными задержками в силу специфики модели Эльмора. Заметим, что по-

строение дерева с минимальными временами прохождения сигнала из корня в терминалы является NP-трудной проблемой, т. к. частный случай, когда сопротивления всех ребер нулевые, есть задача Штейнера на графах.

Топология построенного дерева зависит от подграфа G', параметра q и величин d_{ij}. В качестве G' можно использовать весь граф G, граф Ханана, минимальную решетку, содержащую множество терминалов цепи, либо другие подграфы глобального графа.

Для построения различных деревьев с учетом задержек Эльмора предлагается применять алгоритм МАД итеративно, используя ранее построенные деревья для вычисления d_{ij}. Как видно из формулы (3.3), величина d_{ij} зависит от емкости C_j поддерева T_j. Поскольку емкость поддерева C_j в процессе построения дерева неизвестна, то для вычисления d_{ij} можно использовать значения этой величины в построенных *ранее* деревьях. Например, C_j может быть емкостью поддерева T_j, построенного на предыдущей итерации, или средним арифметическим емкостей поддеревьев T_j во всех ранее построенных деревьях.

Разнообразия деревьев, строящихся алгоритмом МАД, можно добиться изменением значений параметра q. Кроме этого, можно использовать деревья, построенные другими алгоритмами. Например, если сопротивление и емкость соединения пропорциональны длине соединения, можно строить в качестве деревьев-кандидатов минимальные деревья Штейнера, а также деревья, содержащие кратчайшие пути из корня в терминалы. Так как задача построения минимального дерева Штейнера принадлежит классу NP-трудных проблем, то следует применять различные приближенные алгоритмы, которые будут строить *разные* деревья. Это расширит множества деревьев-кандидатов, что может позволить найти лучшее решение с учетом трассировочных ресурсов глобального графа.

Пусть для каждой цепи s найдено множество деревьев-кандидатов Q^s. Если для некоторого s множество Q^s состоит из единственного дерева, то оно включается в решение, после чего пересчитываются пропускные способности глобальных ребер, а цепь s исключается из списка.

Рассмотрим задачу оптимального распределения трассировочных ресурсов, то есть задачу выбора деревьев из множеств Q^s, таких, что $|Q^s| > 1$. Для краткости записи в дальнейшем индекс $e \in E$ используется для ребер графа G, а индекс $t \in J = \bigcup_{s=1}^{S} Q^s$ – для деревьев. Положим $a_{et} = 1$, если ребро e принадлежит дереву t, и $a_{et} = 0$ в противном случае. Переменная x_t принимает значение $x_t = 1$, если дерево t выбрано, и $x_t = 0$ в противном случае.

В принятых обозначениях рассмотрим задачу:

$$f(x) = \sum_{e \in E} \left(\max \left\{ 0, \ \sum_{t \in J} a_{et} x_t - q_e \right\} \right)^2 \to \min_{x_t \in \{0,1\}}; \tag{3.5}$$

$$\sum_{t \in Q^s} x_t = 1, \ \ s = 1, \dots, S. \tag{3.6}$$

Целевая функция (3.5) равна сумме штрафов за превышение трассировочных ресурсов $q_e, e \in E$. Трассировке без превышения пропускных способностей глобальных ребер соответствует нулевое значение целевой функции.

Заменяя условие целочисленности переменных условием их неотрицательности, получаем непрерывную релаксацию задачи (3.5)–(3.6). Гладкость и выпуклость целевой функции позволяет применить для решения последней задачи градиентный алгоритм. Предложен градиентный алгоритм, в котором трудоемкость каждой итерации равна $O(|E| \cdot |J|)$, а общее число итераций не превосходит $O(\varepsilon^{-1} \ln \varepsilon^{-1})$. В описанном методе на каждой итерации мы не стремимся найти направление *наискорейшего* спуска, что увеличило бы трудоемкость алгоритма, а выбираем направление спуска из текущей точки в *целую* точку.

Выбор одного дерева t_s из каждого множества Q^s дает множество деревьев Штейнера, которое является приближенным решением задачи (3.5)–(3.6). Это множество деревьев зависит от текущей точки и поэтому может меняться от итерации к итерации. В процессе спуска будем хранить лучшее (например, в смысле критерия (3.5), или плотности соединений) из найденных решений.

Приведем еще один пример построения сигнального дерева в вычислительной системе. Сигнальная сеть отвечает за синхрониза-

цию работы вычислительной системы. Сигналы-команды генерируются вне системы и подаются в нее через вход (корень). Каждый функциональный элемент связан с корнем посредством сигнальной сети. Элемент выполняет серии логических операций (функций) и ждет сигнала для передачи результатов другим элементам, пока не начнется следующий цикл вычислений. Таким образом, происходит контроль информационных потоков внутри вычислительной системы. *Временно́й перекос* (или *clock skew* в англоязычной литературе) – это максимальная разница между моментами получения сигналов различными компонентами системы. Увеличение временного перекоса в вычислительных системах ведет к уменьшению скорости вычислений. В современных системах, когда размеры элементов существенно меньше микрона, временной перекос является одним из основных факторов, определяющих функционирование системы. Временной перекос уменьшает тактовую частоту, так как период между двумя последовательными сигналами должен быть увеличен, чтобы все компоненты схемы успели получить сигнал. Считается, что для высокоскоростных схем временной перекос не должен превышать 5 % от максимального времени передачи сигнала.

Рассматриваемую проблему синхронизации получения сигналов-команд элементами вычислительной системы можно сформулировать следующим образом. Каждый терминал (элемент схемы) выполняет определенные операции. Все терминалы выполняют часть общей программы и должны работать согласованно. Требование получения сигналов всеми терминалами одновременно можно удовлетворить, если построить дерево, в котором все пути из источника в терминалы имеют одинаковые задержки. Такое дерево назовем допустимым. Кроме того, среди допустимых деревьев следует выбрать дерево минимального веса, что позволит минимизировать занятое деревом пространство.

При этом маршрутизация ребер дерева производится на равномерной прямоугольной плоской решетке (в частности, имеем, что степень каждой вершины в таком графе не превосходит четырех). Следовательно, не всегда возможно построение допустимого дерева.

С математической точки зрения проблема заключается в построении минимального прямоугольного дерева Штейнера, в кото-

ром расстояния (по дереву) из корня до каждого терминала равны радиусу графа-решетки. Известно, что такая задача является NP-трудной. Приближенный алгоритм решения задачи предложен, например, в [11].

3.4. Примеры и упражнения

Пример 3.1. Пусть требуется связать МДШ четыре точки расположенные на плоскости попарно симметрично относительно горизонтальной прямой (рис. 3.6).

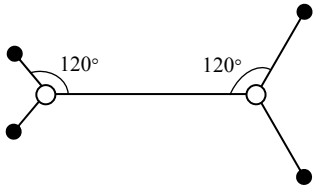


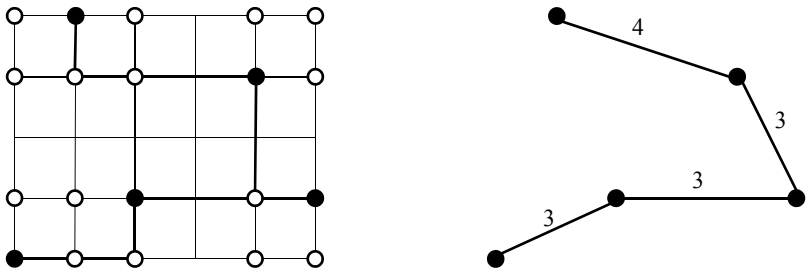
Рис. 3.6. Вершины степени 3 – точки Штейнера

Решение. Из свойств геометрического МДШ следует, что число точек Штейнера не превосходит 2, и инцидентные им ребра образуют угол в 120°. Проведем горизонтальную прямую на одинаковом расстоянии между левыми и правыми точками. Из каждой точки проведем прямую, пересекающуюся с горизонтальной прямой под углом 120°. Точки пересечения – точки Штейнера. МДШ построено, в него вошли 2 точки Штейнера (см. рис. 3.6).

Пример 3.2. Пусть вершины графа расположены в узлах единичной решетки на рис. 3.7. Требуется связать терминалы 2-приближенным деревом Штейнера с использованием прямоугольной метрики L_1.

Решение. Построим сначала решетку Ханана, проведя горизонтальные и вертикальные линии, проходящие через терминалы (рис. 3.7*a*). Найдем длины кратчайших путей между каждой парой терминалов и построим МОД (рис. 3.7*b*). На решетке представление построенного дерева не однозначно, но любое представление будет

2-приближенным решением задачи Штейнера. На рис. 3.7*a* представлено дерево, которое является МДШ.



a) *b*)

Рис. 3.7. Решетка Ханана (*a*) и минимальное остовное дерево (*b*)

Пример 3.3. Для иллюстрации работы метода решения ЗГТ приведем пример. Пусть имеется 11 сетей, 3 слоя и решетка 5×2. Все элементы СБИС расположены на первом (нижнем) слое, межслойные соединения имеют высокую пропускную способность, нулевую емкость и сопротивление 0.00937. Второй слой используется для осуществления «вертикальных» соединений (то есть параллельных оси 0*y*), линия связи (соответствующее ребро глобального графа) имеет пропускную способность 16, удельную емкость 0.470431, удельное сопротивление 0.016209. Межслойные соединения между слоями 2 и 3 имеют высокую пропускную способность, нулевую емкость и сопротивление 0.00781. Третий уровень используется для соединений, параллельных оси 0*x*, имеет пропускную способность – 12, удельную емкость 0.456122 и удельное сопротивление 0.010763.

На рис. 3.8 приведен результат работы метода, который построил трассировку с плотностью (максимальным количеством соединений на одном ребре графа) 5. Время прихода сигнала (AT) в основной выход 12 – это 34 ps (пикосекунды), и AT в основной выход 13 – 36 ps. На рис. 3.9 приведена проекция деревьев на плоскость (*x*, *y*).

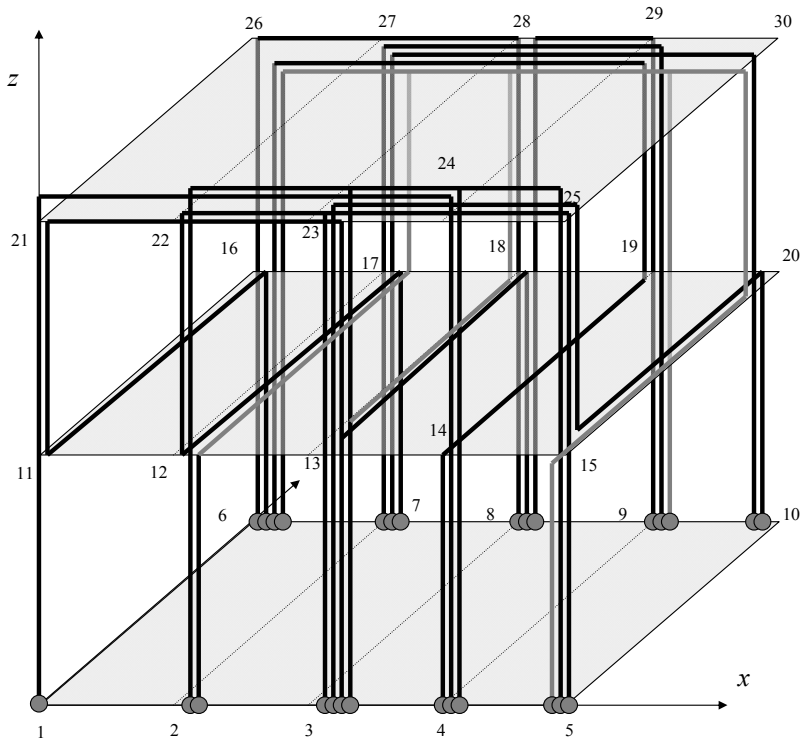


Рис. 3.8. Пример маршрутизации в 3-слойном глобальном графе

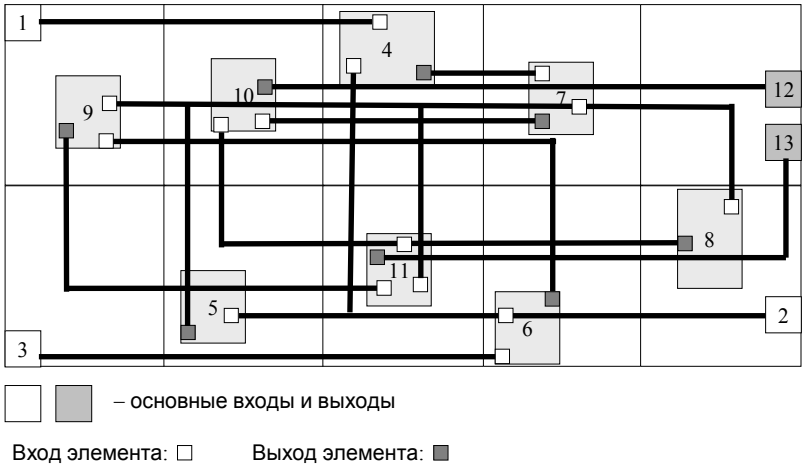


Рис. 3.9. Проекции деревьев

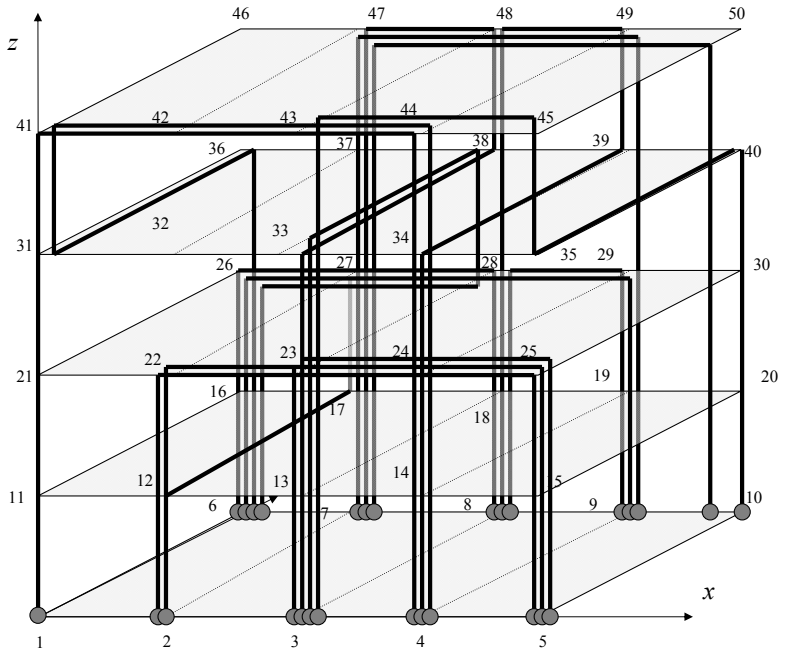


Рис. 3.10. 5-слойная маршрутизация (плотность равна 3)

Этот же пример был решен после добавления двух слоев (удельное сопротивление слоев 4 и 5 мы уменьшили в 10 раз по сравнению со слоями 2 и 3, удельная емкость и характеристики via оставили без изменения, как на слоях 2 и 3 соответственно). Алгоритм построил решение, показанное на рис. 3.10, с плотностью маршрутизации 3. Здесь мы не учитывали плотность via-соединений, т. к. пропускные способности таких ребер в примере не ограничены. Времена прихода сигнала в основные выходы схемы 12 и 13 соизмеримы с аналогичными временами для 3-слойной маршрутизации.

Упражнение 3.1. Построить МДШ в графе, изображенном на рис. 3.7*а*, для случая евклидовой метрики.

Упражнение 3.2. Построить МДШ в графе, изображенном на рис. 3.7*а*, в котором длины путей из левого нижнего терминала минимальны (для случая евклидовой метрики).

Упражнение 3.3. Построить МДШ в графе, изображенном на рис. 3.7*а*, в котором длины путей из левого нижнего терминала минимальны (для случая прямоугольной метрики).

Глава 4. Задача коммивояжера

В задаче коммивояжера задана матрица попарных расстояний между n городами. Требуется найти такой порядок посещения городов, чтобы пройденное расстояние было минимальным, каждый город посещался ровно один раз, и коммивояжер вернулся в тот город, с которого начал свой маршрут. Другими словами, во взвешенном полном неориентированном графе требуется найти гамильтонов цикл минимального веса. Если граф ориентированный и вес прямой дуги может не совпадать с весом обратной дуги, то требуется найти гамильтонов контур минимального веса.

Задача коммивояжера занимает особое место в комбинаторной оптимизации и исследовании операций. Исторически она была одной из тех задач, которые послужили толчком для развития этих направлений. Простота формулировки, конечность множества допустимых решений, наглядность и в тоже время колоссальные затраты на полный перебор до сих пор подталкивают математиков к разработке все новых и новых численных методов. Фактически, все свежие идеи сначала тестируются на этой задаче. С точки зрения приложений она не представляет интерес. Куда важнее ее обобщения для задач транспортировки грузов и логистики, когда несколько транспортных средств ограниченной грузоподъемности должны обслуживать клиентов, посещая их в заданные временные окна. В классической задаче коммивояжера все детали реальных приложений отсутствуют. Оставлена только комбинаторная суть, чисто математическая проблема, с которой не удается справиться уже полвека. Именно этой широко известной задаче посвящена данная глава.

4.1. Вычислительная сложность

Известно, что задача о гамильтоновости графа является NP-полной. В задаче коммивояжера требуется найти гамильтонов цикл минимальной длины. Это не задача распознавания. Она не лежит в классе NP, но она не проще проверки гамильтоновости графа. Действительно, если существует точный полиномиальный алгоритм для задачи коммивояжера, то можно легко построить точный полиномиальный алгоритм и для проверки гамильтоновости графа. Для

этого достаточно по графу $G = (V, E)$, чью гамильтоновость мы исследуем, построить матрицу расстояний (c_{ij}) задачи коммивояжера по следующему правилу:

$$c_{ij} = \begin{cases} 1, \text{если } (i,j) \in E \\ 2, \text{если } (i,j) \notin E \end{cases}, \quad i, j \in V.$$

Если на решении задачи коммивояжера функционал равен n, то граф G является гамильтоновым. Обратное тоже верно. Задачи вне класса NP, которые не проще NP-полных задач, называют *NP-трудными*. Задача коммивояжера принадлежит этому классу. Фактически мы получили доказательство даже более сильного утверждения.

Теорема 4.1. Задача коммивояжера является NP-трудной, даже когда матрица (c_{ij}) является симметричной и состоит из 1 и 2.

Легко проверить, что неравенство треугольника $c_{ij} \leq c_{ik} + c_{kj}$, $1 \leq i, j, k \leq n$, в этом случае также выполняется. Говорят, что задача является NP-трудной в сильном смысле, если она остается NP-трудной даже при унарной кодировке исходных данных. При двоичной кодировке целое число B требует $\lceil \log_2 B \rceil$ ячеек памяти. При унарной кодировке требуется B ячеек памяти. Переход к унарной кодировке влечет рост требуемой памяти и, как следствие, рост длины записи исходных данных. Грубо говоря, задача является NP-трудной в сильном смысле, если она остается NP-трудной, когда все числа в исходных данных ограничены некоторой константой. Из приведенного выше доказательства следует, что задача коммивояжера является NP-трудной в сильном смысле.

Полученное утверждение говорит о том, что точный полиномиальный алгоритм для задачи коммивояжера построить, скорее всего, не удастся. По-видимому, таких просто не существует. Следовательно, надо либо выйти за рамки полиномиальных алгоритмов, либо искать приближенные решения задачи. Следующая теорема говорит о том, что второй подход может оказаться не проще точного решения задачи. Пусть для примера I задачи коммивояжера величина $Opt(I)$ задает длину кратчайшего гамильтонова цикла, а некоторый алгоритм A на этом примере дает ответ $A(I)$.

Теорема 4.2. Если существует приближенный полиномиальный алгоритм A и такая положительная константа r, что для любого примера I задачи коммивояжера справедливо неравенство

$A(I) \leq r \cdot Opt(I)$, то классы P и NP совпадают.

Доказательство. Рассмотрим снова задачу о гамильтоновом цикле и построим матрицу (c_{ij}) по следующему правилу:

$$c_{ij} = \begin{cases} 1, & \text{если } (i,j) \in E \\ nr, & \text{если } (i,j) \notin E \end{cases}$$

Применим приближенный алгоритм A. Если $A(I) = n$, то граф содержит гамильтонов цикл.

Предположим, что $A(I) > n$. Тогда $A(I) \geq nr + (n-1)$, так как путь коммивояжера проходит как минимум через одно «тяжелое» ребро. Но в этом случае граф не может иметь гамильтонов цикл, так как алгоритм ошибается не более чем в r раз. Следовательно, полиномиальный алгоритм A дает правильный ответ для NP-полной задачи и P = NP. ■

Из доказательства теоремы следует, что константу r можно заменить на любую, например, экспоненциально растущую функцию от n. Повторяя рассуждения, получаем, что относительная точность полиномиальных алгоритмов в худшем случае не может быть ограничена, в частности, величиной $2^{p(n)}$, где $p(n)$ – произвольный полином от размерности задачи, если P ≠ NP.

Заметим, что, в отличие от доказательства теоремы 4.1, новая конструкция уже не гарантирует выполнение неравенства треугольника. Это наводит на мысль, что при дополнительных ограничениях на матрицу (c_{ij}) можно получить полиномиальные алгоритмы с гарантированной точностью. Это действительно так, и ниже будут представлены такие алгоритмы для матриц, удовлетворяющих неравенству треугольника. Подробный обзор таких алгоритмов можно найти, например, в [13].

Рассмотрим теперь другой класс алгоритмов – итерационные алгоритмы локального улучшения. Каждая итерация будет иметь полиномиальную трудоемкость, но число итераций может оказаться в худшем случае экспоненциальной величиной. Пусть C – гамильтонов цикл, а $N(C)$ – окрестность цикла C, то есть все циклы, которые можно получить из цикла C некоторой локальной перестройкой. Например, $N(C)$ – окрестность 2-замена: выбираем в C два несмежных ребра и меняем их на два других, чтобы снова получить гамильтонов цикл (рис. 4.1).

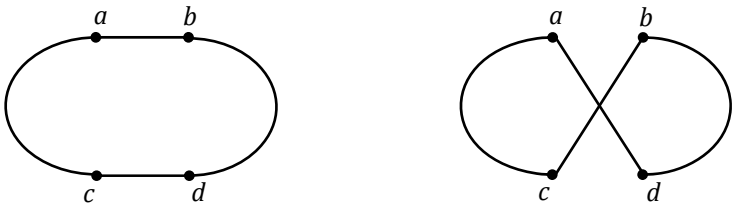


Рис. 4.1. Окрестность 2-замена

Мощность такой окрестности – $n(n-3)/2$. Аналогично можно получить окрестность 3-замена, 4-замена и т. д.

Алгоритм локального улучшения

1. Выбрать начальный цикл C и вычислить его длину $f(C)$.
2. Найти наилучшего соседа C' для цикла C:
$$f(C') = \min\{f(T) \mid T \in N(C)\}.$$
3. Если $f(C') < f(C)$, то положить $C := C'$ и вернуться на 2, иначе STOP, C – локальный минимум.

Теорема 4.3. Пусть A – алгоритм локального улучшения и окрестность $N(C)$ имеет полиномиальную мощность. Если существует такая положительная константа r, что для любого примера I задачи коммивояжера справедливо неравенство $A(I) \le r \cdot Opt(I)$, то классы P и NP совпадают.

Доказательство. (Аналогично предыдущему) Предположим, что такая константа существует. Рассмотрим задачу о гамильтоновом цикле и получим точный полиномиальный алгоритм ее решения. Снова по графу $G = (V, E)$ построим матрицу:

$$c_{ij} = \begin{cases} 1, \text{если } (i,j) \in E \\ nr, \text{если } (i,j) \notin E \end{cases}, \quad i,j \in V.$$

Выбираем произвольный начальный цикл. Его длина в худшем случае равна $n^2 r$. На каждом шаге локального улучшения часть тяжелых ребер заменяется легкими ребрами. Значит, число итераций не превышает n. Каждая итерация имеет полиномиальную трудоем-

кость, так как число соседних решений полиномиально. Следовательно, алгоритм A является полиномиальным для заданного класса примеров и, как следует из доказательства предыдущей теоремы, A – точный алгоритм. Значит, P = NP. ■

4.2. Конструктивные алгоритмы

Конструктивными алгоритмами принято называть такие алгоритмы, которые постепенно, шаг за шагом, строят допустимое решение задачи, например, добавляя одно ребро за другим. Рассмотрим один из конструктивных алгоритмов для задачи коммивояжера, известный под названием «*Иди в ближайший из непройденных городов*». Далее будем предполагать, что матрица (c_{ij}) удовлетворяет неравенству треугольника.

Алгоритм $A_Б$

1. Выбираем произвольный город, скажем, i_1, и помечаем его.
2. Цикл $k := 1, \dots, n-1$
   Находим ближайший город к i_k из непомеченных городов, обозначаем его i_{k+1}: $c_{i_k i_{k+1}} = \min_{j \neq i_1, \dots, i_k} c_{i_k j}$
   и помечаем город i_k.

Теорема 4.4. Для любого $r > 1$ найдется такой пример I задачи коммивояжера, что даже при выполнении неравенства треугольника справедливо неравенство $A_Б(I) > r \cdot Opt(I)$.

Рассмотрим пример задачи коммивояжера, в котором города расположены на окружности на одинаковом расстоянии друг от друга. Длина окружности здесь является оптимальным значением, но алгоритм $A_Б$ может сильно ошибаться. На рис. 4.2 показан пример для $n = 15$. Величина c_{ij} определяется как длина кратчайшего пути между i и j, получаемого с использованием ребер, изображенных на рисунке. Эти расстояния удовлетворяют неравенству треугольника. Длина окружности дает оптимальный маршрут 15, алгоритм $A_Б$ дает маршрут длины 27.

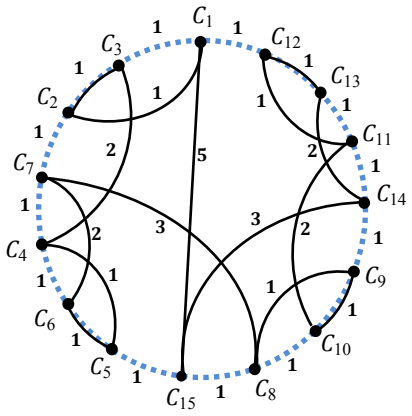


Рис. 4.2. Результат работы алгоритма $A_Б$

Рассмотрим теперь алгоритм, который имеет гарантированную точность $r = 2$. Он основан на следующей идее [14]. Для полного взвешенного графа задачи коммивояжера построим, например, алгоритмом Краскала A_K остовное дерево минимального веса. Заметим, что $A_k(I) \leq Opt(I)$ для любого примера I, т. к. удаление ребра из оптимального решения задачи коммивояжера дает остовное дерево. Заменим каждое ребро на два ребра. Получим эйлеров граф, каждая вершина которого имеет четную степень. Построим эйлеров цикл (рис. 4.3).

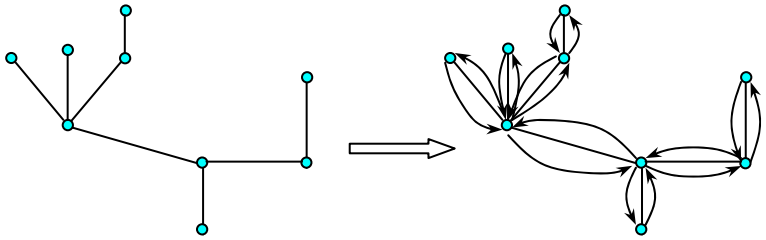


Рис. 4.3. Двойной обход остовного дерева

Перестроим его так, чтобы получился гамильтонов цикл. Начиная с произвольной вершины, двигаемся вдоль эйлерова цикла и помечаем вершины. Если очередная вершина уже помечена, то пропускаем ее и двигаемся дальше, пока не найдем непомеченную вершину или не вернемся в первую вершину. Цепочку дуг для помеченных вершин заменяем прямой дугой в непомеченную или первую вершину (рис. 4.4). Обозначим этот алгоритм A_{ST}.

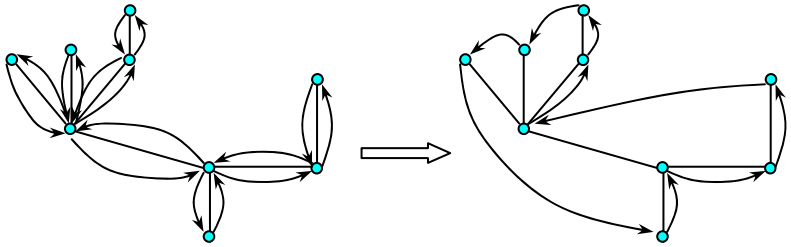


Рис. 4.4. Перестройка остовного дерева

Теорема 4.5. Для любого примера I задачи коммивояжера с симметричной матрицей (c_{ij}), удовлетворяющей неравенству треугольника, алгоритм A_{ST} получает гамильтонов цикл не более чем в два раза длиннее оптимального, то есть $A_{ST}(I) \leq 2Opt(I)$.

Доказательство. Для двойного обхода остовного дерева имеем $2A_K(I) < 2Opt(I)$. Пусть новое ребро (i,j), не содержащееся в двойном обходе, заменяет цепочку ребер $(i,k_1),(k_1,k_2),\dots,(k_m,j)$. Из неравенства треугольника следует, что $c_{ij} \leq c_{ik_1} + \cdots + c_{k_m j}$. Следовательно, $A_{ST}(I) \leq 2A_K(I) \leq 2Opt(I)$.

Теорема 4.6. Оценка точности $r = 2$ является неулучшаемой для алгоритма A_{ST}.

Доказательство. Приведем семейство исходных данных задачи коммивояжера, на котором оценка 2 достигается асимптотически. Рассмотрим следующий пример на евклидовой плоскости с $3(n+1)$ вершинами (рис. 4.5).

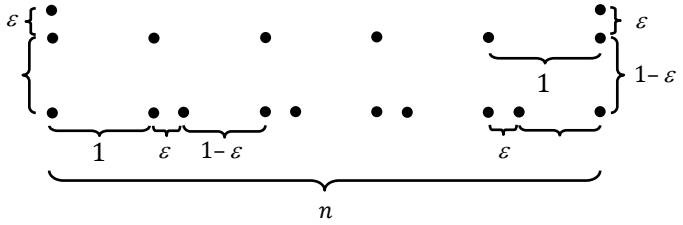


Рис. 4.5. Пример на евклидовой плоскости

Для этого примера минимальное остовное дерево имеет вид (рис. 4.6.):

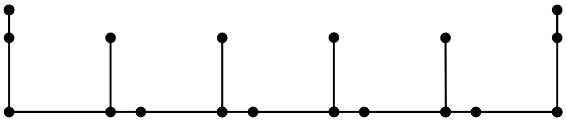


Рис. 4.6. Минимальное остовное дерево

Длина остовного дерева $A_K = n + (n + 1)(1 - \varepsilon) + 2\varepsilon$. Алгоритм перестройки двойного обхода получит решение (рис. 4.7):

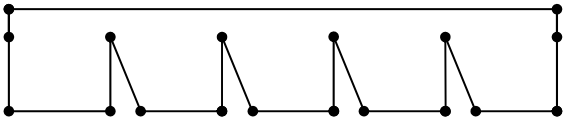


Рис. 4.7. Результат алгоритма A_{ST}

Длина этого гамильтонова цикла $A_{ST} \approx 2n + 2n\,(1 - \varepsilon)$ (рис. 4.8). Оптимальное решение задачи $Opt(I) \approx 2_n + 2$.

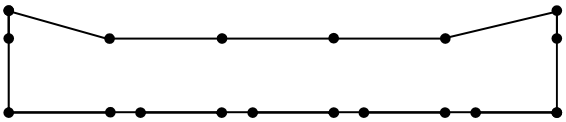


Рис. 4.8. Оптимальное решение

Таким образом, при $n \to \infty$ и $\varepsilon \to 0$ получаем $A_{ST}\ (I)\ /\ Opt(I) \to 2$. ■

Теоремы 4.5 и 4.6 говорят не только о точности алгоритма A_{ST} в худшем случае. Они утверждают также, что оценка справедлива для любого эйлерова цикла, который будет перестраиваться в гамильтонов цикл. Но эйлеровых циклов может быть экспоненциально много. Выбирая из всех вариантов наилучший, можно существенно понизить ошибку. В. Дейнеко и А. Тискин [15] нашли точный полиномиальный алгоритм для выбора наилучшего эйлерова цикла. Оценка точности в худшем случае не изменилась, но поведение алгоритма в среднем показывает, что отклонение от нижней оценки оптимума не превышает 10 %. На сегодняшний день это один из лучших полиномиальных алгоритмов с точки зрения средней погрешности.

В заключение этого раздела приведем еще один полиномиальный алгоритм, предложенный независимо Н. Кристофидесом и А. Сердюковым. Эта улучшенная версия предыдущего алгоритма имеет оценку точности 3/2. Как и раньше, сначала строится остовное дерево минимального веса, затем к нему добавляются ребра так, чтобы получился эйлеров граф, а потом строится эйлеров цикл, который перестраивается в гамильтонов цикл. Новшество состоит в добавлении ребер для получения эйлерова графа. Вместо дублирования ребер остова в дереве выделяется множество вершин V' нечетной степени (рис. 4.9).

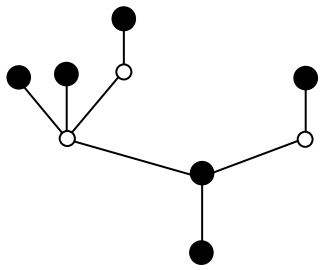


Рис. 4.9. Построение множества V'

Их число четно, $|V'| = 2k$, так как сумма всех степеней вершин графа должна быть четной. На множестве V' находится совершенное паросочетание минимального веса: k ребер, имеющие минимальный суммарный вес и покрывающие все вершины. Эта задача полиномиально разрешима [17]. Более того, вес такого паросочетания не пре-

восходит половины длины любого гамильтонова цикла. Действительно, гамильтонов цикл в исходном графе легко переделать в гамильтонов цикл для подграфа на вершинах из V'. Этого можно добиться, исключив вершины, не принадлежащие V'. В силу неравенства треугольника получим гамильтонов цикл не большей длины, чем исходный цикл. Он задает на множестве V' два паросочетания. Они получаются, если брать ребра через одно. Наименьший из весов этих паросочетаний не превосходит половины длины гамильтонова цикла. Добавим это паросочетание к остовному дереву (рис. 4.10).

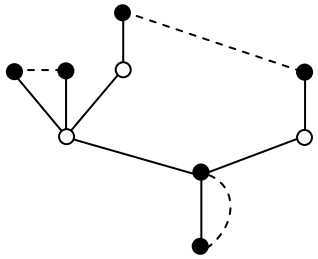


Рис. 4.10. Остовное дерево вместе с паросочетанием

Получим эйлеров граф. Его вес не превосходит 3/2 длины минимального гамильтонова цикла. Строим эйлеров цикл (рис. 4.11).

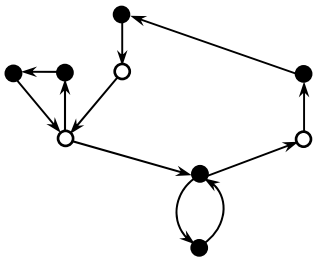


Рис. 4.11. Эйлеров цикл

Перестраиваем эйлеров цикл в гамильтонов цикл описанным выше способом (рис. 4.12). Можно показать, что эту оценку нельзя улучшить, т. е. существуют примеры, на которых этот алгоритм действительно ошибается в 3/2 раза.

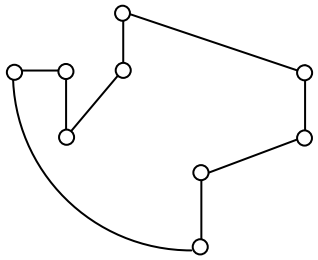


Рис. 4.12. Гамильтонов цикл

Заметим, что новый подход снова порождает эйлеров граф, в котором может оказаться экспоненциально много эйлеровых циклов. Выбирая наилучший из них, можно существенно понизить погрешность получаемых приближенных решений. К сожалению, выбор наилучшего эйлерова цикла в данном случае оказывается NP-трудной задачей. Тем не менее, реализация этой идеи, пусть даже в виде приближенного решения, приводит, как и в предшествующем случае, к решениям с малой погрешностью.

4.3. Нижние оценки

Получение нижних оценок оптимума является важной задачей. Во-первых, они позволяют оценить погрешность конструктивных алгоритмов и вообще любых приближенных алгоритмов. Во-вторых, эти оценки используются в точных методах, например, методе ветвей и границ [12, 16]. Их качество часто оказывается критическим фактором при получении глобального оптимума. Ниже будут представлены четыре способа построения нижних оценок, каждый из которых имеет свои сильные и слабые стороны.

4.3.1. Примитивная нижняя оценка

Пусть величины $a_i = \min_{j \neq i} c_{ij},\ i = 1, \dots, n,$ задают минимальную плату за выезд из города. Так как коммивояжер должен выехать из каждого города, то $\sum_{i=1}^{n} a_i$ является, очевидно, нижней оценкой оптимума. Рассмотрим теперь матрицу $c'_{ij} = c_{ij} - a_i$ и для каждого города посчитаем минимальную плату за въезд: $b_j = \min_{j \neq i} c'_{ij}$, $j = 1, \dots, n.$ Тогда величина $H(c_{ij}) = \sum_{i=1}^{n} a_i + \sum_{j=1}^{n} b_j$ будет также нижней оценкой.

Теорема 4.7. Величина $H(c_{ij})$ является нижней оценкой оптимума.

Доказательство. Оптимальные решения для матриц (c_{ij}) и (c'_{ij}) связаны соотношением $Opt(c_{ij}) = Opt(c'_{ij}) + \sum_{i=1}^{n} a_i$. Положим $c''_{ij} = c'_{ij} - b_j,\ 1 \le i, j \le n$. Тогда

$Opt(c_{ij}) = Opt(c''_{ij}) + \sum_{i=1}^{n} a_i + \sum_{j=1}^{n} b_j \ge \sum_{i=1}^{n} a_i + \sum_{j=1}^{n} b_j$. ■

Основным достоинством этой нижней оценки является ее простота и наглядность. Ее вычисление требует $O(n^2)$ операций, то есть линейной трудоемкости от числа элементов матрицы расстояний.

4.3.2. Оценка линейного программирования

Представим задачу коммивояжера в терминах целочисленного линейного программирования. Введем переменные

$$x_{ij} = \begin{cases} 1, & \text{если из города } i \text{ едем в город } j \\ 0 & \text{в противном случае} \end{cases}.$$

Задача коммивояжера может быть записана следующим образом:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

при ограничениях:

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad 1 \le j \le n,$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad 1 \le i \le n,$$

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \ge 1, \qquad S \ne \emptyset,\ S \subset \{1, \dots, n\},$$

$$x_{ij} \in \{0,1\}, \qquad 1 \le i, j \le n.$$

Целевая функция задает длину гамильтонова цикла. Первое и второе равенство требуют въехать и выехать из каждого города соответственно. Однако этого недостаточно. Набор подциклов, по-

крывающий все города, будет удовлетворять этим равенствам, но не будет гамильтоновым циклом. Поэтому нужно дополнительное неравенство для исключения подциклов. Это неравенство требует для любого подмножества городов S существование хотя бы одной дуги, ведущей в остальные города.

Заменим условие целочисленности переменных на условие $0 \le x_{ij} \le 1,\ 1 \le i,j \le n$. Получим задачу линейного программирования. Оптимальное значение в этой задаче будет нижней оценкой, потому что такая замена расширяет область допустимых решений. Нетрудно показать, что эта оценка не хуже примитивной оценки.

Задача линейного программирования полиномиально разрешима [17]. Однако не стоит торопиться с выводами. Для задачи линейного программирования действительно есть точный алгоритм, трудоемкость которого полиномиально зависит от длины записи исходных данных. Но в задаче коммивояжера только n^2 чисел. Если они ограничены константой, то длина входа – $O(n^2)$. В нашей задаче линейного программирования вход значительно длиннее. Только ограничений для исключения подциклов – $O(2^n)$. Фактически мы не можем даже выписать эту задачу из-за огромного числа ограничений. Ситуация кажется тупиковой. Тем не менее, из нее есть как минимум два выхода. Первый состоит в замене условий на подциклы другим условием, имеющим полиномиальное число ограничений. Это можно сделать несколькими способами (см., например, [18]). Тогда нижняя оценка становится полиномиально вычислимой, но ее качество заметно падает. Второй способ состоит в последовательном наращивании ограничений на подциклы по мере необходимости. Удалим сначала все такие ограничения и решим задачу. Найдем по заданному решению подмножество S, для которого нарушается условие на подциклы, и добавим его в систему ограничений. Действуя таким способом, можно последовательно улучшать нижнюю оценку до тех пор, пока либо не получим точное решение исходной задачи линейного программирования, либо задача не станет слишком большой для расчетов. В любом случае получаем нижнюю оценку оптимума.

4.3.3. 1-деревья для симметричных матриц

Вес минимального остовного дерева, как мы уже знаем, дает оценку снизу, но ее можно улучшить. Остовное дерево состоит из $(n-1)$ ребра, а гамильтонов цикл содержит n ребер. Добавление одного ребра к остовному дереву порождает так называемые *1-деревья*. Они тоже дают нижние оценки. Итак, мы хотим найти гамильтонов цикл минимального веса, т. е. ровно n ребер, которые:

- покрывают все вершины и образуют связный граф,
- имеют минимальный суммарный вес,
- каждая вершина инцидентна ровно двум ребрам.

Ослабим последнее условие, заменив его следующим:

- одна заданная вершина инцидентна ровно двум ребрам.

Так как мы ослабили одно из условий, то получим нижнюю оценку. Алгоритм построения минимального по весу 1-дерева для выделенной вершины состоит в следующем. Удаляем эту вершину и смежные с ней ребра. На получившемся графе строим остовное дерево минимального веса. Получаем $n-2$ ребер. Добавляем два ребра, инцидентных выделенной вершине, имеющих минимальный суммарный вес. Полученное 1-дерево дает нижнюю оценку. Она зависит от выбора вершины. Построив 1-деревья для всех вершин и выбрав из нижних оценок наибольшую, получим требуемое.

4.3.4. Задача о назначениях

Рассмотрим следующую задачу о назначении рабочих на станки [12, 17]. Дано: n рабочих, n станков и матрица (c_{ij}), задающая время выполнения работы на j-м станке i-м рабочим. Требуется найти расстановку рабочих по станкам, которая дает наименьшее суммарное рабочее время. Введем переменные задачи

$$x_{ij} = \begin{cases} 1, & \text{если рабочий } i \text{ работает на станке } j \\ 0 & \text{в противном случае} \end{cases}.$$

Задача о назначениях может быть записана в следующем виде:

$$\min \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}$$

при ограничениях

$$\sum_{i=1}^{n} x_{ij} = 1, \qquad 1 \le j \le n,$$

$$\sum_{j=1}^{n} x_{ij} = 1, \qquad 1 \le i \le n,$$

$$x_{ij} \in \{0,1\}, \qquad 1 \le i, j \le n.$$

Сравнивая эту запись с аналогичной записью для задачи коммивояжера, легко заметить, что они отличаются только одной группой ограничений для исключения подциклов. Следовательно, оптимальное решение задачи о назначениях дает нижнюю оценку для задачи коммивояжера.

Приведем точный полиномиальный алгоритм решения задачи о назначениях. Пусть $\Delta = (\Delta_1, \dots, \Delta_n)$ – некоторый вектор. Элемент c_{ij} называется Δ-минимальным для строки i, если $c_{ij} - \Delta_j \le c_{ik} - \Delta_k$ для всех $k = 1, \dots, n$.

Теорема 4.8. Пусть для некоторого вектора Δ существует набор Δ-минимальных элементов $(c_{1j(1)}, \dots, c_{nj(n)})$ по одному в каждой строке и каждом столбце. Тогда этот набор является оптимальным решением задачи о назначениях.

Доказательство. Решение $(c_{1j(1)}, \dots, c_{nj(n)})$ является допустимым, и

$$\sum_{i=1}^{n} c_{ij(i)} = \sum_{i=1}^{n} (c_{ij(i)} - \Delta_{j(i)}) + \sum_{j=1}^{n} \Delta_j.$$

В правой части равенства первая сумма является минимальной среди всех допустимых назначений. Вторая сумма является константой. Значит, полученное решение является оптимальным. ■

Определение 4.1. Для вектора Δ выделим в каждой строке по одному Δ-минимальному элементу и назовем его *Δ-основой*. Другие Δ-минимальные элементы будем называть *альтернативными Δ-основами.* Число столбцов матрицы (c_{ij}) без Δ-основ назовем *дефектом.*

Идея алгоритма состоит в следующем. Начинаем с нулевого вектора Δ и считаем дефект. На каждом этапе алгоритма дефект

уменьшается на 1, то есть не более чем за n этапов получим оптимальное решение.

Описание одного этапа

1. Выберем столбец без Δ-основы и обозначим его S_1.
2. Увеличим Δ_{S_1} на максимальное δ так, чтобы все Δ-минимальные элементы остались Δ-минимальными (возможно $\delta = 0$). Получим для некоторой строки i_1 новый Δ-минимальный элемент $c_{i_1 S_1}$, назовем его альтернативной основой для строки i_1 (рис. 4.13).

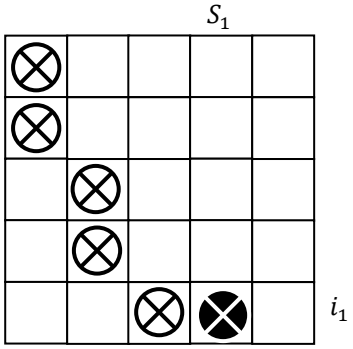


Рис. 4.13. Появление альтернативной основы

3. Для строки i_1 столбец $j(i_1)$ с Δ-основой пометим меткой S_2. (рис. 4.14).

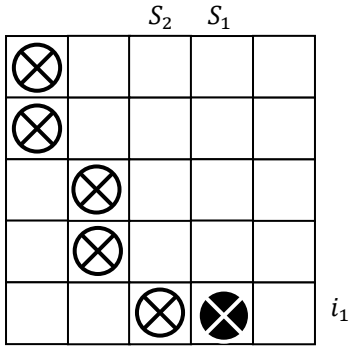


Рис. 4.14. Выбор второго столбца

4. Увеличим Δ_{s_1} и Δ_{s_2} на максимальное δ так, чтобы все Δ-основы остались Δ-минимальными элементами. Найдем новую альтернативную основу в одном из столбцов S_1 или S_2 . Пусть она оказалась в строке i_2. Пометим столбец $j(i_2)$ меткой S_3 (рис. 4.15) и будем продолжать этот процесс до тех пор, пока не встретим столбец с двумя или более основами.

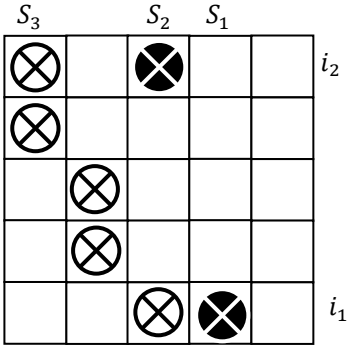


Рис. 4.15. Получение новой основы

5. Строим новый набор Δ-основ. Заменой основы в строке назовем следующую операцию: альтернативная основа становится основой, а старая перестает быть основой. Произведем замену основ в строке, где лежит последняя альтернативная основа (строка i_k). Тогда в столбце $j(i_k)$ число основ уменьшится на 1, но останется положительным (рис. 4.16).

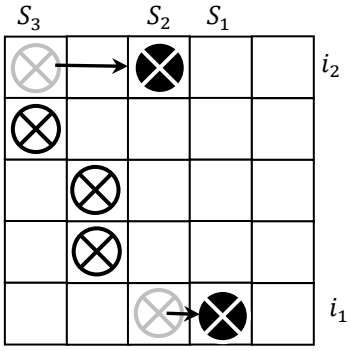


Рис. 4.16. Замена основы

В столбце, где появилась новая основа, возьмем старую основу и в этой строке тоже проведем замену основ и так будем переходить от одного столбца к другому, пока не доберемся до столбца S_1. В итоге, столбец S_1 получит основу, а число основ в столбце $j(i_k)$ уменьшится на 1 (рис. 4.17).

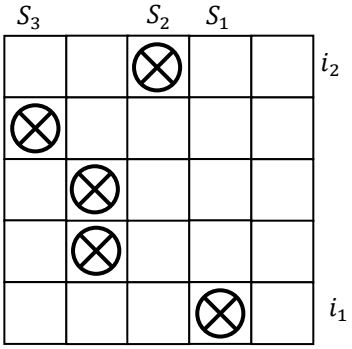


Рис. 4.17. Новое распределение основ

Посчитаем трудоемкость одного этапа. Мы последовательно рассматриваем сначала один столбец S_1, затем два столбца S_1, S_2 и т. д. Всего шагов может быть не больше n. На каждом шаге надо выбрать δ, что требует $O(n^2)$ операций. Значит, всего требуется $O(n^3)$ операций для одного этапа и $O(n^4)$ операций для всего алгоритма.

4.4. Локальный поиск

Идеи локального поиска при решении задач дискретной оптимизации являются, по-видимому, наиболее естественными и наглядными. Первые шаги их реализации относятся к началу 50-х, началу 60-х годов XX столетия. Они связаны, в основном, с задачей коммивояжера. Позднее эти идеи использовались для задач размещения, построения сетей, расписаний и др. Однако, довольно быстро выяснилось, что методы локального улучшения не гарантируют нахождения глобального оптимума, и отсутствие концептуального прогресса ослабило интерес к данному направлению. В последние 20 лет наблюдается возрождение этого подхода. Оно связано как с новыми алгоритмическими схемами, построенными на аналогиях с живой и неживой природой, так и с новыми теоретическими резуль-

татами в области локального поиска. Изменился общий взгляд на построение локальных алгоритмов. Требование монотонного улучшения по целевой функции больше не является доминирующим. Наиболее мощные алгоритмы допускают произвольное ухудшение, и многие из них могут рассматриваться как способ порождения конечных неразложимых цепей Маркова на подходящем множестве состояний. В данном разделе приводится краткое введение в эту бурно развивающуюся область дискретной оптимизации.

4.4.1. Основные определения

Пусть тройка $(\Im, Sol, f)$ задает задачу дискретной оптимизации с множеством входов $\Im$ (исходных данных), конечным множеством допустимых решений $Sol(I), I \in \Im$, и целевой функцией $f: s \to R$, $s \in Sol(I)$. Для определенности будем считать, что требуется найти минимум функции f на множестве $Sol(I)$ и само решение $s^* \in Sol(I)$, на котором этот минимум достигается. Решение s^* будем называть *глобальным минимумом* и множество глобальных минимумов будем обозначать через S^*. Для каждого $s \in Sol(I)$ определим функцию окрестности $N: Sol(I) \to 2^{Sol(I)}$, которая для каждого допустимого решения задает множество соседних решений, в некотором смысле близких к данному. Множество $N(s)$ будем называть окрестностью решения s в множестве $Sol(I)$. Функция окрестности может быть достаточно сложной, и отношение соседства не всегда симметрично.

Определение 4.2. Решение $s \in Sol(I)$ называется *локальным минимумом* по отношению к функции N, если $f(s) \leq f(s')$ для всех $s' \in N(s)$.

Множество локальных минимумов обозначим через $\hat{S}$. Это множество, очевидно, зависит от выбора функции N.

Определение 4.3. Функция окрестности N называется *точной*, если $\hat{S} \subseteq S^*$.

Стандартный алгоритм локального улучшения начинает работу с некоторого начального решения, выбранного случайным образом или с помощью какого-либо вспомогательного алгоритма. На каждом шаге локального улучшения происходит переход от текущего

решения к соседнему решению с меньшим значением целевой функции до тех пор, пока не будет достигнут локальный минимум.

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

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

2. Минимальное остовное дерево. Остовное дерево не является оптимальным, если и только если локальной перестройкой, добавляя одно ребро и удаляя из образовавшегося цикла другое ребро, можно получить новое остовное дерево с меньшим суммарным весом. Операция локальной перестройки задает отношение соседства на множестве остовных деревьев. Окрестность любого дерева имеет полиномиальную мощность, а функция окрестности является точной.

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

совершенных паросочетаний минимального веса, задач о максимальном потоке и потоке минимальной стоимости.

На каждом шаге локального улучшения функция окрестности задает множество возможных направлений движения. Часто это множество состоит из нескольких элементов и имеется определенная свобода в выборе следующего решения. Правило выбора может оказать существенное влияние на трудоемкость алгоритма и результат его работы. Например, в задаче о максимальном потоке алгоритм Форда – Фалкерсона (который тоже можно рассматривать как вариант локального улучшения) имеет полиномиальную временную сложность при выборе кратчайшего пути для увеличения потока и экспоненциальную временную сложность без гарантии получить глобальный оптимум при произвольном выборе пути. Таким образом, при разработке алгоритмов локального поиска важно не только правильно определить окрестность, но и верно задать правило выбора направления спуска.

Интуитивно кажется, что в окрестности надо брать элемент с наименьшим значением целевой функции. Однако, как мы увидим ниже, разумным оказывается не только такой выбор, но и движение в «абсурдном» направлении, когда несколько шагов с ухудшением могут привести (и часто приводят) к лучшему локальному минимуму. При выборе окрестности хочется иметь как можно меньше соседей, чтобы сократить трудоемкость одного шага. С другой стороны, более широкая окрестность, вообще говоря, приводит к лучшему локальному оптимуму. Поэтому при создании алгоритмов каждый раз приходится искать оптимальный баланс между этими противоречивыми факторами. Ясных принципов разрешения этих противоречий на сегодняшний день нет, и для каждой задачи этот вопрос решается индивидуально.

Определение 4.4. *Графом соседства* $G_N = (Sol(I), E)$ будем называть взвешенный ориентированный граф, вершинами которого являются допустимые решения задачи, а дугами – упорядоченные пары (s, s'), если $s' \in N(s)$. Веса в этом графе приписаны вершинам и равны соответствующим значениям целевой функции.

Граф соседства G_N (*neighborhood graph*) иногда называют ландшафтом целевой функции или просто ландшафтом (*landscape, fitness landscape*). При определении функции окрестности важно

следить за тем, чтобы получающийся граф был строго связен, то есть для каждой пары вершин s, s' существовал путь из s в s'. Это свойство является важным при анализе асимптотического поведения алгоритмов, например, вероятностных метаэвристик, о которых пойдет речь ниже. Если же это свойство не выполняется, то стараются получить хотя бы свойство вполне связности, когда из любой вершины существует путь в вершину $s^* \in S^*$. Если же и этого свойства нет, то теряется уверенность в достижении глобального оптимума локальными методами. Приходится ограничиваться локальными оптимумами, либо переопределять функцию окрестности.

Анализ эффективности локального поиска условно можно разделить на два направления: эмпирические и теоретические исследования. Как это ни странно, но они дают разные оценки возможностям этого подхода [19, 20].

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

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

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

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

1) минимальная точная окрестность может иметь экспоненциальную мощность;

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

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

Эти результаты подталкивают к более пристальному рассмотрению задачи нахождения локального оптимума, ее трудоемкости в среднем и худшем случаях. Абстрактная (массовая) задача локального поиска L задается множеством ее индивидуальных примеров, каждый из которых однозначно определяет целевую функцию, функцию окрестности и множество допустимых решений.

Определение 4.5. Задача L принадлежит классу *PLS (polynomial time local search)*, если за полиномиальное время от длины записи исходных данных можно выполнить следующие три операции:

1) Для произвольного примера I проверить его корректность и, если $I \in \mathfrak{I}$, найти некоторое допустимое решение.

2) Для любого решения проверить, является ли оно допустимым и, если да, вычислить значение целевой функции для него.

3) Узнать, является ли данное решение локальным оптимумом и, если нет, найти в его окрестности решение с лучшим значением целевой функции.

Грубо говоря, в классе PLS содержатся все задачи локального поиска, для которых проверка локальной оптимальности может быть осуществлена за полиномиальное время. Этот класс не пуст, так как задача коммивояжера с окрестностью 2-замена содержится в нем. Следующая теорема говорит о том, что, скорее всего, в этом классе нет NP-трудных задач.

Теорема 4.9. Если задача L из класса PLS является NP-трудной, то NP = co-NP.

В классе PLS содержатся полиномиально разрешимые задачи, то есть задачи, для которых можно найти локальный оптимум за полиномиальное время. Многие NP-трудные задачи без весовых функций с любой полиномиально проверяемой окрестностью относятся к таковым: задачи о клике, о покрытии, о независимом множестве и др. Задача коммивояжера с матрицей (c_{ij}), состоящей из 1 и 2, будет таковой при любой полиномиально проверяемой окрестности. Однако, в общем случае вопрос о вычислительной сложности нахождения локального оптимума пока остается открытым. Так же, как и для задач распознавания, в классе PLS можно определить сведение одной задачи к другой и доказать теорему о существовании PLS-полных задач. Это наиболее трудные задачи в этом классе. Существование полиномиального алгоритма хотя бы для одной из них влечет полиномиальную разрешимость для всех задач из класса PLS. Задача коммивояжера с окрестностью k-замена является PLS-полной при $k \geq 8$. Более того, удается показать, что стандартный алгоритм локального улучшения для этой задачи требует в худшем случае экспоненциального числа итераций независимо от того, какое правило замещения будет применяться. Более подробно с теорией PLS-полных задач можно познакомиться в [19, 21].

4.4.2. Окрестности на перестановках

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

Будем по-прежнему рассматривать задачу коммивояжера с симметричной матрицей расстояний и гамильтонов цикл представлять в виде перестановки $\pi = \{i_1, \dots, i_n\}$. Определим окрестность $N(\pi)$ как множество всех перестановок, отличающихся от π только в двух позициях ($city\text{-}swap$). Множество $N(\pi)$ содержит ровно $n(n-1)/2$ элементов, и вычислительная сложность одного шага локального поиска с учетом вычисления целевой функции не превосходит $O(n^2)$ операций.

Обозначим через $f(\pi)$ длину гамильтонова цикла и определим разностный оператор ∇^2 для $f(\pi)$ следующим образом [22]:

$$\nabla^2 f(\pi) = \frac{1}{|N(\pi)|} \sum_{\pi' \in N(\pi)} f(\pi') - f(\pi).$$

Этот оператор задает среднее отклонение целевой функции в окрестности данной перестановки. Для локального оптимума π справедливо неравенство $\nabla^2 f(\pi) \geq 0$. Пусть f_{av} – средняя длина цикла на множестве всех допустимых решений задачи. Тогда справедливы следующие утверждения.

Теорема 4.10. Функция $f' = f - f_{av}$ удовлетворяет уравнению

$$\nabla^2 f' = -\frac{4}{n} f'.$$

Следствие 4.1. Любой локальный минимум π имеет длину $f(\pi) \leq f_{av}$.

Следствие 4.2. Алгоритм локального поиска, начиная с произвольной перестановки, достигнет локального оптимума за $O(nk)$ шагов, если длина максимального тура превосходит среднее значение f_{av} не более чем в 2^k раз.

Аналогичные утверждения справедливы и для следующих задач:

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

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

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

4) о 3-выполнимости в следующей постановке: для n булевых переменных задан набор троек; каждая переменная может входить в

набор с отрицанием или без него; набор считается выполненным, если он содержит разные значения, то есть хотя бы одно *истинное* и хотя бы одно *ложное*; требуется узнать, существует ли назначение переменных, при котором все наборы будут выполнены. Развитие этих идей можно найти в [23].

4.4.3. Окрестности Лина – Кернигана

Рассмотрим гамильтонов цикл как последовательность ребер. Напомним, что окрестностью 2-замена называют множество всех гамильтоновых циклов, получающихся из заданного заменой двух несмежных ребер. Такая окрестность содержит $n(n-3)/2$ элементов, что несколько меньше, чем в окрестности city-swap. На рис. 4.18 показано различие между этими окрестностями. Для окрестности 2-замена удаление ребер (a,b) и (e,f) приводит к появлению двух новых ребер (a,e) и (b,f) и обратному обходу дуг от d до c. Если в окрестности city-swap в перестановке поменять местами b и e, то появятся четыре новых ребра $(a,e),(e,c),(d,b)$ и (b,f), но обход дуг от d до c останется прежним.

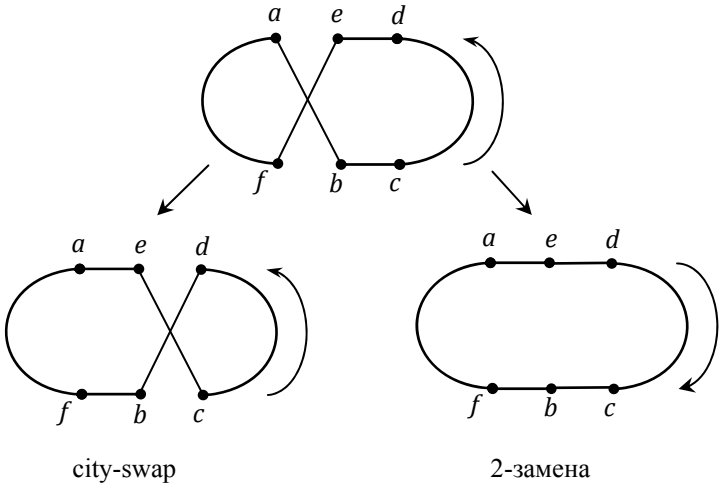


Рис. 4.18. Различия между окрестностями

Если вершины расположены на плоскости, а веса ребер вычисляются как евклидовы расстояния, то пересечение ребер (a, b) и (e, f) свидетельствует о возможности улучшения данного тура в силу неравенства треугольника. Однако это условие не является необходимым, и улучшение возможно даже без пересечения ребер.

На основе окрестности 2-замена Лином и Керниганом предложена оригинальная эвристика. Она позволяет заменять произвольное число ребер и переходить от одного тура к другому, используя принципы жадных алгоритмов. Основная идея эвристики заключается в следующем. Удалим из гамильтонова цикла произвольное ребро, например (a, b). В полученном пути один конец (вершину a) будем считать фиксированным, а другой конец будем менять, перестраивая гамильтонов путь. Добавим ребро из вершины b, например (b, c), и разорвем образовавшийся единственный цикл так, чтобы снова получить гамильтонов путь. Для этого придется удалить ребро, инцидентное вершине c. Обозначим его (c, d). Новый гамильтонов путь имеет концевые вершины a и d (рис. 4.19). Эту процедуру будем называть ротацией. Для получения нового гамильтонова цикла достаточно добавить ребро (a, d). Согласно алгоритму Лина – Кернигана, переход от одного тура к другому состоит из удаления некоторого ребра, выполнения серии последовательных ротаций и, наконец, замыкания концевых вершин полученного гамильтонова пути. Существуют различные варианты этой основной схемы, которые отличаются правилами выбора ротаций и ограничениями на множества удаляемых и добавляемых ребер. В алгоритме Лина – Кернигана ротации выбираются так, чтобы минимизировать разность $c_{bc} - c_{cd}$. При этом множества удаляемых и добавляемых ребер в серии ротаций не должны пересекаться. Последнее ограничение гарантирует, что число ротаций не превысит n^2 и трудоемкость одного шага (перехода от одного цикла к другому) останется полиномиальной. Общее число шагов алгоритма, по-видимому, не может быть ограничено полиномом, и известен вариант окрестности Лина – Кернигана, для которого задача коммивояжера становится PLS-полной [19].

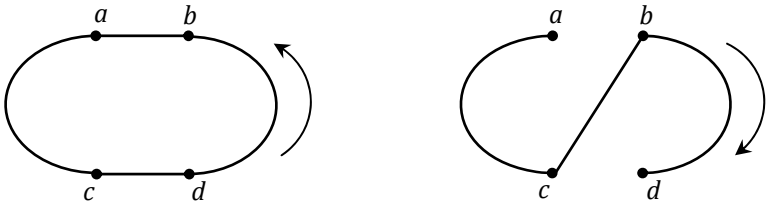


Рис. 4.19. Процедура ротации

4.4.4. Экспоненциальные окрестности

Одним из новых перспективных направлений в области локального поиска является исследование свойств окрестностей экспоненциальной мощности. Если лучшее решение в такой окрестности можно найти за полиномиальное время, то алгоритм локального поиска с такой окрестностью получает определенное преимущество перед остальными. Для задачи коммивояжера первые примеры таких окрестностей были предложены в работе [24]. Изложим, в качестве иллюстрации, идею одного из таких подходов. Этот пример наглядно показывает тесную связь между экспоненциальными окрестностями и полиномиально разрешимыми случаями задачи коммивояжера. Наличие такой связи, по-видимому, будет стимулировать поиск новых полиномиально разрешимых случаев для трудных комбинаторных задач, что в свою очередь может привести к дальнейшему прогрессу в области локального поиска.

Основная идея может быть представлена следующим образом. Предположим, что множество из n городов разбито на две части $x_1, \dots, x_m$ и $y_1, \dots, y_k$, так, что $k + m = n$ и $m \leq k$. Для удобства и простоты изложения добавим в первую группу $k - m$ фиктивных городов $x_{m+1}, \dots, x_k$ и рассмотрим окрестность $N(y_1, \dots, y_k, x_1, \dots, x_m)$, состоящую из всех циклов вида

$$y_1 x_{\tau_1} y_2 x_{\tau_2}, \dots, y_k x_{\tau_k} y_1,$$

где $(\tau_1, \dots, \tau_k)$ – произвольная перестановка k элементов. Содержательно, каждый такой цикл получается вставкой городов x_i между парами городов y_j, y_{j+1}. Множество $N(y_1, \dots, y_k, x_1, \dots, x_m)$ имеет экспоненциальную мощность $\left[\frac{n+1}{2}\right]!$ при $2m \leq n \leq 2m + 1$. Оптималь-

ный тур в этом множестве может быть найден с помощью решения задачи о назначениях следующим образом. При $j = 1, \dots, k$ положим

$$d_{ij} = \begin{cases} c(y_j x_i) + c(x_i y_{j+1}), & i = 1, \dots, m \\ c(y_j y_{j+1}), & i = m+1, \dots, k. \end{cases}$$

Введем переменные

$$z_{ij} = \begin{cases} 1, & \text{если } x_i \text{ располагается между } y_j, y_{j+1} \\ 0 & \text{в противном случае} \end{cases}$$

и рассмотрим задачу

$$\min \sum_{i=1}^{k} \sum_{j=1}^{k} d_{ij} z_{ij}$$

$$\sum_{i=1}^{k} z_{ij} = 1, \ \ j = 1, \dots, k,$$

$$\sum_{j=1}^{k} z_{ij} = 1, \ \ i = 1, \dots, k,$$

$$z_{ij} \in \{0,1\}, \quad i, j = 1, \dots, k.$$

Оптимальное решение задачи определяет наилучшее размещение городов $x_i, i = 1, \dots, m$, между городами $y_j, j = 1, \dots, k$, то есть оптимальный тур в множестве $N(y_1, \dots, y_k, x_1, \dots, x_m)$.

Заметим, что максимальная мощность окрестности $N(y_1, \dots, y_k, x_1, \dots, x_m)$ достигается при $m < n/2$. Точнее, пусть $m = n/2 - p$, $p \geq 0$, целое, $n \geq 5$ и $\sigma(n) \equiv n (\mathrm{mod}\ 2)$. Для действительного числа r обозначим через $[r]_0$ (соответственно $[r]_1$) максимальное целое (полуцелое, то есть число вида $q/2$ для нечетных q), не превосходящее r. Тогда максимальная мощность $N_{\max}(n)$ окрестности превосходит $\left[\frac{n+1}{2}\right]!$ и, согласно [25], равна:

$$N_{\max}(n) = (n/2 + p_0)!/2p_0!,$$

где $p_0 = \left[\sqrt{\frac{1}{8}n + \frac{9}{8}} + \frac{3}{8}\right]_{\sigma(n)}$.

Применение этой формулы позволяет получить асимптотическое выражение для мощности окрестности.

Теорема 4.12. $N_{\max} = \Theta\left(\frac{e^{\sqrt{n/2}}\left[\frac{n+1}{2}\right]!}{n^{1/4+[1/2]_{\sigma(n)}}}\right)$.

Перейдем теперь к наиболее интригующему свойству данной окрестности, связанному к расстоянием между турами в графе окрестностей. Оказывается, что с помощью множества $N(y_1, \dots, y_k, x_1, \dots, x_m)$ можно так определить новую окрестность, что расстояние между любыми двумя турами не будет превосходить 4 [26]. Сам факт, что максимальное расстояние между турами не зависит от размерности задачи, представляется удивительным. Более того, эта константа очень мала; всего 4 шага достаточно сделать, чтобы добраться из заданного решения до любого другого и, в частности, до оптимального. К сожалению, мы не знаем, в какую именно сторону нужно сделать эти четыре шага, иначе задача стала бы полиномиально разрешимой. Кроме того, мы не знаем, как много локальных оптимумов типа 2-замена или city-swap содержится в такой экспоненциальной окрестности. Тем не менее, факт ограниченности диаметра свидетельствует о больших потенциальных возможностях такой окрестности.

Пусть T – произвольный гамильтонов цикл. Обозначим через $\mathcal{F}_{nm}$ семейство всех подмножеств мощности m, состоящих из несмежных вершин цикла T. Каждый элемент семейства $(x_1, \dots, x_m) \in \mathcal{F}_{nm}$ однозначно определяет множество циклов $N(y_1, \dots, y_{n-m}, x_1, \dots, x_m)$. Объединение таких множеств обозначим через $N_m(T)$. Можно показать, что при фиксированном m мощность семейства $\mathcal{F}_{nm}$ равна

$$|\mathcal{F}_{nm}| = \binom{n-m}{m} + \binom{n-m-1}{m-1},$$

то есть выражается полиномом от n, и, следовательно, оптимальный цикл в множестве $N_m(T)$ можно найти за полиномиальное время. Будем говорить, что цикл R является соседним к циклу T, если в цикле T найдутся m несмежных вершин $(x_1, \dots, x_m)$, таких, что $R \in N(y_1, \dots, y_{n-m}, x_1, \dots, x_m)$.

Теорема 4.13. При $m = \lfloor (n-1)/2 \rfloor$ для любых циклов T_1 и T_5 существуют такие циклы T_2, T_3 и T_4, что T_i является соседним к T_{i-1}, $i = 2, 3, 4, 5$.

В общем случае при произвольных m длина пути в графе окрестностей G_N между двумя произвольными циклами ограничена сверху величиной $4\lceil\lfloor(n-1)/2\rfloor/m\rceil$.

4.5. Метаэвристики

Идеи локального поиска получили свое дальнейшее развитие в так называемых *метаэвристиках*, то есть в общих схемах построения алгоритмов, которые могут быть применены практически к любой задаче дискретной оптимизации. Все метаэвристики являются итерационными процедурами, и для многих из них установлена асимптотическая сходимость наилучшего найденного решения к глобальному оптимуму. В отличие от алгоритмов с оценками, метаэвристики не привязываются к специфике решаемой задачи. Это достаточно общие итерационные процедуры, использующие рандомизацию и элементы самообучения, интенсификацию и диверсификацию поиска, адаптивные механизмы управления, конструктивные эвристики и методы локального поиска. К метаэвристикам принято относить методы имитации отжига (*Simulated Annealing* (SA)), поиск с запретами (*Tabu Search* (TS)), генетические алгоритмы (*Genetic Algorithms* (GA)) и эволюционные методы (*Evolutionary Computation* (EC)), а также поиск с чередующимися окрестностями (*Variable Neighborhood Search* (VNS)), муравьиные колонии (*Ant Colony Optimization* (ACO)), вероятностные жадные алгоритмы (*Greedy Randomized Adaptive Search Procedure* (GRASP)) и др. [27].

Идея этих методов основана на предположении, что целевая функция имеет много локальных экстремумов, а просмотр всех допустимых решений невозможен, несмотря на конечность их числа. В такой ситуации нужно сосредоточить поиск в наиболее перспективных частях допустимой области. Таким образом, задача сводится к выявлению таких областей и быстрому их просмотру. Каждая из метаэвристик решает эту проблему по-своему.

Метаэвристики принято делить на траекторные методы, когда на каждой итерации имеется одно допустимое решение и осуществляется переход к следующему, и на методы, работающие с семейством (популяцией) решений. К первой группе относятся TS, SA, VNS. Траекторные методы оставляют в пространстве поиска траекторию,

последовательность решений, где каждое решение является соседним для предыдущего относительно некоторой окрестности. В методах TS, SA окрестность определяется заранее и не меняется в ходе работы. Целевая функция вдоль траектории меняется немонотонно, что позволяет «выбираться» из локальных экстремумов и находить все лучшие и лучшие приближенные решения. Элементы самоадаптации позволяют менять управляющие параметры алгоритмов, используя предысторию поиска. Более сложные методы, например, VNS, используют несколько окрестностей и меняют их систематически в целях диверсификации. Фактически, при смене окрестности происходит смена ландшафта. Сознательное изменение ландшафта, как, например, это происходит в методе шума (*Noising method*, [28]), благотворно сказывается на результатах поиска. Изучение ландшафтов, их свойств, например, изрезанности (ruggedness), позволяет давать рекомендации по выбору окрестностей.

Ко второй группе методов относятся GA, EC, ACO и др. На каждой итерации этих методов строится новое решение задачи, которое основывается уже не на одном, а на нескольких решениях из популяции. В генетических и эволюционных алгоритмах для этих целей используются процедуры скрещивания (crossover) и целенаправленных мутаций. В методах ACO применяется другая идея, основанная на сборе статистической информации о наиболее удачных найденных решениях. Эта информация учитывается в вероятностных жадных алгоритмах и подсказывает, какие именно компоненты решений (ребра графа, предприятия, элементы системы технических средств) чаще всего приводили ранее к малой погрешности. Методы этой группы, как правило, основаны на аналогиях в живой природе. Идея ACO является попыткой имитации поведения муравьев, которые почти не имеют зрения и ориентируются по запаху, оставленному предшественниками. Сильно пахнущее вещество, феромон, является для них индикатором деятельности предшественников. Оно аккумулирует в себе предысторию поиска и подсказывает дорогу к муравейнику. Попытки подглядеть у природы способы решения трудных комбинаторных задач находят все новые и новые воплощения в численных методах. Например, при инфекции организм старается подобрать (породить, сконструировать) наиболее эффективную защиту. Наблюдение за этим процессом привело к рождению нового метода – искусственным иммунным системам. Иссле-

дование жизнедеятельности пчелиного улья, где только одна матка оставляет потомство, послужило основой для новых генетических методов. Однако наибольший прогресс наблюдается на пути гибридизации, например, построения гиперэвристик, автоматически подбирающих наиболее эффективные эвристики для данного примера и симбиоз с классическими методами математического программирования.

Остановимся подробнее на последнем направлении. Здесь наблюдается достаточно широкий спектр исследований [20]:

- построение экспоненциальных по мощности окрестностей, в которых поиск наилучшего решения осуществляется за полиномиальное время путем решения вспомогательной оптимизационной задачи;
- исследование операторов скрещивания, базирующихся на точном или приближенном решении исходной задачи на подмножестве, заданном родительскими решениями;
- гибридизация метаэвристик с точными методами, например, с методами ветвей и границ и методом динамического программирования;
- разработка новых точных методов, использующих идеи локального поиска;
- применение разных математических формулировок решаемой задачи и использование различных кодировок решений и др.

Обзор достижений в данном направлении можно найти в [27]. Ниже будут представлены пять метаэвристик, четыре из которых являются траекторными алгоритмами. Каждая из них имеет свою идею и механизм ее реализации. Несмотря на то, что эта глава посвящена задаче коммивояжера, читатель легко увидит способы адаптации данных алгоритмов к другим задачам.

4.5.1. Алгоритм имитации отжига

Экзотическое название данного алгоритма связано с методами имитационного моделирования в статистической физике, основанными на технике Монте-Карло. Исследование кристаллической решетки и поведения атомов при медленном остывании тела привело

к появлению на свет стохастических алгоритмов, которые оказались чрезвычайно эффективными в комбинаторной оптимизации. Впервые это было замечено в 1983 г. [29]. В настоящее время этот подход является популярным как среди практиков, благодаря своей простоте, гибкости и эффективности, так и среди теоретиков, поскольку для него удается доказать асимптотическую сходимость к глобальному оптимуму.

Алгоритм имитации отжига относится к классу пороговых алгоритмов локального поиска. На каждом шаге в окрестности текущего решения выбирается новое решение. Если разность по целевой функции между ними не превосходит заданного порога t_k, то новое решение заменяет текущее. В противном случае выбирается другое соседнее решение. Общая схема пороговых алгоритмов может быть представлена следующим образом.

Пороговый алгоритм

1. Выбрать начальное решение s и положить $f^* := f(s), k := 0$.
2. Пока не выполнен критерий остановки, делать следующее:
   2.1. Выбрать $s' \in N(s)$ случайным образом.
   2.2. Если $f(s') - f(s) < t_k$, то $s := s'$.
   2.3. Если $f^* > f(s)$, то $f^* := f(s)$.
   2.4. Положить $k := k + 1$.
3. Предъявить наилучшее найденное решение.

В зависимости от способа задания последовательности $\{t_k\}$ различают три типа алгоритмов

1. *Последовательное улучшение*: $t_k = 0$ для всех k – вариант классического локального спуска с монотонным улучшением по целевой функции.

2. *Пороговое улучшение*: $t_k = c_k,\ k = 0,1,2, \ldots,\ c_k \geq c_{k+1} \geq 0$ $\lim_{k\to\infty} c_k \to 0$ – вариант локального поиска, когда допускается ухудшение по целевой функции до некоторого порога, и этот порог последовательно снижается до нуля.

3. *Имитация отжига*: t_k– неотрицательная случайная величина с математическим ожиданием $\mathbb{E}(t_k) = c_k$ – вариант локального поиска, когда допускается произвольное ухудшение по целевой функции, но

вероятность такого перехода обратно пропорциональна величине ухудшения, точнее для любого $s' \in N(s)$

$$P_{ss'} = \begin{cases} 1, & \text{если } f(s') \le f(s), \\ \exp\left(\dfrac{f(s) - f(s')}{c_k}\right), & \text{если } f(s') > f(s). \end{cases}$$

Последовательность $\{c_k\}$ играет важную роль при анализе сходимости и выбирается так, чтобы $c_k \to 0$ при $k \to \infty$. Иногда параметр c_k называют температурой, имея в виду указанные выше истоки. Для анализа данного и последующих алгоритмов потребуются некоторые определения из теории конечных цепей Маркова [19, 30].

Определение 4.6. Пусть $\mathfrak{O}$ обозначает множество возможных исходов некоторого случайного процесса. *Цепь Маркова* есть последовательность испытаний, когда вероятность исхода в каждом испытании зависит только от результата предшествующего испытания.

Пусть $x(k)$ – случайная переменная, обозначающая результат k-го испытания. Тогда для каждой пары $i, j \in \mathfrak{O}$ вероятность перехода от i к j при k-ом испытании задается выражением

$$P_{ij}(k) = \mathbb{P}\{x(k) = j \mid x(k-1) = i\}.$$

Матрица (P_{ij}) называется *переходной матрицей*. Цепь Маркова называется *конечной*, если множество исходов конечно, и *однородной*, если переходные вероятности не зависят от номера шага k.

Для алгоритма имитации отжига испытание состоит в выборе очередного решения на k-м шаге. Вероятность перехода зависит от текущего решения s_k, так как переход возможен только в соседнее решение. Она не зависит от того, каким путем добрались до этого решения. Таким образом, последовательность решений $\{s_k\}$ образует цепь Маркова. Эта цепь конечна, т. к. множество допустимых решений конечно. Для каждой пары $s, s' \in Sol$ вероятность перехода от s к s' задается выражением:

$$P_{ss'}(k) = \begin{cases} D_{ss'}A_{ss'}(c_k), & \text{если } s \ne s', \\ 1 - \sum\limits_{l \ne s} D_{sl}A_{sl}(c_k), & \text{если } s = s'. \end{cases}$$

где $D_{ss'}$ есть вероятность выбора решения s' из окрестности $N(s)$, $A_{ss'}(c_k)$ – вероятность принятия решения s'. Если в окрестности решения выбираются равновероятно, то

$$D_{ss'} = \begin{cases} \dfrac{1}{|N(s)|}, & \text{если } s' \in N(s), \\ 0 & \text{в противном случае,} \end{cases}$$

а вероятность принятия s' определяется как

$$A_{ss'}(c_k) = \exp \frac{\max\{0,\ f(s') - f(s)\}}{c_k}.$$

Если $c_k = c$, то цепь Маркова является однородной. В общем случае алгоритм имитации отжига порождает конечную неоднородную цепь Маркова. При исследовании асимптотического поведения алгоритмов важную роль играет так называемое стационарное распределение, вектор q_i размерности $|\Im|$, компоненты которого определяются как

$$q_i = \lim_{k\to\infty} \mathbb{P}\{x(k) = i \mid x(0) = j\}, i \in \Im,$$

если такой предел существует и не зависит от j. Величина q_i равна вероятности оказаться в состоянии i после достаточно большого числа шагов.

Теорема 4.14. Пусть $c_k = c$ для всех k и для любой пары $s, s' \in Sol$ найдутся положительное целое число p и такие решения $l_0, l_1, \dots, l_p \in Sol$, что $l_0 = s$, $l_p = s'$ и $D_{l_k l_{k+1}} > 0$ для $0 \le k \le p-1$. Тогда для цепи Маркова, порожденной алгоритмом имитации отжига, существует единственное стационарное распределение, компоненты которого задаются формулой

$$q_s(c) = \frac{\exp(-f(s)/c)}{\sum_{l\in Sol} \exp(-f(l)/c)}, \qquad s \in Sol,$$

и

$$\lim_{c\downarrow 0} q_s(c) = \begin{cases} 1/|S^*|, & \text{если } s \in S^*, \\ 0 & \text{в противном случае.} \end{cases}$$

Последнее равенство, по сути, означает, что с ростом числа итераций вероятность оказаться в точке глобального оптимума стремится к 1, если значение порога c_k стремится к нулю, то есть

$$\lim_{c \downarrow 0} \lim_{k \to \infty} \mathbb{P}_c\{s_k \in S^*\} = 1.$$

На практике, конечно, нет возможности выполнить бесконечное число итераций. Поэтому приведенная теорема является только источником вдохновения при разработке алгоритмов. Существует много эвристических способов выбора конечной последовательности $\{c_k\}$ с целью повышения вероятности обнаружения глобального оптимума.

В большинстве работ следуют рекомендациям первопроходцев и используют схему геометрической прогрессии:

1. Начальное значение: $c_0 = \Delta f_{\max}$– максимальная разность между соседними решениями.

2. Понижение порога: $c_{k+1} = \alpha c_k,\ k = 0, 1, \dots, K-1$, где α – положительная константа, достаточно близкая к 1.

3. Финальное значение: $c_K > 0$ определяется либо по числу сделанных изменений, либо как максимальное c_k, при котором алгоритм не меняет текущее решение в течение заданного числа шагов. При каждом значении c_k алгоритм выполняет порядка $|N|$ шагов, не меняя значение порога. Общая схема имитации отжига может быть представлена следующим образом.

Алгоритм SA

1. Выбрать начальное решение s и положить $f^* \coloneqq f(s)$, $s^* \coloneqq s$, определить начальную температуру t и коэффициент α.

2. Повторять, пока не выполнен критерий остановки.

2.1. Цикл $k \coloneqq 1, \dots, |N|$.

Выбрать решение $s' \in N(s)$ случайным образом.

Вычислить $\Delta = f(s) - f(s')$.

Если $\Delta \geq 0$, то $s \coloneqq s'$, иначе $s \coloneqq s'$ с вероятностью $\exp(\Delta/t)$.

Если $f^* > f(s)$, то $f^* \coloneqq f(s)$, $s^* \coloneqq s$.

2.2. Понизить температуру: $t \coloneqq \alpha t$.

3. Предъявить наилучшее найденное решение s^*.

На рис. 4.20 показано типичное поведение алгоритма. На первых итерациях при высокой температуре наблюдается большой разброс

по целевой функции. Процесс поиска чем-то напоминает броуновское движение.

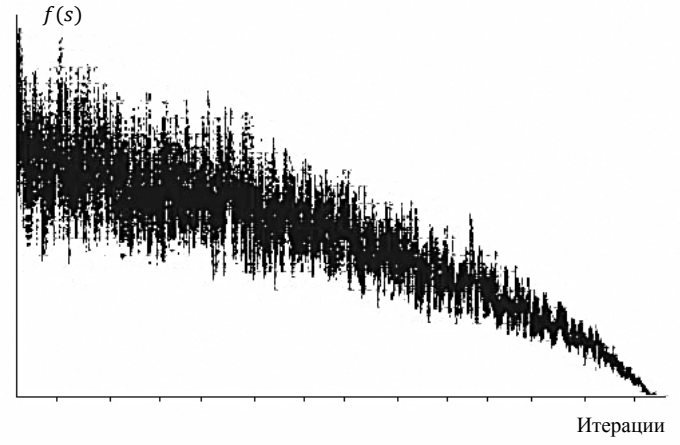


Рис. 4.20. Поведение алгоритма SA

По мере понижения температуры разброс уменьшается, и наблюдается явная тенденция к падению среднего значения целевой функции. При низкой температуре наблюдается стагнация процесса. Все чаще и чаще не удается перейти в соседнее решение, и алгоритм останавливается в одном из локальных оптимумов.

Среди пороговых алгоритмов следует выделить еще один с забавным названием *великий потоп* (Great Deluge). Правда, для задач на минимум его следует называть скорее *великая засуха*. На каждой итерации этого алгоритма имеется некоторое допустимое решение задачи s, и в его окрестности, как и раньше, выбирается решение s' случайным образом. Если $f(s') < f(s)$, то выполняется переход в новое решение. В противном случае значение $f(s')$ сравнивается с *уровнем воды* W. Если $f(s') \leq W$, то все равно выполняется переход в новое решение. Если же переход не выполнен, то выбирается новое соседнее решение. Порог W сначала устанавливается достаточно большим, чтобы были возможны любые переходы. Затем этот порог уменьшается на каждой итерации на заданную величину ΔW. Процесс останавливается в локальном минимуме со значением выше уровня воды. Общая схема алгоритма не сильно отличается от имитации отжига, но приводит к качественно другой картине.

Алгоритм GD

1. Выбрать начальное решение s и положить $f^* := f(s)$, $s^* := s$, определить начальный порог W и скорость его падения ΔW.

2. Повторять, пока не выполнен критерий остановки.

2.1. Выбрать решение $s' \in N(s)$ случайным образом.

2.2. Вычислить $\Delta = f(s) - f(s')$.

2.3. Если $\Delta \geq 0$, то $s := s'$, иначе если $f(s') \leq W$, то $s := s'$.

2.4. Если $f^* > f(s)$, то $f^* := f(s)$, $s^* := s$.

2.5. Понизить порог: $W := W - \Delta W$.

3. Предъявить наилучшее найденное решение s^*.

На рис. 4.21 показано типичное поведение алгоритма. Падение порога на каждой итерации вынуждает искать направления спуска. Как рыба, которая не хочет оказаться на суше, мы ищем все более и более глубокие места. Заметим, что для реализации алгоритма достаточно уметь вычислять значение целевой функции и определять окрестность. Параметры W и ΔW легко настраиваются под специфику задачи, что открывает широкий простор для приложений.

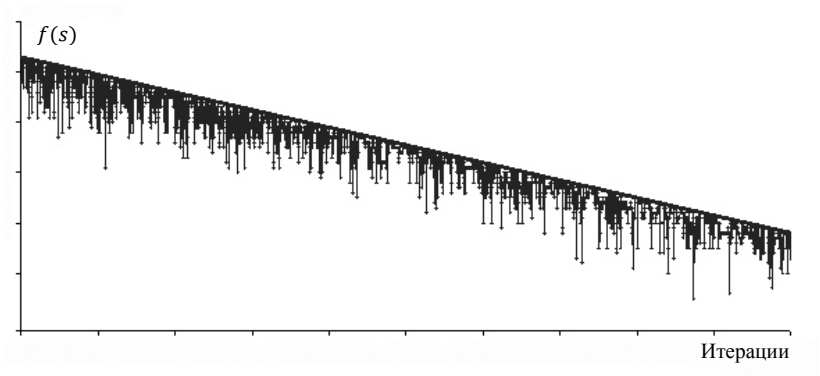


Рис. 4.21. Поведение алгоритма GD

4.5.2. Поиск с запретами

Основоположником алгоритма поиска с запретами является Ф. Гловер, который в 1986 г. предложил принципиально новую схему локального поиска [31]. Она позволяет алгоритму не останавливаться в точке локального оптимума, как это предписано в стандартном алгоритме локального улучшения, а путешествовать от одного оптимума к другому, в надежде найти среди них глобальный оптимум. Основным механизмом, позволяющим алгоритму выбираться из локального оптимума, является список запретов. Он строится по предыстории поиска, то есть по нескольким последним решениям, $s_k, s_{k-1}, \dots, s_{k-l+1}$, и запрещает часть окрестности текущего решения $N(s_k)$. Точнее, на каждом шаге алгоритма новое решение s_{k+1} является оптимальным решением подзадачи

$$f(s_{k+1}) = \min\{f(s) \mid s \in N(s_k) \setminus Tabu_l(s_k)\}.$$

Список запретов учитывает специфику задачи и, как правило, запрещает использование тех «фрагментов» решения (ребер графа, координат вектора, цвета вершин), которые менялись на последних l шагах алгоритма. Константа $l \geq 0$ определяет его память. При «короткой памяти» ($l = 0$) получаем стандартный алгоритм локального улучшения.

Существует много вариантов реализации этой идеи. Приведем один из них, для которого удается установить асимптотические свойства. Рассмотрим рандомизированную окрестность $N_p(s)$. Каждый элемент окрестности $N(s)$ включается в множество $N_p(s)$ с вероятностью p независимо от других элементов. С ненулевой вероятностью множество $N_p(s)$ может совпадать с $N(s)$, может оказаться пустым или содержать ровно один элемент. Общая схема алгоритма поиска с запретами может быть представлена следующим образом.

Алгоритм TS

1. Выбрать начальное решение s и положить $f^* := f(s)$, $s^* := s$, $Tabu_l(s) := \emptyset$.

2. Повторять, пока не выполнен критерий остановки.

   2.1. Сформировать окрестность $N_p(s)$.

   2.2. Если $N_p(s) \setminus Tabu_l(s) \neq \emptyset$, то найти такое решение s', что

$$f(s') = \min\{f(j) \mid j \in N_p(s) \setminus Tabu_l(s)\}.$$

2.3. Если $f^* > f(s')$, то $f^* := f(s')$ и $s^* := s'$.

2.4. Положить $s := s'$ и обновить список запретов.

3. Предъявить наилучшее найденное решение s^*.

Параметры p и l являются управляющими для данного алгоритма. Выбор их значений зависит от размерности задачи и мощности окрестности. Примеры адаптации этой схемы к разным задачам комбинаторной оптимизации можно найти, например, в [20].

Пусть $l = 0$. Тогда алгоритм TS порождает конечную однородную цепь Маркова. Можно показать, что в этом случае переходные вероятности $P_{ij},\ i,j \in \Im,$ определяются равенством

$$P_{ij} = \begin{cases} 0, & \text{если } j \notin N(i), \\ p(1-p)^{k-1}, & \text{если } j \in N(i) \text{ и } f(j) = \min\limits_{\tau \in N(i)}{}^{k}(f(\tau)), \end{cases}$$

где $\min^k$ означает k-й минимальный элемент множества. Если граф окрестностей G_N строго связен, то для такой цепи Маркова существует единственное стационарное распределение, и вероятность не найти глобальный оптимум стремится к нулю со скоростью геометрической прогрессии. Заметим, что для стационарного распределения не получено точной формулы, как это удалось сделать для алгоритма имитации отжига. Утверждается только существование и единственность такого распределения. При малых размерностях его можно получить численно из соотношений

$$q_i = \sum_{j \in \Im} q_j P_{ij},\ i \in \Im, \qquad \sum_{j \in \Im} q_j = 1.$$

Аналитического выражения для решения такой системы не известно. Тем не менее, исследование этого распределения и, в частности, его компонент, соответствующих глобальному оптимуму, представляет несомненный интерес.

Пусть $l > 0$. В этом случае последовательность решений уже не является цепью Маркова, т. к. выбор следующего решения зависит от предыстории. Однако, если рассмотреть множество кортежей $\langle s_k, s_{k-1}, \ldots, s_{k-l+1} \rangle$ длины l, то такая последовательность, порождаемая алгоритмом, уже будет конечной однородной цепью Маркова. Матрица переходных вероятностей имеет теперь значительно большую размерность, но ее элементы будут определяться по сути теми же формулами, что и раньше.

Список запретов оказывает существенное влияние на поведение алгоритма. В частности, если на некотором шаге все соседние решения оказываются под запретом, то процесс поиска прекращается. Таким образом, даже свойство строгой связности графа G_N не гарантирует желаемого асимптотического поведения. Тем не менее, при определенных условиях на длину этого списка удается обеспечить сходимость по вероятности наилучшего найденного решения к глобальному оптимуму.

На рис. 4.22 показано типичное поведение алгоритма TS. На первых итерациях он ведет себя как стандартный алгоритм локального улучшения. Получив первый локальный минимум, алгоритм вынужден выполнить несколько шагов с ухудшением, после чего снова открывается путь к новым локальным оптимумам. Рандомизация окрестности привносит определенное разнообразие в процесс поиска и предотвращает зацикливание. Более того, она позволяет сократить трудоемкость одного шага алгоритма и часто повышает вероятность нахождения глобального оптимума за предписанное число итераций. Чем сложнее и запутаннее пример, тем меньшую долю окрестности следует просматривать. Как правило, значения параметра p в интервале от 0.2 до 0.3 приводят к хорошим результатам даже при небольшом числе итераций.

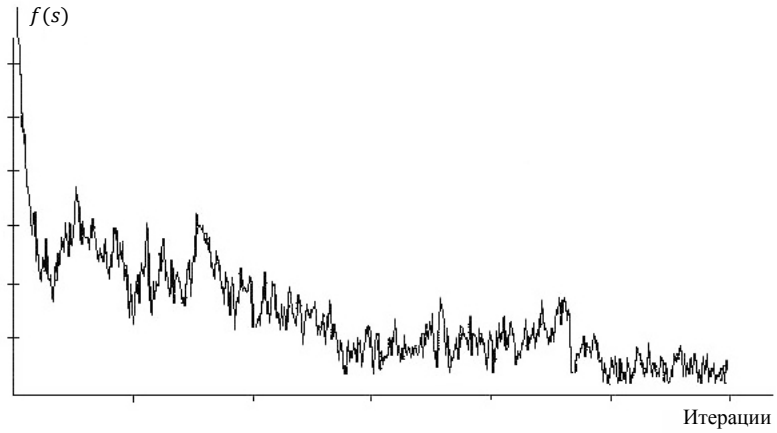


Рис. 4.22. Поведение алгоритма TS

4.5.3. Локальный поиск с чередующимися окрестностями

Идея этого подхода предложена Н. Младеновичем и П. Хансеном в 1997 г. [32]. Она состоит в систематическом изменении функции окрестности и соответствующем изменении ландшафта в ходе локального поиска. Существует много вариантов ее реализации, особенно для задач большой размерности. Легкость адаптации к различным моделям и высокая эффективность, особенно в сочетании с другими метаэвристиками, объясняет ее конкурентоспособность при решении NP-трудных задач.

Обозначим через $N_k, k = 1, \dots, k_{\max}$, конечное множество функций окрестностей, предварительно выбранных для локального поиска. Предлагаемый алгоритм опирается на следующие три тезиса.

– Локальный минимум для одной окрестности не обязательно является локальным минимумом для другой.

– Глобальный минимум является локальным минимумом относительно любой окрестности.

– Для многих задач дискретной оптимизации локальные минимумы относительно одной или нескольких окрестностей расположены достаточно близко друг к другу.

Последний тезис является эмпирическим. Он позволяет сузить область поиска глобального оптимума, используя информацию об уже обнаруженных локальных оптимумах. Именно эта идея лежит в основе алгоритмов связывающих путей, различных операторов скрещивания для генетических алгоритмов и многих других подходов. Алгоритм чередующихся окрестностей неявно использует этот тезис. Общая схема алгоритма может быть представлена следующим образом.

Алгоритм VNS

1. Выбрать окрестности $N_k,\ k = 1, \dots, k_{\max}$, и начальное решение s.
2. Повторять, пока не выполнен критерий остановки.
   2.1. Положить $k := 1$.
   2.2. Повторять, пока $k \le k_{\max}$:
      (a) Выбрать $s' \in N_k(s)$ случайным образом.

(b) Применить локальный поиск с начального решения s'. Полученный локальный оптимум обозначим s''.

(c) Если $f(s'') < f(s)$, то $s := s''$, иначе $k := k + 1$.

3. Предъявить наилучшее найденное решение s.

В качестве критерия остановки может использоваться максимальное время счета или число итераций без смены лучшего найденного решения. Наряду с последовательным порядком смены окрестностей могут использоваться различные групповые стратегии. Случайный выбор на шаге 2.2 (a) применяется во избежание зацикливания. На шаге 2.2 (b) локальный поиск может использовать различные алгоритмы SA, TS и др.

Существует много вариантов этой схемы. Например, на шаге 2.2 (c) переход осуществляется только при уменьшении значения целевой функции. Это условие можно заменить по аналогии с алгоритмом имитации отжига и разрешать движение в точку с большим значением целевой функции с некоторой вероятностью, зависящей от этого ухудшения. Идею наискорейшего спуска можно осуществить путем следующих изменений. Вместо шага 2.2, на котором выполняется переход согласно k-й окрестности, следует выбрать наилучший переход по всем рассматриваемым окрестностям. Другой способ состоит в модификации шага 2.2 (a) и выборе не случайного решения, а наилучшего в k-й окрестности. Порядок смены окрестностей тоже может варьироваться.

На рис. 4.23 показано типичное поведение алгоритма VNS. Высокие пики на последних итерациях соответствуют переходам на шаге 2.2 (a), где выбирается случайным образом соседнее решение. Несмотря на заметное ухудшение по целевой функции, локальный поиск на шаге 2.2.(b) быстро возвращается к хорошим решениям. Тенденция к последовательному улучшению рекорда хорошо видна на этом рисунке.

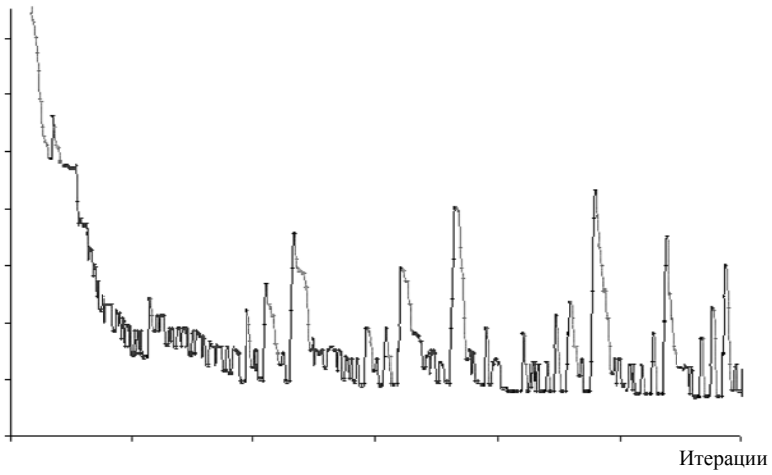


Рис. 4.23. Поведение алгоритма VNS

4.5.4. Генетические алгоритмы

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

Стандартный генетический алгоритм начинает свою работу с формирования начальной *популяции* – конечного набора допустимых решений задачи. Эти решения могут быть выбраны случайным образом или получены с помощью конструктивных алгоритмов. Выбор начальной популяции не имеет значения для сходимости

процесса в асимптотике, однако формирование «хорошей» начальной популяции (например, из множества локальных оптимумов) может заметно сократить время достижения глобального оптимума.

На каждом шаге эволюции с помощью вероятностного оператора *селекции* выбираются два решения, *родители* s_1, s_2. Оператор *скрещивания* по этим решениям строит новое решение s', которое затем подвергается небольшим случайным модификациям. Их принято называть мутациями. Затем решение добавляется в популяцию, а решение с наименьшим значением целевой функции удаляется из популяции. Общая схема такого алгоритма может быть записана следующим образом.

Алгоритм GA

1. Выбрать начальную популяцию.
2. Повторять, пока не выполнен критерий остановки.
   2.1. Выбрать родителей s_1, s_2 из популяции.
   2.2. Построить новое решение s' по решениям s_1, s_2.
   2.3. Модифицировать s'.
   2.4. Обновить популяцию.
3. Предъявить наилучшее решение в популяции.

Остановимся подробнее на основных операторах этого алгоритма: селекции, скрещивании и мутации. Среди операторов селекции наиболее распространенными являются два вероятностных оператора: *пропорциональной* и *турнирной* селекции. При пропорциональной селекции вероятность выбрать решение в качестве одного из родителей зависит от качества этого решения: чем меньше значение целевой функции, тем выше вероятность, а сумма вероятностей равна 1. При турнирной селекции формируется случайное подмножество из элементов популяции, и среди них выбирается один элемент с наименьшим значением целевой функции. Турнирная селекция имеет определенные преимущества. Она не теряет своей избирательности, когда в ходе эволюции все элементы популяции становятся примерно равными по значению целевой функции. Операторы селекции строятся таким образом, чтобы с ненулевой вероятностью любой элемент популяции мог бы быть выбран в качестве одного из родителей. Более того, допускается ситуация, когда оба родителя

представлены одним и тем же элементом популяции. Возвращаясь к алгоритму *VNS*, представляется разумным брать в качестве одного из родителей наилучший элемент популяции.

Как только два решения выбраны, к ним применяется вероятностный оператор скрещивания (*crossover*). Существует много различных версий этого оператора, среди которых простейшим, по-видимому, является однородный оператор. По родительским решениям он строит новое решение, присваивая каждой координате этого вектора с вероятностью 0,5 соответствующее значение одного из родителей. Если родительские вектора совпадали, скажем, по первой координате, то новый вектор «унаследует» это значение. Геометрически, для булевых векторов, оператор скрещивания случайным образом выбирает в гиперкубе вершину s', которая принадлежит минимальной грани, содержащей вершины s_1, s_2. Можно сказать, что оператор скрещивания старается выбрать новое решение где-то между родительскими, полагаясь на удачу. Более аккуратная процедура могла бы выглядеть таким образом. Новое решение выбирается как оптимальное решение исходной задачи на соответствующей грани гиперкуба. Если расстояние между s_1, s_2 достаточно велико, то задача оптимального скрещивания может совпадать с исходной. Тем не менее, даже приближенное решение этой задачи вместо случайного выбора заметно улучшает работу генетического алгоритма.

Оператор мутации с заданной вероятностью p_m меняет значение каждой координаты на противоположное. Например, вероятность того, что вектор (0,0,0,0,0) в ходе мутации перейдет в вектор (1,1,1,0,0) равна $p_m\, p_m\, p_m\, (1-p_m)\, (1-p_m) > 0$. Таким образом, с ненулевой вероятностью новое решение s' может перейти в любое другое решение, что и объясняет асимптотические свойства алгоритма. Отметим, что модификация решения s' может состоять не только в случайной мутации, но и в частичной перестройке решения алгоритмами локального поиска. Применение локального улучшения позволяет генетическому алгоритму сосредоточиться только на локальных оптимумах. Множество локальных оптимумов может оказаться экспоненциально большим, и на первый взгляд такой вариант алгоритма не будет иметь преимуществ. Однако экспериментальные исследования распределения локальных оптимумов свидетельствуют о высокой концентрации их в непосредственной близо-

сти от глобального оптимума. Это наблюдение отчасти объясняет работоспособность генетических алгоритмов. Если в популяции собираются локальные оптимумы, которые сконцентрированы в одном месте, и очередное решение s' выбирается где-то между двумя произвольными локальными оптимумами, то такой процесс имеет много шансов найти глобальный оптимум.

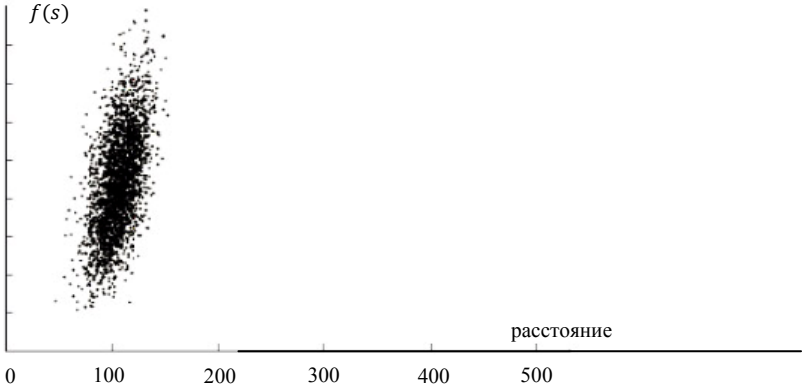


Рис. 4.24. Распределение локальных оптимумов

На рис. 4.24 показаны результаты численных экспериментов для задачи коммивояжера при $n = 532$. Каждая точка на рисунке соответствует локальному оптимуму по окрестности 2-замена. Ось y показывает значение целевой функции для локальных оптимумов. По оси x откладывается расстояние от локального оптимума до глобального. Максимально возможное расстояние равно 532. Однако среднее расстояние не превышает 200. Это означает, что все локальные оптимумы имеют более 300 общих ребер с глобальным оптимумом. Они все достаточно близки друг к другу [33].

В заключение этой главы хочется еще раз подчеркнуть тот факт, что разработка численных методов для решения NP-трудных задач дискретной оптимизации дает широкий простор для творчества. Учет специфики решаемой задачи может подсказать структуру окрестностей, выбор схемы и идею самого алгоритма, но единого наилучшего для всех задач метода построить не удастся, и теоремы

о *небесплатных завтраках* (No Free Lunch Theorems) – ясное тому подтверждение.

4.6. Упражнения

Упражнение 4.1. При построении примитивной нижней оценки для задачи коммивояжера можно сначала вычислять плату за въезд, в потом плату за выезд, но можно и наоборот (сначала плату за выезд, а потом плату за въезд). Покажите, что эти два варианта могут давать разные нижние оценки.

Упражнение 4.2. Покажите, что нижняя оценка линейного программирования из раздела 4.3.2 не хуже оптимального значения в целочисленной задаче о назначениях.

Упражнение 4.3. Приведите пример неотрицательной матрицы расстояний, для которой разница между оптимумом в задаче коммивояжера и оптимумом в задаче о назначениях была бы сколь угодно велика.

Упражнение 4.4. Усовершенствуйте полиномиальный алгоритм для задачи о назначениях, понизив его трудоемкость до $O(n^3)$.

Упражнение 4.5. Приведите пример неотрицательной матрицы расстояний, для которой разница между оптимумом в задаче коммивояжера и длиной минимального 1-дерева была бы сколь угодно велика.

Упражнение 4.6. Покажите, что минимальные 1-деревья дают нижнюю оценку для задачи коммивояжера, которая не больше оптимума задачи о назначениях.

Упражнение 4.7. Докажите, что оценка 3/2 для алгоритма Кристофидеса – Сердюкова является неулучшаемой.

Упражнение 4.8. Известно, что выбор наилучшего эйлерова цикла при построении приближенного алгоритма с оценкой 3/2 по Кристофидесу и Сердюкову является NP-трудной задачей. Постройте для ее решения генетический алгоритм, алгоритм поиска с запретами и алгоритм имитации отжига.

Упражнение 4.9. Для задачи коммивояжера разработайте гибридную схему генетического алгоритма и поиска с запретами, гене-

тического алгоритма и имитации отжига, локального поиска с чередующимися окрестностями и поиска с запретами.

Упражнение 4.10. Постройте генетический алгоритм и алгоритм поиска с чередующимися окрестностями для задачи Штейнера.

Упражнение 4.11. В открытой задаче коммивояжера требуется посетить все города, но не возвращаться в исходный город. Как решить эту задачу с помощью алгоритма для классической задачи коммивояжера?

Упражнение 4.12. Покажите, что открытая задача коммивояжера является NP-трудной.

Упражнение 4.13. Сформулируйте задачу обхода конем всех полей шахматной доски как открытую задачу коммивояжера.

Упражнение 4.14. В задаче коммивояжера на узкое место требуется минимизировать длину максимального ребра вместо суммы длин ребер. Запишите эту задачу в терминах целочисленного линейного программирования.

Упражнение 4.15. Покажите, что задача коммивояжера на узкое место является NP-трудной.

Упражнение 4.6. Предположим, что коммивояжер должен посетить город i сразу (не обязательно сразу) после города j. Запишите эту задачу в терминах целочисленного линейного программирования.

Упражнение 4.17. Предположим, что коммивояжер должен посещать город i не раньше момента времени a_i и не позже b_i, $1 \leq i \leq n$. Если он приезжает раньше срока, то вынужден ждать. Запишите эту задачу в терминах целочисленного линейного программирования.

Упражнение 4.18. Предположим, что коммивояжер посещает города с целью доставки грузов. В городе $i = 1$ находится склад. В каждый город j требуется доставить q_j единиц груза, $j > 1$. У коммивояжера имеется транспортное средство грузоподъемностью Q и $q_j \leq Q$ для всех j. Коммивояжер может возвращаться на склад, брать груз и снова направляться в непосещенные города. Его цель – минимизировать пройденное расстояние, посетить все города и вернуться на склад. Запишите эту задачу в терминах целочисленного линейного программирования.

Упражнение 4.19. В условиях предшествующей задачи будем предполагать, что коммивояжер может посещать любой город много раз и условие $q_j \leq Q$ может нарушаться. Запишите эту задачу в терминах частично-целочисленного линейного программирования.

Упражнение 4.20. Покажите, что задача коммивояжера с экспоненциальной окрестностью из раздела 4.4.4 принадлежит классу PLS.

Упражнение 4.21. Приведите пример задачи распознавания, которая не принадлежит классу NP (в предположении, что P ≠ NP).

Упражнение 4.22. Приведите пример задачи распознавания, которая не принадлежит классу co-NP (в предположении, что P ≠ NP).

Упражнение 4.23. Приведите пример задачи распознавания, которая не принадлежит классам NP и co-NP (в предположении, что P ≠ NP).

Упражнение 4.24. Покажите, что если оптимизационная задача является NP-трудной, то существование для нее точной полиномиально проверяемой окрестности влечет совпадение классов NP и co-NP.

Упражнение 4.25. Покажите, что если оптимизационная задача является NP-трудной в сильном смысле, то существование для нее точной полиномиально проверяемой окрестности влечет совпадение классов P и NP.

Библиографический список

1. Кристофидес Н. Теория графов. Алгоритмический подход. М: Мир, 1978.
2. Wichmann B. A. Ackermann's function: a study in the efficiency of calling procedures // BIT. 1976. Vol. 16. P.103–110.
3. Monma C., Suri S. Transitions in geometric minimum spanning trees // Discrete ans Computational Geometry. 1992. Vol. 8, N 3. P. 265–293.
4. Гэри М., Джонсон Д. Вычислительные машины и труднорешаемые задачи. М.: Мир, 1982.
5. Дементьев В. Т., Ерзин А. И., Ларин Р. М. и др. Задачи оптимизации иерархических структур. Новосибирск: Изд-во Новосибирского ун-та, 1996.
6. Robins G., Zelikovsky A. Improved Steiner tree approximation in graphs // Proc. 10th Ann. ACM-SIAM Symp. on Discrete Algorithms, ACM-SIAM, 2000. P. 770–779.
7. Hanan M. On Steiner's problem with rectilinear distance // SIAM J. Applied Math. 1966. Vol. 14. P. 255–265.
8. Hwang F. K. On Steiner minimal trees with rectilinear distance // SIAM J. Applied Math. 1976. Vol. 30, N 1. P. 104–114.
9. Zelikovsky A. Z. An 11/8-approximation algorithm for the Steiner problem on networks with rectilinear distance // Janos Bolyai Mathematica Societatis Conf.: Sets, Graphs, and Numbers, 1992. P. 733–745.
10. Erzin A. I., Cho J. D. A Deep-Submicron Steiner Tree // Mathematical and Computer Modeling. 2000. Vol. 31. P. 215–226.
11. Ерзин А. И., Чо Д. Д. Задача построения синхронизирующего сигнального дерева // Автоматика и телемеханика. Vol. 2003. № 3. С. 163–176.
12. Ерзин А. И. Введение в исследование операций: учеб. пособие. Новосибирск: НГУ, 2006.
13. Korte B, Vygen J. Combinatorial optimization. Berlin: Springer, 2007.

14. Кормен Т., Лейзерсон Ч., Ривест Р. и др. Алгоритмы. Построение и анализ. 2 издание. М.: Вильямс, 2009.
15. Deineko V., Tiskin A. Fast minimum-weight double-tree shortcutting for metric TSP: is the best one good enough? // ACM Journal on Experimental Algorithmics. 2009. Vol. 14. N 4.6.
16. Сигал И. Х., Иванова А. П. Введение в прикладное дискретное программирование: модели и вычислительные алгоритмы. М.: ФИЗМАТЛИТ, 2007.
17. Пападимитриу Х., Стайглиц К. Комбинаторная оптимизация. Алгоритмы и сложность. М.: Мир, 1985.
18. Алексеева Е. В. Построение математических моделей целочисленного линейного программирования. Примеры и задачи. Новосибирск: НГУ, 2012.
19. Yannakakis M. Computational complexity. In: Aarts E., Lenstra J. (Eds.) Local search in combinatorial optimization. NY: Wiley, 1997.
20. Кочетов Ю. А. Методы локального поиска для дискретных задач размещения Модели и алгоритмы. Saarbrucken: Lambert Academic Publishing, 2011.
21. Кочетов Ю. А. Вычислительные возможности локального поиска в комбинаторной оптимизации // Журнал вычислительной математики и математической физики. 2008. Т. 48, № 5. С. 788–807.
22. Grover L. K. Local search and the local structure of NP-complete problems // Operations Research Letters. 1992. Vol. 12, N 4. P. 235–244.
23. Angel E., Zissimopoulos V. On the classification of NP-complete problems in terms of their correlation coefficient // Discrete Appl. Math. 2000. Vol. 99. P. 261–277.
24. Burkard R. E., Deineko V. G., Woeginger G. J. The traveling salesman problem and the PQ-tree // Lecture Notes in Computer Science, Vol. 1084. Berlin: Springer, 1996. P. 490–504.
25. Gutin G. Exponential neighborhood local search for the traveling salesman problem // Comput. Oper. Res. 1999. Vol. 26. P. 313–320.

26. Gutin G., Yeo A. Small diameter neighborhood graphs for the traveling salesman problem: four moves from tour to tour // Comput. Oper. Res. 1999. Vol. 26. P. 321–327.
27. Talbi El-G. Metaheuristics: From Design to Implementation (Wiley Series on Parallel and Distributed Computing). Wiley, 2009.
28. Charon I., Hudry H. The noising methods. A survey. In: Ribeiro C. C., Hansen P. (eds.) Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization. Boston: Kluwer. Acad. Publ., 1998. P. 245–262.
29. Kirkpatrick S., Gelatt C. D., Vecchi M. P. Optimization by simulated annealing // Science. 1983. Vol. 220. P.671–680.
30. Кемени Дж., Снелл Дж. Конечные цепи Маркова. М.: Наука, 1970.
31. Glover F., Laguna M. Tabu Search. Dordrecht: Kluwer Acad. Publ., 1997.
32. Mladenović N., Hansen P. Variable neighbourhood search // Computers and Operations Research, 1997. Vol. 24. P. 1097–1100.
33. Boese K. D., Kahng A. B., Muddu S. A new adaptive multi-start technique for combinatorial global optimizations // Oper. Res. Lett. 1994. Vol. 16, N 2. P.101–114.
34. Wolsey L. A. Integer Programming. N. Y.: John Wiley & Sons, Inc, 1998.

Учебное издание

Ерзин Адиль Ильясович, Кочетов Юрий Андреевич

Задачи маршрутизации

Учебное пособие

Редактор *А. В. Грасмик*

Подписано в печать 09.06.2014.
Формат 60 × 84 1/16. Печать офсетная.
Уч.-изд. л. 6. Усл. печ. л. 5,6. Тираж 200 экз.

Заказ № 130

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