ЭКОНОМИКА
И МАТЕМАТИЧЕСКИЕ МЕТОДЫ

1965
том I, вып. 1

АЛГОРИТМ ДЛЯ РЕШЕНИЯ ЗАДАЧИ О КОММИВОЯЖЕРЕ

ДЖ. ЛИТЛ, К. МУРТИ, Д. СУИНИ, К. КЭРЕЛ

Мы помещаем ниже перевод работы, выполненной в Массачузетском технологическом институте (США) в 1963 г. четырьмя авторами * из различных организаций, работающих в содружестве с этим институтом (Индийский статистический институт, Международная компания конторских машин — IBM). Эта работа представляет очень большой интерес, так как авторам удалось значительно продвинуть вперед решение важной и трудной задачи о коммивояжере. Этой задаче эквивалентны многие производственные задачи, в частности календарные. Однако до последнего времени не были предложены эффективные методы, позволяющие точно решать задачи о коммивояжере большой размерности. В предлагаемой работе изложен найденный авторами алгоритм, позволяющий решать задачи с количеством переходов (городов), доходящим до 40 (что подтверждается проведенным авторами машинным экспериментом). Это значительно расширяет возможности практического использования данной модели для решения большого класса важных задач.

Метод решения задачи о коммивояжере, разработанный авторами, назван ими методом «ветвей и границ». Он заключается в том, что множество всех допустимых решений задачи, т. е. возможных циклов или маршрутов обхода заданных точек (городов), разбивается на последовательно уменьшающиеся подмножества при помощи процедуры, называющейся ветвлением. Для каждого подмножества особым методом, основанным на обработке (приведении) матрицы издержек по обходу заданных точек, вычисляется минимальная возможная длина цикла или его нижняя граница. Последовательно применяя эту процедуру, приходят к подмножеству, представляющему цикл обхода, длина которого не больше, чем у любого другого возможного цикла, т. е. к оптимальному циклу. Ветвление и вычисление нижних границ длины цикла в подмножестве основаны на идеях, часто используемых при решении задачи о назначениях. Алгоритм дает возможность получения численного решения для задачи без каких-либо специальных предположений о структуре решаемых задач.

* * *

Задачу о коммивояжере легко сформулировать: торговец, начиная с некоторого города, хочет посетить каждый из $(n-1)$ других городов один и только один раз и вернуться в начальный пункт. В каком порядке должен он посещать города, чтобы минимизировать суммарное пройденное расстояние? Под «расстоянием» мы можем подразумевать время, издержки или другой измеритель по нашему желанию. Расстояние (или издержки) между любыми двумя городами считаются известными.

Задача привлекла к себе внимание, так как в ней сочетается простота постановки и трудность решения. Трудности имеют чисто вычислительный характер, ибо существование решения очевидно. Имеется $(n-1)!$ возможных циклов, один или несколько из которых дают минимум издержек (минимальные издержки могут быть и бесконечными — принято считать бесконечными издержки по переезду между городами, между которыми нет прямого сообщения). Задача о коммивояжере приобрела

* John D. C. Little, Katta G. Murty, Dura W. Sweeney, and Caroline Karel. An Algorithm for the Traveling Salesman Problem. Opns. Res., 1963, vol. 11, No. 6, p. 972—990.

известность в США после того, как одна мыловаренная фирма объявила конкурс, предложив призы до 10 000 долл. за отыскание наилучшего пути для задачи с 33 городами. Очень немногие нашли наилучший цикл.

История вопроса освещена в работе М. Флада [1]. В последние годы было предложено много методов решения задачи. Одни из них неэффективны, другие дают не обязательно оптимальное решение, и, наконец, некоторые требуют принятия интуитивных решений, а это затрудняет программирование для машины. Более подробно обсуждение состояния вопроса дано в работе Гонзалеса [2]. Мы ограничимся рассмотрением методов, которые: 1) гарантируют оптимальность, 2) позволяют удобно программировать решение; 3) являются универсальными, т. е. не приспособлены с самого начала для задач специальной структуры.

Среди методов такого типа наиболее эффективными (в вычислительном отношении) оказались основанные на идеях динамического программирования. Гонзалес [2] и Хелд и Карп [3], независимо друг от друга, применив такой подход, решили на машине различные экспериментальные задачи. Гонзалес [2] составил программу для IBM-1620, предназначенную для решения задач размером до десяти городов. У него время решения задачи с ростом числа городов росло несколько быстрее, чем экспоненциально. Задача с пятью городами заняла 10 сек., задача с десятью городами — 8 мин., добавление одного города увеличивает время решения на некоторый множитель, который для десяти городов оказался больше 3. С аналогичной скоростью растут требования к объему памяти. Хелд и Карп [3] решали (на машине IBM-7090) задачи размером до 13 городов. Задача с 13 городами потребовала у них 17 сек. Но быстрота экспоненциального роста такова, что если бы объем вычислений рос с той же быстротой, что и у Гонзалеса, то задача для 20 городов потребовала бы 10 час. Требования к объему памяти могут стать чрезмерными еще раньше. Для задач более чем с 13 городами Хелд и Карп разработали метод, по-видимому, неплохой, но не гарантирующий оптимальности решения. Нам известны два случая, когда задача решалась методами, похожими на наш метод «ветвей и границ». Россман, Тьюери и Стоун в неопубликованной заметке применяют идеи, которые они называют комбинаторным программированием. Для иллюстрации своего метода они берут задачу с 13 городами. Она была решена их методом за 8 чел.-дн. Мы вручную решили их задачу примерно за 3,5 часа. Истмен в неопубликованных тезисах докторской диссертации и в лабораторном отчете предлагает метод решения, в котором есть значительное сходство с нашим методом. Однако его способ выбора ветвей и вычисления границ отличен от нашего. По существу он решает последовательность задач о назначении, получая при этом границы. Наш метод проще, и мы используем совершенно другой принцип ветвления. Задача наибольших размеров, решенная Истменом,— это задача с 10 городами, причем он не приводит данных о времени, затраченном на решение, так что трудно провести эффективное сравнение.

Большинство опубликованных задач симметрично, т. е. расстояние от города i до города j — то же, что от j до i. Предлагаемый алгоритм годится и для асимметричных задач (по-видимому, для таких задач алгоритм работает даже лучше). Асимметричные задачи появляются в разных приложениях. Приведем пример из области календарного планирования производства. Предположим, что в течение некоторого периода времени сборочная линия должна собирать изделия n различных типов. Стоимость перехода от изделия типа i к типу j равна c_{ij}. Какая последовательность типов собираемых изделий минимизирует суммарные издержки? Это задача о коммивояжере, в которой нет надобности предполагать, что $c_{ij} = c_{ji}$.

Итак, 13 городов — наибольший известный нам размер задачи, решенной универсальным методом, гарантирующим оптимальность и пригодным для машинного решения. Наш метод заметно увеличивает это число. Однако с увеличением числа городов необходимое время растет по крайней мере экспоненциально, что, конечно, в конце концов становится ограничивающим фактором. Подробное изложение метода дано ниже.

АЛГОРИТМ

Схема метода решения такова: множество всех возможных циклов разбивается на все меньшие и меньшие подмножества, для каждого из которых вычисляют минимальные возможные издержки (длину) цикла или их нижние границы. В соответствии с вычисленными минимальными значениями границ цикла проводят последовательно разбиение подмножеств и в конце концов определяют оптимальный цикл. Когда найдено подмножество, содержащее единственный цикл, издержки у которого меньше или равны нижним границам для всех остальных подмножеств, то такой цикл — оптимальный.

Подмножества циклов удобно представлять как вершины дерева и процесс разбиения — как ветвление (появление новых ветвей) дерева. Поэтому метод назван методом «ветвей и границ».

Изложение алгоритма будет сопровождаться разбором числового примера. При этом, однако, не требуется ссылок на числовой пример, так что читатели, при желании, могут и пропустить его.

О б о з н а ч е н и я. Издержки коммивояжера образуют матрицу. Пусть города обозначены через $i = 1, \ldots, n$. Элемент матрицы, находящийся на пересечении строки i и столбца j — это издержки для поездки из города i в город j. Пусть $C = [c(i, j)]$ — матрица издержек. Сначала матрица C — это первоначальная матрица издержек, затем она будет преобразовываться по ходу действия алгоритма.

Цикл t может быть представлен как набор из n упорядоченных пар городов, образующий контур, проходящий через каждый город один и только один раз, например $t = [(i_1, i_2)(i_2, i_3) \ldots (i_{n-1}, i_n)(i_n, i_1)]$. Каждое (i, j) изображает дугу маршрута. Затраты для цикла t, соответствующие матрице C,— это сумма элементов матрицы, соответствующих циклу t. Эти затраты обозначаются через $z(t)$.

$$z(t) = \sum_{(i,j) \in t} c(i, j).$$

Заметим, что в цикле t всегда содержится один и только один элемент в каждой строке и в каждом столбце. Итак, пусть X, Y, $\overline{Y}$ — вершины дерева; $w(X)$ — нижняя граница издержек для циклов, принадлежащих X, т. е. $z(t) \geqslant w(X)$ для циклов t, принадлежащих X; Z_0 — издержки наилучшего цикла, найденного на данном этапе действия алгоритма.

Н и ж н и е г р а н и ц ы. При построении нижних границ окажется полезным понятие приведения. Если константа h вычитается из каждого элемента некоторой строки матрицы издержек, то издержки для любого цикла при новой матрице на h меньше, чем при старой. Это происходит потому, что каждый цикл должен содержать один и только один элемент из этой строки. Однако относительные издержки всех циклов неизменны, и поэтому любой цикл, оптимальный для старой матрицы, будет оптимальным и для новой.

Процесс вычитания наименьшего элемента строки из всех ее элементов будем называть п р и в е д е н и е м строки. Матрицу с неотрицательными элементами и по крайней мере одним нулем в каждой строке и каждом столбце будем называть п р и в е д е н н о й м а т р и ц е й.

Приведенную матрицу можно получить, например, путем последовательного приведения по строкам и столбцам. Если $z(t)$ — издержки цикла t для матрицы до приведения, $z_1(t)$ — издержки для матрицы после приведения и h — сумма констант, использованных при приведении, то

$$z(t) = h + z_1(t). \qquad (1)$$

Так как приведенная матрица содержит только неотрицательные элементы, то h является нижней границей издержек цикла t при старой матрице.

От \ До	1	2	3	4	5	6
1	∞	27	43	16	30	26
2	7	∞	16	1	30	25
3	20	13	∞	35	5	0
4	21	16	25	∞	18	18
5	12	46	27	48	∞	5
6	23	5	5	9	5	∞

Рис. 1

	1	2	3	4	5	6
1	∞	11	27	0 (10)	14	10
2	1	∞	15	0 (1)	29	24
3	15	13	∞	35	5	0 (5)
4	0 (1)	0 (0)	9	∞	2	2
5	2	41	22	43	∞	0 (2)
6	13	0 (0)	0 (9)	4	0 (2)	∞

Рис. 2

Рис. 1. Матрица затрат для задачи с шестью городами. Типичным циклом может быть $t = [(1,3)\,(3,2)\,(2,5)\,(5,6)\,(6,4)\,(4,1)]$, что дает издержки (длину) $z = 43 + 13 + 30 + 5 + 9 + 21 = 121$

Рис. 2. Матрица затрат после приведения строк и столбцов. Числа в кружках — величины $\theta(i, j)$

Рассмотрим задачу с шестью городами, показанную на рис. 1. Приводя матрицу сначала по строкам, а затем по столбцам, получаем матрицу на рис. 2.

Сумма приводящих констант равна 48, так что для всех t: $z(t) \geqslant 48$.

Ветвление. Разбиение множества всех циклов на непересекающиеся подмножества будет изображаться путем разветвления дерева, как это показано на рис. 3. Вершина, содержащая «все циклы», не требует пояснений. Вершина, содержащая (i, j), изображает все циклы, которые включают пару городов (i, j). Вершина, содержащая $\overline{(i, j)}$, изображает все циклы, не содержащие (i, j). В вершине (i, j) имеется другое разветвление. Вершина, содержащая (k, l), изображает все циклы, включающие (i, j), но не включающие (k, l), в то время как (k, l) изображает все циклы, включающие как (i, j), так и (k, l). Вообще, если от вершины X пройти дерево по направлению к его началу, то мы сможем узнать, какие вершины обязательно входят в циклы, принадлежащие X, и какие вершины не могут входить в такие циклы. Если процесс ветвления зашел достаточно далеко, то в конце концов некоторые вершины будут изображать просто по одному циклу. Заметим, что на любом этапе процесса объединение множеств, изображаемых концевыми вершинами,— это множество всех циклов.

Когда в вершине X происходит разветвление на две последующие вершины, то вершину с только что введенной парой городов будем обозна-

чать через Y и вершину с только что запрещенной парой городов — через $\overline{Y}$.

Блок-схема алгоритма. Работу алгоритма можно понять, последовательно двигаясь по блок-схеме, представленной на рис. 4.

Блок 1 начинает вычисления, засылая в C первоначальную матрицу издержек, полагая $X = 1$, чтобы изобразить вершину «все циклы», и принимая на данном этапе издержки наилучшего цикла равными бесконечности.

Блок 2 приводит матрицу и помечает вершину X суммой приводящих констант, представляющей ее нижнюю границу $w(X)$.

Блок 3 отбирает пару городов (k, l), на которой основывается ближайшее разветвление с целью разбиения циклов, принадлежащих X, на подмножество (Y), которое с наибольшей вероятностью содержит наилучший цикл вершины, и другое подмножество $(\overline{Y})$, которое с наибольшей вероятностью не содержит его.

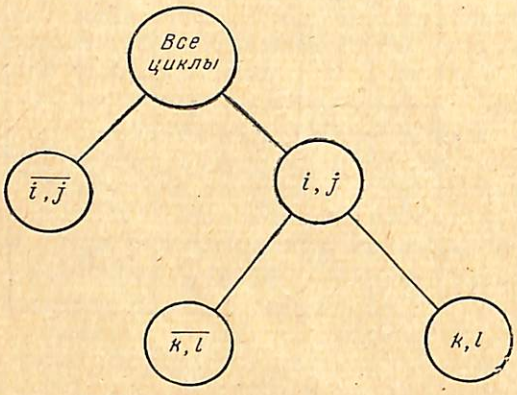


Рис. 3. Начало дерева

Низкие издержки при рассмотрении Y наиболее вероятны для циклов, содержащих (i, j), для которых $c(i, j) = 0$. Поэтому рассмотрим издержки для циклов, не содержащих (i, j), т. е. для $\overline{Y}$. Так как город i должен быть соединен с некоторым другим городом, то этим циклам должны соответствовать издержки, не превышающие наименьшего элемента строки i, не равного $c(i, j)$. Так как город j также должен соединяться с некоторым городом, то этим циклам должны соответствовать издержки, не превышающие наименьшего элемента столбца j, не равного $c(i, j)$. Обозначим сумму этих двух издержек через $\theta(i, j)$. Мы будем выбирать в качестве (k, l) такую пару городов, для которой $\theta(i, j)$ — наибольшее [это равносильно поиску среди (i, j) таких, что $c(i, j) = 0$, так как иначе $\theta(i, j) = 0$]. Заметим, что если $c(i, j) = \infty$, то сумма приводящих констант при приведении строки i и столбца j равна $\theta(i, j)$.

Например, на рис. 2 величины $\theta(i, j)$ выписаны (внутри кружков) в клетках, соответствующих нулям. Поскольку наибольшее θ — это $\theta(1, 4) = 10 + 0 = 10$, то $(1, 4)$ будет первой парой городов, используемой для ветвления.

Блок 4 расширяет дерево от вершины X к вершине $\overline{Y}$. Как будет показано ниже, $w(\overline{Y}) = w(X) + \theta(k, l)$. В примере $w(\overline{Y}) = 10 + 48 = 58$, и соответствующая пометка сделана у вершины на рис. 5, *b*.

Блок 5 вводит Y. Так как теперь пара (k, l) фиксирована для всех циклов, то строка k и столбец l больше не нужны и вычеркиваются из C. Далее, заметим, что (k, l) будет входить в некоторый связный путь между парами городов, фиксированными для циклов, принадлежащих Y. Предположим, что путь начинается в городе p и кончается в городе m (возможно, что $p = k$ или $m = l$, или и то и другое одновременно). Связь m с p должна быть запрещена, ибо при этом возникает подцикл (замкнутый контур, содержащий меньше, чем n городов), а ни один подцикл не может быть частью цикла. Поэтому полагаем $c(m, p) = \infty$. Возможно, что после этих преобразований матрица C может быть приведена по следующим ме-

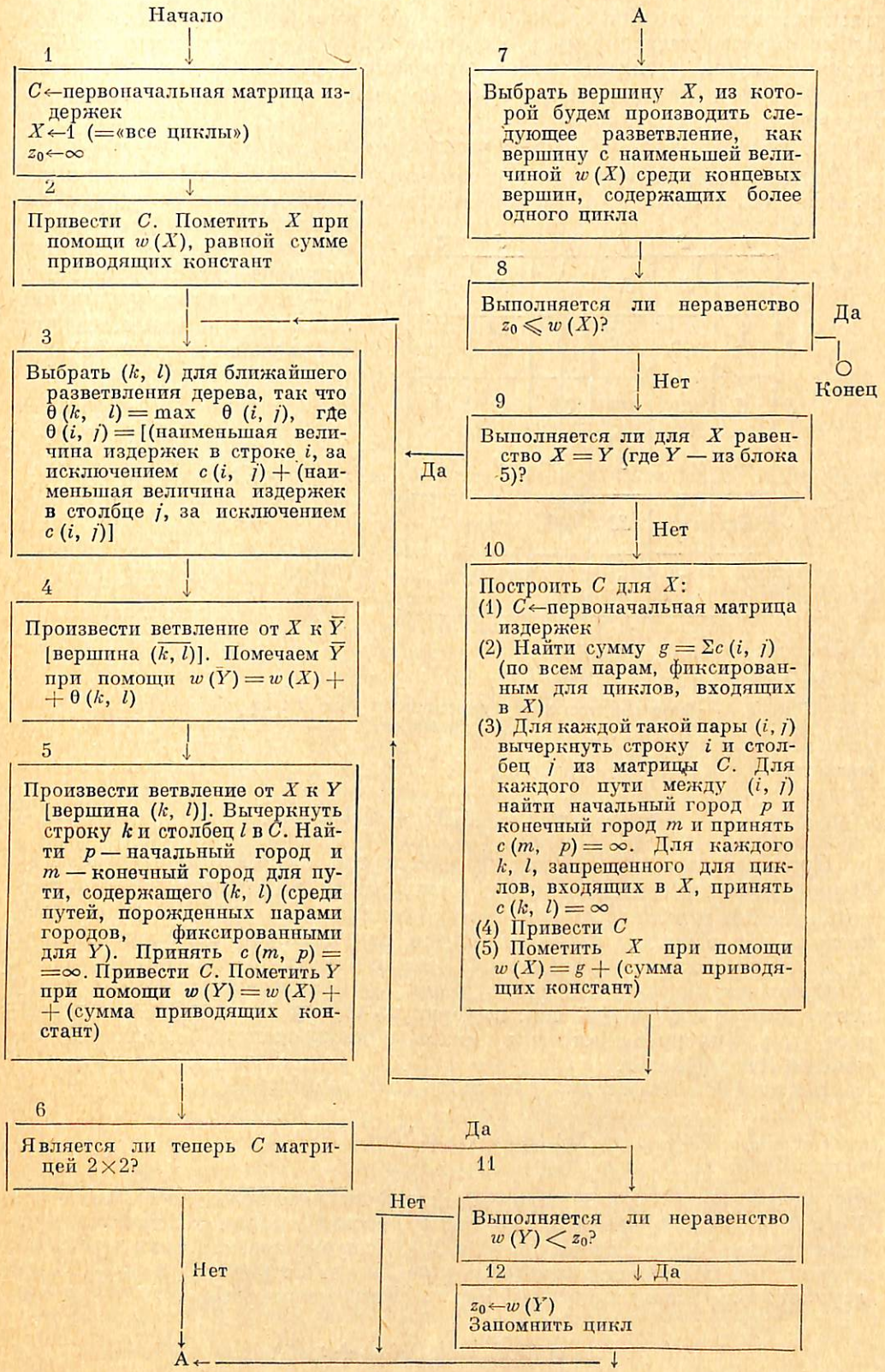


Рис. 4. Блок-схема алгоритма

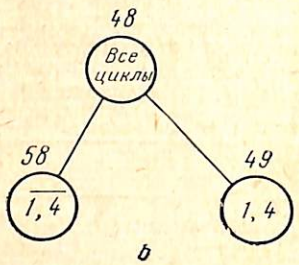
стам: строка m, столбец p, любые столбцы, имеющие нуль в строке k, и любые строки, имеющие нуль в столбце l. Все другие строки и столбцы содержат нули, препятствующие приведению. Пусть h — сумма новых приводящих констант. Теперь будет показано, что нижняя граница для Y равна

$$w(Y) = w(X) + h.$$

Алгоритм действует таким образом, что исследование любой вершины X начинается в блоке 3 с матрицей C и нижней границей $w(X)$. Если t — любой цикл из X, $z(t)$ — соответствующие издержки при первоначальной матрице, t_1 — те пары городов, принадлежащие t, которые остаются после удаления пар, фиксированных для циклов из X, и $z_1(t_1)$ — издержки для t_1 при

	1	2	3	4	5	6
1	~~∞~~	~~11~~	~~27~~	~~0~~	~~14~~	~~10~~
2	0 / ~~1~~	∞	14 / ~~15~~	0	28 / ~~29~~	23 / ~~24~~
3	15	13	∞	35	5	0
4	∞	0	9	∞	2	2
5	2	41	22	43	∞	0
6	13	0	0	4	0	∞

a



b

Рис. 5. *a* — матрица после вычеркивания строки *1* и столбца *4*; *b* — первое ветвление

матрице C, тогда будет показано, что

$$z(t) = w(X) + z_1(t_1). \tag{2}$$

Правильность этого выражения для первой вершины вытекает из (1). Предположим, что, в соответствии с алгоритмом, из границы $w(X_1)$ и матрицы C_1 для вершины X_1 получается граница $w(X_2)$ и приведенная матрица C_2 для вершины X_2 (X_2 находится на некоторой ветви, отходящей от X_1). Будет показано, что если (2) верно для X_1, то оно верно и для X_2.

Операции над C_1 с целью получения C_2 (показанные в блоках 5 и 10) всегда имеют следующий вид: вычеркнуть строку i и столбец j для каждого (i, j), фиксированного для циклов, принадлежащих X_2; вписать бесконечности; привести.

Нижняя граница всегда будет иметь следующий вид:

$$w(X_2) = w(X_1) + \sum c_1(i, j) + h, \tag{3}$$

где суммирование идет по парам городов, фиксированным для X_2, но не для X_1, и h — сумма приводящих констант. Рассмотрим, однако, любое t, принадлежащее X_2 (а следовательно, и X_1). Если мы примем, что $z_1(t_1)$ — издержки для незафиксированных пар из X_1 для матрицы C_1 и $z_2(t_2)$ — издержки для незафиксированных пар из X_2 для матрицы C_2, то

$$z_1(t_1) = \sum c_1(i, j) + h + z_2(t_2)$$

или, используя (2) (предполагая его верным для X_1),

$$z(t) = w(X_1) + \sum c_1(i, j) + h + z_2(t_2) = w(X_2) + z_2(t_2).$$

Тогда (2) верно для X_2, что и требовалось доказать.

Равенство (3) использовано в блоках 4, 5 и 10 для вычисления нижних границ. То, что эти границы верны, установлено при помощи (2) и неотрицательности элементов C.

Например, на матрице рис. 5 вычеркиваются строка *1* и столбец *4*. Связный путь, содержащий $(1, 4)$ — это сам путь $(1, 4)$, так что $(m, p) = (4, 1)$, и мы полагаем $c\ (4, 1) = \infty$. Что касается приведения, то мы находим, что строка *2* может быть приведена (с приводящей константой, равной 1). Тогда $w(Y) = 48 + 1 = 49$, как и показано на рис. 5.

Может быть, стоит дать другой пример отыскания (m, p). Предположим, что были зафиксированы следующие пары городов: $(2, 1)$, $(1, 4)$, $(4, 3)$ и $(5, 6)$, причем (k, l) равнялось $(1, 4)$. Тогда связный путь, содержащий (k, l), будет начинаться в *2* и кончаться в *3*, давая $(m, p) = (3, 2)$.

Блок 6 проверяет, достигли ли мы вершины, содержащей единственный цикл.

Блок 7 выбирает следующую вершину для ветвления. Существует много методов для этого выбора. Метод, употребляемый здесь,— это выбор вершины с наименьшей нижней границей. При этом получается дерево с наименьшим количеством вершин.

Блок 8 проверяет, закончена ли работа алгоритма, т. е. не будут ли издержки наилучшего из уже полученных циклов не меньшими, чем нижние границы для всех концевых вершин дерева.

Блок 9 позволяет сэкономить время. Бóльшая часть ветвлений делается к вершинам Y *, т. е. направо. При этом производится вычеркивание строк и столбцов и другие операции, которые можно производить над матрицей, оставшейся от предшествующих ветвлений. Когда имеет место такой случай, то это обнаруживается при помощи блока *9*, после чего алгоритм возвращает расчеты к блоку 3.

Блок 10 вступает в работу в том случае, если не срабатывает блок 9. Здесь получается нижняя граница и приведенная матрица для произвольного X. Берется первоначальная матрица издержек; вычеркиваются строки и столбцы, соответствующие парам городов, зафиксированным для X, ставятся бесконечности для предотвращения подциклов и в клетках, соответствующих запрещенным парам городов. Нижнюю границу можно получить при помощи (3), если понимать X_1 в (3) как начальную вершину с $w(X_1) = 0$ и с матрицей, равной первоначальной матрице издержек. Так как различные способы приведения матрицы могут давать разные суммы приводящих констант, то вместо прежнего значения $w(X)$ подставляется пересчитанное значение.

Блоки 11 и 12 завершают работу для вершин, содержащих единственный цикл. В некоторый момент матрица C есть матрица 2×2: имеются только два допустимых (i, j), и при их помощи заканчивается построение цикла. Так как работа блока начинается при наличии приведенной матрицы, то издержки, соответствующие элементам цикла (i, j), выбранным в последнюю очередь, равны нулю, и $z = w(Y)$ согласно (2). Если $z < z_0$, то новый цикл будет наилучшим из числа циклов, найденных к данному моменту, и его следует запомнить.

Вернемся к примеру. В блоке 7 в качестве второй вершины для ветвления берется $(1, 4)$ и, так как это ветвление направо, то мы уже располагаем матрицей C в приведенном виде. Как показано на рис. 6, следующее

* В оригинале: «...most branching is from Y nodes».— *Прим. пер.*

ветвление производится к $(2, 1)$, причем $(m, p) = (4, 2)$. Затем мы идем направо от $(2, 1)$ к $(5, 6)$, причем $(m, p) = (6, 5)$, а потом от $(5, 6)$ к $(3, 5)$, причем $(m, p) = (6, 3)$. В этой точке матрица C является матрицей 2×2, и мы переходим к блоку 11, чтобы закончить построение цикла. Мы находим $z = 63$ и запоминаем это значение z в качестве z_0.

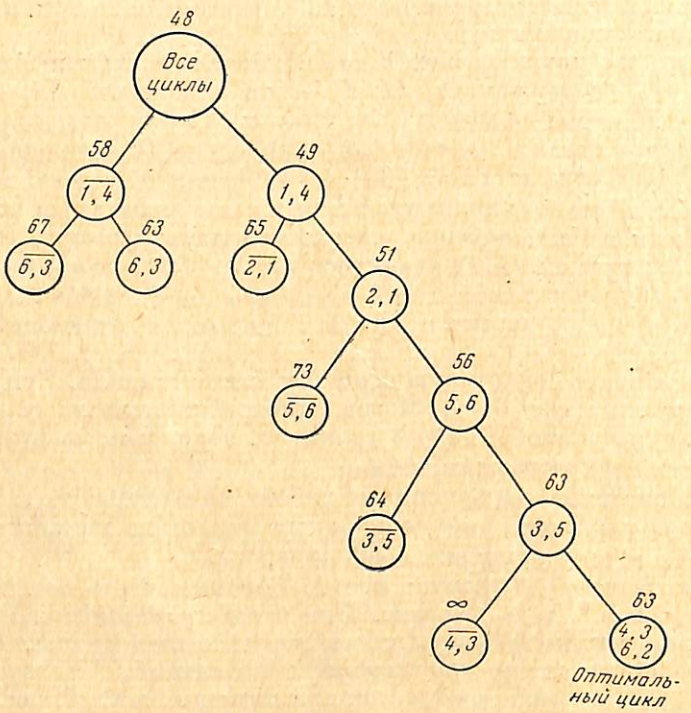


Рис. 6. Законченное дерево

Однако, возвращаясь к блокам 7 и 8, мы видим, что для $\overline{1, 4}$ нижняя граница равна 58. Чтобы получить эту вершину, мы обращаемся к блоку 10. Однако после ближайшего ветвления проверка в блоке 8 показывает, что задача решена.

О б с у ж д е н и е. Теперь мы вернемся назад и рассмотрим некоторые общие соображения в пользу данного алгоритма.

В процессе работы алгоритма производится ветвление, вычеркивание строки и столбца в матрице издержек, предотвращение подциклов и, наконец, приведение матрицы издержек с целью определения нижней границы издержек. Затем все повторяется. Хотя заранее очевидно, что оптимальное решение в конце концов будет получено, но откуда следует, что изложенный выше путь его отыскания окажется эффективным? Прежде всего, приведение — это эффективный способ для определения нижних границ издержек и тем самым — для подбора наиболее подходящих пар городов для включения в цикл. Далее, ветвление делается таким образом, чтобы максимизировать нижнюю границу для вершины $\overline{k, l}$, не слишком беспокоясь о вершине, которая изображает задачу меньшего размера, так как в ней вычеркнуты k-я строка и l-й столбец матрицы издержек. Благодаря тому, что делается упор на отыскание бо́льших нижних границ для задач большего размера, быстрее исключаются неоптимальные циклы.

Идею работы алгоритма можно понять, если учесть, что вычеркивание строк и столбцов и предотвращение образования подциклов приводит к новой задаче коммивояжера с меньшим числом городов.

Используя обозначения, введенные в блоке 5, мы можем представить себе города m и p объединенными в один город, скажем m'. Принять $c(m, p) = \infty$ — это то же самое, что принять $c(m', m') = \infty$. Предотвращение образования подциклов эквивалентно введению ограничения на циклы в задачу, которая в противном случае была бы задачей о назначениях; оно вполне успешно осуществляется при помощи данного алгоритма.

Заметим, что в отличие от большинства алгоритмов математического программирования данный алгоритм требует обширной памяти.

Не нужно на каждом этапе преобразовывать проверяемое решение в новое и лучшее решение, которое будет испытываться на следующем этапе. Испытываемая ветвь может быть отброшена на время, пока испытывается другая ветвь. Поэтому имеется значительный простор для экспериментов по выбору ближайшей ветви. С другой стороны, то же самое свойство приводит к ограничению возможности вычисления: для достаточно большого n приходится исследовать слишком много ветвей, и небольшой рост n приводит к необходимости исследования большого количества новых ветвей.

МОДИФИКАЦИИ

Может быть предложен ряд видоизменений основного метода. Мы отметим некоторые из них, в частности использованные в более поздних вычислениях Суини [5].

Правило «Иди вправо». В вычислительном отношении целесообразно придерживаться ветвления вправо до тех пор, пока это не станет явно неблагоразумным. В частности, ветвление всегда происходит от вершины (k, l), пока ее нижняя граница не станет больше (или равна) издержкам известного цикла. В результате может получиться, что будет испытано несколько лишних вершин, но обычно все же имеет место существенная экономия времени, связанная с меньшим использованием блока 10.

Одним из результатов данной модификации является то, что вычисление прямым путем доводится до искомого цикла. Если вычисления будут прекращены до того, как найдено оптимальное решение, во всяком случае будет найден хороший цикл и нижняя граница оптимального цикла. Нижняя граница может оказаться полезной при принятии решения — является ли цикл достаточно хорошим для некоторых практических целей.

Отбрасывание дерева. Большая задача может включать тысячи вершин; при этом емкость оперативной памяти может оказаться недостаточной. Память можно сэкономить, но обычно лишь за счет увеличения затрат времени, если учесть тот факт, что на любом этапе вычислений издержки наилучшего из полученных к этому моменту циклов дают верхнюю границу для издержек оптимального цикла. Пусть вычисления продолжаются путем ветвления вправо (с запоминанием каждой концевой вершины) и до тех пор, пока не будет найден единственный цикл, которому соответствуют издержки z_0. Дальше следовало бы найти концевую вершину с наименьшей нижней границей и продолжать ветвление от нее. Вместо этого мы проходим назад через все концевые вершины и удаляем их из памяти до тех пор, пока не дойдем до вершины с нижней границей, меньшей, чем z_0. Затем снова производим ветвление вправо, пока не дойдем до единственного цикла, или пока нижняя граница в некоторой (правой) вершине не достигает величины z_0. (Если ветвь доходит до конца, то можно найти лучший цикл, и для z_0 принимается новое, меньшее, значе-

ние.) Повторяем процедуру: снова доводим ветвь до конца, удаляем концевые вершины с границами, равными или бо́льшими, чем z_0, до тех пор, пока не дойдем до первой вершины с меньшей границей, снова производим ветвление вправо и т. д.

В результате применения этой процедуры приходится сохранять в памяти очень немного вершин — приблизительно порядка нескольких n. Они образуют правильную последовательность от обрабатываемой в данный момент вершины до концевой вершины на самой левой ветви (не совпадающей с вершиной «все циклы»).

В качестве иллюстрации рассмотрим задачу и дерево на рис. 6. По ходу вычислений будут запоминаться вершины $\overline{1, 4}$*; $\overline{2, 1}$; $\overline{5, 6}$; $\overline{3, 5}$. На следующем этапе мы находим цикл с $z_0 = 63$ и очевидно бесполезную вершину $\overline{4, 3}$. Цикл запоминается отдельно от дерева. Заканчивая построение ветви, отбрасываем $\overline{3, 5}$, затем $\overline{5, 6}$ и $\overline{2, 1}$; однако оказывается, что $\overline{1, 4}$ имеет границу меньшую, чем z_0. Следовательно, отсюда заново начинаем ветвление. Запоминаем вершину $\overline{6, 3}$ и затем находим вершину справа, у которой нижняя граница равна z_0 и которую можно отбросить. Снова возвращаясь по дереву назад, вычеркиваем $\overline{6, 3}$ и, так как это была единственная сохранившаяся концевая вершина, то на этом работа алгоритма заканчивается.

Эта процедура позволяет сэкономить память, но иногда увеличивает продолжительность вычислений. Если первый же шаг направо приводит к достаточно плохому циклу, т. е. к большому z_0, то критерий для отбрасывания вершин оказывается слишком жестким. Вычисление ускоряется путем ветвления из многих вершин, в которых нижние границы в действительности превосходят затраты при оптимальном цикле. В этом отличие данной модификации от первоначального метода, в котором такие вершины не исследуются до тех пор, пока не завершится исследование каждой вершины с меньшей границей. В процессе работы будет обнаружен оптимальный цикл, так что вершины с бо́льшими границами никогда не будут испытываться.

Использование преимуществ симметрии. Если задача коммивояжера симметрична и t — какой-либо цикл, то при прохождении того же маршрута в обратном направлении получается другой цикл с теми же издержками. Вероятно, большинство многообещающих способов использования этого обстоятельства связано с рассмотрением пары городов (i, j) как неупорядоченной. Это, естественно, приводит к новой и в известной степени более сильно ограничивающей процедуре. Хотя основная идея существенно не меняется, потребуется значительное изменение расчетной процедуры. Пока мы этим не занимались.

Имеется другой способ использовать преимущества симметрии, сохраняя в основе наш метод. Все обратные циклы могут быть запрещены путем видоизменения вершин вдоль самой левой ветви дерева. Это вершины, в которых нет фиксированных пар городов, но есть запрещенные пары. Допустим, что такая вершина X разветвляется к вершине Y с фиксированным (k, l) и $\overline{Y}$ — с запрещенным (k, l). Все обратные циклы, соответствующие Y, содержат в себе (l, k). Они не могут содержаться в Y, ибо наличие в одном и том же цикле (k, l) и (l, k) одновременно является невозможным. Те из обратных циклов, которые попадают в X, содержатся в $\overline{Y}$. Мы можем запретить их, полагая $c(l, k) = \infty$ [так же, как и $c(k, l) = \infty$] для любой матрицы, принадлежащей Y. Поэтому обратный цикл становится запрещенным, как только определен сам цикл,— в том смысле, что зафиксирована одна пара городов.

---

* В тексте *4, 1*,— по-видимому, опечатка.— *Прим. пер.*

Упрощение вычислений. Как при ручном, так и при машинном счете $\theta(k, l)$ легче всего вычисляется в первый раз. Для каждой строки k и столбца l приводимой матрицы имеем:

$\alpha(k)$ — следующий по величине после наименьшего размер издержек в строке k;

$\beta(l)$ — следующий по величине после наименьшего размер издержек в столбце l.

Тогда $\theta(k, l) = \alpha(k) + \beta(l)$ для любой пары (k, l) такой, что $c(k, l) = 0$. При ручном счете $\alpha(k)$ можно записать в дополнительный столбец справа от матрицы, и $\beta(l)$ — в дополнительную нижнюю строку. После решения нескольких задач становится ясно, что при ветвлении вправо не требуется найти целую матрицу, чтобы получить α и β, а нужно проверить лишь определенные строки и столбцы.

Другие возможности. При желании алгоритм можно видоизменить с тем, чтобы он давал все оптимальные решения. Вместо того, чтобы удалять вершины, для которых $w(X) = z_0$, производим в них дальнейшее ветвление, и так до тех пор, пока в конце концов не будет выполнено следующее условие: все концевые вершины либо имеют $w > z_0$, либо являются единственными (и при этом оптимальными) циклами с $z = z_0$. Наша программа вычислений не включает этого видоизменения, так как в некоторых случаях сильно возрастает время счета (предположим, например, что в матрице издержек одни нули).

Можно также уменьшить среднее время счета путем решения в блоке 2 задачи о назначениях для получения исходной матрицы издержек и приведения этой матрицы к оптимальному решению задачи о назначении. (Некоторые методы решения задачи о назначении дают матрицу в приведенной форме). Преимущество такого метода заключается в большей величине нижней границы издержек, с которой начинается решение задачи, а чем ближе эта граница к величине издержек для оптимального цикла, тем на меньшее количество ветвлений можно рассчитывать. Мы не провели более широкого исследования возможных преимуществ такого метода расчета; между тем результаты его применения неоднозначны: задача Грёса [6] с 20 городами была решена быстрее, но для некоторых других задач длительность расчетов увеличилась.

Идея метода, который мы назвали «методом ветвей и границ», может быть применена не только для разработки алгоритма решения задачи о коммивояжере. Она имеет более широкое значение. Решение многих задач на минимум, по-видимому, может быть получено путем разбиения множества всех допустимых решений на непересекающиеся подмножества, нахождения нижней границы (возможного минимального значения) целевой функции на каждом подмножестве, дальнейшего разбиения найденного подмножества с наименьшей нижней границей и т. д., до тех пор, пока не будет найдено оптимальное решение. Однако эффективность процесса в весьма значительной степени зависит от аппарата, используемого для разбиения на подмножества и нахождения нижних границ*. Простой

* Переводчик пытался непосредственно применить метод «ветвей и границ» для решения задачи оптимального размещения производства с линейными ограничениями и целевой функцией следующего вида:

$$f(\{x_{ij}\}) = \sum_{i,\,j}\left[c_{ij}\left(\sum_{k} x_{ik}\right) + t_{ij}\right]x_{ij},$$

где $c_i(z) = \begin{cases} 0 & \text{при } z = 0, \\ a_i + b_i z & \text{при } z > 0 (a_i,\ b_i > 0). \end{cases}$

Попытка окончилась неудачей, так как в получающемся дереве приходилось перебирать слишком много вершин.— *Прим. пер.*

пример использования идеи предложенного здесь метода [если удалить из алгоритма задачи коммивояжера тот этап, на котором полагают $c(m, p) = \infty$] — это решение задачи о назначениях. Еще один пример см. в работе [7].

ВЫЧИСЛЕНИЯ

Задачи с числом городов не более десяти можно легко решить вручную. Хотя мы и не изучали специально время, потребное для ручного счета, наш опыт подсказывает, что задача с десятью городами может быть решена вручную меньше чем за час.

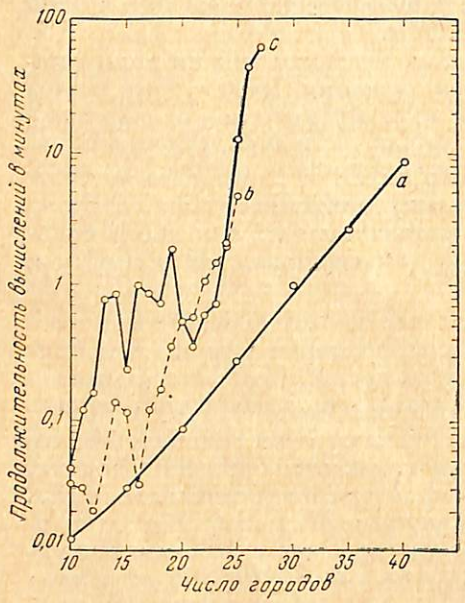


Рис. 7. Машинное время для IBM-7090: *a* — среднее время для задачи с матрицей расстояний, состоящей из трехзначных случайных чисел; *b* — подзадачи, полученные из задачи Хелда и Карпа с 25 городами; *c* — подзадачи, полученные из задачи Хелда и Карпа с 48 городами

Основная проверка предложенного нами алгоритма была проведена на машине IBM-7090. Исследовались задачи двух типов: 1) с асимметричными матрицами расстояний, элементы которых — одинаково распределенные трехзначные случайные числа; 2) различные опубликованные задачи и «подзадачи», полученные из них путем вычеркивания городов. Исходные данные для большинства опубликованных задач были подготовлены по дорожным атласам или картам, так что эти задачи являются симметричными.

Матрицы случайных расстояний имеют то преимущество, что они хорошо статистически определены. Среднее время расчета показано в табл. 1 и на кривой *a* на рис. 7. Задачи с числом городов до 20 обычно решаются за несколько секунд. Однако с увеличением числа городов продолжительность вычислений растет экспоненциально, и для 40 городов составляет в среднем несколько больше 8 мин. Добавление десяти городов увеличивает время решения примерно в 10 раз (весьма приблизительно).

Среднее квадратическое отклонение машинного времени также растет с ростом размера задачи (табл. 1). Так как распределение длительности расчетов несимметрично, то простое среднее квадратическое отклонение может в некоторой степени ввести в заблуждение, во всяком случае при оценке вероятной продолжительности длинного ряда вычислений. Поэтому было построено (по точкам) логарифмически нормальное распределение продолжительности решения. Результаты характеризуются следующими данными. Определяя по табл. 1 двухсигмальное отклонение (в сторону больших значений) для задачи с 40 городами, получаем предельную продолжительность расчета $(3{,}74)^2 \cdot (4{,}55) = 64$ мин., а вероятность того, что продолжительность решения задачи с 40 городами и матрицей случайных расстояний выйдет за пределы 64 мин., оценивается величиной 0,023.

Симметричные задачи обычно решаются заметно дольше по сравнению с задачами того же размера, в которых расстояния случайны. Чтобы полу-

чить последовательность задачи увеличивающегося размера, мы брали опубликованные задачи и получающиеся из них «подзадачи» увеличивающихся размеров. Сначала брались первые десять городов, потом первые 11 и т. д., до тех пор, пока машинное время не становилось чрезмерно большим. Кривые *b* и *c* на рис. 7 соответствуют результатам, полученным для

Таблица 1

Средние значения и средние квадратические отклонения для T при матрицах случайных расстояний (T — время в мин., нужное для решения задачи коммивояжера на IBM-7090)

Число городов	Число решенных задач	Среднее значение величины T	Квадратическое отклонение величины T	Среднее значение * для $\lg T$	Квадратическое отклонение * для $\lg T$
10	100	0,012	0,007	0,015	1,24
20	100	0,084	0,063	0,067	2,09
30	100	0,975	1,240	0,63	2,94
40	5	8,37	10,2	4,55	3,74

* Получено путем нанесения накопленных частот на бумагу с функциональной шкалой логарифмически-нормального закона распределения и последующего подбора выравнивающей прямой; для случая 40 городов произведен численный расчет. В случае десяти городов «нормально-логарифмическая» часть составляет лишь «хвост» распределения: для 30% задач решение получается без дополнительных ветвлений, в связи с чем образуется «сгущение» вероятности y значения, равного 0,002 мин.

«подзадач», получаемых из задач с 25 и 48 городами [3]. Сама задача с 25 городами была решена нами за 4,7 мин. Заметим, что предположение Хелда и Карпа об оптимальности найденного ими решения оказалось правильным.

Кроме того, было решено несколько других задач. Задача Грёса [6] с 20 городами была решена за 0,126 мин., задача о путешествии шахматного коня (с 64 «городами») — за 0,178 мин.

ЛИТЕРАТУРА

1. M. M. Flood. The Traveling Salesman Problem. Opns. Res., 1956, vol. 4, p. 61—75.
2. R. H. Gonzalez. Solution of the Traveling Salesman Problem by Dynamic Programming on the Hypercube. Interim Techn. Rep., 1962, № 18, OR Center, M., I. T.
3. M. Held and R. M. Karp. A Dynamic Programming Approach to Sequencing Problems. J. Soc. Indust. and Appl. Math., 1962, vol. 10, p. 196—210.
4. G. B. Dantzig, D. R. Fulkerson and S. M. Johnson. Solution of a Large Scale Traveling Salesman Problem. Opns. Res., 1954, vol. 2, p. 393—410.
5. D. W. Sweeney. The Exploration of a New Algorithm for Solving the Traveling Salesman Problem. M. S. Thesis, M., I. T., 1963.
6. G. A. Groes. A Method for Solving Traveling Salesman Problems. Opns. Res., 1958, vol. 6, p. 791—814.
7. A. Doig and A. H. Land. An Automatic Method of Solving Discrete Programming Problems. Econometrica, 1960, vol. 28, p. 497—520.

Перевод *Ю. Ю. Финкельштейна*